Assignment 2: Deep Learning

Sander Nugteren 6042023 Ties van Rozendaal 10077391 Joost Baptist 10760105

May 18, 2016

1 Introduction

One of the key challenges in artificial intelligence has been understanding and reasoning with texts. Earlier work in natural language processing successfully applied graphical models and bayesian methods on various NLP tasks.

Lately, Neural Networks (NNs) resulted in a breakthrough in various fields of AI, including computer vision and machine learning [5, 7]. However, various aspects of natural text make it difficult to apply standard neural networks to NLP tasks. The difficulty arises from the need to represent text as numerical vectors, that can be used in neural networks.

1.1 Representing words

Firstly, words need to be turned into numerical vectors. The usual representation in a Bayesian setting is a one-out-of-k vector, in which each word is represented as a vector of the vocabulary length (k) which is zero everywhere except at the word index. A disadvantage of this representation, is that it grows exponentially with vocabulary size. Moreover, words may be more, or less similar compared to other words, but this is not reflected in the one-out-of-k word vectors.

Distributional semantics have been used successfully to learn vector representation of words [9, 1]. These techniques use the contexts in which a word occurs to compute a semantically meaningful vector (of any length). Various approaches exist, including the well known word2vec [9].

1.2 Learning with sentences

Apart from finding a word representation, another problem is that of sentence length. Sentences may vary in length, yet a neural network needs a fixed size vector for each datapoint. This problem was solved by the introduction of recurrent neural networks [4].

The recurrent neural network consist of a single recurrent unit, that is repeated for each word in the sentence (or more general, for each timestep of a sequence). At each timestep t, the unit takes the current word vector ($\mathbf{x}_t$) as input, as well as a vector containing information about the previous timestep $\mathbf{h}_{t-1}$. The unit outputs this 'history'-vector $\mathbf{h}_t$ at each timestep.

Depending on the task, the network may also need to make a prediction at every timestep. This output is called $\mathbf{y}_t$. When predicting a sequence of words, the output at each timestep $\mathbf{y}_t$ is often defined as a probability distribution over the vocabulary.

1.3 Plain Recurrent Neural Networks (RNNs)

The most simple recurrent neural network consists of a single hidden layer, which is passed across time steps. Using weight matrices $\mathbf{W}$, $\mathbf{U}$, a bias $\mathbf{b}$ and a nonlinearity $f(\bullet)$ a formal definition is:

$$\mathbf{h}_t = f(\mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b}) \tag{1}$$

An output distribution is most easily defined by a softmax-distribution, based on the hidden state of the unit:

$$\hat{\mathbf{y}}_t = \sigma(\mathbf{V}\mathbf{h}_t + \mathbf{b}_s) \tag{2}$$

Where the softmax function σ is defined by:

$$\sigma(\mathbf{z})_i = \frac{\exp\{z_i\}}{\sigma_{k=1}^{K} \exp\{z_k\}} \tag{3}$$

When a loss is defined over the output sequence $(\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_N)$, the network can be trained by back-propagating the gradient over this loss trough all enrolled recurrent units.

1.4 Gated Recurrent Units (GRUs)

In a simple RNN, long-range time dependencies get propagated trough the hidden states, involving a multiplication with $\mathbf{U}$ at every timestep. As the length of the dependency increases, this iterative multiplication has been found to result in both exploding and vanishing gradient-values [10]. This problem poses a severe limitation on the power of RNNs [8].

The problem of the exploding gradient can be solved by simply clipping the gradient at too high values, but the vanishing gradient problem is more complicated. A solution was found by gating the feedback in each unit. Although the idea of gating was initially proposed in the Long-Short-Term-Memory network (LSTM) [6], a simpler version of this network known as the Gated Recurrent Unit (GRU) has become increasingly popular [3].

The GRU uses two gates to control the information flow. Both gates can be controlled by the data $\mathbf{x}_t$ and the previous state $\mathbf{h}_{t-1}$, based on trainable parameters. A sigmoid function (denoted $\tilde{\sigma}$) scales the values of the gates between 0 and 1 representing a fully closed and a fully opened gate respectively.

The equations for the gates are similar to the update equations of a regular RNN:

$$z_t = \tilde{\sigma}\left(\mathbf{W}_z \mathbf{x}_t + \mathbf{U}_z \mathbf{h}_{t-1} + \mathbf{b}_z\right) \tag{4}$$

$$r_t = \tilde{\sigma}\left(\mathbf{W}_r \mathbf{x}_t + \mathbf{U}_r \mathbf{h}_{t-1} + \mathbf{b}_r\right) \tag{5}$$

where $\odot$ denotes element wise multiplication

The GRU does not compute the hidden state $\mathbf{h}_t$ directly, but uses an internal 'propal' state $\widetilde{\mathbf{h}}_t$. Again, the way this proposal states is computed, is very similar to the computation of the hidden state in a regular RNN. However, instead of using the complete previous state, the reset gate controls how much of the old information is discarded.

$$\widetilde{\mathbf{h}}_t = f(\mathbf{W}\mathbf{x}_t + \mathbf{U}\left(r_t \odot \mathbf{h}_{t-1}\right)) \tag{6}$$

Lastly the actual hidden state is computed. The true hidden state is simply a linear interpolation between the proposal hidden state and the previous hidden state. The update gate controls the balance between these two vectors.

$$\mathbf{h}_t = (1 - z_t)\mathbf{h}_{t-1} + z_t \mathbf{widetildeh}_t \tag{7}$$

Analogous to the simple RNN, the hidden state $\mathbf{h}_t$ can be used to produce an output.

1.5 Encoder-Decoder

The RNN (and the abstraction to the GRU) define models which are able to turn a sequence into a vector, and vice versa. However, many NLP taks require sequence to sequence mapping. An obvious example of this is machine translation, where a sequence in a source language has to be converted to a sequence in a target language. Interestingly, most problems of reasoning and question answering can also be defined in this framework, as they can essentially by described as problems of mapping sentences to other sentences.

A framework which allows for sequence-to-sequence mapping is the encoder-decoder model. The encoder converts an input into a fixed length vector, which we call the context vector $\mathbf{c}$. This context vector is then passed to the decoder to produce an output. When RNNs are used as encoder and the decoder, this model maps sequences to sequences.

When using the RNN, the last hidden state serves well as a context vector [2, 12, 11]. At each timestep t, the hidden state of a RNN contains information about all the previous states, and as a result the last hidden state can represent the entire sequence.

The context vector can then be fed into the decoder RNN, either as the initial state $\mathbf{h}_0$, or as a separate input at each timestep or both.

1.6 Our experiments

Taken altogether, the RNN Encoder-Decoder model is suited for the task of question answering, and may perform well on this task. We will investigate this, by evaluating several recurrent encoder-decoder models on a question answering task. We used the facebook bAbI dataset for question answering.

First, we tested our models on a reversal task to investigate their handling of long term dependencies. Next, we used our models for question answering.

We evaluated a simple RNN model, and a single layer GRU model to allow for longer time dependencies. We also optimised a multi-layer GRU to achieve optimal results on this dataset.

2 Model

Though we tested various recurrent models, they all had the same basic structure. Each model consisted of an encoder- and a decoder unit, that could be a single- or multi- layer recurrent neural networks (RNN or GRU). Multilayer networks used the hidden state $\mathbf{h}_t$ of the lower layer, to serve as input vector of the higher layer.

Encoder-units, were implemented as described in 1.3 (recurrent units), and 1.4 (GRU units). The context vector $\mathbf{c}$ was passed to the decoder at each timestep and thus equations (1), (4), (5) and (6) were modified to include a term $\mathbf{Cc}$ within each nonlinearity function (where $\mathbf{C}$ is a trainable parameter converting dimensions of the context vector to the dimensions of that layer).

Furthermore, the context vector $\mathbf{c}$ was used to initialise the hidden state (for each of the possibly stacked layers) of the decoder. A convertor consisting of linear layer and a nonlinearity was used to transform $\mathbf{c}$ into $\mathbf{h}_0$. For all nonlinearities (except the gates in the GRU) a tanh function was used.

An embedder was used to convert words from a single index value, into a word embedding used as input for the decoder or encoder. The embedder shared parameters for the encoding and decoding phase. We tried initialising the embeddings using word2vec. But as this made no difference we did not report the results. All embeddings reported were initialised randomly.

Input propagation trough the embedder, encoder and convertor did not differ during testing and training. The use of the decoder varied for both phases.

2.1 Training

Figure 1 depicts the network during the training of a single datapoint (input and output sentence). The last token of the input sequence was used as the first input $\mathbf{x}_1$ for the decoder. At each timestep, the decoder outputted a distribution over te vocabulary using a softmax. This distribution was compared with the true distribution using a categorical cross entropy loss. The true distribution was defined as a one-out-of k distribution for the correct word (i.e. the probability of the true word was 1, the other probabilities were 0).

For the following timesteps, $t \in 2, \ldots, N$, the true output word for the previous timestep ($\mathbf{y}_{t-1}$) was used as input for the decoder ($\mathbf{x}_t$), correcting possible mistakes. As a result, the predicted sequence always had the same length as the target sequence, regardless if the decoder did or did not produce the end-of-sequence token.

The model was trained using minibatches, and the loss was averaged over all words in all sentences equally. All parameters were updated using back-propagation, and the gradient was clipped between -1, and 1 to prevent exploding values.

2.2 Predicting

Figure 2 shows the network during prediction. The main difference with the network during training is that at each timestep $t > 1$, the prediction of the previous time ($\hat{\mathbf{y}}_{t-1}$) is used as input ($\mathbf{x}_t$). In order to predict a single word, based on the distribution over the vocabulary, we used an argmax module, selecting the word with the highest probability. The decoder kept predicting words untill the end-of-sentence token was generated, or until the sentence had reached a maximum length.

3 Experiments

The model was used on two different tasks: question answering and sequence reversal. The facebook bAbI dataset was used for both tasks.

3.1 Tasks

3.1.1 Learning to answer

The bAbI dataset is an artificial dataset containing short simple stories. Each story is built up by simple factual statements, together forming the context in which the question has to be answered. Each story is intermingled with

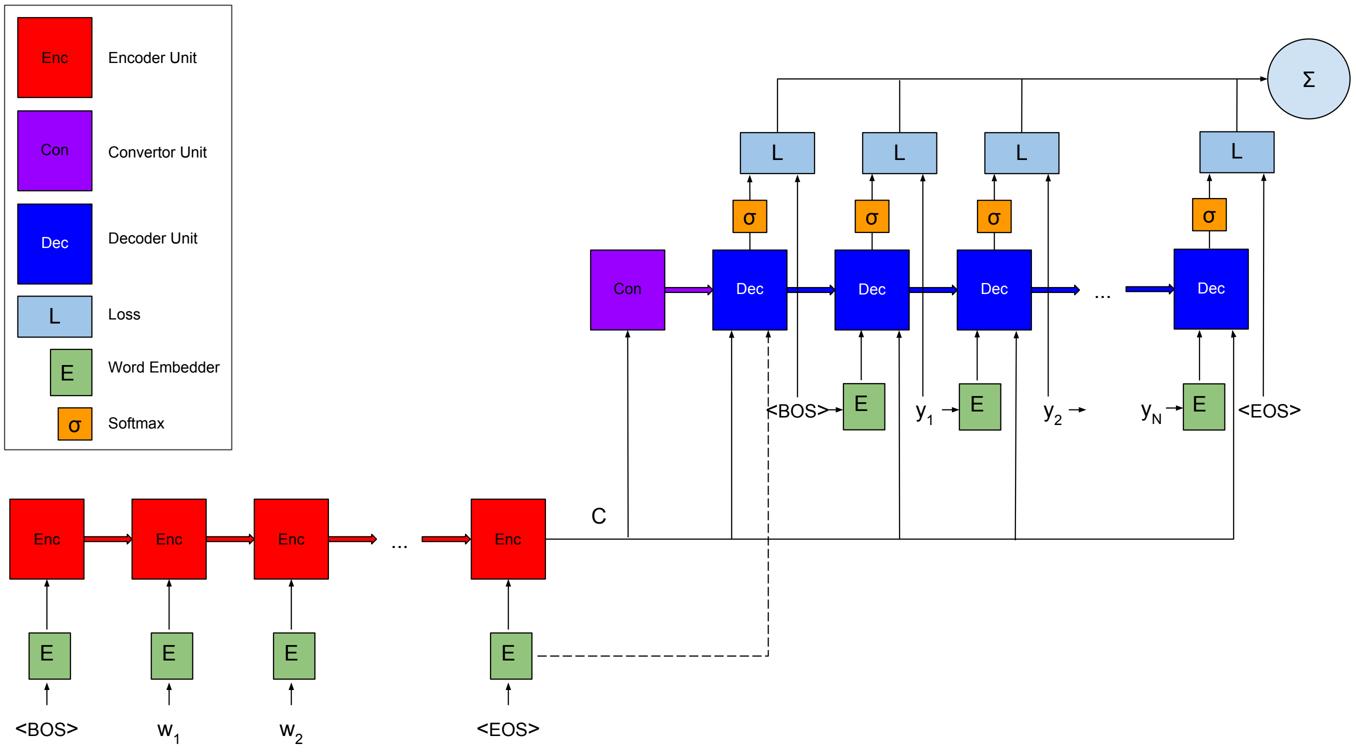


Figure 1: **The network during training.** During training, the target output words (y_t) are put into the decoder at each timestep t, correcting possible mistakes. $<BOS>$ and $<EOS>$ denote the beginning and end of sequence token respectively. w_i: denotes of the input sentence.

several questions. The answer to each question can be found in the preceding sentences.

The dataset has 20 different sub-tasks on which we performed our question-answering experiments. These range from very simple ones (answering questions which need only one supporting fact to be answered) to very difficult (inferring the motivation of the agents performing certain actions).

We preprocessed the data by merging the question with the story facts. The question was always prepended to the story facts. We used all sentences preceding the question, resulting in datapoints of various lengths.

Even though the inputs consist of natural language sentences, we noted that the output always consisted of a single word. As all sentences were assigned end- and beginning- of sentence tokens, each target sequences consisted of 3 tokens. We therefore set the maximum sentence length during prediction to 5.

3.1.2 Learning to reverse

For sequence reversal we took the first five datasets and simply used the reversed sequences as the target output for the decoder has to generate. This transformed the question answering task into a sequence reversal task. Otherwise, the same setup was used for both the reversal task and the question-answering task. The maximum sentence length during prediction was set to 50 tokens.

3.2 Model selection

Our models have many parameters that can be tuned for an optimal performance. We conducted experiments with the most important parameters, listed below. When trying different values for one parameter, we kept the other parameters fixed to a default value, marked by an asterisk. We tested all parameters on the first question answering task.

- **Units.** We tested our model with standard recurrent units and with GRU* units.
- **Number of layers.** We tested our model with 1*, 2, and 4 layers (both the encoder and decoder).
- **Word embedding size.** We tested with values 10*, 20, 30 and 40.

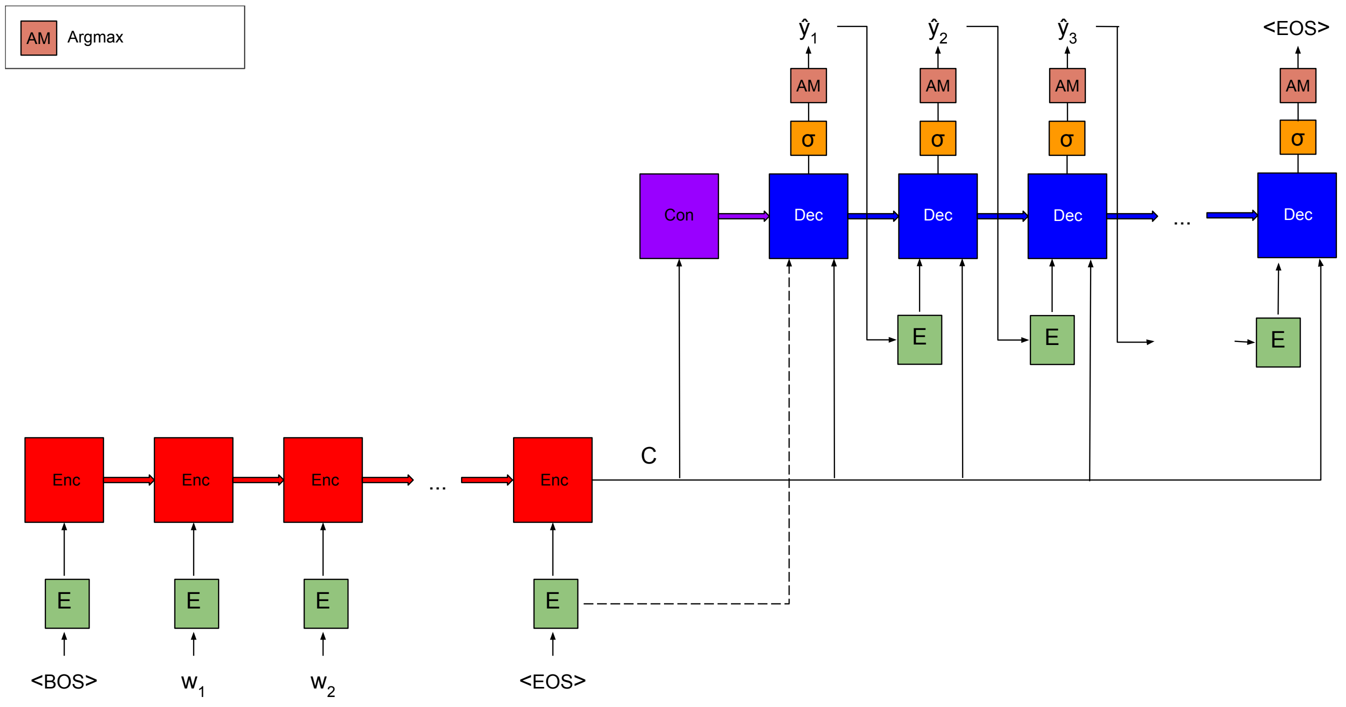


Figure 2: **The network during prediction.** The encoder and convertor units function identical as in the training phase. An argmax module selects the most probable word $\hat{y}_t$. Each predicted word $\hat{y}_t$ is used as input for the next timestep $t + 1$ untill an <EOS> (end of sequence) token is generated. For a complete legend see Figure 1.

- **Hidden layer size.** We tested with values 20, 40* and 80.
- **Learning rate.** We tested with three learning rate regimes: a linearly spaced range of values between 0.5 and 0.001*, a log-spaced range of values between 0.5 and 0.001, and a fixed value of 0.01.
- **Momentum.** We tested with three momentum regimes: a linearly spaced range of values between 0.5 and 0.9*, a log-spaced range of values between 0.5 and 0.9, and no momentum.
- **Minibatch size.** We tested with values 1, 10* and 20.
- **Weight decay.** We tested with values 0, 0.001*, 0.01 and 0.1.
- **Epochs.** This parameter is fixed to 50 for all experiments.

Using the optimal parameters fount in the above experiments, and some informal experimentation, we defined an optimal model. We call this model OPTIMAL, and evaluated its performance. With this model, we compared the effect of trainable word embeddings to a fixed one-out-of-k vector representation (also know as a bag-of-words approach, which we will call BOW). Furthermore, we tested the effect of feeding context vectors **c** at each timestep to the decoder, versus only using the converter to input the context vector as the initial state $\mathbf{h}_0$. We evaluated the OPTIMAL model on both the sequence reversing and question answering tasks, and the BOW and NOCONTEXT models on just the question answering task.

OPTIMAL looks as follows:

- GRU units.
- 2 encoder and 2 decoder layers.
- Word embedding size of 40.
- Hidden layer size of 80.
- Learning rate is a linearly spaced range of values between 0.5 and 0.001.
- Momentum is a linearly spaced range of values between 0.5 and 0.9.

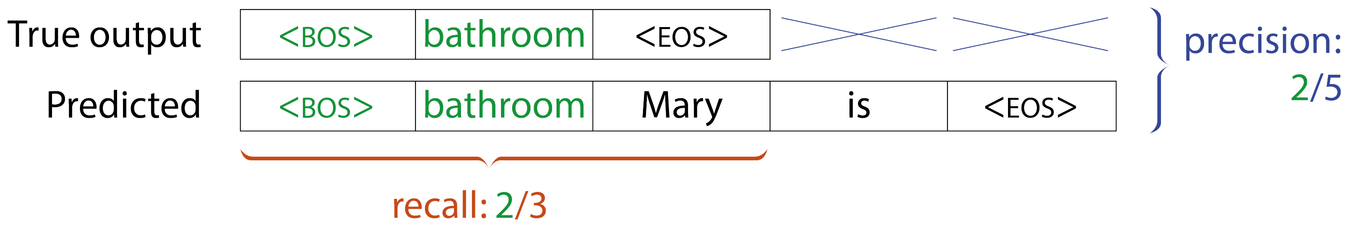


Figure 3: Precision and recall

- Minibatch size of 10.
- Weight decay is 0.001.

3.2.1 Datasets

We ran the parameter selection experiments and the experiments with the BoW and NoContext models for the question answering task on a single dataset: *single-supporting-fact*. We test our Optimal model on five datasets for both tasks:

- *single-supporting-fact* (vocabulary size: 23): Only a single statement is needed to answer a particular question.
- *two-supporting-facts* (vocabulary size: 37): A combination of three statements is needed to answer the question.
- *three-supporting-facts* (vocabulary size: 38): A combination of three statements is needed to answer the question.
- *two-arg-relations* (vocabulary size: 19): These questions involve relations between objects with two arguments.
- *three-arg-relations* (vocabulary size: 43): These questions involve relations with three arguments.

3.3 Evaluation

Because of the nature of our prediction phase, the predicted sequences and the true target sequences could differ in length. We defined two measures to compare the two sequences. Both measures count the number of exact matches. In recall, the shape of the predicted sequence was adjusted to the shape of the target sequence and the proportion correct was calculated.

For precision, we used the accuracy when adjusting the shape of the target sequence to the shape of the predicted sequence.

Figure illustrates these measures. Note that the predicted sequence could be longer or shorter than the target sequence. As a result, adjusting the size of the other sequence can mean extending or shortening of the sequence. However, because each answer consisted of only one word. The true output usually was equal to, or shorter than the true output, and we can interpret our measures as precision and recall.

Another remark is that our definition of precision and recall differs from the conventional definition used in information retrieval. Our measures consider exact matches that are order sensitive. We will use the terms precision and recall throughout this report, always referring to the described definition.

Note that when we mention precision and recall throughout this paper, we mean ordered precision and recall as defined here.

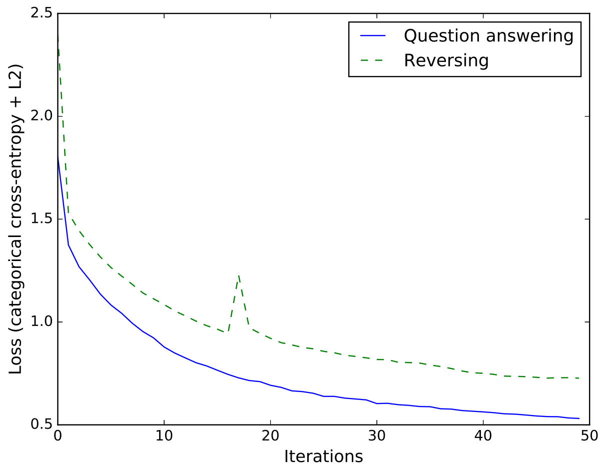
4 Results

4.1 Model selection

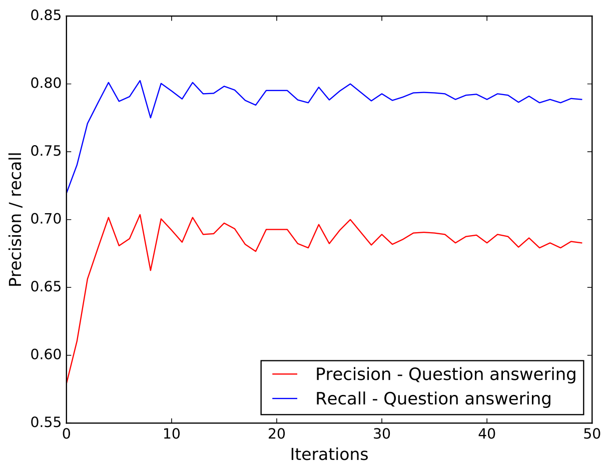
See Table 1 for the results of our model selection experiments on the *single-supporting-fact* dataset. The rest of the experiments were performed with the optimal parameters given by these results (i.e., our Optimal model uses these parameters). The results show that 1) the GRU outperforms the standard recurrent unit, 2) having 2 hidden layers is optimal, 3) larger word embeddings and larger hidden layer sizes are optimal, 4) linear or log spaced learning rate and momentum regimes are optimal, 5) a minibatch size of 10 is optimal, and 6) a smaller weight decay is optimal.

Parameter	**Value**	**Precision (train)**	**Recall (train)**	**Precision (test)**	**Recall (test)**
Unit type	GRU*	0.720	0.813	0.688	0.792
Unit type	Recurrent	0.600	0.733	0.584	0.723
Hidden layers	1*	0.689	0.793	0.663	0.776
Hidden layers	2	0.736	0.824	0.692	0.795
Hidden layers	4	0.589	0.726	0.594	0.729
Word embedding size	10*	0.736	0.824	0.681	0.788
Word embedding size	20	0.728	0.818	0.680	0.787
Word embedding size	30	0.759	0.839	0.680	0.787
Word embedding size	40	0.749	0.832	0.686	0.791
Hidden layer size	20	0.679	0.786	0.654	0.770
Hidden layer size	40*	0.693	0.795	0.663	0.775
Hidden layer size	80	0.749	0.832	0.685	0.790
Learning rate	0.01	0.589	0.726	0.594	0.729
Learning rate	linear*	0.692	0.794	0.662	0.775
Learning rate	log	0.699	0.799	0.674	0.783
Momentum	0	0.717	0.811	0.680	0.787
Momentum	linear*	0.740	0.827	0.684	0.790
Momentum	log	0.739	0.826	0.678	0.785
Minibatch size	1	0.657	0.771	0.639	0.760
Minibatch size	10*	0.715	0.810	0.689	0.792
Minibatch size	20	0.714	0.809	0.678	0.785
Weight decay	0	0.829	0.886	0.681	0.787
Weight decay	0.001*	0.715	0.810	0.679	0.786
Weight decay	0.01	0.587	0.724	0.579	0.719
Weight decay	0.1	1.000	0.667	1.000	0.667

Table 1: Results for the model selection experiments performing the question answering task on the *single-supporting-fact* dataset. *Default values for parameters: parameters were fixed to these values when testing the other parameters.



(a) Categorical cross-entropy loss with L2.



(b) Precision and recall evaluated on the test set.

Figure 4: Loss and performance of the OPTIMAL model over 50 iterations on the *single-supporting-fact* dataset.

Dataset	**Precision (train)**	**Recall (train)**	**Precision (test)**	**Recall (test)**
single-supporting-fact	0.762	0.841	0.684	0.790
two-supporting-facts	0.608	0.738	0.592	0.728
three-supporting-facts	0.601	0.734	0.590	0.726
two-arg-relations	0.590	0.727	0.586	0.724
three-arg-relations	0.654	0.769	0.642	0.761

Table 2: Performance of the OPTIMAL model on the question answering task on five different datasets.

4.2 Tasks

Figure **??** shows the loss and precision/recall of our OPTIMAL model over 50 iterations on the *single-supporting-fact* dataset. We see that although the loss continues to drop the entire time, the precision/recall maximize quite early. Unfortunately, we do not have results that show the precision/recall over time for the sequence reversing task.

Table 2 shows the results of our OPTIMAL model performing the question answering task on five different datasets. We see that the model performs best on the *single-supporting-fact* dataset, and slightly worse on the other datasets.

Table 3 shows the results of our OPTIMAL model performing the sequence reversing task averaged over five different datasets.

4.3 Optimal vs. BoW vs. NoContext

Table 4 shows the effect of using a bag-of-words approach (instead of word embeddings) and dropping the context vector in the decoder. We see that both the word embeddings and the context vectors are a crucial part of the model, and dropping them leads to a significant performance drop.

5 Analysis

5.1 Sentence lengths

Figures 5 and 6 show the effect of input length on performance. For the question answering task, we see that input length does not affect performance a lot, and it is still fine with longer input sequences. However, for the sequence reversing task, we see that the model can deal with short sequences, but performance drops a lot when input sequences

Precision (train)	**Recall (train)**	**Precision (test)**	**Recall (test)**
0.499	0.505	0.487	0.494

Table 3: Performance of the OPTIMAL model on the sequence reversing task averaged over five different datasets.

Model	Precision (train)	Recall (train)	Precision (test)	Recall (test)
OPTIMAL	0.762	0.841	0.684	0.790
BOW	0.588	0.725	0.577	0.718
NOCONTEXT	0.589	0.726	0.594	0.729

Table 4: Results for experiments where components were dropped from the OPTIMAL model. BOW uses no word embeddings but a bag-of-words approach, and NOCONTEXT does not inject a context vector into the decoder.

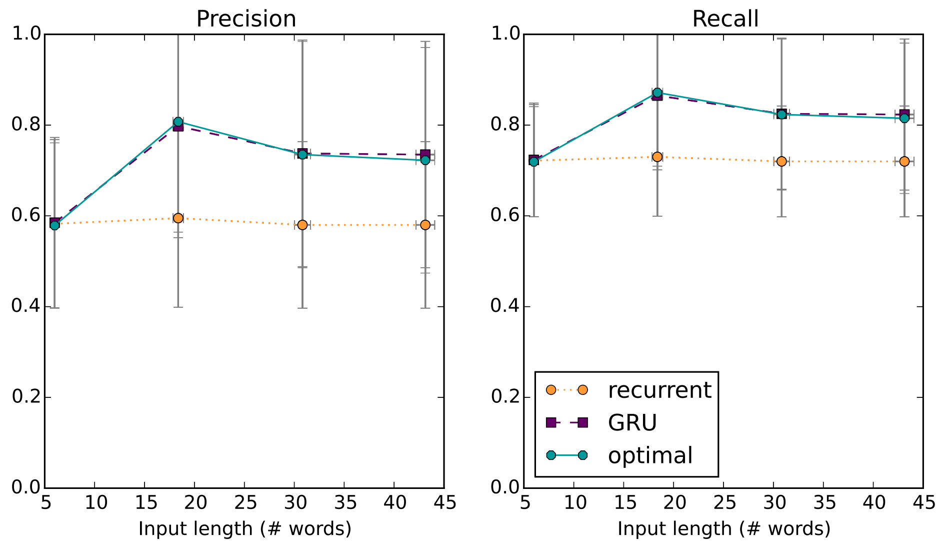


Figure 5: Precision and recall for two models on the question answering task.

get longer. It seems that the model lacks a long term memory for these kinds of tasks. Finally, for both tasks, we see that the GRU performs better, as expected due to its more sophisticated memory.

5.2 Word embeddings

Figure 7 shows 2D visualization of the word embeddings found by the OPTIMAL model on the *single-supporting-fact* dataset. Although we do not observe perfect clusters, we can see that it has managed to find a meaningful representation. For example, we see that locations, such as *bedroom*, *bathroom*, *garden*, *hallway* and *office* appear relatively close to each other. Verbs such as *is*, *went*, *moved*, *travelled* appear to be clustered as well.

6 Conclusion

We can conclude that reversal definitely is the hardest task for this model, though this could also be due to the fact that the generated sequences are much longer than the ones for question answering (where the answer were one word long).

Performance was generally close to the state of the art, with similar performance on both training and test set, suggesting a lack of overfitting.

Also, both the context and the embeddings help the model, since removing them hurts performance significantly.

References

[1] Enrique Alfonseca and Suresh Manandhar. Extending a lexical ontology by a combination of distributional semantics signatures. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 1–7. Springer, 2002.

[2] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

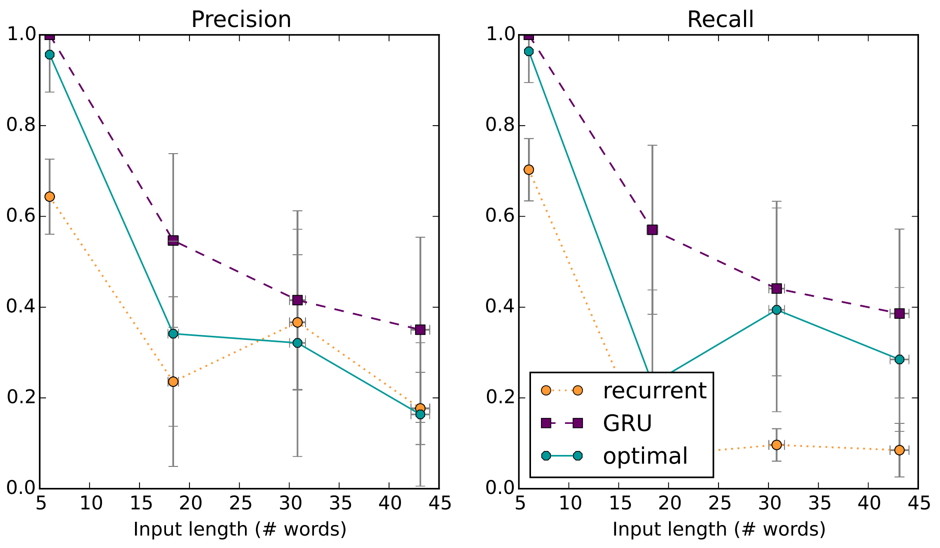


Figure 6: Precision and recall for two models on the sequence reversing task.

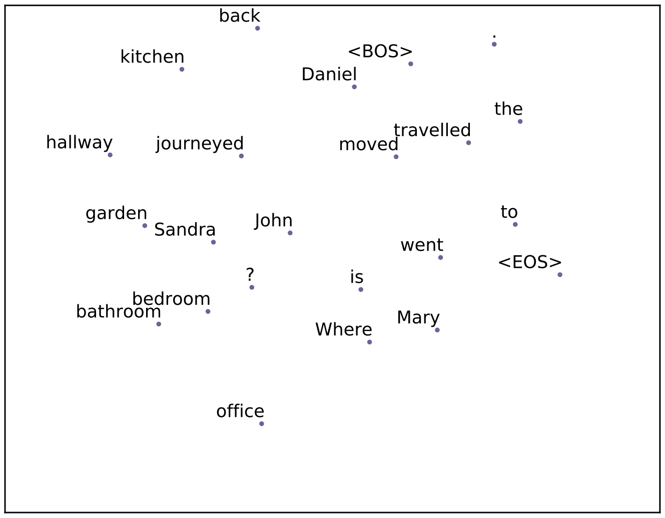


Figure 7: 2D visualization (using t-SNE) of the word embeddings found by the Optimal model on the *single-supporting-fact* dataset.

[3] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. *arXiv preprint arXiv:1502.02367*, 2015.

[4] Laurene Fausett. Fundamentals of neural networks: architectures, algorithms, and applications. 1994.

[5] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Education Upper Saddle River, 2009.

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

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[8] LR Medsker and LC Jain. Recurrent neural networks. *Design and Applications*, 2001.

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

[10] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2012.

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[12] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. Towards ai-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*, 2015.