Attentive Language Models

Giancarlo D. Salton and **Robert J. Ross** and **John D. Kelleher**
Applied Intelligence Research Centre
School of Computing
Dublin Institute of Technology
Ireland
giancarlo.salton@mydit.ie {robert.ross,john.d.kelleher}@dit.ie

Abstract

In this paper, we extend Recurrent Neural Network Language Models (RNN-LMs) with an attention mechanism. We show that an *Attentive* RNN-LM (with 14.5M parameters) achieves a better perplexity than larger RNN-LMs (with 66M parameters) and achieves performance comparable to an ensemble of 10 similar sized RNN-LMs. We also show that an *Attentive* RNN-LM needs less contextual information to achieve similar results to the state-of-the-art on the wikitext2 dataset.

1 Introduction

Language Models (LMs) are an essential component in a range of Natural Language Processing applications, such as Statistical Machine Translation and Speech Recognition (Schwenk et al., 2012). An LM provides a probability for a sequence of words in a given language, reflecting fluency and the likelihood of that word sequence occurring in that language.

In recent years Recurrent Neural Networks (RNNs) have improved the state-of-the-art in LM research (Józefowicz et al., 2016). Sequential data prediction, however, is still considered a challenge in Artificial Intelligence (Mikolov et al., 2010) given that, in general, prediction accuracy degrades as the size of sequences increase.

RNN-LMs sequentially propagate forward a context vector by integrating the information generated by each prediction step into the context used for the next prediction. One consequence of this forward propagation of information is that older information tends to fade from the context as new information is integrated into the context. As a result, RNN-LMs struggle in situations where there is a long-distance dependency because the relevant information from the start of the dependency has faded by the time the model has spanned the dependency. A second problem is that the context can be dominated by the more recent information so when an RNN-LM does make an error this error can be propagated forward resulting in a cascade of errors through the rest of the sequence.

In recent sequence-to-sequence research the concept of "attention" has been developed to enable RNNs to align different parts of the input and output sequences. Examples of attention based architectures include Neural Machine Translation (NMT) (Bahdanau et al., 2015; Luong et al., 2015) and image captioning (Xu et al., 2015).

In this paper we extend the RNN-LM context mechanism with an attention mechanism that enables the model to bring forward context information from different points in the context sequence history. We hypothesis that this attention mechanism enables RNN-LMs to: (a) bridge long-distance dependencies, thereby avoiding errors; and, (b) to overlook recent errors by choosing to focus on contextual information preceding the error, thereby avoiding error propagation.

We show that a medium sized[1] *Attentive* RNN-LM[2] achieves better performance than larger "standard" models and performance comparable to an ensemble of 10 "medium" sized LSTM RNN-LMs on the PTB. We also show that an *Attentive* RNN-LM needs less contextual information in order to achieve similar results to state-of-the-art results over the wikitext2 dataset.

Outline: §2 introduces RNN-LMs and related research, §3 outlines our approach, §4 describes our experiments, §5 presents our analysis of the models' performance and §6 our conclusions.

[1]We adopt the terminology of Zaremba et al. (2015) and Press and Wolf (2016) when referring to the size of the RNNs.

[2]Code available at https://github.com/giancds/attentive_lm

2 RNN-Language Models

RNN-LMs model the probability of a sequence of words by modelling the joint probability of the words in the sequence using the chain rule:

$$p(w_1, \ldots, w_N) = \prod_{t=1}^{N} p(w_n | w_1, \ldots, w_{n-1}) \quad (1)$$

where N is the number of words in the sequence. The context of the word sequence is modelled by an RNN and for each position in the sequence the probability distribution over the vocabulary is calculated using a softmax on the output related to that position of the RNN‘s last layer (i.e., the last layer‘s hidden state) (Józefowicz et al., 2016). Examples of such models include Zaremba et al. (2015) and Press and Wolf (2016). These models are composed of LSTM units (Hochreiter and Schmidhuber, 1997) and apply regularization to improve the RNN performance. In addition, Press and Wolf (2016) also uses the same embedding matrix that is used to transform the input words to transform the output of the last RNN layer to feed it to the softmax layer to make the next prediction.

Attention mechanisms were first proposed in “encoder-decoder” architectures for NMT systems. Bahdanau et al. (2015) proposed a model that stores all the encoder RNN’s outputs and uses them together with the decoder RNN’s state h_{t-1} to compute a context vector that, in turn, is used to compute the state h_t. In Luong et al. (2015) a generalization of the model of Bahdanau et al. (2015) is presented which uses the decoder RNN‘s state, in this instance h_t rather than h_{t-1}, along with the outputs of the encoder RNN to compute a context vector that it then concatenated with h_t before making the next prediction. Both models have similar performance and achieve state-of-the-art performance for some language pairs; however, they suffer from repeating words or dropping translations at the output (Mi et al., 2016).

There is previous work on using past information to improve RNN-LMs. Tran et al. (2016) propose an extension to LSTM cells to include memory areas, which depend on input words, at the output of every hidden layer. The model produces good results but the dependency on input words expands the number of parameters in each LSTM cell in proportion to the vocabulary size in use.

Similarly, Cheng et al. (2016) propose storing the LSTM‘s memory cells of every layer at each timestep and draw a context vector for each memory cell for each new input to attend to previous content and compute its output. Although their model requires fewer parameters than the model of Tran et al. (2016), the performance of the model is worse than regularized “standard” RNN-LM as in Zaremba et al. (2015) and Press and Wolf (2016).

Daniluk et al. (2017) propose an augmented version of the attention mechanism proposed by Bahdanau et al. (2015) on which their model outputs 3 vectors called *key-value-predict*. The *key* (a vector of real numbers) is used to retrieve a single hidden state from the past. Grave et al. (2017) propose an LM augmented with a “memory cache” that stores tuples of hidden-states plus word embeddings (for the word predicted from that hidden state). The memory cache is used to help the current prediction by retrieving the word embedding associated with the hidden state in the memory most similar to the current hidden state. Merity et al. (2017) proposed a mixture model that includes an RNN and a pointer network. This model computes one distribution for the softmax component and one distribution for the pointer network, using a sentinel gating function to combine both distributions. In spite of the fact that their model is similar to the model of Grave et al. (2017), their model requires an extra transformation between the current state of the RNN and those stored in the memory.

These recent models have a number of drawbacks. The systems that extend the architecture of LSTM units struggle to process large vocabularies because the system memory expands to the size of the vocabulary. For systems that retrieve a single hidden-state or word from memory, if the prediction is not correct, the RNN-LM will not receive the correct past information. Finally, the models of Merity et al. (2017) and Grave et al. (2017) use a fixed-length memory of L previous hidden states to store and retrieve information from the past (100 states in the case of Merity et al. (2017) and 2,000 states in the case of Grave et al. (2017)). As we shall explain in §3 our “attentive” RNN-LMs have a memory of dynamic-length that grows with the length of the input and therefore, in general, are computationally cheaper.

We see our “attentive” RNN-LM (see §3) as a generalized version of these models as we rely on the encoded information in the hidden state of the

RNN-LM to represent previous input words and we use a set of attention weights (instead of a *key*) to retrieve information from the past inputs. The main advantages of our approach are: (a) our model does not need vocabulary sized matrices in the computations of the attention mechanism and therefore has a reduced number of parameters; and (b) as we use all previous hidden states of the RNN-LM in the computation for the attention weights, all of those states will influence the next prediction based on the weights calculated.

3 Attentive Language Models

In this work we extend RNN-LMs to include an attention mechanism over previous inputs. We employ a multi-layered RNN to encode the input and, at each timestep, we store the output of the last recurrent layer (i.e., its hidden state $\mathbf{h}_t$) into a memory buffer. We compute a score for each hidden state $\mathbf{h}_i$ ($\forall\ i \in \{1, \dots, t-1\}$) stored in memory and use these scores to weight each $\mathbf{h}_i$. From these weighted hidden states we generate a context vector $\mathbf{c}_t$ that is concatenated with the current hidden state $\mathbf{h}_t$ to predict the next word in the sequence. Figure 1 illustrates a step of our model when predicting the fourth word in a sequence.

We propose two different attention score functions that can be used to compute the context vector $\mathbf{c}_t$. One calculates the attention score of each $\mathbf{h}_i$ using just the information in the state (the $single(\mathbf{h}_i)$ score introduced below). The other calculates the attention scores for each $\mathbf{h}_i$ by combining the information from that state with the information from the current state $\mathbf{h}_t$ (the $combined(\mathbf{h}_i, \mathbf{h}_t)$ score described below). Each of these mechanisms defines a separate *Attentive* RNN-LMs and we report results for each of these LMs in our experiments.

More formally, each $\mathbf{h}_t$ is computed as follows, where $\mathbf{x}_t$ is the input at timestep t:

$$\mathbf{h}_t = RNN(\mathbf{x}_t, \mathbf{h}_{t-1}) \tag{2}$$

The context vector $\mathbf{c}_t$ is then generated using Eq. (3) where each scalar weight a_i is a softmax (Eq. (4)) and the score for each hidden state ($\mathbf{h}_i$) in the memory buffer is one of Eq. (5) or Eq. (6).

$$\mathbf{c}_t = \sum_{i=1}^{t-1} a_i \mathbf{h}_i \tag{3}$$

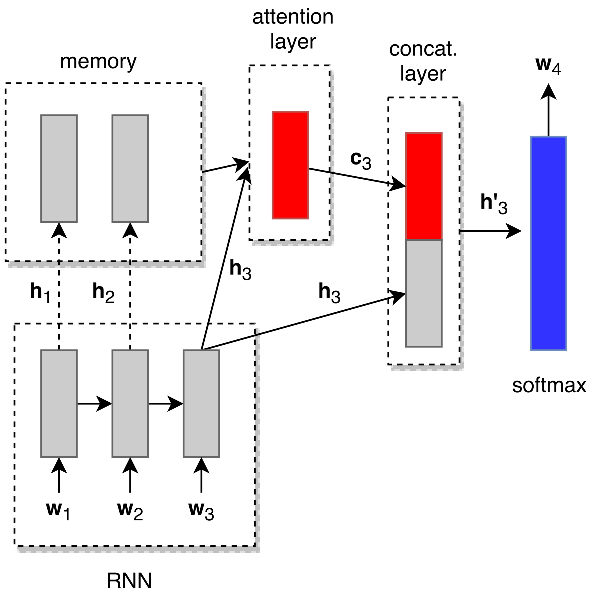

Figure 1: Illustration of a step of the *Attentive* RNN-LM with *combined* score. In this example, the model receives the third word as input ($\mathbf{w}_3$) after storing the previous states ($\mathbf{h}_1$ and $\mathbf{h}_2$) in memory. After producing $\mathbf{h}_3$, the model computes the context vector (in this case $\mathbf{c}_3$) that will be concatenated to $\mathbf{h}_3$ before the softmax layer for the prediction of the fourth word $\mathbf{w}_4$. Note that if the *single* score is in use (Eq. (9)), the arrow from the RNN output for $\mathbf{h}_3$ to the attention layer is dropped. Also note that $\mathbf{h}_3$ is stored in memory only at the end of this process.

$$a_i = \frac{exp(score(\mathbf{h}_i, \mathbf{h}_t))}{\sum_{j=1}^{t-1} exp(score(\mathbf{h}_j, \mathbf{h}_t))} \tag{4}$$

$$score(\mathbf{h}_i, \mathbf{h}_t) = \begin{cases} single(\mathbf{h}_i) & (5) \\ combined(\mathbf{h}_i, \mathbf{h}_t) & (6) \end{cases}$$

We then merge $\mathbf{c}_t$ with the current state $\mathbf{h}_t$ using a concatenation layer[3], where $\mathbf{W}_c$ is a matrix of parameters and $\mathbf{b}_t$ is a bias vector.

$$\mathbf{h}'_t = tanh(\mathbf{W}_c[\mathbf{h}_t; \mathbf{c}_t] + \mathbf{b}_t) \tag{7}$$

We compute the next word probability using Eq.8 where $\mathbf{W}$ is a matrix of parameters and $\mathbf{b}$ is a bias vector.

[3]We also have experimented with using a dot product and a feedforward layer to combine $\mathbf{h}_t$ and $\mathbf{c}_t$ and also using only $\mathbf{c}_t$, but those results were far below previous work in RNN-LM so we do not report them here.

$$p(w_t|w_{<t}, x) = softmax(\mathbf{W}\mathbf{h}'_t + \mathbf{b}) \quad (8)$$

***Single* score.** This score is calculated for each $\mathbf{h}_i$ using just the information stored the state in itself. The score $single(\mathbf{h}_i)$ is defined as

$$single(\mathbf{h}_i) = \mathbf{v}_s \odot tanh(\mathbf{W}_s \mathbf{h}_i) \quad (9)$$

where the parameter matrix $\mathbf{W}_s$ and vector $\mathbf{v}_s$ are both learned by the attention mechanism and $\odot$ represents the dot product.

When applying the $single(\mathbf{h}_i)$ score, we can think of the score a_i as a scalar summary of the "absolute relevance" of the state $\mathbf{h}_i$. When a new state $\mathbf{h}_t$ is added to the buffer its scalar summary $\mathbf{a}_i$ is calculated by first using Eq.9 to get the score for the state and then applying a softmax function over the set of state scores including the score for the new state. Although the scores for each state do not change from one timestep to the next, applying the softmax results in recalculation of the distribution of the scalar summaries for all the states $\mathbf{h}_0, \ldots, \mathbf{h}_t$. As a result the $\mathbf{a}_i$'s for each state in Eq.3 changes from one prediction to the next as new states are added and the weight mass is distributed across more states.

***Combined* score.** This score is calculated for each $\mathbf{h}_i$ by combining the information from that state with the information from the current state $\mathbf{h}_t$. The score $combined(\mathbf{h}_i, \mathbf{h}_t)$ is defined as

$$combined(\mathbf{h}_i, \mathbf{h}_t) = \mathbf{v}_s \odot tanh(\mathbf{W}_s \mathbf{h}_i + \mathbf{W}_q \mathbf{h}_t) \quad (10)$$

where the parameter matrices $\mathbf{W}_s$ and $\mathbf{W}_q$ and vector $\mathbf{v}_s$ are learned by the attention mechanism, and $\odot$ is the same as in Eq. 9. Notice that because $\mathbf{W}_q \mathbf{h}_t$ does not depend on any other state and is used in the computations with all other h_i, we can efficiently compute it once and use the results in Eq. 10, thus reducing computation time.

The score $\mathbf{a}_i$ defined by $combined(\mathbf{h}_i, \mathbf{h}_t)$, can be understood as the "relative relevance" of state $\mathbf{h}_i$ to the current state $\mathbf{h}_t$. Using this attention mechanism the score for each $\mathbf{h}_i$ is different for each timestep according to its relevance to the current hidden state $\mathbf{h}_t$ of the RNN. Consequently, the scores for each $\mathbf{h}_i$ and the distribution over these scores changes from one timestep to the next. Using this scoring, the model can decide whether it should pay more attention to the current state, to a previous state or use past states to "supplement" the information for the next prediction. In §5 we present and analysis of how the model attends to different parts of its history as it generates a sequence of predictions.

4 Experiments

To evaluate our *Attentive* RNN-LMs we conducted experiments over the PTB (Marcus et al., 1994) and wikitext2 (Merity et al., 2017) datasets. We first describe the setup of our *Attentive* RNN-LM for the PTB (§4.1) and wikitext2 (§4.2) datasets and then discuss the results (§4.3). We compare our results on PTB to Zaremba et al. (2015) and Press and Wolf (2016) the best performing LSTM-LMs on the PTB, two memory augmented networks (Grave et al. (2017) and Merity et al. (2017)) and PTB state-of-the-art ensemble models of Zaremba et al. (2015). On wikitext2 we take (Merity et al., 2017), the creators of the dataset, and (Grave et al., 2017), the current state-of-the-art, as our baselines.

4.1 PTB Setup

We evaluate our *Attentive* RNN-LM over the PTB dataset using the standard split which consists of 887K, 70K and 78K tokens on the training, validation and test sets respectively.

We follow, in part, the parameterization used by Zaremba et al. (2015) and Press and Wolf (2016) with some changes. We trained an *Attentive* RNN-LM with 2 layers of 650 LSTM units using Stochastic Gradient Descent (SGD) with an initial learning rate of 1.0, halving the learning rate at each epoch after 12 epochs, to minimize the average negative log probability of the target words.

We train the models until we do not get any perplexity improvements over the validation set with an early stop counter of 10 epochs (i.e., patience of 10 epochs). Once the model runs out of patience, we rollback its parameters and use the model that achieved the best validation perplexity to calculate the perplexity over the test set. We initialize the weight matrices of the network uniformly in $[-0.05, 0.05]$ while all biases are initialized to a constant value at 0.0. We also apply 50% dropout (Srivastava et al., 2014) to the non-recurrent connections and clip the norm of the gradients, normalized by mini-batch size, at 5.0. In all our experiments, we follow Press and Wolf (2016) and tie the matrix $\mathbf{W}$ in Eq. (8) to be the embedding matrix (which also has 650 dimensions) used to represent the input words.

Contrary to Zaremba et al. (2015) and Press and Wolf (2016), we do not allow successive mini-batches to sequentially traverse the dataset. In other words, we follow the standard practice to reinitialize the hidden state of the network at the beginning of each mini-batch, by setting it to all zeros. This way, we do not allow the attention window to span across sentence boundaries[4]. We use all sentences in the training set, we truncate all sentences longer than 35 words and pad all sentences shorter than 35 words with a special symbol so all sentences are the same size. We use a vocabulary size of 10K words and a batch size of 32. All UNK words (following the pre-processing of (2015)) were kept during the training, validation and testing phases.

4.2 wikitext2 Setup

We also evaluate our *Attentive* RNN-LM over the wikitext2 dataset (Merity et al., 2017). We use the standard train, validation and test splits which consists of around 2M, 217K tokens and 245k tokens respectively. This dataset is composed of "Good" and "Featured" articles on Wikipedia.

There is an important difference between how we trained and tested our models on the wikitext2 dataset and how the baseline systems were trained and tested. Both Merity et al. (2017) and Grave et al. (2017) permitted the memory buffers of their systems to span sentence boundaries (and, indeed, they also did mini-batch traversal which allowed the memory buffers to traverse mini-batch boundaries) whereas we reset our systems memory at each sentence boundary. This difference is important because in the wikitext2 dataset the sentences are not shuffled and, therefore, are sequentially related to each other. As a result, systems that carry sequential information from previous sentences into the current sentence have an advantage in that they utilise contextual cues from the preceding sentence to inform the predictions at the start of the new sentence. By comparison, systems that reset their memory at the start of each sentence must reconstruct their context models from scratch and face a "cold-start" problem for the early predictions in the sentence.

The core reason why (Merity et al., 2017) and (Grave et al., 2017) did not reset their memories across sentence boundaries and we do is that these baseline systems use a fixed length memory whereas our "attention" mechanism has a variable length memory. A variable length memory has benefits in terms of both computational cost and the fact that the memory size is dynamically fitted to the context. However, just as the system designer for a fixed length memory LM must fix the memory size hyper-parameter in some fashion, so to the designer of a variable length memory LM must decide when the memory buffer is reset. For our work, we have decided to reset our memory buffer at sentence boundaries because many of the tasks for which LMs are used (e.g. NMT) work on a sentence by sentence basis. If required it would be possible for us to extend the memory buffer to the start of the preceding sentence (or some other landmark is the sequence history). However, this would incur extra computational cost, and as we shall see our *Attentive* RNN-LMs are still competitive on the wikitext2 dataset despite the fact that the baselines systems are given access to longer context sequences.

We trained an *Attentive* RNN-LM with 2 layers of 1000 LSTM units using Stochastic Gradient Descent (SGD) with an initial learning rate of 1.0, decaying the learning rate by a factor of 1.15 at each epoch after 14 epochs, to minimize the average negative log probability of the target words.

Similarly to the PTB model we also train this model with an early stop counter of 10 epochs and we initialize the weight matrices of the network uniformly in $[-0.05, 0.05]$ while all biases are initialized to a constant value at 0.0. We apply 65% dropout to the non-recurrent connections and clip the norm of the gradients, normalized by mini-batch size, at 5.0. In all our experiments, we also follow Press and Wolf (2016) and tie the matrix $\mathbf{W}$ in Eq. (8) to be the embedding matrix (which has 1000 dimensions for this model) used to represent the input words. We use all sentences in the training set, we truncate all sentences longer than 35 words and pad all sentences shorter than 35 words with a special symbol so all sentences are the same length. We use a vocabulary size of

[4]We also experimented to with successive mini-batches to sequentially traverse the dataset as in Zaremba et al. (2015) but the performance of the model dropped considerably so we do not report those results here.

33,278 and a batch size of 32. All UNK words (following the pre-processing of (2017)) were kept during the training, validation and testing phases.

4.3 Results

In Table 1 we report the results of our experiments on the PTB dataset. As we can see in this table, the *Attentive* RNN-LMs outperforms all other single models on the PTB dataset. Although *Attentive* RNN-LMs have less parameters (10M) than the large "regularized" LSTM-LMs (66M parameters), they were capable of reducing the perplexity over both validation and test sets. This result shows that using an *Attentive* RNN-LM it is possible to achieve better perplexity scores with far fewer model parameters. Furthermore, *Attentive* RNN-LMs are able to achieve roughly the same results as the averaging of 10 RNN-LM models (with no attention) of the same size.

In addition, there is little difference between the results of the *Attentive* RNN-LM with *single* score (Eq.9) and the *Attentive* RNN-LM with *combined* score (Eq.10) with the *single* score slightly outperforming the the *combined* score. We believe this is because the model using the $combined(\mathbf{h}_i, \mathbf{h}_t)$ score has more parameters to optimize and, thus, more difficulties in settling to a good local optima.

In Table 2 we report the results on the wikitext2 dataset. Despite the fact that we reset the memory for the *Attentive* RNN-LM at each sentence boundary whereas the caches for the baseline systems span sentence boundaries, our best *Attentive* RNN-LM is within 1 perplexity point of the system of (2017) (which is allowed to cache 2,000 previous hidden states), and has a lower perplexity than all of the other baselines.

5 Analysis of the Models

The purpose of our attention mechanism is to enable an RNN-LM to bridge long distance dependencies in language. Therefore, we expect the attention mechanism to select previous hidden states that are relevant to the current predictions. To analyse whether the attention mechanism is functioning as intend we analysed the evolution of attention weights in our *Attentive* RNN-LM as we calculated the perplexity for samples sentences using the models trained over the wikitext2[5].

[5]The behaviour of the models on wikitext2 is similar to that of the models trained and evaluated on the PTB dataset, so for space reasons we only present the wikitext2 analysis here.

Figure 2 show the evolution of attention weights, using both single and combined scoring, when calculating perplexities for 2 sentences containing nominal modifiers. In addition, Figure 3 show the evolution of attention weights for two sentences containing relative clauses, once again using both single and combined scoring. The words in the X-axis (horizontal) are the inputs at each timestep and the words in the Y-axis (vertical) are the next (or predicted) words. We suppressed weights that either equal to 1.0 (black squares) or 0.0 (white squares). Note that given the rounding to 4 decimal places, weights in some rows of the matrices may not sum to 1.0.

None of the attention mechanisms worked as a proper attention mechanism. In other words, none of the mechanisms generated larger weights for specific words in the sentence, in comparison to the other words in the same sentence. Comparing the attention weights generated by both *combined* score and *single* score for both sentences, it is striking that the distribution of attention weights is very similar. For both *Attentive* RNN-LM models the attention spreads out across the history in a relatively equal fashion.

Indeed, both models seem to take into consideration all previous states, creating a *smoothing effect* for the hidden states in the buffer. Therefore, no single state dominates the context vector by receiving a much larger attention weight than the others. We believe that this behaviour enables the models to bring forward information from the beginning of the sentence at the time it is making a prediction. This way, the models do not let information fade away from the context as it progresses to subsequent steps in a sequence and all previous information about the words that preceded the current timestep is available to the classifier in a manner that disregards recency.

As a consequence of the *smoothing effect*, the model does not necessarily need to store information about the context of the sequence in the recurrent connections of the RNN. This behaviour enable the model to retrieve information from the buffer to remember past words without relying solely on the RNN's internal "memory". Therefore, the model can maximize the features extracted about an input word, creating an advantage over other RNN-LMs that need to both extract features and keep context regarding the sequence in its connections.

Model	Params	Valid. Set	Test Set
Single Models			
Medium Regularized LSTM (Zaremba et al., 2015)	20M	86.2	82.7
Large Regularized LSTM (Zaremba et al., 2015)	66M	82.2	78.4
Large + BD + WT (Press and Wolf, 2016)	51M	75.8	73.2
Neural cache model (size = 500) (Grave et al., 2017)	-	-	72.1
Medium Pointer Sentinel-LSTM (Merity et al., 2017)	21M	72.4	70.9
Attentive LM w/ *combined* score function	14.5M	72.6	70.7
Attentive LM w/ *single* score function	14.5M	71.7	**70.1**
Model Averaging			
2 Medium regularized LSTMs (Zaremba et al., 2015)	40M	80.6	77.0
5 Medium regularized LSTMs (Zaremba et al., 2015)	100M	76.7	73.3
10 Medium regularized LSTMs (Zaremba et al., 2015)	200M	75.2	72.0
2 Large regularized LSTMs (Zaremba et al., 2015)	122M	76.9	73.6
10 Large regularized LSTMs (Zaremba et al., 2015)	660M	72.8	69.5
38 Large regularized LSTMs (Zaremba et al., 2015)	2508M	71.9	**68.7**

Table 1: Perplexity results over the PTB. Symbols: WT = weight tying (Press and Wolf, 2016); WD = weight decay and BD = Bayesian Dropout, both suggested by Gal and Ghahramani (2015).

Model	Params	Valid. Set	Test Set
Zoneout + Variational LSTM (Merity et al., 2017)	20M	108.7	100.9
LSTM-LM (Grave et al., 2017)	-	-	99.3
Variational LSTM (Merity et al., 2017)	20M	101.7	96.3
Neural cache model (size = 100) (Grave et al., 2017)	-	-	81.6
Pointer LSTM (window = 100) (Merity et al., 2017)	21M	84.8	80.8
Attentive LM w/ *combined* score function	50M	74.3	70.8
Attentive LM w/ *single* score function	50M	73.7	69.7
Neural cache model (size = 2000) (Grave et al., 2017)	-	-	**68.9**

Table 2: Perplexity results over the wikitext2.

Another interpretation of the *smoothing effect* is that it "reinforces" the signal in a similar fashion to residual connections in other RNNs and Deep Neural Networks architectures. Other RNN architectures use these residual connections as a shortcut to "reinforce" the signal of the current input and, thus, it still considers the current input only. The *Attentive* RNN-LM, however, uses all the previous hidden states to achieve a similar effect and create a stronger signal to the softmax classifier.

6 Conclusions

This paper proposes the use of attention mechanisms in RNN-LMs. These attention mechanisms enable an RNN-LM to consider information from its past when it is predicting the next word. We believe that this can help the LM to overcome the fading of relevant information as it traverses a long-distance dependency within a sequence and also to recover from a mistaken prediction by focusing on the context preceding the error.

Our results show that an *Attentive* RNN-LM outperforms both RNN-LM models that use and that do not use past information to predict the next word in a sequence when trained on the PTB dataset. Furthermore, our *Attentive* RNN-LM achieves this performance using far fewer units than the "standard" RNN-LM and, therefore, less model parameters. Our results also show that our *Attentive* RNN-LM achieves similar results to an ensemble that averages over 10 similar sized (in terms of number of LSTM units) RNN-LMs.

In addition, our results demonstrate that our *Attentive* RNN-LM achieves similar to state-of-the-art results over the wikitext2 dataset. It is an interesting result given that we do not allow our model to look beyond the boundaries of the current sequence it is processing, whilst the state-of-the-art model is allowed to store 2,000 previous states in its cache.

In future work we plan to (a) test the performance of ensembles of *Attentive* RNN-LMs and (b) to study the use of the *Attentive* RNN-LM as the decoder within an NMT system.

Acknowledgments

This research was partly funded by the ADAPT Centre. The ADAPT Centre is funded under the SFI Research Centres Programme (Grant 13/RC/2106) and is co-funded under the European Regional Development Fund. Giancarlo D. Salton would like to thank CAPES ("Coordenação de Aperfeiçoamento de Pessoal de Nível Superior") for his Science Without Borders scholarship, proc n. 9050-13-2.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, volume abs/1409.0473v6.

Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561, Austin, Texas. Association for Computational Linguistics.

Michal Daniluk, Tim Rocktäschel, Johannes Welbl, and Sebastian Riedel. 2017. Frustratingly Short Attention Spans in Neural Language Modeling. *5th International Conference on Learning Representations (ICLR'2017)*.

Yarin Gal and Zoubin Ghahramani. 2015. A theoretically grounded application of dropout in recurrent neural networks.

Edouard Grave, Armand Joulin, and Nicolas Usunier. 2017. Improving neural language models with a continuous cache. *5th International Conference on Learning Representations (ICLR'2017)*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. volume 9, pages 1735–1780.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling.

Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, pages 114–119.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. *5th International Conference on Learning Representations (ICLR'2017)*.

Haitao Mi, Baskaran Sankaran, Zhiguo Wang, and Abe Ittycheriah. 2016. Coverage embedding models for neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 955–960, Austin, Texas. Association for Computational Linguistics.

Tomas Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH 2010, 11th Annual Conference of the International Speech Communication Association, Makuhari, Chiba, Japan, September 26-30, 2010*, pages 1045–1048.

Ofir Press and Lior Wolf. 2016. Using the output embedding to improve language models. volume abs/1608.05859.

Holger Schwenk, Anthony Rousseau, and Mohammed Attik. 2012. Large, pruned or continuous space language models on a gpu for statistical machine translation. In *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, pages 11–19.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Ke M. Tran, Arianna Bisazza, and Christof Monz. 2016. Recurrent memory network for language modeling. *arXiv*, abs/1601.01272.

Kelvin Xu, 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 *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2048–2057.

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2015. Recurrent neural network regularization. volume abs/1409.2329.

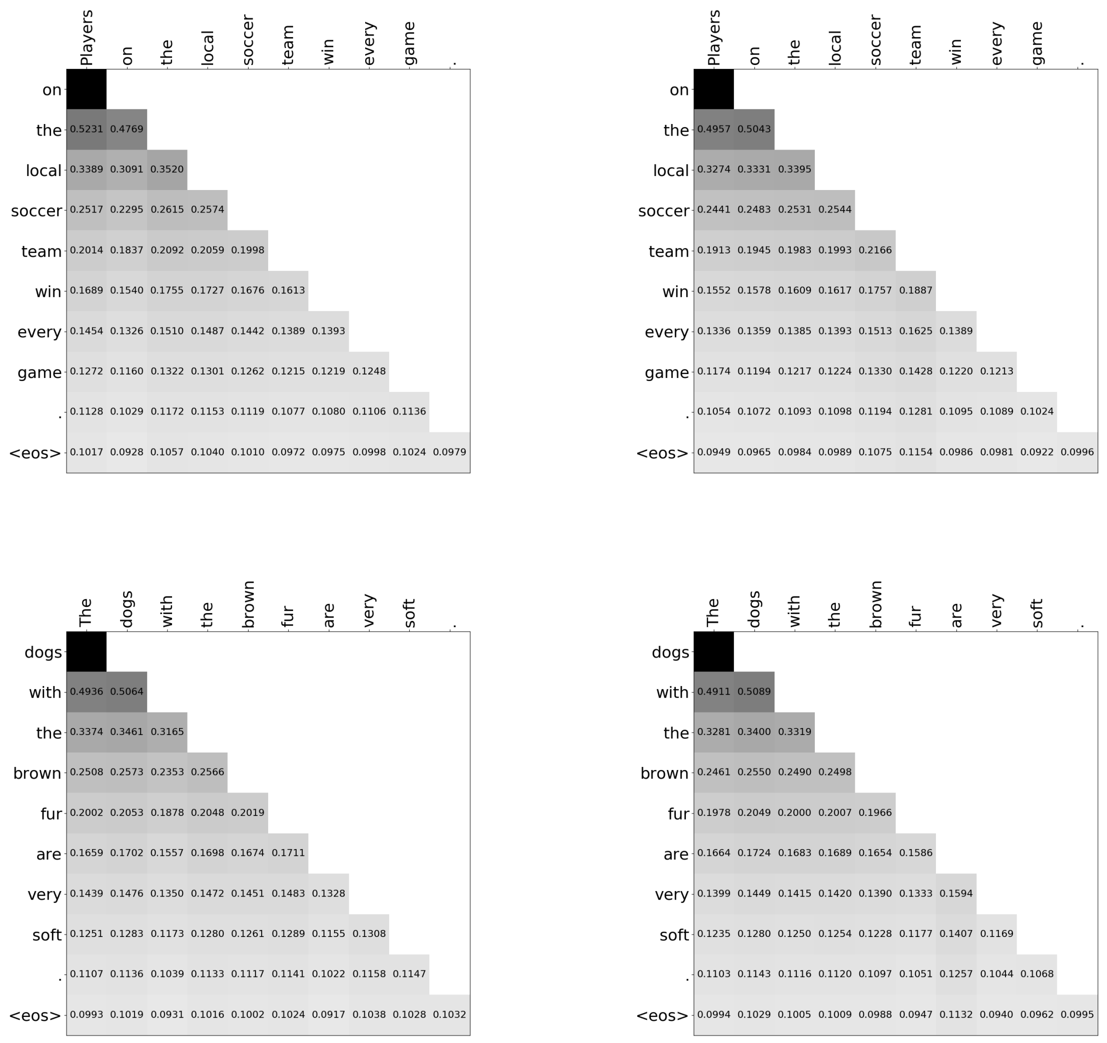

Figure 2: Plot of attention weights for two sentences containing nominal modifiers. On the left column are the attention weights calculated by the *combined* score. On the right column are the attention weights calculated by the *single* score. The words in the X-axis (horizontal) are the inputs at each timestep and the words in the Y-axis (vertical) are the next (or predicted) words. We suppressed weights that either equal to 1.0 (black squares) or 0.0 (white squares). Note that given the rounding to 4 decimal places, weights in some rows of the matrices may not sum to 1.0.

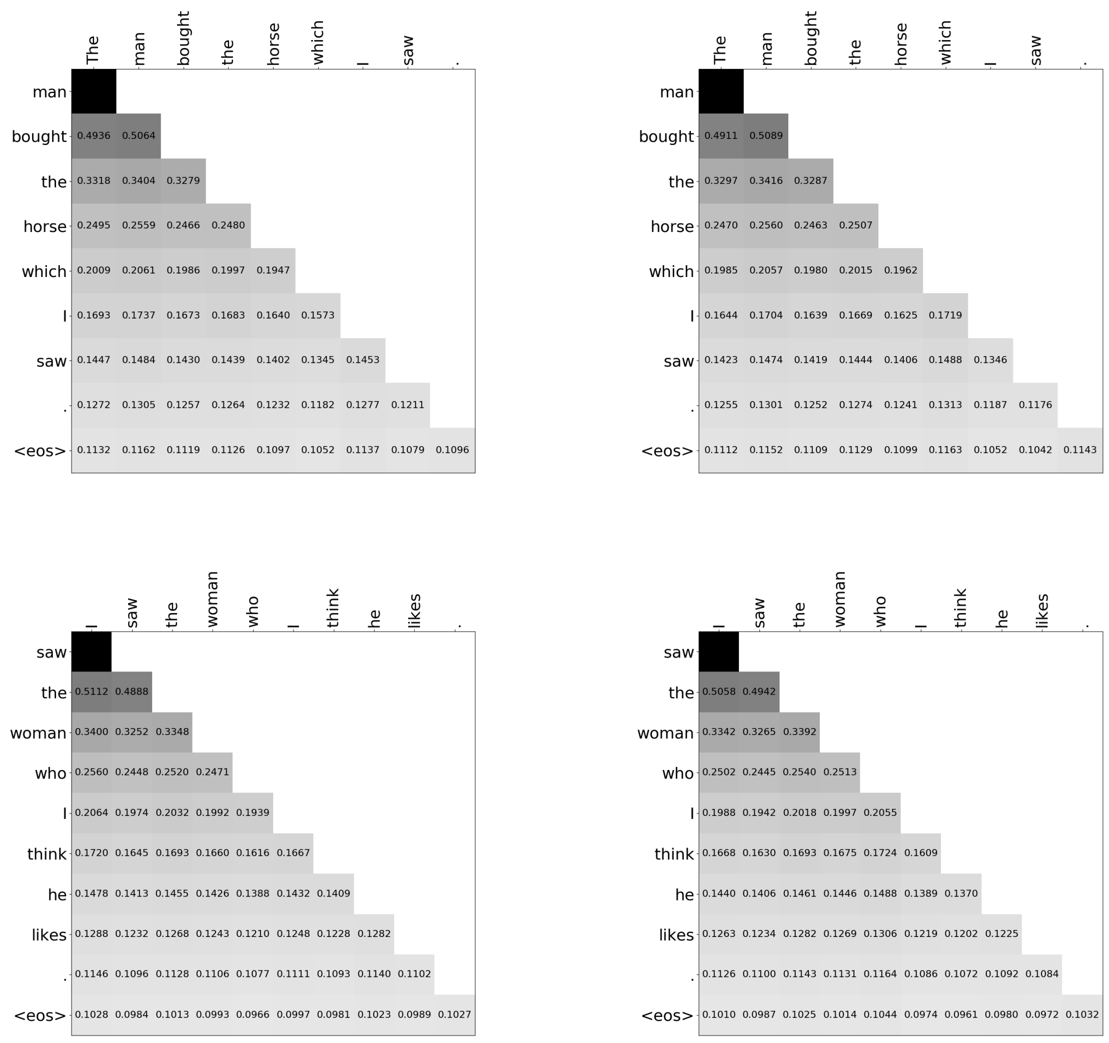

Figure 3: Plot of attention weights for two sentences containing relative clauses. On the left column are the attention weights calculated by the *combined* score. On the right column are the attention weights calculated by the *single* score. The words in the X-axis (horizontal) are the inputs at each timestep and the words in the Y-axis (vertical) are the next (or predicted) words. We suppressed weights that either equal to 1.0 (black squares) or 0.0 (white squares). Note that given the rounding to 4 decimal places, weights in some rows of the matrices may not sum to 1.0.