Eötvös Loránd University
Faculty of Informatics

Ph.D. Dissertation

Dávid Márk Nemeskey

Natural Language Processing Methods for Language Modeling

Doctoral School of Informatics
Dr. Csuhaj Varjú Erzsébet D.Sc.

Foundation and Methodology of Informatics Doctoral Programme
Zoltán Horváth Ph.D.

Supervisors:
András Benczúr Ph.D.
András Kornai D.Sc.

Budapest, 2020

Eötvös Loránd Tudományegyetem
Informatikai Kar

Doktori Értekezés

Nemeskey Dávid Márk

Természetes nyelvfeldolgozási módszerek a nyelvmodellezésben

Informatika Doktori Iskola
Dr. Csuhaj Varjú Erzsébet D.Sc.

Az informatika alapjai és módszertana Doktori Program
Horváth Zoltán Ph.D.

Témavezetők:
ifj. Benczúr András Ph.D.
Kornai András D.Sc.

Budapest, 2020

Acknowledgements

I would like to thank my advisors András Kornai and András Benczúr for their ongoing support, for providing a motivating work environment and particularly for their infinite patience.

The thesis benefited immensely from the insightful remarks of the two reviewers, András Lukács and András Micsik.

I would also like to thank all my current and former colleagues at the SZTAKI HLT Research Group: Judit Ács, Gábor Borbély, Dániel Lévai, Márton Makrai, Katalin Pajkossy, Gábor Recski, Eszter Simon and Attila Zséder, as well as those in the wider NLP "family". I feel honored to be part of such an inspiring community.

Last, but definitely not least, I would like to thank my family for their love and understanding, especially in the hectic final weeks. This thesis would have never been finished without their support.

* * *

Research partially supported by National Research, Development and Innovation Office (NKFIH) grants #115288: "*Algebra and algorithms*" and #120145: "*Deep Learning of Morphological Structure*", as well as by National Excellence Programme 2018-1.2.1-NKP-00008: "*Exploring the Mathematical Foundations of Artificial Intelligence*".

Experiments in Chapter 2 were made possible by a hardware grant from NVIDIA Corporation; `huBERT` in Chapter 4 was trained on TPUs provided by the Tensorflow Research Cloud program. Their support is gratefully acknowledged.

Contents

Introduction

The field of natural language processing (NLP) is contemporaneous with computers. Machine translation systems were developed as early as the 1950s, and the widespread use of personal computers and digitalization of business processes led to an overabundance of textual data that had to be processed, organized, and even understood to a certain degree. NLP systems arose to meet these challenges, so that today millions of computer users routinely use spell and syntax checkers and translation services in their daily work and speech-to-text interfaces when interacting with their devices. Behind the scenes, NLP modules such as lemmatizers, named entity recognizers and parsers are cornerstones in many information processing systems.

Compared to this ubiquity, language modeling has been, until recently, an obscure field even among NLP practitioners. This was in spite of the fact that language models played an integral role in some of the tasks mentioned above, particularly machine translation and speech recognition. Only in recent times, with the advent of deep, neural language models has the situation changed. Word embeddings revolutionized computational semantics, and unsupervised representations supplanted linguistic features in NLP systems. Today, language modeling has become pervasive in all fields of NLP.

In this thesis, we study the interaction between language modeling and NLP, with special focus on two aspects. First, most of the work done for language modeling, surely all success stories, concentrated on English. In this work, we examine how language modeling techniques developed for English perform on Hungarian, and what modifications might be needed to adapt them to a different language. Second, historically the inclusion of language modeling advanced natural language processing systems; here we are concerned with the opposite direction and study how linguistics or NLP can help in training or evaluating language models.

The thesis is structured as follows. Chapter 1 presents an overview of the field of language modeling, with particular focus on its connection to natural language processing. The main discrete and continuous techniques for autoregressive language modeling are introduced, as well as word embeddings, a staple in modern NLP.

In Chapter 2, we evaluate some of the state-of-the-art language modeling methods on Hungarian. To standardize language model assessment in the future, a benchmark corpus is introduced. We also present a morphological method for alleviating the vocabulary problem, which is much more pronounced in Hungarian than it is in English. Continuing with the evaluation theme, Chapter 3 proposes a novel method for assessing multi-sense embeddings based on sense distinctions made in monolingual dictionaries.

Work on Hungarian contextualized embeddings is presented in the next two chapters. Chapter 4 details our work of compiling Webcorpus 2.0, a new Hungarian gigaword corpus, from the Common Crawl and the Hungarian Wikipedia. Its main purpose being a training set for contextual embeddings, it is the largest Hungarian corpus yet by a factor of 3.5. Results on `huBERT`, a preliminary Hungarian BERT model, are also reported here. Chapter 5 describes `emBERT`, a module of the `e-magyar` text processing system, that allows integration of contextualized embedding-based token classifiers into the pipeline. The NP chunker model is the current state-of-the-art for Hungarian. Chapter 6 wraps up the thesis.

Theses

The main theses of the dissertation are the following:

(T1) An exhaustive evaluation of discrete and continuous language modeling methods for Hungarian and the associated benchmark corpus

(T2) The "gluten-free" format, a morphological method of alleviating the vocabulary problem in Hungarian language modeling

(T3) A novel intrinsic evaluation method for multi-sense embeddings based on sense distinctions made in monolingual dictionaries

(T4) Webcorpus 2.0, the largest Hungarian corpus to date by a factor of 3.5, compiled from the Common Crawl web archive and the Hungarian Wikipedia

(T5) `huBERT`, a preliminary Hungarian BERT model based on Wikipedia, which outperforms the multi-language BERT on four Hungarian benchmark tasks

(T6) The `emBERT` module that allows integration of contextualized embedding-based classifiers into the `e-magyar` pipeline, thereby improving the state of the art in NP chunking

Contributions

All work presented in the thesis is the contribution of the author. While Chapter 3 is based on joint research with Gábor Borbély, Márton Makrai and András Kornai, and Chapter 4 benefits from prior work by Balázs Indig, the parts detailed in the text constitute the author's own.

The research described in the thesis has been partly presented in the following papers, listed in the order of the corresponding chapters:

(T1, T2) Results of Chapter 2 were published in Nemeskey (2017);

(T3) The evaluation method described in Chapter 3 constitutes the first half of Borbély; Makrai, et al. (2016);

(T4, T5) Chapter 4 will be published later; Webcorpus 2.0 and `huBERT`, introduced in the chapter, will be uploaded to the SZTAKI HLT repository;

(T6) The `emBERT` module appeared in Nemeskey (2020), which has won Special Award at the XVI. Conference on Hungarian Computational Linguistics.

Resources

Various resources have been created during our research, all of which are publicly available for download under permissive or public domain licenses. Table 1 lists the two corpora presented in Chapters 2 and 4.

Name	Description	Introduced in
emLam	Language modeling benchmark corpus for Hungarian	Chapter 2
Webcorpus 2.0	Our 9 billion token corpus built from Common Crawl and Wikipedia	Chapter 4

Table 1: Corpora introduced in this dissertation

`huBERT` (T5) and the models trained by the `emBERT` module (T6) are listed in Table 2.

Name	Description
`huBERT`	Hungarian BERT model trained on Wikipedia
szeged_ner_bioes	NER model trained on the Szeged NER corpus
szeged_basenp_bioes	State-of-the-art base NP chunker model
szeged_maxnp_bioes	State-of-the-art maximal NP chunker model

Table 2: `huBERT` and the `emBERT` models discussed in Chapter 5

All software used to create the corpora and models described here are available for download under permissive open source licenses. Table 3 enumerates all packages and the GitHub repositories they are preserved in.

Package	Description	Introduced in
emLam	Preprocessing and training scripts for Hungarian language modeling	Chapter 2
cc_corpus	Tools for compiling corpora from Common Crawl	Chapter 4
zim_to_corpus	Scripts to extract Wikipedia pages from .zim archives.	Chapter 4
emBERT	`emtsv` module for pre-trained Transformer-based models	Chapter 5

Table 3: Software libraries presented in the thesis

The hyperlinks given in blue above take readers of the pdf version directly to the right webpage. For the convenience of readers of the paper version we note that the SZTAKI HLT repository that hosts the corpora and models is accessible at `https://hlt.bme.hu/en/resources`; the `emLam` software is accessible from the `e-magyar` GitHub organization at `https://github.com/dlt-rilmta`; and the rest of the software is hosted under the author's GitHub account at `https://github.com/DavidNemeskey`.

Chapter 1

Language Modeling

This work studies the interaction between natural language processing and language modeling. In this chapter, we define the task of language modeling and survey the mainstream techniques used in the last three decades to address it.

To put the main question into context however, in Section 1.1 we begin with a brief overview of the field of natural language processing and the tasks it deals with. Section 1.2 gives the mathematical and historical background behind language modeling. Section 1.3 presents the main language modeling method prior to the advance of deep learning: n-grams. Neural network language models are introduced in Section 1.4, and the most popular neural architectures, LSTM and transformers, are discussed in Sections 1.5 and 1.6, respectively. An overview of "the crown jewels of NLP", word embeddings[1], is given in Section 1.7. Finally, in Section 1.8, we examine the role natural language processing and language modeling had in each other's development.

1.1 Natural language processing

The field *Natural language processing (NLP)* is concerned with creating computer algorithms and systems to process and analyze natural language data. In this thesis we assume a passing familiarity on part of the reader both with NLP and with (high-school level) linguistics. What follows is a short summary of the main tasks in NLP. We also review the basics of machine learning, which underlies most modern NLP systems. For those wishing to know more, the go-to book is Jurafsky and Martin (2009)[3].

[1] `http://bit.ly/1ipv72M`[2], cited as bad example of early overenthusiasm about embeddings in Baroni et al. (2014).

[2] All URLs in the thesis were retrieved on May 16, 2020.

[3] The draft of the 3rd edition is available at `https://web.stanford.edu/~jurafsky/slp3/`.

1.1.1 NLP tasks

The workhorses of NLP are the text processing pipelines that perform routine linguistic analysis on large amounts of text. For English, the most well-known software libraries are Stanford CoreNLP[4] and spaCy[5]. The tasks they address have well-established formalisms and efficient algorithms to solve them. Table 1.1 and Figure 1.1 illustrate how the sentence "*Mary had a little lamb.*" is analyzed by Stanford CoreNLP.

Tokenization is the task of breaking the input text into *tokens*, typically words and punctuation marks (which are split from words).

Lemmatization is performed to determine the lemma (dictionary form) of a word.

Part-of-speech (POS) tagging decides the part-of-speech category (noun, verb, etc.) of words. In languages with richer morphological structure, full *morphological analysis* is performed instead, which assigns a tag to each morpheme (e.g. Hungarian *láthatjátok* 'you can see', may be analyzed as `lát[/V]hat[_Mod/V]játok[Prs.Def.2Pl]`).

Named Entity Recognition (NER) is the task of finding named entities in the text and determining their type (people, organizations, locations, etc.).

Syntactic parsing is performed to build a *parse tree* of the sentence that represents the relations between its words. The two most popular formalisms are Phrase Structure Grammar (PSG) (Chomsky, 1957) and Dependency Grammar (Tesniére, 1959); see Figure 1.1 for an example for both.

Sentence	*Mary had a little lamb.*					
Tokens	Mary	had	a	little	lamb	.
Lemmas	Mary	have	a	little	lamb	.
Part-of-speech	NNP	VBD	DT	JJ	NN	.
Named entities	PERSON					

Table 1.1: Analysis of an example sentence.

There are tasks that are usually tackled by stand-alone systems, though some pipelines (like CoreNLP) do include them:

Coreference resolution finds expressions that refer to the same entity. For instance, if the sentence following the example starts with "*It ...*", it presumably refers to the lamb, not Mary.

[4] https://stanfordnlp.github.io/CoreNLP/
[5] https://spacy.io/

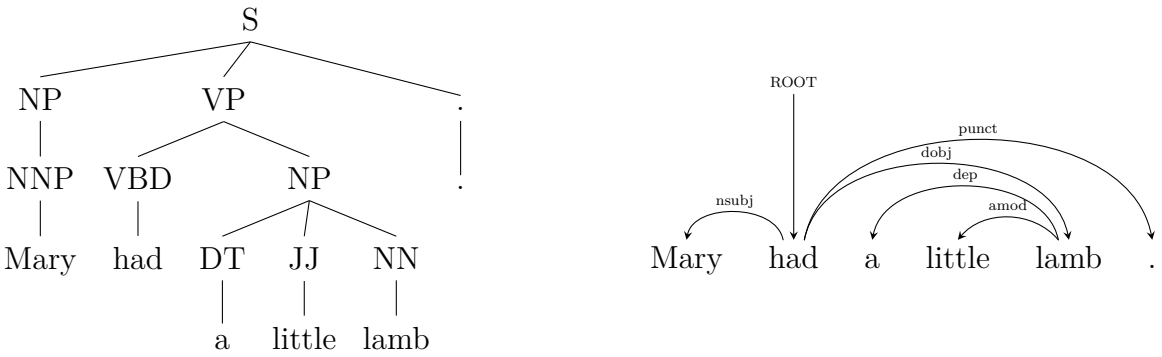


Figure 1.1: Constituency (left) and dependency (right) parse of the example sentence.

Sentiment analysis identifies the sentiment expressed in text (usually 'positive' or 'negative').

Finally, NLP also features high-level *natural language understanding (NLU)* tasks. These are hard problems which are far from being solved. As such, they are always addressed by standalone (mostly research) systems. These include but are not limited to *question answering*; *machine translation*; *text summarization* and *textual entailment*, in which the system has to decide whether a statement can be inferred from a piece of text.

1.1.2 Machine learning

With the exception of tokenization and morphological analysis, all tasks mentioned in the previous section are tackled by machine learning (also: *statistical*) models. Here we briefly introduce the basic machine learning concepts used in the rest of the thesis.

The goal of machine learning is to discover patterns in data. There are two basic paradigms for agentless machine learning: *supervised* and *unsupervised learning*. In the former, models learn a mapping between inputs and known (typically human-supplied) outputs; an example is sentiment analysis, where the inputs are a sentence and its linguistic features, and the output is the sentiment judgement (+/−) made by a human. Unsupervised learning, on the other hand, looks for patterns in unlabeled data; an example is clustering, which tries to group similar objects together based solely on their properties.

Supervised learning can be further divided into *classification*, when there is a predefined set of output labels (POS tags, sentiments) and *regression*, where the output can take any real value. NLP tasks, such as POS tagging and NER, belong to a special case of classification called *sequence labeling*. As with regular classification, the task is to tag each item in the sequence with the correct label; however, the items are not independent

(words in a sentence rarely are), and only algorithms that exploit the dependencies in the sequence can hope to perform well[6].

Supervised models are *trained* on *training data* that consists of manually labeled *training examples*; i.e. input–output pairs. It is a well-established practice to split this data into training, validation and tests sets. The model is trained on the training set, and its performance is evaluated on the test set. The validation set can be used to fine-tune *hyperparameters* of the model.

For NLP tasks, the training data is typically a labeled corpus; syntactic parsers are trained on *treebanks*, where each training example consists of a sentence and its parse tree. One example is the *Penn TreeBank (PTB)* (Marcus et al., 1993), which has served as the *gold standard dataset* for various NLP tasks, such as POS and NER taggers, syntactic parsers, etc. Higher-level tasks have their own set of benchmarks: for question answering, the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016, 2018) is the most popular, while GLUE (Wang et al., 2018), with its 9 tasks, provides a rich testbed for various NLU systems.

Annotated datasets tend to be small (50–100 thousand sentences) due to the financial costs associated with the linguistic expertise required to create them.

1.2 Statistical language modeling

Statistical language modeling (LM), at its most generic, is the task of capturing regularities of natural language in a probabilistic model (Rosenfeld, 2000). In almost all cases however, language modeling is understood to be the task of estimating the probability of a sequence of words. The history of LM can be traced back to Markov (1913) and Shannon (1948), who used n-grams to predict the next character or word, respectively, in natural language text. Language modeling, however, got its real start when it began to be included in *automatic speech recognition (ASR)* systems to improve their performance (Bahl et al., 1983; Baker, 1975; Jelinek et al., 1975). Other NLP fields soon followed suit and before long, language models were incorporated into systems addressing a variety of tasks, including *optical character recognition (OCR)* (Breuel, 1994; Kornai, 1994); *machine translation (MT)* (Brown et al., 1990); and *information retrieval (IR)* (Berger and Lafferty, 1999; Ponte and Croft, 1998). Lately, the deep learning revolution has made language models ubiquitous in all areas of NLP by supplanting (or at least complementing) manual features with word embeddings (see Section 1.7.5).

[6]For instance, there is no way to tell whether the word '*saw*' in a sentence is a noun or the past form of the verb '*see*' without looking at the rest of the words.

Capturing regularities in language is traditionally the domain of linguistic theories of grammar (and morphology, pragmatics, etc). These approaches are based on *explicit* linguistic knowledge, and are typically top-down in the sense that word order and placement are governed by higher-level linguistic relations and structures such as phrases[7].

In contrast, language models are mostly built from the bottom up, from corpus statistics and the word sequence itself. They do not aim to *explain* linguistic phenomena and any linguistic regularities learned by the models are *implicit*. This is not to say that such regularities are not captured, however. Even n-grams are able to find idiomatic phrases, and continuous models are able to represent more. As we shall see, the geometry of embeddings is strongly influenced by the semantic and syntactic properties of words (Mikolov; Sutskever, et al., 2013), while contextual models are able to realize syntactic and dependency relations (Hewitt and Manning, 2019).

We return to the interaction between language models and NLP in Section 1.8.

1.2.1 Motivation

Language models naturally arise from the application of the *noisy channel model* (Shannon, 1948) to "recognition" tasks such as ASR, OCR or machine translation (Jurafsky and Martin, 2009, p.287, 875). Following Brown et al. (1990), here we derive language modeling from the latter. The derivation is analogous in the other tasks as well.

Let us take a French-English machine translation system, which takes French sentences, such as "*les enfants et les femmes enceintes*" and translates them into English. In this particular example, we hope to see "*children and pregnant women*" instead of "*pregnant children and women*", although the latter mistranslation was actually generated by an early MT system (Hutchins, 1995).

The intuition behind the noisy channel model is that all communication happens in English; that the words we see are French is the result of the "noise" introduced by the communications channel. The task of machine translation is then to recover the *source* language text S whose distortion most likely resulted in the *target* language sentence T. More formally, if we have several (source-language) candidate translations, we are looking for $\hat{S}$ that maximizes $P(S|T)$:

$$\hat{S} = \arg\max_{S} P(S|T). \tag{1.1}$$

Using Bayes' theorem, we can rewrite this as

[7]This is irrespective of whether the algorithm used to compute the phrases is itself top-down or not. The CYK algorithm (Younger, 1967), for instance, is bottom-up.

$$\hat{S} = \arg\max_S \frac{P(T|S)P(S)}{P(T)}. \tag{1.2}$$

Since $P(T)$ does not depend on S, we arrive at the final form by omitting it:

$$\hat{S} = \arg\max_S P(T|S)P(S). \tag{1.3}$$

Here $P(S)$, the prior probability, is the language model. The likelihood $P(T|S)$ is called the *translation probability* in MT, the *acoustic model* in ASR, etc; it represents the transformation affected by the noisy channel on S.

As we can see, the language model is an integral part of "recognition" tasks. Its main purpose is to ensure that the text we arrive at is indeed a sentence in the source language. It is easy to see that while the likelihood is task-specific, the language modeling component is universal, and the same model used in an MT system can be easily plugged into a speech or optical character recognizer. This is also the reason why we can study language modeling as a separate task.

1.2.2 Mathematical formulation

In this section, we take a closer look at $P(S)$ and present a simple mathematical formulation for language modeling. Let us assume that we wish to model a sentence S, which consists of N words $S = w_1 w_2 ... w_N$. Then, our goal is to estimate the probability

$$P(S) = P(w_1 w_2 ... w_N). \tag{1.4}$$

While the formulation in Equation 1.4 is still too generic to be of practical use, it can be rewritten using the *chain rule of probability* as

$$\begin{aligned} P(w_1 w_2 ... w_N) &= P(w_1)P(w_2|w_1)...P(w_N|w_1...w_{N-1}) \\ &= \prod_{i=1}^{N} P(w_i|w_1...w_{i-1}) \\ &= \prod_{i=1}^{N} P(w_i|h_i), \end{aligned} \tag{1.5}$$

where $h_i = w_1...w_{i-1}$ is called the *history* of word w_i. Equation 1.5 essentially rephrases the problem of estimating the probability of a whole sentence into guessing the probability of the next word, given all the words that precede it. For instance, given the sentence "*Language modeling is* _____", a good language model will estimate a high probability

to continuations like "*hard*" and "*fun*", and close to zero to unrelated tokens such as "*sit*" and "*!*".

While this formulation is mathematically motivated, it also fits well with the linearity and sequentiality of human language. Since computation follows the left-to-right (or right-to-left, depending on language) direction of writing, it can be used for *text generation*. Starting from a predefined (possibly empty) history, the distribution $P(w_i|h_i)$ is sampled, and the chosen word is appended to the history. The process is continued until a certain number of iterations pass, or until a special end-of-document token is generated. Mathematically speaking, generation is a *random process* where the output only depends on its previous values, and models that support it are called *autoregressive* language models.

For early language models, such as n-grams, generation was more of a theoretical possibility, as the resulting text was of very low quality. However, recent Transformer-based LMs (see Section 1.6) are able to produce texts of close to human-level grammaticality. A few examples are given in Appendix B.

The unidirectional language model, while both intuitive and useful, is not the only way the probability of a word sequence can be computed. A more generic decomposition exists:

$$P(w_1 w_2 ... w_N) = \prod_{i=1}^{N} P(w_i|c_i), \tag{1.6}$$

where c_i is the *context* of w_i. The history is one possible context, but not the only one:

1. In bidirectional models such as BERT (Devlin et al., 2019), the context includes the whole sentence, except the target word;

2. In skip-gram models, such as w2v (Mikolov; Chen, et al., 2013), the context includes a number of randomly selected words in a window around the target word;

3. The syntactic properties of a word, such as dependency relations, can also serve as context (Dyer et al., 2016).

1.2.3 Training

Language modeling is a machine learning task, though it seems to fall between the two main paradigms. On the one hand, it does not require labeled training data, and it is often termed 'unsupervised'. On the other hand, it is essentially a sequence labeling problem, where the ground truth comes from the data itself. Because of this, a more appropriate term would be *self-supervised*.

In order to build a language model, the $P(w_i|c_i)$ probabilities must be established. Like all statistical models, the predictions a language model makes are data-driven, and so the probabilities are learned from data during a *training* procedure. For language models, data is natural language text, which is acquired from a *training corpus*. Once the model is trained, it can be used for *inference*: i.e. computing text probabilities.

Equation 1.6 serves as the theoretical basis for language model training. Broadly speaking, we maintain a $P(w_i|c_i)$ table for all word – context pairs. The training process visits every word in the corpus, records its history (or context) and updates the conditional probabilities accordingly. The exact details are different for each LM technique. n-gram models maintain the conditional probability table in memory, and compute the probabilities from *global* statistics. Most other models encode the estimates implicitly in their parameter space and update it iteratively based on the local context of each word in the corpus.

In practice, the naive process outlined above is intractable. The main culprit is the unbounded memory: the prediction of the i^{th} word is conditioned on *all preceding words*, from the beginning of a potentially very long sentence or document[8]. Under this condition, almost all word occurrences will have a unique context. This is problematic for two reasons. First, storing very long, unique histories would grow the size of the $P(w_i|c_i)$ table beyond any measure. Second, it leads to the issue of *data sparsity*: during inference, most words will have previously unseen histories, and so all probability estimates will be zero.

Because of this, all language modeling techniques impose a limit on the history taken into account for a particular word. n-grams and attention-based neural models restrict the number of context tokens; recurrent models have a set memory capacity, and only remember the – relatively – recent part of the history.

1.2.4 Evaluation

The quality of a language model is assessed on new or held-out data, typically the dedicated test set of the training corpus. Performance is most commonly reported in terms of *perplexity (PPL)* (Jelinek et al., 1977). Given a held-out data of $w_1 w_2 ... w_N$, perplexity is defined as:

$$\text{Perplexity}(M) = 2^{H(P,M)} = 2^{-\sum_{i=1}^{N} P(w_i) \log M(w_i)}, \tag{1.7}$$

where P is the real data distribution, M is the model distribution and $H(P, M)$ is the *cross*

[8]Proust's *À la recherche du temps perdu* starts and ends with the same word. For a language model to give a good prediction for the last word, it might have to include in its history all seven volumes of the book. Needless to say, such extremes are rare, but it showcases the problems of unbounded history.

entropy of the test data given the model. The lower the perplexity is, the closer M is to the "true" data distribution. Unfortunately, the latter is not known, so it is approximated with a degenerate distribution centered on w_i:

$$P(w_j) = \begin{cases} 1, & \text{if } w_j = w_i \\ 0, & \text{otherwise.} \end{cases} \tag{1.8}$$

With this, the cross entropy can be replaced with the average log likelihood of the held-out data. This leads to the formula used to compute perplexity for LMs:

$$\text{Perplexity}(M) = 2^{\sum_{i=1}^{N} \log M(w_i)}. \tag{1.9}$$

We note that most language modeling libraries use base e instead of base 2 (*nats* instead of *bits* as standard in information theory), and the widely used SRILM system (Stolcke et al., 2011) uses base 10. Luckily, this choice does not affect perplexity, as the exponentiation and the logarithm cancel out.

Perplexity also has an intuitive meaning. If a model assigns equal probability to 40 words, the perplexity will be exactly 40. Consequently, it can be seen as the average "branching factor" of a language model; the average number of words it considers at any given context. Interestingly, for character-level language models, entropy is usually reported instead of perplexity; perhaps because the data compression aspect of LMs is more important there.

Yet for all its intuitiveness and ubiquity, there are signs indicating that perplexity might not be the most accurate measure of LM quality. It has been shown to not correlate very well with downstream (e.g. translation) performance (Goodman, 2001); and while there are no exact measurements, it was found that it might take a perplexity reduction of up to 30% to translate into improvements in speech recognition (Rosenfeld, 2000). Because of this, papers that study language modeling in relation to speech recognition routinely report the *word error rate (WER)* to assess how the language model benefits the system as a whole (Chen et al., 1998). However, for evaluating language models in separation, perplexity remains the preferred metric.

1.2.5 Discrete and continuous methods

Language modeling techniques can be divided into two groups: *discrete* and *continuous space* methods. The two differ fundamentally in how words are represented: discrete models treat words as isolated entities, while continuous models *embed* them into a vector space, typically $\mathbb{R}^d$.

The two approaches utilize radically different branches of mathematics. Continuous models employ linear operations, whereas discrete methods depend on set operations, Markov processes and on populating (database) tables. Another crucial difference is that continuous models can impose a distance metric between words, while in discrete space, we can only check for identity.

Historically, discrete methods predate continuous ones. In the next sections, we take a survey of the most important language modeling techniques, in the order of their appearance.

1.3 n-grams

Before the advent of neural net language models, n-grams were the language modeling technique of choice for three decades. Nowadays, they are only used as baselines for more advanced methods. Due to the simplicity of the idea, however, n-grams are still taught at universities[9] as an introduction to language modeling (Jurafsky and Martin, n.d., ch. 3).

n-gram models are a direct approximation of Equation 1.5. Instead of computing the probability of a word given its entire history, an n-gram model truncates the history to the last $n-1$ words:

$$P(w_i|w_1w_2...w_{i-1}) \approx P(w_i|w_{i-n+1}w_{i-n+2}...w_{i-1}) =: P(w_i|w_{i-n+1}^{i-1}) \tag{1.10}$$

In other words, w_i is assumed to be conditionally independent of the rest of the sequence, given its immediate predecessors. This is called the $n-1$-order Markov assumption, and it helps to make the estimation problem tractable. The choice of n depends on the size of the corpus and of available memory; for a corpus of about 1B words, 6-grams already provide diminishing returns. The other end of the spectrum, when $n = 1$, is called the *unigram* model. In unigram models, words are taken to be completely independent of one another. Since this ignores any text cohesion, unigrams are mainly used for modeling bag-of-word content, such as queries in a web search engine.

Available memory imposes a strong limit on the n-gram order. For a moderate vocabulary of 30,000 words, there are $30,000^4 = 810$ quadrillion possible 4-grams. Luckily, with finite training data, only a fraction of this number actually manifests: a corpus of 1B words contains at most one billion different 4-grams. While much more manageable, with four 16-bit word ids for the history and 32-bit floating point numbers for the probability, the $P(w_i|h_i)$ table still occupies up to 12GB – not negligible even by today's standards.

[9] https://web.stanford.edu/class/cs124/

Given the fact that – as we shall see presently – a full n-gram language model includes sub-models of all orders from 1 to n, it is evident that high order n-gram language models can use an inordinate amount of memory. For this reason, trigrams stayed in use until a few years back (Tarján et al., 2016), even though the performance advantage 5-grams hold over them had long been proven (Goodman, 2001).

Table 1.2 shows the history taken into account by n-grams of increasing order when estimating the probability of the last word in the sentence "*That Sam-I-Am! That Sam-I-Am! I do not like that Sam-I-Am!*"[10]. It can be seen how the Markov assumption prevents the model from utilizing the context necessary to predict the right word. The history of orders 2 through 6 is very generic and contains only function words. It is only in the 7-gram model that a content word (incidentally, the one to guess) finally appears. Unfortunately, as we have seen, the memory requirements of 7-grams make them unusable in practice.

This loss of context is typical to n-grams and it severely restricts their language modeling performance. It is worth mentioning though that the effect is much less pronounced with the character-level n-grams used in e.g. OCR. As the character vocabulary is much smaller[11] it is possible to train character n-gram models of much higher orders.

Order	Unseen	History
1	That Sam-I-Am I do not like that	
2	That Sam-I-Am I do not like	that
3	That Sam-I-Am I do not	like that
4	That Sam-I-Am I do	not like that
5	That Sam-I-Am I	do not like that
6	That Sam-I-Am	I do not like that
7	That	**Sam-I-Am** I do not like that

Table 1.2: The history considered by n-grams of various orders for the last word of the sentence "*That Sam-I-Am! That Sam-I-Am! I do not like that Sam-I-Am!*". Punctuation marks are omitted for clarity.

[10]From *Green Eggs and Ham* by Dr. Seuss. The idea of using this example comes from Jurafsky and Martin (2009, ch. 4).

[11]Clearly, this argument does not stand for languages with a logographic writing system, such as Chinese. In this work, when referring to character-level language models, we always mean *small-vocabulary* models, on the order of a few hundred characters at most. This includes all languages with alphabetic, syllabic, moraic and related scripts.

1.3.1 Training

The probabilities of n-grams can be estimated from relative frequency counts. Continuing with the example above, we count how many times the history (*do not like that*) occurs in the training corpus, and how many of these is followed by the word *Sam-I-am*:

$$P(\textit{Sam-I-am}|\textit{do not like that}) \approx \frac{C(\textit{do not like that Sam-I-am})}{C(\textit{do not like that})}. \tag{1.11}$$

In the general case, 1.11 becomes

$$P(w_i|w_{i-n+1}^{i-1}) \approx \frac{C(w_{i-n+1}^{i-1}w_i)}{C(w_{i-n+1}^{i-1})}. \tag{1.12}$$

Equation 1.12 is called the *maximum likelihood estimation (MLE)*, because a language model with n-gram probabilities obtained this way is the most likely to reproduce the counts observed in the training data. Given a large enough training corpus representative of the language, MLE is expected to approximate the "real" n-gram probabilities.

1.3.2 Smoothing

The maximum likelihood estimation as described above has several problems due to data sparsity. The most glaring issue is that n-grams not found in the training corpus will have 0 probability. Consequently, these n-grams will never be produced in generation mode, and will be rejected outright during inference, when the model is used in e.g. machine translation or speech recognition, hurting downstream performance. Even for n-grams present in the training corpus, MLE will yield poor estimates when the counts are small.

These issues can be mitigated somewhat with larger training corpora. However, human language is a creative process, and no corpus can cover all the possible sentences or utterances the model will have to predict. For this reason, instead of using the maximum likelihood estimates, n-gram models always employ some form of *smoothing*. This comes in three flavors.

Discounting methods aim to flatten the probability distribution following a particular history by taking some of the probability mass from the most frequent continuations and redistributing it to the less frequent (or zero-count) words. This ensures that all n-grams have nonzero probabilities, irrespective of whether they occur in the training corpus or not.

A related technique to address the zero-count problem is using the n-gram "hierarchy". The idea is that lower order n-grams are less sparse, so we can use them to estimate the probability of missing higher-order n-grams. This can be done in two ways: *backoff* models

recursively fall back to coarser ($n-1$, $n-2$...-gram) models when the context of a word was not seen during training, while *interpolated* models always incorporate the lower orders into the probability estimation:

$$P(w_i|w_{i-n+1}^{i-1}) = \lambda P_{MLE}(w_i|w_{i-n+1}^{i-1}) + (1-\lambda)P(w_i|w_{i-n+2}^{i-1}), \tag{1.13}$$

where P is the smoothed probability.

A variety of smoothing models have been proposed over the years: Laplace (additive) smoothing (Lidstone, 1920) and Good-Turing discounting (Good, 1953) are not very efficient by themselves, but the latter is used as a basis for other methods. Katz backoff (Katz, 1987), Jelinek-Mercer (Jelinek and Mercer, 1980) and Kneser-Ney (Ney et al., 1994) smoothing were the "workhorses" of n-gram modeling in the 90s. For higher order n-grams, the most effective method is the modified Kneser-Ney (KN) smoothing introduced in Chen and Goodman (1999).

Out-of-vocabulary (OOV) words (unigrams), i.e. those not seen during training are commonly accounted for by introducing a new token, usually written `<unk>`. In order to get a good estimate for it, occurrences of low-frequency words in the corpus are replaced (wholly or partially) with `<unk>` prior to training. The exact cutoff is corpus- and language specific; for English, 3 or 5 are common choices (Chelba et al., 2014).

1.3.3 Class-based models

A major disadvantage of discrete methods, n-grams included, is the lack of a similarity measure between words. This prevents the model to exploit the semantic clustering of words. For example, if the training corpus contains phrases like "*lives in London*" or "*lives in Berlin*", we might expect the model to also give a higher probability to the words "*Budapest*" or "*Tokyo*" after "*lives in*", even if those phrases have not been observed. However, this falls beyond the capabilities of regular n-grams.

Class-based models, first proposed in Brown et al. (1992), establish connections between words by clustering the vocabulary. This enables the model to condition not (just) on the previous words, but their classes as well. One possible formulation is

$$P(w_i|w_{i-n+1}^{i-1}) = P(w_i|C_i)P(C_i|C_{i-n+1}^{i-1}), \tag{1.14}$$

where C_i is the class of the i^{th} word, and $C_i|C_{i-n+1}^{i-1}$ is a class n-gram.

For the mathematically inclined, this formula provides an interesting parallel to Equation 1.10 describing n-grams in general. While n-grams can be thought of as $n-1$-order *Markov chains*, Equation 1.14 clearly defines an $n-1$-order *Hidden Markov model (HMM)*.

Note that while the factorization allows for a compression of the language model (specifically, the class transition probabilities) (Goodman and Gao, 2000), it also introduces a bottleneck, since the emission probability of a word is conditioned solely on its class.

As the performance of a class-based model depends heavily on the quality of clustering, a lot of effort has gone into finding the best clusters (Brown et al., 1992; Kneser and Ney, 1993; Ney et al., 1997; Pereira et al., 1993). Automatic clustering was found to work better than part-of-speech (POS) tags (Niesler et al., 1998). This should come as no surprise, as POS categories are coarser and have a syntactic, and not semantic, function.

In general, class-based models by themselves usually perform worse than their word-level counterparts (Martin et al., 1998; Niesler et al., 1998), while interpolating the two is said to reduce perplexity by 5–20% (Emami and Jelinek, 2005; Kneser and Ney, 1993). However, it is worth mentioning that the improvements were reported over weak bi- and trigram baselines. More thorough work reveals that class-based models fail to bring any advantage over 5-gram (Goodman, 2001).

Lackluster performance notwithstanding, the idea of using semantic relatedness in language modeling is well grounded, and as we shall see in Sections 1.4.3 and 1.7, it is one of the major advantages continuous models have over their discrete cousins.

1.3.4 Outlook

In this section, we present a bird's-eye view on the landscape of discrete language modeling techniques beyond n-grams. Some of the methods are extensions to n-gram models; others are based on completely different theoretical foundations. Since these methods are not central to the thesis, we only touch on them briefly; the interested reader may refer to the papers cited below for details, or to Goodman (2001) and Rosenfeld (2000) for a more complete survey.

Skipping (or *skip-gram*) models are a simple extension to n-grams, which condition on a discontinuous history (e.g. $P(w_i|w_{i-3}w_{i-1})$) (Huang et al., 1993; Ney et al., 1994). While the improvement they provide is negligible (Goodman, 2001), the idea makes a comeback in Section 1.7.2.

Cache models (Kuhn and De Mori, 1990) aim to rectify the context loss that comes from the truncated n-gram history by maintaining a cache of recent words. Kuhn and De Mori (1990) reports up to 60% perplexity improvements, and the idea works well with any type of language model.

An interesting line of research focused on incorporating linguistic information to language models. Moore et al. (1995) supplanted a trigram model with syntactic and semantic features extracted from a lexicon, while Chelba and Jelinek (1998) and Charniak

(2001) used a statistical CFG parser as model. All papers reported 10%-25% perplexity improvements over their respective baselines.

In *Factored Language Models (FLMs)* (Bilmes and Kirchhoff, 2003; Kirchhoff et al., 2008), each word is represented by a feature vector comprised of linguistic features, such as morphological classes, stems, etc. An n-gram model is then trained over these vectors. The papers also introduce the idea of *generalized parallel backoff*, in which a feature-n-gram can fall back to a lower order of any subset of its features. Unfortunately, the full backoff model was computationally too expensive to train, and GPB-FLM 3-grams failed to decisively outperform regular word trigrams. On the other hand, the joint LM of Filimonov and Harper (2009) successfully integrated FLMs and CFG features into a decision tree, and even outperformed 5-gram KN models by a small margin.

Exponential, or *Maximum Entropy (ME)* models (Darroch and Ratcliff, 1972) are a step towards continuous space language modeling. The generic formula is

$$P(w|h) = \frac{1}{Z(h)} \exp\Big(\sum_i \lambda_i f_i(w,h)\Big), \tag{1.15}$$

where the features f_i are arbitrary functions of w and h (n-grams, skip-grams, etc.), $Z(h)$ is a normalizing term and λ_i are the parameters to train. The main advantages of ME models are their ability to incorporate various features into the model (Berger et al., 1996; Della Pietra et al., 1992; Lau et al., 1993; Rosenfeld, 1994) and their smoothing effect (Chen and Rosenfeld, 1999). However, at the time they were computationally intensive to train (this is no longer true, compared to modern neural language models), and did not outperform KN 5-grams consistently.

1.4 The first neural models

By the beginning of the millenium, discrete language modeling has reached its limits: no method seemed to provide any improvement over cached Kneser-Ney 5-grams (see Section 1.3.4). This prompted very cautious or outright pessimistic paper titles such as "*A Bit of Progress in Language Modeling*" (Goodman, 2001) or "*Two decades of Statistical Language Modeling: Where Do We Go From Here?*" (Rosenfeld, 2000). The solution, however, came not from the discrete world, but with the introduction of *neural (net) language models (NNLM)*.

1.4.1 Neural networks

Before we delve into the details on NNLMs, a short summary of neural networks is in order. Readers familiar with the concepts outlined here can safely skip this section. Those who wish to learn more may find numerous books on the subject; we recommend Goodfellow et al. (2016)[12].

Artifical Neural Networks (ANNs) or *Neural Networks (NN)* for short, are a family of machine learning models loosely inspired by biological neural networks. Many neural network architectures exist, such as convolutional networks (Fukushima, 1980; Le Cun, 1989), Boltzmann machines (Ackley et al., 1985; Smolensky, 1986), or recurrent and attention networks, which we shall introduce later. Figure 1.2 shows the most common architecture, the *Feedforward neural network (FFNN)* or *Multilayer Perceptron (MLP)*:

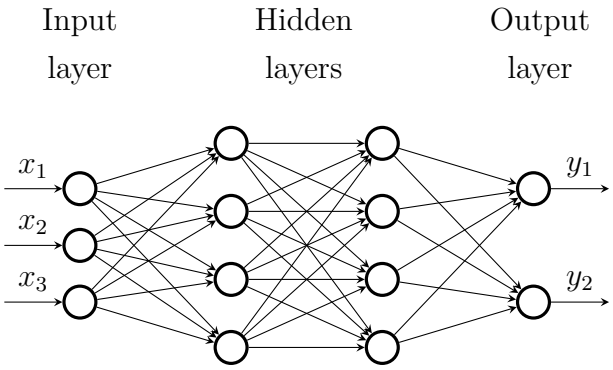


Figure 1.2: A feedforward network with two hidden layers[13]

The network is made up of *layers*, each of which consists of a number of *neurons*. The NN takes its inputs $\mathbf{x} = [x_1 x_2 \cdots x_n]$ through its input layer, and present its outputs $\mathbf{y} = [y_1 y_2 \cdots y_m]$ in the output layer. Both $\mathbf{x}$ and $\mathbf{y}$ are real-valued vectors. The neurons in the hidden layers perform the following transformation:

$$y = f(\mathbf{w}^\intercal \mathbf{x} + b) = f(\sum_i w_i x_i + b). \tag{1.16}$$

Here $\mathbf{x}$ and y are the inputs and outputs of the neuron, respectively. The *input weights* ($\mathbf{w}$), and the *bias* term (b) are parameters of the model, which must be trained. f is the *activation function* that defines the output of the neuron in terms of its inputs. The activation function is almost always non-linear to allow the network to model non-linear problems (Minsky and Papert, 1969); hence, it is often called "*the nonlinearity*".

[12] https://www.deeplearningbook.org/

[13] Figure based on https://github.com/PetarV-/TikZ/tree/master/Multilayer%20perceptron

For efficiency reasons, modern deep learning libraries, such as Pytorch (Paszke et al., 2017, 2019) or Tensorflow (Abadi et al., 2016a), treat the layer as the basic unit instead of the neuron. Also, the implementation might split the right side of Equation 1.16 into two layers: a linear one, which performs the matrix multiplication and a nonlinear layer, which applies the activation function.

Although the original perceptron (Rosenblatt, 1957) used the *Heaviside step function*, differentiable alternatives such as the *logistic sigmoid* or the *hyperbolic tangent* functions soon took its place. Recently, the *Rectified Linear Unit (ReLU)* (Hahnloser et al., 2000; Jarrett et al., 2009) has gained popularity. Each of these functions have various pros and cons: sigmoid functions tend to *saturate* easily, meaning in most of their domain, they are very flat and their value does not change much when the input does. For the most part, ReLUs rectify this problem, but are prone to becoming inactive when $x < 0$.

In regression (real-valued function approximation) problems, the outputs of the network, (y_i in Figure 1.2) can be used as is. For classification tasks, such as language modeling, a final *softmax* nonlinearity

$$\sigma(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \tag{1.17}$$

is applied to the outputs in order to normalize them into a probability distribution. In this case, the y_is are called *logits*.

It has been shown that feedforward networks are universal function approximators (Hornik et al., 1989). In fact, even a single hidden layer is sufficient to model any function (Cybenko, 1989). However, using multiple hidden layers is preferred to a single one for two reasons. First, a single layer might require an exponential number of neurons to perform the same task. Second, stacking several layers allow each layer to "specialize" and extract different features; later layers typically learn higher level features. This phenomenon is especially well documented in image classification (LeCun et al., 1998; Zeiler and Fergus, 2014), where *convolutional neural networks (CNNs)* can be hundreds of layers deep (He et al., 2016). This gave rise to the current name of the field: *deep learning*.

1.4.2 Training

Discrete models, such as n-grams, usually employ global optimization strategies using corpus-level statistics. This is made possible by the fact that their parameter space essentially coincides with these statistics (such as n-gram frequencies). The parameters, or weights $\boldsymbol{\theta}$ of continuous models, on the other hand, are a property of the model itself, and not derived from the training data. This necessitates a completely different training

regime.

The aim of the training process is to tune the weights of the model, so that it can predict the right output(s) y for each input $\mathbf{x}$. The quality of the prediction is characterized by a *cost function*, which we want to minimize:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},y)\sim\hat{p}_{\text{data}}} L(f(\mathbf{x};\boldsymbol{\theta}), \hat{y}), \text{[14]} \tag{1.18}$$

where $\mathbf{x}$ is the input for a training example, $\hat{y}$ and $y \coloneqq f(\mathbf{x};\boldsymbol{\theta})$ are the target and the predicted output, respectively, and L is the per-example loss function. In effect, the cost function is the averaged loss w.r.t. the empirical distribution $\hat{p}_{\text{data}}$.

The *loss* function for classification tasks is typically the cross-entropy loss:

$$L(y, \hat{y}) = -\sum_i \hat{y}_i \log y \doteq -\log y_i. \tag{1.19}$$

The dotted equation applies when $\hat{y}$ is a *one-hot* vector, i.e. it is 1 for the true class $\hat{y}_i$ and 0 otherwise (see also Equation 1.8).

Neural networks are trained via iterative optimization algorithms, such as *stochastic gradient descent (SGD)* (Robbins and Monro, 1951) or one of its variants. Such algorithms try to find a local minimum of $J(\boldsymbol{\theta})$ by iteratively taking steps in the weight space along the negative gradient of J. Regular gradient descent (Cauchy, 1847) computes the real gradient on the whole training set. This requires that we store the gradients for each training example, and is only feasible for very small training sets. Stochastic methods like SGD visit a *(mini)batch* of examples at each *step* instead, and compute an approximate gradient from the batch. The weights are then updated as follows:

$$\boldsymbol{\theta} \coloneqq \boldsymbol{\theta} - \eta\nabla\Big(\frac{1}{n}\sum_{i=1}^{n} L(f(\mathbf{x}, \boldsymbol{\theta}, \hat{y}_i))\Big), \tag{1.20}$$

where n is the size of the minibatch and η is the learning rate, a hyperparameter. The update in neural networks is effectuated by the *backpropagation (backprop)* algorithm (Rumelhart et al., 1986), which recursively applies the chain rule from last layer to first using dynamic programming techniques for efficiency.

During training, SGD sweeps through the whole training set; one "sweep" is called an *epoch* or *iteration*. The training lasts either until a specific number of epochs elapse, or until the cost (measured on the validation set to avoid *overfitting* our model to the training set) stops decreasing. The latter strategy is called *early stopping*.

Neural networks have existed for over six decades; however, they have only become

[14] Formulation from Goodfellow et al. (2016, ch.8)

mainstream in the last ten years. The reason for this is that, efficient as backpropagation is, they are very slow to train. It is the availability of powerful GPUs, actually driven by the gaming industry, that finally made deep learning possible. Nowadays, the largest models run on specialized architectures, such as clusters of data center GPUs[15] or Google's *tensor processing units (TPUs)*[16].

The outline above is but a very brief summary of how neural networks are trained. In practice, the process is somewhat brittle; specifically, there are no guarantees that it finds a good local minimum. This is because there are many free variables and hyperparameters, which all interact in unpredictable ways (Greff et al., 2015). What architecture to choose and how big should the model be? What learning rate should to use, what optimizer? Should the learning rate be fixed or changed according to a schedule? Even seemingly innocuous choices, such as the batch size, can affect performance (author's own experiments, or e.g. Liu et al. (2019)). Because of this, it is considered good practice to try to find the best settings through a hyperparameter search; unfortunately, it is not an option for those on a constrained budget. *Regularization* techniques may also alleviate some of the issues; nevertheless, training a neural network still remains an art to some degree.

1.4.3 Bengio's model

We can now return to neural language modeling. While a proof-of-concept neural LM had existed before (Xu and Rudnicky, 2000), the first mature model, one that also serves as a basis for today's architectures, was published in Bengio et al. (2003).

The model itself is the neural equivalent of n-gram models. When predicting the i^{th} word, the model takes the previous $n-1$ words as input. At this point, the words are represented with their index in the vocabulary V. These word ids are then converted into *word feature vectors* (real valued vectors in $\mathbb{R}^d$) via a table lookup in the $V \times d$ matrix C. The vectors are concatenated and passed through the hidden FF layer. Finally, a softmax layer is applied to produce the output distribution. The model architecture is depicted in Figure 1.3.

At its heart, the model is a regular neural classification architecture. The main contribution of the paper is the look-up table matrix C, which converts words to vectors in $\mathbb{R}^d$. This component is not fixed, but trained together with the rest of the network. Its main use is to *embed* discrete entities (words) into continuous space; this gives it the name it is known today: *embedding*. Such a component is a must for all neural language models to

[15] https://www.nvidia.com/en-us/data-center/
[16] https://en.wikipedia.org/wiki/Tensor_processing_unit

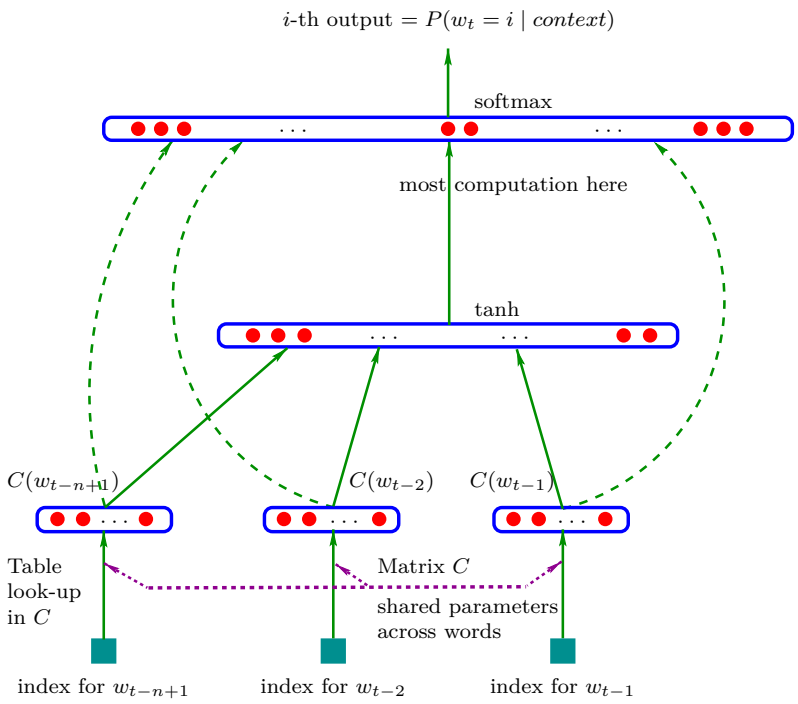


Figure 1.3: Architecture of a neural feed forward language model

bridge the gap between the discrete nature of the task and the continuous representation of the model.

Yet the embedding matrix is also what gives the model most of its power. Because its weights are trained along with the rest of the architecture, it can potentially learn to map words that occur frequently in the same context to similar vectors. Bengio et al. (2003) quotes the two sentences "*The cat is walking in the bedroom*" and "*A dog was running in a room*", which gives good examples for words with similar semantic and/or grammatical role at every word position (*a* and *the*, *cat* and *dog*, etc). Embeddings allow the model the generalize better to previously unseen sentences, as "the presence of only one of the above sentences in the training data will increase the probability, not only of that sentence, but also of its combinatorial number of "neighbors" in sentence space" (Bengio et al., 2003).

Exploiting word similarity is not a new idea; Section 1.3.3 already introduced class-based n-grams, an early attempt. However, utilization of word similarity through embeddings is an innate property of all neural LMs, and is exempt from the disadvantages of class-based n-grams such as the emission bottleneck in HMM-style models.

A 4-gram version of the model was evaluated on the Brown corpus (1.2M tokens) and the AP News corpus (14M tokens). Compared to the best performing n-grams (3-gram on Brown, KN 5-gram on AP), the neural model achieved 20% (312 to 252) and 7% (117 to 109) perplexity reduction, respectively.

1.4.4 Performance

n-grams and neural language models have widely different performance characteristics. n-grams can be trained relatively quickly, but due to the curse of dimensionality marring discrete methods, the models may take up a large amount of memory, potentially on the order of $|V|^n$ (see Section 1.3). Neural LMs, on the other hand, can be very economic with their parameter space. Since the embedding matrix is shared among word positions, and the size of hidden layer increases only linearly with n, the model introduced above could easily accommodate a longer history than what is possible with n-grams.

The power and economy in memory space comes at a price, however. Neural LMs are infamously slow to train. The model above, with very small embedding and hidden layer sizes (30 and 50, respectively), was trained for 3 weeks on 40 CPUs. While this particular network could be trained today in a few minutes on modern GPUs or TPUs, since recent models use more complex and larger architectures, training times of days or weeks are still common.

As Figure 1.3 indicates, the main offender is the final softmax layer. The total computational cost of processing a single training example in the n-gram setup is (in order of layers) $\mathcal{O}(n \times d + (n \times d) \times h + h \times |V|)$, where d is the embedding dimension and h the size of the hidden layer. Since in language modeling, $|V| \gg d$ (in Bengio et al. (2003), $|V|$ is around 16-18,000, while d is 30 or 60), the final softmax term $h \times |V|$ dominates the sum. Taking the best performing system in Bengio et al. (2003), where $|V| = 16000$, $d = 30$, $n = 5$ and $h = 50$, softmax is responsible for 99% of the total computation and memory cost.

Much research has been devoted to develop computationally favorable alternatives to softmax. Both *Noise Contrastive Estimation (NCE)* (Gutmann and Hyvärinen, 2010; Mnih and Teh, 2012) and *Importance Sampling* (Bengio and Senécal, 2003, 2008) sample k random "noise" words for each training example and try to model softmax as a classification between true and noise words (Jozefowicz et al., 2016), thereby bringing the cost down from $|V| \times h$ to $k \times h$. *Hierarchical softmax* (Morin and Bengio, 2005) builds a decision tree based on a hierarchical clustering of V, where each leaf node stores the probability of a single word, while inner nodes keep track of the probability mass associated with their children. The tree can compute the probability of a word with a single lookup, changing the complexity to $\log_2 |V|$. *Differentiated Softmax* (Chen et al., 2016) is based on the intuition that we know less about rare words and assigns fewer parameters to them than to their frequent siblings. Implementation-wise, the dense softmax layer is replaced with a sparse diagonal block matrix, each block corresponding to a bucket of words with similar counts. Finally, *CNN-Softmax* (Jozefowicz et al., 2016) does away with the matrix

completely, and computes the output embedding with a character-based convolutional network.

The methods described above have all been used more or less successfully to train neural language models. Several experiments prove that models equipped with importance sampling, differentiated and hierarchical softmax attain perplexity scores similar to the full softmax case; NCE and CNN-Softmax, on the other hand, generally perform much worse and are not recommended for language modeling (Chen et al., 2016; Jozefowicz et al., 2016). The speed-up for the sampling methods and hierarchical softmax can fall anywhere between 19 (Bengio and Senécal, 2003) and 258 (Morin and Bengio, 2005); in reasonably sized recurrent LMs, it is much closer to the former. With differentiated and CNN-Softmax, the speed-up is negligible; however, these are the only methods that also decrease the memory usage of the softmax layer.

Another line of study attempted to compute the exact loss efficiently instead of approximating it (Brébisson and Vincent, 2015; Vincent et al., 2015). While a 3,000-fold speed-up was reported on the spherical softmax loss family, the results could not be applied to regular softmax.

An early approach meant to circumvent the performance problem by relegating the neural LM to augment a main n-gram model (Schwenk and Gauvain, 2005; Schwenk, 2007). In such systems, the n-gram would be used for high-frequency words, with the neural model as the backoff. Alternatively, the neural LM can be used to rescore the top words predicted by the n-gram model. The approximate methods described above, together with advances in GPU technology have made such hybrid systems mostly outdated. n-grams and neural models were still routinely interpolated for a time, but strictly for the perplexity reduction achievable in this way.

1.5 Recurrent neural network language models

While Bengio et al. (2003) showed the potential of continuous space language modeling, it was not until the emergence of *recurrent neural network (RNN)* language models that such methods became widespread. The first system, based on "vanilla" RNN, was presented by (Mikolov, 2010), who continued to develop the concept and related techniques in Mikolov (2012), Mikolov; Kombrink, et al. (2011), and Mikolov; Deoras, et al. (2011). The first model based on the more advanced LSTM cell was proposed in (Sundermeyer et al., 2012).

RNN LMs have dominated the language modeling landscape ever since their introduction. Only in the last two years have attention-based models (see Section 1.6.1) become mature enough to compete with them. While attention-based models will probably emerge

victorious, RNN LMs can be regarded as being the first widely successful family of neural language models.

1.5.1 Recurrent neural networks

The modern notion of recurrent neural networks and their name was probably[17] established in (Jordan, 1986; Rumelhart et al., 1985). The idea of recurrence, however, had been around much longer, at least since Minsky and Papert (1969).

Neural networks can be thought of as *directed graphs*, where each node is a neuron and activation spreads along the directed edges (see Figure 1.2). In general, a network can be called *recurrent*, if this graph has cycles in it. In modern RNNs, however, units have recurrent connections only to units in their own layer. The left side of Figure 1.4 illustrates the idea with a single recurrent unit with input x, output y and *inner state* h[18]. Note that in the simplest case, $y = h$.

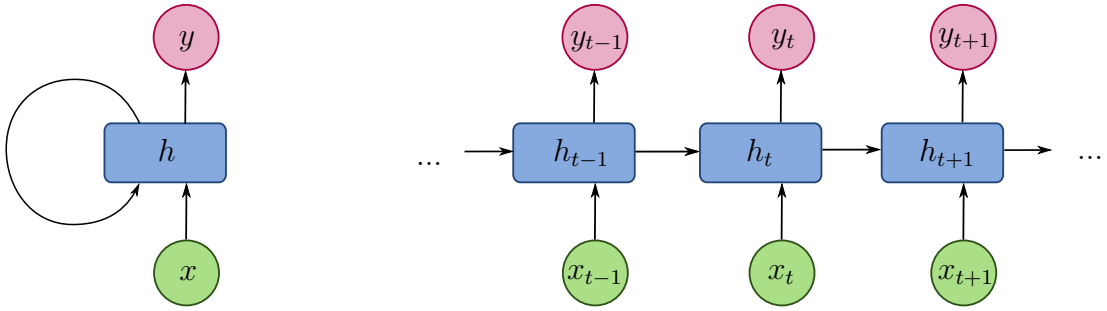


Figure 1.4: Left: a recurrent unit; right: the same unit unrolled for three timesteps

The recurrent connection can be thought of as memory, as it allows the layer to retain information from previous inputs. Because of this, RNNs lend themselves naturally to *temporal* or, more generally, *sequence modeling*. In these tasks, the inputs are part of a sequence, and the network has to predict the next item in the sequence based on the items seen thus far. This is in stark contrast to non-recurrent networks that processes training examples in isolation. Accordingly, RNNs have been used for tasks such as music composition (Mozer, 1992), speech recognition (Graves et al., 2013), handwriting recognition (Graves and Schmidhuber, 2009), sequence transduction (Graves, 2012), text generation (Sutskever et al., 2011) – or indeed, language modeling.

In the mathematic formulation, x, h and y are indexed with the time step $t \in 1...T$ to

[17] It is hard to find definitive evidence as to the origins of RNNs. Even high-profile papers, such as Hochreiter and Schmidhuber (1997), weasel out of the question by omitting citations altogether when introducing the concept.

[18] Based on https://commons.wikimedia.org/wiki/File:Recurrent_neural_network_unfold.svg

account for the temporal dimension. The formulae below are the recurrent counterpart to Equation 1.16 that describes the feedforward neuron:

$$\begin{aligned} h_t &= f(\mathbf{w}_{xh}^\intercal \mathbf{x}_t + \mathbf{w}_{hh}^\intercal \mathbf{h}_{t-1} + b_h) \\ y_t &= \mathbf{w}_{hy}^\intercal \mathbf{h_t} + b_y \end{aligned} \tag{1.21}$$

As before, f is the activation function; $\mathbf{w}_{xh}$, $\mathbf{w}_{hh}$ and $\mathbf{w}_{hy}$ are weights on the various connections; b_h and b_y are bias terms, to be trained along with the weight vectors.

RNNs are trained with the *Backpropagation Through Time (BPTT)* algorithm (Werbos, 1988; Williams and Zipser, 1995), which is a more involved version of regular backprop. In effect, the algorithm presents the input sequence $(\mathbf{x}_1, ..., \mathbf{x}_T)$ to the network one-by-one. The error gradients for all outputs $(\mathbf{y}_1, ..., \mathbf{y}_T)$ are collected, and propagated back to the weights of the network. The algorithm is based on an observation by Minsky and Papert (1969) that for every recurrent network, there exists a feedforward network with identical behavior (if T is finite). The algorithm converts the RNN to a multilayer feedforward network by *unrolling* it along the time dimension (as shown on the right side of Figure 1.4); the resulting network will thus have one layer for each time step. Backpropagation can then be applied to this network as usual.

As the length of the modeled sequences can be large, the unrolled feedforward network can be potentially be so deep that it does not fit into (GPU) memory[19]. To make training feasible, it is a ubiquitous practice, to split the sequence into T-sized chunks and treat each as a separate training example. The main drawback of this approach, a naive version of *truncated BPTT* (Williams and Peng, 1990), is that temporal dependencies crossing chunk boundaries are lost to training. One possible strategy to mitigate this issue is to sample the chunk size from a normal distribution centered on T, which makes sure that the chunk boundaries vary between epochs (Merity et al., 2018).

1.5.2 Gated architectures

Deep neural networks are difficult to train due to the *vanishing gradient problem* (Glorot and Bengio, 2010; Hochreiter, 1991). Since backpropagation uses the chain rule to compute the gradients for each layer, the error signal diminishes as it is propagated back from the last toward the first layers. This causes the earlier layers to effectively stop training. The

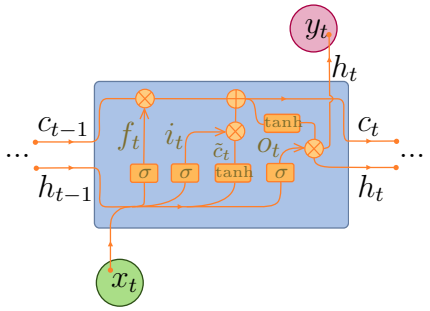
[19]For neural networks, it is always the accelerator (GPU or TPU) memory that counts, for two reasons. First, large networks can only ever be trained efficiently on accelerators; second, it is a more constrained resource than main memory.

opposite problem, that of *exploding gradients*, also exists: in this case, the gradients get exponentially large, inducing numerical errors in the weight update process.

The problem is even more pronounced for recurrent networks, where the layers share weights (Bengio et al., 1994; Hochreiter and Schmidhuber, 1997). It has been shown that if the norm of the hidden-hidden weigh matrix W_{hh} is less than one, the long term components of the gradient will vanish, preventing the network from learning long dependencies (Pascanu et al., 2013).

The exploding gradient problem can be easily remedied by clipping the norm of the gradients (Mikolov, 2012; Pascanu et al., 2013). For recurrent networks, the vanishing gradient problem is addressed by replacing the "vanilla" RNN with a gated architecture.

The *Long Short-term Memory (LSTM)* cell (Hochreiter and Schmidhuber, 1997) extends a standard recurrent unit with multiplicative gates to enforce constant error flow through the recurrent connection. It introduces another internal state variable, c_t, in addition to h_t, that is responsible for regulating the error flow. The cell architecture and its mathematical formulation are depicted in Figure 1.5.



$$
\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f]) \\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i]) \\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o]) \\
\tilde{c}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\
c_t &= f_t * c_{t-1} + i_t * \tilde{c}_t \\
h_t &= o_t * \tanh(c_t)
\end{aligned}
\tag{1.22}
$$

Figure 1.5: Structure and equations of the LSTM cell.

i_t, o_t and f_t are the input, output and forget gates, respectively. f_t might be the most important one, as it controls how much information should be retained from the previous state c_{t-1}. Its bias (b_f) is often initialized to 1.0 to allow free gradient backflow; in layman's terms, the cell starts remembering everything, and it gradually "learns to forget" (Gers et al., 2000).

The main reason why LSTM is less affected by vanishing gradients is the way its inner state is updated. While regular RNNs compute h_t by applying a vector product and an activation function to h_{t-1} (see Equation 1.21), LSTM computes the state *update* $\tilde{c}_t$ from c_{t-1}. The new value of c_t is then a linear combination of c_{t-1} and $\tilde{c}_t$, allowing the gradient update to flow through the cell unchanged (Jozefowicz et al., 2015).

Owing to its novel structure, the LSTM can "remember" much farther back in time than regular RNNs. In an artificial task, LSTMs could recall information about an item

after 1,000 time steps, while RNNs trained with BPTT failed after 10 (Hochreiter and Schmidhuber, 1997). LSTMs excel on real tasks as well; most of the RNN examples in the last section were realized with LSTM networks.

The success of LSTM prompted the appearance of other gated architectures. The GRU unit (Cho et al., 2014) is a simpler variant of LSTM, which omits the output gate. There were large-scale experiments to find which components make gated cells effective (Greff et al., 2015) and to find better architectures with a genetic algorithm (Jozefowicz et al., 2015) or reinforcement learning (Zoph and Le, 2017). While these experiments yielded insights into the inner workings of gated cells, no variant was significantly better than LSTM or GRU. In particular, LSTM was found to outperform all other cells (including GRU) in language modeling (Jozefowicz et al., 2015; Melis et al., 2018).

1.5.3 Language modeling advances

In this section, we examine how RNN-based, especially LSTM-based, systems pushed the state-of-the-art in language modeling. Most of the improvements were due to increasingly advanced machine learning techniques, as well as to our growing familiarity with how RNNs should be trained. We touch on these ideas very briefly; the interested reader is referred to the original papers for details.

Most of the research focused on modeling one of three corpora, which have become the de facto standard benchmarks for RNN LMs. Having standard datasets greatly facilitated the comparability and evolution of different approaches. Table 1.3 lists the main attributes of each corpus.

The Penn TreeBank (PTB) is the smallest of the three corpora. While the original corpus is behind a paywall, Mikolov (2012) has released a preprocessed version into public domain; it is this corpus we shall refer to as "PTB" henceforth. This is a rather small dataset at one million tokens, and is heavily preprocessed: numbers and punctuation marks have been omitted and all words have been lowercased. The vocabulary is capped at 10,000 word *types* (i.e. unique tokens).

The *WikiText-2 (WT2)* corpus (Merity; Xiong, et al., 2017) consists of curated Wikipedia pages. It is about twice the size of PTB, and its vocabulary consists of 30,000 words.

The *One Billion Word Benchmark (1B)* is a gigaword corpus released by Chelba et al. (2014) to measure progress in language modeling; it consists of about 800 million tokens. Words below 3 occurrences were replaced with `<UNK>` leaving a vocabulary of 793,471 word types. Due to its size, few papers took up the challenge, and even fewer attempted to model the whole vocabulary, opting instead to cap it at 10 or 100 thousand. This is the only corpus where the sentence order is randomized, preventing models from utilizing

long-range dependencies.

Dataset	Sentences	Tokens	Vocabulary	OOVs	Sentence order
PTB	49,199	1,134,978	10,000	53,299	normal
WT-2	–	2,506,962	33,254	81,561	normal
1B	30,607,716	829,250,940	793,471	509,258	shuffled

Table 1.3: Comparison of the three LM benchmark corpora

The PTB and WT2, at 1M and 2.5M tokens respectively, are very small as far as corpora go; (n-gram) language models were already trained on much larger corpora (365M in Brown et al. (1992) and 284M in Goodman (2001)). Models trained on the PTB and WT2 are prone to overfit the training data, making the method of regularization the deciding factor in LM performance.

Penn TreeBank

Tables 1.4 and 1.5[20] list the most influential systems developed for the PTB and WT2, respectively. In what follows, we only survey single model performance. Early papers often report results for ensemble models, which usually outperform the single model by 5–10 points of perplexity. Later papers omit these, most likely because under a certain perplexity threshold, ensembles ceased to provide any discernible benefit.

The first two lines in Table 1.4 represent the best discrete baselines on the PTB from Mikolov (2012); it also illustrates that the real impact of a cache over a state-of-the-art n-gram model is much less than the 60% reported by Kuhn and De Mori (1990) (see also Section 1.3.4). The same paper introduced the first RNN language model, which already outperformed the baseline by a small margin. Based on a vanilla RNN, it could not utilize the history to its full effect due to the vanishing gradient problem. Mikolov and Zweig (2012) dealt with the issue by encoding history into a "context vector" using Latent Dirichlet Allocation (LDA) (Blei et al., 2003). While this approach achieved a lower perplexity score, future systems would instead follow the example of Sundermeyer et al. (2012), who was the first apply LSTM to the problem of language modeling.

In experimenting with 2-layer LSTM networks, Zaremba et al. (2014) found that only small (200 cells per layer) models can be trained without regularization. *Dropout* (Srivastava et al., 2014), the most successful regularization technique for neural networks, however, did not work for RNNs at the time. Dropout induces noise in the training process by masking a different part of the network in each training batch, which forces the

[20]Tables 1.4 and 1.5 are slightly modified and extended versions of those in Merity et al. (2018)

Model	Parameters	Perplexity
Mikolov (2012) - KN 5-gram	2M	141.2
Mikolov (2012) - KN 5-gram + cache	2M	125.7
Mikolov (2012) - RNN	6M	124.7
Mikolov and Zweig (2012) - RNN-LDA	7M	113.7
Mikolov and Zweig (2012) - RNN-LDA + KN 5 + cache	9M	92.0
Sundermeyer et al. (2012) - LSTM	4.3M?	118.0
Zaremba et al. (2014) - LSTM (small)	4.6M	114.5
Zaremba et al. (2014) - LSTM (medium)	20M	82.7
Zaremba et al. (2014) - LSTM (large)	66M	78.4
Dyer et al. (2016) - RNN grammar	–	114.5
Gal and Ghahramani (2016) - Variational LSTM (medium)	20M	78.6
Gal and Ghahramani (2016) - Variational LSTM (large)	66M	73.4
Kim et al. (2016) - CharCNN	19M	78.9
Merity; Xiong, et al. (2017) - Pointer Sentinal LSTM	21M	70.9
Grave et al. (2017) - LSTM + continuous cache pointer	–	72.1
Inan et al. (2017)† - Variational LSTM (medium) + augmented loss	24M	73.2
Inan et al. (2017)† - Variational LSTM (large) + augmented loss	51M	68.5
Zilly et al. (2017)† - Variation RHN	23M	65.4
Zoph and Le (2017)† - NAS Cell (medium)	25M	64.0
Zoph and Le (2017)† - NAS Cell (large)	54M	62.4
Melis et al. (2018)† - 4-layer skip connection LSTM	24M	58.3
Merity et al. (2018)† - 3-layer AWD-LSTM	24M	57.3
Merity et al. (2018)† - 3-layer AWD-LSTM + continuous cache pointer	24M	52.8
Yang et al. (2018)† - AWD-LSTM + MoS	22M	54.44
Yang et al. (2018)† - AWD-LSTM + MoS + dynamic evaluation	22M	47.69
Gong et al. (2018)† - AWD-LSTM + cache pointer + FRAGE	24M	51.8
Gong et al. (2018)† - AWD-LSTM + MoS + dynamic evaluation + FRAGE	24M	46.54

Table 1.4: Single model perplexity of various neural (mostly RNN) LMs on the PTB. n-gram models are included for reference; † indicates tied input and output embeddings.

remaining units to learn better representations. Unfortunately, the recurrent connections in an RNN amplify this noise, which was found to hurt training.

Zaremba et al. (2014) sidestepped this problem by applying dropout on the forward connections between layers only. This allowed them to train LSTM LMs with 650 (medium) and 1,500 (large) cells per layer, outperforming even the interpolated RNN + n-gram model by 15%. Later, Gal and Ghahramani (2016) successfully applied dropout to the recurrent connections as well, by using the same mask in each time-step[21].

The Zaremba et al. (2014) model was the first to show the power of LSTM LMs, and has become widely successful. It quickly became the new baseline for language modeling

[21]Semeniuta et al. (2016) proposed applying dropout on the state update vector $\tilde{c}_t$. Sadly, the network used in the experiments was much smaller than those in Gal and Ghahramani (2016) and Zaremba et al. (2014), preventing direct comparison.

on the PTB; it was also one of the first models to be included in Tensorflow[22]. Its main disadvantage (especially in the large configuration) is the number of parameters it uses, which also negatively affects training times. Different systems came up with different answers to this issue. Kim et al. (2016) replaced the embedding with character n-gram features (followed by a *highway layer* (Srivastava et al., 2015) to transform orthographic features to semantic vectors), decreasing the number of parameters by 60%. Both Inan et al. (2017) and Press and Wolf (2017) found that sharing the weight matrix between the input embedding and the softmax layer (also called "output embedding") not only decreases the number of parameters by a wide margin, but also improves performance by increasing the frequency of updates to the embedding layer. Weight tying has thus become a common feature in all subsequent systems.

Architectural improvements seemingly also yielded considerable perplexity reductions. Zilly et al. (2017) developed a new network structure by arranging LSTM cells in a many-layer deep highway network. Parallelly, Zoph and Le (2017) employed a reinforcement-learning based architecture search to optimized the cell structure itself. Both claimed superiority over the by then traditional 2-layer LSTM setup. However, it was subsequently proven by an independent reevaluation of several architectures with large-scale hyperparameter tuning (Melis et al., 2018) that regular LSTM models actually outperform the more modern contenders.

An orthogonal line of research, which rekindled the idea of the (discrete) word cache, proved more successful. Following the prior work of Graves et al. (2014), Vinyals et al. (2015), and Weston et al. (2015) on *memory augmented networks*, both Pointer Sentinel (Merity; Xiong, et al., 2017) and continuous cache (Grave et al., 2017) models extend the base LM with a neural memory component. Using the memory as a model of recent history, both systems outperform their respective base models by 10–30 perplexity points. Similar to its discrete counterpart, the continuous cache requires no training and can be combined with any language model.

LSTM models also benefited from regularization techniques beyond dropout. Inan et al. (2017) derived an additional loss term, based on a better estimation for the true data distribution[23]; the gains were negligible (less than 1 PPL). AWD-LSTM (Merity et al., 2018), among *many* other tricks, applied *activation regularization (AR)* and *temporal activation regularization (TAR)* (Merity; McCann, et al., 2017) to the model. Most regularization methods, such as L_2, weight decay or dropout, affect the weights; instead, AR and TAR penalizes large activation values and large inner state shifts, respectively.

[22] https://github.com/tensorflow/models/tree/r1.10.0/tutorials/rnn/ptb

[23] To understand why this is important, refer back to Equation 1.18.

Together, they are responsible for about 3 perplexity points reduction.

Finally, machine learning experts identified weaknesses in the model outside the central RNN component. Gong et al. (2018) developed FRAGE, an embedding training method that improves the representation of rare words, achieving a slight perplexity reduction. Yang et al. (2018) found that the softmax layer presents a bottleneck for a high-rank problems such as language modeling. Replacing it with a *mixture of softmaxes* (MoS in Tables 1.4 and 1.5) yielded substantial improvements. Lastly, Merity et al. (2018) supplanted regular SGD with *NT-AvSGD*, a non-monotonically triggered, averaged version of SGD. According to ablation tests, this change in the optimization strategy alone was responsible for a 10% perplexity reduction on the PTB.

WikiText-2

Table 1.5 lists results for systems tested on the WikiText-2 corpus. Since its characteristics are close to the PTB, the same techniques proved effective there. Consequently, models perform similarly relative to each other, though the exact numbers differ from those in Table 1.4. An interesting observation can be made here: counting only the systems listed in both tables, while the perplexity scores start higher in Table 1.5, they also end lower, resulting in a 55% improvement on Inan et al. (2017), as opposed to the 32% on the PTB. To our knowledge, this discrepancy has, as of yet, not been addressed by any study.

Model	Parameters	Perplexity
Inan et al. (2017) - Variation LSTM	28M	87.7
Inan et al. (2017) - Variation LSTM + augmented loss	58M	87.0
Grave et al. (2017) - LSTM + continuous cache pointer	–	68.9
Melis et al. (2018) - 1-layer LSTM	24M	65.9
Merity et al. (2018) - 3-layer AWD-LSTM	33M	65.8
Merity et al. (2018) - 3-layer AWD-LSTM + continuous cache pointer	33M	52.0
Yang et al. (2018) - AWD-LSTM + MoS	35M	61.45
Yang et al. (2018) - AWD-LSTM + MoS + dynamic evaluation	35M	40.68
Gong et al. (2018) - AWD-LSTM + cache pointer + FRAGE	33M	49.3
Gong et al. (2018) - AWD-LSTM + MoS + dynamic evaluation + FRAGE	35M	39.14

Table 1.5: Single model perplexity of various LSTM LMs on WikiText-2.

Before turning to the 1B corpus, let us make a final remark about the accuracy of these results. We have seen that common datasets facilitate comparison; however, naively sorting systems by the published numbers may lead to invalid observations. Most systems use different code bases, are implemented with different deep learning libraries, and enjoy different levels of hyperparameter tuning. Without a thorough reevaluation (such as performed in Melis et al. (2018)), any survey is inevitably on a best-effort basis.

Luckily, most of the development in the field is open source, making such reevaluations possible. Authors customarily upload the code associated with their paper to GitHub or other software hosting sites. Reproducibility, however, can still be a problem. For instance, AWD-LSTM was originally implemented in PyTorch version 0.1.12. When adapted to the changes in PyTorch 0.4 or later, the reported perplexity numbers could no longer be replicated. As far as we know, no one has yet made the endeavor to run a full hyperparameter search to ensure that the model reaches its previous performance.

One Billion Word

Model	Parameters	Perplexity
Chelba et al. (2014) KN 5-gram	1.76B	67.6
Chelba et al. (2014) RNN-1024 + ME 9-gram features	20B	51.3
Jozefowicz et al. (2016) LSTM-1024-512	0.82B	48.2
Jozefowicz et al. (2016) LSTM-2048-512	0.83B	43.7
Jozefowicz et al. (2016) 2-layer LSTM-8192-1024 (Big)	1.8B	30.6
Jozefowicz et al. (2016) Big LSTM + CNN inputs	1.04B	30.0
Jozefowicz et al. (2016) Big LSTM + CNN inputs + CNN Softmax	0.39B	35.8
Jozefowicz et al. (2016) Big LSTM + CNN inputs + char LSTM	0.23B	47.9
Kuchaiev and Ginsburg (2017) Big LSTM G-4	1.75B?	28.17
Kuchaiev and Ginsburg (2017) Big LSTM G-4 (2 weeks)	1.75B?	24.29
Yang et al. (2018)* 2-layer LSTM	119M	42.77
Yang et al. (2018)* 2-layer LSTM + MoS	113M	37.1

Table 1.6: Single model perplexity of RNN LMs on the One Billion Word (1B) benchmark. The systems marked with an asterisk (*) are limited to the top 100k words.

Language modeling results for the One Billion Word benchmark are presented in Table 1.6. As before, the first block represent the best n-gram model and a mixed RNN–maximum entropy model by (Chelba et al., 2014). The latter shows clearly how the curse of dimensionality affects discrete models: the number of parameters increased tenfold from the 5-gram model.

The second block reports results for LSTM LMs that model the whole vocabulary. Since the corpus is huge, regularization is not such a burning issue as with the other two datasets. Jozefowicz et al. (2016) essentially experimented with inflated versions of the original (Zaremba et al., 2014) model; the largest LSTM architecture contains 8192 hidden units per layer. The models deviate from the Zaremba ones in one respect: a projection layer (Sak et al., 2014) is added before the softmax layer to keep the number of parameters in check. (This is the second number in the table: 2048–512 means hidden layer(s) with 2048 cells and a projection layer with 512 outputs.) After a week of training on 32 GPUs,

the largest models achieve perplexity scores around 30 – a 42% improvement over the best discrete model, and 55% over the best n-gram.

The main challenge on 1B is that of scaling: both the corpus and its vocabulary is huge, so computation and memory cost of the models must be controlled.

Jozefowicz et al. (2016) focused on making the input embedding and the softmax layer more efficient. Computational cost of the latter was addressed by using importance sampling instead of raw softmax. On the memory front, several techniques were tested. The model with "CNN inputs" reuses the idea of Kim et al. (2016) and replaces the embedding with character convolutions. This variant proved to be the best model, outperforming the raw LSTM model by a small margin. In contrast, CNN Softmax hurts language modeling performance, albeit at 35.8 PPL it still improves on the 5-gram model by 41%, while using 78% less parameters. Another variant predicted output words with a character LSTM. Although it needed the fewest parameters, it performed much worse than CNN Softmax.

Another way of attacking the performance problem is to make the LSTM network faster. Kuchaiev and Ginsburg (2017) developed a method to factor the weight matrix of an LSTM layer into smaller matrices, and the state and input vectors into 4 groups, decreasing the number of parameters and increasing parallelism. The resulting speed-up allowed the model to train faster, thereby achieving 2 points lower perplexity in the same timeframe as the previous best model.

The last two lines in the table show that the Mixture of Softmaxes approach is beneficial on 1B as well. However, the results are not comparable with the others, as only a fraction of the vocabulary was modeled.

1.6 Transformer-based language models

While RNNs had the language modeling stage to themselves for a good 6 years, they have recently been overshadowed by Transformer-based LMs. This section reviews what Transformers are, where they came from and how they are used in language modeling.

1.6.1 Neural machine translation

The origins of the Transformer can be traced back to the field of *neural machine translation (NMT)*. The history of MT starts shortly after the invention of the (electronic) computer (Dostert, 1955). By the end of the noughties, commercial grade systems, such as Google Translate, were widely in use, and research was carried out in both rule-based and statistical (classical machine learning) directions: see Apertium (Corbí-Bellot et al.,

2005; Forcada et al., 2011) for the former, and Moses (Koehn et al., 2007) for the latter. The first end-to-end NMT system was only introduced in 2014.

In his seminal paper, Sutskever et al. (2014) introduced the *sequence to sequence (seq2seq)* model, which is a generic framework for sequence learning. It consists of two components: an *encoder* and a *decoder*, both of which are LSTM networks; for machine translation in particular, both are LSTM language models. The encoder consumes the input sequence and maps it to a vector of fixed length. The decoder's hidden state is initialized from this vector, and it is run in generation mode to predict the output[24]. The system achieved close to state-of-the-art results. Figure 1.6 illustrates its application in English–French translation[25].

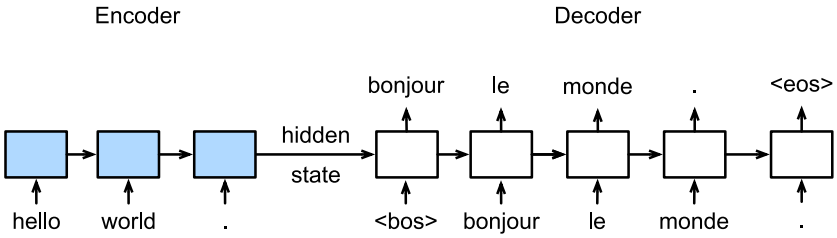


Figure 1.6: Sequence to sequence architecture at work

The main drawback of the seq2seq architecture is the bottleneck presented by the fixed length vector output by the encoder. In most cases, its capacity is not enough to represent all necessary information in the source sequence. The *attention* mechanism (Bahdanau et al., 2015; Luong et al., 2015b) alleviates this issue by allowing the decoder to access the hidden states of the encoder at each time step. This enables the decoder to find which source words are most relevant for predicting a particular output word. In effect, the model learns a *(soft) alignment* between the input and output sequences.

(Hard) word- (Brown et al., 1993) and phrase-based (Och et al., 1999) alignment has been an important part of the classical MT toolbox since IBM System 1. It is basically a bipartite graph, in which source and target-language words (phrases) are the nodes and an edge between two means that they are (part of the) translations of each other. Soft alignment is based on the same principles, but the alignments an output word participates in are represented by a probability distribution on the whole input sentence. In other words, an output word may *attend* to any number of input words to different degrees. Figure 1.7 illustrates the idea with a per-row heat map. Brighter squares indicate a higher level of attention[26].

[24] We do not go into details in this section, as it is only tangentially related to our main topic.
[25] Image taken from https://d2l.ai/chapter_recurrent-modern/seq2seq.html
[26] Heat map generated with https://git.io/JfITo

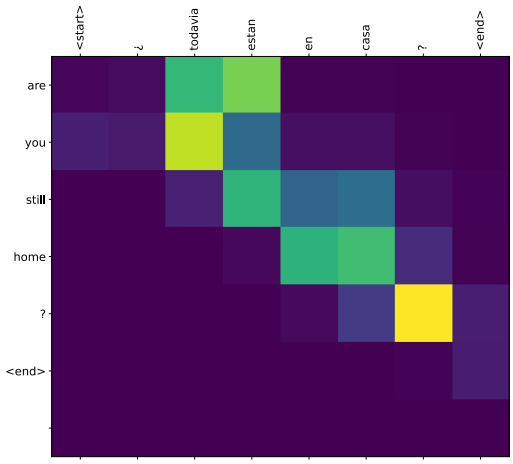


Figure 1.7: Soft attention plot for an English–Spanish sentence pair

Attention proved very effective in machine translation; the first system already outperformed Moses when `<unk>` tokens were disallowed (Bahdanau et al., 2015) and Luong et al. (2015a) established state-of-the-art results on several MT benchmarks.

1.6.2 The Transformer

The Transformer is a machine translation model that was introduced in the seminal paper "*Attention is all you need*" (Vaswani et al., 2017). It is an encoder-decoder architecture that does away with the RNN components and rely solely on the attention mechanism to model the input and output sequences, as well as their relation to each other.

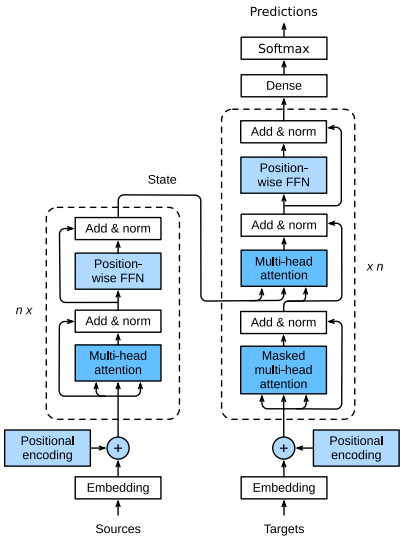


Figure 1.8: A single-layer Transformer, encoder on the left.

Both the encoder and the decoder consist of identical layers stacked on one another. An

encoder layer contains two sublayers: a FF component and a *self-attention* layer in which a position (word) attends to all positions in the output of the previous layer. The decoder layer is similar in structure, with two differences. First, it contains an encoder-decoder attention sublayer; this corresponds to the regular attention component in Bahdanau et al. (2015). Second, in the self-attention layer, words can only attend on positions on their left to make output generation possible. The original model had 6 encoder and decoder layers, but this number is not set in stone; Figure 1.8 portrays a single-layer model.

Two details of the architecture are worth mentioning. One is that Transformer uses *multi-head attention* in all layers. The multi-head attention mechanism builds several (one per "head") attention distribution for each word position, potentially drawing information from different representation subspaces (Vaswani et al., 2017). It can provide a richer representation than regular attention, though it was later found that many of the heads in the encoder can be pruned (Voita et al., 2019). The other detail is the augmentation of the regular embeddings with *positional encodings*. Since the architecture contains no recurrence, the temporal information necessary to model sequences is injected by adding to the embedding of each word the value of a sinusoid function of the word position.

The Transformer, at the very beginning, outperformed all previous single models on the two MT datasets it was tested on, even including all ensembles of earlier models on one.

1.6.3 Transformers in language modeling

We have seen how LSTM language models inspired neural machine translation, which in turn gave birth to the Transformer. It was only a matter of time until things went full circle and the first Transformer-based language models appeared.

Transformer LMs come in two forms, depending on the layer type used. Regular left-to-right language models are built on a variant of the decoder, which has the encoder-decoder attention sublayer removed (Liu et al., 2018). Bidirectional models use the encoder. Here we review the most successful autoregressive LMs; bidirectional models are described in Section 1.7.4.

Sadly, it is impossible to give a similarly detailed overview of Transformer language models as we did for RNNs in Section 1.5.3. As we shall see, the models are very large, and require huge amounts of data to train. The lack of large, openly available preprocessed corpora, however, prompted every research group to assemble its own corpus. This precludes meaningful comparison between the models, and the fact that their creators refrain from publishing their datasets or results on standard corpora makes attempts at reproducibility more or less futile and Table 1.7, which lists the most important Transformer

LMs, grievously incomplete.

The purpose of language modeling has changed as well. Previously, quantitative evaluation based on perplexity was the norm. Qualitatively, text generated by the models often left a lot to be desired. Transformer models, on the other hand, can generate much more consistent and syntactically (though not semantically) flawless texts. While this makes quantitative evaluation less relevant, it is not clear yet how these close-to-human-level models should be assessed.

Early decoder-based Transformer LMs were used for various tasks, such as document summarization (Liu et al., 2018) and generative pretraining for high-level NLP tasks (Radford et al., 2018). For these early systems, language modeling was but a tool, and not the goal; yet they laid the foundation for others to build on.

The first language model that showed the power of the Transformer approach was GPT-2 (Radford et al., 2019). It was trained on 40GB of web text, and made available in four configurations, with parameter budgets from 117M to 1.5B. The generated texts, especially for the large model, are qualitatively barely distinguishable from those written by humans[27]. The model achieves state-of-the-art performance on several LM benchmarks, with 1B being the notable exception (see Table 1.7). This may be justified by the fact that Transformer language models are typically trained with full documents in order to take advantage of long-term dependencies – something which the sentence-shuffled 1B lacks.

The CTRL model (Keskar et al., 2019) is similar to GPT-2, but it conditions the language model on control codes that can be used to dictate the style, topic and task-specific behavior (e.g. question answering) of the generated text. The mechanism works well for the most part, but abrupt style shifts do happen, and the control codes are haphazard, obviously based on what datasets were at hand (Wikipedia, MT tasks, etc.).

Model	Corpus	Parameters	PTB	WT2	1B
GPT (Radford et al., 2018)	Books (1B words)	110M	–	–	–
GPT-2 (Radford et al., 2019)	Web scrape (40GB)	117M	65.85	29.41	75.20
		345M	47.33	22.76	55.72
		762M	40.31	19.93	44.58
		1.5B	35.76	18.34	42.16
CTRL	Web scrape (140GB)	1.63B	–	–	–
Transformer XL (Dai et al., 2019)	Same as evaluation	24M–0.8B	54.52	–	21.8

Table 1.7: Performance of Transformer language models on standard corpora.

[27]So much so, that OpenAI only published the smallest model at first "*due to concerns about large language models being used to generate deceptive, biased, or abusive language at scale*" (Radford; Wu; Amodei, et al., n.d.). Only when other large models became available did they decide on releasing the rest of the models.

Both RNN and Transformer language models are trained by chunking the corpus into segments and processing a single segment in a training step. However, the recurrent hidden states of RNNs are preserved between segments, allowing them to retain information from previous steps as much as their capacity permits. Since Transformers have no recurrent state, they cannot learn dependencies that span segments. Transformer XL (Dai et al., 2019) overcomes this issue by allowing the model to look back (but not backpropagate to) the latest preceding segment, introducing a limited form of recurrence.

Transformer-XL is unique among the rest of the models in that it was trained and evaluated on the benchmark datasets instead of a proprietary corpus, allowing a direct comparison with RNN methods. Notably, it achieved state-of-the-art performance on all datasets.

1.6.4 Performance considerations

The Transformer architecture promised performance improvements over RNNs, at least as far as training speed is concerned. The speed-up comes from the lack of recurrent connections and the fact that all words are processed in parallel. The increased efficiency, however, is offset by the sheer size of modern Transformer models.

There is one aspect in which RNNs actually outperform Transformers: evaluation or text generation. At each time step, the RNN reads a single input word and predicts the next word (see e.g. the right side of Figure 1.6). This is possible because the RNN can "remember" the input history via its recurrent state. The Transformer, which lacks such a recurrent state, has to re-read the whole history sequence at each time step.

Yet the main bottleneck for attention-based models is memory. While RNNs run in memory linear in the sequence size n, attention uses $\mathcal{O}(n^2)$. When training a Transformer model on the same GPU as an RNN with the same parameter budget, either the sequence size or the batch size (or both) must be decreased to fit into memory. The first choice sacrifices model performance; the second leads to longer training times. Most models above or in Section 1.7.4 have many times the parameters of a standard LSTM model and are also trained on much larger corpora (see Table 1.7). These factors all add up, and as a result, training a modern Transformer model takes days or weeks on hundreds of GPUs or TPUs.

The hardware requirements and the length of the training cycle have left their mark on research culture as well. The financial costs incurred by such a training regimen are prohibitive for smaller laboratories, preventing them from participating in state-of-the-art research. To counter this 'big science' effect, it has become customary to release the models along with the paper introducing them, allowing less well endowed research

groups to use or experiment with them. However, this does not change the fact that the leaderboards of shared tasks are dominated by huge Transformer models created by large organizations (usually companies) (Rogers, 2019), and that the unequal playing field raises various ethical issues (Parra Escartín et al., 2017).

It is important to note that there already exist solutions to the technical issues raised above. The limited recurrence introduced by Transformer XL puts it on par with RNNs for evaluation speed, achieving a huge (uup to 1,800x) speed-up compared to regular Transformer models (Dai et al., 2019). The quadratic memory usage of the attention mechanism has also been addressed recently. The Reformer architecture uses locality-sensitive hashing to bring the memory footprint down to $\mathcal{O}(n \log n)$ (Kitaev et al., 2020). These solutions are not yet widespread, and it will be interesting to see how they affect the Transformer landscape.

There is one area where Transformer models improve on memory usage and computation cost compared to RNNs: their handling of the embedding and softmax layers. In RNN language models, the computational cost was addressed by approximate methods (see Section 1.4.4). Transformers mitigate the problem using a linguistically motivated approach. Since the size of the softmax layer is proportional to that of the vocabulary, decreasing the latter will keep the former small as well. An obvious solution would be to switch to character-based language modeling, where the vocabulary consists of a few hundred characters at most; however, their performance leaves a lot to be desired (Radford et al., 2019).

Subword- or *wordpiece*-level vocabularies, generated by algorithms such as *WordPiece-Model* (Schuster and Nakajima, 2012), *Byte Pair Encoding (BPE)* (Sennrich et al., 2016) or the confusingly named *unigram language model* (Kudo, 2018), represent a reasonable compromise between character- and word-level approaches. They include the most frequent character sequences in the training data, from individual characters to the most common words. Inputs to the language model are segmented into subword tokens according to the vocabulary; prediction is also done at the subword level, although perplexity scores are always normalized with the word count. Most models have a vocabulary of about 30,000 subwords.

Of course, subword vocabularies are not Transformer-specific, and can be used with any language modeling technique. However, it was Transformer LMs that made them ubiquitous. Yet their effect on the modeling performance has not been studied in much detail: the author is not aware of any work that evaluates word- against subword-level vocabulary with the same Transformer (or RNN) model.

1.7 Embeddings

We have seen how embeddings are an integral (and mandatory) part of neural language models (see Section 1.4.3. Much of the performance of a model depends on the quality of vector representations learned by the embedding. In this section, we explore why embeddings work and how they became a household term in NLP.

1.7.1 Vector space semantics

In the seminal paper dedicated to this topic, Mikolov; Yih, et al. (2013) discovered that embedding vectors encode meaningful syntactic and semantic information about words. The language model uses this information to make its predictions.

Embeddings map words to vectors in $\mathbb{R}^d$, which is an *inner product space*. It defines three vector operations: addition, scalar multiplication and dot product. It is not at all obvious that this should be the case, but all three operations have a linguistic interpretation:

Similarity Embeddings assign similar words to similar vectors (Bengio et al., 2003). Normalized dot product (i.e. the cosine of the angle between two vectors) is therefore a good measure of word similarity.

Relatedness Syntactic and semantic relationships seem to be encoded as approximately constant vector offsets between word pairs sharing a relation (Mikolov; Yih, et al., 2013). Figure 1.9 illustrates a semantic (gender) and a syntactic (plural) relation[28].

Frequency Since the length of a vector seems proportional to the logarithm of the word frequency (Arora et al., 2016), scalar multiplication or normalization of a vector does not change its semantics. This property ensures that similarity and relatedness can be computed as described above.

The properties above make it possible to treat more advanced concepts, such as *analogy*, in terms of vector operations as well. The question "*What is to king as woman is to man?*" can be formalized as $v_{king} - v_{man} + v_{woman}$, which (given a large enough training corpus) yields a vector close to v_{queen}. Syntactic analogies (e.g. $v_{apple} : v_{orange} \approx v_{apples} : v_{oranges}$) work similarly. Later studies showed that embeddings capture all kinds of linguistic information about words from POS category (Borbély; Kornai, et al., 2016) to sentiment or concreteness (Rothe et al., 2016).

[28]Figure reproduced from Mikolov; Yih, et al. (2013).

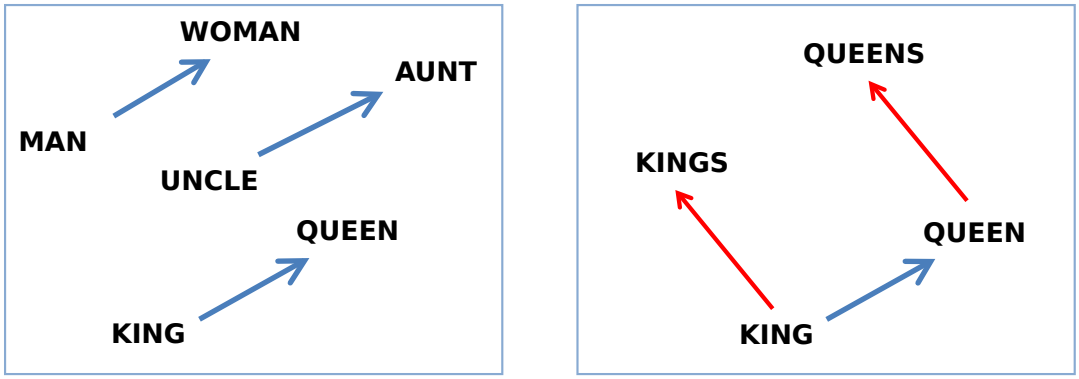


Figure 1.9: Left panel shows vector offsets for the "gender" relation. Right panel shows two relations: "gender" and (the grammatical) "number".

As embeddings use real vectors for word representation, they belong to the *vector space models of semantics*. Another name for this field is *distributional semantics*, because it is based on the *distributional hypothesis* (Firth, 1957; Harris, 1954), which asserts that similar words occur in similar contexts. Since language models use the context to predict the next word, the quality of representations learnt this way is a strong argument in favor of the hypothesis.

Distributional semantics is not a new concept. Word frequency count-based methods, such as *Latent Semantic Analysis (LSA)* (Deerwester et al., 1990) or *Pointwise Mutual Information (PMI)* (Church and Hanks, 1990) have been around for decades. There are conflicting assessments about whether embeddings clearly outperform (Baroni et al., 2014), or are mostly on par with (Levy et al., 2015), count-based models; there is some evidence that both optimize the same objective (Levy and Goldberg, 2014b). However, embeddings clearly improve on earlier methods in two regards. First, they capture syntactic and morphological regularities in addition to semantics. Second, they are much lower-dimensional, typically d =300–1000, as opposed to $\mathcal{O}(|V|)$ in the count-based case, allowing them to be used outside of research environments.

Embeddings can be evaluated in two ways. *Intrinsic* evaluation assesses the learned representations on similarity (such as SimLex-999 (Hill et al., 2014)) or analogy (see e.g. WS-353 (Finkelstein et al., 2002) or the Google analogy dataset (Mikolov; Yih, et al., 2013)) tasks. *Extrinsic* evaluation measures what performance improvements an embedding brings to downstream tasks (see Section 1.7.5). Unfortunately, there seems to be little correlation between the two approaches (Faruqui et al., 2016). While earlier work concentrated on intrinsic benchmarks, the focus has recently shifted to extrinsic evaluation.

The vector space properties were first demonstrated for embeddings of RNN language

models. However, their performance is usually substandard, due to the lack of a right-hand side context and the generic LM objective (Mikolov; Chen, et al., 2013). Stand-alone embeddings algorithms, better suited to representation learning, soon appeared. They belong to two main groups: *static embeddings* assign a single vector to a word *type*; i.e. all occurrences of a word are mapped to the same vector; *dynamic (contextualized) embeddings* compute a different vector for each occurrence based on its immediate context.

1.7.2 Static embeddings

The first general purpose embedding, *word2vec (w2v)* was published in the seminal Mikolov; Chen, et al. (2013), although a similar method has already been described in Collobert and Weston (2008). Word2vec is trained with one of two language modeling objectives based on skip-grams. Four words are selected in a local context window of k positions around the current word, and either the current word has to be predicted based on the other four, or conversely. The architecture is based on the feedforward NNLM in Section 1.4.3, but without the hidden layer. Word2vec significantly outperforms embeddings extracted from early RNN models, and it is very fast to train (Mikolov; Chen, et al., 2013). It has become the first deep learning success story in NLP, even though its NN architecture is anything but deep.

Word2vec was followed by other embedding algorithms, all of which tried to improve on it in various ways. *GloVe* (Pennington et al., 2014) mixes local context window methods with global corpus statistics; Levy and Goldberg (2014a) eschews linear context in favor of a syntactic one based on dependency parse-trees. FastText (Bojanowski et al., 2017) represents words as a bag of character n-grams. Yet performance-wise, neither method is superior to the others: some of them are better at syntactic, others at semantic tasks; some at word similarity, others at relatedness (Levy et al., 2015). Nowadays, FastText is usually preferred, because of its ability to model out-of-vocabulary words.

Besides OOVs, embeddings algorithms have problems learning good representations for rare words as well. The neighbors of infrequent words tend to be other, unrelated rare words, with actual semantic neighbors placed far away (Gong et al., 2018); their neighborhoods are also less stable across different training runs (Wendlandt et al., 2018). There are attempts to address this issue directly, such as FRAGE (Gong et al., 2018), but due to this instability, hyperparameter choice is as important as ever (Levy et al., 2015; Mimno and Thompson, 2017).

1.7.3 Multi-sense embeddings

The embeddings discussed thus far represent each word with a single vector. This is problematic for words with multiple meanings, such as *bank* 'financial institute' and 'river bank'. Bank is a *homonym*, i.e. its senses are unconnected, and it is only a coincidence that they share the same word form – evidenced by the fact that other languages do not partition these meanings to the same character sequence (Youn et al., 2016). The algorithm, however, tries to consolidate the senses into one, ending up with a vector that is a blend of the different meanings. *Polysemes* such as *head* ('body part', 'top part', 'leader'), where the senses are closely related, are perhaps less affected, as we expect their vectors to be close as well. Yet there is a continuum of words between homonymy and polysemy, and for most, the meanings (or usage patterns) are separate enough to warrant distinct vectors.

Multi-sense embeddings (MSEs), proposed by Reisinger and Mooney (2010), model different meanings of word forms with different vectors. Several algorithms have been proposed over the years using techniques such as spherical clustering (Huang et al., 2012), stochastic modeling (Li and Jurafsky, 2015), or multi-sense extensions to w2v (Bartunov et al., 2016; Neelakantan et al., 2014). Some of these are simpler, assigning a fix number of senses for lexically ambiguous words, others try to find the optimal number statistically. All of them report improvements over single-sense embeddings, though the gains seem surprisingly small even on datasets created specifically to test MSEs, such Stanford Contextual Word Similarities (Huang et al., 2012).

Assessing MSE performance is a complex matter; we return to in Chapter 3.

1.7.4 Contextual word embeddings

The main drawback of static embeddings is that a word is represented by the same vector regardless of context. Even current MSEs have to choose between a number of precomputed sense vectors for each occurrence of the word. As such, when encountering a word in a specific sentence, static embeddings cannot accurately reflect the syntactic or semantic role it plays in its current context.

In contextual embeddings, such as ELMo (Peters et al., 2018) or BERT (Devlin et al., 2019), the vector of a word depends on its immediate surroundings as well. Accordingly, each occurrence of a word is assigned a different vector, which is then able to implicitly encode the function of the word in the sentence.

We have seen how static embeddings came to be by "shedding" their language modeling skin and keeping only the embedding matrix. Contextual embeddings mark the return of

the full language modeling machinery; in effect, a contextual embedding is nothing more than a (usually bidirectional) language model, where the word representation is taken from the output of the network (or specific hidden layers). The input is usually a full sentence. This allows the model to adapt the word representations to the context; hence the name. This is in contrast to static embeddings, where the vector is the output of the embedding layer, which operates on a per-word basis.

Contextual embeddings are *pretrained* on very large corpora upwards of 1B (One Billion Word (Chelba et al., 2014)) or 2,5B (English Wikipedia[29]) tokens all the way up to 2.5TB of filtered Common Crawl data (Conneau and Lample, 2019). They are evaluated on high-level downstream tasks, such as question answering, textual entailment or named entity recognition; for contextual embeddings, the idea of intrinsic evaluation has been completely abandoned.

The first contextual embedding, *ELMo* (*Embeddings from Language Models*, Peters et al. (2018)) employs a single-layer bidirectional LSTM (i.e. a left-to-right and a right-to-left sublayer); the word representation is the outputs of the two sublayers concatenated. ELMo can be used as a drop-in replacement for static embeddings in NLP systems; such systems attained state-of-the-art results on SQuAD and four other high-level tasks.

Besides English, ELMo models have been released for another 43 languages (Che et al., 2018). However, the training data for each language was limited (20M words apiece), so the models are not expected to perform as well as 'regular' ELMo.

Keeping with the Sesame Street[30] theme, ELMo was soon followed by *BERT* (*Bidirectional Encoder Representations from Transformers*, Devlin et al. (2019)). As its name implies, it repurposes the Transformer encoder as a generic language representation model. It is pretrained on two tasks. In *masked language modeling*, the system is presented with a single sentence in which some words are masked. The model has to guess these words[31]. This is analogous to the traditional LM objective of guessing the next word; however, the parallel processing of words in the Transformer encoder makes standard left-to-right conditioning impossible. In the *next sentence prediction* task, BERT is presented with two sentences, and has to decide whether the second sentence follows the first in the training corpus.

The model can be adapted to downstream tasks by adding a classifier on top. Even with a simple feed-forward network as the classifier, BERT was able to achieve state-of-the-art performance on several high-level benchmarks, such as SQuAD or GLUE (Devlin et al., 2019). The model is *fine-tuned* separately for each task. While pretraining BERT is

[29] https://en.wikipedia.org/wiki/Main_Page
[30] https://en.wikipedia.org/wiki/Sesame_Street
[31] In linguistics, masked LM is known as a *cloze test* (Taylor, 1953)

very costly, fine-tuning is relatively inexpensive, and (depending on the size of the training data) takes only a few hours on a modern GPU.

The English BERT models are available in two sizes: the `Base` model has 110 million parameters, BERT `Large` 340 million; each with a cca. 30 thousand wordpiece vocabulary. There is also a multi-language model that was pretrained jointly on 104 languages, with a vocabulary of approximately 120 thousand wordpieces.

BERT begat a whole family of models. One line of research kept the architecture and sought improvements by finetuning its details: XLNet (Yang et al., 2019) enhanced handling of masked tokens; *SpanBERT* (Joshi et al., 2020) extended the masking scheme to word spans and added a span boundary detection pretraining task. *ALBERT* (Lan et al., 2019) replaced the next sentence prediction task with the more general sentence order prediction, while *RoBERTa* (Liu et al., 2019) abandoned the next sentence prediction task altogether. It also used larger batches and trained with a dynamically changing masking pattern.

Other systems, such as *BART* (Lewis et al., 2019) and *UniLM* (Dong et al., 2019), opted to use the full Transformer architecture instead of just the encoder and introduced sequence-to-sequence training objectives. Finally, the XLM family of models (Conneau and Lample, 2019) focused on improving multi-language performance, and *XLM-RoBERTa* is even competitive with monolingual models (Conneau et al., 2019). All models reported state-of-the-art results on various downstream tasks and benchmarks.

Pretraining a BERT (or derived) model is no easier or cheaper than training an autoregressive Transformer LM (see Section 1.6.4). *DistilBERT* (Sanh et al., 2019) aims to alleviate the impact on end users by training a leaner BERT model via *knowledge distillation* (Hinton et al., 2015); i.e. with the objective to reproduce the original model's behavior. ALBERT incorporates cross-layer parameter sharing, making the model not only cheaper to use, but faster to train as well. Yet pretraining a state-of-the-art contextual embedding is still out of the reach of most research groups.

1.7.5 Embeddings in NLP

In NLP systems, most sequence- or token classification tasks (POS tagging, NER, sentiment analysis, etc.) are solved by machine learning. These systems typically consist of two parts: a *featurizer* that creates features for each token of its input (generally, a word) and a *classifier*, that takes these features as its input and, depending on the task, emits a token or sequence label. In the traditional paradigm, features are created manually, based on the linguistic and orthographic properties of words. The features are then fed into a usually off-the-shelf classifier, such as a logistic regression or conditional random

field (CRF) model.

The idea that word embeddings can supplant manual features goes back to (Collobert and Weston, 2008; Turian et al., 2010). Collobert et al. (2011) trained a neural tagging system jointly for several NLP tasks (POS, chunking, NER and semantic role labeling). The results with purely automatic features were already competitive with the state-of-the-art, and adding manual features allowed the system to slightly improve on them.

The first public NLP pipeline to adopt word embeddings was Stanford CoreNLP, where they were used to augment manual features in the parser (Socher et al., 2013). Today, word vectors can be found in most NLP software packages (spaCy, UDPipe[32], etc.). To our knowledge, however, the only NLP library to fully eschew manual features in favor of pretrained vectors is Flair[33]; in fact, the core idea of Flair is to make "mixing and matching" various embeddings as simple as possible (Akbik; Bergmann; Blythe, et al., 2019). At the time of writing, Flair represents the state-of-the-art in POS tagging, chunking and NER (Akbik et al., 2018; Akbik; Bergmann, and Vollgraf, 2019).

Deep contextualized embeddings have not yet reached this level of penetration. While there exist libraries that allow their integration into NLP pipelines (such as `spacy-transformers`[34]), most of the models are published as-is, with only the scripts required to reproduce results in the associated papers. Also, BERT and its lineage excel in higher level tasks, such as question answering and the various language understanding tasks in GLUE, which are not regularly part of a text processing pipeline. However, if the ubiquity of static embeddings is any indication, this is bound to change in the near future.

1.8 Language modeling and NLP

Reading the previous sections, an attentive reader might have noticed how little role linguistics, or NLP, played in the evolution of language modeling techniques. Even though language modeling has its origins in two NLP tasks: speech recognition and machine translation, there seems to be little influx of ideas from linguistics to the new field. The basic axioms (i.e. languages have vocabularies that consist of words; (written) words are composed of characters, etc.) are exceptions, but those can hardly be considered revolutionary.

There are linguistically inspired techniques, such as subwords, vocabulary clustering in class-based n-grams and hierarchical softmax, or the idea of training bidirectional Trans-

[32] https://ufal.mff.cuni.cz/udpipe
[33] https://github.com/zalandoresearch/flair
[34] https://github.com/explosion/spacy-transformers

former LMs with the Cloze task. But only a handful of systems (such as the joint LM of Filimonov and Harper (2009), the RNN grammar of Dyer et al. (2016) and the dependency-based embedding of (Levy and Goldberg, 2014a)) used actual NLP techniques (CFG features in the former and dependency parsing in the latter two) to address issues in language modeling. Even then, these systems are regularly outperformed by models built without any (deep) linguistic thought; see Section 1.5.3.

Yet the opposite direction seems exceptionally fruitful: modern LMs generate texts of human-level local consistency, while embeddings have revolutionized how NLP systems are built. The difference is especially palpable on high-level natural language understanding tasks, which suddenly seem doable. While the AlphaGo (Silver et al., 2016) moment of NLP has not yet come, it certainly seems closer than ever before.

In a way, these developments are not surprising. Linguists have long struggled to explain the workings of language; and given the number of competing theories, it may safely be said that they have not yet succeeded. Yet humans have no issues using language, irrespective of the level of grammar education they might have had. There is one crucial difference between LM training and first language acquisition: modern language models need orders of magnitude more text to reach the performance reported in the papers than humans do to learn their mother tongue. There is still room for improvement.

As there might also be room for NLP in language modeling. We might observe that the results presented in this chapter were all achieved by English-language models on English benchmarks. Due to it being an analytic language, English lends itself well to word-level language modeling. Agglutinative languages, such as Hungarian, might pose problems for the language modeling techniques presented above. In this thesis, we explore some of the ways in which language models and linguistic methods interact, with a major focus on Hungarian.

Chapter 2

`emLam` – a Hungarian Language Modeling baseline

2.1 Introduction

We saw in Chapter 1 how language modeling is an integral part of several NLP applications, such as speech recognition, optical character recognition and machine translation. It has been shown that the quality of the LM has a significant effect on the performance of these systems (Brants et al., 2007; Chelba et al., 2012). Accordingly, evaluating language modeling techniques is a crucial part of research. For English, a thorough benchmark of discrete techniques was carried out by Goodman (2001); and Section 1.5.3 demonstrated how the availability of standard benchmark corpora facilitated progress in neural language modeling. All three corpora mentioned there, the preprocessed version of the Penn TreeBank (PTB), WikiText-2 (WT2) and the One Billion Word Benchmark (1B), were published for the sole reason of measuring advances in statistical language modeling.

Chapter 1 has also shown the dramatic advances the last decade saw in language modeling. Training corpora grew from a few million words (e.g. the Brown corpus) to gigaword and beyond, while vocabulary size increased from a few 10k to several hundred thousands (as in the 1B). Neural networks overtook n-grams as the language model of choice. State-of-the-art LSTM and Transformer models achieve up to 63–67% reductions in perplexity compared to 5-gram models (see Tables 1.4, 1.6 and 1.7).

Surprisingly, these developments left few traces in the Hungarian NLP literature. Aside from an interesting line of work on morphological modeling for speech recognition (Mihajlik et al., 2010; Németh et al., 2007), no study is known to the author that addresses issues of Hungarian language modeling. While quality works have been published in related fields, language model performance is often not reported, or is not competitive: e.g. in their

otherwise state-of-the-art system, Tarján et al. (2016) use a 3-gram model that achieves a perplexity of 400[1] on the test set — a far cry from the numbers reported in Chapter 1 and here.

Hungarian poses a challenge to word-level LM because of its agglutinative nature. While verb conjugation is mostly fusional (i.e. person, number and tense are fused into a single inflectional suffix, such as in "*lát·tuk*" 'we saw'), verbs still have about 40 surface forms, as opposed to the maximum four[2] in English. The nominal paradigm, on the other hand, is fully agglutinative, so nouns, adjectives and numbers may have as many as 700 different inflected forms.

The proliferation of word forms inflates the vocabulary, which has various adverse effects. Firstly, it increases the memory usage of all language models and increases the computational cost associated with the softmax layer (see Section 1.4.4). Secondly, and more importantly, it decreases the number of contexts a word form is seen during training (and conversely: the number of word forms seen after a specific context), making the data sparsity problem much more pronounced than it is for English. The results are worse probability estimates and ultimately, lower performance. Lastly, even frequent words will have forms not present in the training corpus, increasing the number of OOVs in open-vocabulary language modeling.

In this chapter, we intend to decrease the gap between English and Hungarian language modeling in two ways. First, we report baselines for various language modeling methods on three publicly available Hungarian corpora. In light of the issues mentioned above, it will be especially interesting to see how the performance of the tested methods translate to Hungarian.

Second, we present a version of the Hungarian Webcorpus (Halácsy et al., 2004) that can be used as a benchmark for language models. Our motivation was to create the Hungarian equivalent of the One Billion Word Benchmark corpus for English: a freely available data set that is large enough to enable the building of high-quality LMs, yet small enough not to pose a serious barrier to entry for researchers. We hope that the availability of the corpus will facilitate research into newer and better LM techniques for Hungarian.

The software components required to reproduce this work, as well as the benchmark corpus, comprise the emLam module[3] of e-magyar[4] (Váradi et al., 2017). The scripts have been released as free software under the MIT license, and can be downloaded from the

[1] Personal communication with the author.
[2] Namely present, past simple, past participle and gerund.
[3] http://e-magyar.hu/hu/textmodules/emlam
[4] http://e-magyar.hu

emLam repository[5].

The rest of the chapter is organized as follows. The benchmark corpora, as well as our solution to the data sparsity problem is described in Section 2.2. In Section 2.3, we introduce the LM methods to be evaluated. Results are presented in Section 2.4. Finally, Section 2.5 contains our conclusions and ideas left for future work.

2.2 The Hungarian Datasets

We selected three publicly available Hungarian corpora for benchmarking. The corpora are of various sizes and domains, which enabled us to evaluate both small- and large-vocabulary LM configurations. The corpus sizes roughly correspond to those of the English corpora commonly used for LM benchmarks, making a comparison between the two languages easier.

The Szeged Treebank (Csendes et al., 2003; Vincze et al., 2014) is the largest manually annotated corpus of Hungarian. The treebank consists of CoNLL-style tsv files; we used a version in which the morphological features had been converted to KR codes to keep in line with the automatic toolchain described below. At around 1.5 million tokens, it falls between PTB and WT2 in size, allowing us a direct comparison of small-vocabulary LM techniques.

The filtered version of the Hungarian Webcorpus (Halácsy et al., 2004) is a semi-gigaword corpus at 589M tokens. It consists of webpages downloaded from the `.hu` domain that contain no more than 4% word tokens unrecognized by HunSpell; i.e. "fewer typos than average printed materials" (Halácsy et al., 2004). The downloadable corpus is already tokenized; we further processed it by performing lemmatization, morphological analysis and disambiguation with Hunmorph (Trón et al., 2005): `ocamorph` for the former two and `hunlex` for the latter.

The Hungarian Gigaword Corpus (Oravecz et al., 2014) is the largest public Hungarian corpus. It is based on the Hungarian National Corpus[6] (Váradi, 2002), a 187M word corpus with newswire, literature, scientific and legal texts supplemented with material downloaded from online forums. This core was gradually expanded to 1.5 billion tokens with additional data crawled from the web. In this paper, we used an in-progress version, denoted MNSZ2. At around one billion tokens, it is comparable in size to the English 1B corpus. The raw text was preprocessed with the same tools as above.

We decided to use the 'old' `hun*` tools because at the time of writing, the `e-magyar`

[5] https://github.com/dlt-rilmta/emLam
[6] Magyar Nemzeti Szövegtár

toolchain was not yet production ready, and the version of the Szeged corpus that uses the new universal POS tags still contained conversion errors. Therefore, the results published here might be slightly different from what one can attain by running the scripts in the `emLam` repository, should the issues above be addressed. However, any such differences will most likely be inconsequential.

2.2.1 Preprocessing

As mentioned before, the main challenge of modeling an agglutinative language is the number of distinct word forms. The solution that works well for English — putting all word forms into the vocabulary — is not reasonable: on one hand, the vocabulary size would explode (see Table 2.1); on the other, there is a good chance the training set does not contain all possible word forms in the language.

The most common solution in the literature is to break up the words into smaller segments (Afify et al., 2006; Botha and Blunsom, 2014; Hirsimäki et al., 2005). The two main directions are statistical and morphological word segmentation. In this chapter, we study the latter. Not only is it linguistically more motivated, it also ensures that the tokens we end up with are meaningful, making the LM easier to debug.

We ran the pipeline described above on all words in the corpus, and split all inflectional suffixes (as well as some derivational ones, such as `<COMPAR>`, the tag for the comparative marker '*-bb*' and `<SUPERLAT>`, the tag for the superlative marker '*leg-*') into separate tokens. Only inflections marked by the KR code are included; the default zero morphemes (the nominative case marker and the present-tense third person singular for verbs) are not. A few examples:

```
jelmondatával  →  jelmondat <POSS> <CAS<INS>>
      akartak  →  akar <PAST> <PLUR>
 otthonainkba  →  otthon <PLUR> <POSS<1> <PLUR>> <CAS<ILL>>
```

One could say that by normalizing agglutinative forms like this, we "deglutenized" it; therefore, the resulting variants of the corpora shall be referred to as *gluten-free (GLF)* from now on. The full preprocessing pipeline is as follows:

1. Tokenization and normalization. The text was lowercased, converted to `utf-8` and deglutenized

2. (Webcorpus only) Duplicate sentences were removed, resulting in a 32.5% reduction in corpus size.

3. Tokens below a certain frequency count were converted into `<unk>` tokens. The word distribution proved different from English: with the same threshold as in the 1B corpus (3), much more distinct types remained. To be able to test LMs with a vocabulary size comparable to 1B, we worked with different thresholds for the two gigaword corpora: Webcorpus was cut at 5 words, MNSZ2 at 10. An additional thresholding level was introduced at 30 (50) tokens to make RNN training tractable.

4. Sentence order was randomized. For the Szeged corpus, we also created a version without sentence shuffling. The two are differentiated with subscripts: $_{\text{S}}$ marks the shuffled version, while $_{\text{O}}$ the corpus with the original sentence order.

5. The data was divided into train, development and test sets; 90%–5%–5% respectively. For shuffled datasets, this is a simple matter of random sampling. With Szeged$_{\text{O}}$, the situation is not so straightforward: from the 15 files of the corpus we could not single out a single one as test, as they all represent different genres or domains; nor could we randomly pick 5% of all documents, as the boundaries are unmarked. In the end, we set apart the last 10% of each file for the validation and test sets.

2.2.2 Corpus Statistics

Table 2.1 lists the main attributes of the datasets created from the three corpora. Where not explicitly marked, the default count threshold (3) is used. It is clear from comparing the raw and GLF datasets that deglutenization indeed decreases the size of the vocabulary and the number of OOVs by about 50%. Although not shown in the table, this reduction ratio remains consistent among the various thresholding levels.

Also apparent is that, compared to the English corpora in Table 1.3, the number of unique tokens is much bigger even in the default Hungarian GLF datasets. Preliminary inquiry into the data revealed that three phenomena account for the majority of the token types between the 3 and 30 (50) count marks: compound nouns, productive derivations and named entities (with mistyped words coming in at fourth place). Since neither the Szeged corpus, nor (consequently) the available morphological disambiguators take compounding and derivation into account, no immediate solution was available for tackling these issues. Therefore, we decided to circumvent the problem by introducing the higher frequency thresholds and concentrating on the problem of inflections in this study.

The preprocessing scripts are available in the emLam repository.

Dataset	Sentences	Tokens	Vocabulary	OOVs	Analysis
Szeged	81,967	1,504,801	38,218	125,642	manual
Szeged GLF		2,016,972	23,776	55,067	
Webcorpus	26,235,007	481,392,824	1,971,322	5,750,742	automatic
Webcorpus GLF		683,643,265	960,588	3,519,326	
Webcorpus GLF-5		id.	625,283	4,647,706	
Webcorpus GLF-30		id.	185,338	9,393,015	
MNSZ2	44,329,309	624,830,138	2,988,629	11,614,583	automatic
MNSZ2 GLF		852,232,675	1,714,844	5,729,509	
MNSZ2 GLF-10		id.	630,863	10,845,301	
MNSZ2 GLF-50		id.	197,542	19,547,859	

Table 2.1: Comparison of the three Hungarian corpora

2.2.3 The Benchmark Corpus

Of the three corpora above, the Hungarian Webcorpus is the only one that is freely downloadable and available under a share-alike license (Open Content[7]). Therefore, we decided to make not only the scripts, but the preprocessed corpus as well, similarly available for researchers.

The corpus can be downloaded as a list of tab-separated files. The three columns are the word, lemma and disambiguated morphological features. A unigram (word and lemma) frequency dictionary is also attached, to help create count-thresholded versions. The corpus is available under the Creative Commons ShareAlike (CC SA) license.

Such a corpus could facilitate language modeling research in two ways. First, any result published using the corpus is easily reproducible. Second, the fact that it has been preprocessed similarly to the English 1B corpus, makes comparisons such as those in this paper possible and meaningful.

2.3 Language model evaluation

We had neither the time, nor the means to evaluate all (or even most) of the language modeling methods described in Chapter 1. We opted instead to test the ones that were sufficiently state-of-the-art and either had an implementation available or could be recreated with reasonable effort.

We chose a 5-gram model with modified Kneser-Ney (KN) smoothing as our discrete baseline, since it is simple, and it reportedly outperforms all other n-gram models (Good-

[7] http://www.opencontent.org/definition/

man, 2001). We used the implementation in the SRILM (Stolcke et al., 2011) library, and tested two configurations: a pruned backoff (the default)[8] and, similar to Chelba et al. (2014), an unpruned interpolated model[9]. All datasets described in Table 2.1 were evaluated; in addition, we also tested POS models (both full-KR and GLF, where the inflectional tags are split from the main category), where lemmas were replaced with their respective part-of-speech tags. These test the model's ability to learn basic syntax and morphotactics (as encoded by POS sequences) and it also serves as the state transition table for class-based n-grams (see below). Due to the considerably reduced vocabulary (less than two thousand for full KR and two hundred for GLF), this task is much easier than word-level language modeling.

It is worth mentioning that the GLF format (and subwords in general) is not optimal for n-gram modeling. As words are broken up into multiple tokens, they use up more of the n-gram's history, which is already a very limited resource. In the extreme case, such as the last example under Section 2.2.1, the whole history could be taken up by inflectional tokens, preventing the model from accessing relevant context. RNN LMs are less affected by this issue as they can retain information from their history much longer.

We also evaluate a class-based n-gram setup to see if they can improve on regular 5-grams. Our working hypothesis, based on Section 1.3.3, was that they cannot. Out of the various clustering options available, we elected to use part-of-speech categories as clusters, since a full morphological analysis was already available as a by-product of deglutenization.

We ran three RNN baselines:

1. the Medium regularized LSTM setup in Zaremba et al. (2014). We used the implementation[10] in Tensorflow (Abadi et al., 2016b)

2. AWD-LSTM (Merity et al., 2018) with the hyperparameter settings tuned to WT2, as the size of its vocabulary matches Szeged's better than PTB's does.

3. `LSTM-512-512`, the smallest configuration described in Jozefowicz et al. (2016). The model was reimplemented in Tensorflow, and is available from the `emLam` repository.

Due to time and resource constraints, the first two baselines were only run on the Szeged corpus, and the last one only on the smallest, GLF-30 (50) variants of the gigaword corpora.

[8] `-kndiscount`

[9] `-kndiscount -gt1min 1 -gt2min 1 -gt3min 1 -gt4min 1 -gt5min 1 -interpolate1 -interpolate2 -interpolate3 -interpolate4 -interpolate5`

[10] https://github.com/tensorflow/models/tree/r1.10.0/tutorials/rnn/ptb

Language models typically perform worse when tested on a different corpus, due to the differences in vocabulary, word distribution, style, etc. To see how significant this effect is, the models trained on the two gigaword corpora were evaluated not only on the test set of their training corpus, but on the other corpus as well.

2.4 Results

2.4.1 n-grams

The results achieved by the n-gram models are reported in Table 2.2–2.5. Table 2.2 lists the perplexities achieved by KN 5-grams of various kinds; the odd one out is POS GLF, where the limited vocabulary enabled us to create up to 9-gram models. For MNSZ2, the reported score is from the 7-gram model, which outperformed 8- and 9-grams. Results of English models on the PTB and 1B are included for comparison.

A glance at the table reveals how the vocabulary problem affects language modeling performance: the perplexities attained by word-level 5-grams are anywhere between 85%–265% higher than the results of the corresponding English models. GLF 5-grams performed even worse than word-level models, confirming our suspicion that subword-level n-grams are suboptimal.

Several interesting observations can be made. First, with the exception of the first Webcorpus word model, models trained on larger corpora attain lower perplexity scores, underlying the importance of training data size. Second, it seems that as the size of the vocabulary decreases (with larger frequency thresholds on one hand, and with the POS models on the other), so does the perplexity – POS n-grams are probably at their limit at 10–12 points.

Finally, an interesting trend emerges when comparing the results of the two gigaword corpora: the perplexities of both word and GLF models are about 50% higher on Webcorpus than on MNSZ2. Finding the cause of this discrepancy requires further research. Two possible candidates are data sparsity (at the same vocabulary size, Webcorpus is 25% smaller) and a difference in the distribution of inflection configurations.

For n-gram models, Szeged_S and Szeged_O are almost equivalent, as they cannot benefit from long-term dependencies. Here we only report results for the former.

Table 2.3 shows the best n-gram perplexities achieved by GLF models. It can be seen that interpolated, unpruned models perform much better than backoff models.

Corpus	Threshold	Word	GLF	Full POS	POS GLF
Szeged_S	3	262.77	637.29	35.20	66.48
Webcorpus	1	N/A	N/A	10.21	*12.89*
	5	328.22	399.49	N/A	N/A
	30	259.79	362.74	N/A	N/A
MNSZ2	1	N/A	N/A	11.88	*12.47*
	10	233.52	277.94	N/A	N/A
	50	174.65	239.57	N/A	N/A
PTB (Mikolov and Zweig, 2012)	N/A	141.2			
1B (Chelba et al., 2014)	3	90			

Table 2.2: 5-gram (*9 for POS GLF*) KN test results (PPL)

Model	Pruned backoff	Unpruned interpolated
Szeged_S GLF	637.29	**587.11**
Webcorpus GLF-5	399.49	**324.24**
Webcorpus GLF-30	362.74	**291.75**
MNSZ2 GLF-10	277.94	**214.57**
MNSZ2 GLF-50	239.57	**186.63**
PTB (Mikolov and Zweig, 2012)	141.2	N/A
1B (Chelba et al., 2014)	90	**67.6**

Table 2.3: The best KN 5-gram results

2.4.2 Class-based n-grams

Our class-based experiments confirmed our working hypothesis regarding clustered models. Contrary to the general consensus, our findings (Table 2.4) show that interpolating class- and token-level (in this case, GLF) LMs do not lead to any improvement over the latter. The class-based model could only improve on the unigram model, and failed to do so for the higher orders.

Why the discrepancy? The most likely explanation is that as the size of the vocabulary grows larger, the emission entropy increases, which is mirrored by the perplexity. This would explain why class-based n-grams seem to work on small corpora, such as the PTB, but not on MNSZ2. Part of the blame also lies with us for using POS-based clusters, which reportedly underperform automatic methods (Niesler et al., 1998). Our results confirm that POS-based clusters simply do not work.

Another point of interest is the diminishing returns of PPL reductions as the n-gram orders grow. While we have not experimented with 6-grams or higher orders, it seems

probable that performance of GLF models would peak at 6- or 7-grams on MNSZ2 (and Webcorpus). For word-level models, this saturation point arrives much earlier: while not reported here, the perplexity difference between 4- and 5-gram models is only 1-2 point. This implies that GLF models are less affected by data sparsity.

Model	GLF-10	POS → GLF-10	GLF-50	POS → GLF-50
1-gram	34,208	6,918	35,654	5,719
2-gram	741.76	2,698	649.57	2,238
3-gram	426.87	2,329	349.93	1,932
4-gram	303.59	2,121	262.69	1,760
5-gram	277.94	1,986	239.57	1,649

Table 2.4: Comparison of GLF and class-based model performance on the MNSZ2. POS → GLF-n denotes a HMM model with POS as class and GLF-n as the surface model.

2.4.3 Cross-evaluation

It is a well-known fact that the performance of LMs degrade substantially when they are not evaluated on the corpus they were trained on. This effect is clearly visible in Table 2.5, which shows how word-level and GLF n-gram models trained on one of the two gigaword corpora perform on the other.

Again, the two corpora display wildly different characteristics. Webcorpus word models exhibit the smallest perplexity increase of 12-15%, followed by the GLF models, whose score grows by three times as much. Models trained on the MNSZ2 are affected the most; interestingly, both the word and GLF models see their perplexity increased by about 120–140%. Contrasting this result with how the models perform on their own training corpus see the *Evaluated on self*) seems to suggest that there exists a trade-off between predictive power and universality.

As POS models capture syntactic regularities, they are affected to a much lesser extent than token-level models (with the exception of the Webcorpus word model). Curiously, the Webcorpus GLF model seems to be more stable than the full POS one; a pattern not replicated in the MNSZ2 case. This, and the fact that there is performance degradation in POS models at all again hints at a different distribution of inflections in the two corpora.

2.4.4 RNN language models

Table 2.6 reports the perplexities achieved by the RNN models. Disregarding the last column for a moment, three clear conclusions can be drawn from the numbers. First, in

Trained on	Threshold	Evaluated on self	Evaluated on other	Increase
Webcorpus word	5	328.22	377.88	15%
	30	259.79	291.98	12%
MNSZ2 word	10	233.52	566.60	142%
	50	174.65	397.13	127%
Webcorpus GLF	5	399.49	606.43	52%
	30	362.74	495.38	37%
MNSZ2 GLF	10	277.94	619.81	123%
	50	239.57	548.76	129%
Webcorpus full POS	1	10.21	16.14	58%
MNSZ2 full POS	1	11.88	16.49	39%
Webcorpus POS GLF	1	12.89	18.08	40%
MNSZ2 POS GLF	1	12.47	18.25	46%

Table 2.5: Performance of word and GLF LMs when evaluated on their own training dataset and on the other gigaword corpus. Perplexities in the *Evaluated on self* column are copied over from Table 2.2.

line with what has been reported for English by many authors, RNNs clearly outperform even the best n-gram models. Second, the performance of GLF models is much closer to the word-level ones than it was for n-grams (about 30% increase on Szeged as opposed to the 140% in Table 2.2), proving that the technique is a much better fit for neural LMs. Finally, models that preserve the overall text sequence order perform better than shuffled datasets, if not by much. However, keeping the text structure intact allows the application of neural cache or pointer sentinel extensions, which further improve results by 7%–9%.

It is important to note that while the GLF format makes neural modeling of large-vocabulary corpora (such as Webcorpus and MNSZ2) possible, the results are not quite comparable to what Jozefowicz et al. (2016) reported for 1B; especially considering that these GLF models used very high frequency thresholds, and the GLF or GLF-10 configurations might have fared slightly worse (as in Table 2.2). The odd one out is Szeged, where the performance of both the word and GLF models are reasonably close to their PTB equivalents. This gives us hope that the "curse of agglutination" can be dealt with in future work.

2.4.5 Pseudo-Hungarian

Attentive readers might have noticed that the perplexity scores reported here are different from those in the original paper. The reason for this is that in Nemeskey (2017), we

Model	Dataset	Perplexity	Per-token PPL
Medium regularized	Szeged_S	101.74	
Medium regularized	Szeged_O	98.88	
AWD-LSTM	Szeged_S	71.70	
AWD-LSTM	Szeged_O	68.30	
AWD-LSTM + pointer	Szeged_O	62.08	
Medium regularized	Szeged_S GLF	131.39	38.07
Medium regularized	Szeged_O GLF	126.78	37.07
AWD-LSTM	Szeged_S GLF	94.19	29.70
AWD-LSTM	Szeged_O GLF	89.17	28.51
AWD-LSTM + pointer	Szeged_O GLF	81.13	26.57
LSTM-512-512	Webcorpus GLF-30	191.51	40.46
LSTM-512-512	MNSZ2 GLF-50	146.82	38.78
Medium regularized	PTB	82.07	
AWD-LSTM	PTB	57.3	
AWD-LSTM + pointer	PTB	52.8	
LSTM-512-512	1B	54.1	

Table 2.6: LSTM model performance

published per-token perplexity, which gave incorrect results for the GLF datasets, where a word may consist of multiple tokens. In this chapter, all perplexity values are normalized for the word count. While this results in 34%–42% higher perplexity scores (depending on the corpus), all findings of the original paper still stand; in particular, GLF models still achieve lower perplexity than word-level ones.

This mistake, however, was not entirely without merit. It allowed us to inadvertently perform a language modeling experiment on a fully analytic version of Hungarian. In English, many of the inflectional tokens in the GLF format are actually separate words: most case markings are expressed with prepositions, such as `<DAT>` with '*for*'; and for most adjectives, the `<COMPAR>`ative and `<SUPERLAT>`ive forms are marked with the words '*more*' and '*most*', respectively. Other GLF tokens, such as the plural marker `<PLUR>` and those related to the verb conjugation are either part of the word in English, or are missing altogether; yet we find examples for them in other languages (Japanese 達 '*tachi*' for the plural and e.g. なさい '*nasai*' for imperative). This goes to show how arbitrary the realization of grammatical functions is across languages – and that our "analytic Hungarian" could well be a real language. For this reason, we also record the per-token perplexity values in Table 2.6 as a curiosity.

2.4.6 Into the Unknown

As we have seen previously, the GLF format has various advantages: it reduces the size of the vocabulary and the number of OOVs, which benefits neural language modeling. However, GLF models have a higher perplexity than the corresponding word models.

In this section, we argue that the practice of replacing low-frequency words with `<unk>` tokens and using "vanilla" perplexity as the metric does not accurately represent the uncertainty present in the model. While the model learns when to predict a "generic infrequent word", it cannot make an informed guess as to its identity. Yet frequency thresholding is merely a technical necessity rather than proper modeling of the language. This is especially evident in text generation, where the presence of `<unk>` tokens in the output is undesirable. In order to generate valid text, we need to replace them with actual words.

In order to evaluate the performance of a model that is not allowed to generate `<unk>`s, we propose a new metric: *rectified perplexity*. It augments the "vanilla" perplexity score with the perplexity of the model that predicts valid words for each `<unk>` token:

$$\mathrm{PPL}_{corr} = \exp\left(\frac{|C|\log(\mathrm{PPL}_{raw}) + |\mathrm{OOVs}|\log(\mathrm{PPL}_{unk})}{|C|}\right) \tag{2.1}$$

where PPL_{raw} is the uncorrected perplexity, PPL_{unk} is the perplexity of the sublanguage covered by the `<unk>` token, and $|OOVs|$ and $|C|$ are the number of `<unk>` tokens and the total number of tokens in the corpus, respectively.

This rectified metric is equivalent with the perplexity of a class-based model where words in the vocabulary are singleton classes and `<unk>` emits out-of-vocabulary words. There are several options to estimate the emission probabilities; one example would be a character-level model, such as the character LSTM in Jozefowicz et al. (2016). Since we operate in a closed vocabulary setting, we use the MLE of the token distribution under the frequency threshold.

Table 2.7 shows the performance of the main models under the new metric. It is apparent at first sight that GLF models are much less affected by the perplexity correction coming from OOVs; a natural consequence of them having about half the number of OOVs compared to word models. While the Szeged and Webcorpus word n-gram models still perform better than their GLF counterparts, the tables are already turned on MNSZ2. In the RNN case, the GLF version attain about 33% lower perplexity than the word model, showing its superiority in fully specified language modeling.

Model	Word	GLF
Szeged_S 5-gram	262.77 → **655.92** (149%)	637.29 → 848.50 (33%)
Webcorpus 5-gram (5)	328.22 → **389.11** (19%)	399.49 → 426.38 (7%)
Webcorpus 5-gram (30)	259.79 → **398.73** (53%)	362.74 → 416.35 (15%)
MNSZ2 5-gram (10)	233.52 → 362.24 (55%)	277.94 → **337.33** (21%)
MNSZ2 5-gram (30)	174.65 → 401.20 (130%)	239.57 → **336.52** (40%)
Szeged_O medium regularized	98.88 → 258.66 (162%)	126.78 → **169.65** (34%)
Szeged_O AWD-LSTM	68.30 → 178.67 (162%)	89.17 → **119.32** (34%)
Szeged_O AWD-LSTM + pointer	62.08 → 162.40 (162%)	81.13 → **108.56** (34%)

Table 2.7: Perplexity correction for OOV tokens

2.5 Conclusion

The work presented in this chapter contributes to Hungarian language modeling in two ways. First, we reported state-of-the-art LM baselines for three Hungarian corpora, from million to gigaword size. We found that word-level LMs performed worse than they do for English, not in the least because of the increased vocabulary size and number of OOV tokens. The vocabulary problem could be alleviated with splitting words into lemmas and inflectional affixes (the "gluten-free" format). GLF models had a higher "vanilla" perplexity than word models, but outperformed them when no `<unk>` tokens were allowed in the output.

Second, we introduced a benchmark corpus for language modeling. To our knowledge, this is the first such dataset for Hungarian. This specially prepared version of the Hungarian Webcorpus is freely available, allowing researchers to easily and reproducibly experiment with new language modeling techniques. It is comparable in size to the One Billion Word Benchmark corpus of English, making comparisons between the two languages easier.

2.5.1 Future work

While the methods reported here can be called state-of-the-art, many similarly effective modeling approaches are missing. Evaluating them could provide additional insight into how Hungarian "works" or how Hungarian and English should be modeled differently. Understanding the unusual behavior of word models on Webcorpus also calls for further inquiry into language and corpus structure.

The performance of the models here was measured in isolation. Putting them into use (maybe with some adaptation) in NLP applications such as ASR or MT could answer the

question of whether the reduction in perplexity translates to similar reductions in WER or BLEU.

The most glaring problem touched upon, but not addressed, in this paper, is the effect of compounding and derivation on vocabulary size. A way to reduce the number of words could be a more thorough deglutenization algorithm, which would split compound words into their parts and strip productive derivational suffixes, while leaving *frozen* ones such as *ház·as·ság* untouched. This could indeed be a case when a gluten free diet does make one slimmer.

Chapter 3

Evaluating multi-sense embeddings for semantic resolution

As promised in Section 1.7.3, in this chapter we return to the problem of evaluating multi-sense embeddings. We propose a method that evaluates MSEs intrinsically, based on a comparison with monolingual dictionaries. The content of this chapter is a result of joint work presented in Borbély; Makrai, et al. (2016). The parts included here are all contributions of the author, with the exception of Section 3.4, which gives a short summary of the rest of the paper for context.

3.1 Introduction

Gladkova and Drozd (2016) calls polysemy "the elephant in the room" as far as evaluating embeddings are concerned. Here we attack this problem head on, by proposing a method for evaluating multi-sense word embeddings (MSEs), where allegedly polysemous or homonymous words have multiple vectors, ideally one per sense.

Work on the evaluation of MSEs (for lexical relatedness) goes back to the seminal Reisinger and Mooney (2010), who note that usage splits words more finely (with synonyms and near-synonyms ending up in distant clusters) than semantics. The differentiation of word senses is fraught with difficulties, especially when we wish to distinguish homonymy, using the same written or spoken form to express different concepts, such as Russian *mir* 'world' and *mir* 'peace' from polysemy, where speakers feel that the two senses are very strongly connected, such as in Hungarian *nap* 'day' and *nap* 'sun'. To quote Zgusta (1971): "Of course it is a pity that we have to rely on the subjective interpretations of the speakers, but we have hardly anything else on hand". Etymology makes clear that different languages make different lump/split decisions in the conceptual space, so much so that

translational relatedness can, to a remarkable extent, be used to recover the universal clustering (Youn et al., 2016).

Another confounding factor is part-of-speech. Very often, the entire distinction is lodged in the POS, as in *divorce* (Noun) and *divorce* (Verb). In this case, both words relate to the same *concept*, so a fully semantic MSE would map them to the same vector. However, at other times the connection is less clear: compare the verbal *to bank* 'rely on a financial institution' and *to bank* 'tilt'. Clearly the former is strongly related to the nominal *bank* 'financial institution' while the semantic relation 'sloping sideways' that connects the tilting of the airplane to the side of the river is somewhat less direct, and not always perceived by the speakers. This ambiguity makes the evaluation of such word pairs in MSEs all the more difficult.

In this chapter, we propose an MSE evaluation method based on sense distinctions made in traditional monolingual dictionaries. We investigate the correlation between the number of senses of each word-form in the embedding and in the manually created inventory as a proxy measure of how well embedding vectors correspond to concepts in speakers' (or at least, the lexicographers') mind. The details and results are discussed in Section 3.2. Section 3.3 investigates what linguistic factors other than polysemy might affect the training of MSEs and to what extent. Section 3.4 briefly summarizes the other evaluation method proposed in Borbély; Makrai, et al. (2016). Interested readers are referred to the original paper for details. Finally, some very preliminary conclusions are offered in Section 3.5, more in regards to the feasibility of the evaluation method we propose than about the merits of the systems we evaluated.

3.2 Comparing lexical headwords to multiple sense vectors

We present a preliminary evaluation of four MSE algorithms on two languages, English and Hungarian.

3.2.1 Resources to be evaluated

The four implementations include the released result of the spherical context clustering method `huang` (Huang et al., 2012) (English only); the learning process of Neelakantan et al. (2014) with adaptive sense numbers (we report results using their release MSEs and their tool itself, calling both `neela`); the parametrized Bayesian learner of Bartunov et al. (2016) where the number of senses is controlled by a parameter α for semantic resolution,

here referred to as AdaGram; and `jiweil` (Li and Jurafsky, 2015).

MSEs with multiple instances are suffixed with their most important parameters, i.e. the learning rate for AdaGram ($a = 0.5$); the number of multi-prototype words and whether the model is adaptive (NP) for release `neela`; and the number of induced word senses ($s = 4$) for our non-adaptive `neela` runs.

Where applicable, MSEs were trained on UMBC Webbase (Han et al., 2013) for English and Webkorpusz (Halácsy et al., 2004) for Hungarian.

3.2.2 Lexical resources

Two dictionaries per language serve as ground truth. For English, we use the *Collins-COBUILD* (CED, Sinclair (1987)) dictionary and the *Longman Dictionary of contemporary English* (LDOCE, Boguraev and Briscoe (1989)). For Hungarian, we had access to the *Explanatory Dictionary of Hungarian*[1] (EKSZ, Pusztai (2003)) and a subsample (the letter '*b*') of the *Comprehensive Dictionary of Hungarian*[2] (NSZ, Ittzés (2011)). For the Hungarian dictionaries, we relied on the versions created in Mihaltz (2010) and Recski et al. (2016).

Our lexicographic sources take different positions as to whether semantic concepts or POS categories should be the main organizational factor. CED starts with the semantic distinctions and subordinates POS distinctions to these, while LDOCE starts with a POS-level split and puts the semantic split below. Of the Hungarian dictionaries, NSZ is closer to CED, while EKSZ is closer to LDOCE in this regard. Since we expect MSEs to exhibit semantic properties, we consider CED as our primary ground truth.

In addition to the machine-readable versions of paper-based dictionaries, we include a lexical database that was purportedly designed for program control from the get-go: WordNet (Miller, 1995). In terms of size, the English database is on par with the more traditional dictionaries; in fact, it is the largest resource for English. The Hungarian edition (Miháltz et al., 2008) is less developed, and also contains spurious entries that we had to filter before the experiments. In WordNet, synsets are differentiated by both semantic and syntactic roles, which introduces a high variance in sense numbers: the English WordNet contains a word with as many as 75 different meanings. The quality of the entries is also questionable. For the word *guard*, five out of the ten noun synsets are about the same position in various team sports. Additionally, the entries include very fine differentiation of concepts, such as "*a position on a basketball team*" and "*the person who plays the position of guard on a basketball team*"; a distinction entirely missing for other

[1] *Magyar értelmező kéziszótár*
[2] *A magyar nyelv nagyszótára III-IV*

professions, such as *CEO*, *colonel* or *engineer*. For these reasons, we do not recommend WordNet as ground truth.

We simulate the case of languages without a machine-readable monolingual dictionary with `OSub`, a dictionary extracted from the OpenSubtitles parallel corpus (Tiedemann, 2012) automatically: the number of the senses of a word in a source language is the number of words it translates to, averaged among many languages. More precisely, we use the unigram perplexity of the translations instead of their count to reduce the considerable noise present in automatically created dictionaries.

3.2.3 Evaluation

Table 3.1 summarizes the distribution of word senses (how many words with 1,…,6+ senses) and the major statistics (size, mean, and variance) both for our lexicographic sources and for the automatically generated MSEs.

Resource	Size	1	2	3	4	5	6+	Mean	Std
CED	82,024	80,003	1,695	242	69	13	2	1.030	0.206
LDOCE	30,265	26,585	3,289	323	56	11	1	1.137	0.394
`OSub`	75,718	58,043	14,849	2,259	431	111	25	1.354	0.492
WordNet	149,400	122,993	15,571	5,048	2,178	1,166	2,444	1.387	1.447
AdaGram	476,827	122,594	330,218	11,341	5,048	7,626	0	1.836	0.663
`huang`	100,232	94,070	0	0	0	0	6,162	1.553	2.161
`neela`.30k	99,156	69,156	0	30,000	0	0	0	1.605	0.919
`neela`.NP.6k	99,156	94,165	2,967	1,012	383	202	427	1.101	0.601
`neela`.NP.30k	99,156	71,833	20,175	4,844	1,031	439	834	1.411	0.924
`neela`.s4	578,405	574,405	0	0	4,000	0	0	1.021	0.249
EKSZ	121,578	66,849	628	57	11	1	0	1.012	0.119
NSZ (b)	5,594	5,225	122	13	3	0	0	1.029	0.191
`OSub`	169,244	159,843	9,169	229	3	0	0	1.144	0.199
WordNet	47,767	41,248	4,493	1,153	440	204	229	1.220	0.739
AdaGram	238,462	135,052	76,096	15,353	5,448	6,513	0	1.626	0.910
`jiweil`	285,856	57,109	92,263	75,710	39,624	15,153	5,997	2.483	1.181
`neela`.s2	771,870	767,870	4,000	0	0	0	0	1.005	0.072
`neela`.s4	771,870	767,870	0	0	4,000	0	0	1.016	0.215

Table 3.1: Size (in words), sense distribution, mean, and standard deviation of the number of senses in English and Hungarian lexicographic and automatically generated resources

While the lexicographic sources all show roughly exponential decay of the number of senses, only some of the automatically generated MSEs replicate this pattern, and only at well-chosen hyperparameter settings. `huang` has a hard switch between single-sense (94%

of the words) and 10 senses (for the remaining 6%), and the same behavior is shown by the released Neela.300D.30k (70% one sense, 30% three senses). The English AdaGram and the Hungarian `jiweil` have the mode shifted to two senses, which makes no sense in light of the dictionary data. Altogether, we are left with only two English candidates, the adaptive (NP) `neela`s; and one Hungarian, AdaGram, that replicate the basic exponential decay.

The figure of merit we propose is the correlation between the number of senses obtained by the automatic method and by the manual (lexicographic) method. We experimented both with Spearman ρ and Pearson r values, the entropy-based measures Jensen-Shannon and KL divergence, and cosine similarity and Cohen's κ. The entropy-based measures failed to meaningfully distinguish between the various resource pairs. The cosine similarities and κ values would also have to be taken with a grain of salt: the former does not take the exact number of senses into account, while the latter penalizes all disagreements the same, regardless of how far the guesses are. On the other hand, the Spearman and Pearson values are so highly correlated that Table 3.2 shows only ρ of sense numbers attributed to each word by different resources, comparing lexicographic resources to one another (top panel); automated to lexicographic (mid panel); and different forms of automated English (bottom panel). The top two values in each column are highlighted in the last two panels, n is the number of headwords shared between the two resources.

The dictionaries themselves are quite well correlated with each other. The Hungarian values are considerably larger both because we only used a subsample of NSZ (the letter *b*) so there are only 5,363 words to compare, and because NSZ and EKSZ come from the same Hungarian lexicographic tradition, while CED and LDOCE never shared personnel or editorial outlook. The Hungarian WordNet does not correlate well with the traditional lexical resources, and the English edition more with LDOCE, casting further doubts on the validity of WordNet as ground truth data.

Two English MSEs `neela` and `huang`, show perceptible correlation with a lexical resource, LDOCE, and only two systems, AdaGram and `neela`, correlate well with each other (ignoring different parametrizations of the same system, which of course are often well correlated to one another).

3.3 Parts of speech and word frequency

Since no gold dataset exists, against which the results could be evaluated and the errors analyzed, we had to consider if there exist factors that might have affected the results. In particular, the better correlation of the adaptive methods with LDOCE than with CED

Resources compared	n	ρ
LDOCE vs CED	23,702	0.266
CED vs WordNet	41,076	0.195
LDOCE vs WordNet	26,376	0.477
EKSZ vs NSZ (b)	3,484	0.648
EKSZ vs WordNet	16,786	0.103
NSZ (b) vs WordNet	966	0.052
`neela`.30k vs CED	23,508	0.089
`neela`.NP.6k vs CED	23,508	0.084
`neela`.NP.30k vs CED	23,508	0.112
`neela`.30k vs LDOCE	21,715	0.226
`neela`.NP.6k vs LDOCE	21,715	**0.292**
`neela`.NP.30k vs LDOCE	21,715	0.278
`huang` vs CED	23,706	0.078
`huang` vs LDOCE	21,763	**0.280**
`neela`.s4 vs EKSZ	45,401	0.067
`jiweil` vs EKSZ	32,007	0.023
AdaGram vs EKSZ	26,739	0.086
AdaGram.a05 vs EKSZ	26,739	0.088
`neela`.30k vs `huang`	99,156	0.349
`neela`.NP.6k vs `huang`	99,156	**0.901**
`neela`.NP.30k vs `huang`	99,156	0.413
`neela`.s4 vs `jiweil`	283,083	0.123
AdaGram vs `neela`.s4	199,370	**0.389**
AdaGram vs `jiweil`	201,291	0.140

Table 3.2: Word sense distribution similarity between various resources

Resources compared	n	ρ
CED vs POS	42,532	0.052
LDOCE vs POS	28,549	**0.206**
WordNet vs POS	46,802	0.052
`OSub` vs POS	48,587	**0.141**
EKSZ vs POS	52,158	0.080
NSZ vs POS	3,532	0.046
`huang` vs POS	98,405	0.026
CED vs freq	36,709	0.124
LDOCE vs freq	27,859	0.317
WordNet vs freq	52,042	0.432
AdaGram vs freq	399,985	0.343
`huang` vs freq	94,770	0.376
`neela`.s4 vs freq	94,044	**0.649**
`neela`.NP.30k vs freq	94,044	0.368
`neela`.NP.6k vs freq	94,044	**0.635**
UMBC POS vs freq	136,040	-0.054

Table 3.3: Word sense distribution similarity with POS tag perplexity (top panel) and word frequency (bottom panel)

raises suspicions. The former groups entries by part of speech, the latter by meaning, implying that the methods in question might be counting POS tags instead of meanings.

Another possible bias that might have influenced the results is word frequency (Manin, 2008). This is quite apparent in the release version of the non-adaptive methods `huang` and `neela`: the former expressly states in the README that the 6,162 words with multiple meanings "roughly correspond to the most frequent words".

To examine the effect of these factors, we measured their correlation with the number of meanings reported by the methods above. For each word, the frequency and the POS perplexity was taken from the same corpora we ran the MSEs on: UMBC for English and Webkorpusz for Hungarian. Table 3.3 shows the results for both English and Hungarian. The correlation of automatically generated resources with POS tags is negligible: all other

embeddings correlate even weaker than `huang`, the only one shown. From the English dictionaries, LDOCE produces the highest correlation, followed by `OSub`; the correlation with CED, as expected, is very low. The Hungarian dictionaries are around the level of CED.

In comparison, the correlation between sense numbers and word frequency is much more evident. Almost all English resources correlate with the word frequency by at least 0.3 (the notable exception being CED which is the closest to a gold standard we have); furthermore, the highest correlation we measured are between two versions of `neela` and the word frequency. Adding to this the low correlation of the gold CED against the other resources (see Table 3.2), it appears the multi-prototype embeddings included in the study were trained to assign more vectors to frequent words instead of trying this for truly polysemous ones.

To disentangle these factors further, we performed partial correlation analysis with the effect of frequency (or its log) or POS perplexity removed. Recall that LDOCE and CED originally correlated only to $\rho = 0.266$. After removing POS, we obtain 0.545, removing frequency yields 0.546, and removing log frequency brings this up to 0.599. On select embeddings such as `neela`.NP.6k correlations with CED improve from a negligible 0.093 to a respectable 0.397 if POS, and an impressive 0.696 if log frequency is factored out.

3.4 Cross-linguistic treatment of concepts

Since monolingual dictionaries are an expensive resource, Borbély; Makrai, et al. (2016) also proposes an automatic evaluation of MSEs based on the discovery of Mikolov; Le, et al. (2013) that embeddings of different languages are so similar that a linear transformation can map vectors of the source language words to the vectors of their translations. The linear mapping is trained on a seed of 5,000 thousand word pairs, and evaluated on 1,000.

The proposed quality measure is the ratio of correctly translated words between the multi-sense and the single-sense cases. Of the two MSEs that could be trained for Hungarian, Adagram clearly improved the translation with a ratio around 2; while `jiweil` with a ratio of 0.25, had a highly detrimental effect.

3.5 Conclusions

To summarize, we have proposed a monolingual method that evaluates word embeddings in terms of their semantic resolution (ability to distinguish multiple senses). Our monolingual

task, match with the sense-distribution of a dictionary, yields an intrinsic measure in the sense of Chiu et al. (2016).

The original paper proposes an extrinsic bilingual evaluation metric based on word translation. For now, the two measures are not particularly well correlated, though the low/negative result of `jiweil` in Table 3.1 could be taken as advance warning for its low performance in MT. The reason, we feel, is that both kinds of performance are very far from expected levels, so little correlation can be expected between them: only if the MSE distribution of senses replicates the exponential decay seen in dictionaries (both professional lexicographic and crowdsourced products) is there any hope for further progress.

The central linguistic/semantic/psychological property we wish to capture is that of a *concept*, the underlying word sense unit. To the extent standard lexicographic practice offers a reasonably robust notion (this is of course debatable, but we consider a straight correlation of 0.27 and and a frequency-effect-removed correlation of 0.60 over a large vocabulary a strong indication of consistency), this is something that MSEs should aim at capturing. We leave the matter of aligning word senses in different dictionaries for future work, but we expect that it can improve the inter-dictionary (inter-annotator) agreement considerably, to provide a more robust gold standard.

Since no manual steps are involved, other researchers can accurately reproduce these kinds of evaluations. Some glue code for this project can be found at `https://github.com/hlt-bme-hu/multiwsi`.

Chapter 4

Habeas Corpus

This chapter describes the new Hungarian Webcorpus 2.0, built from the Hungarian subset of the Common Crawl. On the one hand, our work is the spiritual successor of the Hungarian Webcorpus (see Section 2.2): a freely downloadable, cleaned crawl of the Hungarian web. On the other, it is a continuation of the effort described in Indig (2018). The author of the paper generously provided us with their corpus, which we used to bootstrap ours. The software published along with the paper[1] also served as a basis for our code. The main contributions of our work are

- an incremental, continuous expansion of the corpus;
- greatly improving the corpus quality by language identification, various deduplication steps, lemmatization and POS tagging;
- including a snapshot of the Hungarian Wikipedia;
- a full software stack used to perform the tasks mentioned above;
- preliminary work on pretraining a Hungarian BERT model on the corpus.

The rest of the chapter is structured as follows. Section 4.1 discusses the goals and design considerations behind the new corpus. In Section 4.2, we survey already existing Hungarian corpora and introduce the Common Crawl briefly. Our work is presented in the next four sections. The software architecture is outlined in Section 4.3; the corpus building procedure and its individual subtasks are elucidated in Section 4.4. Section 4.5 describes how the Hungarian Wikipedia was incorporated into the corpus. Section 4.6 presents initial work the Hungarian BERT trained on Webcorpus 2.0. Finally, we draw our conclusions and outline our plans for future improvements in Section 4.7.

[1] https://github.com/ppke-nlpg/commoncrawl-downloader

4.1 Goals and design considerations

We set out with the aim of creating the largest Hungarian corpus to date that can be used to train modern Transformer language models and contextual embeddings. Here we review the requirements of such a corpus and the design choices implied by them.

4.1.1 Goals and constraints

To expand on the motivation above, the corpus should

- be in the gigaword range, at least as large as the English Wikipedia (2.5B tokens);
- consist of high quality pages based on some criteria that can be enforced at said magnitude;
- be expandable with new data and it should be easy to regenerate from raw data if the processing pipeline changes;
- be, most importantly, freely available for NLP practitioners.

4.1.2 Design considerations

Representativeness

Traditionally, corpus compilation is preceded by careful consideration of various factors in order to ensure *representativeness* of the final product (Biber, 1993). These include the amount and type of texts to include, a (possible hierarchical) categorization of the text into genres and subcorpora, and well-defined population boundaries, such *British English texts from 1961*[2] for the Lancaster–Oslo/Bergen (LOB) corpus (Johansson et al., 1978). With well defined population(s), it possible to select sampling rates to ensure that the resulting corpus represents the targeted language well.

While not a trivial undertaking, such a rigorous text selection process is feasible on the scale of the LOB or the Brown corpus (Francis and Kucera, 1979) (cca. 1M words). As corpora got bigger and the web became the main source of text, representativity proved unattainable and the focus moved on to the much more laxly defined *balancedness* (Váradi, 2002). Finally, if we fast-forward to the last 1–2 years, we can see state-of-the-art Transformer LMs trained on web crawl data blithely ignoring any attempt at representativity or balancedness.

[2] https://www1.essex.ac.uk/linguistics/external/clmt/w3c/corpus_ling/content/corpora/list/private/LOB/lob.html

In this work we follow the latter practice, and leave the compilation of a balanced subcorpus for future work.

Text organization

Modern, contextual language models are able to learn long-distance dependencies in text. However, many of these dependencies, such as topics and coreferences, are inter-sentential in nature. As a result, training contextual language models necessitates that the document structure remains intact.

The main goal of our corpus is to serve as training data for statistical NLP models. This significantly decreases the importance of document metadata, which would be essential for corpus linguistic research. In keeping with the main goal, we include only the basic properties: URL, domain, date of acquisition, etc.

Quality

Even if we do not aim for representativeness, document quality remains an important factor. Transformer LMs try to ensure quality by using data sources with at least some editorial control: BERT was trained on Wikipedia and e-books (Devlin et al., 2018), and RoBERTa's sources include newswire texts (Liu et al., 2019). Since such corpora are generally hard to come by, especially on web scale, they are usually augmented with data acquired from noisier sources, which are then cleaned with heuristic methods. Examples include WebText, used to train GPT-2, which includes web pages linked from Reddit (Radford et al., 2019); or Stories, pages filtered from Common Crawl that resemble the structure of Winograd schemas (Trinh and Le, 2018).

For Hungarian, we face the additional issue of not having access to semi-edited corpora on the same scale. We are including the Hungarian Wikipedia in our corpus, but it is only about 7% of the English edition, and the situation is similar with other resources, such as Project Gutenberg[3]. Indirect quality indicators, like Reddit karma, are also unavailable. In light of this, we decided to use a threshold on document length as the quality metric. While far from perfect, this heuristic allows us to filter microblogs, news headers, and automatically generated content, such as soft error messages (i.e. those unmarked by HTML error codes).

[3] https://www.gutenberg.org/

Uniqueness

For the sake of simplicity, and so as to be able to use the URL as a document ID, we decided to keep a single copy of each document. In Common Crawl, however, a URL can be included in any number of the monthly crawls, which raises the question: don't we lose valuable data by ignoring later snapshots of the same page? It is hard to say for sure without a thorough study, but we argue that the pages whose content changes frequently most likely contain data we are less interested in. An example would be the front pages of online news outlets, which consist of the headers of the latest news items. Such fragmented content would not serve the goal of LM training very well, as opposed to the full text of the individual news items themselves (which are hopefully in the crawl as well).

Future-proofing

Once compiled, regular corpora never change. Our ambition is to keep our corpus up-to-date, which requires some future-proofing.

First, we do not believe that our current processing pipeline is perfect, and it is possible that some of its parts (e.g. boilerplate removal, deduplication) might be replaced with better alternatives in the future. In order to allow the quick (as quick as size permits) regeneration of the corpus in such cases, Indig (2018) has already argued for the storing of raw HTML data. Since our pipeline is much longer, we opted for storing the results of the intermediary operations as well.

Second, we plan to incrementally extend our corpus with each monthly snapshot of the Common Crawl. In order to do that, not just the corpus, but temporary working files (URL cache, the LSH database, etc.; see Section 4.4) have to be retained as well.

Licensing

We intend the corpus to be freely available, in the same way its predecessor, Webcorpus, or the upstream Common Crawl are. Since the crawl data comes from Common Crawl, we publish them under the same terms of use. Our additions, mainly the morphological analysis and lemmatization performed on the documents, as well as the Wikipedia part of the corpus is licensed under the same Creative Commons Attribution-ShareAlike 4.0 Internation (CC BY-SA 4.0)[4] license that Wikipedia itself uses.

Unfortunately, the introduction of the General Data Protection Regulation (GDPR) (Council of European Union, 2016) creates uncertainties as to the legality of web crawls. No clear guidelines have yet been published that address this very issue, which gives rise to

[4] https://creativecommons.org/licenses/by-sa/4.0/

various interpretations. In the worst case, crawlers could be required to request permission from the copyright owner of each source – an impossible requirement at web scale (it is an interesting question whether a permissive `robots.txt` constitutes consent). In any case, we publish the corpus in good faith and so implement a take down policy that allows owners of copyrighted material to request removal of their data.

The corpus can be downloaded from https://hlt.bme.hu/en/resources/webcorpus2.

4.2 Related work

Hungarian is in a fortunate situation, as there are several good quality corpora available for it. In this section, we first review these earlier efforts, then introduce Common Crawl and the studies that used it as a training corpus for language models.

4.2.1 Preexisting corpora

We have already introduced Szeged Treebank, Webcorpus 1.0 and the Hungarian Gigaword Corpus (MNSZ2) in Section 2.2. The first two are well below the one billion word mark. MNSZ2 is a proper gigaword corpus, but at 1.5 billion tokens, it is still smaller than the English Wikipedia used to train BERT (see Section 1.7.4).

The largest Hungarian corpus to date is the huTenTen corpus (Jakubíček et al., 2013; Suchomel; Pomikálek, et al., 2012). It is part of the TenTen family of corpora[5] by Sketch Engine and comprises of "almost 2.5 billion words" crawled from the web in 2012[6]. While the corpus is of the right size and it is the type of data we are after, it is unfortunately behind a paywall.

There are two Hungarian corpora based on Common Crawl. OSCAR (Ortiz Suárez et al., 2019), created by Inria[7], is a multilingual corpus compiled from the November 2018 crawl data. It has 166 language-specific subcorpora of various sizes. The deduplicated Hungarian subcorpus is 2.3 billion words in size. Unfortunately, OSCAR is shuffled at the line level, so language models trained on it will not be able to take advantage of long-range dependencies in real documents.

Preliminary work on a Hungarian Common Crawl corpus was started by Indig (2018). The work aimed at downloading the whole Hungarian "vertical", i.e. Hungarian pages from all monthly snapshots. Due to technical difficulties, deduplication was not performed, and the end product has not been published. However, the files contain the documents in their

[5] https://www.sketchengine.eu/documentation/tenten-corpora/
[6] https://www.sketchengine.eu/hutenten-hungarian-corpus/
[7] https://www.inria.fr/en

entirety and the code that came with the paper is free and open-source; hence, we used this work as the basis for our own. The author of the paper generously provided us with their corpus, which we used to bootstrap ours.

4.2.2 Common Crawl

The Common Crawl (CC)[8] is a web crawl archive, continually created and maintained by the Common Crawl Foundation with the aim of "*democratizing access to web information by producing and maintaining an open repository of web crawl data that is universally accessible and analyzable*"[9]. The data is available on Amazon S3[10] in (roughly) monthly digests since November 2013, in the following formats:

- raw HTML in Web ARChive (WARC) format
- metadata in WAT files
- plaintext extracted from HTML in WET files.

It is not trivial to ascertain the size of the Common Crawl corpus. The homepage does not contain exact numbers. Based on the listing in the Wikipedia page[11], in August 2018 the corpus consisted of about 129 billion pages for a total of 8.8 PiB of uncompressed content. Still, it is not clear whether these numbers include only the WARC archives, or the WAT and WET files as well.

Aside from the main archive files, monthly snapshots also include a URL index. This contains the WARC file and offset of each URL in the snapshot, allowing a URL-based prefiltering of the data prior to downloading. In our case, we only keep URLs in the `.hu` domain.

It goes without saying that the sets of Hungarian pages and pages from Hungary do not overlap completely. On the one hand, foreign-language pages, such as English versions of organizational webpages are served under `.hu`. On the other, with about 25% of native speakers living outside Hungary (Balázs, 2013), a significant number of Hungarian electronic text sources are found under foreign top-level domains. The first issue can be dealt with language detection; the second we leave for future work.

[8] http://commoncrawl.org/
[9] https://commoncrawl.org/about/
[10] https://aws.amazon.com/s3/
[11] https://en.wikipedia.org/wiki/Common_Crawl#History_of_Common_Crawl_data

4.2.3 As a training corpus

There are differing accounts as to the effectiveness of Common Crawl as a training corpus. It has been used successfully to train various flavors of language models, include n-grams (Buck et al., 2014), static (Pennington et al., 2014) and contextual embeddings (Baevski et al., 2019), both in mono and in multi-language setups (Grave et al., 2018). In particular, both Baevski et al. (2019) and Buck et al. (2014) report better results for their systems when trained on CC as opposed other corpora news task in WMT 2018 (Bojar et al., 2018),

Yet there are also concerns about the quality of text in CC. Trinh and Le (2018) found many documents whose "content are mostly unintelligible". They arrived at a good training corpus by keeping only the documents that were similar in structure to their target task, the Winograd Schema Challenge (Levesque et al., 2011).

Our main takeaway from this section is that Common Crawl is a very useful resource that can be used to train state-of-the-art systems provided there is some form of quality assurance in place.

4.3 Architecture

Due to the sheer size of the Common Crawl, processing it for any purpose requires a lot of compute and storage. Derived corpora are created either by utilizing massive map-reduce clusters (Panchenko et al., 2018) or from a subset of CC, such as a single monthly snapshot (Grave et al., 2018; Ortiz Suárez et al., 2019).

As our targeted data (Hungarian pages across all monthly releases) is comparable in size with a monthly crawl, it is still possible to process it using a small cluster of machines – with a few caveats.

4.3.1 Computing environment

Our cluster consists of four desktop-grade machines with 4 hyperthreading (HT) cores at 4GHz and 32GB memory, and a single server machine with an 8 HT core Xeon processor that clocks at 2.4GHz. The latter compensates its slower CPU with 217GB RAM.

All machines are connected to the LAN over a 10Gb connection. Data is kept on a 96TB network file system; the local disks are only used for storing code and execution environments.

The machines run Linux. We implemented all steps of our pipeline in Python 3.6.

Miniconda[12] was used to ensure an identical Python environment on all hosts.

Note that our cluster setup has a few peculiarities, which we take advantage of. The amount of memory on our main machine might not be readily available in homogeneous clusters. Having a large NFS share is by no means unheard of, but is also not a given. Since some of our scripts (to be introduced in Section 4.4) depend on these resources, they might have to be reworked to some extent if the code is to be used in a different cluster setup.

4.3.2 The pipeline

Instead of putting much engineering effort into building a complex processing pipeline (that works on a cluster of machines in a distributed fashion), we decided to employ simple Python scripts for each step. Following the Unix philosophy "*do one thing and do it well*"[13], each script performs a single task on a single machine. The scripts take their inputs from the shared file system, process them in parallel and write the results back to a separate directory. This allows us to snapshot the compilation process, thereby ensuring reproducibility of successive steps (see Section 4.1.2).

Parallelism is of utmost importance when processing data at web scale. While we do not employ a map-reduce framework, the basic ideas translate well to our setup. Our scripts can also be classified into '*map*' and '*reduce*' types. The former transform the data document-by-document; the latter perform aggregations. Examples for 'map' tasks include boilerplate removal and filtering, while document deduplication is a 'reduce' operation. Most reduce tasks are run on the main machine of our cluster where they can take advantage of the additional memory.

Map tasks are run in parallel on all machines in the cluster. The data is distributed into per-host directories so that the input/output directory-based logic of the scripts can remain unchanged. We used Ansible[14] to distribute the tasks, which are executed in a tmux[15] multiplexer. Logs are saved to the shared file system. Collecting the per-host outputs is as simple as copying them to an "aggregate" output directory. However, this is only necessary if the subsequent task is a reduce operation; a next map task can simply take the output of the last one as its input.

[12] https://docs.conda.io/en/latest/miniconda.html
[13] https://en.wikipedia.org/wiki/Unix_philosophy#Do_One_Thing_and_Do_It_Well
[14] https://www.ansible.com/
[15] https://github.com/tmux/tmux/wiki

4.4 Running the pipeline

This section elucidates the various steps in our corpus compilation pipeline. The effect of each step on the corpus size is listed in Table 4.2 for the bootstrap corpus and Table 4.1 for our additions up to April 2019 at the end of this section. The former starts at the "Boilerplate removed" step, because that is the dataset we received.

Ideally, the Common Crawl archives should be processed month-by-month to minimize the amount of new data that needs to be processed at the same time and to maximize the filtering effect of deduplication steps based on the already downloaded monthly batches. In practice, we had to deal with much larger data sizes for two reasons. First, we already had in our possession a bootstrap corpus that we had to process in bulk. Second, we did not have the benefit of hindsight, and on occasion processed as much as 11 monthly archives in one batch. While this means that our scripts are proven to work under a much larger load, it also resulted in a few suboptimal design choices. We return to this issue in Section 4.7.

4.4.1 Download

The download step is largely identical to the one described in Indig (2018). The one significant change we decided to make was to move boilerplate removal to a separate script. Our reason for decoupling the two steps was twofold: first, it allows us to keep the raw crawl data; second, it leaves the door open for trying other boilerplate removal methods in the future. For the intricacies of the downloading process, the reader is referred to Indig (2018).

4.4.2 Boilerplate removal

Web pages (including those downloaded by and from Common Crawl) incorporate not only textual content, but various forms of *boilerplate*: scripts, navigation elements, headers and footers, etc. Most pages are also marred by an overabundance of advertisements.

As is well known, boilerplate causes problems when included in a search index or text corpora (Kohlschütter et al., 2010). It is, therefore, important to detect and extract only the main content from the raw HTML. Unfortunately, this is not a trivial task and some (typically, textual) boilerplate is expected to have made its way into the corpus.

There are several tools available for boilerplate removal. In order to maintain compatibility with the bootstrap corpus, we decided to use Justext (Pomikálek, 2011). Justext has the additional benefit of splitting the text into paragraphs, a subdivision we maintain

going forward.

4.4.3 Content-based filtering

We already emphasized the importance of quality in Section 4.1.2. Two filtering steps were applied to the downloaded corpus in order to improve its quality.

Language-based filtering To remove non-Hungarian content, language identification was performed on the corpus. We used the `CLD2`[16] and `langid.py`[17] (Lui and Baldwin, 2012) libraries, the former via the `cld2-cffi`[18]) wrapper. Our primary tool was `CLD2`, as `langid.py` proved too slow to consider, given the size of the corpus. However, it was kept in the toolchain as a backup for texts that `CLD2` could not reliably identify.

Language identification was run on a per-paragraph basis. This allowed us to handle documents which were mostly in Hungarian, but included foreign-language paragraphs (quotations, translations, etc.). Had we done the filtering on a document level, such documents would have been either removed entirely, or left in the corpus with the foreign text intact; either outcome would have been undesirable.

Length filtering As mentioned in Section 4.1.2, we set up a document length threshold to catch "low quality" pages. In absence of a theoretically justified value, we heuristically decided on a minimum length of 1500 characters, which roughly corresponds to three paragraphs of text. Documents below the threshold were dropped from the corpus.

4.4.4 Deduplication

Web crawl data shows a high degree of duplication on several levels. We already mentioned in Section 4.1.2 how a URL can appear many times in the data. We assume that each occurrence of the same URL points to the same document, and discard all but the first instance.

Even if we only consider pages with different URLs, we are still left with a lot of matching content. These come from three main sources:

- variations of the same URL with different parameters might point to the same page;

[16] https://github.com/CLD2Owners/cld2
[17] https://github.com/saffsd/langid.py
[18] https://pypi.python.org/pypi/cld2-cffi

- the same content can be replicated under different URLs, such as the news taken verbatim from MTI, the Hungarian news agency by news websites;
- sometimes duplication only affects certain paragraphs. Examples include site-specific boilerplate or paragraphs copied from another page.

The first two cases, while different, can be tackled with document-level deduplication. We do not perform paragraph-level deduplication, as it would disrupt the text flow in documents. The only exception is site-specific boilerplate paragraphs, such as the introductory panel in the sidebar on the `blog.hu` pages, and intra-document paragraph repetition.

The sections below describe these deduplication steps in a little more detail.

URL deduplication

We performed URL deduplication on the monthly URL indices, so as not to download duplicate documents needlessly. Given the roughly month-by-month processing of Common Crawl, there were two separate deduplication (sub-)steps, within-month and across-month.

Filtering URLs from earlier indices. To this end, we created a URL list file that keeps track of all URLs we have downloaded previously. When a monthly index is fetched, this file is consulted first, and all URLs in it are removed from the index. This step can be trivially parallelized.

Deduplicated URLs in the same index. It is not uncommon to see the same URL more than once even in a single monthly index. Deduplicating a single dataset is an inherently sequential process. In order to do it in parallel we employ a Redis[19] database to keep track of each URL seen in any of the index files. All Python processes have concurrent access to the database. The first process to see a URL registers it in Redis, and any further occurrences are dropped from the index. In the end, all URLs in the database are added to the static URL list mentioned in the previous paragraph, and hence automatically skipped in future indices as well.

Document deduplication

Deduplication is a much less precise matter for documents than it is for URLs, as usually we are not looking for *exact* matches. Even downloading the same webpage from the same URL twice might results in slightly different documents due to dynamic elements on the page, such as a simple clock; and a document that plagiarizes another might introduce

[19] https://redis.io/

more subtle differences. The standard practice of finding such *near duplicates* is to convert documents into sets and look for sets with a relatively large intersection (Leskovec et al., 2014, ch. 3). Due to the size of the corpus, both steps must use approximate methods to remain feasible.

MinHashing MinHash (Broder et al., 1998) is an algorithm that converts documents into a set of n-grams. Instead of working with the whole, potentially very large set, a fingerprint of the document is created from several permutations of the set. We used 256 permutations computed from character 5-grams.

Compared to e.g. language filtering, MinHash is computationally expensive, but it can be similarly parallelized (i.e. it is a 'map' task).

Locality-sensitive Hashing (LSH) A full deduplication of a corpus with $|C|$ documents would require $\mathcal{O}(|C|^2)$ comparisons, already impracticable at much lower scales than ours. We used Locality-sensitive Hashing (Gionis et al., 1999; Indyk and Motwani, 1998), an approximate method, to speed up fingerprint matching. Documents that produce a Jaccard similarity over 0.95 to an already processed document are deemed duplicates and are removed from the corpus.

Similarly to URL deduplication, LSH is performed in two steps. First, pages in the monthly crawl are deduplicated against documents already in the corpus; in order to do this, a database with the minhashes of all documents is maintained. Second, the remaining documents in the monthly batch are deduplicated against one another.

Both steps above were implemented with the `datasketch`[20] library.

Site-level deduplication of paragraphs

Some sites contain "boilerplate" paragraphs that occur in many of their documents. This kind of repetition adversely affects all language models: their presence disrupts text cohesion and makes count-based methods overestimate the importance of words in the duplicate paragraphs. Their detection and removal is, however, more involved than the document deduplication case described above. It consists of the following steps.

Indexing In order to detect paragraphs common to pages on a site, the documents first must be grouped by network domain. To avoid sorting the whole corpus, we create an *index* of the documents and sort only the index. It is then partitioned so that documents in a domain are not separated, and the partitions are distributed to each machine in the cluster for parallel processing.

[20] https://github.com/ekzhu/datasketch

Frequent paragraphs The index shards are processed domain-by-domain to find "boilerplate" paragraphs in each. Since at this point the HTML structure of the pages is no longer available, we redefine the task in terms of frequency: a paragraph is "boilerplate" if it appears in at least 50% of a site's pages *and* at least in three. Once again, we depend on MinHash and LSH for efficient near-duplicate finding.

As a domain may contain an arbitrary large number of paragraphs, keeping the MinHash of each in memory is infeasible. Instead, we reconceptualize the task as frequent item search in streams. Each domain can be considered a stream of documents; the items we count are the paragraphs. We implemented the algorithm in Leskovec et al. (2014, sec. 4.7.3), which uses a decaying window to limit the number of paragraphs in memory at any given time. While this is an approximate algorithm, and therefore lossy, it works well in practice, as the number of "boilerplate" paragraphs on any given site is bound to be small, typically one.

Filtering Once we have the frequent paragraphs for each domain, we collect those that occur in at least 50% of the site's pages and filter them from the corpus. Using the stream analogy again, we keep the first occurrence of each "boilerplate" paragraph and discard the rest.

As before, the frequent paragraph list is not thrown away after filtering, but is stored between the monthly expansions of the corpus. We consider domain "streams" continuous between monthly crawls and always initialize the stream history from the list saved in the last month.

Document-level deduplication of paragraphs

There is yet another form of paragraph duplication, not addressed above: when paragraphs are repeated within a document. Occasionally, this repetition may come naturally from the logic of the text, but most of the time, it is purely an artifact of (bad?) HTML page design (e.g. including the same content once for static and once for dynamic presentation) and Justext's inability to cope with it.

To clean up such pages, we included a script that deletes all repeated paragraphs within the same document. As opposed to the steps above, the scripts only filters *exact duplicates* to minimize the chance of deleting near duplicates included in the text for pragmatical reasons.

Both paragraph deduplication steps decrease the length of the documents affected, possibly below the threshold established in Section 4.1.2. Such documents are removed from the corpus by re-running the length filtering script.

4.4.5 Linguistic analysis

As a last step, the corpus is annotated with the `emtsv`[21] (Indig et al., 2019) pipeline. Due to the computational cost associated with the higher-level components (parsing, NER, etc.), only tokenization, morphological analysis and lemmatization was performed. We handled each paragraph separately to the pipeline. At the time of writing, the tokenizer included in `emtsv`, quntoken[22] (Mittelholcz, 2017), handled short documents very inefficiently, and it quickly became a bottleneck. We fixed this issue in our own fork[23], and also provided a Python interface to quntoken via the Python C API. As our changes have not yet been merged into the main line, all scripts use the fork above.

`emtsv` outputs a tsv file with separate columns for each annotation. Each line contains the annotations for a single token, with empty lines between sentences. We expanded this format with CoNLL-U Plus[24]-style comments to store the document URL and the raw untokenized text in the file as well, so that the original text-only format can be reproduced.

Dataset	Documents	Paragraphs	Words	Bytes / Characters*
Full index	244,544,787			4,521,082,715,740
Filtered index	181,504,355			2,915,894,695,649
Deduplicated index	68,940,595			1,182,608,541,578
Downloaded	69,065,628			6,147,433,875,452
Boilerplate removed	36,396,615	446,695,117	15,368,435,601	115,037,549,741
Language filtering	35,832,801	429,280,674	15,204,586,348	113,955,430,850
Length filtering	15,063,882	310,009,838	13,240,540,507	99,222,973,992
Document deduplication	9,965,023	189,777,575	8,637,317,522	64,975,071,293
Frequent P filtering	9,062,484	158,772,773	7,194,048,746	53,790,414,086
Same-doc. P dedup.	9,052,140	153,732,983	7,086,763,519	52,986,174,365
Final length filtering	8,326,246	149,945,430	6,989,318,926	52,263,895,796

Table 4.1: Effects of the various filtering steps on corpus size (2018) (*Number of compressed bytes above the middle rule; number of characters below)

4.4.6 Final statistics

The almost valid CoNLL-U Plus format described in the last section is the canonical format of our corpus. We split up the corpus into files of 2,500 documents each to make it easier to download and work with the corpus; it also enables prospective users to scale

[21] https://github.com/dlt-rilmta/emtsv
[22] https://github.com/dlt-rilmta/quntoken
[23] https://github.com/DavidNemeskey/quntoken/tree/v1
[24] https://universaldependencies.org/ext-format.html

Dataset	Documents	Paragraphs	Words	Characters
Boilerplate removed	85,744,991	118,0528,666	44,648,104,301	333,609,180,321
Language filtering	84,144,867	951,425,101	43,535,586,773	326,744,152,415
Length filtering	33,231,844	785,179,604	39,053,562,970	292,548,775,478
URL deduplication with 2018	13,917,392	280,518,344	10,414,399,879	76,099,744,178
Document deduplication†	3,587,552	64,064,899	2,408,994,751	17,555,305,848
Document deduplication with 2018†	2,905,457	52,885,668	1,931,680,268	14,092,687,812
Frequent P filtering	2,594,896	47,242,751	2,159,385,560	16,061,541,290
Same-doc. P dedup.	2,591,241	45,718,462	2,126,606,691	15,815,393,328
Final length filtering	2,412,516	44,706,281	2,101,516,357	15,628,819,466

Table 4.2: Effects of the various filtering steps on corpus size (2013–2017) (†A small number of duplicate URLs remained in this dataset)

the data they wish to work with. The order of the documents is randomized to make the each minibatch as representative of the whole corpus as possible, which is very important for stochastic gradient-based learning algorithms (Bengio, 2012).

The size of the final corpus is shown in Table 4.3. With a little over 9 billion tokens, the corpus is about 3.5 times the size of huTenTen and 4 times the size of the deduplicated OSCAR corpus. The Wikipedia subcorpus, described in the next section, is also included in the table for comparison. As can be seen, with about 56 times as many tokens, the Common Crawl portion of the corpus dwarfs Wikipedia's contribution. Yet the usefulness of a clean, semi-edited source should not be overlooked.

Source	Documents	Paragraphs	Words	Characters
Common Crawl	10,738,762	194,651,711	9,090,835,283	67,892,715,262
Wikipedia	301,312	7,513,420	163,450,200	1,118,833,863
Sum	11,040,074	202,165,131	9,254,285,483	69,011,549,125

Table 4.3: The size of the current version of Webcorpus 2.0

Table 4.4 lists all domains with at least 0.5% of the total number of documents in the corpus. As can be seen, the top is dominated by news outlets (8.81%), blog services (7.23%) and document players (4.17%), which host longer documents such as legal texts and slides. At a glance, the top domains are of good quality and the content distribution seems useful for a training corpus. Blogs, news and legal documents will probably be three pillars of a future balanced corpus as well.

The presence of the e-commerce site `arukereso.hu` in the list is questionable, and needs further investigation.

Domain	Pages	Percentage
blog.hu	589,346	5.49%
docplayer.hu	384,136	3.58%
index.hu	199,790	1.86%
24.hu	187,686	1.75%
hvg.hu	131,918	1.23%
blogspot.hu	130,690	1.22%
delmagyar.hu	114,585	1.07%
kisalfold.hu	106,402	0.99%
napi.hu	75,487	0.70%
arukereso.hu	70,789	0.66%
origo.hu	68,465	0.64%
slideplayer.hu	63,701	0.59%
fidelio.hu	61,818	0.58%
xfree.hu	56,068	0.52%
webbeteg.hu	55,510	0.52%
Total	2,296,391	21.4%

Table 4.4: Top domains in terms of the number of documents

4.5 Wikipedia

In order to use as many semi-edited sources as possible, we are including the Hungarian Wikipedia (WP)[25] in Webcorpus 2.0. Wikipedia is not incorporated into Common Crawl, so it had to be obtained and processed separately. The code that implements the steps below can be downloaded from GitHub[26].

4.5.1 Wikihopping[27]

Extracting text from Wikipedia proved more challenging than we previously imagined. On the face of it, the procedure is simple: the Wikimedia Foundation regularly publishes dumps for each language. These dumps can be downloaded and processed with any of the third party parsers[28] available. As we have learned, however, nothing could be farther from the truth. We include our whole Odyssey towards a working solution to help readers who find themselves in the same boat.

As it turns out, while many parsers can extract data from *infoboxes* on WP pages, only WikiExtractor[29], also endorsed on the BERT GitHub page[30], can parse the main text. Unfortunately, WikiExtractor fails to expand some templates in the text, which

[25] https://hu.wikipedia.org/wiki/Kezdőlap
[26] https://github.com/DavidNemeskey/zim_to_corpus
[27] https://www.urbandictionary.com/define.php?term=wikihopping
[28] https://www.mediawiki.org/wiki/Alternative_parsers
[29] https://github.com/attardi/wikiextractor
[30] https://github.com/google-research/bert

makes certain elements, such as numbers with units (e.g. 40 miles) to disappear from the output[31].

Not wanting to lose sentence cohesion, we turned next to the official MediaWiki software. While there is no way to extract text from the dump directly, there are two tools that can import the dump to a local MediaWiki server, from which the pages can be queried via a REST API. In practice, the built-in `importDump.php` is prohibitively slow, and MWDumper[32], while much faster, failed to build a valid local WP mirror.

The final solution came in the form of parsing not the MediaWiki source, but the HTML pages themselves. Unfortunately, Wikimedia discontinued the distribution of static dumps in 2008; however, we found another source in the form of the Kiwix ZIM archives[33]. These archives are updated less frequently (every 1–2 years), but are freely available and they can be parsed with the open source libzim[34] library. We converted the 2017 Hungarian WP ZIM archive into a set of files in the JSON Lines[35] format. Disambiguation and redirect pages were purposely omitted from the output, as they contain no coherent text.

4.5.2 Processing

The HTML source of Wikipedia pages is free of advertisements and similar boilerplate, and extracting the main text is straightforward via CSS selectors. Data duplication is also nonexistent, as each page corresponds to a different topic, usually a named entity. As a consequence, we could arrive at a good quality corpus with only two preprocessing steps.

First, sections that consist only of enumerations (e.g. *Lásd még* 'See also' or *Források* 'Sources') are not useful for language modeling. To filter them, we assembled a list of section titles that 1. usually occur around the end of the document (last 1–2 paragraphs) and 2. contain only a list 80% of the time. While the numbers are ad-hoc, the resulting list seems to include most of the worst offenders. The listed sections were then removed from all documents.

Second, low quality pages were also removed; the metric, once again, was length. Documents shorter than 100 words were deleted from the corpus; these mostly concerned one-sentence descriptions of asteroids, probably autogenerated from an astronomical database.

As with the Common Crawl part, the filtered corpus was converted to tsv format, and morphologically analyzed and lemmatized with `emtsv`. Since here we parsed the HTML format directly (as opposed to having it done for us by Justext), we could give meaningful

[31] https://github.com/attardi/wikiextractor/issues/189
[32] https://www.mediawiki.org/wiki/Manual:MWDumper
[33] https://www.mirrorservice.org/sites/download.kiwix.org/zim/wikipedia/
[34] https://github.com/openzim/libzim
[35] http://jsonlines.org/

IDs to the paragraphs based on their location in the DOM. We use these IDs to convert the tsvs into popular language modeling formats, such as BERT, which requires the removal of lists and headers, and WT2.

Dataset	Documents	Paragraphs	Sentences	Words
Full data	418,673	8,197,320	13,952,709	169,541,528
Length filter	301,312	7,513,420	13,098,808	163,450,200
BERT	260,181	1,996,905	6,093,173	114,093,719

Table 4.5: Effects of the various filtering steps on corpus size (Hungarian Wikipedia)

The full WP contains 418,673 pages; as Table 4.5 shows, removing short documents left us with about 72% of that, though we only lost 3.5% of tokens. Conversion to the BERT format reduces the token count by another 30%.

4.6 huBERT

One of our main goals with the corpus was to train a Hungarian BERT model, nicknamed `huBERT`. Unfortunately, at the time of writing, no results were available yet. In this section, we present preliminary work based on the Wikipedia subcorpus.

4.6.1 Pretraining

As we have seen in 1.6.4, training of large Transformer models is prohibitively costly for smaller research groups; and in Hungarian NLP, all groups can be considered small. Luckily, the Wikipedia subcorpus is small enough that a BERT `Base` model can still be pretrained on a single TPU. We had access to a small number v3 TPUs via Google's TensorFlow Research Cloud[36] program, which allowed us to train the Wikipedia BERT models in a couple of days.

BERT models usually come in two flavors: *cased* and *uncased*. The former operates on unprocessed text; in the latter, tokens are lower cased and diacritical marks are removed. In keeping with this practice, we also trained two variants. However, as diacritics are *distinctive* in Hungarian, we could not afford to lose them, and replaced the uncased model with a *lower cased* one.

As is the case with the English BERT, our models are all pretrained on sequences of up to 512 wordpieces. Since the training configurations published in the literature are for

[36] https://www.tensorflow.org/tfrc

much larger corpora, they are not directly adaptable to our case. Hence, we experimented with different training regimens for both the cased and lower cased variants:

1. Three models were trained with full-length sequences for 50,000, 100,000 and 200,000 steps. These roughly correspond to 90, 180 and 360 epochs, respectively;

2. Following the recommendation in the BERT repository, one model was trained with a sequence length of 128 for 500,000 steps (600 epochs) and with a sequence length of 512 for an additional 100,000 steps (or 180 epochs).

All models were trained with a maximum learning rate of 10^{-4} and the maximum possible batch size: 1024 for the model with 128-long sequences and 384 for the rest.

Model	Seq. length	Steps	Hours	Masked LM	Next sentence
Cased	512	50,000	13	0.5544961	0.97125
	128	500,000	59	0.6669028	0.995
	512	+100,000	25	0.6657926	0.99
Lower	512	50,000	13	0.5538445	0.9825
	512	100,000	25	0.6100383	0.9975
	512	200,000	50	0.6273391	0.9975
	128	500,000	59	0.6425686	0.99125
	512	+100,000	25	0.665879	0.9975

Table 4.6: Training times and accuracies of the different BERT models on the two training tasks

Table 4.6 compares the different configurations. In the cased case, the TPU went down for maintenance during the training, so the 100,000 and 200,000-step models are missing from the results. Even without them, several observations can be made. First, the 50,000-step models clearly underfit the data, even though they were trained for twice as many epochs as the English BERT. On the other hand, the difference between the 100,000 and 200,000-step models is much smaller than between the 50,000 and 100,000-step models, suggesting a performance peak around 300,000–400,000 steps.

Second, in line with the findings of Lan et al. (2019) and Liu et al. (2019), the next sentence prediction task seems very easy, as all but the first models attain over 99% accuracy. In contrast, the masked LM task proved much harder, and its accuracy seems rather low. Unfortunately, the evaluation results for the English BERT are not published anywhere, which makes it difficult to put the numbers in context. Based on the diminishing returns, the longest-trained models are likely to be close to the maximum achievable on the Wikipedia subcorpus.

Finally, our experiences confirmed that the two-stage training regiment recommended in the BERT repository leads to better results. The rationale behind this method is that the first phase trains most of the model weights and the second phase is "*mostly needed to learn positional embeddings, which can be learned fairly quickly*"[37]. While this seems to be the case for the cased model, the masked LM accuracy of the lower cased model improved by more than 2% in the second phase, indicating that substantial learning still happens at this stage.

4.6.2 Evaluation

BERT models are usually evaluated on high-level natural language understanding tasks, such as question answering or textual entailment. Unfortunately, no Hungarian benchmark datasets exist for these tasks. Because of this, we evaluate our models by contrasting their performance to the multi-language version of BERT in three ways:

1. We compare their accuracy in the two training tasks on a held-out portion of the Wikipedia subcorpus.

2. We include our models in the `emBERT` module (see Chapter 5) and measure their performance on named entity recognition and NP chunking.

3. The quality of the tokenization vocabulary used by the models is also compared.

Table 4.7 presents the results of the first experiment. Both our cased and lower cased models achieve similar accuracies on the held-out set as on the training data, allaying any suspicion of overfitting. In addition, the cased model clearly outperforms multi-BERT on both tasks (multi-BERT is only available in the cased configuration). In fact, the accuracy of multi-BERT on masked LM is equivalent to a perplexity of about 50,000, which, given its vocabulary of 120,000 wordpieces, is little better than random. As multi-BERT was also trained on Wikipedia, this abysmal performance is especially surprising.

The other two experiments will be discussed in Chapter 5, but to summarize the results briefly, our models outperform multi-BERT in both aspects, proving the effectiveness of native Hungarian contextual embeddings over multi-language models and their necessity for progress in Hungarian NLP.

The models can be downloaded from `https://hlt.bme.hu/en/resources/hubert`.

[37] `https://github.com/google-research/BERT/#pre-training-tips-and-caveats`

Model	Masked LM accuracy	Masked LM loss	Next sentence accuracy accuracy
Cased 128–512	0.6414373	1.85	0.98625
multi-BERT	*0.0857188*	*10.89*	0.51625
Lower 128–512	0.6415611	1.84	0.99

Table 4.7: Performance of our best models and multi-language BERT in the two training tasks on the held-out set.

4.7 Conclusion and future work

In this chapter, we presented our work on a Webcorpus 2.0, the largest Hungarian corpus to date. The corpus is composed of the Hungarian pages in the Common Crawl archives and a copy of the Hungarian Wikipedia. At around 10 billion tokens, the resulting corpus is 3.5–4 times as large as the previous record holders huTenTen and OSCAR.

The main purpose of the new corpus is to provide training grounds for modern Transformer-based language models. As preliminary work to a Hungarian BERT, we experimented with a model pretrained on the Wikipedia subcorpus. Our model outperformed the multi-language version of BERT in all experiments we conducted.

The largest remaining task is a BERT model built on the whole Webcorpus 2.0. We are working towards this goal, and intend to publish the results soon.

A key issue is that of training text quality. BERT training corpora exclude headers and lists. We followed the convention in the bootstrap corpus that each extracted text chunk is considered a paragraph, irrespective of its originating HTML tag. Since this gives us no information on how the paragraphs should be interpreted, we cannot put a similar filtering step in place. The exception is the Wikipedia subcorpus, which does feature typed paragraphs. It turns out, however, that it is possible to extend this solution to the main corpus as well, by using the `xpath` attribute of the paragraphs emitted by Justext. In the next version of the corpus, we intend to make use of this feature to create a better quality BERT training corpus. Filtering lists and tables will probably result in the downsampling of domains like `arukereso.hu` as well.

We are grateful for Indig's bootstrap corpus, which allowed us to start our work. However, having to deal with a huge dataset not yet deduplicated, and without the original WARC sources, meant we could not fully operate on a month-by-month basis. This made the code more complex, and as mentioned above, prevented us from changing the output of the boilerplate removal. In the next version of the corpus, we intend to re-download the 2013–2017 archives and integrate them fully into our pipeline.

The next version of the corpus might also explore some of the issues raised in this chapter, such as the compilation of a balanced subcorpus or the inclusion of Hungarian pages from the Common Crawl archives of neighboring countries.

Lastly, the implementation also might need some overhaul. Python is a good language for fast prototyping and due to its excellent third party library support. However, its preference of multiprocessing over multithreading results in reduced performance, increased memory usage or the need to introduce a database (see Section 4.4.4) for tasks that need a large shared resource (e.g. a URL index). Reimplementing the tasks affected in a language that supports multithreading could speed them up considerably.

Chapter 5

`emBERT`: language modeling for NLP

In the previous chapters, we presented various ways in which "traditional" natural language processing can be used to evaluate or improve certain aspects of language modeling. Notably, modern language modeling depends on the existence of gigaword corpora, such as the one introduced in Chapter 4 – which utilizes numerous NLP tools, from language identification and boilerplate removal to tokenization and morphological analysis.

In this chapter, we give an example of the opposite direction, when language modeling is used to improve an NLP system. Specifically, we train classifiers based on contextual embeddings for two NLP tasks: NP chunking and named entity recognition. Our work is certainly not without precedent (see Section 1.7.5); however, to our knowledge, it is the first time such a study is conducted for Hungarian.

Most of the content of this chapter was published in Nemeskey (2020), which has won Special Award at the XI. Conference on Hungarian Computational Linguistics.

5.1 Deep learning in NLP

We have already discussed how in the last couple of years deep, contextual embeddings superseded traditional, manually compiled feature sets in most NLP tasks (see Section 1.7.5). In spite of this development, Hungarian text processing pipelines, namely `e-magyar` (Váradi et al., 2017) and `magyarlanc` (Zsibrita et al., 2013), still operate on manual features. In this chapter, we introduce the `emBERT` module, which enables the integration of sequence classifiers based on contextual embeddings into `emtsv`, the new version of `e-magyar`. We trained models for two tasks: NP chunking and named entity recognition (NER). `emBERT` performs comparably to the best NER tagger for Hungarian, while achieving state-of-the-art performance on NP chunking.

Another key aspect of the deep learning revolution is the retirement of off-the-shelf

classifiers in favor of custom, usually very large neural architectures. The exploration of how much this development affects or will affect NLP is beyond the scope of this thesis; at least for now, traditional machine learning models seem prevalent in NLP toolkits. While the system described in this chapter uses a neural classifier, it is much simpler than the maximum entropy Markov models or CRFs commonly used for token classification.

The results presented in Section 1.7.5 were all achieved by English-language models on English benchmarks. Owing to the resource-hungry nature of training large contextual embeddings, only a handful of languages have followed suit[1]. In this chapter, we investigate if contextual embeddings are able to achieve similar results for Hungarian. Due to the lack of evaluation datasets akin to SQuAD or GLUE, we assess model performance on two token classification tasks: NP chunking and NER. The models are integrated into the `e-magyar` text processing system as a new module.

5.2 BERT

5.2.1 Why BERT?

From the list of contextual embeddings introduced in Section 1.7.4, few are available in languages other than English. In particular, only one of them has a Hungarian version, although some of the models have configurations trained on multilingual text. ELMo (Che et al., 2018) has individual models for 44 languages, trained on 20-million-word samples of the CoNLL 2017 shared task *Multilingual Parsing from Raw Text to Universal Dependencies* (Zeman et al., 2017). BERT, XLM and XLM-RoBERTa have joint multilingual models, which have been trained on roughly 100 languages (BERT on 104). These models have an extended vocabulary (120 thousand for BERT, 240 thousand for XLM) compared to their single-language versions, but the number of parameters is the same; additionally, the multi-language BERT is only available in the `Base` (110M) parameter configuration. *multi-BERT* was trained on Wikipedia; XML-RoBERTa on 2.5TB of Common Crawl data.

In this study, we decided to focus on BERT. The main advantage over ELMo is that BERT models can be *fine-tuned* on downstream tasks easily, whereas ELMo only replaces the featurizer component. Secondly, the BERT family of models generally achieve better performance on high-level tasks. NP chunking and NER are notable exceptions from this rule, so replacing manual features with ELMo in `emChunk` and `emNER` is a possible future avenue for research.

We include two BERT models in our study. The first one is multi-BERT; the second is

[1] https://huggingface.co/transformers/pretrained_models.html

Word	Multilingual BERT	Hungarian BPE	English gloss
Nemzeti	Nemzeti	Nemzeti	National
Andersen	Andersen	Andersen	Andersen
labdarúgó	labdarúgó	labdarúgó	footballer
zambiai	za mbia i	z amb iai	of Zambia
megmaradt	meg maradt	megmaradt	remained
hétfő	hét f ő	hétfő	Monday
keddtől	ke dd től	kedd től	from Tuesday
edényben	ed ény ben	edény ben	in a pot
Hétfőn	H ét f ő n	Hétfőn	On Monday
tájékoztatták	tá j ék ozta tták	tájékoztat ták	have been notified
leggazdagabb	leg ga zda gab b	leggazdagabb	wealthiest
elpártolt	el pá rto lt	el párt olt	renounced

Table 5.1: Tokens generated from a few words by the dictionary of multilingual BERT and a Hungarian BPE dictionary built from Webcorpus 2.0 (see Chapter 4)

the preliminary `huBERT` model introduced in Section 4.6. We use the cased `huBERT` model in all experiments, as case information is important in named entity recognition.

5.2.2 Does multi-BERT speak Hungarian?

The multi-language BERT model was trained on 104 languages. This raises the question: how well does it model Hungarian? More specifically, we can consider the following two facets:

1. To what extend do wordpieces in a tokenized sentence correspond to (Hungarian) morphemes?

2. Are wordpiece vectors semantically correct? This question is especially interesting with regard to interlingual homographs; e.g. "*leg*" (the superlative prefix in Hungarian), "*old*" (Hungarian *solve*)?

A proper discussion of the second question would be beyond the scope of this chapter. An implicit answer can be construed from the results presented in Section 5.5.

In order to answer the first, we tokenized all words in the Szeged NER corpus into wordpieces in two ways. First with the tokenizer of multi-BERT; second, with a 30,000-token BPE vocabulary trained on Webcorpus 2.0 that was introduced in Chapter 4. Table 5.1 lists a few select examples.

Word type	Multilingual BERT	Hungarian BPE	Difference
lowercased	2.24	1.34	67%
capitalized	1.86	1.75	6%
all	2.14	1.44	49%
lowercased (type)	3.97	2.41	65%
capitalized (type)	4.65	4.27	9%
all (type)	4.12	2.83	45%

Table 5.2: Mean number of subwords per token (above) and type (below)

As seen in the table, words can be divided into roughly three groups. Words in the first group are handled similarly by the two tokenizers; either because they are present in both vocabularies (in other words, they are wordpieces themselves), or because neither tokenizer can split them up into meaningful units. "*zambiai*" is an example for the latter.

In case of the second group, the Hungarian BPE tokenizer always splits words into morphologically sound units, whereas multi-BERT's tokenization also contains meaningless n-grams. Furthermore, the number of wordpieces on the multilingual side is always greater than on the Hungarian side.

In the third group, the gap widens further. While the Hungarian BPE tokenization is still semantic, multi-BERT's wordpieces deteriorate into random n-grams. The latter also needs 4–5 wordpieces to cover longer words, as opposed to the 1–2 needed by the Hungarian tokenizer. Additionally, the fact that "*hétfő*" and "*Hétfőn*" are tokenized differently hints at a difficulty in properly handling sentence-initial words and named entities.

Table 5.2 provides quantitative confirmation for the observations above. Multi-BERT produces on average 50% more wordpieces than the Hungarian BPE. Since punctuation marks and the most common function words, such as "*a*", "*az*" (English *the*) and "*és*" (English *and*), are included in both vocabularies, the effect on content words is likely even more pronounced.

The disparity holds for words and types alike. Comparing the top and bottom half of Table 5.2, however, reveals that the average number of subwords for types is about 78% higher than it is for downcased tokens. Since infrequent words account for a much higher proportion, this hints at how much worse infrequent words are represented across the board.

Interestingly, there is no significant (quantitative) difference in how capitalized words are handled: both tokenizers split them up into 4–5 wordpieces. This can be explained by the relative sparsity of capitalized words in the text. At the same time, it also suggests that BERT (multi-language or otherwise) might not be the optimal choice for NER.

Inspecting huBERT's vocabulary in a similar way, we find that it behaves very similarly to the Hungarian BPE vocabulary. Minor differences include the morphologically valid splitting of "*hétfő*" into tokens "*hét*" és "*fő*", and a 1–8% increase in wordpieces, compared to the numbers in Table 5.2. The differences, however, are insignificant compared to the ones between multi-BERT and the two native Hungarian vocabularies.

5.3 The emBERT module

So that researchers and NLP practitioners may benefit from the models resulting from this study, we decided to integrate them into the e-magyar pipeline. The new version of e-magyar, emtsv (Indig et al., 2019), has greatly simplified the the extension of the core system with new modules. Thus emBERT was born.

emBERT follows the same conventions as other emtsv modules. When installed, the tools bert-base-chunk, bert-max-chunk and bert-ner are made available in emtsv. Like the rest of the modules, the tools expect tokenized text as input. Since BERT operates on raw text, in theory, emBERT could as well; however, in order to keep the tokenization consistent with the rest of the pipeline, we delegate the task to emToken. In contrast with other high-level modules (emChunk and emNER in particular) emBERT requires no morphological information, rendering morphological analysis and lemmatization unnecessary[2].

We used Huggingface's transformers[3] (Wolf et al., 2019) library to fine-tune and run BERT. A considerable advantage of transformers is that it contains the implementations of not only BERT, but other Transformer-based architectures (XLNet, RoBERTa) as well. This makes it possible to later improve emBERT by integrating other embeddings into it.

As opposed to most other emtsv modules[4], emBERT contains code for both training and running BERT-based models, and it also works as a standalone Python package. We chose to include the training code for two reasons: first, the complexity (or lack thereof) of the code did not justify a segregation of the two functions; and second, it allows users to freely experiment with the models and possibly tailor them to their own needs. Similarly to the rest of the modules, the code lives in a GitHub repository[5].

At around 700MB apiece, BERT models are sizable even in the Base configuration. In order to keep the code manageable, the models have been moved into a separate GitHub repository, emBERT-models[6]. As with the HunTag3 and parser modules, the models can

[2]Unless the user or other modules need that information, of course.
[3]https://github.com/huggingface/transformers
[4]HunTag3 is an exception
[5]https://github.com/DavidNemeskey/emBERT
[6]https://github.com/dlt-rilmta/emBERT-models

be acquired in two ways: on the one hand, the `emBERT` repository includes `emBERT-models` as a submodule, so when *cloned* recursively, the models are downloaded as well. On the other, we included our models in `emtsv`'s model downloader script to give users more control over their `emtsv` setup, and to protect them from spurious downloads.

5.4 Experiments

The performance of the models was measured on two token classification tasks: NP chunking and named entity recognition. In order to facilitate comparison with previous work, the models were trained and evaluated on standard benchmark corpora.

Similarly to all Hungarian statistical NP-chunkers (`hunchunk` (Recski, 2010) and its successors), our chunker models were trained on the Szeged Treebank 2.0 (Csendes et al., 2005). The corpus contains 82,099 sentences. These we split randomly into train, validation and test sets (80%–10%–10%). Both subtasks, minimal and maximal NP chunking, were handled the same way: the models were fine-tuned for 4 epochs and evaluated on the test set. We used a linear learning rate schedule with 10% warm-up, with no early stopping.

The NER model was fine-tuned on the Szeged NER corpus (Szarvas et al., 2006), a subset of the Szeged Treebank. Since the corpus is much smaller than the full Treebank (with 8172–502–900 train/validation/test cuts), several different hyperparameter configurations were explored. The best model was trained for 30 epochs with a linear learning rate schedule peaking at 10^{-5}.

The experiments were implemented with the PyTorch version of the `transformers` library. Training was run parallely on three GeForce RTX 2080 Ti cards, with a batch size of 16. This configuration allowed us to train both the NER and chunking models (the latter of which ran for far less epochs) in 3 hours. For chunking, the values of most hyperparameters were left at their defaults. In case of NER, the hyperparameter settings that performed the best were also using the default values, with the number of epochs and the learning rate being the notable exceptions.

The exact values of all hyperparameters are recorded in the configuration files of the downloaded models.

5.5 Results

5.5.1 Chunking

Table 5.3 compares the performance of `emBERT` and members of the `hunchunk` family. Clearly, both `emBERT` models outperform all previous systems and achieve state-of-the-art results in both minimal and maximal NPs chunking.

For minimal NPs, multi-BERT's improvement over `hunchunk`, the only model trained for recognizing them, is not significant. On the other hand, the maximal NP chunker surpasses Huntag3, `e-magyar`'s tagger of choice, by a considerable 1.5% F1. `huBERT` improves both scores by a further 1–1.5%, improving on the previous state of the art by 1.16% in minimal and 2.82% in maximal NP chunking.

System	Minimal	Maximal
`hunchunk`/HunTag (Recski, 2010)	95.48%	89.11%
HunTag3 (Endrédy and Indig, 2015)	–	93.59%
`emBERT` w/ multi-BERT	95.58%	95.05%
`emBERT` w/ `huBERT`	**96.64%**	**96,41%**

Table 5.3: Comparison of Hungarian NP chunkers

5.5.2 Named entity recognition

The results for named entity recognition are more mixed (see Table 5.4). While `emBERT` achieves 2% higher F1 score than Szarvas et al. (2006) and Varga and Simon (2007), it falls shorts of Huntag3's performance. Also, multi-BERT and `huBERT` perform virtually identically in this task. This implies a bottleneck outside the main model, probably in the simple feedforward classifier placed on top.

spaCy differs from the other systems in that its training data was augmented with the hunNERwiki corpus (Nemeskey and Simon, 2012). As such, any comparison with it is purely informal, and it is only listed for the sake of completeness.

During NER training, we ran into a typical problem that plagues most machine, and especially deep, learning systems: while results depend heavily on hyper-parameter choice, finding the optimal hyperparameters is extremely costly. This issue affects small corpora, such as Szeged NER, even more, as the model has magnitudes more parameters than there are training instances.

System	F1
(Szarvas et al., 2006)	94,77%
`hunner` (Varga and Simon, 2007)	95.06%
HunTag3 (Endrédy and Indig, 2015)	**97.87%**
`emBERT` w/ multi-BERT	97,08%
`emBERT` w/ `huBERT`	97,03%
spaCy[7]	*93,95%*

Table 5.4: Comparison of Hungarian NER taggers

Compared to NER, chunker training was extremely stable: we experimented with training regimens of various epoch length, with no discernible effect on the final result. This indicates that the hyper-parameter problem can at least be mitigated with a large enough training corpus. We believe NER training could also benefit from the availability of a larger and more varied NER corpus; a possible option could be the inclusion of a manually revised subset of hunNERwiki.

5.6 Future work

The `emBERT` module, while improves on the previous state-of-the-art in NP chunking, must still be considered proof-of-concept. We review the outstanding issues, as well as the corresponding lines of future research, below.

First, we have seen that basing the module on the multi-language BERT is undeniably suboptimal. Apart from it being available only in the `Base` configuration, the capacities of both the model and the vocabulary are divided among 104 languages. The fact that `huBERT` outperforms multi-BERT by more than a full percent in both chunking tasks suggests that a BERT model trained on the full Webcorpus 2.0, especially a `Large` one, would presumably achieve further improvements. As mentioned in Chapter 4, we plan to train and publish such models in the future.

Second, BERT is but one of the many contextual embeddings published in the last two years. As we have seen, ELMo, RoBERTa or Flair surpass BERT in certain tasks; and named entity recognition is one of them. We intend to include other embeddings in `emBERT` as well, should a multi-language or Hungarian version become available.

Third, we should find out what language tasks, other than chunking and NER, can benefit from BERT. The obvious candidate is morphological analysis, for which a deep

learning approach has already proven successful (Ugray, 2019). Provided that resources similar to GLUE (Wang et al., 2018) or SQuAD (Rajpurkar et al., 2016) become available for Hungarian, the model could also be adapted to higher-level tasks such as sentiment analysis, paraphrase detection or question answering. In incorporating them, `emBERT` would not only improve on functions already provided by `e-magyar`, but could bring new capabilities to it as well.

5.7 Conclusion

In this chapter, we have have introduced `emBERT`, a new module of the `e-magyar` text processing system. `emBERT` allows the integration of classifiers based on contextual embedding into the pipeline. We fine-tuned both the multilingual BERT and the preliminary `huBERT` models on the tasks of NP chunking and named entity recognition. Our models perform comparably to previous NE recognizers, and achieve a new state-of-the-art in NP chunking.

`emBERT` offers various opportunities for improvement. The module can easily be extended to support additional embeddings and tasks, as long as the necessary resources (a training corpus and the embedding itself) are available.

Chapter 6

Conclusions

In this thesis we concentrated on two issues: how modern NLP (or traditional linguistic) techniques can improve language modeling, and how to improve the state of the art for Hungarian.

Chapter 2 highlighted the problems of word-based language modeling for Hungarian and introduced the "gluten-free" format, a morphological segmentation algorithm that alleviates the adverse effects of the overabundance of word forms in the language. Chapter 3 proposed a novel method for evaluating multi-sense embeddings based on lexicographical resources. Chapter 5 gave an example for the opposite direction, when language models are used to improve the performance of an NLP system.

We improved on the state of the art in Hungarian language modeling in several ways. First, we presented a set of language modeling benchmarks on three Hungarian corpora in Chapter 2. A preprocessed version of the Hungarian Webcorpus has been released to serve as a standard dataset for language model assessment. A new Hungarian corpus has been created in Chapter 4. Webcorpus 2.0 was compiled from Hungarian pages in the Common Crawl and the Hungarian Wikipedia. At 9 billion tokens, it is 3.5 times the size of the previous largest (commercial) corpus, and can serve as training data for large-scale language models. Finally, the `emBERT` module developed in Chapter 5 enables the integration of modern contextualized embedding-based classifiers into the `e-magyar` pipeline. Based on our preliminary Hungarian BERT model, the NP chunker outperforms the previous best system by 2.8% in F1 score.

All resources (corpora, models and software alike) presented in the thesis are freely downloadable under permissive licenses.

Összefoglalás

A disszertációban két kérdésre koncentráltunk. Egyfelől célnak tűztük ki a magyar nyelvmodellezési eredmények javítását; másrészt azt vizsgáltuk, hogy alkalmasak-e erre a célra a modern természetesnyelv-feldolgozás (vagy a hagyományos nyelvészet) módszerei.

A 2. fejezet bemutatta a problémákat, amivel a magyar nyelv szószintű modellezésekor találkozunk. Itt mutattuk be a „gluténmentes" formátumot, egy morfológiai szegmentáló algoritmust, ami ha nem is orvosolja, de csökkenti a szóalakok túlburjánzása okozta problémákat. A 3. fejezet egy értelmező szótárakon alapuló módszert javasolt többjelentésű szóbeágyazások kiértékelésére. A 5. fejezet az ellenkező irányra ad példát, amikor nyelvmodellek alkalmazása javítja egy nyelvfeldolgozó rendszer eredményét.

Kutatásunk keretében több módon is korszerűsítettük a magyar nyelvmodellezés állapotát. Egyrészt több nyelvmodellt is kimértünk három magyar korpuszon. Ezek közül egyiket, a Magyar Webkorpusz előfeldolgozott változatát javasoltuk sztenderd nyelvmodell-kiértékelő korpusznak a 2. fejezetben. A Webkorpusz 2.0 létrehozását mutatja be a 4. fejezet. Az új korpusz a Common Crawl webarchívumból és a magyar Wikipédiából tevődik össze. Kilenc milliárd szavas méretével a legnagyobb magyar korpusz, mérete 3,5-szörösen haladja meg a legnagyobb (kereskedelmi) korpuszét. Végül a 5. fejezet mutatja be az `emBERT` modult, ami lehetővé teszi modern kontextualizált beágyazások integrálását az `e-magyar` szövegfeldolgozó rendszerbe. A kísérleti magyar BERT modellünkre építő főnévicsoport-felismerő 2,8% F1-ponttal múlja felül a korábbi legjobb rendszert.

Minden bemutatott erőforrás (korpuszok, modellek és szoftver) szabadon letölthető.

Appendices

Appendix A

Abbreviations used in the thesis

1B	One Billion Word Benchmark	**ME**	maximum entropy
AR	activation regularization	**MEMM**	maximum entropy Markov model
BERT	Bidirectional Encoder Representations from Transformers	**ML**	machine learning
		MLE	maximum likelihood estimation
BPTT	backpropagation through time	**MLP**	multilayer perceptron
BPE	byte-pair encoding	**MSE**	multi-sense embedding
CC	Common Crawl	**NER**	named entity recognition
CLD2	Compact Language Detector 2	**NFS**	network file system
CNN	Convolutional neural network	**NLP**	natural language processing
CRF	conditional random field	**NLU**	natural language understanding
DOM	document object model	**(N)MT**	(neural) machine translation
ELMo	Embeddings from Language Models	**NNLM**	neural net language model
FFNN	feedforward neural network	**NP**	noun phrase (as in "NP chunking")
FLM	factored language model	**OOV**	out-of-vocabulary word
GLF	gluten-free (form)	**POS**	part-of-speech
GPU	graphics processing unit	**PTB**	Penn TreeBank
HMM	hidden Markov model	**QA**	question answering
KN	Kenser-Ney (n-gram model)	**SGD**	stochastic gradient descent
LAN	local area network	**TPU**	tensor processing unit
LM	language model(ing)	**tsv**	tab separated values
LOB	Lancaster–Oslo/Bergen (corpus)	**w2v**	word2vec
LSH	locality-sensitive hashing	**WP**	Wikipedia
LSTM	long short-term memory	**WT2**	WikiText-2

Appendix B

Sample texts generated by Transformer models

There is little doubt how President Trump ...

> views Russia's invading of Ukraine and annexation of Crimea. He sees the actions of Russian leader Vladimir Putin as evidence of Russian political subversion in America.
>
> We may never know the extent to which Trump and his advisers also suspect that a key feature of American politics is a "deep state" that is working to subvert the will of the American people at home and to promote Kremlin policies abroad. One sees them attempting to prevent Trump's nomination as a Republican candidate and then to sabotage his presidency by publishing damaging articles about him from inside his administration.
>
> You want to know the best thing President Trump said that day? "I won't stand up to Russia."
>
> That's not such a bad thing. Trump

Chess great Magnus Carlsen made history Tuesday as he extended his unbeaten run to 111 successive classical games, breaking the 110-game streak set by Sergei Tiviakov in 2005.

> Carlsen, who is ranked No. 1 on the world rankings, won a tense match against fellow Norwegian Vassily Ivanchuk of Russia at the World Youth Championship tournament held in Moscow, Russia. The 20-year-old has now broken the old mark three times and tied it once. He also broke the previous record, which was established when Vladimir Kramnik achieved 113 games last year.
>
> "I think I am just getting started," said Carlsen. "This will be my ninth year playing chess. I've been trying all these years but this time I'm really happy with how things are going."

A few weeks ago, Carlsen became only the second player since 1972 —after Anatoly Karpov —to reach 100 consecutive wins from one start in competition. But that didn't stop him from celebrating what would have been his 100th win Tuesday.

Bendable glass is the missing ingredient in today's foldable phones. But what if bendable screens were made from one of the hardest materials on Earth?

That's exactly what's being proposed by a team at Stanford University. The group has developed an extremely hard and lightweight material that they say would be ideal for making flexible displays. They call it "diamond" glass. It's not as strong or durable as carbon fiber, but it's still strong. In fact, this new composite will bend like paper when squeezed with your fingers. If you can make something out of diamond, then we might just have to start using these phones more often.

[Stanford's Technology Review]

Bibliography

Abadi, Martín et al. (2016a). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. arXiv: 1603.04467 [cs.DC].

Abadi, Martín et al. (2016b). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. arXiv: 1603.04467 [cs.DC].

Ackley, David H; Hinton, Geoffrey E, and Sejnowski, Terrence J (1985). "A learning algorithm for Boltzmann machines". In: *Cognitive science* 9.1, pp. 147–169.

Afify, Mohamed; Sarikaya, Ruhi; Kuo, Hong-Kwang Jeff; Besacier, Laurent, and Gao, Yuqing (2006). "On the use of morphological analysis for dialectal Arabic speech recognition." In: *INTERSPEECH*, pp. 277–280.

Akbik, Alan; Blythe, Duncan, and Vollgraf, Roland (Aug. 2018). "Contextual String Embeddings for Sequence Labeling". In: *Proceedings of the 27th International Conference on Computational Linguistics*. Santa Fe, New Mexico, USA: Association for Computational Linguistics, pp. 1638–1649. URL: https://www.aclweb.org/anthology/C18-1139.

Akbik, Alan; Bergmann, Tanja; Blythe, Duncan; Rasul, Kashif; Schweter, Stefan, and Vollgraf, Roland (June 2019). "FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 54–59. DOI: 10.18653/v1/N19-4010. URL: https://www.aclweb.org/anthology/N19-4010.

Akbik, Alan; Bergmann, Tanja, and Vollgraf, Roland (June 2019). "Pooled Contextualized Embeddings for Named Entity Recognition". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, pp. 724–728. DOI: 10.18653/v1/N19-1078. URL: https://www.aclweb.org/anthology/N19-1078.

Arora, Sanjeev; Li, Yuanzhi; Liang, Yingyu; Ma, Tengyu, and Risteski, Andrej (2016). "RANDWALK: A Latent Variable Model Approach to Word Embeddings". In: *Transactions of the Association for Computational Linguistics (TACL)* 4, pp. 385–399.

Baevski, Alexei; Edunov, Sergey; Liu, Yinhan; Zettlemoyer, Luke, and Auli, Michael (Nov. 2019). "Cloze-driven Pretraining of Self-attention Networks". In: *Proceedings of the 2019 Confer-*

ence on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). Hong Kong, China: Association for Computational Linguistics, pp. 5360–5369. DOI: 10.18653/v1/D19-1539. URL: https://www.aclweb.org/anthology/D19-1539.

Bahdanau, Dzmitry; Cho, Kyunghyun, and Bengio, Yoshua (2015). "Neural machine translation by jointly learning to align and translate". In: *International Conference on Learning Representations (ICLR 2015)*.

Bahl, Lalit R.; Jelinek, Fred, and Mercer, Robert (1983). "A Maximum Likelihood Approach to Continuous Speech Recognition". In: *IEEE Trans. PAMI* 5.2, pp. 179–190.

Baker, James (1975). "The DRAGON system–An overview". In: *IEEE Transactions on Acoustics, speech, and signal Processing* 23.1, pp. 24–29.

Balázs, Kapitány (2013). "Kárpát-medencei népszámlálási körkép". In: *Demográfia* 56.1, pp. 25–64.

Baroni, M.; Dinu, G., and Kruszewski, G. (2014). "Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors". In: *Proceedings of ACL 2014*, pp. 237–247.

Bartunov, Sergey; Kondrashkin, Dmitry; Osokin, Anton, and Vetrov, Dmitry (May 2016). "Breaking Sticks and Ambiguities with Adaptive Skip-gram". In: *Proceedings of Machine Learning Research* 51: Artificial Intelligence and Statistics, pp. 130–138.

Bengio, Yoshua; Simard, Patrice, and Frasconi, Paolo (1994). "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2, pp. 157–166.

Bengio, Yoshua; Ducharme, Réjean; Vincent, Pascal, and Janvin, Christian (2003). "A Neural Probabilistic Language Model". In: *Journal of Machine Learning Research* 3, pp. 1137–1155. URL: http://www.jmlr.org/papers/v3/bengio03a.html.

Bengio, Yoshua and Senécal, Jean-Sébastien (2003). "Quick Training of Probabilistic Neural Nets by Importance Sampling". In: *AISTATS*.

Bengio, Yoshua and Senécal, Jean-Sébastien (2008). "Adaptive importance sampling to accelerate training of a neural probabilistic language model". In: *IEEE Transactions on Neural Networks* 19.4, pp. 713–722.

Bengio, Yoshua (2012). "Practical recommendations for gradient-based training of deep architectures". In: *Neural networks: Tricks of the trade*. Springer, pp. 437–478.

Berger, A.L.; Pietra, S.A. Della, and Pietra, V.J. Della (1996). "A maximum entropy approach to natural language processing". In: *Computational Linguistics* 22.1.

Berger, Adam and Lafferty, John (1999). "Information Retrieval as Statistical Translation". In: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ' 99. Berkeley, California, USA: Association for Computing Machinery, pp. 222–229. ISBN: 1581130961. DOI: 10.1145/312624.312681. URL: https://doi.org/10.1145/312624.312681.

Biber, Douglas (1993). "Representativeness in corpus design". In: *Literary and linguistic computing* 8.4, pp. 243–257.

Bilmes, Jeff A. and Kirchhoff, Katrin (2003). "Factored Language Models and Generalized Parallel Backoff". In: *Proceedings of HLT/NACCL*, pp. 4–6.

Blei, David M; Ng, Andrew Y, and Jordan, Michael I (2003). "Latent dirichlet allocation". In: *Journal of machine Learning research* 3.Jan, pp. 993–1022.

Boguraev, Branimir K. and Briscoe, Edward J. (1989). *Computational Lexicography for Natural Language Processing*. Longman.

Bojanowski, Piotr; Grave, Edouard; Joulin, Armand, and Mikolov, Tomas (2017). "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* 5, pp. 135–146. ISSN: 2307-387X. URL: https://transacl.org/ojs/index.php/tacl/article/view/999.

Bojar, Ondřej; Federmann, Christian; Fishel, Mark; Graham, Yvette; Haddow, Barry; Koehn, Philipp, and Monz, Christof (Oct. 2018). "Findings of the 2018 Conference on Machine Translation (WMT18)". In: *Proceedings of the Third Conference on Machine Translation: Shared Task Papers*. Belgium, Brussels: Association for Computational Linguistics, pp. 272–303. DOI: 10.18653/v1/W18-6401. URL: https://www.aclweb.org/anthology/W18-6401.

Borbély, Gábor; Kornai, András; Nemeskey, Dávid, and Kracht, Marcus (2016). "Denoising composition in distributional semantics". In: *DSALT: Distributional Semantics and Linguistic Theory*. poster.

Borbély, Gábor; Makrai, Márton; Nemeskey, Dávid Márk, and Kornai, András (2016). "Evaluating multi-sense embeddings for semantic resolution monolingually and in word translation". In: *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*. Berlin, Germany: Association for Computational Linguistics, pp. 83–89. DOI: 10.18653/v1/W16-2515. URL: http://www.aclweb.org/anthology/W16-2515.

Botha, Jan A and Blunsom, Phil (2014). "Compositional Morphology for Word Representations and Language Modelling". In: *ICML*, pp. 1899–1907.

Brants, Thorsten; Popat, Ashok C.; Xu, Peng; Och, Franz J., and Dean, Jeffrey (June 2007). "Large Language Models in Machine Translation". In: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*. Prague, Czech Republic: Association for Computational Linguistics, pp. 858–867. URL: http://www.aclweb.org/anthology/D/D07/D07-1090.

Brébisson, Alexandre de and Vincent, Pascal (2015). "An Exploration of Softmax Alternatives Belonging to the Spherical Loss Family".

Breuel, Thomas M (1994). "A system for the off-line recognition of handwritten text". In: *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3-Conference C: Signal Processing (Cat. No. 94CH3440-5)*. Vol. 2. IEEE, pp. 129–134.

Broder, Andrei Z; Charikar, Moses; Frieze, Alan M, and Mitzenmacher, Michael (1998). "Min-wise independent permutations". In: *Proceedings of the thirtieth annual ACM symposium on Theory of Computing*. ACM, pp. 327–336.

Brown, P.F.; Pietra, V.J. Della; Souza, P.V. de; Lai, J.C., and Mercer, R.L. (1992). "Class–based n–gram models of natural language". In: *Computational Linguistics* 18.4, pp. 467–480.

Brown, Peter; Cocke, John; Pietra, Stephen Della; Pietra, Vincent J. Della; Jelinek, Fredrick; Lafferty, John D.; Mercer, Robert L., and Roossin, Paul S. (1990). "A statistical approach to machine translation". In: *Computational Linguistics* 16, pp. 79–85.

Brown, Peter F; Pietra, Vincent J Della; Pietra, Stephen A Della, and Mercer, Robert L (1993). "The mathematics of statistical machine translation: Parameter estimation". In: *Computational linguistics* 19.2, pp. 263–311.

Buck, Christian; Heafield, Kenneth, and Ooyen, Bas van (May 2014). "N-gram Counts and Language Models from the Common Crawl". In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA), pp. 3579–3584. URL: http://www.lrec-conf.org/proceedings/lrec2014/pdf/1097_Paper.pdf.

Cauchy, Augustin (1847). "Méthode générale pour la résolution des systemes d' équations simultanées". In: *Comp. Rend. Sci. Paris* 25.1847, pp. 536–538.

Charniak, Eugene (June 2001). "Immediate-Head Parsing for Language Models". In: *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*. Toulouse, France: Association for Computational Linguistics, pp. 124–131. URL: https://www.aclweb.org/anthology/P01-1017.

Che, Wanxiang; Liu, Yijia; Wang, Yuxuan; Zheng, Bo, and Liu, Ting (Oct. 2018). "Towards Better UD Parsing: Deep Contextualized Word Embeddings, Ensemble, and Treebank Concatenation". In: *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Brussels, Belgium: Association for Computational Linguistics, pp. 55–64. URL: http://www.aclweb.org/anthology/K18-2005.

Chelba, Ciprian and Jelinek, Frederick (Aug. 1998). "Exploiting Syntactic Structure for Language Modeling". In: *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 1*. Montreal, Quebec, Canada: Association for Computational Linguistics, pp. 225–231. DOI: 10.3115/980845.980882. URL: https://www.aclweb.org/anthology/P98-1035.

Chelba, Ciprian; Bikel, Dan; Shugrina, Maria; Nguyen, Patrick, and Kumar, Shankar (2012). *Large Scale Language Modeling in Automatic Speech Recognition*. Tech. rep. Google. URL: https://research.google.com/pubs/pub40491.html.

Chelba, Ciprian; Mikolov, Tomas; Schuster, Mike; Ge, Qi; Brants, Thorsten; Koehn, Phillipp, and Robinson, Tony (2014). "One billion word benchmark for measuring progress in statistical

language modeling". In: *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pp. 2635–2639.

Chen, Stanley; Beeferman, Douglas, and Rosenfeld, Ronald (1998). "Evaluation Metrics For Language Models". In: *Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, pp. 275–280.

Chen, Stanley F and Goodman, Joshua (Oct. 1999). "An empirical study of smoothing techniques for language modeling". In: *Computer Speech & Language* 13.4, pp. 359–394.

Chen, Stanley F. and Rosenfeld, Ronald (1999). *A Gaussian prior for smoothing maximum entropy models*. Tech. rep. CMU-CS-99-108. Computer Science Department, Carnegie Mellon University.

Chen, Wenlin; Grangier, David, and Auli, Michael (Aug. 2016). "Strategies for Training Large Vocabulary Neural Language Models". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1975–1985. URL: http://www.aclweb.org/anthology/P16-1186.

Chiu, Billy; Korhonen, Anna, and Pyysalo, Sampo (2016). "Intrinsic Evaluation of Word Vectors Fails to Predict Extrinsic Performance". In: *Proc. RepEval (this volume)*. Ed. by Omer Levy. ACL.

Cho, Kyunghyun; Merriënboer, Bart van; Gulcehre, Caglar; Bougares, Fethi; Schwenk, Holger, and Bengio, Yoshua (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar, pp. 1724–1734. URL: http://www.aclweb.org/anthology/D14-1179.

Chomsky, Noam (1957). *Syntactic Structures*. The Hague: Mouton.

Church, Kenneth W. and Hanks, Patrick (1990). "Word association norms, mutual information, and lexicography". In: *Computational Linguistics* 16.1, pp. 22–29.

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K., and Kuksa, P. (2011). "Natural Language Processing (Almost) from Scratch". In: *Journal of Machine Learning Research (JMLR)*.

Collobert, Ronan and Weston, Jason (2008). "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: ACM, pp. 160–167.

Conneau, Alexis and Lample, Guillaume (2019). "Cross-lingual Language Model Pretraining". In: *Advances in Neural Information Processing Systems*, pp. 7057–7067.

Conneau, Alexis et al. (2019). "Unsupervised cross-lingual representation learning at scale".

Corbí-Bellot, Antonio M.; Forcada, Mikel L.; Ortiz-Rojas, Sergio; Pérez-Ortiz, Juan Antonio; Ramírez-Sánchez, Gema; Sánchez-Martínez, Felipe; Alegria, Iñaki; Mayor, Aingeru, and Sara-

sola, Kepa (May 2005). “An open-source shallow-transfer machine translation engine for the romance languages of Spain”. In: *Proceedings of the Tenth Conference of the European Association for Machine Translation*. Budapest, Hungary, pp. 79–86.

Council of European Union (2016). “REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)”. In: *Official Journal* L 119, pp. 1–88. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679.

Csendes, Dóra; Hatvani, Csaba; Alexin, Zoltán; Csirik, János; Gyimóthy, Tibor; Prószéky, Gábor, and Váradi, Tamás (Dec. 2003). “Kézzel annotált magyar nyelvi korpusz: a Szeged Korpusz”. In: *I. Magyar Számítógépes Nyelvészeti Konferencia előadásai: MSZNY 2003*, pp. 238–245.

Csendes, Dóra; Csirik, János; Gyimóthy, Tibor, and Kocsor, András (2005). “The Szeged Treebank”. In: *Lecture Notes in Computer Science: Text, Speech and Dialogue*. Springer, pp. 123–131.

Cybenko, George (1989). “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of control, signals and systems* 2.4, pp. 303–314.

Dai, Zihang; Yang, Zhilin; Yang, Yiming; Carbonell, Jaime; Le, Quoc, and Salakhutdinov, Ruslan (July 2019). “Transformer-XL: Attentive Language Models beyond a Fixed-Length Context”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 2978–2988. DOI: 10.18653/v1/P19-1285. URL: https://www.aclweb.org/anthology/P19-1285.

Darroch, J.N. and Ratcliff, D. (1972). “Generalized iterative scaling for log-linear models”. In: *The Annals of Mathematical Statistics* 43, pp. 1470–1480.

Deerwester, Scott C.; Dumais, Susan T, and Harshman, Richard A. (1990). “Indexing by latent semantic analysis”. In: *Journal of the American Society for Information Science* 41.6, pp. 391–407.

Della Pietra, S.; Della Pietra, V.; Mercer, R. L., and Roukos, S. (1992). “Adaptive Language Modeling Using Minimum Discriminant Estimation”. In: *Speech and Natural Language: Proceedings of a Workshop Held at Harriman, New York, February 23-26, 1992*. URL: https://www.aclweb.org/anthology/H92-1020.

Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton, and Toutanova, Kristina (Oct. 11, 2018). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. Version 1. In: *arXiv preprint arXiv:1810.04805*. arXiv: 1810.04805v1 [cs.CL]. URL: http://arxiv.org/abs/1810.04805v1.

Devlin, Jacob; Chang, Ming-Wei; Lee, Kenton, and Toutanova, Kristina (2019). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proc. of NAACL*.

Dong, Li; Yang, Nan; Wang, Wenhui; Wei, Furu; Liu, Xiaodong; Wang, Yu; Gao, Jianfeng; Zhou, Ming, and Hon, Hsiao-Wuen (2019). “Unified language model pre-training for natural

language understanding and generation". In: *Advances in Neural Information Processing Systems*, pp. 13042–13054.

Dostert, Leon E (1955). "The georgetown-ibm experiment". In: *Machine Translation of Languages: Fourteen Essays*. MIT Press, pp. 124–135.

Dyer, Chris; Kuncoro, Adhiguna; Ballesteros, Miguel, and Smith, Noah A. (June 2016). "Recurrent Neural Network Grammars". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 199–209. arXiv: 1602.07776 [cs.CL]. URL: http://www.aclweb.org/anthology/N16-1024.

Emami, Ahmad and Jelinek, Frederick (2005). "Random Clusterings for Language Modeling". In: *ICASSP (1)*, pp. 581–584.

Endrédy, István and Indig, Balázs (2015). "HunTag3, a General-purpose, Modular Sequential Tagger – Chunking Phrases in English and Maximal NPs and NER for Hungarian". In: *7th Language & Technology Conference*. Poznan: Uniwersytet im. Adama Mickiewicza w Poznaniu, 213–218.

Faruqui, Manaal; Tsvetkov, Yulia; Rastogi, Pushpendre, and Dyer, Chris (2016). "Problems With Evaluation of Word Embeddings Using Word Similarity Tasks".

Filimonov, Denis and Harper, Mary (2009). "A joint language model with fine-grain syntactic tags". In: *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*. Association for Computational Linguistics, pp. 1114–1123.

Finkelstein, Lev; Gabrilovich, Evgeniy; Matias, Yossi; Rivlin, Ehud; Solan, Zach; Wolfman, Gadi, and Ruppin, Eytan (2002). "Placing Search in Context: The Concept Revisited". In: *ACM Transactions on Information Systems* 20(1), pp. 116–131.

Firth, John R. (1957). "A synopsis of linguistic theory". In: *Studies in linguistic analysis*. Blackwell, pp. 1–32.

Forcada, Mikel L; Ginestí-Rosell, Mireia; Nordfalk, Jacob; O' Regan, Jim; Ortiz-Rojas, Sergio; Pérez-Ortiz, Juan Antonio; Sánchez-Martínez, Felipe; Ramírez-Sánchez, Gema, and Tyers, Francis M (2011). "Apertium: a free/open-source platform for rule-based machine translation". In: *Machine translation* 25.2, pp. 127–144.

Francis, W Nelson and Kucera, Henry (1979). *Brown Corpus manual: Manual of information to accompany a standard corpus of present-day edited American English for use with digital computers*. Providence, Rhode Island, USA: Brown University.

Fukushima, Kunihiko (1980). "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological cybernetics* 36.4, pp. 193–202.

Gal, Yarin and Ghahramani, Zoubin (2016). "A Theoretically Grounded Application of Dropout in Recurrent Neural Networks". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee; M. Sugiyama; U. V. Luxburg; I. Guyon, and R. Garnett. Curran Associates,

Inc., pp. 1019–1027. arXiv: 1512.05287 [stat.ML]. URL: http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-recurrent-neural-networks.pdf.

Gers, Felix A; Schmidhuber, Jürgen, and Cummins, Fred (2000). "Learning to forget: Continual prediction with LSTM". In: *Neural computation* 12.10, pp. 2451–2471.

Gionis, Aristides; Indyk, Piotr; Motwani, Rajeev, et al. (1999). "Similarity search in high dimensions via hashing". In: *VLDB*. 6, pp. 518–529.

Gladkova, Anna and Drozd, Aleksandr (2016). "Intrinsic Evaluations of Word Embeddings: What Can We Do Better?" In: *Proc. RepEval (this volume)*. Ed. by Omer Levy. ACL.

Glorot, Xavier and Bengio, Yoshua (May 2010). "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html.

Gong, Chengyue; He, Di; Tan, Xu; Qin, Tao; Wang, Liwei, and Liu, Tie-Yan (2018). "FRAGE: Frequency-Agnostic Word Representation". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio; H. Wallach; H. Larochelle; K. Grauman; N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., pp. 1334–1345. URL: http://papers.nips.cc/paper/7408-frage-frequency-agnostic-word-representation.pdf.

Good, I.J. (1953). "The population frequencies of species and the estimation of population parameters". In: *Biometrika* 40, pp. 237–264.

Goodfellow, Ian; Bengio, Yoshua, and Courville, Aaron (2016). *Deep Learning*. MIT Press. URL: http://www.deeplearningbook.org.

Goodman, Joshua and Gao, Jianfeng (2000). "Language Model Size Reduction By Pruning And Clustering". In: *In ICSLP' 00*, pp. 110–113.

Goodman, Joshua T. (2001). "A Bit of Progress in Language Modeling". In: *Computer Speech & Language* 15.4, pp. 403–434.

Grave, Edouard; Joulin, Armand, and Usunier, Nicolas (2017). "Improving neural language models with a continuous cache". In: *International Conference on Learning Representations (ICLR 2017)*. URL: https://openreview.net/pdf?id=B184E5qee.

Grave, Edouard; Bojanowski, Piotr; Gupta, Prakhar; Joulin, Armand, and Mikolov, Tomas (2018). "Learning Word Vectors for 157 Languages". In: *Proc. of LREC*. URL: https://www.aclweb.org/anthology/L18-1550.

Graves, Alex and Schmidhuber, Jürgen (2009). "Offline handwriting recognition with multidimensional recurrent neural networks". In: *Advances in neural information processing systems*, pp. 545–552.

Graves, Alex (2012). "Sequence transduction with recurrent neural networks". In: *Representation Learning Workshop, ICML 2012*. Edinburgh, Scotland. arXiv: 1211.3711 [cs.NE].

Graves, Alex; Mohamed, Abdel-rahman, and Hinton, Geoffrey (2013). "Speech recognition with deep recurrent neural networks". In: *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, pp. 6645–6649.

Graves, Alex; Wayne, Greg, and Danihelka, Ivo (2014). *Neural turing machines*. Tech. rep. arXiv: 1410.5401 [cs.NE].

Greff, Klaus; Srivastava, Rupesh Kumar; Koutník, Jan; Steunebrink, Bas R, and Schmidhuber, Jürgen (2015). "LSTM: A search space odyssey". In: arXiv: 1503.04069 [cs.NE].

Gutmann, Michael and Hyvärinen, Aapo (2010). In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*.

Hahnloser, Richard HR; Sarpeshkar, Rahul; Mahowald, Misha A; Douglas, Rodney J, and Seung, H Sebastian (2000). "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". In: *Nature* 405.6789, pp. 947–951.

Halácsy, Péter; Kornai, András; Németh, László; Rung, András; Szakadát, István, and Trón, Viktor (2004). "Creating open language resources for Hungarian". In: *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004)*. ELRA, pp. 203–210.

Han, Lushan; Kashyap, Abhay L.; Finin, Tim; Mayfield, James, and Weese, Jonathan (June 2013). "UMBC_EBIQUITY-CORE: Semantic Textual Similarity Systems". In: *Second Joint Conference on Lexical and Computational Semantics (*SEM)*. Atlanta, Georgia, USA: Association for Computational Linguistics, pp. 44–52.

Harris, Zellig S. (1954). "Distributional structure". In: *Word* 10.23, pp. 146–162.

He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing, and Sun, Jian (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hewitt, John and Manning, Christopher D (2019). "A structural probe for finding syntax in word representations". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4129–4138.

Hill, Felix; Reichart, Roi, and Korhonen, Anna (2014). "Simlex-999: Evaluating semantic models with (genuine) similarity estimation". In: *Computational Linguistics* 41.4, pp. 665–695.

Hinton, Geoffrey; Vinyals, Oriol, and Dean, Jeff (2015). "Distilling the knowledge in a neural network".

Hirsimäki, Teemu; Creutz, Mathias; Siivola, Vesa, and Kurimo, Mikko (June 2005). "Morphologically Motivated Language Models in Speech Recognition". In: *Proceedings of AKRR'05, International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning*. Ed. by Timo Honkela; Ville Könönen; Matti Pöllä, and Olli Simula. Espoo, Finland: Helsinki University of Technology, Laboratory of Computer and Information Science, pp. 121–126. URL: http://www.cis.hut.fi/AKRR05/papers/akrr05tuulos.pdf.

Hochreiter, Sepp (1991). “Untersuchungen zu dynamischen neuronalen Netzen”. Diplomarbeit. Munich: Josef Hochreiter Institut für Informatik, Technische Universität München.

Hochreiter, Sepp and Schmidhuber, Jürgen (Nov. 1997). “Long Short-Term Memory”. In: *Neural Computation* 9.8, pp. 1735–1780.

Hornik, Kurt; Stinchcombe, Maxwell; White, Halbert, et al. (1989). “Multilayer feedforward networks are universal approximators.” In: *Neural networks* 2.5, pp. 359–366.

Huang, Eric; Socher, Richard; Manning, Christopher, and Ng, Andrew (2012). “Improving Word Representations via Global Context and Multiple Word Prototypes”. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (ACL 2012)*. Jeju Island, Korea: Association for Computational Linguistics, pp. 873–882.

Huang, Xuedong; Alleva, Fileno; Hon, Hsiao-Wuen; Hwang, Mei-Yuh; Lee, Kai-Fu, and Rosenfeld, Ronald (1993). “The SPHINX-II speech recognition system: an overview”. In: *Computer Speech & Language* 7.2, pp. 137–148. ISSN: 0885-2308. DOI: https://doi.org/10.1006/csla.1993.1007. URL: http://www.sciencedirect.com/science/article/pii/S0885230883710077.

Hutchins, John (May 1995). “”The whisky was invisible”, or Persistent myths of MT”. In: *MT News International* (11), pp. 32–34.

Inan, Hakan; Khosravi, Khashayar, and Socher, Richard (2017). “Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling”. In: *International Conference on Learning Representations (ICLR 2017)*. arXiv: 1611.01462 [cs.LG].

Indig, Balázs (2018). “Közös crawlnak is egy korpusz a vége – Korpuszépítés a CommonCrawl .hu domainjából”. In: *XIV. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2018)*. Ed. by Veronika Vincze. Szegedi Tudományegyetem Informatikai Intézet. Szeged: Szegedi Tudományegyetem Informatikai Tanszékcsoport, 125–134.

Indig, Balázs; Sass, Bálint; Simon, Eszter; Mittelholcz, Iván; Kundráth, Péter, and Vadász, Noémi (2019). “emtsv –Egy formátum mind felett [emtsv – One format to rule them all]”. In: *XV. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY 2019)*. Ed. by Gábor Berend; Gábor Gosztolya, and Veronika Vincze. Szegedi Tudományegyetem Informatikai Tanszékcsoport, pp. 235–247.

Indyk, Piotr and Motwani, Rajeev (1998). “Approximate nearest neighbors: towards removing the curse of dimensionality”. In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, pp. 604–613.

Ittzés, Nóra, ed. (2011). *A magyar nyelv nagyszótára III-IV*. Akadémiai Kiadó.

Jakubíček, Miloš; Kilgarriff, Adam; Kovář, Vojtěch; Rychlỳ, Pavel, and Suchomel, Vít (2013). “The tenten corpus family”. In: *7th International Corpus Linguistics Conference CL*, pp. 125–127.

Jarrett, Kevin; Kavukcuoglu, Koray; Ranzato, Marc'Aurelio, and LeCun, Yann (2009). "What is the best multi-stage architecture for object recognition?" In: *2009 IEEE 12th international conference on computer vision*. IEEE, pp. 2146–2153.

Jelinek, F.; Bahl, L.R., and Mercer, R.L. (1975). "Design of a linguistic statistical decoder for the recognition of continuous speech". In: *IEEE Transactions on Acoustics, Speech and Signal Processing* IT-21, pp. 250–256.

Jelinek, Frederick; Mercer, Robert L.; Bahl, Lalit R., and Baker, James K. (Nov. 1977). "Perplexity – a measure of the difficulty of speech recognition tasks". In: *Journal of the Acoustical Society of America* 62. Supplement 1, S63.

Jelinek, Frederick and Mercer, Robert (1980). "Interpolated estimation of Markov source parameters from sparse data". In: *Proceedings of the Workshop on Pattern Recognition in Practice*. Ed. by E. S. Geltsema and L. N. Kanal. Amsterdam: North-Holland.

Johansson, Stig; Leech, Geoffrey N, and Goodluck, Helen (1978). *Manual of information to accompany the Lancaster-Oslo/Bergen Corpus of British English, for use with digital computer*. Department of English, University of Oslo.

Jordan, Michael I. (May 1986). *Serial order: a parallel distributed processing approach*. Tech. rep. ICS 8604. San Diego, California: Institute for Cognitive Science, University of California.

Joshi, Mandar; Chen, Danqi; Liu, Yinhan; Weld, Daniel S; Zettlemoyer, Luke, and Levy, Omer (2020). "SpanBERT: Improving pre-training by representing and predicting spans". In: *Transactions of the Association for Computational Linguistics* 8, pp. 64–77.

Jozefowicz, Rafal; Zaremba, Wojciech, and Sutskever, Ilya (2015). "An empirical exploration of recurrent network architectures". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 2342–2350.

Jozefowicz, Rafal; Vinyals, Oriol; Schuster, Mike; Shazeer, Noam, and Wu, Yonghui (2016). "Exploring the limits of language modeling". In: *arXiv preprint arXiv:1602.02410*.

Jurafsky, Daniel and Martin, James H. (n.d.). *Speech and Language Processing*. 3rd edition. URL: https://web.stanford.edu/~jurafsky/slp3/.

Jurafsky, Daniel and Martin, James H. (2009). *Speech and Language Processing*. 2nd edition. Pearson.

Katz, S. (1987). "Estimation of probabilities from sparse data for the language model component of a speech recognizer". In: *IEEE Transactions on Acoustics, Speech and Signal processing* 35.3, pp. 400–401.

Keskar, Nitish Shirish; McCann, Bryan; Varshney, Lav R; Xiong, Caiming, and Socher, Richard (2019). "CTRL: A conditional transformer language model for controllable generation".

Kim, Yoon; Jernite, Yacine; Sontag, David, and Rush, Alexander M (2016). "Character-Aware Neural Language Models". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*. AAAI Press, pp. 2741–2749.

Kirchhoff, Katrin; Bilmes, Jeff, and Duh, Kevin (2008). *Factored Language Models Tutorial*. Tech. rep. UWEETR-2008-00048. https://www.ee.washington.edu/techsite/papers/refer/UWEETR-2008-0004.html. University of Washington, Dept. of EE.

Kitaev, Nikita; Kaiser, Łukasz, and Levskaya, Anselm (2020). "Reformer: The Efficient Transformer". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=rkgNKkHtvB.

Kneser, Reinhard and Ney, Hermann (1993). "Improved clustering techniques for class-based statistical language modelling". In: *Eurospeech*. Vol. 93, pp. 973–76.

Koehn, Philipp et al. (2007). "Moses: Open source toolkit for statistical machine translation". In: *Proceedings of the 45th annual meeting of the ACL*. Association for Computational Linguistics, pp. 177–180.

Kohlschütter, Christian; Fankhauser, Peter, and Nejdl, Wolfgang (2010). "Boilerplate detection using shallow text features". In: *Proceedings of the third ACM international conference on Web search and data mining*. ACM, pp. 441–450.

Kornai, András (1994). "Language models: where are the bottlenecks?" In: *AISB Quarterly* 88, pp. 36–40.

Kuchaiev, Oleksii and Ginsburg, Boris (2017). "Factorization tricks for LSTM networks". In: *International Conference on Learning Representations (ICLR 2017)*. arXiv: 1703.10722 [cs.CL]. URL: https://openreview.net/forum?id=ByxWXyNFg.

Kudo, Taku (July 2018). "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 66–75. DOI: 10.18653/v1/P18-1007. URL: https://www.aclweb.org/anthology/P18-1007.

Kuhn, Roland and De Mori, Renato (1990). "A cache-based natural language model for speech recognition". In: *IEEE transactions on pattern analysis and machine intelligence* 12.6, pp. 570–583.

Lan, Zhenzhong; Chen, Mingda; Goodman, Sebastian; Gimpel, Kevin; Sharma, Piyush, and Soricut, Radu (2019). "ALBERT: A lite bert for self-supervised learning of language representations".

Lau, Raymond; Rosenfeld, Ronald, and Roukos, Salim (1993). "Trigger-Based Language Models: A Maximum Entropy Approach". In: *Proceedings of the 1993 IEEE International Conference on Acoustics, Speech, and Signal Processing: Speech Processing - Volume II*. ICASSP '93. Minneapolis, Minnesota, USA: IEEE Computer Society, pp. 45–48. ISBN: 0780309464.

Le Cun, Yann (June 1989). *Generalization and Network Design Strategies*. Tech. rep. CRG-TR-89-4. Department of Computer Science, University of Toronto.

LeCun, Yann; Bottou, Léon; Bengio, Yoshua, and Haffner, Patrick (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324.

Leskovec, Jure; Rajaraman, Anand, and Ullman, Jeffrey David (2014). *Mining of Massive Datasets*. 2nd. USA: Cambridge University Press. ISBN: 1107077230.

Levesque, Hector J; Davis, Ernest, and Morgenstern, Leora (2011). "The Winograd schema challenge." In: *AAAI Spring Symposium: Logical Formalizations of Commonsense Reasoning*. Vol. 46, p. 47.

Levy, Omer and Goldberg, Yoav (June 2014a). "Dependency-Based Word Embeddings". In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 302–308. URL: http://www.aclweb.org/anthology/P14-2050.

Levy, Omer and Goldberg, Yoav (2014b). "Neural Word Embedding as Implicit Matrix Factorization". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani; M. Welling; C. Cortes; N.D. Lawrence, and K.Q. Weinberger, pp. 2177–2185.

Levy, Omer; Goldberg, Yoav, and Dagan, Ido (2015). "Improving Distributional Similarity with Lessons Learned from Word Embeddings". In: *Transactions of the Association for Computational Linguistics* 3, pp. 211–225.

Lewis, Mike; Liu, Yinhan; Goyal, Naman; Ghazvininejad, Marjan; Mohamed, Abdelrahman; Levy, Omer; Stoyanov, Ves, and Zettlemoyer, Luke (2019). "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension".

Li, Jiwei and Jurafsky, Dan (Sept. 2015). "Do Multi-Sense Embeddings Improve Natural Language Understanding?" In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1722–1732. DOI: 10.18653/v1/D15-1200. URL: http://www.aclweb.org/anthology/D15-1200.

Lidstone, G.J. (1920). "Note on the general case of the Bayes-Laplace formula for inductive or a posteriori probabilities". In:

Liu, Peter J.; Saleh, Mohammad; Pot, Etienne; Goodrich, Ben; Sepassi, Ryan; Kaiser, Lukasz, and Shazeer, Noam (2018). "Generating Wikipedia by Summarizing Long Sequences". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Hyg0vbWC-.

Liu, Yinhan et al. (2019). *Roberta: A robustly optimized bert pretraining approach*. arXiv: 1907.11692 [cs.CL].

Lui, Marco and Baldwin, Timothy (2012). "langid.py: An off-the-shelf language identification tool". In: *Proceedings of the ACL 2012 system demonstrations*. Association for Computational Linguistics, pp. 25–30.

Luong, Minh-Thang; Pham, Hieu, and Manning, Christopher D (2015a). "Bilingual Word Representations with Monolingual Quality in Mind". In: *Proceedings of NAACL-HLT*, pp. 151–159.

Luong, Thang; Pham, Hieu, and Manning, Christopher D. (2015b). "Effective Approaches to Attention-based Neural Machine Translation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, pp. 1412–1421. DOI: 10.18653/v1/D15-1166. URL: http://www.aclweb.org/anthology/D15-1166.

Manin, Dmitrii Y. (2008). "Zipf' s Law and Avoidance of Excessive Synonymy". In: *Cognitive Science* 32 (7), pp. 1075–1098.

Marcus, Mitchell; Santorini, Beatrice, and Marcinkiewicz, Mary Ann (1993). "Building a Large Annotated Corpus of English: The Penn Treebank". In: *Computational Linguistics* 19, pp. 313–330.

Markov, A. A. (1913). "Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain')". In: *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*. 6th ser. 7. English translation by Morris Halle, 1956., pp. 153–162.

Martin, Sven; Liermann, Jörg, and Ney, Hermann (1998). "Algorithms for bigram and trigram word clustering". In: *Speech communication* 24.1, pp. 19–37.

Melis, Gábor; Dyer, Chris, and Blunsom, Phil (2018). "On the State of the Art of Evaluation in Neural Language Models". In: *International Conference on Learning Representations (ICLR 2018)*. arXiv: 1707.05589 [cs.CL]. URL: https://openreview.net/forum?id=ByJHuTgA-.

Merity, Stephen; Xiong, Caiming; Bradbury, James, and Socher, Richard (2017). "Pointer Sentinel Mixture Models". In: *International Conference on Learning Representations (ICLR 2017)*. arXiv: 1609.07843 [cs.LG].

Merity, Stephen; McCann, Bryan, and Socher, Richard (2017). *Revisiting Activation Regularization for Language RNNs*. arXiv: 1708.01009 [cs.CL].

Merity, Stephen; Keskar, Nitish Shirish, and Socher, Richard (2018). "Regularizing and optimizing lstm language models". In: *International Conference on Learning Representations (ICLR 2018)*. arXiv: 1708.02182 [cs.LG].

Mihajlik, Péter; Tuske, Zoltán; Tarján, Balázs; Németh, Bottyán, and Fegyó, Tibor (2010). "Improved recognition of spontaneous Hungarian speech —Morphological and acoustic modeling techniques for a less resourced task". In: *IEEE Transactions on Audio, Speech, and Language Processing* 18.6, pp. 1588–1600.

Miháltz, Márton; Hatvani, Csaba; Kuti, Judit; Szarvas, György; Csirik, János; Prószéky, Gábor, and Váradi, Tamás (2008). "Methods and results of the Hungarian WordNet project". In: *Proceedings of the Fourth Global WordNet Conference (GWC-2008)*. Citeseer.

Miháltz, Márton (2010). "Semantic resources and their applications in Hungarian natural language processing". PhD thesis. Pázmány Péter Catholic University. URL: https://itk.ppke.hu/uploads/articles/163/file/Mihaltz_diss.pdf.

Mikolov, Tomas and Zweig, Geoffrey (2012). "Context dependent recurrent neural network language model". In: *SLT*, pp. 234–239.

Mikolov, Tomas (2012). "Statistical Language Models Based On Neural Networks". PhD thesis. Faculty of Information Technology, Brno University of Technology.

Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S, and Dean, Jeff (2013). "Distributed Representations of Words and Phrases and their Compositionality". In: *Advances in Neural Information Processing Systems 26*. Ed. by C.J.C. Burges; L. Bottou; M. Welling; Z. Ghahramani, and K.Q. Weinberger. Curran Associates, Inc., pp. 3111–3119. URL: https://bit.ly/39HikH8.

Mikolov, Tomas; Chen, Kai; Corrado, G.s., and Dean, Jeffrey (May 2013). "Efficient Estimation of Word Representations in Vector Space". In: *1st International Conference on Learning Representations, ICLR 2013, Workshop Track Proceedings*. Ed. by Y. Bengio and Y. LeCun. arXiv: 1301.3781 [cs.CL]. URL: http://arxiv.org/abs/1301.3781.

Mikolov, Tomas; Le, Quoc V, and Sutskever, Ilya (2013). "Exploiting similarities among languages for machine translation". arXiv preprint arXiv:1309.4168.

Mikolov, Tomas; Yih, Wen-tau, and Zweig, Geoffrey (2013). "Linguistic Regularities in Continuous Space Word Representations". In: *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT 2013)*. Atlanta, Georgia: Association for Computational Linguistics, pp. 746–751.

Mikolov, Tomáš (2010). *Recurrent neural network based language model*. Presentation at Google.

Mikolov, Tomáš; Kombrink, Stefan; Burget, Lukáš; Černocký, Jan, and Khudanpur, Sanjeev (2011). *Extensions of recurrent neural network language model*. Presentation at Google.

Mikolov, Tomáš; Deoras, Anoop; Povey, Daniel; Burget, Lukáš, and Černocký, Jan (2011). "Strategies for training large scale neural network language models". In: *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, pp. 196–201.

Miller, George A. (1995). "WordNet: a lexical database for English". In: *Communications of the ACM* 38.11, pp. 39–41.

Mimno, Davidan and Thompson, Laure (2017). "The strange geometry of skip-gram with negative sampling". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp. 2873–2878. DOI: 10.18653/v1/D17-1308. URL: http://aclweb.org/anthology/D17-1308.

Minsky, Marvin and Papert, Seymour (1969). "An introduction to computational geometry". In:

Mittelholcz, Iván (2017). "`emToken`: Unicode-képes tokenizáló magyar nyelvre". In: *XIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2017)*. Szeged, (this volume).

Mnih, Andriy and Teh, Yee Whye (2012). "A Fast and Simple Algorithm for Training Neural Probabilistic Language Models". In: *Proceedings of the 29th International Coference on Machine Learning*. ICML'12. Edinburgh, Scotland: Omnipress, pp. 419–426. ISBN: 9781450312851.

Moore, Robert; Appelt, Douglas; Dowding, John; Gawron, J. Mark, and Moran, Douglas (1995). "Combining Linguistic and Statistical Knowledge Sources in Natural-Language Processing for ATIS". In: *In ARPA Spoken Language Technology Workshop*.

Morin, Frederic and Bengio, Yoshua (2005). "Hierarchical Probabilistic Neural Network Language Model". In: *Aistats*. Vol. 5. Citeseer, pp. 246–252.

Mozer, Michael C (1992). "Induction of multiscale temporal structure". In: *Advances in neural information processing systems*, pp. 275–282.

Neelakantan, Arvind; Shankar, Jeevan; Passos, Alexandre, and McCallum, Andrew (2014). "Efficient Non-parametric Estimation of Multiple Embeddings per Word in Vector Space". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1059–1069. DOI: 10.3115/v1/D14-1113. URL: http://www.aclweb.org/anthology/D14-1113.

Nemeskey, Dávid Márk and Simon, Eszter (2012). "Automatically generated NE tagged corpora for English and Hungarian". In: *Proceedings of the 4th Named Entity Workshop*. Association for Computational Linguistics, pp. 38–46.

Nemeskey, Dávid Márk (2017). "`emLam` – a Hungarian Language Modeling baseline". In: *XIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2017)*. Szeged, pp. 91–102. arXiv: 1701.07880 [cs.CL].

Nemeskey, Dávid Márk (2020). "Egy `emBERT` próbáló feladat". In: *XVI. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2020)*. Szeged, pp. 409–418.

Németh, Bottyán; Mihajlik, Péter; Tikk, Domonkos, and Trón, Viktor (Nov. 2007). "Statisztikai és szabály alapú morfológiai elemzők kombinációja beszédfelismerő alkalmazáshoz". In: *Proceedings of MSZNY 2007*. Ed. by Attila Tanács and Dóra Csendes. Szegedi Tudományegyetem, pp. 95–105.

Ney, Hermann; Essen, Ute, and Kneser, Reinhard (1994). "On structuring probabilistic dependences in stochastic language modelling". In: *Comput. Speech Lang.* 8, pp. 1–38.

Ney, Hermann; Martin, Sven, and Wessel, Frank (1997). "Statistical language modeling using leaving-one-out". In: *Corpus-based methods in Language and Speech processing*. Springer, pp. 174–207.

Niesler, Thomas R; Whittaker, Edward WD, and Woodland, Philip C (1998). "Comparison of part-of-speech and automatically derived category-based language models for speech recognition". In: *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*. Vol. 1. IEEE, pp. 177–180.

Och, Franz Josef; Tillmann, Christoph, and Ney, Hermann (1999). "Improved Alignment Models for Statistical Machine Translation". In: *1999 Joint SIGDAT Conference on Empirical*

Methods in Natural Language Processing and Very Large Corpora, pp. 20–28. URL: https://www.aclweb.org/anthology/W99-0604.

Oravecz, Csaba; Váradi, Tamás, and Sass, Bálint (2014). "The Hungarian Gigaword Corpus". In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. Reykjavik, Iceland: European Language Resources Association (ELRA). URL: http://www.aclweb.org/anthology/L14-1536.

Ortiz Suárez, Pedro Javier; Sagot, Benoît, and Romary, Laurent (July 2019). "Asynchronous Pipeline for Processing Huge Corpora on Medium to Low Resource Infrastructures". In: *7th Workshop on the Challenges in the Management of Large Corpora (CMLC-7)*. Cardiff, United Kingdom. URL: https://hal.inria.fr/hal-02148693.

Panchenko, A; Ruppert, E; Faralli, S; Ponzetto, S.P, and Biemann, C (2018). "Building a Web-Scale Dependency-Parsed Corpus from Common Crawl". In: *Proceedings of LREC 2018*. ELRA.

Parra Escartín, Carla; Reijers, Wessel; Lynn, Teresa; Moorkens, Joss; Way, Andy, and Liu, Chao-Hong (Apr. 2017). "Ethical Considerations in NLP Shared Tasks". In: *Proceedings of the First ACL Workshop on Ethics in Natural Language Processing*. Valencia, Spain: Association for Computational Linguistics, pp. 66–73. DOI: 10.18653/v1/W17-1608. URL: https://www.aclweb.org/anthology/W17-1608.

Pascanu, Razvan; Mikolov, Tomas, and Bengio, Yoshua (2013). "On the difficulty of training recurrent neural networks". In: *ICML (3)* 28, pp. 1310–1318.

Paszke, Adam et al. (2017). "Automatic Differentiation in PyTorch". In: *NIPS Autodiff Workshop*.

Paszke, Adam et al. (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach; H. Larochelle; A. Beygelzimer; F. d'Alché-Buc; E. Fox, and R. Garnett. Curran Associates, Inc., pp. 8026–8037. URL: http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Pennington, Jeffrey; Socher, Richard, and Manning, Christopher (2014). "Glove: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: http://www.aclweb.org/anthology/D14-1162.

Pereira, Fernando; Tishby, Naftali, and Lee, Lillian (1993). "Distributional clustering of English words". In: *Proceedings of the 31st annual meeting on Association for Computational Linguistics*. Association for Computational Linguistics, pp. 183–190.

Peters, Matthew; Neumann, Mark; Iyyer, Mohit; Gardner, Matt; Clark, Christopher; Lee, Kenton, and Zettlemoyer, Luke (2018). "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans,

Louisiana: Association for Computational Linguistics, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. URL: http://aclweb.org/anthology/N18-1202.

Pomikálek, Jan (2011). "Removing boilerplate and duplicate content from web corpora". PhD thesis. Brno, Czech Republic: Faculty of informatics, Masaryk university.

Ponte, Jay M. and Croft, W. Bruce (1998). "A language modeling approach to information retrieval". In: *Proc SIGIR*. ACM Press, pp. 275–281.

Press, Ofir and Wolf, Lior (Apr. 2017). "Using the Output Embedding to Improve Language Models". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Valencia, Spain: Association for Computational Linguistics, pp. 157–163. arXiv: 1608.05859 [cs.CL]. URL: http://www.aclweb.org/anthology/E17-2025.

Pusztai, Ferenc, ed. (2003). *Magyar értelmező kéziszótár*. Akadémiai Kiadó.

Radford, Alec; Wu, Jeffrey; Amodei, Dario; Amodei, Daniela; Clark, Jack; Brundage, Miles, and Sutskever, Ilya (n.d.). *Better Language Models and Their Implications*. https://openai.com/blog/better-language-models/. Accessed: 2020-03-18.

Radford, Alec; Narasimhan, Karthik; Salimans, Tim, and Sutskever, Ilya (2018). "Improving language understanding by generative pre-training". https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf.

Radford, Alec; Wu, Jeffrey; Child, Rewon; Luan, David; Amodei, Dario, and Sutskever, Ilya (2019). "Language Models are Unsupervised Multitask Learners". https://github.com/openai/gpt-2. URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

Rajpurkar, Pranav; Zhang, Jian; Lopyrev, Konstantin, and Liang, Percy (Nov. 2016). "SQuAD: 100,000+ Questions for Machine Comprehension of Text". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, pp. 2383–2392. DOI: 10.18653/v1/D16-1264. URL: https://www.aclweb.org/anthology/D16-1264.

Rajpurkar, Pranav; Jia, Robin, and Liang, Percy (July 2018). "Know What You Don't Know: Unanswerable Questions for SQuAD". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Melbourne, Australia: Association for Computational Linguistics, pp. 784–789. DOI: 10.18653/v1/P18-2124. URL: https://www.aclweb.org/anthology/P18-2124.

Recski, Gábor (2010). "Főnévi csoportok azonosítása szabályalapú és hibrid módszerekkel". In: *VII. Magyar Számítógépes Nyelvészeti Konferencia*. Ed. by Attila Tanács and Veronika Vincze, pp. 333–341.

Recski, Gábor; Borbély, Gábor, and Bolevácz, Attila (2016). "Building definition graphs using monolingual dictionaries of Hungarian". In: *XI. Magyar Számítógépes Nyelvészeti Konferencia*

[11th Hungarian Conference on Computational Linguistics. Ed. by Attila Tanács; Viktor Varga, and Veronika Vincze.

Reisinger, Joseph and Mooney, Raymond J (2010). "Multi-prototype vector-space models of word meaning". In: *The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, pp. 109–117.

Robbins, Herbert and Monro, Sutton (1951). "A stochastic approximation method". In: *The annals of mathematical statistics*, pp. 400–407.

Rogers, Anna (June 2019). *How the Transformers broke NLP leaderboards*. URL: https://hackingsemantics.xyz/2019/leaderboards/.

Rosenblatt, Frank (1957). *The Perceptron: a perceiving and recognizing automaton*. Tech. rep. 85-460-1.

Rosenfeld, R. (1994). "Adaptive Statistical Language Modeling: A Maximum Entropy Approach". PhD thesis. Carnegie Mellon University.

Rosenfeld, Ronald (Aug. 2000). "Two decades of Statistical Language Modeling: Where Do We Go From Here?" In: *Proceedings of the IEEE* 88.8.

Rothe, Sascha; Ebert, Sebastian, and Schütze, Hinrich (June 2016). "Ultradense Word Embeddings by Orthogonal Transformation". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, pp. 767–777. arXiv: 1602.07572 [cs.CL]. URL: http://www.aclweb.org/anthology/N16-1091.

Rumelhart, David E; Hinton, Geoffrey E, and Williams, Ronald J (Sept. 1985). *Learning internal representations by error propagation*. Tech. rep. ICS 8504. San Diego, California: Institute for Cognitive Science, University of California.

Rumelhart, David E.; Hinton, Geoffrey E., and Williams, Ronald J (1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–536.

Sak, Hasim; Senior, Andrew W, and Beaufays, Françoise (2014). "Long short-term memory recurrent neural network architectures for large scale acoustic modeling." In: *INTERSPEECH*, pp. 338–342.

Sanh, Victor; Debut, Lysandre; Chaumond, Julien, and Wolf, Thomas (2019). "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter".

Schuster, Mike and Nakajima, Kaisuke (2012). "Japanese and korean voice search". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 5149–5152.

Schwenk, Holger and Gauvain, Jean-Luc (2005). "Training neural network language models on very large corpora". In: *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pp. 201–208.

Schwenk, Holger (July 2007). "Continuous Space Language Models". In: *Comput. Speech Lang.* 21.3, pp. 492–518. ISSN: 0885-2308. DOI: 10.1016/j.csl.2006.09.003. URL: http://dx.doi.org/10.1016/j.csl.2006.09.003.

Semeniuta, Stanislau; Severyn, Aliaksei, and Barth, Erhardt (2016). "Recurrent Dropout without Memory Loss". In: *The 26th International Conference on Computational Linguistics (COLING)*. Osaka, Japan, pp. 1757–1766. URL: http://www.aclweb.org/anthology/C16-1165.

Sennrich, Rico; Haddow, Barry, and Birch, Alexandra (Aug. 2016). "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Berlin, Germany: Association for Computational Linguistics, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. URL: https://www.aclweb.org/anthology/P16-1162.

Shannon, Claude E. (1948). "A Mathematical Theory of Communication". In: *Bell System Technical Journal* 27, pp. 379–423, 623–656.

Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529, pp. 484–503. URL: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

Sinclair, John M. (1987). *Looking up: an account of the COBUILD project in lexical computing*. Collins ELT.

Smolensky, Paul (1986). "Information Processing in Dynamical Systems: Foundations of Harmony Theory". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*. Ed. by James Lloyd McClelland and David Everett Rumelhart. Cambridge, MA, USA: MIT Press, pp. 194–281. ISBN: 026268053X.

Socher, Richard; Bauer, John; Manning, Christopher D., and Andrew Y., Ng (2013). "Parsing with Compositional Vector Grammars". In: *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (ACL 2013)*. Sofia, Bulgaria: Association for Computational Linguistics, pp. 455–465.

Srivastava, Nitish; Hinton, Geoffrey E; Krizhevsky, Alex; Sutskever, Ilya, and Salakhutdinov, Ruslan (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Srivastava, Rupesh K; Greff, Klaus, and Schmidhuber, Jürgen (2015). *Training very deep networks*, pp. 2377–2385.

Stolcke, Andreas; Zheng, Jing; Wang, Wen, and Abrash, Victor (2011). "SRILM at sixteen: Update and outlook". In: *Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop*. Vol. 5.

Suchomel, Vít; Pomikálek, Jan, et al. (2012). "Efficient web crawling for large text corpora". In: *Proceedings of the seventh Web as Corpus Workshop (WAC7)*, pp. 39–43.

Sundermeyer, Martin; Schlüter, Ralf, and Ney, Hermann (2012). "LSTM Neural Networks for Language Modeling". In: *INTERSPEECH*, pp. 194–197.

Sutskever, Ilya; Martens, James, and Hinton, Geoffrey E (2011). "Generating text with recurrent neural networks". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024.

Sutskever, Ilya; Vinyals, Oriol, and Le, Quoc V. (2014). "Sequence to Sequence Learning with Neural Networks". In: *Proc. NIPS*. Montreal, CA, pp. 3104–3112. URL: http://arxiv.org/abs/1409.3215.

Szarvas, György; Farkas, Richárd, and Kocsor, András (2006). "A Multilingual Named Entity Recognition System Using Boosting and C4.5 Decision Tree Learning Algorithms". In: *Discovery Science, 9th International Conference, DS 2006, Barcelona, Spain, October 8-10, 2006, Proceedings*, pp. 268–278.

Tarján, Balázs; Varga, Ádám; Tobler, Zoltán; Szaszák, György; Fegyó, Tibor; Bordás, Csaba, and Mihajlik, Péter (2016). "Magyar nyelvű, élő közéleti- és hírműsorok gépi feliratozása". In: *Proc. MSZNY 2016*. Ed. by Attila Tanács; Viktor Varga, and Veronika Vincze. Szegedi Tudományegyetem, pp. 89–99.

Taylor, Wilson L. (1953). " "Cloze Procedure" : A New Tool for Measuring Readability". In: *Journalism Quarterly* 30.4, pp. 415–433. DOI: 10.1177/107769905303000401.

Tesniére, Lucien (1959). *Élements de syntaxe structurale*. Paris: Klincksieck.

Tiedemann, Jörg (May 2012). "Parallel Data, Tools and Interfaces in OPUS". In: *LREC*. Ed. by Nicoletta Calzolari. Istanbul, Turkey: European Language Resources Association (ELRA). ISBN: 978-2-9517408-7-7. URL: http://www.lrec-conf.org/proceedings/lrec2012/summaries/463.html.

Trinh, Trieu H and Le, Quoc V (2018). *A simple method for commonsense reasoning*. arXiv: 1806.02847 [cs.AI].

Trón, Viktor; Gyepesi, György; Halácsky, Péter; Kornai, András; Németh, László, and Varga, Dániel (2005). "Hunmorph: Open Source Word Analysis". In: *Proceedings of the ACL Workshop on Software*. Ann Arbor, Michigan: Association for Computational Linguistics, pp. 77–85.

Turian, Joseph; Ratinov, Lev-Arie, and Bengio, Yoshua (2010). "Word Representations: A Simple and General Method for Semi-Supervised Learning". In: *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Uppsala, Sweden: Association for Computational Linguistics, pp. 384–394.

Ugray, Gábor (2019). "PoS-tagging and lemmatization with a deep recurrent neural network". In: *XV. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2019)*. Szeged, pp. 215–224.

Váradi, Tamás (2002). "The Hungarian National Corpus". In: *Proceedings of the Third International Conference on Language Resources and Evaluation*, pp. 385–389.

Váradi, Tamás et al. (2017). "`e-magyar`: digitális nyelvfeldolgozó rendszer". In: *XIII. Magyar Számítógépes Nyelvészeti Konferencia (MSZNY2017)*. Szeged.

Varga, Dániel and Simon, Eszter (Feb. 2007). "Hungarian Named Entity Recognition with a Maximum Entropy Approach". In: *Acta Cybern.* 18.2, pp. 293–301.

Vaswani, Ashish; Shazeer, Noam; Parmar, Niki; Uszkoreit, Jakob; Jones, Llion; Gomez, Aidan N; Kaiser, Łukasz, and Polosukhin, Illia (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon; U. V. Luxburg; S. Bengio; H. Wallach; R. Fergus; S. Vishwanathan, and R. Garnett. Curran Associates, Inc., pp. 5998–6008. arXiv: 1706.03762 [cs.CL]. URL: http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

Vincent, Pascal; Brébisson, Alexandre de, and Bouthillier, Xavier (2015). "Efficient exact gradient update for training deep networks with very large sparse targets". In: *Advances in Neural Information Processing Systems*, pp. 1108–1116.

Vincze, Veronika; Varga, Viktor; Simkó, Katalin Ilona; Zsibrita, János; Nagy, Ágoston; Farkas, Richárd, and Csirik, János (May 2014). "Szeged Corpus 2.5: Morphological Modifications in a Manually POS-tagged Hungarian Corpus". English. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Ed. by Nicoletta Calzolari (Conference Chair); Khalid Choukri; Thierry Declerck; Hrafn Loftsson; Bente Maegaard; Joseph Mariani; Asuncion Moreno; Jan Odijk, and Stelios Piperidis. Reykjavik, Iceland: European Language Resources Association (ELRA). ISBN: 978-2-9517408-8-4.

Vinyals, Oriol; Fortunato, Meire, and Jaitly, Navdeep (2015). "Pointer networks". In: *Advances in Neural Information Processing Systems*, pp. 2692–2700.

Voita, Elena; Talbot, David; Moiseev, Fedor; Sennrich, Rico, and Titov, Ivan (July 2019). "Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy: Association for Computational Linguistics, pp. 5797–5808. DOI: 10.18653/v1/P19-1580. URL: https://www.aclweb.org/anthology/P19-1580.

Wang, Alex; Singh, Amanpreet; Michael, Julian; Hill, Felix; Levy, Omer, and Bowman, Samuel R (2018). *Glue: A multi-task benchmark and analysis platform for natural language understanding*. arXiv: 1804.07461 [cs.CL].

Wendlandt, Laura; Kummerfeld, Jonathan K., and Mihalcea, Rada (June 2018). "Factors Influencing the Surprising Instability of Word Embeddings". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, pp. 2092–2102. DOI: 10.18653/v1/N18-1190. URL: https://www.aclweb.org/anthology/N18-1190.

Werbos, Paul J (1988). "Generalization of backpropagation with application to a recurrent gas market model". In: *Neural networks* 1.4, pp. 339–356.

Weston, Jason; Chopra, Sumit, and Bordes, Antoine (2015). "Memory networks". In: *International Conference on Learning Representations (ICLR 2015)*. arXiv: 1410.3916 [cs.CL].

Williams, Ronald J and Peng, Jing (1990). "An efficient gradient-based algorithm for on-line training of recurrent network trajectories". In: *Neural computation* 2.4, pp. 490–501.

Williams, Ronald J and Zipser, David (Feb. 1995). "Gradient-based learning algorithms for recurrent networks and their computational complexity". In: *Back-propagation: Theory, architectures and applications*. Ed. by Yves Chauvin and David E Rumelhart. Chap. 13, pp. 433–486. ISBN: 978-0-8058125-9-6.

Wolf, Thomas et al. (2019). *HuggingFace's Transformers: State-of-the-art Natural Language Processing*. arXiv: 1910.03771 [cs.CL].

Xu, Wei and Rudnicky, Alexander I (2000). "Can artificial neural networks learn language models?" In: *International Conference on Statistical Language Processing*. Beijing, China, pp. 202–205.

Yang, Zhilin; Dai, Zihang; Salakhutdinov, Ruslan, and Cohen, William W. (2018). "Breaking The Softmax Bottleneck: A High-Rank RNN Language Model". In: *International Conference on Learning Representations (ICLR 2018)*. arXiv: 1711.03953 [cs.LG].

Yang, Zhilin; Dai, Zihang; Yang, Yiming; Carbonell, Jaime; Salakhutdinov, Ruslan, and Le, Quoc V (2019). "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *Advances in neural information processing systems*, pp. 5754–5764. arXiv: 1906.08237 [cs.CL]. URL: https://papers.nips.cc/paper/8812-xlnet-generalized-autoregressive-pretraining-for-language-understanding.pdf.

Youn, Hyejin; Sutton, Logan; Smith, Eric; Moore, Cristopher; Wilkins, Jon F.; Maddieson, Ian; Croft, William, and Bhattacharya, Tanmoy (2016). "On the universal structure of human lexical semantics". In: *PNAS* 113.7, pp. 1766–1771.

Younger, Daniel H. (1967). "Recognition and Parsing of Context-Free Languages in Time n^3". In: *Information and Control* 10, pp. 189–208.

Zaremba, Wojciech; Sutskever, Ilya, and Vinyals, Oriol (2014). "Recurrent neural network regularization". If you want to cite this paper, please cite Pham:2014 instead.

Zeiler, Matthew D and Fergus, Rob (2014). "Visualizing and understanding convolutional networks". In: *European conference on computer vision*. Springer, pp. 818–833.

Zeman, Daniel et al. (Aug. 2017). "CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies". In: *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Vancouver, Canada: Association for Computational Linguistics, pp. 1–19.

Zgusta, Ladislav (1971). *Manual of lexicography*. Prague: Academia.

Zilly, Julian Georg; Srivastava, Rupesh Kumar; Koutník, Jan, and Schmidhuber, Jürgen (2017). "Recurrent Highway Networks". In: *Proceedings of the 34th International Conference on Machine Learning*. arXiv: 1607.03474 [cs.LG].

Zoph, Barret and Le, Quoc (2017). “Neural Architecture Search with Reinforcement Learning”. In: *International Conference on Learning Representations (ICLR 2017)*. URL: https://openreview.net/forum?id=r1Ue8Hcxg.

Zsibrita, János; Vincze, Veronika, and Farkas, Richárd (2013). “magyarlanc: A Tool for Morphological and Dependency Parsing of Hungarian”. In: *Proceedings of the International Conference Recent Advances in Natural Language Processing (RANLP 2013)*. Hissar, Bulgaria: INCOMA Ltd. Shoumen, pp. 763–771.

[1]**ADATLAP**
a doktori értekezés nyilvánosságra hozatalához

I. A doktori értekezés adatai
A szerző neve: **Nemeskey Dávid Márk**
MTMT-azonosító: **10019809**
A doktori értekezés címe és alcíme: **Natural Language Processing Methods for Language Modeling**
DOI-azonosító[2]: **10.15476/ELTE.2020.066**
A doktori iskola neve: **Informatika Doktori iskola**
A doktori iskolán belüli doktori program neve: **Az informatika alapjai és módszertana**
A témavezető neve és tudományos fokozata: **Benczúr András Ph.D. , Kornai András D.Sc.**
A témavezető munkahelye: **Számítástechnikai és Automatizálási Kutatóintézet**

II. Nyilatkozatok
1. A doktori értekezés szerzőjeként[3]

a) hozzájárulok, hogy a doktori fokozat megszerzését követően a doktori értekezésem és a tézisek nyilvánosságra kerüljenek az ELTE Digitális Intézményi Tudástárban. Felhatalmazom az Informatika Doktori Iskola hivatalának ügyintézőjét, Kulcsár Adinát, hogy az értekezést és a téziseket feltöltse az ELTE Digitális Intézményi Tudástárba, és ennek során kitöltse a feltöltéshez szükséges nyilatkozatokat.

b) kérem, hogy a mellékelt kérelemben részletezett szabadalmi, illetőleg oltalmi bejelentés közzétételéig a doktori értekezést ne bocsássák nyilvánosságra az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástárban;[4]

c) kérem, hogy a nemzetbiztonsági okból minősített adatot tartalmazó doktori értekezést a minősítés (*dátum*)-ig tartó időtartama alatt ne bocsássák nyilvánosságra az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástárban;[5]

d) kérem, hogy a mű kiadására vonatkozó mellékelt kiadó szerződésre tekintettel a doktori értekezést a könyv megjelenéséig ne bocsássák nyilvánosságra az Egyetemi Könyvtárban, és az ELTE Digitális Intézményi Tudástárban csak a könyv bibliográfiai adatait tegyék közzé. Ha a könyv a fokozatszerzést követőn egy évig nem jelenik meg, hozzájárulok, hogy a doktori értekezésem és a tézisek nyilvánosságra kerüljenek az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástárban.[6]

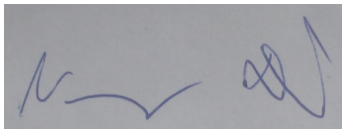
2. A doktori értekezés szerzőjeként kijelentem, hogy

a) az ELTE Digitális Intézményi Tudástárba feltöltendő doktori értekezés és a tézisek saját eredeti, önálló szellemi munkám és legjobb tudomásom szerint nem sértem vele senki szerzői jogait;

b) a doktori értekezés és a tézisek nyomtatott változatai és az elektronikus adathordozón benyújtott tartalmak (szöveg és ábrák) mindenben megegyeznek.

3. A doktori értekezés szerzőjeként hozzájárulok a doktori értekezés és a tézisek szövegének plágiumkereső adatbázisba helyezéséhez és plágiumellenőrző vizsgálatok lefuttatásához.

Kelt: **Budaörs, 2020. május 11.**



Nemeskey Dávid Márk

---

[1] Beiktatta az Egyetemi Doktori Szabályzat módosításáról szóló CXXXIX/2014. (VI. 30.) Szen. sz. határozat. Hatályos: 2014. VII.1. napjától.
[2] A kari hivatal ügyintézője tölti ki.
[3] A megfelelő szöveg aláhúzandó.
[4] A doktori értekezés benyújtásával egyidejűleg be kell adni a tudományági doktori tanácshoz a szabadalmi, illetőleg oltalmi bejelentést tanúsító okiratot és a nyilvánosságra hozatal elhalasztása iránti kérelmet.
[5] A doktori értekezés benyújtásával egyidejűleg be kell nyújtani a minősített adatra vonatkozó közokiratot.
[6] A doktori értekezés benyújtásával egyidejűleg be kell nyújtani a mű kiadásáról szóló kiadói szerződést.