Positional Attentive Language Modelling

Gautier Dagan
University of Amsterdam
gautier.dagan@student.uva.nl

ABSTRACT

In this paper we propose using the Positional Attention mechanism in an Attentive Language Model architecture. We evaluate it compared to an LSTM baseline and standard attention, and find that it surpasses standard attention on both validation and test perplexity on both the Penn Treebank and Wikitext-02 datasets while still using less parameters. Using the attention distribution vectors we are able to analyze the differences between the two mechanism and offer insight into the potential benefits of positional attention.

1 INTRODUCTION

Language modelling (LM) is traditionally framed as the task of predicting a word given the previously occurring sequence of words and encompasses the understanding of language by a model. LM is often used to pre-train embeddings which serve as the base of any NLP tasks. Many recent papers have shown that state of the art results can be obtained on tasks such as Neural Machine Translation, Sentiment Classification or Question Answering by using pre-trained descriptive embeddings such as BERT (Devlin et al., 2018) or ELMo (Peters et al. 2018) [6] [12]. Since embeddings are trained using LM, any successful language model has the potential to bring immediate gains to a slew of other problems.

Our attention mechanism is done by using a separate recurrent neural network block referred to as a *position generator* and basic *building blocks* to identify positions to attend to in the sequence (see section 3.1.1). A Gaussian probability density function is used to fuzzy the attention and make it more human-like where humans focus on single words at a time but can still perceive surrounding words due to the eccentricity effect (Carrasco et al., 1995) [3]. Unlike the traditional attention mechanism, we inject the network with positioning information as it is reading a sequence. While this is also done in the transformer architecture (Vaswani et al., 2017) using positional embeddings, *positional attention* differs in its implementation and can be thought as the specialization of an attention head towards position rather than content [16].

Positional attention focuses on the position of the word relative to the sequence rather than content. This has been shown by Dubois et al. (2019, unpublished) to improve the extrapolation capabilities and sample efficiency of a model on the lookup table dataset. With a specialized position component, the model is able to better and more quickly generalize to lookup sequences of lengths which it has never seen before.

LM requires an understanding of the implicit connections between words in a sequence in order to predict the next word more accurately. When understanding text, one must not only base their understanding on the content or the meaning of the words, but also the order in which they appear (position). Thus, injecting the network with the knowledge of the word positions and letting it learn how the position of words impact the occurrence of following word should translate to better performance.

Project AI 2, February 2019, University of Amsterdam
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 RELATED WORK

Recurrent neural networks due to their sequential structure have long history of been used for the task Language Modelling. Using a multi-layer Long Short-Term Memory (LSTM) network, an embedding layer, an output (decoding) layer and some training tricks, Merity et al. has shown that one is able to approach state of the art perplexity on the major LM datasets [10].

However, modern RNNs still fail to explicitly reason over the relations between tokens and their underlying structure (Cheng et al, 2016) [4]. Additionally, if the sequence is long, then the underlying Markov property of the RNN updates fails since the current state might have trouble representing all the tokens it has previously seen [4]. Zhang et al. also notes that modern LSTM use gating mechanism in order to control the influence of each token on the final representation, but those are not conditioned on the entire sequence [17]. This can lead the LSTM to misrepresent or incompletely represent an encoded token, and fail to model the position of entities in a sequence [17]. The outlined limitations of RNNs motivate our choice of testing positional attention in a language model setting.

The modern attention mechanism for language was originally proposed by Badhanau et. al. in 2014 as a mechanism for allowing the model to focus on separate parts of the input for translation depending on the time step [1]. Since then, attention has been widely adopted as a technique for improving the performance of any Sequence-to-Sequence or Sequence-to-One models.

Attention can be tagged on top of recurrent neural network architectures by allowing the decoder to attend to separate parts of the encoded states h_t. Typically, this is done by first computing a score for each previous time steps (queries) and comparing each to the current state (key). This score can simply be the dot product, or it can be computed using more complex methods such as an MLP. The scores are then soft-maxed or normalized:

$$a_i = \frac{exp(score(\boldsymbol{h_i}, \boldsymbol{h_t}))}{\sum_{j=1}^{t-1} exp(score(\boldsymbol{h_j}, \boldsymbol{h_t}))} \tag{1}$$

The context vector at time step t can then be found by summing all previous states h_i with their attention score a_i:

$$c_t = \sum_{i=1}^{t-1} a_i \boldsymbol{h_i} \tag{2}$$

Using a context vector partially solves the decoder-encoder bottleneck caused by passing a single final encoded hidden state to a decoder. Instead of solely using encoded state h_t to decode at time step t, the decoder is also able to use c_t which is based on all hidden states until h_{t-1}. This context vector encodes information

the network found relevant for the current state and allows the previously sequential recurrent network to model relations and structures between tokens [4].

In 2017, Vaswani et al. proposed the transformer architecture which effectively replaced the need for a recurrent component and instead uses self-attention, batch-normalization and fully connected feed-forward layers to obtain state of the art results in machine translation [16]. In the transformer architecture the key, value and query components are all generated from the same sequence (self-attention):

$$Attention(K, V, Q) = softmax(\frac{QK^\intercal}{\sqrt{n}})V \tag{3}$$

It also makes use multi-head attention, which can be seen as the concatenation of multiple attentions in parallel.

In order to understand the current position of the token it is encoding, the transformer architecture uses absolute positional embeddings that are generated using a sinusoidal function and injected along with the input embeddings [16]. This absolute positioning can be a limitation when dealing with text data, as for instance the negation 'not' negates the phrase to the right of it, and regardless of the specific or absolute position. These structural and relative interactions occur often in text and explain why alternate positioning methods might be beneficial [2].

Shaw et al. expands the transformer architecture by adding a special relative position encoding instead of the absolute originally proposed [15]. This uses the representations of relative positions or distance between sequence elements in the self attention step to obtain a better BLEU score than the original architecture [15]. Further improvements to the transformer's positioning were proposed recently in the Transformer-XL paper (Dai et. al 2019), where the authors introduce a new relative positioning mechanism that separate content specific keys and the position keys in the self attention score [5]. The Transformer-XL is also able the cache the hidden state sequence computed for the previous segment or batch so that it can be used as an extended context [5]. This allows it to outperform the current state of the art in Language Modelling on the Penn-Tree Bank, WikiText-103 and enwiki8 corpora.

Bilan et al. also showed that relative positioning can be applied in combination with a self-attention layer on the TACRED (Entitity Extraction) dataset to improve performance with respect to the state of the art [2]. Their approach uses an additional position-aware attention layer that takes into account positions of the query and the object in the sequence [2]. However, unlike positional attention, this approach enforces relative positioning by considering all pairwise interactions in the sequence.

A neural network which is able to generalize across tasks is called a meta-learner. In 2018, Mishra et. al [11] proposed the meta-learner architecture called SNAIL which combines temporal convolution layers with soft attention in order to design an architecture that can generalizes to a variety of tasks. This hand-crafted meta-learner was able to beat SOTA on a variety of tasks, by simply having a well designed generalizable architecture. The exploration of positional attention in Language Modelling is a first step in seeing if similarly it can become a meta-learner component. It could then added to an existing network in order to endow it with positional understanding.

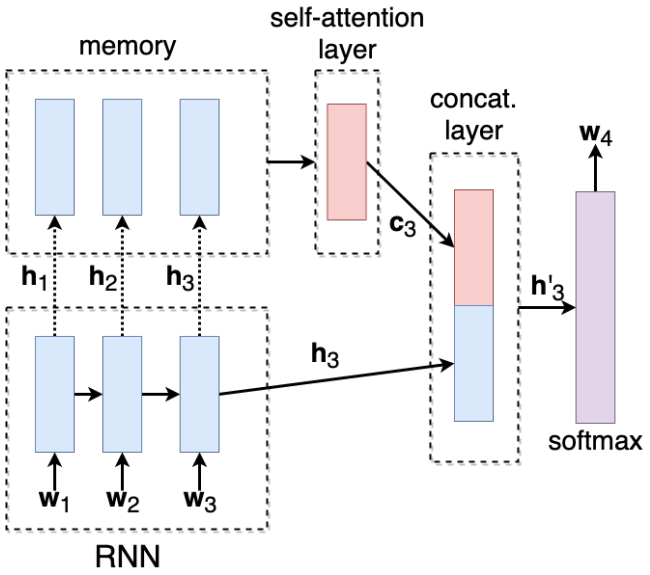


Figure 1: Architecture of the Attentive Language model proposed by Salton et al. [14]. In order to predict word w_4, the previous encoded steps of all previously seen words are attended onto using a self-attention mechanism. This constitutes the context vector, which is then concatenated to the encoded state h_3 of word w_3 (last encoded state). Softmax is applied to find a probability distribution over vocabulary size and predict w_4.

The first Attention-based RNN Language Model was proposed by Mei et al. (2016) [9]. It was evaluated on a dialogue task and dynamically increased the attention scope along with the conversation. They showed that his allowed the model to outperform more complex models by allowing for flexible, long-distance memory. Salton et al. proposed a similar Attentive Language Model and tested it on the task of Language Modelling specifically [14]. They found that it was able to achieve perplexities of 70.1 on the PTB dataset (close to SOTA at the time of publication) with significantly less parameters than the best models [14]. The attentive model proposed by Salton is the one that we have decided to modify in order to switch out standard attention with positional attention.

3 METHOD

3.1 Model

As can be seen in Figure 1, designing an attentive language model essentially consists of adding an attention layer on top of the RNN encoder to find a context vector describing the time steps 1 to $t-1$. This context vector is then concatenated with the encoded h_t and used to output a prediction for the next word in the sequence. In practice the concatenated vector is passed through a concatenation layer to reduce the dimensions back to the original embedding dimensions such that the weights between the embedding layer and output layer are tied. This is a language modelling technique suggested by Press and Wolf (2016) and Inan et al. (2016) [13] [7].

It is important to note that unlike a traditional attention layer which has access to all hidden states of the encoder, this model cannot attend on the encoded future. This means that the attention 'memory' is dynamic and grows with the sequence. When predicting word $t+1$, the model is only allowed to attend at the encoded words up to $t-1$. If we allowed the network to attend to future states then it would be trivial for it to always attend to the next word and obtain a perfect prediction scheme.

Additionally in our implementation of both the standard and positional attention, self-attention is done on the hidden states in the memory. Therefore the h_3 from the RNN to the memory is not used as the single key for the current context vector. The attention key, query, and values are all found using the states in memory at that time step. This was done because Salton et al. showed that the model performed slightly better using self-attention on the memory rather than using the hidden state h_3 as a key [14].

For the standard attention mechanism we use equations 1 and 2. And since we are doing self attention, we can calculate the score using the following:

$$score(\boldsymbol{h_i}) = \boldsymbol{W_2} \odot tanh(\boldsymbol{W_1 h_i}) \tag{4}$$

Where $\boldsymbol{W_1}$ and $\boldsymbol{W_2}$ are both learnable parameters (linear layers) of dimensions $d \times d$ and $d \times 1$ respectively if d is the hidden dimension of the RNN layer, and $\odot$ represents the dot product.

3.1.1 Positional Attention. To implement the positional attention we first run the encoded hidden states through a positioning generator which is a smaller RNN layer with hidden dimension $\tilde{d}$, and is used to obtain compressed encoding of the states:

$$\tilde{\boldsymbol{h}} = RNN_{\text{pos\_gen}}(\boldsymbol{h}) \tag{5}$$

Similarly the original positional attention, we use three building blocks, namely the previous μ step (initialized to 0 at step 0), the length of the interval of one step normalized to 1, and the current position in the sequence. Thus we can define our building block matrix as b_t:

$$\boldsymbol{b_t} = \begin{bmatrix} \mu_{t-1} & \frac{1}{N} & \frac{t}{N} \end{bmatrix} \tag{6}$$

Where N is the total length of the sentence. We can then find μ_t and σ_t:

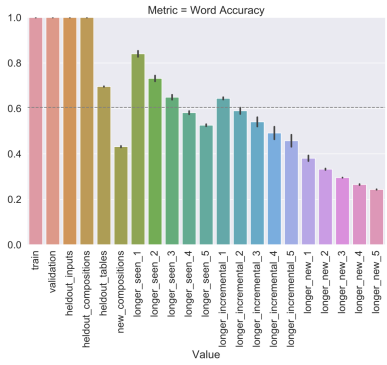
$$\boldsymbol{\mu\_weights_t} = ReLU(\boldsymbol{W_\mu}\tilde{\boldsymbol{h}}) \tag{7}$$

$$\mu_t = max(\boldsymbol{\mu\_weights_t} \odot \boldsymbol{b_t}, \frac{t}{N}) \tag{8}$$

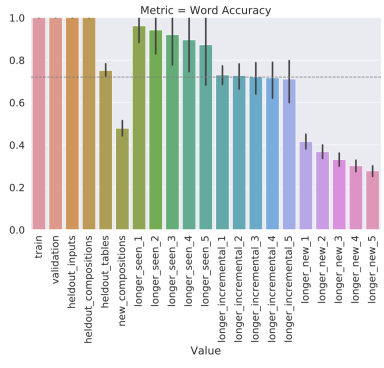
$$\sigma_t = sigmoid(\boldsymbol{W_\sigma}\tilde{\boldsymbol{h}}) \tag{9}$$

Where $\boldsymbol{W_\mu}$ and $\boldsymbol{W_\sigma}$ are learnable parameters of dimensions $\tilde{d} \times 3$ and $\tilde{d} \times 1$ respectively. We clamp μ_t between 0 and $\frac{t}{N}$ using a ReLU activation function and max. This is done in order to prevent the attention from 'looking' forward in the sentence, since position in whole sentence is represented in percentile (0, 1). The sigmoid activation is used to generate the σ in order to obtain a number in the interval of (0, 1), while trying to avoid the edge cases.

We then use a Gaussian probability density function to obtain an attention score γ for each encoded hidden state h_i that occurred before time step t:



(a) Vanilla RNN Word accuracy



(b) Positional RNN Word accuracy

Figure 2: Word (token) accuracy of running vanilla RNN vs. RNN with Positional Attention on the extended table lookup dataset

$$\gamma_{it} = exp(\frac{(p_i - \mu_t)^2}{2\sigma_t^2 + \epsilon}) \tag{10}$$

$$\gamma_{it} = \frac{\gamma_{it}}{\sum_{i=1}^{t-1} \gamma_{it}} \tag{11}$$

Here p_i encodes the percentile positional description at position i with respect to the full sentence:

$$p_i = \frac{i}{N} \tag{12}$$

Also note that in equation 10 we use an ϵ term in order to prevent a potential divide by zero, which can happen if σ approaches zero. We can then use our attention score γ_{it} to generate the positional context vector similarly to equation 2:

$$c_t = \sum_{i=1}^{t-1} \gamma_{it} \boldsymbol{h_i} \tag{13}$$

All the coding implementation of the equations described above can be found in the project's public repository[1] using the pytorch framework.

[1] https://github.com/gautierdag/pytorch-attentive-lm

4 EXPERIMENTS

4.1 Lookup Table Task

The first step of this experiment consisted of replicating the effects of positional attention in the original lookup table task. The lookup table composition task was proposed by Liska et al. in order to test compositional behavior and demonstrate that a standard RNN could theoretically learn to behave compositionally [8]. This is done by showing lookup tables as atomic units in training along with some compositions, and holding out other compositions. A successful model then has to extrapolate the pattern of the composition from its atomic parts, and learn to replicate it to unseen combinations. The task was extended by Dubois et al. to include longer sequences and different interactions between lookup tables.

Figure 2 shows the results of running the positional attention on the extended lookup task in comparison to using a simple RNN. It shows that the positional attention model is able to better generalize to longer unseen sequences and obtain a degree of compositionality that the baseline model is unable to achieve. This indicates that the mechanism of positional attention

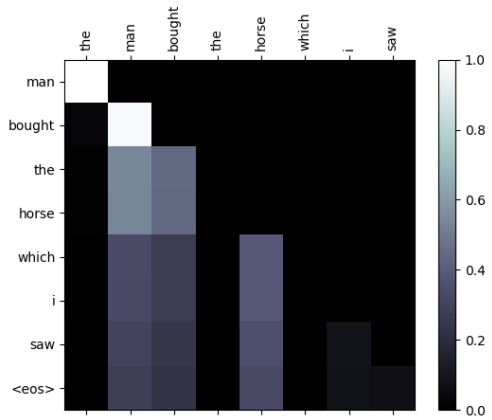
4.2 Language Modelling

Pre-processing was done according to Salton et al. [14] so as to replicate the Attentive RNN. Unlike the traditional batching applied to the LM corpora, in this case, it is split such that every sentence is its own training example. We select a max sentence length of 35, clipping sentences which are longer, and padding shorter sentences. We use a fix vocabulary size similar to what is done in the literature: 10,000 + 1 (padding) token for PTB and 33,278 + 1 (padding) for WikiText-02. The pre-processing of WikiText is done using the nltk punkt python package, and we split each line at every sentence ending order to obtain one per line.

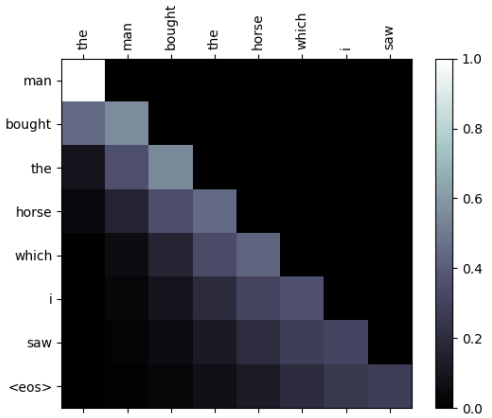
A key difference with state of the art Language Models and the attentive architecture, is that we are not propagating encoded hidden states beyond a batch. This means that our network is susceptible to the cold start problem since we reset its hidden state at every new batch. This was done because of the dynamic nature of the attention span. Dividing the document cleanly into sentences allows us to control the attention span to within a sentence and not deal with overflowing the attention over other older sentences. Furthermore, increasing the attention window would also require additional computational resources and could always be expanded ulteriorly.

The baseline model used was proposed by Merity et al. and consists of a three-layer LSTM LM. For the purpose of this paper, it was re-implemented and tested using our new dataset split. It has an embedding dimension of 400 and LSTM hidden size of 400. As for all three models, the weights between the embedding layer and final output layer are tied according to Press and Wolf and Inan et al. [13] [7]. As suggested by Merity et al. a batch size of 20 is used. The parameters of this model remain the same for both datasets.

Our implementation of the Attentive LM proposed by Salton et. al, differs slightly from the original parameters. We use an embedding size of 400 and two-layer LSTM hidden size of 400 on both the PTB and the the Wikitext-02 datasets. This helps us compare models with the same number of parameters, but also unlike the original authors we obtained better perplexity scores using these



(a) Standard Attention



(b) Positional Attention

Figure 3: Comparison of the attention distribution for the same sentence using the standard Attention vs proposed Positional Attention. The words in the X-axis are the inputs at each time step and the words in the Y-axis are the targets

parameters. Salton et al. reports using embedding and hidden sizes of 650 on PTB and 1,000 on Wikitext-02, yet we found that the model suffered from high over-fitting with this many parameters.

Our Positional Attentive LM uses an embedding size of 400 and two-layer LSTM hidden size of 400 on both PTB and Wikitext-02. We use a single-layer LSTM with a hidden size of 20 for the positioning generator.

For all models SGD was used with an aggressive initial learning rate of 30 and a learning rate plateau scheduler which decrease the learning rate by a factor of 0.5 if the validation loss does not decrease within 5 epochs. The norm of the gradients are clipped at 0.25. We initialize all the weight matrices of the network uniformly in range $[-0.1, 0.1]$. A Dropout of 50% is applied to the non-recurrent connections, and a dropout of 20% to the recurrent connections. Using early stopping with a patience of 10 epochs, we return the

Model	Params	Valid. Set	Test Set
LSTM Baseline (Merity et al., 2017) [10]	7.86M	**66.77**	**64.96**
Attentive Language Model (Salton et al., 2017) [14]	7.06M	79.09	76.56
Positional Attentive Language Model (ours)	**6.9M**	72.69	70.92

Table 1: Perplexity results for each implemented model tested on the PTB dataset

Model	Params	Valid. Set	Test Set
LSTM Baseline (Merity et al., 2017)[10]	7.86M	**72.43**	**68.50**
Attentive Language Model (Salton et al., 2017) [14]	7.06M	78.43	74.37
Positional Attentive Language Model (ours)	**6.9M**	74.39	70.73

Table 2: Perplexity results for each implemented model tested on the Wikitext-02 dataset

model with the best validation loss and evaluate on the test set. We set the max epochs to be 500 for the PTB dataset and 750 for the Wikitext-02 dataset. Cross-entropy loss over the words is used, and we calculate perplexity by taking its exponent.

4.2.1 Penn Tree-Bank. We show the results for the PTB dataset in table 1. All three models shown in the table were implemented and ran using the same pre-processing and sentence split.

We find that the baseline LSTM model was the best at predicting the next word in a sequence, with a validation perplexity of 66.77, and a test perplexity of 64.96. However our positional model showed an improvement over the Attentive LM with a validation perplexity of 72.69 and test perplexity of 70.92. The Attentive LM obtained the worse results with a validation perplexity of 81.72 and test perplexity of 79.86.

4.2.2 WikiText-02. We show the results for the Wikitext-02 dataset in table 2. Again, we find that the baseline LSTM model performed best, with a validation perplexity of 72.43, and a test perplexity of 68.50. The Attentive LM, again obtained the worst results with a validation perplexity of 81.02 and test perplexity of 76.62. Our positional model obtained a validation perplexity of 74.39 and test perplexity of 70.73.

Figure 3 shows the result of plotting the attention distribution vectors for both the Attentive and Positional Attentive models trained on the Wikitext dataset. We can see from this plot the difference in attending based on content rather than using our position generator and attending using the position attention mechanism.

The content attention will focus on the same words over a sequence, deemed on the content of those and how relevant they are. Standard or content-only attention will thus not focus on filler words like 'the' and 'which', as can be seen in figure 3. This can be an advantage for tasks in which a stronger signal of what was in the context is needed, since this attention focuses on a few key encoded states. On the other hand, the positional attention model distributes the attention with a bias towards most recently seen words. In doing so it almost creates a context window of sorts around the current word, and can be seen as augmenting the information from the base RNN layer with an additional focus on the immediate previous steps.

Both attention types however fail to outperform the baseline LSTM model in both datasets. While the Positional Attention model does have the least number of trainable parameters it is not necessarily faster due to the dynamic attention that must be performed on top of the RNN layer.

5 CONCLUSION

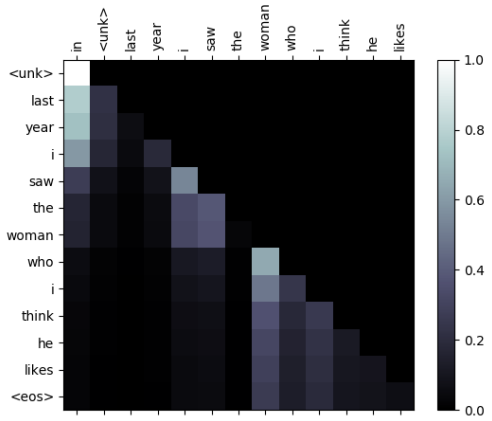
In this paper we have shown the effects of running positional attention in comparison to standard attention on a LM task. This is the first real application of the positional attention mechanism and showed that there are tasks in which it promises to be more successful than the standard attention mechanism.

Comparing both attentions in particular proved interesting, as we see the advantage to using a model which can attend to specific regions in the input using positions. The task of predicting next word however could have impacted heavily the positional attention, as it focused on the narrow context preceding each word it would predict. This suggest that positional attention would have been unable to attend to distant but relevant words in the context, as in practice the best next word to choose is often a result of a few words preceding it. The issue with long term dependencies also plagues traditional LSTM, and is often solved using traditional attention. Thus, it remains to be seen what effect could mixing both attention mechanisms bring to Language Modelling.

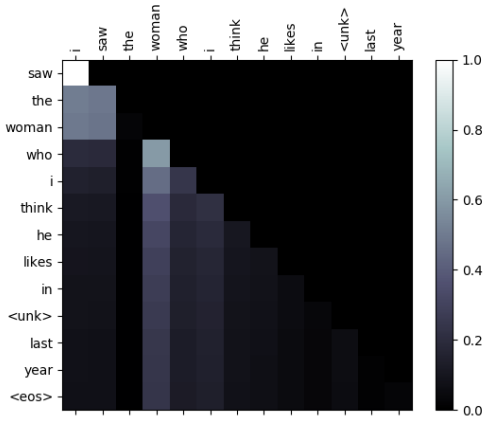
Additionally only the self-attention method was tested as it showed the most promise in Salton et. al [14]. Using the current encoded hidden state h_t as a key to the queries (all previous hidden states), could potentially help in promoting a more diverse attention span in the positional mechanism since it would indicate to the attention the information contained at step t.

Finally more fine-tuning could be done on the attentive models in order to bring their scores to similar levels as the baseline. Minimal tuning was unfortunately done due to computation and time constraints. The baseline however uses parameters which have been very fine tuned by Merity et. al and therefore this could have skewed the results in its favor [10].

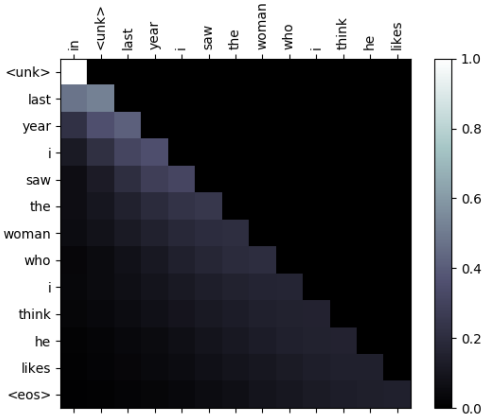
These results conclude this independent research project. Beyond this paper, analysis, re-implementations of the baseline models in the pytorch framework, and the development of a Positional Attentive RNN-LM, I also worked on improving the ILLC's Seq2Seq



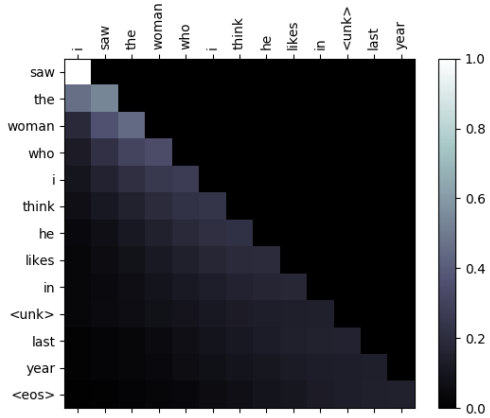
(a) Standard Attention Sentence 1



(b) Standard Attention Sentence 2



(c) Positional Attention Sentence 1



(d) Standard Attention Sentence 2

Figure 4: Extra Comparisons of the attention distribution for the same sentence using the standard Attention vs proposed Positional Attention. Again we see that content attention seems to completely ignore filler words and instead focus on the key words in the sequence like 'woman' in this example. On the other hand the positional attention focuses its attention to the few last states of a sequence. in this manner it helps the RNN gets extra information about the most recent states previous to state t.

machine library. In my pull requests to the library, I implemented callbacks and a default language model, as well as maintained the library and upgraded it from pytorch 0.4 to pytorch 1.0.0. Thank you to my supervisor Elia Bruni for advising and encouraging me on this project, and to Yann Dubois for deepening my understanding of attention mechanisms.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. *CoRR* abs/1409.0473 (2014). arXiv:1409.0473 http://arxiv.org/abs/1409.0473

[2] Ivan Bilan and Benjamin Roth. 2018. Position-aware Self-attention with Relative Positional Encodings for Slot Filling. *CoRR* abs/1807.03052 (2018). arXiv:1807.03052 http://arxiv.org/abs/1807.03052

[3] Marisa Carrasco, Denise L. Evert, Irene Chang, and Svetlana M. Katz. 1995. The eccentricity effect: Target eccentricity affects performance on conjunction searches. *Perception & Psychophysics* 57, 8 (01 Nov 1995), 1241–1261. https://doi.org/10.3758/BF03208380

[4] Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long Short-Term Memory-Networks for Machine Reading. *CoRR* abs/1601.06733 (2016). arXiv:1601.06733 http://arxiv.org/abs/1601.06733

[5] Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *arXiv preprint arXiv:1901.02860* (2019).

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

[7] Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. *CoRR* abs/1611.01462 (2016). arXiv:1611.01462 http://arxiv.org/abs/1611.01462

[8] Adam Liska, Germán Kruszewski, and Marco Baroni. 2018. Memorize or generalize? Searching for a compositional RNN in a haystack. *CoRR* abs/1802.06467 (2018). arXiv:1802.06467 http://arxiv.org/abs/1802.06467

[9] Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. Coherent Dialogue with Attention-based Language Models. *CoRR* abs/1611.06997 (2016).

arXiv:1611.06997 http://arxiv.org/abs/1611.06997

[10] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and Optimizing LSTM Language Models. *CoRR* abs/1708.02182 (2017). arXiv:1708.02182 http://arxiv.org/abs/1708.02182

[11] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. 2017. Meta-Learning with Temporal Convolutions. *CoRR* abs/1707.03141 (2017). arXiv:1707.03141 http://arxiv.org/abs/1707.03141

[12] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR* abs/1802.05365 (2018). arXiv:1802.05365 http://arxiv.org/abs/1802.05365

[13] Ofir Press and Lior Wolf. 2016. Using the Output Embedding to Improve Language Models. *CoRR* abs/1608.05859 (2016). arXiv:1608.05859 http://arxiv.org/abs/1608.05859

[14] Giancarlo D. Salton, Robert J. Ross, and John D. Kelleher. 2017. Attentive Language Models. In *IJCNLP*.

[15] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. 2018. Self-Attention with Relative Position Representations. *CoRR* abs/1803.02155 (2018). arXiv:1803.02155 http://arxiv.org/abs/1803.02155

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762 http://arxiv.org/abs/1706.03762

[17] Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D. Manning. 2017. Position-aware Attention and Supervised Data Improve Slot Filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 35–45. https://doi.org/10.18653/v1/D17-1004