Integrating Encyclopedic Knowledge into Neural Network Language Models

Bachelor's Thesis of

Yang Zhang

at the Department of Informatics
Institute for Anthropomatics and Robotics (IAR)
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany

School of Computer Science
Interactive Systems Labs (ISL)
Carnegie Mellon University (CMU)
Pittsburgh, United States

Advisor:	Dr. Jan Niehues, Karlsruhe Institute of Technology - Karlsruhe, Germany
Reviewer:	Prof. Dr. Alexander Waibel, Carnegie Mellon University - Pittsburgh, USA
Second Reviewer:	Prof. Dr. Tamim Asfour, Karlsruhe Institute of Technology - Karlsruhe, Germany

Duration: July 1, 2016 – October 31, 2016

Abstract

Machine translation deals with translation done by computers and has been an ongoing research topic for decades. Recently, phrase-based machine translation was revolutionized by significant improvements of neural models. Recurrent language models, in particular, have been a great success due to their ability to model arbitrary long context. However, one serious drawback of neural models is their high demand for large training data for good performance, which is not always available for less common languages. In this work, we investigate the integration of global semantic information extracted from large independent encyclopedia sources into neural network language models. We integrate semantic word classes extracted from Wikipedia and sentence level topic information into an RNN-based language model. The new resulting models exhibit great potential in alleviating data sparsity problems with the additional knowledge provided. This approach of integrating global information is not restricted to language modeling but can also be easily applied to models that profit from context or further data resources, e.g. neural machine translation.

Acknowledgments

I wish to thank my advisor Dr. Jan Niehues for his support and guidance on my bachelor's thesis. I would also like to thank Professor Waibel and the CLICS exchange Program for offering me the unique opportunity to write my thesis at the Interactive Systems Lab at CMU in Pittsburgh.

Lots of thanks to my colleagues and friends at CMU, in particular Professor Metze, Susi Burger, Jae Cho, and Eric Riebling for taking me warmly into the group and giving me lots of valuable advices.

Last but not least, I am very grateful to my family and friends who encouraged and believed in me, especially my wonderful mother and my dearest friend Johannes Bader.

Contents

1 Introduction

1.1 Introduction

Statistical machine translation (SMT) is a technology to automatically translate text from one language into another, using probabilistic models induced from a parallel text corpus that was previously translated by humans. Generally speaking, a statistical machine translator is composed of three parts: a translation model, that provides a mapping of words or phrases between two languages; a language model, that checks the fluency of the target language; and a decoder, that consults the translation and language model to find the best translation to a given sentence.

One focus to improve SMT has been to improve language models. Language modeling is primarily approached by the use of n-gram language models, which predict the next word in a sentence in the context of its preceding words. Since then, a wide range of statistical language models have been proposed that are based on n-gram models. However, they all have the same dilemma caused by data sparsity: trading larger context information for more reliable statistics.

Recently, with the increasing popularity of neural networks these have been employed successfully in language modeling. Recurrent neural network language models (RNNLMs), in particular, have shown great improvement in SMT, both during decoding and rescoring. The use of continuous word representations has achieved better generalization of the data which effectively lowered data sparsity problems. Furthermore, the recurrent connections are able to model long range dependencies. Yet, most of these models strictly depend on monolingual and parallel data, which is sometimes not available in large amounts or only for specific domains, especially with low-resource languages. This has motivated neural network language models that take multiple parallel streams of data as input instead of just a single stream of surface words. These so-called factors can be used to add additional information, e.g. part-of-speech (POS) or clustered word classes, which is mainly helpful for morphologically rich languages (e.g. Romanian, German). However, so far the additional factors were only limited to syntactic or local context information around the current word. Especially for languages without sufficient training data it is important to take advantage of other knowledge sources, e.g. encyclopedia knowledge. For example, Wikipedia has become a growing source for learning general concepts since the emergence of the Internet has led to an explosion of textual data. These data sources give insights into a variety of human endeavors waiting to be computationally analyzed.

In this paper, we studied the integration of large encyclopedic knowledge into recurrent neural network-based language models by using two approaches. In the first approach we used a factored model to integrate Wikipedia categories as factors.

In order to understand large unstructured datasets great achievements have been attained in latent concept learning in the area of text mining. Techniques include categorization of documents using latent semantic analysis and probabilistic topic modeling. In this work, we used techniques like term frequency–inverse document frequency (tf-idf), latent semantic analysis (LSA) and latent dirichlet allocation (LDA) to compute a real-valued topic vector for each sentence that is fed into the network as additional input.

These approaches utilize general word categories and global topic features in combination with local contexts implicitly provided by recurrent neural models to improve lexical selection.

We show that this has led to an improvement over the baseline system, that uses recurrent neural network based language models for rescoring, tested on English-Chinese and English-Romanian translation systems.

- Chapter 2 gives an overview of SMT with focus on language modeling. It describes the different components of a SMT system, particularly the phrase-based and log-linear models that we used as our experimental framework, together with various evaluation metrics.
- Chapter 3 gives a short overview of work that has been done related to incorporating additional information into neural language models and introduces previous work on topic modeling.
- Chapter 4 proposes our two approaches to integrate encyclopedic information into neural language models.
- Chapter 5 presents and explains experimental results.
- Chapter 6 discusses the implications of the experimental results, and suggests future research directions.

2 Background

2.1 Statistical Machine Translation

SMT translates a source sentence of one language into a target sentence of another language, utilizing only statistical methods generated from the analysis of parallel text corpora with aligned sentences. SMT requires huge amounts of data and large computing power. These parallel text data are often obtained from web crawling and text extraction. In training, statistical models are generated, that extract word and phrase translations from the parallel corpus. Once they are trained, the models can yield different candidate translations, referred to as translation hypotheses, for a given test sentence and finally choose the one with the highest probability. The fundamental problem of SMT can be described with Bayes' theorem

$$P(\mathbf{t}|\mathbf{s}) = \frac{P(\mathbf{s}|\mathbf{t}) * P(\mathbf{t})}{P(\mathbf{s})}, \tag{2.1}$$

with $\mathbf{s}$ denoting the source sentence and $\mathbf{t}$ the target sentence. $P(\mathbf{t}|\mathbf{s})$ is the probability of translating $\mathbf{s}$ into $\mathbf{t}$, which is influenced by the independent probabilities of the sentences $\mathbf{s}$ and $\mathbf{t}$ and the probability that $\mathbf{t}$ is a translation of $\mathbf{s}$. The goal of decoding is to find

$$\tilde{\mathbf{t}} = \arg\max_{\mathbf{t}} P(\mathbf{t}|\mathbf{s}) = \frac{P(\mathbf{s}|\mathbf{t}) * P(\mathbf{t})}{P(\mathbf{s})} \tag{2.2}$$

which is equivalent to

$$\tilde{\mathbf{t}} = \arg\max_{\mathbf{t}} P(\mathbf{t}|\mathbf{s}) = P(\mathbf{s}|\mathbf{t}) * P(\mathbf{t}) \tag{2.3}$$

$P(\mathbf{s}|\mathbf{t})$ is realized by the translation model, $P(\mathbf{t})$ by the language model.

2.1.1 Language Model

To model the fluency and language accuracy of a sentence $\mathbf{w}$ we use a language model $P(\mathbf{w})$. For example, the translation model on its own would probably translate "peanut butter" as "花生黄油", which is the result if we looked up both words separately. However, a language model would remember from previous training data that "butter" in the context of "peanut" has another meaning, and would translate it correctly with "花生酱", which literally translates to "peanut paste". The most common way to model contextual information is to use an n-gram model, which uses the Markov assumption to predict the next words in consideration of the last $n-1$ words. Let's take a look at a sentence $\mathbf{w} = w_1, w_2, \ldots, w_m$ and its probability when using a trigram language model, that is an n-gram model with $n = 3$.

$$\begin{aligned} p_{LM}(\mathbf{w}) &= p(w_1, w_2, \ldots, w_m) \\ &= p(w_1)p(w_2|w_1)\cdots p(w_m|w_1, w_2, \ldots, w_{m-1}) \\ &\simeq p(w_1)p(w_2|w_1)\cdots p(w_m|w_{m-2}, w_{m-1}) \end{aligned} \tag{2.4}$$

i		**know**	
Translation	Probability $p(t\|s)$	Translation	Probability $p(t\|s)$
我	0.8	知道	0.4
余	0.08	认识	0.2
吾	0.07	知	0.2
朕	0.05	懂得	0.2

Figure 2.1: Translation tables

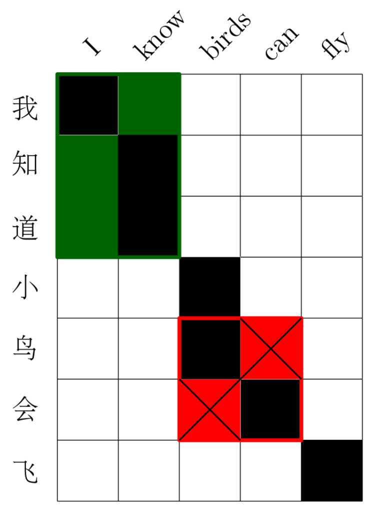

Figure 2.2: Word alignment matrix. The region with x marks is an example of a wrong phrase.

where the trigram occurrence frequencies are gathered statistics from training data

$$p(w_i|w_{i-2}, w_{i-1}) = \frac{count(w_{i-2}, w_{i-1}, w_i)}{\sum_w count(w_{i-2}, w_{i-1}, w)} \tag{2.5}$$

2.1.2 Translation Model and Word Alignment

The translation model provides the mapping of translation units, e.g. words or phrases, from one language into another along with the translation probability, as shown in Figure 2.1.

The translation probability distribution induces a word alignment which is often illustrated using a word alignment matrix A, as shown in Figure 2.2.

The problem is that in the beginning neither the translation probability distribution nor the word alignment is given. But knowing one of them is enough to deduce the other. The expectation maximization (EM) algorithm solves this dilemma.

2.1.3 Phrase Translation Model and Reordering

The translation model so far is based on an alignment model with words as its units. This can be a problem when a word translates to several words or none at all. In this case, a model based on phrases, i.e. successive words, as its smallest translation units is usually a better alternative to comprise more contextual information. Note that phrases in SMT are not restricted to linguistic phrases. However, sequence of words are only considered

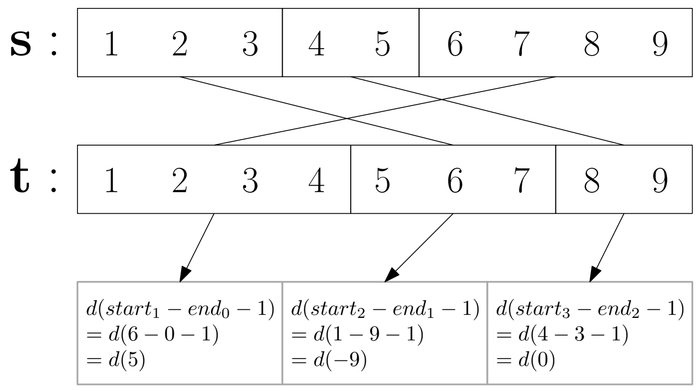

Figure 2.3: Distance-based reordering

phrases if they conform to the previously acquired word alignment. A phrase pair $(\bar{s}, \bar{t})$ with $\bar{s} = s_1, s_2, .., s_m$ and $\bar{t} = t_1, t_2, .., t_n$ is said to be consistent with an alignment A, iff

$$\forall t_i \in \bar{t} : (t_i, s_j) \in A \Rightarrow s_j \in \bar{s} \tag{2.6}$$

$$\forall s_j \in \bar{s} : (t_i, s_j) \in A \Rightarrow t_i \in \bar{t} \tag{2.7}$$

$$\exists t_i \in \bar{t}, s_j \in \bar{s} : (t_i, s_j) \in A \tag{2.8}$$

In other words, words within the phrase pair cannot be also aligned to any neighboring words outside of the phrase. In the example shown in Figure 2.2, (I know, 我知道) would be a valid phrase pair since it is consistent with the alignment. However, (birds can, 鸟会) is not a SMT phrase pair since is does not comply with Formula 2.7.

After decomposing a sentence into phrases, the translated phrases often need to be rearranged to fit the structure of the target language. The phrase-based translation model can be described as

$$P(\mathbf{s}|\mathbf{t}) = \prod_{i=1}^{I} \phi(\bar{s_i}|\bar{t_i}) d(start_i - end_{i-1} - 1) \tag{2.9}$$

where the target sentence $\mathbf{t}$ is broken up into I phrases, where each phrase $\bar{t_i}$ is aligned to the source phrase $\bar{s_i}$ along with the phrase translation probability $\phi(\bar{s_i}|\bar{t_i})$.

d is a distance-based reordering model, that penalizes reordered phrase pairs. Given the aligned phrase pair $(\bar{s_i}, \bar{t_i})$, the reordering distance is computed as $start_i = end_{i-1} - 1$. $start_i$ defines the first word's position of $\bar{s_i}$ and end_i last word's position of $\bar{s_i}$. An example is shown in Figure 2.3.

The probability distribution over extracted phrase pairs $(\bar{s}|\bar{t})$ can simply be determined using maximum likelihood

$$\phi(\bar{s}|\bar{t}) = \frac{count(\bar{t}, \bar{s})}{\sum_{s_i} count(\bar{t}, \bar{s_i})} \tag{2.10}$$

Some languages vary greatly in their syntactic sentence structure. This reordering model could put too much penalty on correct translations. In this case, it is better to use models that facilitate bigger reordering distances, such as tree-based reordering models.

2.1.4 Log Linear Model

So far the phrase-based model consists of three components: the phrase translation table $\phi(\bar{s}|\bar{t})$, the reordering model d and the language model $p_{LM}(t)$. We can then rewrite

Formula 2.3 as

$$\tilde{\mathbf{t}} = \arg\max_{\mathbf{t}} \prod_{i=1}^{I} \phi(\bar{s}_i|\bar{t}_i) d(start_i - end_{i-1} - 1) \prod_{i=1}^{|\mathbf{t}|} p_{LM}(t_i|t_1...t_{i-1}) \tag{2.11}$$

Sometimes a SMT system does not yield very fluent sentences based on Formula 2.11, which can be influenced by the language model. To give the language model more weight Formula 2.11 can be generalized by assigning individual weights to each of the components:

$$\tilde{\mathbf{t}} = \arg\max_{\mathbf{t}} \prod_{i=1}^{I} \phi(\bar{s}_i|\bar{t}_i)^{\lambda_\phi} d(start_i - end_{i-1} - 1)^{\lambda_d} \prod_{i=1}^{|\mathbf{t}|} p_{LM}(t_i|t_1...t_{i-1})^{\lambda_{LM}} \tag{2.12}$$

A big drawback of building a model on Formula 2.12 lies in its costly computation. Also, adding or removing model scores is not easily done due to heavy multiplications. A common solution is to transform it into log-linear form, which decomposes the original model into several weighted feature components, that can be trained separately. The log-linear model holds the following form:

$$p(x) = \exp(\sum_{i=1}^{n} \lambda_i h_i(x)) \tag{2.13}$$

To convert the phrase-based translation model 2.12 into this form, the following assignments are needed:

$$\begin{aligned} \text{number of components } n &= 3, \\ \text{random variable } x &= (\mathbf{t}, \mathbf{s}, start, end), \\ \text{feature functions } h_1 &= \log \phi, h_2 = \log d, h_3 = \log p_{LM}, \\ \text{feature weights } \lambda_1 &= \lambda_\phi, \lambda_2 = \lambda_d, \lambda_3 = \lambda_{LM}. \end{aligned}$$

This way, to find the best translation $\tilde{\mathbf{t}}$ a simpler formula can be solved instead:

$$\tilde{\mathbf{t}} = \arg\max_{\mathbf{t}} P(\mathbf{s}|\mathbf{t}) = \arg\max_{\mathbf{t}} \exp(\sum_{i=1}^{n} \lambda_i h_i(x)) = \arg\max_{\mathbf{t}} \sum_{i=1}^{n} \lambda_i h_i(x) \tag{2.14}$$

Another big advantage of Formula 2.14 is that the features $h_i(x)$ in this form do not necessarily need to be probability distributions. Model weights can be adjusted without too much effort. For example, increasing λ_ϕ will lead to more accurate translations, increasing λ_d will lead to less reordered translations, and increasing λ_{LM} will result in more fluent sentences.

2.1.5 Decoding

Given an input sentence, the goal of decoding is to find the best translation with the highest score according to the underlying model features. Simply translating the source sentence phrase-by-phrase, picking out the highest-rated translation from the translation table neither yields in fluent language nor reflects the meaning of the original sentence. Instead, creating multiple hypotheses that consider various translations of the same phrase at multiple different positions in the output sentence increase the chance of generating the most-probable translation. For this, the output sentence is incrementally built from left to right, which yields multiple partial hypotheses. With each newly added phrase the corresponding partial score is determined by consulting the model features using the log-linear model, and added to the overall score. The multiple ways to expand a hypothesis lead to an exploding number of hypotheses that grows exponentially in the length of the input sentence for two main reasons: 1. There are several options to pick the next phrase. 2. A certain phrase can, yet again, be translated in multiple ways according to the phrase translation table. 3. Phrases can be reordered.

For this reason, searching through all possible hypotheses to find the optimal translation is infeasible. Hence, in practice heuristic methods are employed, that skip parts of the search space efficiently, hopefully without omitting the most promising candidate translations.

If a translator fails to find the best translation, either the decoding or the model are to blame. A decoding error occurs when the search algorithm fails to find the most-probable translation due to its heuristic nature. In case of a model error the best translation does not get the highest score based on the model and, thus, gets discarded in decoding.

The first step to restrict the search space is to use hypothesis recombination, that merges all partial hypotheses with the same words translated so far, and takes the score of the best hypothesis. This way, only one hypothesis needs to be extended in the future. However, hypothesis recombination is not enough to cut down the computational complexity from exponential to polynomial. For this, we introduce two heuristic methods.

2.1.5.1 Beam-Search Stack Decoding

First, hypotheses can be organized into stacks according to their number of translated words. The worst hypotheses are dropped according to their partial score when the stack becomes too large. This is called histogram pruning. In addition, a fixed threshold can cut off hypotheses that are worse than the current best hypothesis by this factor. This is known as threshold pruning. Together, they constitute the beam-search stack decoding which has a polynomial computational complexity of

$$O(\text{max stack size} \times \text{sentence length}^2) \tag{2.15}$$

2.1.5.2 A* Search

A drawback of beam search is that future improvements of a path are not taken into account when hypotheses are dropped. Therefore, it runs the risk of mistakingly dropping a hypothesis that may turn out to be the best later on. While the partial score $g(n)$ of a path n alone can be misleading, adding an estimated future cost $h(n)$ for the remaining path may be a more realistic criteria for pruning. The heuristic cost function $f(n) := g(n) + h(n)$ is guaranteed to find the best path if it is admissible, that is if $f(n)$ never overestimates the actual cost. One example of an admissible heuristic function is $g(n)$ since it is always below the actual cost of a completed path. Depth-first search can be used to expand the next hypothesis according to the heuristic function. After each expansion the estimated score for the newly added phrase is overridden by the real cost.

2.1.6 Evaluation

Fluency and adequacy determine translation quality and play a major role in the evaluation of a SMT system. Difficulties include inherent linguistic and non-linguistic problems, such as lexical and structural ambiguities as well as stylistic differences between the source and target language. The advantages of automatic evaluation against human translation are low-cost, language-independence and normalized judgment due to standardized metrics. The basic idea of an automatic evaluation system is to determine the similarity between the generated translations and the reference translations that were previously translated by humans. The challenge is to come up with a good similarity scoring that reflects the correlation between human translations and automatic translations. Some of the important metrics are introduced in the following.

2.1.6.1 Precision and Recall

Precision is a very simple metric that counts the number of correct words in the output sentence. A word is correct if it also appears in the reference translation.

$$\text{precision} = \frac{\text{correct}}{\text{output-length}} \tag{2.16}$$

$$\text{recall} = \frac{\text{correct}}{\text{reference-length}} \tag{2.17}$$

Unlike precision, recall considers omitted words in a sentence. Both metrics, however, ignore the original word order.

2.1.6.2 WER, PER, TER

Word error rate (WER) [30] is adopted from speech recognition and takes word order into account. Based on the Levenshtein distance, it uses operations such as “insertion” (ins), “deletion” (del), and “substitution” (sub) to find the distance between the output translation and the reference.

$$\text{WER} = \frac{\#\text{ins} + \#\text{del} + \#\text{sub}}{\text{reference-length}} \tag{2.18}$$

A problem of WER is that the exact same order of words in hypothesis and translation are required for a low error. A grammatically correct sentence with a slightly varied structure can have a high WER. For example, the hypothesis “On a sunny day, they like to drink lemonade” of the given reference “They like to drink lemonade on a sunny day” results in a 100% WER.
Position-independent word error rate (POS) [43] neglects word order completely.
Translation error rate (TER) [40] also employs the idea of editing distance, but uses an additional operation that swaps segments of words to allow varied word order in translations.

2.1.6.3 BLEU and NIST

Bilingual evaluation understudy (BLEU) [35] is an automated and inexpensive evaluation metric most commonly used in SMT. The main idea is to measure the frequency with which the n-grams in the output appear in the reference translation, which is known as n-gram precision. For this reason, BLEU is considered a precision-based metric. To prevent words from being dropped, BLEU penalizes outputs that are shorter than the reference. Since a lot of n-grams do not occur in the reference sentence, BLEU is computed over the entire test set and is therefore considered a document-level metric. BLEU is defined as:

$$\text{BLEU-}n = \text{BP} \cdot \exp\left(\sum_{i=1}^{n} \lambda_i \log p_i\right) \tag{2.19}$$

where n is the maximum order of n-grams which is considered for statistics, BP is the brevity penalty for too short output, and p_i is the n-gram precision. Usually, baseline systems use 4-gram and uniform weights $\lambda_i = \frac{1}{n}$:

$$\text{BLEU-4} = \text{BP} \cdot \prod_{i=1}^{4} p_i \tag{2.20}$$

NIST [15] is a variation of BLEU, that assigns different weights λ_i to n-grams according to their rarity and co-occurrence statistics. For example, "it is" is given a lower weight than "excessive consumption".

2.1.6.4 METEOR

METEOR [7] is a more recent evaluation metric, that considers stemming and synonymy. That is, if output and reference do not match to the exact word, their word stem or even word class will be taking in account, utilizing WordNet [1]. This leads to more human-like translations. However, one of the drawbacks is that METEOR is much more complicated to apply and so far has been only reliably employed for English translations due to required background knowledge.

2.1.7 Parameter Tuning and Rescoring

Dicriminative training eliminates translation errors by directly maximizing the posterior probability $P(\mathbf{t}|\mathbf{s})$ rather than using the argmax model according to Formula 2.3. Parameter tuning optimizes model parameters in the log-linear model directly against the underlying evaluation metric, usually BLEU. Since evaluating Formula 2.14 is expensive, the idea is to carry out the optimization on a development or tuning set that is smaller than the training set and similar to the actual test set in the hope that if the model performs well on the development set it also does on the test set. Rescoring can be used for better translation results and is performed on an extracted n-best list. It allows reranking of the top hypotheses by using more sophisticated models, that are too expensive to be applied on the complete data. These complex models can comprise multiple features that can be easily integrated thanks to the log-linear model. A variety of additional features can be found in [33].

2.1.7.1 MERT

Minimal error rate training (MERT) [32] is a batch tuning algorithm used for rescoring and parameter tuning. The tuning process follows the following steps:

1. Initialize model parameters.
2. Decoder generates an n-best list based on model parameters.
3. Optimize parameters according to the evaluation metric.
4. Unless the parameters converge apply changes to parameters and repeat from step 2.
5. Finished.

[1] https://wordnet.princeton.edu/

To speed up tuning in MERT, it is common to use the heuristic of Powell's search [36] with smart step size, that varies one parameter at a time to find an optimum for each feature dimension. One problem of this method is that it does not scale well with many features. The number of features should not exceed 20.

2.1.7.2 ListNet

The ListNet-based n-best list rescoring method [27] uses the ListNet algorithm [12] which minimizes the difference between the log-linear model ranking and the reference ranking with respect to the underlying BLEU evaluation metric. For this it uses the cross entropy loss function that considers the whole n-best list during learning instead of single pairs of entries. Unlike MERT it scales well with many features. The idea is to define two probability distributions over hypotheses that determine the likelihood of a hypothesis to be ranked first in list, one for the hypothesized ranking and one for the reference ranking. Then the algorithm minimizes the Kullback-Leibler divergence between the distributions to optimize the model parameters.

2.2 Language Models

One problem of n-gram models is data sparsity. Any n-grams not seen in training would be assigned a probability of zero in testing if their empirical count is zero according to Formula 2.5. This is unwanted behavior since, although a lot of n-grams in testing will probably not appear in training, we still want to be able to predict sentences containing these n-grams. Therefore, strings should never be given a probability of zero. The data sparsity problem becomes more severe the bigger n becomes, which conflicts with the original intention to model more contextual information for more fluent translation. To solve this dilemma smoothing techniques and back-off models can be employed, which avoid probabilities of zero by taking probability mass from observed n-grams and adding it to unseen ones.

2.2.1 Factored Language Model

Factored language models (FLMs) [8] generalize n-gram model by substituting a word with a bundle of features, e.g. a word's POS, surface form or stem. This allows additional information to be incorporated into an FLM, which offers the option to layer linguistic information on top of the original n-gram model. In an FLM, a word w_i is considered a vector of K features f_i, referred to as factors.

$$w_i \equiv f_i^1, f_i^2, \dots, f_i^K = f_i^{1:K} \tag{2.21}$$

The FLM probability over a sentence $\mathbf{w} = w_1 w_2 \dots w_m$ is described as

$$P(w_1, w_2, \dots, w_m) = P(f_1^{1:K}, f_2^{1:K}, \dots, f_m^{1:K}) = P(f_{1:m}^{1:K}) \tag{2.22}$$

Analogous to an n-gram model this can be further transformed into a product of probabilities of the form $P(f|f_1, f_2, \dots, f_M)$ like in Formula 2.4. Also, a variation of back-off can be used for the statistical model. However, unlike a word-only model factors do not necessarily exhibit temporal sequencing, so dropping the oldest factor in the history to back off may not make any sense. For example, factors, such as word classes, provide information about the whole sentence and, therefore, should be given equal weight in back-off. Various options of back-off for FLMs are proposed by [8], which can be altogether depicted in a back-off graph.

2.2.2 Perplexity

The premise for a language model is to assign a higher probability to good language and a lower score to bad language. One way to evaluate language models is to use perplexity, which is interpreted as the "branching factor" of a language. Another interpretation is to regard perplexity as the number of words a language model chooses uniformly and randomly from a set of words to predict the next word. Perplexity is based on the distribution's entropy, which is defined as

$$\begin{aligned} H(p_{LM}) &= -\frac{1}{n}\log p_{LM}(w_1, w_2, \dots w_n) \\ &= -\frac{1}{n}\sum_{i=1}^{n}\log p_{LM}(w_i|w_1, w_2, \dots w_{i-1}) \end{aligned} \tag{2.23}$$

Then, perplexity is defined as a transformation of the distribution's entropy.

$$PP = 2^{H(p_{LM})} \tag{2.24}$$

The better a language model, the higher the probability of a well-formed sentence and the lower the perplexity of a sentence.

2.3 Neural Networks

An artificial neural network can be described as a weighted graph, consisting of an input layer of the size of the number of input features, one or multiple hidden layers, and an output layer with each of its nodes corresponding to a label in supervised learning. Generally speaking, a neural network is a nested composite function where the computation is carried out in distributed nodes. The final result is displayed at the output layer. A feed forward neural network, as shown in Figure 2.4, channels the input straight through the network, never touching the same node twice. Therefore, the information of an input is lost after it is processed, and each input is treated completely independent from any other.

The nodes of an m-layered feed forward neural network are computed as follows:

$$o_i^j = \begin{cases} f_1(\sum_{k=1}^{n} w_{k,i}^1 \cdot x_k + \theta_i^1) & \text{if } j = 1 \\ g(\sum_{k=1}^{n} w_{k,i}^m \cdot o_k^{m-1} + \theta_i^m) = y_i & \text{if } j = m \\ f_j(\sum_{k=1}^{n} w_{k,i}^j \cdot o_k^{j-1} + \theta_i^j) & \text{else} \end{cases} \tag{2.25}$$

where x_k denotes the k-th entry of the input, $w_{k,i}^j$ the weight from the k-th node in layer $j-1$ to the i-th node in layer j, θ_i^j the bias of the i-th node in layer j, f_j an activation function, such as sigmoid or tanh, and g the softmax function that generates a probability distribution over the output vocabulary. The output value y_i represents the probability of label i corresponding to the input x.

Backpropagation is a method used in conjunction with an optimization method to train a network and comprises two steps. In a forward pass, the results are generated at the output layer and the error E is calculated according to a certain cost function, e.g cross entropy. In a backward pass, the error is propagated backward through the layers and the contribution of a network's weight w to a reduction of error E is represented by its gradient:

$$\frac{\partial E}{\partial w} \tag{2.26}$$

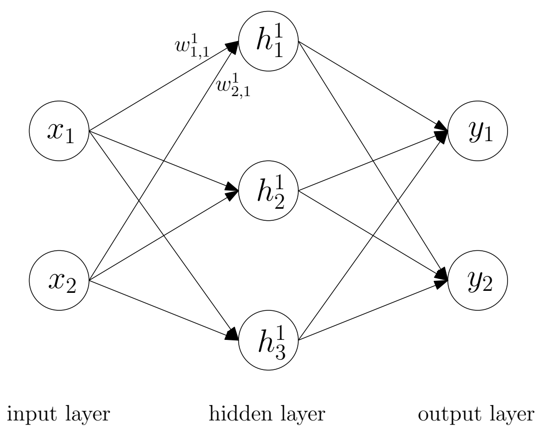

Figure 2.4: Feed forward neural network

Gradient descent is one way to optimize the model based on backpropagation. Given a learning rate η, the weight w is adjusted as follows:

$$w_{new} = w - \eta \cdot \frac{\partial E}{\partial w} \tag{2.27}$$

Usually the training algorithm iterates multiple times (often 20-50 times) over the training data, referred to as epochs, and uses validation data for early stopping as well as controlling of the learning rate. Weights are initialized to small random values. After each epoch, the network is tested on the validation data to see if the likelihood has improved. If not, the learning rate is decreased for the next epoch. Training is finished if learning converges or is stopped.

2.3.1 Recurrent Neural Network

Recurrent neural networks are optimized neural network for sequence learning where the previous output will influence the next prediction. This is realized by a feed-back loop that directs outputs back to the beginning of a hidden layer, thus retaining the sequential information in the network's hidden state. The new input in conjunction with the former hidden state determines the next output. Examples of recurrent neural networks in natural language processing (NLP) include machine translation, speech recognition and hand writing. In this work, we used recurrent neural networks for language modeling.

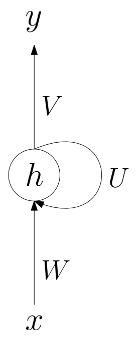

Figure 2.5: Recurrent neural network

Given a simple recurrent neural network with an input layer x and an output layer y. Let x_t denote the input vector, y_t the output, and h_t the hidden state or context vector at time t which has a recurrent connection with the time-delayed context vector h_{t-1}. Then the layers are calculated as follows:

$$\begin{aligned} h_t &= f(Wx_t + Uh_{t-1})) \\ y_t &= g(Vh_t) \end{aligned} \tag{2.28}$$

where f is the activation function, g the softmax function, and U, V, W are parameters to be learned in training. h_0 is usually initialized to zero.

2.3.1.1 Back-Propagation Through Time

To train a recurrent neural network an extension of backpropagation, known as back-propagation through time (BPTT), is used to treat the feed-back loop. The original backpropagation cannot be applied to the network right away since the recurrent connection's gradient V cannot be explicitly determined. If the training data can be represented as an ordered sequence of input-output pairs, for example $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$, BPTT is able to unfold the recurrent neural network into a feed-forward network through time. Here, time is simply expressed by a well-defined, ordered series of calculations linking one time step to the next, which is the prerequisite for backpropagation to work. In the above example there are m time steps. Let's illustrate this using an example of a recurrent neural network with one recurrent layer and one feed forward layer, as shown in Figure 2.5.

The unfolded network substitutes the recurrent component with m feed forward components. For example, if $m = 3$ the unfolded network looks like Figure 2.6.

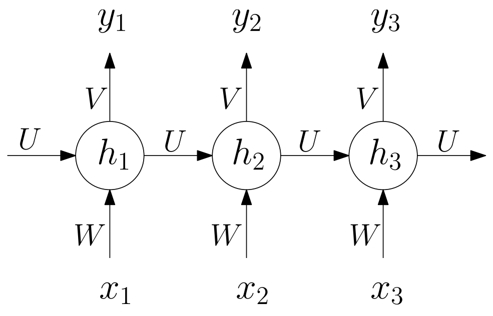

Figure 2.6: Unfolded recurrent neural ntwork

After transforming the recurrent neural network into an equivalent feed forward neural network, the previously discussed backpropagation can now be used to calculate the derivatives. Unlike regular feed forward neural networks all of the components corresponding to the same recurrent connection must share the same weights in the unfolded network after each iteration, as indicated in Figure 2.6. To achieve this, the weights' derivatives are summed up, which becomes the new weight of the corresponding recurrent connection. One problem is that m can get very large. But if m increases, the gradients at early time steps decay quickly. As a consequence, the first part of the training sequence hardly influences the outcome at the output layer. This problem is known as the vanishing gradient problem [19] and entails that a basic recurrent neural network cannot model long-term dependencies.

2.3.1.2 Long Short-Term Memory

Long short-term memory (LSTM) [20] is a recurrent neural network architecture which suffers less from the vanishing gradient problem and is able to learn long-term dependencies. The key idea of an LSTM is to preserve a constant error over multiple time steps which causes the gradient to reach further back in time. This is realized by utilizing a cell state that can incorporate input over a span of steps, but can also forget history. This is controlled by three gates: a forget gate to reset the cell state; an input state to preserve information about the current input in the cell state; and an output gate that decides what to output depending on the cell state and the input. This way, BPTT is able to consider more than 1000 time steps.

3 Related Work

3.1 Additional Information in Neural Language Models

Language models are a critical component of machine translation, yet they have always faced the problem of data sparsity. When FLMs (Section 2.2.1) were developed, they outperformed previous n-gram models without expanding the training data due to additional information. Compared to FLMs and feed forward neural networks, recurrent neural networks do not rely on backing off to shorter context nor require a fixed context length. In addition, recurrent neural networks have a lower model complexity. This inspired neural language models that incorporate additional information to model longer contexts and counter data sparsity problems.

3.1.1 Factored Recurrent Neural Network Language Models

Inspired by RNNLMs and FLMs, factored recurrent neural network language models (FRNNLMs) [44] can be considered as a generalization of RNNLMs. Like FLMs they use a factored input layer that takes a bundle of factors which can be used to integrate additional linguistic information. For this reason, FRNNLMs are better in dealing with data sparsity in morphologically rich languages [44] and exploiting commonalities and specialties among diverse data, upon which reordering or grammatical coherence decisions are made. If a word's surface form is the only factor used, the model becomes an ordinary RNNLM. Let's take a look at the example phrase "difference between developed countries and developing countries". Whereas a RNNLM would treat the bigrams "developed countries" and "developing countries" independent of each other, a FRNNLM with stem features would recognize "develop countri" as their common stem and therefore establish a connection between the two phrases. Let's assume we need to evaluate "developing countries" in testing, but this phrase has never been seen in training. If we have already observed "developed countries" and therefore its stem "develop countri", a FRNNLM is capable of giving the unseen phrase a higher and, in this example, a more realistic score than a regular RNNLM.

The work of [44] proposes a FRNNLM that predicts $P(w_i|f_{i-1}, s_{i-1})$, that is the next word w_i based on the previous source factor f_{i-1} and hidden state s_{i-1}, as opposed to the conventional RNNLM that predicts $P(w_i|w_{i-1}, s_{i-1})$. Their model uses a structured output layer based on word classes which is able to handle large vocabularies. The factors used include a word's surface form, stem and POS. They claim that POS yields better results than stem by explaining that the similarity between a word's stem and surface form does not add too much information to the network.

Motivated by multi-task learning in NLP, the paper [28] proposes a multi-factor recurrent neural language model which jointly predicts different output factors by mapping the output of the LSTM-layer to as many softmax layers as there are output factors. The model architecture is shown in Figure 3.1. The model generates one probability distribution for each output factor. In the rescoring of an n-best list, this model can be included in the log-linear model as either one or several additional features depending on whether the output is treated as a joint probability or individual probabilities. Due to its design, this model can also be used as a bilingual model. In the paper, they used the conjunction of the previous target word's and the current source word's factor set to predict the next target word. This way, the model computes the translation probability rather than the language model probability. On the source side the word's surface form and POS are used as factors; on the target side the word's surface form, POS and clustered word classes are used as factors.

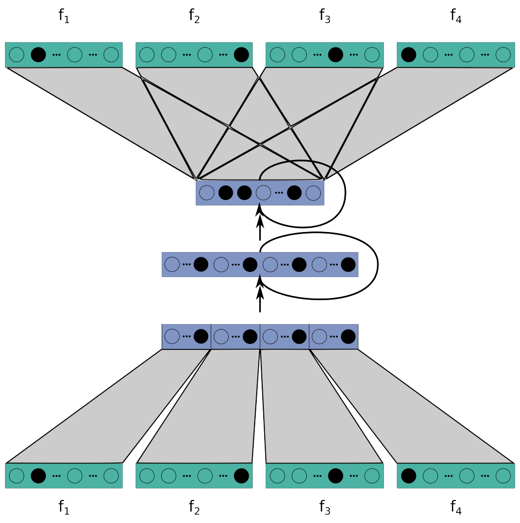

Figure 3.1: Multi-factor recurrent neural network-based language model. The figure is taken from the paper [28].

3.1.2 Other RNNLM Variations

So far only factors have been associated with each word. However, using a dedicated continuous space vector would offer more flexibility and possibility to store additional side information, especially for complex higher-level concepts, such as topic information. In the paper [26] a first approach to use an additional vector for higher-level concepts was proposed. In their work, a vector instead of a factor is associated with each word and connected to both the hidden and output layer. However, this vector depends only on a word's earlier local context, thus neglecting the influence of the future context on a word's meaning. In fact, often the meaning of a word cannot be just derived from its preceding words but by content words in the entire sentence or surrounding sentences. Also, the additional information is extracted from the same corpus. Therefore, this information will not be enough to learn knowledge beyond the given training data.

Since RNNLMs are too expensive to use during decoding, all of the mentioned variations of RNNLMs are mainly used in rescoring, their probabilities interpolated with n-gram probabilities.

3.2 Concept Learning and Wikipedia

Topic models play a great role in text mining because they summarize large amounts of documents into fewer concepts by capturing word co-occurrence information. Essentially, topic models can be divided into vector space models, e.g. LSA [14], and probability models, e.g. LDA [9]. The successful usage of Wikipedia to devise methods for computing semantic relatedness of documents was reported in [16] and [42]. Generative probabilistic models were employed in [17] to link named entities in text documents by using information extracted from Wikipedia. In the work of [13] and [11] vector space models were employed to resolve word disambiguations based on entities derived from Wikipedia. Similar approaches were applied on other knowledge sources, such as the semantic network WordNet [18].

However, to our knowledge, it is the first time encyclopedic knowledge and neural language models are used together.

3.2.1 TF-IDF

Tf-idf [38] is a co-occurrence measure and determines the importance of a word to one or multiple documents. Having the capability of grading down a term appearing in multiple documents, tf-idf is computed by multiplying a local component (term frequency or tf) with a global component (inverse document frequency or idf). Term frequency $tf(t, d)$ is defined as the relative number of times a term t appears in a document d:

$$tf(t, d) = \frac{count_d(t)}{|d|} \tag{3.1}$$

Inverse document frequency $idf(t, D)$ measures how much information a term t provides regarding the set of documents D, that is, whether it is common or rare in D:

$$idf(t, D) = \log_2 \frac{|D|}{|\{d \in D : t \in d\}|} \tag{3.2}$$

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{3.3}$$

To compute a ranking of similar documents for a given sentence query, the query's tf-idf vector q is computed and compare it to each of the documents' tf-idf vectors $d_1, d_2, \ldots, d_{|D|}$ using the cosine to calculate the angle θ between vector pairs:

$$\cos\theta_i = \frac{q \cdot d_i}{|q| \cdot |d_i|} \tag{3.4}$$

The closer this value is to one the better the match. Shortcomings of this model include the inability to reduce the description length of the document, since words are only replaced with values. Also, it does not tell much about the statistical structure within and between documents.

3.2.2 Latent Semantic Indexing

LSA [14] is a method that discovers hidden concepts in documents by using single value decomposition (SVD) on the set of documents D. LSA uses a term-document matrix A whose rows correspond to terms and columns correspond to documents. The entry $A_{i,j}$ equates to the tf-idf value of the term i for the document j. SVD decomposes A into U, Σ, and V, such that $A = U\Sigma V^*$, with U and V being orthonormal matrices and Σ being a diagonal matrix with the single values on its diagonal. LSA uses the decomposition to find a low-rank approximation, that is, a matrix $A_k = U_k\Sigma_k V_k^*$ of a predefined lower rank k closest in similarity to the original matrix A. This is done by deleting all but the k biggest single values in Σ. LSA minimizes the Frobenius distance $\|A - A_k\|_F$. In this application, k is the number of hidden concepts to be learned. The vector representations of the documents based on this model can be found in the columns of $\Sigma_k V_k^*$. The number of dimensions k is an empirical question. Essentially, k is much smaller than the original space dimension, which is usually the total number of documents. Previous papers show that for Wikipedia dumps a good value for k should be chosen between 200 to 500 [10].

3.2.3 Latent Dirichlet Allocation

LDA [9] is a generative probabilistic model that automatically discovers topics from a data collection. The basic idea is that documents are represented as random mixtures over latent topics, where each topic is characterized by a distribution over words. The model is a three-level hierarchical Bayesian model with the first level being the corpus-level, the second being the document-level, and the third being the word-level. Setting the number of topics to be learned to K, LDA makes the following assumptions for generating a document $d = \{w_1, \ldots, w_N\}$:

1. Choose the document length $N \sim Poisson(\xi)$
2. Choose the document's distribution of topics $\theta \sim Dir(\alpha)$ with K dimensions
3. For each document word w_n:
   a) Choose topic $z_n \sim Multinomial(\theta)$
   b) Choose w_n from $p(w_n|z_n, \beta)$, a multinomial probability conditioned on the topic z_n. β is a $K \times V$ matrix with $\beta_{ij} = p(w^j = 1|z^i = 1)$

The model is learned with Bayesian inference, e.g. by using collapsed Gibbs sampling and expectation propagation. As for K, [21] discusses how to choose the number of topics. Given the computed model, the topic of a word is predicted with Bayes' theorem.

4 Encyclopedic Knowledge Integration into RNNLMs

4.1 Motivation

In the past, RNNLMs were mainly fed with additional information that are syntactic or restricted to the local context. This can be problematic when translating sentences like

$$\text{“A journalist wrote articles for a column”} \tag{4.1}$$

The problem here is that “column” has ambiguous meaning and can both refer to an upright pillar or a newspaper section, although it usually refers to a pillar. If the training data is not taken from news domains or is limited in size so the words “column” and “journalist” have not been seen in the same context, the system is likely to choose the wrong translation. For us humans it is obvious that the newspaper section is meant here, as the content words “journalist” and “articles” suggest.

In this work, we want to provide RNNLMs with global information extracted from offline encyclopedia, such as Wikipedia. By doing so, the model may create stronger ties between words from the same domain, e.g. “column”, “articles”, and “journalist”, that did not often appear together in the past, with the purpose to find more suitable translations.

We propose two approaches, one that integrates side information on the word level into a FRNNLMs, and one that uses a dedicated continuous space vector determined on the sentence level in conjunction with a RNNLM, which we will refer to as an extended recurrent language model.

The proposed models were used to rescore English-Chinese and English-Romanian baseline systems.

In this work, we used the torch7 [1] implementation of a recurrent neural network-based language model [24] to train all of the models.

4.2 Encyclopedia and Knowledge Extraction

Wikipedia offers all of the available content to users for offline use [6] in a number of different languages. The contents are available for download both in XML data format or as SQL databases. In this work, XML files without links and media content [2] were used because they mostly contain textual data. The English (EN) archive has a size of 12.2 GB, the Chinese (ZH) archive a size of 1.2 GB and the Romanian (RO) archive a size of 363 MB. For each language, the Wikipedia content needs to be preprocessed in the beginning. A characteristic of Wikipedia is that most of its articles only refer to a small group of lexical categories. For example, there are a lot more articles about objects than activities. To complement the weaknesses of Wikipedia and show our model's

[1]http://torch.ch/
[2]zhwiki-latest-pages-articles.xml

performance independently of one specific encyclopedia, we web-crawled a Chinese lexicon from zdic.net [1] which offers short but precise Chinese explanations for all types of words. The following experiments that are based on Wikipedia will be referred to as "WIKI" and those based on the lexicon will be referred to as "ZDICT".

4.2.1 Word Level Information

In this approach, for each word in the training sentence we search for a corresponding category from Wikipedia, which we use as a label. On this basis, we create a parallel data stream of words and their labels, and use this as input into a factored neural language model in order to improve the rescoring of n-best lists. The motivation behind using word categories from Wikipedia in association with words is to strengthen the relatedness of words in a sentence, which is actually a good translation but gets a low score according to the underlying translation model. Given the previous sentence 4.1, Example 4.1 shows two potential hypotheses in decoding.

1. $\underbrace{\overset{\text{一位}}{\text{one}}}_{\text{NR}}\ \underbrace{\overset{\text{记者}}{\text{journalist}}}_{\text{新闻 (=news)}}\ \underbrace{\overset{\text{给}}{\text{for}}}_{\text{P}}\ \underbrace{\overset{\text{柱子}}{\text{column}}}_{\text{建筑(=archit.)}}\ \underbrace{\overset{\text{写}}{\text{write}}}_{\text{VV}}\ \underbrace{\overset{\text{文章}}{\text{article}}}_{\text{作品}}$

2. $\underbrace{\overset{\text{一位}}{\text{one}}}_{\text{NR}}\ \underbrace{\overset{\text{记者}}{\text{journalist}}}_{\text{新闻 (=news)}}\ \underbrace{\overset{\text{给}}{\text{for}}}_{\text{P}}\ \underbrace{\overset{\text{专栏}}{\text{column}}}_{\text{新闻(=news)}}\ \underbrace{\overset{\text{写}}{\text{write}}}_{\text{VV}}\ \underbrace{\overset{\text{文章}}{\text{article}}}_{\text{作品}}$

Example 4.1: Two hypotheses labeled with Wikipedia categories

Although the second translation is obviously the better one, the first translation could be more likely according to the translation model since, generally, a column describes more often a pillar than a newspaper area, so "柱子 (=archit.)" gets a higher model score. For example, the words "journalist" and "column" might have not been seen together. However, their Wikipedia categories "news" and "news" might have appeared togethers in the context of other news-related words. Our approach to solve this problem is to tag the words in the sentence with according Wikipedia categories. By labeling the words, the bond between words related to each other, such as "column" and "journalist", is reinforced by their common factor "新闻 (=news)" which contributes to a higher score for a correct translation.

4.2.1.1 Word Category Extraction

In the following, the Wikipedia category extraction and word tagging will be explained. First, the Wikipedia database dump is cleaned with the Wikipedia extractor tool WikiExtractor.py [3], that discards any information but plain text or annotation present in Wikipedia pages, such as images, tables, references and lists. In case of the Chinese Wikipedia, the extracted Wikipedia text must be further cleaned and normalized in multiple steps due to several characteristics specific to the Chinese Wikipedia. First, the entire content is displayed in traditional Chinese characters by default, so it needs to be transformed into simplified Chinese to be consistent with the training data. Another characteristic of the Chinese Wikipedia is that it uses five instead of two character sets, which are: simplified Chinese, Hongkong traditional, Macao traditional, Malaysia/Singapore simplified and

[3]http://medialab.di.unipi.it/wiki/Wikipedia_Extractor

Taiwanese style. These modes differ not only in the two standard character sets but also in local expressions specific to certain regions. This makes the transformation between character sets confusing, especially because the Chinese Wikipedia does not offer a general rule to retrieve simplified Chinese phrases. Afterwards, we segmented it with the Stanford Word Segmenter [4].

Once the preprocessing is done, the target side of the language pair is tagged with POS information. The POS tags for the Chinese corpus are generated by the Stanford Tagger [5]; the tags for the Romanian corpus are created by the tagger described in [22]. Figure 4.1 shows a snippet of POS-tagged Chinese text. A list of all possible Chinese POS tags can be found in "Part-Of-Speech Tagging Guidelines for the Penn Chinese Treebank" [3].

史蒂芬帕伦#NR ： #PU 追寻#VV 水银#NN 的#DEC 踪迹#NN
海洋#NN 是#VC 一#CD 个#M 非常#AD 复杂#JJ 的#DEG 事物#NN 。#PU

Figure 4.1: Segmented text labeled with POS

A Wikipedia page, labeled with a title, is either an article, a redirection page, or a category page which encompasses one or multiple pages. We define the search space as the set of all page titles. Given a word, we search for the page with the same word as title and retrieve its category which is found at the bottom of the page, as shown in Figure 4.2. In this example, the article has the title "马"(horse) and has three categories "驯养动物"(domestic animals), "马属"(Equus) and "生肖"(Chinese zodiac).

分类: 驯养动物|马属|生肖

Figure 4.2: Word categories at the bottom of a page

The equivalent information can be found in the XML file, as shown in Figure 4.3. The XML file has a list structure of all Wikipedia pages, which are included between the tags <page></page>. The tag <ns></ns> at the beginning of a new page indicates whether it is an article (0) or a category (4). For simplicity, the first category in the list, which usually serves as a good categorization, is taken as a word's category.

```
<page>
  <title>马</title>
  <ns>0</ns>
  <id>39564</id>
  <revision>
          .
          .
          .
          [[Category:驯养动物|EC]]
          [[Category:马属|FC]]
          [[Category:生肖|H]]</text>
          <sha1>5z7i2gcm92uxhcsvxkkh4kuompeqin7</sha1>
  </revision>
</page>
```

Figure 4.3: Title and category information in the XML file

The steps of generating a mapping of pages to their category are as follows: 1. Extract the categories of articles. 2. Extract the categories of redirected pages and add them to categories list of the redirecting pages. 3. Extract the categories of category pages. 4. Merge the results of the above steps to one final lookup table that does not distinguish between articles and category pages.

For the English-Chinese system it is sufficient to only tag the nouns in the data set with Wikipedia categories, whereas for the English-Romanian system we also tagged the entire corpus. This is because a big fraction of the Wikipedia articles deal with objects (nouns) rather than activities (verbs). Also, the Chinese Wikipedia is much larger than the Romanian Wikipedia. The results are presented in Chapter 5.

Using the POS tag as a word's default factor, we look up it's Wikipedia category and, if such one exists, update the word's factor accordingly. Denoting the set of unique words in the training corpus with $W = \{w_1, \dots, w_N\}$, the set of categories of words that appear in training with $C = \{C_1, \dots, C_M\}$, we define the frequency of n-sized categories as $|\{\ C_j \in C \mid |C_j| = n\ \}|$.

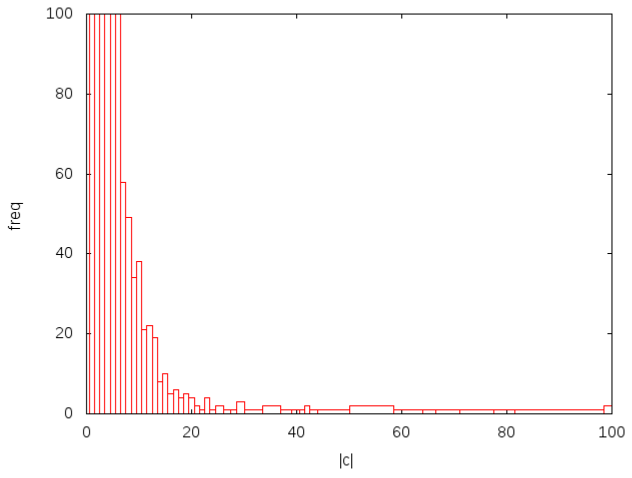

Figure 4.4: Frequency of categories based on category size

For the English-Chinese system, 5166 Wikipedia categories in total are seen in training. Figure 4.4 shows the distribution of the number of categories of a certain size. We can see, that the frequency of larger categories decreases rapidly. For example, there are less than ten categories that share the common size of 15. On the other hand, there are alot of categories that only contain one element. To reduce the number of categories we set a lower boundary x on a category's size, so in case a category has less than x elements we resort to the next higher category. We recursively go up the tree of categories and their children. We stop at a category that either fulfills the size requirement or is a root. Note that there can be multiple category trees. Among experiments with $x = 5, 10, 15$, those with $x = 5$ yielded the best results, which we present in Chapter 5. The total numbers of categories used for the Chinese corpus are listed in Table 4.1.

4.2.1.2 Integration into Factored Neural Language Model

In the input data preparation process a truncated vocabulary with the 10000 most frequent words is used, which is motivated by Zipf's law in NLP. It says there is only a small fraction of words that constitute a big portion of the corpus, the rest of the corpus is made out of less common words. The common words are mostly stop or function words,

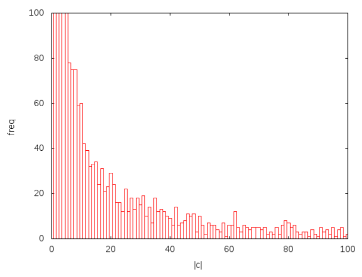

Figure 4.5: x=5

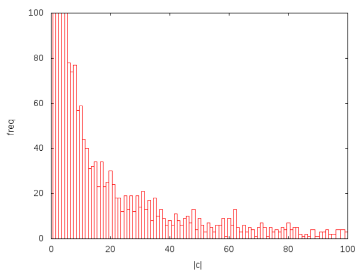

Figure 4.6: x=10

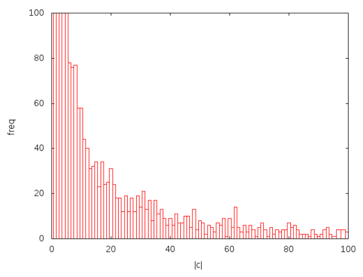

Figure 4.7: x=15

x	#categories
original	5166
5	3323
10	3301
15	3296

Table 4.1: Number of seen categories in training

e.g. “the”, that indicate how content words relate to each other. Content words usually have lower frequency and refer to objects, actions or properties. Within the vocabulary three entries are reserved, one for the start of a sentence; one for the end of a sentence; and one that represents out-of-vocabulary (OOV) words. Based on the vocabulary the data is transformed into a 2D-Tensor of word indices, where each row corresponds to a sentence. To speed up training time corpus is broken down into mini-batches and the model is trained on graphics processing units (GPUs).

For training we used the torch framework and the neural network packages nn, rnn. The factored language model proposed in the paper [28] served as the model basis, which takes one or multiple factors at the input layer and offers the option of a factorized output layer.

The input layer consists of a “LookupTable” and a “SplitTable” that project the input onto a lower dimensional word embedding vector, which is dense and real-valued. Each dimension of the vector represents a latent aspect of the word, and captures its syntactic and semantic properties [25]. The “SplitTable” splits the sentence into a set of word embedding vectors, which are then sent through one or multiple LSTM-based layers with the support of “Sequencer” modules and, finally, projected onto factored output probabilities. The instance of the FRNNLM we used is based on the work in [28] and takes two factors, the word’s surface form and its Wikipedia category. These are mapped to an embedding vector of size 100, which is the same size as the first LSTM-layer. For the second LSTM-layer we use 200 nodes. For the output we only used the surface form of the next word as the only factor. In model training statistical gradient descent is employed according to the negative log-likelihood loss function.

4.2.2 Sentence Level Information

The idea to use a dedicated topic vector is motivated by a more flexible representation of information and methods that are capable of drawing correlations betweens words independently of their distance. For example, the sentence

“Columns contain articles written by journalists” (4.2) exhibits several problems. First, when translated into a low-resource language, e.g. Romanian, some of the topic words do not have their own Wikipedia entry, so we cannot take full advantage of the first approach. Second, the contextual word of “columns”, which is “journalists”, can appear later in the sentence. This has motivated us to take words both left and right of the current word into account during translation. A topic vector is created for the current sentence based on topic-related Wikipedia articles. In the above example these would most probably be news related articles. By compressing the information of these articles into a vector which is associated with each word, we can improve the translation of single words by considering global topic information spanning neighboring words. To create a ranking of similar documents and their representation to a given sentence, vector space models are employed, such as tf-idf [38] and LSA [10], as well as probabilistic models, such as LDA [9]. After representing the sentences and cleaned encyclopedia articles as bag-of-words, we query for each sentence the most similar documents from the total set of encyclopedia articles. These documents provide general information, such as higher-level concepts, about the current sentence. Based on the chosen model, e.g. LSA, the best documents are transformed into vector representations and the averaged sum of the vectors is used as the additional feature input for the extended neural language model. This process is illustrated in Figure 4.8.

We used the python gensim library [4] to create the tf-idf, LSA and LDA models on the encyclopedia data. For the LSA model we chose $k = 300$ as the number of hidden concepts; for the LDA model we chose $K = 100$ and used the default values for the remaining parameters suggested by the gensim library.

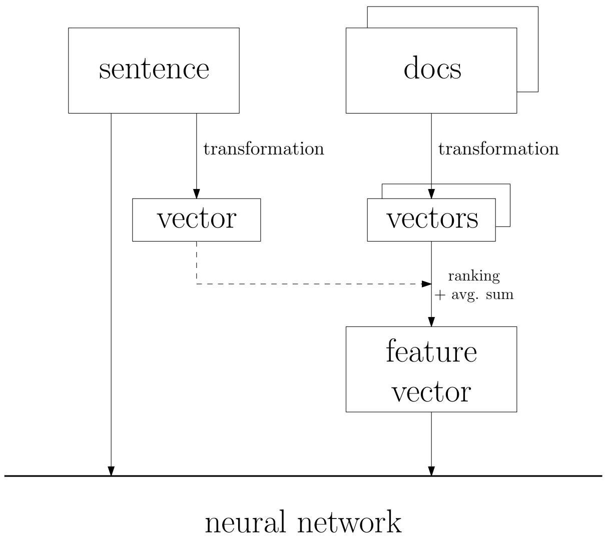

Figure 4.8: Feature input creation process for the extended language model. Sentence and encyclopedia articles are transformed into vector representations based on the underlying model. The articles most similar to the sentence are chosen; then their averaged sum constitutes the feature input.

[4]https://radimrehurek.com/gensim/

4.2.2.1 Integration into Extended Neural Language Model

The basic RNNLM consists of an input layer, a hidden layer with recurrent connections that maintains a representation of the sentence history, and an output layer which produces the probability distribution over words. We propose an extension of the RNNLM by extending it with an additional sentence-based feature layer that is connected to the output layer, using the nngraph package [2] of the torch framework. Since this real-valued feature vector stays the same for all words in the current sentence, it is replicated for each word beforehand. This way, the feature information is retained in the model while the same sentence is being processed. LSTM-based layers are used for the m hidden layers. Given a sentence $\mathbf{w} = \{w_1, w_2, \dots, w_n\}$, the sentence-level information $\mathbf{f}$ is computed for the sentence $\mathbf{w}$ based on the underlying topic model and duplicated for each word. For the i-th word we denote its representation with x_i, which is encoded using 1-of-N coding, the feature input with f_i, the hidden layers with $s^1, s^2, \dots, s^m$ and the output layer with y_i. With one connection from the feature input to the output layer, the hidden and output layers are computed according to (4.3).

$$\begin{aligned} s_i^1 &= f_1(U_1 x_i + W_1 s_{i-1}^1) \\ s_i^j &= f_j(U_j s_i^{j-1} + W_j s_{i-1}^j), \\ j &\in \{2, \dots, m\} \\ y_i &= g(V s_i^m + F \mathrm{f}_i) \end{aligned} \tag{4.3}$$

where f_i represents activation functions, and g the softmax function. For training of the network, that is finding the weight matrices $U_{1,\dots,m}, V, W_{1,\dots,m}, F$, stochastic gradient descent is used according to the negative log-likelihood loss function. We also tested the option of adding an additional connection from the feature layer to the first hidden layer, as indicated in Figure 4.9.

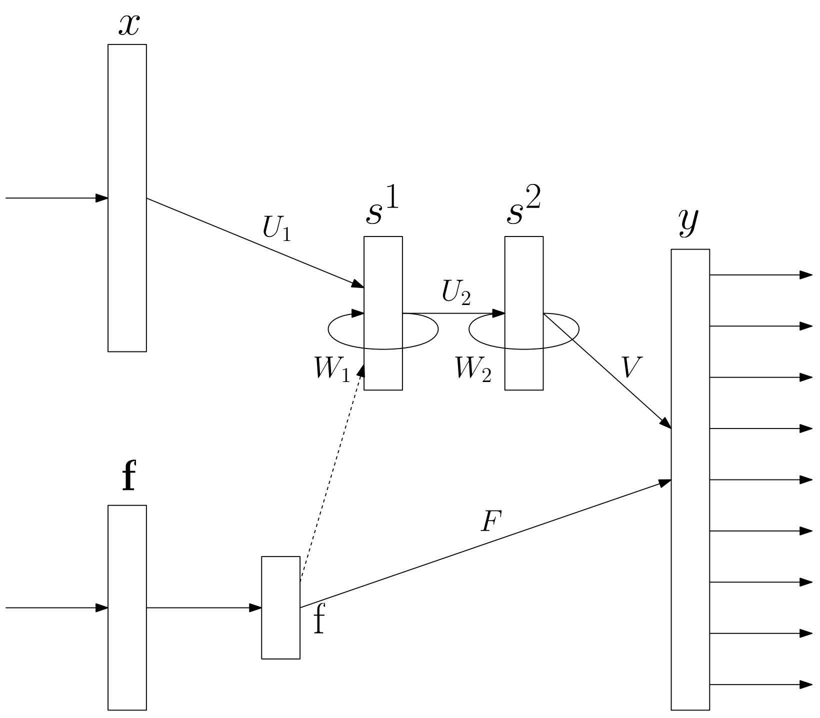

Figure 4.9: Extended recurrent language model with additional feature input **f** and two LSTM-based hidden layers. The dashed line from the feature input to the first hidden layer represents an optional connection.

5 Evaluation

5.1 Used Data and System Description

We evaluated the FRNNLM and the extended language model on English-Chinese and English-Romanian language pairs. For each language pair we created the n-best lists using the in-house phrase-based machine translation system and used the models as additional features in rescoring.

All recurrent neural network-based language models are trained on the target side of the parallel training data, and used in addition to the baseline features to improve rescoring.

5.1.1 English-Chinese

The English-Chinese phrase-based baseline systems are trained on the TED and UN corpus, optimized and rescored on the TED dev2010 and tested on the test2010 corpora. The Chinese corpus contains 148.968 sentences, the development set 887 sentences, and the test set 1570 sentences.

To build the baseline systems preprocessing including tokenization and smartcasing is applied to the data. Then, a word alignment is created over the parallel data with the GIZA++ toolkit [34]. A phrase table is built on top of that with the Moses toolkit [23]. In addition, two other phrase tables are created: an indomain phrase table which is only based on the TED data, and a union of the basic phrase table and an adapted phrase table (conjunction of large out-of-domain phrase table and indomain phrase table) according to [29]. Corpus adaptation is motivated by the problem of gathered indomain corpora usually being limited in size, which makes it impossible to train a standalone model with respect to the target recognition task. For this reason, it is common to use a large background corpus for model training and an indomain corpora for fine tuning.

Several language models were tested. First, a 4-gram language model with modified Kneser-Ney smoothing was trained with the SRILM toolkit [41]. Second, an indomain language model based on the indomain phrase table was created. Finally, a bilingual language model and a 9-gram cluster language model with 1000 word classes and witten-bell smoothing was created according to the MKCLS algorithm [31]. The latter alleviates data sparsity problems.

As for reordering two different rules were used on the source language. First, short range reordering rules were learned from POS information produced by a TreeTagger [39]. Second, tree-based reordering rules were learned with the Stanford Parser [37] to model longer dependencies and ranges. The resulting reordering possibilities are then encoded in a lattice.

Multiple SMT systems with different configurations were created. After comparing them with each other, we chose two baseline systems for which we will present the results: a basic system and a system with the highest BLEU score on the test set.

The first baseline system, which will be referred to as "basic", uses a basic 4-gram language model and short reordering rules. In total, eight features are used to create an n-best list with 300 entries, which include the language model, the reordering model, word count, phrase count and four translation models (forward and backward phrase translation and forward and backward lexical translation).

The second baseline system, which will be referred to as "complex", uses the adapted union phrase table, a general and an indomain language model, as well as long range tree reordering rules. In total 12 features are incorporated to create an n-best list with 3000 entries, which include two language models, one reordering model, word count, phrase count, and six reordering models (both-way phrase translations, both-way lexical translations, both-way adapted phrase translations, and the lattice score).

Minimum error rate training [32] is used for optimization in decoding as well as for rescoring of the n-best list.

All of the Chinese neural language models use a vocabulary of size 10K.

For sentence level side information, as described in Section 4.2.2, an additional web-crawled Chinese lexicon from zdic.net [1] is used in the English-Chinese system to show the model's performance independently of a specific encyclopedia.

5.1.2 English-Romanian

The English-Romanian baseline system is trained on corpora of WMT 2015 Shared Translation Task, optimized on the first half of news-dev 2016 and tested on the second half of news-dev 2016. In addition, for rescoring a subset of 2000 sentences of the SETimes corpus is used for further optimization.

The English-Romanian baseline system uses two word-based language models, 2 cluster-based models with 50, 100 or 1000 clusters, and a POS-based language model. In total 22-23 features are used to generate an n-best list of size 300. A full system description can be found in [28].

Minimum error rate training [32] is used for optimization in decoding, whereas ListNet [27] is used for rescoring of the n-best list.

All of the Romanian neural language models use a vocabulary of 5K.

5.2 Results

In this section, we show the results to all of the experiments conducted on the English-Chinese and English-Romanian system. The results are displayed in BLEU.

5.2.1 English-Chinese

The English-Chinese (ZH) systems before rescoring gives a BLEU score of 16.35 ("basic") and 17.05 ("complex") in testing. For all of the following experiments, a system that is rescored with a basic RNNLM will serve as the baseline system system. We labled only nouns in the data sets, which led to word coverages as shown in Table 5.1. In the first experiment of English-Chinese the scores of the FRNNLM are used in addition to the baseline features, the results are shown in Table 5.2 and Table 5.3. BLEU score improvements over the baseline system are displayed in parentheses. The FRNNLM that uses surface form and Wikipedia categories as factors performs 0.66 BLEU score points better in testing than the same model without Wikipedia categories. The system that uses words, Wikipedia categories, and POS as factors performs 0.84 BLEU points better than the same system without Wikipedia categories.

In a second experiment, extended language models along with the different topic models to compute the feature input vector based on Wikipedia are studied. Tf-idf, LSA

Data	devdata	testdata
basic	11.19%	10.68%
complex	11.21%	10.61%

Table 5.1: ZH factored language models: word coverage when only nouns are labeled

experiment	devdata	testdata
Baseline	14.07	16.61
+Factored LM POS	14.14	17.10
+Factored LM Cat	14.09 (-0.05)	16.96 (-0.14)
+Factored LM Cat + POS	14.11 (-0.03)	17.43 (+0.33)

Table 5.2: ZH factored language models (“basic”)

Model	devdata	testdata
Baseline	14.7	17.02
+Factored LM POS	14.77	16.97
+Factored LM Cat	14.89 (+0.12)	17.63 (+0.66)
+Factored LM Cat + POS	14.75 (-0.02)	17.81 (+0.84)

Table 5.3: ZH factored language models (“complex”)

and LDA are used for similar document ranking as well as vector representation. It is worth mentioning that the method for ranking can be paired with a different choice for representation. The performance results of various combinations of pairings are presented in Table 5.4 and Table 5.5.

Rank	Vect	devdata	testdata
Baseline		14.07	16.61
TFIDF	TFIDF	14.05 (-0.02)	17.26 (+0.65)
LSA	TFIDF	14.03 (-0.04)	16.68 (+0.07)
LSA	LSA	14.10 (+0.03)	16.78 (+0.17)
LDA	TFIDF	14.08 (+0.01)	16.54 (-0.07)
LDA	LDA	14.06 (-0.01)	16.88 (+0.27)

Table 5.4: ZH extended language models: overview of different feature vectors (“basic”)

Rank	Vect	devdata	testdata
Baseline		14.70	17.02
TFIDF	TFIDF	14.78 (+0.08)	17.68 (+0.63)
LSA	TFIDF	14.78 (+0.08)	17.31 (+0.29)
LSA	LSA	14.83 (+0.13)	17.80 (+0.78)
LDA	TFIDF	14.79 (+0.09)	17.41 (+0.39)
LDA	LDA	14.79 (+0.09)	17.27 (+0.25)

Table 5.5: ZH extended language models: overview of different feature vectors (“complex”)

In case of tf-idf and LSA choosing the same method for both ranking and vector representation causes a higher gain. For example, the tf-idf model creates an increase of 0.65 BLEU and 0.63 BLEU respectively on the baseline systems, and LSA an increase of 0.17 BLEU and 0.78 BLEU. The only exception to this rule is LDA. One reason for this could be suboptimal parameter picks for this generally more complex generative model. Despite the good performance of the LSA model, we use the tf-idf model in the following experiments due to its simplicity and, thus, faster training and evaluation. Models trained on Wikipedia use the top two articles for each sentence and are trained with a learning rate of 0.05 for 50 epochs.

The Chinese lexicon crawled from zdic.net [1] contrasts Wikipedia in the variety and length of definitions. The lexicon provides more but shorter compact explanations for all types of words, particularly verbs. The models based on zdict ("ZDICT") use the top ten most similar articles for each sentence and are trained for 40 epochs with a learning rate of 0.05. Despite the differences between Wikipedia and zdict, the use of the lexicon shows an increase of 0.13 BLEU and 0.56 BLEU in testing, which is comparable to the model improvement based on Wikipedia. The results are shown in Table 5.6 and Table 5.7.

Model	devdata	testdata
Baseline	14.07	16.61
+WIKI	14.05 (-0.02)	17.26 (+0.65)
+ZDICT	14.10 (+0.03)	16.74 (+0.13)

Table 5.6: ZH extended language models: comparison between different encyclopedia sources ("basic")

Model	devdata	testdata
Baseline	14.7	17.02
+WIKI	14.78 (+0.08)	17.68 (+0.66)
+ZDICT	14.91 (+0.21)	17.58 (+0.56)

Table 5.7: ZH extended language models: comparison between different encyclopedia sources ("complex")

An overview of all models along with their perplexities is given in Table 5.8.

Model	PPL
Baseline	128.17
Factored LM POS	110.86
Factored LM Cat	109.73
Factored LM Cat+POS	110.38
WIKI	118.11
ZDICT	118.64
WIKI+ZDICT	119.02
WIKI 4-CONTEXT	118.46

Table 5.8: ZH model perplexities

All models exhibit an evident reduction in perplexity compared to the baseline system, which is consistent with the rescoring results. In addition, two extended language models whose feature inputs are determined differently are listed with their model perplexities.

The first model's feature input depends on both Wikipedia and zdict.net; the second model's feature input is based on the last and next four sentences.

As indicated in Figure 4.9, an additional connection was established between the feature layer and the first hidden layer in another experiment. As a result, the model with two connections gained a small increase of 0.01 BLEU and 0.16 BLEU on the model with just one connection, as shown in Table 5.9 and Table 5.10.

Model	devdata	testdata
Baseline	14.07	16.61
+TFIDF	14.05 (-0.02)	17.26 (+0.65)
+2Con TFIDF	14.05 (-0.02)	17.27 (+0.66)

Table 5.9: ZH extended language models: connecting feature input with two layers ("basic")

Model	devdata	testdata
Baseline	14.70	17.02
+TFIDF	14.78 (+0.08)	17.68 (+0.63)
+2Con TFIDF	14.74 (+0.04)	17.81 (+0.79)

Table 5.10: ZH extended language models: connecting feature input with two layers ("complex")

5.2.2 English-Romanian

In the previous work [28], the factored language model for English-Romanian integrated four factors: the word's surface form, POS, and word clusters with 100 and 1000 class respectively. Our English-Romanian (RO) experiments build on top that, using a vocabulary size of 5K for all systems as well as the same denotations to illustrate the following results. For this language pair we extracted Wikipedia categories for both nouns and all word types. The word coverages of tagged words can be found in Table 5.11, which shows significant differences between the two choices.

Data	Nouns	All
Devdata	1.94%	9.06%
Testdata	2.62%	10.17%
Setimes	3.48%	11.14%

Table 5.11: RO factored language models: word coverage by Wikipedia categories

In the first experiment, the FRNNLMs are evaluated without the baseline features. The system from the previous work with all four factors for input and prediction reached a BLEU score of 28.54, as shown in Table 5.12. This model will serve as reference for our models. The FRNNLMs with Wikipedia categories as factor show an improvement of 0.17 BLEU and 0.30 BLEU points respectively depending on how the data is tagged.

Table 5.13 shows the results of the model's joint probability used in addition to the baseline features in three configurations [28]. Adding Wikipedia categories for all word types achieves an improvement of about 0.1 BLEU in two configurations (Conf2 and Conf3) over the reference model.

Input	Prediction	Single
Word	Word	27.88
All factors	All factors	28.54
+Cat (Nouns)	+Cat (Nouns)	28.71 (+0.17)
+Cat (All Words)	+Cat (All)	28.84 (+0.30)

Table 5.12: RO factored language models: single scores

Model	Conf1	Conf2	Conf3
Baseline	29.86	30.00	29.75
+All factors	29.94	30.01	30.01
+All factors + Nouns	29.94 (+0.00)	30.31 (+0.30)	29.99 (-0.02)
+All factors + All Words	29.95 (+0.01)	30.13 (+0.12)	30.14 (+0.13)

Table 5.13: RO factored language models: end scores

In another experiment, the extended language model is used for rescoring in addition to the previously discussed factored neural language models. The results are illustrated in Table 5.14. The improvement over the original factored neural language model with four factors is indicated in parentheses. It turns out that the combined use of Wikipedia categories and Wikipedia topic information performs about 0.2 BLEU better in two different configurations (Conf2 and Conf3). The system that uses Wikipedia categories for all word types in conjunction with the extended language model achieves an improvement of 0.07 BLEU over the system without extended language model scores in these configurations. The best system exhibits a score of 30.23 BLEU, which is 0.22 BLEU better than the best system of the previous work [28] that has the same vocabulary size.

Model	Conf1	Conf2	Conf3
All factors	29.99 (+0.05)	30.19 (+0.18)	29.99 (-0.02)
All factors + Nouns	29.99 (+0.05)	30.29 (+0.28)	30.23 (+0.24)
All factors + All Words	30.00 (+0.05)	30.20 (+0.19)	30.21 (+0.20)

Table 5.14: RO extended language models: end scores

6 Conclusion

6.1 Summary

Language modeling is an important issue in statistical machine translation because it determines the fluency and adequacy of the language. The success of neural networks revolutionized language modeling and machine translation in general. At the same time, they exacerbate the data sparsity problem, that is the need of large amounts of training data to generate a reliable model. To counter this problem, this thesis focuses on adding related external information, that contributes to a better understanding of a sentence, to neural network-based language models. The additional higher-level information is obtained from encyclopedic sources, such as Wikipedia. We then use these extended language models in the rescoring of n-best lists of a statistical machine translation system.

We propose two approaches to integrate side information into recurrent neural language models. The first is based on a factored neural language model, that is able to create multiple predictions of parallel streams of input data – the factors. In previous works, the use of factors was limited to basic syntactic information, such as POS. We create word classes extracted from Wikipedia and use them as factors to complement the information in the sentence. In another approach we extend the conventional recurrent neural network by a dedicated feature vector in the input layer. The idea is to incorporate more complex information into the neural network. For each sentence in training, topic related encyclopedia articles are searched and compressed into a vector form, which then serves as the additional feature input. We introduce three methods to determine the relatedness between articles and sentences and represent the documents in the required form.

By using global information from large encyclopedia, we have improved translation systems on two different language pairs by up to 0.84 BLEU and 0.24 BLEU respectively. This work exhibits great potential for low-resource translation tasks.

6.2 Future Work

For future work it would be very helpful to extend the knowledge source, in our case Wikipedia, by semantic networks, such as WordNet. For some languages that are less common, this will be a challenging task. But also for more popular languages such as Chinese, it can be difficult to find freely available sources. On the basis of a given external source, other co-occurrence measures or topic models than the ones proposed can be applied to supplement RNNLMs, e.g. frequency and mutual information. Depending on the language and knowledge base, the same method can produce different results. Moreover, bilingual RNNLMs can be adopted that also consider the n-grams of the source sentence for the next word prediction. An advantage of those can be the incorporation of additional information on both the source and target side of the language pair to facilitate lexical disambiguation. Finally, the proposed methods can be applied in neural machine translation, especially for low-resource languages.

Bibliography

[1] 汉典 zdic.net. `http://www.zdic.net/`. (Accessed on 08/14/2016).

[2] Github - torch/nngraph: Graph computation for nn. `https://github.com/torch/nngraph`. (Accessed on 09/16/2016).

[3] repository.upenn.edu/cgi/viewcontent.cgi?article=1039&context=ircs_reports. `http://repository.upenn.edu/cgi/viewcontent.cgi?article=1039&context=ircs_reports`. (Accessed on 08/16/2016).

[4] The stanford natural language processing group. `http://nlp.stanford.edu/software/segmenter.shtml#Download`. (Accessed on 08/16/2016).

[5] The stanford natural language processing group. `http://nlp.stanford.edu/software/tagger.shtml`. (Accessed on 08/16/2016).

[6] Wikimedia downloads. `https://dumps.wikimedia.org/`. (Accessed on 09/29/2016).

[7] Satanjeev Banerjee and Alon Lavie. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, volume 29, pages 65–72, 2005.

[8] Jeff A Bilmes and Katrin Kirchhoff. Factored language models and generalized parallel backoff. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology: companion volume of the Proceedings of HLT-NAACL 2003–short papers-Volume 2*, pages 4–6. Association for Computational Linguistics, 2003.

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

[10] Roger B Bradford. An empirical study of required dimensionality for large-scale latent semantic indexing applications. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 153–162. ACM, 2008.

[11] Razvan C Bunescu and Marius Pasca. Using encyclopedic knowledge for named entity disambiguation. In *EACL*, volume 6, pages 9–16, 2006.

[12] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136. ACM, 2007.

[13] Silviu Cucerzan. Large-scale named entity disambiguation based on wikipedia data. In *EMNLP-CoNLL*, volume 7, pages 708–716, 2007.

[14] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990.

[15] George Doddington. Automatic evaluation of machine translation quality using n-gram co-occurrence statistics. In *Proceedings of the second international conference on Human Language Technology Research*, pages 138–145. Morgan Kaufmann Publishers Inc., 2002.

[16] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJcAI*, volume 7, pages 1606–1611, 2007.

[17] Xianpei Han and Le Sun. An entity-topic model for entity linking. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 105–115. Association for Computational Linguistics, 2012.

[18] Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics-Volume 2*, pages 539–545. Association for Computational Linguistics, 1992.

[19] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[21] Matthew Hoffman, Francis R Bach, and David M Blei. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*, pages 856–864, 2010.

[22] Radu Ion, Elena Irimia, Dan Stefanescu, and Dan Tufis. Rombac: The romanian balanced annotated corpus. In *LREC*, pages 339–344. Citeseer, 2012.

[23] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics, 2007.

[24] Nicholas Léonard, Sagar Waghmare, and Yang Wang. Rnn: Recurrent library for torch. *arXiv preprint arXiv:1511.07889*, 2015.

[25] Shujie Liu, Nan Yang, Mu Li, and Ming Zhou. A recursive recurrent neural network for statistical machine translation. In *ACL (1)*, pages 1491–1500, 2014.

[26] Tomas Mikolov and Geoffrey Zweig. Context dependent recurrent neural network language model. In *SLT*, pages 234–239, 2012.

[27] Jan Niehues, Quoc Khanh Do, Alexandre Allauzen, and Alex Waibel. Listnet-based mt rescoring. *EMNLP 2015*, page 248, 2015.

[28] Jan Niehues, Thanh-Le Ha, Eunah Cho, and Alex Waibel. Using factored word representation in neural network language models.

[29] Jan Niehues and Alex Waibel. Detailed analysis of different strategies for phrase table adaptation in smt. In *Proceedings of the Tenth Conference of the Association for Machine Translation in the Americas (AMTA)*, 2012.

[30] Sonja Nießen, Franz Josef Och, Gregor Leusch, Hermann Ney, et al. An evaluation tool for machine translation: Fast evaluation for mt research. In *LREC*, 2000.

[31] Franz Josef Och. An efficient method for determining bilingual word classes. In *Proceedings of the ninth conference on European chapter of the Association for Computational Linguistics*, pages 71–76. Association for Computational Linguistics, 1999.

[32] Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 160–167. Association for Computational Linguistics, 2003.

[33] Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alexander M Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, et al. A smorgasbord of features for statistical machine translation. In *HLT-NAACL*, pages 161–168, 2004.

[34] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51, 2003.

[35] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[36] Michael JD Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7(2):155–162, 1964.

[37] Anna N Rafferty and Christopher D Manning. Parsing three german treebanks: Lexicalized and unlexicalized baselines. In *Proceedings of the Workshop on Parsing German*, pages 40–46. Association for Computational Linguistics, 2008.

[38] Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1986.

[39] Helmut Schmid. Probabilistic part-ofispeech tagging using decision trees. In *New methods in language processing*, page 154. Routledge, 2013.

[40] Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of association for machine translation in the Americas*, volume 200, 2006.

[41] Andreas Stolcke et al. Srilm-an extensible language modeling toolkit. In *Interspeech*, volume 2002, page 2002, 2002.

[42] Michael Strube and Simone Paolo Ponzetto. Wikirelate! computing semantic relatedness using wikipedia. In *AAAI*, volume 6, pages 1419–1424, 2006.

[43] Christoph Tillmann, Stephan Vogel, Hermann Ney, Arkaitz Zubiaga, and Hassan Sawaf. Accelerated dp based search for statistical translation. In *Eurospeech*, 1997.

[44] Youzheng Wu, Xugang Lu, Hitoshi Yamamoto, Shigeki Matsuda, Chiori Hori, and Hideki Kashioka. Factored language model based on recurrent neural network. 2012.

Glossary

n-best list top n candidate translations. 9, 10, 15, 20, 27, 28, 33

n-gram model predicts the next word in a sequence depending on the previous $n-1$ words. 1, 3, 10, 15

backpropagation method used together with an optimization method, e.g. gradient descent, to train a neural network. 11, 13

baseline system reference system which serves as a basis for change. 2, 28

corpus set of texts for statistical analysis. 1, 27

decoder finds the best translation to a given sentence. 1

epoch one full training cycle on the training set. 12

feed forward neural network an artificial neural network without cycles. 11, 13

hypothesis candidate translation. 3, 6–10, 20

language model models the language fluency of a word sequence. 1, 3, 5, 6, 11, 19, 27, 28

log-linear model see Formula 2.13. 6, 9, 10, 15

neural network see Section 2.3. 11, 12

perplexity measure how well a probability distribution predicts a sample. 11

recurrent neural network an artificial neural network with cycles. 12–14, 19

reordering model measures the likelihood of the movements of words and phrases. 5, 27, 28

translation model provides a mapping of words or phrases between two languages. 1, 3, 4, 28

Acronyms

BLEU bilingual evaluation understudy. 8–10, 27, 28, 30, 31, 33

BPTT backpropagation through time. 13, 14

EM expectation maximization. 4

FLM factored language model. 10, 15

FRNNLM factored recurrent neural network language model. 15, 19, 23, 27, 28, 31