UPPSALA
UNIVERSITET

Neural Language Models with Explicit Coreference Decision

Jenny Kunz

Uppsala University
Department of Linguistics and Philology
Master's Programme in Language Technology
Master's Thesis in Language Technology

December 11, 2018

Supervisors:
Christian Hardmeier, Uppsala University

Abstract

Coreference is an important and frequent concept in any form of discourse, and Coreference Resolution (CR) a widely used task in Natural Language Understanding (NLU). In this thesis, we implement and explore two recent models that include the concept of coreference in Recurrent Neural Network (RNN)-based Language Models (LM). Entity and reference decisions are modeled explicitly in these models. Both models learn to save the previously observed entities in a set and to decide if the next token created by the LM is a mention of one of the entities in the set, an entity that has not been observed yet, or not an entity.

After a theoretical analysis where we compare the two LMs to each other and to a state of the art Coreference Resolution system, we perform an extensive quantitative and qualitative analysis. For this purpose, we train the two models and a classical RNN-LM as the baseline model on the OntoNotes 5.0 corpus with coreference annotation. While we do not reach the baseline in the perplexity metric, we show that the models' relative performance on entity tokens has the potential to improve when including the explicit entity modeling.

We show that the most challenging point in the systems is the decision if the next token is an entity token, while the decision which entity the next token refers to performs comparatively well. Our analysis in the context of a text generation task shows that a wide-spread error source for the mention creation process is the confusion of tokens that refer to related but different entities in the real world, presumably a result of the context-based word representations in the models.

Our re-implementation of the DeepMind model by Yang et al., 2016 performs notably better than the re-implementation of the EntityNLM model by Ji et al., 2017 with a perplexity of 107 compared to a perplexity of 131.

Contents

Preface

First of all, I would like to thank my supervisor Christian Hardmeier for his assistance during the work at this thesis, for many valuable comments and suggestions and for the idea to use PyTorch :-)

I also thank all of my teachers and fellow students for their inspirational inputs, assistance and support in the last two years.

1 Introduction

Coreference is a concept that very frequently occurs in human language, and it is a fundamental property of coherent communication. It occurs when two or several expressions refer to the same entity within a text, a conversation or another form of discourse.

[Donald Knuth] will not do any [further development] of [TeX], but **[he]** will continue to fix [bugs].

In this example sentence, noun phrases (that can also be called mentions) are in brackets. [he] is a pronoun referring back to [Donald Knuth], who was introduced as an entity in the beginning of the sentence.

The aim of coreference resolution is to determine which noun phrases in a given text refer to the same entity in the real world. The noun phrases in the text are partitioned into clusters so that one clusters refers to one real-world entity. It is an important auxiliary task for problems that depend on natural language understanding, such as information extraction and question answering.

Although for humans it is mostly intuitively clear what is meant, it is a challenging task for a machine as the question which entity is meant often relies on background knowledge that the discourse partners typically share. An example for this is:

When **[the Pope]** met **[Brad Pitt]**, **[he]** gave **[him]** [a blessing].

Most people will intuitively know that it is most probably the Pope who gives Brad Pitt an audience, because giving blessings is a common thing to do for Popes, but not for actors. But there is no indicator in the sentence itself that it is not the other way round, it is world knowledge that humans have acquired during their lives in the society, but that is still very hard to include in computer programs.

A new milestone in coreference resolution, as in many other problems in Natural Language Processing (NLP), has been the usage of Neural Networks (NN). It has not only led to performance gains but also to a trend to turn away from having many hand-engineered features, which can make models more easily applicable to other languages or domains.

Neural Networks have also been a milestone in the development of Language Models (LM). Recently, there have also appeared LM approaches that incorporated entity representations to explicitly model coreference, following the idea that explicitly modeling coreference and deciding how to form the reference can influence the created text, and thereby improve it.

The aim of this thesis is to introduce and compare neural approaches to the explicit modeling of reference in language models. It consists of a theoretical comparison of the architecture and features of two recent systems by Yang et al., 2016 and Ji et al., 2017 with a state of the art system for coreference resolution by K. Lee et al., 2017, as well as a practical comparison of the two LM systems that is designed to answer the following research questions:

- Does the reference modeling improve a LM?
- Which of the LM systems performs better?
- Do their errors of the system correlate with certain linguistic or structural properties of the entities?
- Can the errors be associated with a system's architecture or features?

For the practical comparison, we implemented models that are based on the two language model systems and trained them in order to perform a comparative quantitative and qualitative analysis.

The terminology and backgrounds for this work are introduced in chapter 2 to 4. Chapter 2 introduces the concept of neural networks with a focus on the features and architectures used in the approaches that are subject to this work. It is followed by two theoretical chapters for the backgrounds and approaches to coreference resolution (chapter 3) and modeling coreference in language models (chapter 4), with a strong focus on the neural approaches. With a detailed overview of the architectures of the three systems, they pave the way for a theoretical comparison and discussion in chapter 5. Chapter 6 describes the setup of the experiments conducted in this work in terms of data sets used, the implemented variants of the systems, implementational details and the evaluation metrics and procedure. In chapter 7, the implemented systems are evaluated in different metrics and aspects, regarding both the overall performance as a language model and the success in selecting and forming entities. Chapter 8 discusses the evaluation results and correlates them to the different designs of the systems, also based on the findings of the theoretical comparison, and chapter 9 concludes this thesis.

2 Neural Networks in NLP

Artificial Neural Networks are an approach to Machine Learning (ML) that has proven to be very powerful. In their most common form, they can learn to recognize patterns in supervised training data and approximate a wide variety of functions.

Although it was already in the 1950s that artificial neural networks were implemented for the first time, they have not shown today's potential at that time. The amount of training data was very limited back then, while with the age of Big Data, it has grown dramatically and made both the training of better performing NNs possible and the use of NNs more attractive. A second reason for their recent reappearance is that the training of neural networks needs extensive computational resources, which have now become available and relatively cheap. (Goodfellow et al., 2016) Today, their usage in NLP is wide-spread and they have been successfully adopted to many problems, often outperforming previous results from other statistical and from deterministic systems. (Goldberg, 2016)

In section 2.1, we will focus on the input to a neural network, and particularly on word representations in section 2.1.1. Section 2.2 will introduce feed-forward neural networks as the most basic type of neural networks, and their training in section 2.2.1. This section will give backgrounds for section 2.3 that will introduce recurrent neural networks and their currently most popular type, the LSTM (section 2.3.1. Section 2.4 is about convolutional neural networks and section 2.5 finally introduces the attention mechanism in neural networks.

2.1 Feature Vectors

As for many other machine learning algorithms, the representation of the data is crucial for the performance of neural networks. This representation consists of so-called features: measurable properties of the observation that we wish to process. What we want for machine learning is a set of features that captures as much relevant and as little irrelevant information as possible and that the algorithm can understand.

Generally, for the input to the network, a set of features $f_1, ..., f_n$ is created. For example, for a noun phrase, we could have two features: the gender and the grammatical number. They can be represented as binary variables: 0 for male and 1 for female and 0 for singular and 1 for plural respectively. For the phrase "the old men", f_1 will be 0 and f_2 will be 1. Then the corresponding vector representations for the features are retrieved. For "the old men", we can combine the gender and the number feature vector to a 2-dimensional vector [0 1].

But the representation of the features is usually more complex as the example above where both features are binary. When we want to encode each word in a

vocabulary in a vector, there are many possible methods to get the representations. A very simple but also very sparse representation of features is the one-hot encoding. It represents each word as an N-dimensional vector, where N is the size of the vocabulary and each component of the vector corresponds to a word in the vocabulary. A word feature vector is then a vector that consists of zeroes in all components except one, the one that corresponds to the word we want to encode. If we for example have a vocabulary of three, component 1 can correspond to „I“, component 2 to "love", component 3 to "vectors", the vector for the word "vectors" would be [0 0 1]. As typical vocabularies have thousands or tens of thousands of words, the vectors will be very high-dimensional.

One of the advantages of neural networks is their ability to work with dense, lower-dimensional feature vectors instead of these sparse representations. This is often considered desirable because it allows similar features to have similar vectors, so that common information can be used by the network. (Goldberg, 2016) So, for example, the word "sine" can have a similar vector like "cosine" or "tangent", and the network could make use of the fact that these words will often appear in similar contexts.

2.1.1 Word Representations

The performance of many NLP tasks can be improved using pre-trained word representations that provide additional information about a word. They have therefore become widely used. (Goldberg, 2016)

A word embedding (WE) is a dense vector that captures information about a word. It is usually derived by unsupervised training, using the context of a word, based on the assumption that similar words occur in similar contexts.

There are two main approaches to creating such representations.

The frequency-based matrix factorization methods are starting with a co-occurrence matrix M of size $W \times W$, where W is the vocabulary size and where entry M_{ij} denotes how often word j occurs in the context of word i. Each row M_i is the representation of the corresponding word at index i with dimensionality W. A simple example for such a matrix M could be, for the text "Words need embeddings. Embeddings are vectors." and the sentence as the context:

.	words	need	embeddings	are	vectors
words	1	1	1	0	0
need	1	1	1	0	0
embeddings	1	1	2	1	1
are	0	0	1	1	1
vectors	0	0	1	1	1

The vocabulary size W in this example is 5 and the vector for the word "embedding", which is the third row (M_3): [1 1 2 1 1]. It is visible from the vector that "embeddings" co-occurs with every other word in the vocabulary, and that "embeddings" itself occurs two times (however, this field for a word co-occuring with itself will be set to 0 in many approaches).

The next step is a dimensionality reduction. Although in the example above it does not seem useful, it is important to remember that common vocabulary sizes

are very high. Therefore M is mapped to a new matrix m of size $W \times d$, where d has fewer dimensions than W, using a function g, where $m = g(M)$. The row m_w now represents word w as a vector with only d dimensions.

An early NLP approach with this method was Latent Semantic Analysis (Dumais, 2004). The popular word embedding model GloVe (Pennington et al., 2014) is also based on this approach.

The second approach is coming from the field of language modeling and derives the WEs with a neural network. It is based on fixed-size local context windows. It was first proposed by Bengio et al., 2003 as a secondary output of a neural language model. Its most famous representant is word2vec (Mikolov et al., 2013). A neural network model is trained on the context of each word. There are two variants of word2vec: The Continuous Bag-of-Words (CBOW) model that predicts the current word at position n given the words at position $1, ..., n-1$, and the skip-gram model that predicts the words at positions $1, ..., n-1$ given the word at position n.

It can be useful to combine these methods as the approaches provide different information about the word. It is also common to combine them with embeddings that provide lexical information, for example obtained running a neural network over a word's characters, and features for morphological information, because word embeddings do not work well for rare words and are not prepared for out-of-vocabulary (OOV) words (Goldberg, 2016).

2.2 Feed-Forward Neural Networks

Fully connected feed-forward neural networks (FFNN) are the most basic type of NN, and although the systems that are the subject to this work are based on recurrent neural networks (RNNs), we will firstly introduce them for a general overview of how a neural network works and then extend the principle to the RNN type.

Assuming that $f^*(x) = y$ is a function that maps an input x to its actual category y, the goal of a feed-forward neural network is to approximate $f^*(x)$ by learning parameters θ so that $y = f(x, \theta)$ best matches $f^*(x)$.

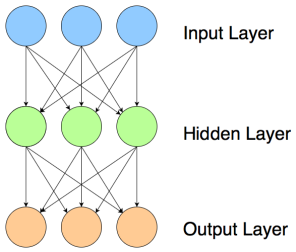

Figure 2.1: FFNN Architecture

The NN consists of a composition of functions, also called layers. Figure 2.1 gives an intuition about the architecture of a FFNN. The colored circles are the so-called neurons and the arrows are the connections between them. The neurons are arranged in layers; one color of the neurons corresponds to one layer.

The layers of a FFNN can be divided into three types: input layer (the blue neurons in Figure 2.1), hidden layer(s)(green) and output layer(red). The input layer processes the input vector x. It is followed by one or more hidden layers that are the core of the neural network. The final layer creates the output of the network.

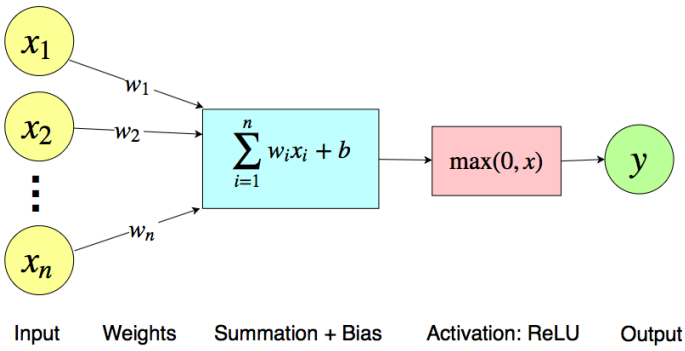

Figure 2.2: Architecture of a Neuron in a Hidden Layer

In a hidden layer, each of the inputs $x_1, ..., x_n$ is multiplied with a corresponding learned weight $(w_1, ..., w_n)$ to scale it up or down to vary the effect on the output. Then these products are summed up to a single value ($w_1x_1 + w_2x_2 + ... + w_nx_n$ = Wx). A learned bias b is added ($Wx + b$) and this sum is passed through a so-called activation function g. Then we have:

$$\text{FFNN}(x) = g(Wx + b) \tag{2.1}$$

g is a non-linear differentiable function that is necessary for approximating nontrivial functions. g controls if, depending on the output value, the neuron should be considered as *activated* (that is, if it should contribute to the network's output) or not, or how high its *firing rate* (that is its contribution to the output) should be. The choice of g is an empirical decision (Goldberg, 2016). A common variant is for example the hyperbolic tangent $\tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$. Its graph is shown in Figure 2.3:

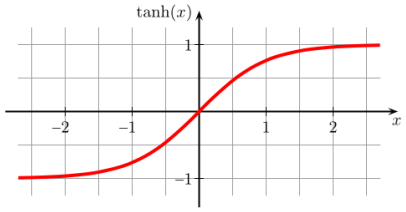

Figure 2.3: Graph of tanh activation function

It is visible in the figure that the values are bounded to the range $[-1, 1]$, but it can make learning hard when the values of the output are close to the limits -1 or 1. This is because the derivatives of the functions in the nearby regions are nearly zero and therefore their weights will not get updated. We will come

back to the relevance of the derivatives for the training process and about weight updates in the next subsection.

An alternative that solves this problem at least for positive values is the rectified linear unit (ReLU) that is currently very popular because of its fast convergence and computational efficiency. It is defined as:

$$\text{ReLU}(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases} = \max(0, x). \tag{2.2}$$

In the final layer of the feed-forward network, the output is created. In the case of a multi-class classifier, a so-called softmax function is often used to get a probability distribution over all dim_{out} possible outcomes by dividing the exponential of each input by the sum of exponential values of all inputs:

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{k=1}^{dim_{out}} e^{z_k}} \tag{2.3}$$

The probabilities are numbers between 0 and 1 and sum up to 1.

2.2.1 Training

The backpropagation algorithm for the training of neural networks was popularized by Rumelhart et al., 1986. It allows a training algorithm to adjust the weights and biases in the network in proportion to how much they contribute to the overall error in the network.

Backpropagation consists of a forward and a backward step. In the forward step, the input is propagated through the network and the output value is calculated.

Then, this value is compared to the desired output using a cost function that is a measurement of how much the actual output from the classifier differs from the desired output. A popular cost function for softmax outputs is the cross-entropy between the training data and the model's predictions. It increases when the predicted probability diverges from the actual label. It is calculated as follows:

$$C(o, p) = -\sum_{c=1}^{dim_{out}} y_{o,c} \log(p_{o,c}) \tag{2.4}$$

o is the current observation, $p_{o,c}$ the predicted probability if observation o is of class c and y is a binary variable that is defined as:

$$y = \begin{cases} 1 & \text{if } c \text{ is correct class for } o \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

The cost is then propagated through the network in backward direction from output layer to input layer to adjust the weights and biases with the aim to minimize the cost function. For this objective, in a procedure called gradient descent the partial derivatives of the loss function with respect to the learnable

parameters are calculated and the parameters are changed accordingly to reach a local minimum. The formula for parameters p at training step n is:

$$p_{n+1} = p_n - \gamma \nabla C(p_n) \tag{2.6}$$

$-\nabla C(p_n)$ indicates the direction for the steepest descent that is given by the partial derivative. The learning rate γ is a tunable parameter and indicates the size of gradient descent step in each training step.

There are many other optimizers than the classic stochastic gradient descent described above. A very popular choice is Adam (Kingma and Ba, 2014) with adaptive learning rates for each parameter because it has shown to be faster and more robust than classic stochastic gradient descent (Ruder, 2016).

The backpropagation process is repeated for many training examples and often for several training cycles.

2.2.2 Dropout

Overfitting to the training data is a frequent and serious problem in machine learning systems and also in neural networks. Dropout (Srivastava et al., 2014) is a technique that addresses this problem by randomly removing (*dropping*) units and all of their incoming and outgoing connections temporarily from the network during training.

The Dropout technique samples a thinned network for each training case by removing each unit with a probability p, which is called the Dropout Rate. Srivastava et al. show that the Dropout significantly reduces overfitting and performs better than other methods that prevent overfitting.

Figure 2.4 is an illustration of a two-layered FFNN, where the crossed units are units that have been dropped along with their connections.

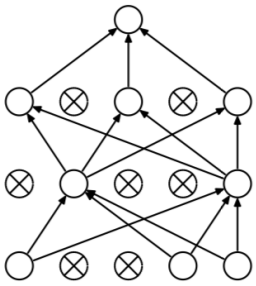

Figure 2.4: Srivastava et al., 2014: Dropout

Dropout does the forward and backward step for each training case on this thinned network and averages this exponential number of smaller networks with the equally weighted geometric mean of the predictions.

2.3 Recurrent Neural Networks

Recurrent neural networks (RNN, Elman, 1990) are designed to model arbitrarily sized sequences while preserving the order of features. In contrast to FFNNs

which process only one single item at a time, they can process a sequence of items with an internal temporal order.

This is especially significant for NLP as natural language consists of many types of sequences, like sequences of letters, of words and of sentences, and their order can be crucial as the meaning of a sentence can change when the word order is changed. For example, when doing a classification task with two classes: People who like coffee and people who dislike it, the sentence "I like tea but I hate coffee" would be classified differently if the words "tea" and "coffee" were swapped.

RNNs resemble FFNNs that perform the same task for every element of a sequence, with the output being dependent on the previous computations. The input to an RNN is an ordered list of vectors $x_1, ..., x_n$ and an initial state vector S_0. It returns an ordered list of state vectors $s_1, ..., s_n$ and an ordered list of output vectors $y_1, ..., y_n$. The input vectors are fed into the RNN sequentially and s_t and y_t are the state of the RNN after observing $x_{1:t}$, recursively defined by:

$$s_i, y_i = \text{RNN}(s_{i-1}, y_{i-1}) \tag{2.7}$$

The state vectors can be thought of as the memory of the network as they allow information to persist. Through them, the RNN has a feedback connection to itself as it has access to its output of previous time steps.

A simple extension to this idea is bidirectional RNNs (Schuster and Paliwal, 1997) that assume that the output does not only depend on previous but also on future time steps. To create a bidirectional RNN, two RNNs are stacked on each other and the hidden states of both RNNs are used to compute the output.

The backpropagation algorithm for FFNNs is modified to an RNN-specific version called Backpropagation through time (BTT), where the gradient depends on the calculation of the current but also of the previous time steps (Werbos, 1990). For this, the RNN can be thought of as „unrolled“ to a FFNN-like network with as many layers as the length of the sequence requires.

2.3.1 LSTM

A long short-term memory network (LSTM, Hochreiter and Schmidhuber, 1997) is a special type of recurrent neural networks (RNN) that has become popular for many types of NLP applications because of its better capacity to handle long-term dependencies that standard RNNs in practice cannot handle (Bengio et al., 1994), but that are frequent in natural language.

Although in theory RNNs could be able to make use of information in arbitrarily long sequences, the effect of an input becomes undetectable after only a few time steps. This issue is described in the vanishing gradient problem (Hochreiter, 1998) that states that the magnitude of the gradient decreases exponentially as the backpropagation progresses.

The difference of LSTMs to standard RNNs is in the computation of the hidden state. They use so called gate layers for information management within

the network. These layers are trained to keep information also over long distances if they are considered important for the output, while other information will be filtered out. The output of each step is regulated by them as well, so that only the most useful information is passed through the gate.

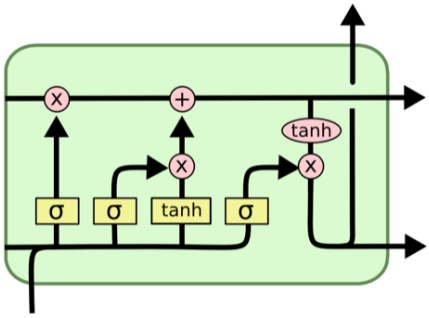

Figure 2.5: Olah, 2015: LSTM Architecture

The memory of the cell is represented in cell state C_t (t is the current timestep). Three gates control the cell state. Generally, a gate controls how mich of the current input should be written to or forgotten by the memory cell. It has the form of a layer with sigmoid activation and a pointwise multiplication operation, with values in the range $[0, 1]$. 0 means that no information is forwarded and 1 that everything is forwarded.

The three gates in the LSTM architecture are the input gate i_t, the output gate o_t and the forget gate f_t.

At first the forget gate is passed. It decides what information from the previous cell state C_{t-1} is discarded.

$$f_t = \sigma(W_f[s_{t-1}, x_t] + b_f) \tag{2.8}$$

Then, the input gate is passed. Its task is to decide which values should be updated.

$$i_t = \sigma(W_i[s_{t-1}, x_i] + b_i) \tag{2.9}$$

A tanh layer then creates a vector of new candidate values that could be added to the state. It is combined with to update the new cell state.

$$\tilde{C}_t = \tanh(W_C[s_{t-1}, x_t] + b_C) \tag{2.10}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.11}$$

Finally, the output gate is passed. It creates a modified version of the cell state by deciding which parts of C_t should be forwarded to the next cell. tanh is then applied to bring the values in the desired range between $[-1, 1]$:

$$o_t = \sigma(W_o[s_{t-1}, x_t] + b_o) \tag{2.12}$$

$$s_t = o_t * tanh(C_t). \tag{2.13}$$

This rather complex architecture archieves that the cell can memorize values over arbitrary periods of time, and thereby the performance is often improved over a standard RNN (Goodfellow et al., 2016).

2.4 Convolutional Neural Networks

Convolutional neural networks (CNNs, LeCun et al., 1989) are another type of neural networks that is in one of the systems studied in this work used for a character embedding. A CNN consists of a stack of layers. Its core component are the convolutional layers that make use of filters with learnable weights and biases to detect features. A set of filters moves over each part of the input, computing the dot product between the entries of the filter and the input. As each neuron in the output of the convolutional layer is only connected to a certain region of the input layer, local patterns are enforced.

A following layer, for example a ReLU layer, will apply an activation function to every element.

Afterwards, there is commonly a pooling layer that reduces the dimensionality of the representation, what prevents overfitting to the training data. The most common pooling procedure is the so-called max pooling outputs the maximum of the values of a window. A following layer, for example a ReLU layer, will apply an activation function to every element.

After usually several convolutional and pooling layers, a fully connected layer is used to compute the class scores.

A substantial advantage of the CNN architecture is their ability to focus on the relevant parts using a compact representation. Although they originally assume images as inputs to the network, they have also shown to perform well on some NLP problems(Goldberg, 2016).

2.5 Attention

Attention mechanisms were proposed by Bahdanau et al., 2014, who firstly used one in a Neural Machine Translation system, addressing the problem that LSTM still only partly removed the vanishing gradient problem as all necessary information needs to be stored in a fixed-length vector. This makes the translation harder the longer the sentences get. Attention adds a search over the source sentence to get the positions where the relatively most relevant information for this particular output position is concentrated. With this source position information and the previously generated target words, the model predicts a target word. So, it „attends“ to parts that it has learned are most relevant.

The attention weights α_{ij} express how relevant each input state is. They are defined as:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{2.14}$$

with the „associated energy“ $e_{ij} = a(s_{i-1}, h_j)$ being a score on how well the input at position j and the output at position i match.

The context vector c_i is a weighted sum of the input states:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{2.15}$$

Together with previous hidden state s_{i-1}, it is used to compute the hidden state s_i.

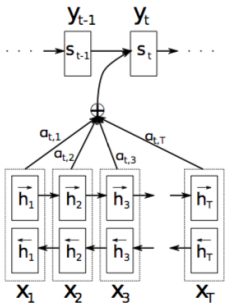

Figure 2.6: Bahdanau et al., 2014: Attention Architecture

In figure 2.5, the simplified architecture is shown: Attention weights $a_{t,i}$ are assigned to each state h_i and are assembled as a weighted sum that is then used to compute the new hidden state s_i.

This attention mechanism is often referred to as *soft attention* to distinguish it from other attention approaches, like for example the local attention mechanism by Luong et al., 2015 and hard attention by Xu et al., 2015.

Referring back to and searching over the input sentence increases the computational resources needed, but attention has lead to better-performing results on various NLP tasks and therefore rapidly become popular in NLP.

3 Coreference Resolution

Coreference resolution is the task of determining which mentions in a discourse unit refer to the same real-world entity. It is a clustering task, where mentions $m_1, ..., m_n$ are partitioned into a sequence of clusters $X^{(1)}, ..., X^{(M)}$ so that all mentions in one cluster refer to the same entity. Commonly, each mention is only assigned to one cluster in a valid clustering. Coreference systems typically search for the best clustering using a scoring function (S. Wiseman et al., 2016).

Coreference resolution has been studied since the 1960s and its methods moved from heuristic approaches to machine learning approaches in the 1990s. However, the winner of the CoNLL 2011 shared task on coreference resolution („modeling unrestricted coreference in OntoNotes“) was still a pipeline of deterministic hand-engineered sieves (H. Lee et al., 2011, Raghunathan et al., 2010). All state-of-the-art pre-neural machine learning approaches depended on a large set of features like string-matching, discourse-level, lexical, morphological, syntactic and semantic features. They typically also relied on syntactic parsers to determine the NPs, although the performance of training a „direct“ mention detector, learned jointly or in a pipeline, turned out to be better. (Ng, 2010)

The following section 3.1 will give an overview about the common system architectures for machine learning approaches to coreference resolution. Section 3.2 will then introduce and compare the recent neural models for coreference resolution, with a detailed description of K. Lee et al., 2017's approach in section 3.2.1.

3.1 Machine Learning Approaches to Coreference Resolution

There is a variety of approaches to Coreference Resolution using statistical learning methods, the most common being the Mention Pair Model, the Mention Ranking Model and the Entity Mention Model.

The Mention Pair Model is based on a classifier over pairs of mentions m_1, m_2 that in independent pairwise decisions tries to find out whether a nominal phrase (NP) is a new entity or a mention of another entity. The objective is to maximize the probability of coreference link between two mentions argmax $L_{coref}(m_i, m_j)$. The clustering is done as a second, separate machine learning task after making all pairwise decisions.

It was first proposed by Aone and Bennett, 1995 and McCarthy and Lehnert, 1995.

The Mention Pair Model has become popular mainly because of its simplicity, but has several drawbacks. Firstly, it does not necessarily produce transitive relations, but transitivity is a core property of coreference relations: When mention a is coreferent with mention b, and mention a is also coreferent with mention c, then mention b and mention c will necessarily also refer to the same entity..

Secondly, it provides no information about how good an antecedent is relative to others. Thirdly, performing such an two-step approach (classification and then, separately, clustering) is considered undesirable in machine learning as the classifier is not optimized for cluster-level accuracy, and finally, the information of two NPs is often not sufficient for an informed decision as it lacks global information. (Ng, 2010)

The Mention Ranking Model addresses the problem that the Mention Pair Model does not provide information about an antecedent candidates likeliness compared to others and the transitivity problem. In this model, all candidate antecedents are ranked in order to find the most probable candidate. The clusters are then built with transitivity. (Ng, 2010)

The Entity Mention Model is a classifier for a pairwise decision if a mention m belongs to a cluster X. It addresses the expressiveness problem of the Mention Pair Model by using a global scoring function in order to find argmax $L_{coref}(m, X)$. (Ng, 2010) But though being theoretically the more convincing approach as global information is often crucial, its performance was never convincing with linear classifiers and stayed behind that of the Mention Pair or Mention Ranking Model. This failure to outperform the local methods can possibly be due to the difficulty to find discrete, fixed-length features on clusters as the clusters can have very different sizes, so the representations tend to be too coarse or too sparse to use the information properly (S. Wiseman et al., 2016).

Beside those three most traditional methods, more approaches have come up in the recent years. The Cluster Ranking Model is an entity-based version of the Mention Ranking Model that, like the Entity Mention Model, attempts to find argmax $L_{coref}(m, X)$, but by ranking the preceding clusters. Agglomerative Clustering starts with each mention as its own cluster and repeatedly tries to merge clusters. The Partition-based Model ranks the possible partitions to find the most probable one. (Ng, 2017)

3.2 Neural Networks for Coreference Resolution

The application of Neural Networks have 2015 first lead to state-of-the-art results in coreference resolution and since then even better results have been reached.

The first state-of-the-art performance was reached by S. J. Wiseman et al., 2015, who implemented a mention ranking model based on a feed-forward neural network that used only raw features. A year later they augmented their local mention ranking system with entity level information running an LSTM over candidate antecedent clusters and thereby significantly outperforming state of the art. This addressed the paradoxon that the benefit of global information could not be seen in coreference resolution systems with the previous machine learning methods, that state of the art was always reached with local-only methods. Their system is trained end-to-end. The state of the cluster is learned as the hidden state of an LSTM over each sequence of mentions belonging to a cluster. (S. Wiseman et al., 2016)

K. Clark and Manning, 2016 proposed the first direct, „real“ clustering model with neural networks, using the agglomerative approach. With a cluster-pair representation, the feed-forward neural network learns when combining two

clusters is desirable by using learned, continuous features. The system iterates through a document and first merges the clusters with the highest score (easy-first). It assumes that the mentions have already been extracted in a preprocessing step, and a small set of hand-crafted features was crucial for the performance of their model.

3.2.1 Lee et al. (2017) Coreference Resolution System

Both the S. Wiseman et al., 2016 and the K. Clark and Manning, 2016 system used external syntactic parsers for head word features and mention detection. 2017, a new state of the art system implemented by K. Lee et al., 2017 without any external resources. The model jointly learns to detect mentions and to cluster them. All possible texts spans up to a maximum length are assigned an antecedent or a dummy antecedent ϵ, and the spans connected by these decisions then get clustered.

They used pre-trained word embeddings from GloVe and Turian, which is a shallow window-based method (Turian et al., 2010), and character embedding that they trained with a convolutional neural network.

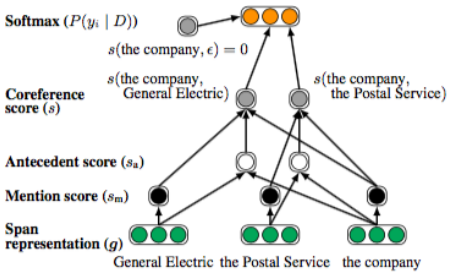

Figure 3.1: System Architecture (K. Lee et al., 2017)

A bi-directional LSTM is used to encode each word in its sentence context. As head features are crucial for the performance of coreference resolution systems, they also introduce the soft head word vector $\hat{x}_i$ that is created with attention over the words of each span. The LSTM hidden states for the first and the last word of the span are then concatenated with $\hat{x}_i$ and a feature vector $\phi(i)$ encoding the size of span i:

$$g_i = [x^*_{START(i)}, x^*_{END(i)}, \hat{x}(i), \phi(i)] \tag{3.1}$$

Using this representation, the mention score $s_m(i)$ for i is calculated:

$$s_m(i) = w_m \cdot \mathrm{FFNN}_m(g_i). \tag{3.2}$$

The antecedent score $s_a(i,j)$ for each pair i,j of mentions with $1 \leq j \leq i-1$ is defined as:

$$s_a(i,j) = w_a \cdot \mathrm{FFNN}_a([g_i, g_j, g_i \circ g_j, \phi(i,j)]), \tag{3.3}$$

where $\phi(i, j)$ is a feature vector encoding speaker and genre information and the distance between the two spans and the span width, $\cdot$ denotes the dot product and $\circ$ denotes element-wise multiplication.

The coreference score is then:

$$s(i,j) = \begin{cases} 0 & \text{for } j \neq \epsilon \\ s_m(i) + s_m(j) + s_a(i,j) & \text{for } j = \epsilon \end{cases} \tag{3.4}$$

Finally, a softmax layer is used to get a probability distribution over the candidate antecedents.

As this span-wise scoring is computationally highly expensive, they prune the mention candidates aggressively by only considering spans up to a maximum length of 10 and, after having calculated their unary mention score, keeping only the $0.4T$ (T being the number of words in the document) spans with the highest mention score. They keep only 250 antecedents for each span.

In their evaluation, they show that they outperform previous systems in all explored coreference resolution evaluation metrics. They point out that this comes from the end-to-end-training of their system that improves the recall by not relying on a mention detection system and thereby avoiding error propagation.

In a qualitative analysis, they show that their model can detect long and complex noun phrases and that the head word detection works well so that model can pay attention to the most relevant parts of the mentions. As a weakness they point out the fact that the word embeddings lead to false positive links. For example, *pilots* and *flight attendants* have similar word embeddings so that these non-coreferent phrases are mistakenly linked together. Also, their model cannot perform well on decisions that require world knowledge, which is not surprising though as it does not include external knowledge sources and the training data was too limited.

4 Language Models

Statistical Language Models (LM) are probability distributions over sequences of words. Commonly they are calculating the probability of the next word w_t in a sequence dependent on the previously observed words:

$$P(w_t \mid w_1, \ldots, w_{t-1}), \tag{4.1}$$

and for a sequence of m words:

$$P(w_1, \ldots, w_m) = \prod_{t=1}^{m} P(w_i \mid w_1, \ldots, w_{t-1}). \tag{4.2}$$

The classical statistical language model is the n-gram model. An n-gram model assumes that this probability can be approximated with looking at only the last n words:

$$\prod_{t=1}^{m} P(w_t \mid w_1, \ldots, w_{t-1}) \approx \prod_{t=1}^{m} P(w_t \mid w_{t-(n-1)}, \ldots, w_{t-1}) \tag{4.3}$$

The n-gram model in its easiest form calculates the conditional probability with frequency counts:

$$P(w_i \mid w_{t-(n-1)}, \ldots, w_{t-1}) = \frac{\text{count}(w_{t-(n-1)}, \ldots, w_{t-1}, w_t)}{\text{count}(w_{t-(n-1)}, \ldots, w_{t-1})} \tag{4.4}$$

But in practice, a smoothing method that is used to handle unobserved n-grams is crucial for the performance of the LM. A common choice is to work with back-off models that recursively looks at the probability predicted using a smaller context size (Katz, 1987). The very effective approach by Kneser and Ney, 1995 adds the likelihood for the w_t to appear in unfamiliar n-gram contexts.

While n-gram language models are a simple and take relatively little training time, they have some drawbacks, like taking more time for prediction once they are trained and not being able to make use of similar n-grams. Also, the history of a word is limited. Therefore neural networks, particularly recurrent neural networks, have also become popular for language modeling, and lead to superior performance on large data sets (Mikolov et al., 2011). In section 4.1, we will describe neural approaches to language modeling. Section 4.2 describes the inclusion of coreference prediction in the language model, particularly by the models by Yang et al., 2016 in section 4.2.1 and by Ji et al., 2017 in section 4.2.2.

4.1 Neural Networks for LM

Like n-gram models, neural language models (NLM) use the context to predict the word at timestep t: $(w_t|\text{context}) \forall t \in V$. NLM address the n-gram data sparsity issue and, like for other problems, make use of dense word embeddings. NLMs solve the problem of unknown n-grams implicitly and always estimate the probabilities based on the full history.

The first NLM was proposed by Bengio et al., 2003 who used a feed-forward neural network. The objective of their model is to learn a function f with

$$f(w_t, ..., w_{t-n+1}) = P(w_t|w_1, ..., w_{t-1}). \tag{4.5}$$

f is composed of two layers: a linear layer that creates a mapping C from every word i in the vocabulary V to a vector $C(i) \in \mathbb{R}^m$ that represents the destributed feature vector for the word, and a tanh layer for the conditional probability distribution g over these words. g maps an input sequence for words in context $C(w_{t-n+1}), ..., C(w_{t-1})$ to a conditional probability distribution over words in V for the next word

$$f(i, w_{t-1}, ..., w_{t-n+1}) = g(i, C(w_{t-1}), ..., C(w_{t-n+1})). \tag{4.6}$$

But NLMs did not get popularized until Mikolov et al., 2010 proposed an RNN-based LM. They finally outperform all count-based n-gram language models (Mikolov et al., 2011) and RNNs lead repeatedly strong results for LM in the following years (Goldberg, 2016).

The main drawback of the FFNN approach was its continued reliance on a specific size of the context, which made their gains over the simpler and more efficient traditional n-gram models limited. Recurrent neural networks in contrast made the history length theoretically unlimited and could compress the history in a low-dimensional space. However, going beyond 6-grams was still difficult (Mikolov et al., 2011). Sundermeyer et al., 2012 applied LSTMs instead of standard RNNs and showed that the performance could be improved by them.

Newer approaches include character-based NLMs, like in Kim et al., 2016 who reached state-of-the-art performance in English and outperformed their baselines in many morphologically rich languages (that often benefit from character-level features). They used a convolutional neural network for subword information and then an RNN-LM without any additional features like morphemes that earlier character-based approaches included.

4.2 Modeling Reference in Language Models

In the last years the idea appeared that language models could be enriched with explicit coreference information, particularly by Yang et al., 2016 and Ji et al., 2017. They follow the idea that entities are a very central element in a coherent text and that therefore their generation should be an explicit decision rather than forming them just like any other word. Both models save a representation of the already created entities and, whenever a new entity mention is formed, decide if it is one of the known entities. Yang et al's model, which we will refer to as the DeepMind model as it was developed at the DeepMind Technologies

company, is described in section 4.2.1. Section 4.2.2 provides details about Ji et al's model that is also called EntityNLM.

4.2.1 Yang et al. (2016) : The DeepMind Model

Yang et al., 2016 were to our knowledge the first to develop a Language Model that includes explicit coreference decisions. They treat references in a text as explicit latent variables.

Their approach is designed for three application areas: the modeling of references to a list (as in the application area recipe generation, where a recipe is generated from a list of ingredients), to a database (as in a dialogue system where the system answers a user's questions using a table that refers to restaurants and their attributes) and to a document context. The last area is the language model that includes coreference modeling. We will now have a closer look at the procedures in this third area.

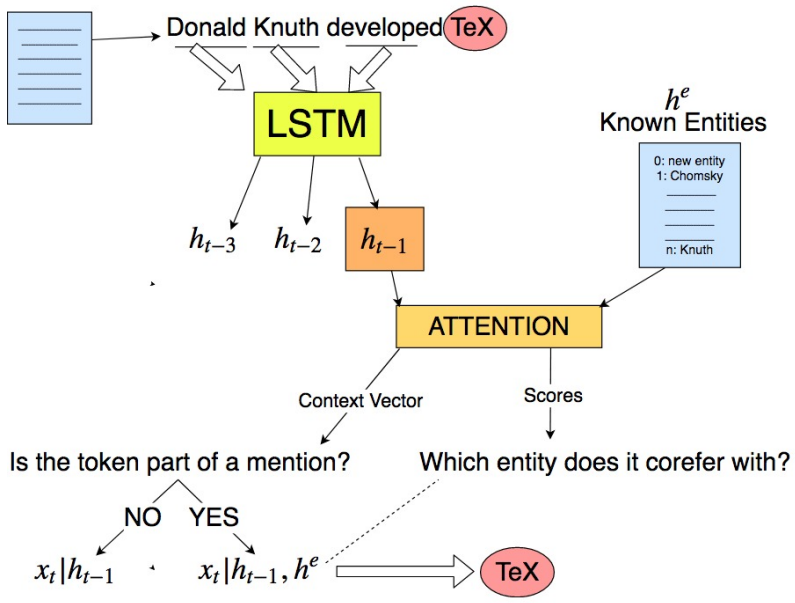

Figure 4.1: (Simplified) Illustration of Yang et al. (2016)'s model

Yang et al. define each document X as a series of tokens $x_1, \ldots, x_m$. The document is modeled with an LSTM that is defined as:

$$h_t = \text{LSTM}(x_t h_{t-1}). \tag{4.7}$$

A set of entities $h^e = h_0^e, h_1^e, ..., h_M^e$ is introduced, where h_j^e is the state of entity j, to store the entities that have already been observed in the document. The representation of an entity in this set is the LSTM hidden state representation of its last mention.

They also define the variable z_t at each position t that controls whether the corresponding x_t is an entity or not, and if it is, which entity it refers to. For this decision, an attention mechanism is with over the hidden state of the LSTM

over the set of observed entities h_e to predict the probability if the word is a mention of one of the entities:

$$p^{coref}(v_t|h^e, h_{t-1}) = \text{ATTN}(h^e, h_{t-1}) \tag{4.8}$$

Then the weighted sum with the attention weights is calculated:

$$d_t = \sum_{v_t} p(v_t) h^e_{v_t} \tag{4.9}$$

This sum is then used to calculate the conditional probability for z_t :

$$p(z_t|h_{t-1}) = \text{sigmoid}(W[h_{t-1}, d_t]) \tag{4.10}$$

If $z_t = 1$ (i.e. the word is an entity mention), the probability for the next word is calculated based on v_t (see Equation 4.8), the previous hidden state of the LSTM and the set h^e:

$$p(x_t|v_t, h_{t-1}, h^e) = \text{softmax}(W_1 \tanh(W_2[h_{t-1}, h^e_{v_t}])) \tag{4.11}$$

and if $z_t = 0$ (i.e. the word is no entity mention) then the next word is predicted based on the previous hidden state of the LSTM only:

$$p(x_t|h_{t-1}) = \text{softmax}(W_1 h_{t-1}) \tag{4.12}$$

Now that the word has been formed, in case $z_t = 1$ the entity state update is performed. If $v_t = 0$ (i.e. the next word is a new entity), h_t is appended to h^e. If $v_t > 0$, they set $h^e[v_t] = h_t$.

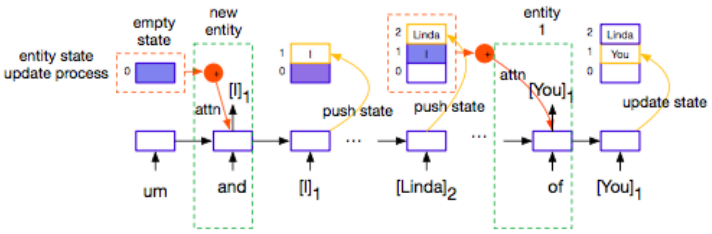

Figure 4.2: Yang et al. (2016): LM Architecture

The preprocessing is notable as they ignore entities with only one mention and reduce multiple-token entities to the word that is most frequent among the mentions. They also initialize the weights of their model with weights learned from a traditional RNN LM. Their final model outperforms their baseline with a traditional RNN LM with a notable difference in entity tokens and a slight difference in non-entity tokens.

They do not include any ablation studies or further error analysis in their paper.

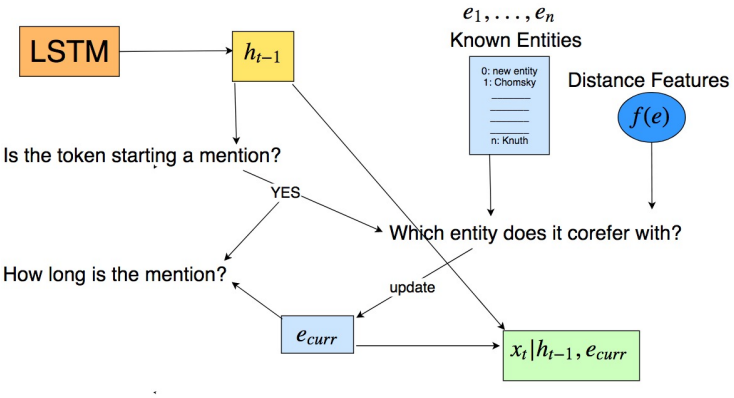

Figure 4.3: (Simplified) Decision Steps in Ji et al. (2017)

4.2.2 Ji et al. (2017): EntityNLM

Ji et al., 2017 use a generative language model for three tasks: language modeling, coreference resolution and entity prediction. We will focus on the LM task.

Like Yang et al., 2016, they model their document X with the tokens $x_1, ..., x_m$ with an LSTM:

$$h_t = \text{LSTM}(x_t h_{t-1}). \tag{4.13}$$

The hidden state is used to compute R_t, a random variable that indicates if x_t is an entity (then $R_t = 1$) or not ($R_t = 0$), using $\mathbf{r}$, a parametrized embedding associated with r. So, for $r \in \{0, 1\}$:

$$p(R_t = r|h_{t-1}) = \text{softmax}(h_{t-1}^T W_r \mathbf{r}) \tag{4.14}$$

Then the question which entity the current word should correspond to is addressed. The random variable E_t denotes the index of the current entity in the set of known entities E_t in case $R_t = 1$.

The heart of their model are the entity representations that are continuously updated in order to save the entity's evolution. The new entity state $e_{e_t,t}$ is derived combining the previous representation $e_{e_t,t-1}$ and the hidden state h_t with the interpolation δ_t:

$$\delta_t = \sigma(h_t^T W_\delta e_{e_t,t-1}) \tag{4.15a}$$

$$u = \delta_t e_{e_t,t-1} + (1 - \delta_t) h_t \tag{4.15b}$$

$$e_{e_t,t} = \frac{u}{||u||_2} \tag{4.15c}$$

Embeddings for new entities are created with the learnable variable r_1 according to a normal distribution N that adds some variance to the representation, and then become part of to set E_t.

$$u \sim N(r_1, \sigma^2 I) \tag{4.16a}$$

$$e_{e'_t, t-1} = \frac{u}{||u||_2} \tag{4.16b}$$

For every known entity $e \in E_t$, the hidden state h_{t-1} and a distance feature vector $f(e)$ are used to predict the probability that $\mathrm{E_t}$ is e:

$$p(\mathrm{E_t} = e | R_t = 1, h_{t-1}) = \mathrm{softmax}(h_{t-1}^T W_{ent} e_{e_t, t-1} + w_{dist}^T f(e)) \tag{4.17}$$

L_t is a random variable indicating the remaining length of the mention including the current word. When $L_t = 1$, the last word in a mention is reached (or the current word does not belong to a mention). L_t is predicted with the last hidden state h_{t-1} and the most recent embedding of the entity e_t. The probability for lengths l is predicted as follows:

$$p(L_t = l | h_{t-1}, e_{e_t, t-1}) = \mathrm{softmax}(W_{len,l}[h_{t-1}, e_{e_t, t-1}]) \tag{4.18}$$

When $L_t > 1$, the prediction of R_t and E_t is obviously obsolete in the next time step.

The prediction of a word x is based on the LSTM hidden state h_{i-1} and the representation of the current entity, e_{curr}:

$$p(X_t = x | h_{i-1}, e_{curr}) = \mathrm{CFSM}(h_{i-1} + W_e e_{curr}) \tag{4.19}$$

CFSM is a class-factorized softmax function, that, in contrast to a standard softmax function, uses a two-step prediction with predefined word classes rather than the whole vocabulary.

If the word is not an entity ($R_t = 0$), e_{curr} simply stays the embedding of the most recently mentioned entity and is still used for the calculation.

In a preprocessing step, they remove enclosed mentions and only keep the outermost of any embedded mentions.

Ji et al. compare their model to a 5-gram language model and an RNN LM baseline and outperform them on the English section in the CoNLL 2012 shared task data set. Like Yang et al., 2016, they do not provide any ablation studies or qualitative evaluation. They do non evaluate their model on entity and non-entity tokens.

5 Theoretical comparison

In this chapter we will theoretically compare the approaches in the language models by Yang et al., 2016 and Ji et al., 2017. Additionally, we will compare them to the coreference resolution system by K. Lee et al., 2017 in order to work our architectural differences and to see which ideas can or cannot be adopted from a CR system. We will point out the most relevant differences and commonalities in their architectures and features in section 5.1 to 5.5, and finally discuss how we expect them to affect the results of the systems in section 5.6.

5.1 Mention Spans and Restrictions

A fundamental difference is the units they are looking at. In Yang et al., mentions only consist of a single word. The decision which word represents the mention in the training data is derived from statistics over all mentions of the entity, adding the mention tag only for the word in a mention that is most frequent among all mentions.

Ji et al.'s language model creates word for word, but considers mentions as spans: When a new mention is started at time step t, a random variable L_t is used to predict the number of words that will be created for the mention. When mentions are nested in other mentions in the training data, they only keep the enclosing mention.

Lee et al. also work with spans, and consider all spans in the text from single words to spans up to a maximum length as possible mentions. This procedure would not be possible for the language models that depend on reference decisions while the text is being produced and can therefore only consider the tokens that were already created and that are on the left side of the current token.

5.2 Feature Representation

All three models use an LSTM to model a word in its sentence context. In Lee et al., the LSTM is bi-directional, as it can use context from both sides of a token while the LMs only have access to the left-side context.

The Lee et al. coreference resolution system represents the spans with the first and last word of each span and adds a feature vector encoding the size of the span, speaker and genre information. It uses word embeddings and a character embedding created with a CNN to provide additional information about the words. The model creates a head-word feature using an attention mechanism over the span representations. This rich feature representation's usefulness is shown in ablation studies, but again, many of the features like the span heads, a mention's last word and the size of the span are not viable for the language models.

Ji et al. include a distance feature vector in the entity decision process. Yang et al. did not report the use of any explicit features.

5.3 Decision Steps

The set and order of the explicit decisions are different in the three models, and the variables that a decision depends on and the technical realization of the decision steps also differ fundamentally. For the two Language Models, please refer to Figure 4.1 and Figure 4.3 for simplified and easily comparable illustrations of the general architectures.

The question addressed first is generally comparable in the three models. For Yang et al., it is: *Is the word an entity mention?* while for Ji et al. it is: *Does the word belong to an entity mention?* and for Lee et al. who are looking for scores for two entity candidates as their task is coreference resolution it is: *Is span i a mention and is span j a mention?*.

The next question for the two LMs is: *Which entity does the word corefer with?*. In the Ji et al. model, this question can be obsolete when the word continues an already started entity, visible through the variable L_t, or if the model has decided that the word is not part of an entity mention. For Lee et al. the question is: *Is span i an antecedent for span j?*.

In the LMs, there is a third question, that is: *How to form the word based on this information?*. In Ji et al., the question *How long is the mention?* is also posed when a mention is started.

In the Coreference Resolution system, the next question is: *Which mentions can be grouped together to one cluster, i.e. one entity?*

In the Yang et al. system the first two questions are both handled with an attention mechanism with the LSTM hidden state over the set of observed entities. They create the probability distribution over the known entities and use the weighted sum for the decision whether the word is part of an entity mention. Ji et al. decide whether the word belongs to an entity mention solely based on the LSTM hidden state. The decision which entity it refers to is handled with the LSTM hidden state, the set of entities and a distance feature vector that incorporates the distance from the current mention to the closest mention from each previously mentioned entity.

Ji et al. sample the next word x_t from hidden state and the current entity embedding even if it is not an entity. Yang et al. sample it by using the hidden state and the entity representation in the entity set with the highest attention score if the word is an entity, and solely based on the hidden state otherwise.

The Lee et al. Coreference Resolution system uses the span representation to calculate a mention score with an FFNN for each mention and, independently, an antecedent score with another FFNN for pairs of mentions. The unary mention scores of both mentions and their antecedent score are added to calculate the coreference score. The most likely configuration is kept and clustered in the end.

5.4 Detection and Representation of the Entities

The representation for previously seen entities for the LMs is a set with a vector representation for each entity. Yang et al. use the hidden state of the last mention of the entity while Ji et al.'s representation for the entity is continuously updated in order to save the entity's evolution while the text is being produced. Their new entity state is derived combining the previous entity state representation and the hidden state h_t with an interpolation (see Equation 4.15).

The detection of a new entity is slightly different in the LMs. Yang et al. use a learnable variable h_0^e that is a virtual empty entity at the start of the set of previously seen entities h^e. Ji et al. also use a learnable parameter that becomes part of the set of known entities, but adds some variance to the entity embedding using a normal distribution.

In Lee at al.'s model, the set of possible assignments for each span y_t is all proceeding words $Y_t = \epsilon, 1, \ldots, t-1$ as well as all preceeding spans of these words. There is no explicit tracking of already known entities but a pairwise decision over spans and a clustering in the end. Spans that have not been attached to any other span get attached to a dummy antecedent ϵ.

5.5 Data and Preprocessing

Lee et al. and Ji et al. use the English benchmark data of the CoNLL-2012 package based on OntoNotes 5.0 (Weischedel et al., 2013), with slightly over a million words from newswire, broadcast news, broadcast conversation, web data and telephone conversations. They use the standard training/development/test split of the CoNLL task and Ji et al. use a vocabulary size of 10,000.

Yang et al. use a part of the Xinhua news dataset from the English Gigawords Fifth Edition Corpus with automatic coreference annotation (Parker et al., 2011), sampling 100,000 documents with a length between 100 and 500 words, and with 23 million tokens in total and a vocabulary size of 50,000. They split the data into training, validation and test set.

Yang et al. remove entities with only one mention and reduce mentions with multiple tokens to one word, the one that is most frequent among all mentions for the same entity. Ji et al. also remove the singleton mentions in texts. They remove embedded mentions, keeping only the enclosing entity mention.

5.6 Discussion

We note that many features that have proven to he helpful in Coreference Resolution systems cannot be applied in the Language Model systems as only the left part of the text is known in advance. Therefore less information is available for the decisions if the token is part of an entity mention and which entity the token belongs to. This can cause the results to be less accurate than in a Coreference Resolution system.

The mention representation as a single word by Yang et al. can be considered questionable, and reducing the annotated mention length to one is also declared an inappropriate assumption by Ji et al., 2017. But it could be an issue with the

Yang et al. model that their entity representation may profit from this decision, as they take only the hidden state of an entity's last mention as the entity representation without explicitly accumulating previous information in it. But it is possible that sufficient information is stored implicitly in the hidden state of the last word in a mention.

Including the current entity embedding even if the next word is not an entity as Ji et al. do could be a useful feature as entities are considered central in coherent texts (E. Clark et al., 2018). But again, the benefit of explicit incorporation of information in contrast to letting the model decide what it regards as relevant is not clear without an explicit evaluation. The same also applies for the distance feature vector in the Ji et al. model. But as coreference resolution systems commonly do include a distance feature despite also working with LSTM-based representations, we would expect a certain benefit.

The order of decisions is another major difference to pay attention to in the evaluation, especially the order of the questions if the next token belings to an entity mention and which of the entities it is as those are the central decisions that the model must make.

A big influence on the reported results of each model has the data set. While for Lee et al. OntoNotes is an appropriate choice as it is a Coreference Resolution system, it may be too little training data for a good Language Model. The Gigawords corpus contains much more training data, but it is automatically annotated, which could lead to errors induced by the coreference resolution tool that was used for the annotation. It will be necessary to evaluate the models on the same data to get insights to their comparative performance.

6 Experimental Setup

In this chapter, we will describe our experimental setup, with the OntoNotes 5.0 corpus and preprocessing of the data in section 6.1, the evaluation methods in section 6.2 and the implementation details for our models in section 6.3. Our baselines are described in section 6.4.

6.1 Data: OntoNotes 5.0

OntoNotes 5.0 (Weischedel et al., 2013) is a corpus with documents of various genres like news, broadcast, usenet newsgroups and conversational telephone speech. Besides English, it also contains Chinese and Arabic texts, but in this thesis we will only work with the English subset. The English corpus includes 1.5 million words and anaphoric coreference annotation within a document. Names, nominals and pronouns can have coreference tags, as well as verbs that are coreferent with a noun phrase. The coreference annotation has 90% consistency between different annotators (Hovy et al., 2006).

We will use the CoNLL-2012 split into training, validation and test set. We lowercased all tokens, replaced all numbers by a special symbol and all tokens with less than 5 occurrences with a special token for rare words, resulting in a vocabulary size of 11539. We keep only the embedding mentions where mentions are nested and removed all mention annotation where the mention length is higher than 25. The length distribution of the mentions in the OntoNotes training set with 25 tokens or less is plotted in figure 6.1:

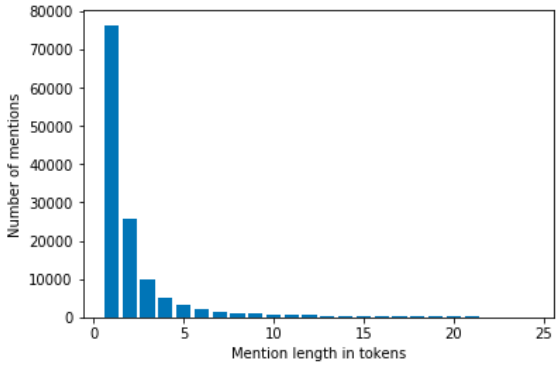

Figure 6.1: OntoNotes: Mention Length Distribution

It is visible that mentions with one token dominate the data, mentions with two tokens are still relatively frequent, but then the number is approaching negligible sizes rapidly, what lead us to the conclusion that very long mentions will not be useful for our models.

To be able to evaluate the mention length predictions of EntityNLM, we did not reduce the length of the mentions in the data for the DeepMind Model to one as in the original model but worked with the full mention length with up to 25 tokens as described above. Preliminary experiments did not show performance drops when training on the same data as EntityNLM.

6.2 Evaluation metrics and procedures

As the main metric for the language models' general performance, we measure the perplexity as described in section 6.2.1. We also evaluate the entity prediction process qualitatively as described in section 6.2.2 and measure precision and recall for the question if the next token is part of an entity mention and the accuracy for the choice of the entity from the set, and evaluate EntityNLM's length prediction. Finally, we describe a setting for an examplary qualitative analysis in a text generation task with focus on mention generation in section 6.2.3.

6.2.1 Perplexity

A model's perplexity is its degree of uncertainty when predicting the next word. Having a high perplexity means that the probability that the next word is correct is low, which means that the model is uncertain how to select the word. Vice versa, a low perplexity means a high probability and that the model is more certain. So, minimizing the perplexity is equal to maximizing the probability of a model, and a Language Model with a lower perplexity (on the same vocabulary) is considered the better model.

For probability distributions o and p, the perplexity is defined as

$$2^{C(o,p)} \tag{6.1}$$

where $C(o,p)$ is the cross-entropy of the distribution $p(X)$ over all words x in a test set X (see equation 2.4 in section 2.2.1).

We measure the perplexities in three settings: for all tokens in the test set, for entity tokens only and for non-entity tokens only.

6.2.2 Evaluating the Entity Mention Prediction

As the task of entity modeling in language models is different from the clustering task of coreference resolution, we do not measure the correctness of the prediction in the common measures for coreference resolution but evaluate the accuracy of the prediction at each time step, looking at the questions if the word is an entity, which entity it is and how the mention is formed.

For the question if the word is an entity, which corresponds to the variable z_t in the DeepMind model and R_t in combination with L_t in the EntityNLM, we measure precision and recall for the created entities of the model.

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \tag{6.2}$$

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \tag{6.3}$$

For the question which entity to choose out of the set of already created entities and the new entity embedding, we evaluate the accuracy of the corresponding variables. It is defined as the percentage of correct predictions:

$$\text{Accuracy} = \frac{\text{correct classifications}}{\text{all classifications}} \tag{6.4}$$

Finally, we evaluate EntityNLM's length prediction variable L_t. We measure the accuracy as defined above, the average distance of the incorrect predictions to the correct length of the mention and the averages of the correct and predicted lengths. For comparison, we also measure the average length of the mentions in the DeepMind reimplementation.

6.2.3 Text Generation

Besides quantitative measures, we perform a text generation task with the goal to continue a start passage of a news story in a coherent way. For this purpose, we chose 50 documents randomly from the OntoNotes validation set and cut them off at a point where a new mention is about to start, but at least after 100 and at most after 150 words. A (shorter) example could be:

> *[more than 200 overseas chinese and taiwan compatriots living in japan] gathered happily together tonight in the chinese embassy to celebrate the year of the goat. [ambassador zhenya yang] first extended [his] [spring festival greetings] to*

where the original document continues with: *[those overseas chinese and the taiwan compatriots].*

In our analysis, we pay attention to the following points:

- Is a new mention started at this point?
- Is it a mention of a previously seen entity?
- Does it refer to the same entity as in the original document?
- Does the mention make sense at this point?
- Does the form of the mention make sense?
- Does the form of the mentions of the same entity vary?

We will report our summarized observations on frequent phenomena in the generation task.

6.3 Implementation

We implemented all models in Python using the PyTorch deep learning library (Paszke et al., 2017). The implementations are publicly available at https://github.com/jekunz/entitynlm and https://github.com/jekunz/deepmind.

As the hyperparameter settings for the best models have not been reported in the papers, we experimented with different hyperparameters to optimize our re-implemented models. As candidate hyperparameters for the hidden size of the LSTM and word embedding layer, we tried the values 34, 64, 128, 256, 512. We employ dropout Srivastava et al., 2014 with candidate rates of 0.0, 0.1 or 0.2 and for the Adam optimizer Kingma and Ba, 2014, we tried the learning rates 0.01, 0.005 and 0.001. We tried the models with GloVe and with randomly initialized, learnable word embeddings. We also experimented with a weighted loss with the intention to force the models to produce more entity mentions.

Based on the experimentally results on the development set, we chose a hyperparameter setting for the DeepMind model with 64 hidden units for both LSTM hidden size and word embedding size, Adam optimizer with $\lambda = 0.005$, a dropout rate of 0.2 and randomly initialized word embeddings. The model was trained for 20 epochs.

The best hyperparameter setting for EntityNLM was very similar. We only used the model with 128 hidden units and after 22 epochs as it performed slightly better on the development set.

We found the structure of both models to be hard to train with its many variables. The different losses in the models and evaluation metrics were sensitive to hyperparameters but did not correlate with each other at tuning time, what made the choice of the best model non-trivial. We chose to take the perplexity (section 6.2.1) as the only criterion for the choice as a leading research question in this thesis is if the inclusion of explicit entity information can improve a language model, which we suggest can best be answered with the standard LM metric. Furthermore the other metrics oscillated heavily for different epochs on the development set while the perplexity turned out to be a more reliable measure.

6.4 Baselines

For the main evaluation metric that is the token perplexity, we implement a baseline model that is a purely LSTM-based LM. We use it in two settings: one in the same hyperparameter setting as the best DeepMind model with a hidden size of 64, trained for twelve epochs, and one optimized and very strong model with a notably bigger hidden size of 512, trained for three epochs, which we chose because it archived far better results on the development set.

7 Evaluation

In this chapter, we perform the evaluation as described in chapter 6.2. The perplexities are reported in section 7.1, the decision steps in the entity prediction process are evaluated in section 7.2 and the observations from the text generation task are reported in section 7.3.

7.1 Perplexity Results

We report the results for our models and baselines on the test set along with the original results from Ji et al., 2017 in table 7.1. Please note that we give the re-implementations access to the correct entity lists at test time in order to be able to evaluate the decision steps without being misleaded by cascading errors. The original results by Ji et al., 2017 did not have this access to gold entity information, so that the results are not directly comparable.

.	All Tokens	Entity Tokens	Non-Entity Tokens
RNN-64	121	177	114
RNN-512	96	126	88
EntityNLM Reported Res.	132	-	-
EntityNLM Reimpl.	131	154	127
DeepMind Reimpl.	107	132	101

Table 7.1: Perplexity results

Based on these results, we cannot confirm that the models outperform a simple RNN-LM. Both RNN-LM baselines easily outperform both the re-implemented and the reported results of EntityNLM, and the optimized baseline (RNN-512) also performs much better than the DeepMind model which though outperforms the RNN that has same hidden size like itself (RNN-64). Both baselines by far outperform the baselines that Ji et al., 2017 used in their evaluation.

The DeepMind reimplementation has by far the best perplexity of the models with entity modeling, outperforming both the reported (with the reserve that in contrast to the Ji et al. results, we provide gold entity lists) and the re-implemented results of EntityNLM.

The decision on how to select a token seems to be generally harder on entity tokens in the data set as they generally and for all models have a lower perplexity than non-entity tokens. But measured by their overall performance, the entity tokens in the LMs with explicit entity modeling are relatively better than in the other models. The ratio between the overall perplexity and the entity token perplexity can be seen in table 7.2. It is highest for EntityNLM with 0.85, followed by the DeepMind Model with 0.81 and the RNN with 512 Hidden Units with 0.76.

.	Perplexity Relation
RNN-64	0.68
RNN-512	0.76
DeepMind	0.81
EntityNLM	0.85

Table 7.2: Relation Perplexity All Tokens / Entity Tokens

These results must be seen with the constraint that the models with explicit entity modeling have access to gold entity lists, which can be an advantage.

7.2 Entity Prediction Process

The precision and recall for the question if the next token is an entity and the accuracy of the choice of the entity from the list are reported for both reimplementations in table 7.3:

.	Entity Y/N Precision	Entity Y/N Recall	Entity Choice Acc.
DeepMind	60.9%	30.2%	65.8%
EntityNLM	39.0%	53.9%	67.8%

Table 7.3: Entity Prediction Results

As the models are optimized for perplexity only, these results would possibly have been better in other hyperparameter settings. We observed higher values on the development set during tuning and great oscillations for different epochs which makes it hard to interpret specific results.

The precision and recall of both models are low, suggesting that the question if the next word is an entity seems to be the most challenging point in the models.

The decision which entity to choose when the decision if the next token is an entity is given is performed with an accuracy of 67.8% for EntityNLM and 65.8% for the DeepMind model. So the EntityNLM model outperforms the Deepmind Model by 2%. These results must not be overinterpreted as these entity prediction results, as mentioned before, were true snapshots and varied strongly.

EntityNLM's length prediction is correct in 59.4% of all cases, with the average distance of the false predictions to the gold mention length being 2.85 tokens. The average length of the entity mentions in both models as well as the gold length is reported in table 7.4:

.	Average Length
GOLD	2.25
DeepMind	1.43
EntityNLM	1.53

Table 7.4: Average Length Prediction

Both models tend to create (too) short mentions, with an average length of 1.43 and 1.53 tokens. Mentions longer than five tokens were never created by the models, but are also very uncommon in the training data as shown in Figure 6.1.

7.3 Text Generation Task

The 50 generated samples for each model reflect the low recall of the systems and rarely start a mention at the starting point: in 17 samples of the DeepMind Model and 12 of the EntityNLM model, the first token is an entity token. In 13/17 cases, the DeepMind Model chooses the right entity. For the EntityNLM model, this is the case in 7/12 examples.[1]

Exactly the same representation for the entity as in the original text is only chosen once by EntityNLM, where a new sentence is started with the word *I*. The DeepMind model never chooses the same repreentation. But, of course, there is generally a high randomness in a created text when sampling so that sampling the next word exactly as in the original text is an unlikely event.

Two of DeepMind's samples continue with an entity representation very close to the original one. An example for this is a text about the Bank of Montreal that is referred to as *the bank* by the original span, while the sample creates *grand bank*.

In all models, the networks' dense word representations lead to the confusion of noun phrases that have certain commonalities (for example both being a country, or a president) but refer to different entities in the real world. Both models made mistakes like referring to an entity that was previously referred to as *Mexico* with *Taiwan*, or to *Putin* with *Clinton* and with *Shimon Peres*. So, when starting a new entity mention and having the choice between a already started similar entity and the embedding for a new entity, the models often tend to choose the existing similar entity.

Similarly, the DeepMind model refers to *peng li , premier of the state department* with *the kingdom representative*, where the original text continues with *peng li*. This generally makes sense despite the fact that the system lacks the world knowledge that China is not a kingdom anymore. The model also refers to *the new pudong region including the <-UNK-> bonded area* with *china* and to *a temporary increase in the state 's sales tax* with *its shares*, which in both cases can be seen as a similar category but is obviously not the same. EntityNLM refers to the previous mention *palestinian authority president yasser arafat* with *yugoslavia*, which we also relate to the word representations as both Arafat and Yugoslavia often occur as conflict parties in the texts.

The pronouns are used wrongly in some cases. EntityNLM refers to *us* with *he* and to *the avenue of stars* with *they*. The DeepMind model refers to *countries such as cuba , yemen , etc.* with *he*.

Interestingly, it is the EntityNLM model that in one example refers to the earlier mention *myself and an italian woman* with *she*. Although EntityNLM has a sophisticated update for the entity representation, it still seems to be mislead by the last word (*woman*) of the mention in this example.

Generally, our qualitative impression of the errors of both models from the text generation task is similar. The errors are of the same categories, particularly choosing wrong pronouns and mention representations of related but different entities in the real world.

[1] Please note that the TGT evaluation for EntityNLM has been performed with an earlier, erroneous version of the re-implemented model and will be subject to revision.

8 Discussion

While we cannot confirm that the incorporation of explicit entity information is helpful in a general language modeling task, and the models' abilities especially to predict where to form entities have shown to be limited, we see a potential for the continuous representations for entities to become better and to be useful in different kinds of tasks. We will discuss the architectural details and the features of the two models with respect to their contributions to a successful result in section 8.1, implementation and training aspects in section 8.2 and the impact of the data in section 8.3. In chapter 8.4, we will discuss our expectations on the models' performance and utility in real-world settings.

8.1 Architecture and Features

A main challenge in both models is the difficulty to find a training setting that performs well for all subtasks. The different losses did not correlate with each other, causing them to interfere. We would find it desirable to develop models with a simpler architecture that contain fewer losses, which are therefore easier to train, more robust and would possibly perform better.

The two models' results for the entity prediction suggest that handling the questions if the word belongs to an entity mention and which entity it is jointly is preferable over first deciding if the word belongs to an entity mention. The list of entities seems to be a helpful context for the question if the next word is an entity. As the question if the next token is an entity is by far the biggest error source, it will lead to relevant error propagation in real-world applications. Optimizing this prediction step is therefore an important challenge. Its placement in the model should be evaluated, with possible variants before, during or after the choice of the entity, and if it is necessary to address this question explicitly with its own loss.

We did not find evidence that EntityNLM's continuous entity update procedure for the representations is preferable over taking the last mention's hidden state as in the DeepMind model, although this idea still seems plausible in theory. It is, however, possible that the LSTM hidden state alone already contains sufficient information about earlier mentions.

We found that the original idea by Yang et al., 2016 to reduce all mentions to one token did not improve the results on the development set, and we find it undesirable also in theory. The model is able to continue an already started mention as well as EntityNLM with its length prediction. We therefore suggest that the length prediction is not crucial for a well-performing model. Both systems tend to create shorter mentions than in the gold predictions, and are failing at creating long ones. This is comprehensible as long mentions are most probably also very complex mentions, and they are rare in the training data what makes it hard for the model to learn them properly.

The dense embeddings lead to the confusion of similar but different real world entities, as shown in the text generation task. As stated in the background chapters, this is a common problem in neural network based NLP systems and also lead to wrong mention clusterings in in coreference resolution systems, and it is never completely avoidable. But as these errors have shown to be very frequent in the generation of mentions, it is still a problem that needs to be addressed.

8.2 Implementation and Training

For EntityNLM, the perplexity for our re-implementation is slightly netter than the reported results by Ji et al., 2017, with a perplexity of 131 instead of 132.

For the DeepMind Model, we were not able to produce directly comparable results as Yang et al. did not specify the exact training data that they have selected randomly from a English Gigawords (Parker et al., 2011) subset. Our results on OntoNotes compared to EntityNLM and the baselines show a high plausibility that our re-implementation is performing very well.

Unsatisfactorily, we did not find a setting where the models can profit from more hidden units. A high number of hidden units resulted in a great performance boost for the RNN LM, which we were not able to use for the models. We find it promising that with 64 hidden units, the DeepMind model performs noticeably better than the RNN LM, but this effect is not scalable for greater hidden sizes where the RNN LM gets much better perplexity results. The relatively better performance of both models on entity tokens compared to the baselines also suggest that taking profit from the models could be possible.

8.3 Data

Short texts that are mostly news texts are probably not the genre that takes most profit of explicit entity information. It is possible that in longer texts or texts with complexly interacting characters that develop in the text, a language model would take greater profit from explicit entity modeling and from features like the entity update procedure in EntityNLM.

While OntoNotes 5.0 is an appropriate data set for coreference resolution, it is common to use much larger data sets for language modeling as the general LM task does not need further annotation. Yang et al., 2016 used a subset of the English Gigawords Fifth Edition corpus (Parker et al., 2011) that is annotated automatically. But as automatic annotation still has an error rate notably above disagreement between human annotators, even when annotated with state of the art systems like the one by K. Lee et al., 2017, cascading errors are expectable. A training strategy that combines the general language modeling capability of a large corpus and the high-quality annotation of OntoNotes could be worth trying.

8.4 Utility in Real-World Settings

We regard our two model's utility in their current form in real-world settings as limited. All results must be seen under the reserve that gold entity lists

were provided to the models during testing, so that in a real-world setting the models would suffer from error propagation in the entity prediction process that we eliminated in our experiments. The models contain various steps with performance far from perfect, resulting in much incorrect information coming from the entity prediction process. We must expect that the models will be provided with a high amount of possibly harmful erroneous information in a real-world setting.

The models' performance as a language model in terms of perplexity is below the performance of an optimized RNN-LM which is much faster to train and very simple. The performance of the two models on entity tokens is relatively improved, but especially the results of EntityNLM must be treated with caution as entity tokens are equipped with gold information, and as it could be due to the generally high perplexity results of EntityNLM that the difference between entity and non-entity tokens decreases. The perplexity is a metric that grows exponentially, which makes the ratio only comparable to a limited extend especially when the values differ widely.

But despite the limitations of the current models, we regard it as worthwhile to invest in improvements, especially in the development of models that are less prone to error propagation. Acquiring more high-quality annotated training data would most probably result in more useful models that may outperform a purely RNN-based model.

9 Conclusion

We compared, re-implemented and evaluated two recent approaches to the new idea to incorporate explicit entity decisions in an RNN-based language model. While the stronger, well-optimized one of the RNN-baselines has a better overall performance on our data set, we showed that both language models with explicit entity information have an improved relative performance on entity tokens in relation to all tokens in the data set at least when having access to gold entity information. Therefore we suggest that there can be a potential to improve a language model with explicit entity decisions.

The implementation of the DeepMind Model by Yang et al., 2016 performs clearly better in the perplexity metric than the implementation of the EntityNLM model by Ji et al., 2017. It is challenging to find a good training setting where all of the various losses in the models can reach their optima and where both perplexity and entity prediction metrics reach good results. This affects both models but is even harder for the EntityNLM model with one more loss than the DeepMind model, which may explain the difference in perplexity performance.

As we compared our own re-implementations of the two systems, the evaluated systems may differ from the original implementations. However, we were able to reveal crucial aspects for the performance of the systems.

We showed that the most challenging point in the systems is the decision if the next token is an entity token. This decision has a low precision and recall in both systems, suggesting that future research should pay particular attention to possible improvements at this point. In contrast, the decision which entity the next token refers to has a comparatively high accuracy.

We suggest extensive ablation studies of the two models in future work that evaluate if any of the losses are dispensable and can therefore be dropped for easier training. In particular, we suggest that the fusion of the decisions if the next token is an entity and which entity it is is preferable over a pipelined approach. Our results also indicate that EntityNLM's length prediction variable is not necessary, and that it is sufficient to save the last hidden state as the representation of the entities rather than following a sophisticated update of the representations as done by EntityNLM. A combination of evidentially useful features of both models may result in a better model.

Another crucial aspect for further improvements is acquiring more high-quality annotated training data, and possibly data from other genres where complex interactions of entities play an even more important role.

Bibliography

Aone, Chinatsu and Scott William Bennett (1995). "Applying machine learning to anaphora resolution". In: *International Joint Conference on Artificial Intelligence*. Springer, pp. 302–314.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural machine translation by jointly learning to align and translate". *arXiv preprint arXiv:1409.0473*.

Bengio, Yoshua, Réjean Ducharme, Pascal Vincent, and Christian Jauvin (2003). "A neural probabilistic language model". *Journal of machine learning research* 3.Feb, pp. 1137–1155.

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

Clark, Elizabeth, Yangfeng Ji, and Noah A Smith (2018). "Neural Text Generation in Stories Using Entity Representations as Context". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. Vol. 1, pp. 2250–2260.

Clark, Kevin and Christopher D Manning (2016). "Improving coreference resolution by learning entity-level distributed representations". *arXiv preprint arXiv:1606.01323*.

Dumais, Susan T (2004). "Latent semantic analysis". *Annual review of information science and technology* 38.1, pp. 188–230.

Elman, Jeffrey L (1990). "Finding structure in time". *Cognitive science* 14.2, pp. 179–211.

Goldberg, Yoav (2016). "A primer on neural network models for natural language processing". *Journal of Artificial Intelligence Research* 57, pp. 345–420.

Goodfellow, Ian, Yoshua Bengio, Aaron Courville, and Yoshua Bengio (2016). *Deep learning*. Vol. 1. MIT press Cambridge.

Hochreiter, Sepp (1998). "The vanishing gradient problem during learning recurrent neural nets and problem solutions". *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02, pp. 107–116.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". *Neural computation* 9.8, pp. 1735–1780.

Hovy, Eduard, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel (2006). "OntoNotes: the 90% solution". In: *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*. Association for Computational Linguistics, pp. 57–60.

Ji, Yangfeng, Chenhao Tan, Sebastian Martschat, Yejin Choi, and Noah A Smith (2017). "Dynamic entity representations in neural language models". *arXiv preprint arXiv:1708.00781*.

Katz, Slava (1987). "Estimation of probabilities from sparse data for the language model component of a speech recognizer". *IEEE transactions on acoustics, speech, and signal processing* 35.3, pp. 400–401.

Kim, Yoon, Yacine Jernite, David Sontag, and Alexander M Rush (2016). "Character-Aware Neural Language Models." In: *AAAI*, pp. 2741–2749.

Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". *arXiv preprint arXiv:1412.6980*.

Kneser, Reinhard and Hermann Ney (1995). "Improved backing-off for m-gram language modeling". In: *icassp*. Vol. 1, 181e4.

LeCun, Yann, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel (1989). "Backpropagation applied to handwritten zip code recognition". *Neural computation* 1.4, pp. 541–551.

Lee, Heeyoung, Yves Peirsman, Angel Chang, Nathanael Chambers, Mihai Surdeanu, and Dan Jurafsky (2011). "Stanford's multi-pass sieve coreference resolution system at the CoNLL-2011 shared task". In: *Proceedings of the fifteenth conference on computational natural language learning: Shared task*. Association for Computational Linguistics, pp. 28–34.

Lee, Kenton, Luheng He, Mike Lewis, and Luke Zettlemoyer (2017). "End-to-end neural coreference resolution". *arXiv preprint arXiv:1707.07045*.

Luong, Minh-Thang, Hieu Pham, and Christopher D Manning (2015). "Effective approaches to attention-based neural machine translation". *arXiv preprint arXiv:1508.04025*.

McCarthy, Joseph F and Wendy G Lehnert (1995). "Using decision trees for coreference resolution". *arXiv preprint cmp-lg/9505043*.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). "Efficient estimation of word representations in vector space". *arXiv preprint arXiv:1301.3781*.

Mikolov, Tomáš, Anoop Deoras, Stefan Kombrink, Lukáš Burget, and Jan Černockỳ (2011). "Empirical evaluation and combination of advanced language modeling techniques". In: *Twelfth Annual Conference of the International Speech Communication Association*.

Mikolov, Tomáš, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur (2010). "Recurrent neural network based language model". In: *Eleventh Annual Conference of the International Speech Communication Association*.

Ng, Vincent (2010). "Supervised noun phrase coreference research: The first fifteen years". In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, pp. 1396–1411.

Ng, Vincent (2017). "Machine Learning for Entity Coreference Resolution: A Retrospective Look at Two Decades of Research." In: *AAAI*, pp. 4877–4884.

Olah, Christopher (2015). "Understanding lstm networks". *GITHUB blog, posted on August* 27, p. 2015.

Parker, Robert, David Graff, Junbo Kong, Ke Chen, and Kazuaki Maeda (2011). "English Gigaword Fifth Edition LDC2011T07". *Linguistic Data Consortium, Philadelphia, PA*.

Paszke, Adam, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer (2017). "Automatic differentiation in PyTorch".

Pennington, Jeffrey, Richard Socher, and Christopher Manning (2014). "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543.

Raghunathan, Karthik, Heeyoung Lee, Sudarshan Rangarajan, Nathanael Chambers, Mihai Surdeanu, Dan Jurafsky, and Christopher Manning (2010). "A multi-pass sieve for coreference resolution". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, pp. 492–501.

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms". *arXiv preprint arXiv:1609.04747*.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1986). "Learning representations by back-propagating errors". *nature* 323.6088, p. 533.

Schuster, Mike and Kuldip K Paliwal (1997). "Bidirectional recurrent neural networks". *IEEE Transactions on Signal Processing* 45.11, pp. 2673–2681.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting". *The Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Sundermeyer, Martin, Ralf Schlüter, and Hermann Ney (2012). "LSTM neural networks for language modeling". In: *Thirteenth annual conference of the international speech communication association*.

Turian, Joseph, Lev Ratinov, and Yoshua Bengio (2010). "Word representations: a simple and general method for semi-supervised learning". In: *Proceedings of the 48th annual meeting of the association for computational linguistics*. Association for Computational Linguistics, pp. 384–394.

Weischedel, Ralph, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, et al. (2013). "Ontonotes release 5.0 LDC2013T19". *Linguistic Data Consortium, Philadelphia, PA*.

Werbos, Paul J (1990). "Backpropagation through time: what it does and how to do it". *Proceedings of the IEEE* 78.10, pp. 1550–1560.

Wiseman, Sam Joshua, Alexander Matthew Rush, Stuart Merrill Shieber, and Jason Weston (2015). "Learning anaphoricity and antecedent ranking features for coreference resolution". In: Association for Computational Linguistics.

Wiseman, Sam, Alexander M Rush, and Stuart M Shieber (2016). "Learning global features for coreference resolution". *arXiv preprint arXiv:1604.03035*.

Xu, Kelvin, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio (2015). "Show, attend and tell: Neural image caption generation with visual attention". In: *International conference on machine learning*, pp. 2048–2057.

Yang, Zichao, Phil Blunsom, Chris Dyer, and Wang Ling (2016). "Reference-aware language models". *arXiv preprint arXiv:1611.01628*.