Autoregressive Attention for Parallel Sequence Modeling

Dillon Laird
Department of Computer Science
Stanford University
Stanford, CA 94305
`dalaird@cs.stanford.edu`

Jeremy Irvin
Department of Computer Science
Stanford University
Stanford, CA 94305
`jirvin16@cs.stanford.edu`

Abstract

We introduce an autoregressive attention mechanism for parallelizable character-level sequence modeling. We use this method to augment a neural model consisting of blocks of causal convolutional layers connected by highway network skip connections. We denote the models with and without the proposed attention mechanism respectively as Highway Causal Convolution (Causal Conv) and Autoregressive-attention Causal Convolution (ARA-Conv). The autoregressive attention mechanism crucially maintains causality in the decoder, allowing for parallel implementation. We demonstrate that these models, compared to their recurrent counterparts, enable fast and accurate learning in character-level NLP tasks. In particular, these models outperform recurrent neural network models in natural language correction and language modeling tasks, and run in a fraction of the time.

1 Introduction

Sequence-to-sequence tasks including natural language correction (NLC) and language modeling (LM) have been dominated by recurrent neural networks (RNNs) [2, 12, 13]. RNNs and variants such as long short-term memory networks (LSTMs) [23] provide an effective framework for capturing longer-term dependencies in sequences, making them well-suited for modeling language as well as numerous other settings. However, these models tend to suffer from long training times due to their inherent serial nature. Moreover, in the sequence-to-sequence framework, these recurrent models often compress the source sequence into a fixed-size representation, forcing the model to maintain an information-efficient memory. This problem only worsens with longer sequences, especially due to vanishing gradient [14]. This makes it difficult to model particularly long-term dependencies, rendering character-level tasks very difficult. Character-level models, however, have attractive traits, including the ability to handle out of vocab words. Furthermore, these models are ideal in tasks such as NLC where many words might be misspelled, making it difficult and unnatural to model language errors with word-level techniques.

Convolutional neural networks (CNNs) provide a natural framework for performing parallel computation as well as robustly modeling large context windows. For these reasons and many others, CNNs have historically demonstrated great success in image tasks [24]. Recently CNNs have shown promising results in various natural language processing tasks as well [1, 3, 5, 11]. Character-level CNN variants have been shown to outperform LSTMs while reducing computation time [3, 11]. Moreover, ByteNet has near state-of-the-art scores on the Hutter dataset using dilated, causal convolutions [1] and Gated Convolutional Networks has near state-of-the-art on WikiText-103 word-level test set. Both of these datasets require models which capture long-term dependencies, exemplifying that convolutions are able to succeed at such tasks.

Importantly, attention has been applied successfully in many NLP tasks ranging from translation [12, 13] to natural language correction [17]. Attention has recently been added to convolutional architectures for language modeling [3], but to our knowledge it has not yet been utilized in causal convolutional models. In this work we introduce an autoregressive attention mechanism which can be used to augment causal convolutional models without any computational overhead. With the ability to model large contexts efficiently and effectively, causal CNN architectures with autoregressive attention improve the feasibility of using character-level models in text processing. We demonstrate that these models are adept at tasks which have been dominated by recurrent models, and run in a fraction of the time.

2 Related Work

Numerous neural models have recently been proposed for character-level sequence modeling [2, 3, 11, 15, 16, 17, 18]. The majority of these models are recurrent and leverage an addition or small change to the standard recurrent model to increase model complexity or ease training. For example, HyperNetworks generates non-shared weights for an LSTM, and Recurrent Highway Networks increase depth in the recurrent transition using Highway layers, rather than stacking recurrent layers in the usual way [2, 15]. Moreover, it is standard in these recurrent models to include a form of attention in sequence-to-sequence tasks, most notably in machine translation [12, 13]. Attention is an effective mechanism of alignment and can be viewed as a way to pass information across timesteps between the encoder and decoder. However, it important to note that these models (both with and without attention) are inherently sequential and thus cannot be implemented in parallel, and furthermore, struggle to capture long-term dependencies.

There have been several recent advancements in using CNNs in sequence modeling tasks in order to tackle these issues. WaveNet uses stacks of dilated, causal convolutions in order to model long-range temporal dependencies for generating raw audio [7]. ByteNet generalizes this idea to the encoder-decoder framework for sequence modeling, using a dilated convolutional network as an encoder and a dilated, causal convolutional network as a decoder [1]. CNNs applied at the character level have been shown to succeed in text classification, often better than word-based CNNs and recurrent networks [19]. Gated Convolutional Network (GCN) utilizes gated blocks of causal convolutions for word-level language modeling [5]. These methods surpass the performances of many of their recurrent counterparts, and can be run in parallel.

Our method combines ideas from many of these recent successes and introduces a parallelizable attention mechanism. Explicitly, we use dilated, causal convolutions with highway gates (akin to GCN and Highway Networks [4, 5]) and autoregressive attention in the decoder which attends to previous decoder outputs in parallel. This allows for efficient training and predicting on character-level sequence-to-sequence tasks, whilst leveraging the advantages of convolutional architectures. Our model consistently outperforms standard recurrent models at character-level language modeling on Penn Treebank, and achieves near state-of-the-art performance on natural language correction.

3 Methods

Our model consists of three main building blocks: causal convolutions, highway connections, and autoregressive attention. We discuss how these blocks are combined in an encoder-decoder framework for general sequence modeling in the final part of this section.

3.1 Causal Convolutions

The use of causal convolutions [1] is the essential factor which allows the model to be run in parallel. In standard sequence modeling using recurrent networks, the ground truth is typically fed into the decoder at each timestep to ease training, which we also adapt in our setting. Causal convolutions ensure that the model does not violate the order in which predictions must be made by peeking ahead and using the future ground truth in its current prediction. Moreover, it is important to note that the causal convolution outputs a representation for each timestep - there is no temporal compression. Due to this reason and preservation of causality, the model can be run in parallel.

Formally, 1-dimensional causal convolutions can be viewed as masked convolutions (where the right half of the kernel is masked) or shifted convolutions (where the kernel is shifted left and the

[1]Causal convolutions are used in the decoder. This is elaborated in Section 3.4.

input is padded). We implement the latter for computational efficiency. Note that due to this causal structure, the prediction $p(x_{t+1}|x_1, \ldots, x_t)$ output at timestep t only contains information from previous timesteps. Figure 1 illustrates this general structure.

Because of this, causal convolutions are ideal for making fast predictions, as the operation can be done in parallel over all time steps [2]. However, they must be stacked many times to increase the receptive field of each output. You can see from Figure 1 that after three layers each node has a receptive field of four. Dilations have been proposed which increase the receptive field exponentially in the number of layers [7, 1], but they fail to improve performance in our experiments and thus we do not discuss them in this work.

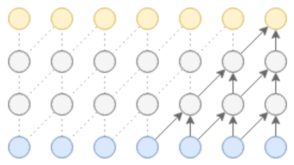

Figure 1: Causal convolution with kernel size 2 and 3 layers.

3.2 Highway Connection Blocks

We stack causal convolutions in blocks in a similar way as done in Gated Convolutional Networks [5]. Each block consists of some number of causal convolutions followed by a nonlinearity such as a rectified linear unit (ReLU) with the exception of the last layer which does not include the nonlinearity. Using $*$ as the causal convolution operator, we have $\text{CausalConv}(\boldsymbol{X}; \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W} * \boldsymbol{X} + \boldsymbol{b}$ for some parameters $\boldsymbol{W}, \boldsymbol{b}$. The number of times L these are stacked is referred to as the number of block layers. The input to the block is then connected to the output via a gated connection similar to highway connections in [4], described below. Our gate is a function of the last layer of the block, rather than the input to the block as in [4]. The entire block is called a *highway connection block*, and we stack B blocks together (each with the same number of layers). Formally, for each block $b = 1, ..., B$,

$$h_{b_\ell}(\boldsymbol{X}) = \text{ReLu}\left(\text{CausalConv}(\boldsymbol{X}; \boldsymbol{W}_{b_\ell}, \boldsymbol{b}_{b_\ell})\right), \ \ell = 1, ..., L-1$$
$$h_{b_L}(\boldsymbol{X}) = \text{CausalConv}\left(h_{b_{L-1}} \circ \cdots \circ h_{b_1}(\boldsymbol{X}); \boldsymbol{W}_{b_L}, \boldsymbol{b}_{b_L}\right)$$
$$g_b(\boldsymbol{X}) = \sigma\left(\text{CausalConv}(h_{b_L}(\boldsymbol{X}); \boldsymbol{W}_{g_b}, \boldsymbol{b}_{g_b})\right)$$
$$h_b(\boldsymbol{X}) = g_b(\boldsymbol{X}) \odot \boldsymbol{X} + (1 - g_b(\boldsymbol{X})) \odot h_{b_L}(\boldsymbol{X})$$

where $\boldsymbol{X}$ denotes the input to the layer or block (the input to the first layer of the first block is the character embeddings of the sequence), $h_{b_\ell}(\boldsymbol{X})$ denotes the output of the ℓth layer of the bth block, $g_b(\boldsymbol{X})$ denotes the highway gate of the bth block, and $h_b(\boldsymbol{X})$ denotes the output of the bth block. A visual representation of a single block can be seen in Figure 2.

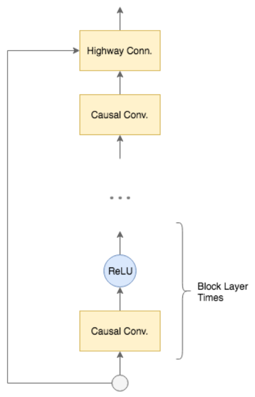

Figure 2: Highway Connection Block.

[2]This can be viewed as temporal batching.

Without the attention mechanism, we use a final causal convolution to embed the output of the last block in vocab space and perform a softmax over each timestep in parallel:

$$\text{Probabilities} = \text{Softmax}(\text{CausalConv}(h_B(\boldsymbol{X}); \boldsymbol{W}_o, \boldsymbol{b}_o)).$$

3.3 Autoregessive Attention

The final component is what we call the *autoregressive attention mechanism*. This computes attention over the previous (decoder) outputs in such a way as to avoid using future information to make current predictions (to preserve the notion of causality). If we have a set of T outputs from the model for each timestep $o_1, \ldots, o_T$, then each output o_t is used to compute a context vector over $o_1, \ldots, o_{t-1}$. This can quickly be computed with a matrix multiply and a mask. Here we use inner products for our attention scores as in [13]. First let $\boldsymbol{O} = [o_1, \ldots, o_T]^T$ be a $T \times H$ matrix where H is the dimension of the output vectors. Then we calculate the matrix product and add the following mask $\boldsymbol{M}$:

$$\boldsymbol{O}\boldsymbol{O}^T + \boldsymbol{M} = \begin{bmatrix} o_1^T o_1 & o_1^T o_2 & \cdots & o_1^T o_T \\ o_2^T o_1 & o_2^T o_2 & \cdots & o_2^T o_T \\ \vdots & \vdots & \ddots & \vdots \\ o_T^T o_1 & o_T^T o_2 & \cdots & o_T^T o_T \end{bmatrix} + \begin{bmatrix} -\infty & -\infty & \cdots & -\infty \\ 0 & -\infty & \cdots & -\infty \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -\infty \end{bmatrix}$$

The function of the mask is to change all of the elements in the upper-right triangle and the diagonal to $-\infty$ [3]. This ensure that all the scores that would introduce future information into the attention calculation are equal to 0 after the softmax:

$$\text{Softmax}(\boldsymbol{O}\boldsymbol{O}^T + \boldsymbol{M}) = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ \alpha_{2,1} & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \alpha_{T,1} & \cdots & \alpha_{T,(T-1)} & 0 \end{bmatrix}$$

We then compute the final context vectors by $C = \text{Softmax}(\boldsymbol{O}\boldsymbol{O}^T + \boldsymbol{M})\boldsymbol{O}$ so each row is $C_i = \sum_{t=1}^{i-1} \alpha_{i,t} o_t$, and concatenate with the corresponding current timesteps in $h_B(\boldsymbol{X})$ before performing the final causal convolution embedding and then softmax. For clarity, this method differs from standard attention in the literature because it attends to previous hidden states in the decoder, rather than all hidden states in the encoder.

3.4 Encoder-Decoder

For the encoder decoder-framework we employ both traditional attention over the encoder and autoregressive attention over the decoder. For the encoder, we use a similar architecture as described in Section 3.2 but we use regular convolutions instead of causal convolutions as there is no need to worry about future information since predictions are made in the decoder. The decoder can then be run in parallel using causal convolutions. At this point autoregressive attention can be run on the decoder over the decoded outputs and then normal attention can be run over the encoded outputs.

For the traditional attention component we used inner products for the scoring function. We experimented with a scoring function similar to that used [25],

$$A(e_i, d_j) = \sum_{d=1}^{n} V_d \tanh((e_i U)_d) \tanh((d_j W)_d),$$

where e_i is the output of the encoder and d_j is the output of the decoder, but we could not obtain good results.

To combine autoregressive attention and traditional attention we first calculate autoregressive attention on the decoder which outputs a context vector $c_i^{(ARA)}$ and the decoder output o_i. We then use o_i for traditional attention obtaining a context vector $c_i^{(Attn)}$ and then we pass the concatenated vector $[c_i^{(ARA)}; c_i^{(Attn)}; o_i]$ to the final projection/softmax layer.

[3]In practice we deal with the first row separately to avoid NaNs.

4 Experiments

4.1 Character-level Language Modeling

4.1.1 Character-level Penn TreeBank

Our model is evaluated on the character-level prediction task on the Penn TreeBank (PTB) corpus [27] using the train/dev/test split as commonly used in the literature. The training set has 5M characters, valid 400K, and test 450K. The vocab size is 50.

We use Convolution Aware Initialization [8] with a scale of 0.001, which yields slightly better results and faster convergence times than normal and uniform initialization. We apply a dropout of 0.85 (keep probability) and L2 weight decay of 0.005. We optimize using standard SGD with a learning rate of 0.4 that is adaptively annealed by 0.5 based on validation scores until 8 decays. We train in batches of 20 sequences.

The best two models from our hyperparameter searches [4] are tested with and without autoregressive attention. They both use 7 blocks, 3 layers, and no dilation. The smaller models use 256 feature maps (and character embedding dimension) and a kernel size of 3, whereas the larger models use 300 feature maps and a kernel size of 4. With these settings, the models without autoregressive attention have a theoretical perceptive field of 70 [5]. We train on overlapping sequences of length 80, where the smaller models are allowed 20 characters of context before evaluating prediction [6] and the larger models are allowed 40 characters of context. We compare our results with several recent baselines. These and our best results (bits-per-character[7]) are summarized in Table 1.

Each of our models beats a standard 2-layer LSTM with significantly less parameters, and the regular causal convolutional models beat a more advanced LSTM which uses layer normalization. Our Causal Conv model, with 5.5M parameters, performs about 18K predictions per second, while ARA-Conv gets 17.5K predictions per second and a basic 2-Layer LSTM with roughly twice the number of parameters achieves 2K predictions per second. A 2-Layer LSTM with 5.5 million parameters, comparable to the size of our best small model, achieves 3.6K predictions per second. Hence the convolutional models achieve a factor of 5 in computational gains.

However, we see that the autoregressive attention mechanism did not aid in performance, getting very similar performance to the standard causal convolution in both cases. The ARA-Conv model strictly encapsulates the modeling power of the Causal Conv (by zeroing out the context vector the models are equivalent), so this slight drop in performance is likely due to optimization noise. We elaborate on this in Section 4.1.4.

4.1.2 Character-level WikiText-2

We also evaluate our model on the larger, more difficult raw character-level WikiText-2 corpus [21]. The training set has 11M characters, valid 1.1M, and test 1.3M. The vocab size is 193. These experiments are mainly meant to test the effectiveness of the autoregressive attention mechanism on a dataset which requires long-term modeling capability. We do this by varying the sequence length and context parameters to observe their effect on the ARA-Conv model.

Again we use Convolution Aware Initialization with a scale of 0.001. We apply a dropout of 0.825 (keep probability) and L2 weight decay of 0.005. We optimize using standard SGD with a learning rate of 0.4 that is adaptively annealed by 0.5 based on validation scores until 8 decays. We train in batches of 32 sequences.

We test regular causal convolutions with and without autoregressive attention. We use models with 9 blocks, 3 layers per block, a kernel size of 4, and 300 feature maps. With these settings, the models without autoregressive attention have a theoretical perceptive field of 90. For both of these models, we train on overlapping sequences of length 80 and allow 40 characters of context before making

[4]Grid search for the larger model can be seen in Figure 4 in Section A.1

[5]Each block 3 layers and kernel size 4 adds 10 characters to the perceptive field.

[6]To ensure we make a prediction on every character in the dataset, we do not mask the first set of characters for which we do not have context. For all other sets, we use a mask over the predictions in the context sequence when computing the loss.

[7]BPC=$\log_2$(perplexity).

Model	Test	Valid	Param. Count
ME n-gram (Mikolo et al., 2012)	1.37		
LSTM, 1000 units (Ha et al., 2016)	1.312	1.347	4.25M
2-Layer LSTM, 1000 units (Ha et al., 2016)	1.281	1.312	12.26M
Layer Norm LSTM, 1000 units (Ha et al., 2016)	1.267	1.300	4.26M
Zoneout RNN (Krueger et al. 2017)	1.252	1.362	
Layer Norm HM-LSTM (Chung et al., 2016)	1.23		
BN-LSTM (Cooijmans et al., 2017)	1.22		
2-Layer Norm HyperLSTM, 1000 units (Ha et al., 2016)	1.219	1.245	14.41 M
Causal Conv	1.266	1.300	5.5M
Causal Conv	1.264	1.299	10.1M
ARA-Conv	1.268	1.301	5.5M
ARA-Conv	1.267	1.304	10.1M

Table 1: Bits-per-character on the Penn Treebank validation and test sets.

predictions (again except for the first set of characters), and do the same for pairs (sequence length, context) of $(160, 80)$, and $(240, 120)$[8]. Our results for these six experiments (bits-per-character) are summarized in Table 2.

We see that a sequence length of 160 and context of 80 yield the best performance using our models.

Model	Valid	Test	Param Count
Causal Conv (80, 40)	1.630	1.661	13.1M
Causal Conv (160, 80)	1.628	1.660	13.1M
Causal Conv (240, 90)	1.653	1.685	13.1M
ARA-Conv (80, 40)	1.646	1.678	13.1M
ARA-Conv (160, 80)	1.623	1.658	13.1M
ARA-Conv (240, 120)	1.628	1.660	13.1M

Table 2: Bits-per-character on the WikiText-2 character-level test set.

4.1.3 Visualizing Autoregressive Attention

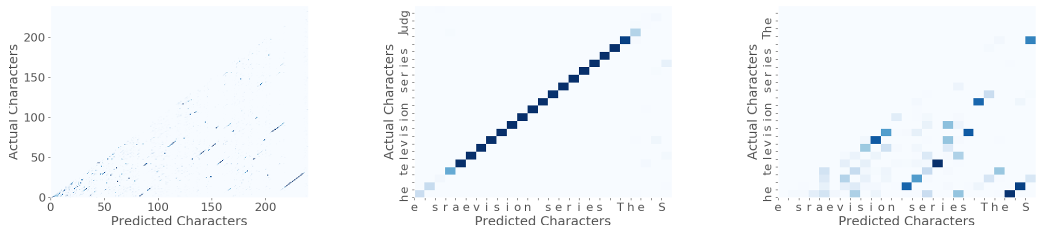

Figure 3: Attention score for ARA-Conv (240,120) on WikiText-2. The x-axis represents the predicted characters while the y-axis represents the actual characters. The second plot is the bottom right corner of the first plot zoomed in. The third plot is the first plot shifted up to focus on the diagonal (current prediction).

Table 3 illustrates three graphs each visualizing the autoregressive attention for the ARA-Conv (240,120) model on WikiText-2. The first graph shows us autoregressive attention over the entire 240

[8]In Table 2 we only show a context of 90 for the Causal Conv model since this is its maximum perceptive field, unlike the ARA-Conv model which can attend to the entire history of the sequence.

characters. We can see particularly strong sequence of attention scores in the lower right portion of the graph. The second plot shows that the mechanism is attending to the words "television series". If we then look at what the actual sequence of words it is trying to predict, we can see from the third graph that it is indeed "television series".

These results are promising and show us that the model is able to refer back to previous words. We hypothesize that by using the second to last layer for attention the model will be able to better incorporate the attention information by passing it through a final layer before it makes a prediction.

4.1.4 Autoregressive Attention in Language Modeling

The results shown in Table 1 show that the autoregressive attention mechanism does not help on the PTB dataset. The differences shown in the table likely reflect optimization noise between experiments. However the visualizations demonstrate that the model is attending to correct parts of the past sequence. We believe the model is unable to fully utilize this information in making predictions.

It appears from Table 2 that ARA-Conv is better able to handle increased sequence length than Causal Conv. Sequence length can be seen as temporal batching and may make optimization for Causal Conv more difficult (similar to how the batch size parameter affects optimization).

We hypothesize that using attention on the second to last layer (or multiple inner layers) might improve results. This way the network can use the final layer to better make sense of the attention output and decide whether or not to use it for prediction. In this sense, attention acts as a way to retrieve past states much like [28, 29] are able to effectively use similar history in making predictions.

4.2 Sequence-to-Sequence Experiments

For our sequence-to-sequence experiments on translation and natural language correction we used the same set of parameters. We used 7 blocks for the encoder and decoder each with 3 layers per block. We used 256 feature maps with a kernel size of 3 and a batch size of 128. Our keep probability was 0.95 and our initialization scale was 0.1 with truncated normal initialization. We used a L2 regularization of 0.0025. For learning we used stochastic gradient descent with a learning rate of 1.0 and we annealed by 0.5 based on validation scores a maximum of 8 times.

4.2.1 Character-level Machine Translation

For character-level machine translation we use IWSLT German-English 2015 translation dataset [26]. The dataset contained about 165K training sentences and 567 development sentences. We did not use the test set for this project. We use this dataset to evaluate the effect of autoregressive attention in sequence-to-sequence tasks.

Model	Valid Perplexity	BLEU
Causal Conv	2.538	8.04
Causal Conv + Attn	1.887	**11.36**
Causal Conv + Attn + ARA	1.900	10.81

Table 3: IWSLT 2015 development set performance.

We can see here that adding an attention mechanism clearly increases perplexity and BLEU but adding autoregressive attention does not significantly change results and might even hurt performance slightly. We should note that the Causal Conv + Attn perplexity ranged from 1.887 to 1.921. There is still a large gap to close in BLEU score - other character-level models like [3] get 19.41 BLEU on IWSLT 2014 test set.

4.2.2 Character-level Natural Language Correction

For natural language correction we use a training set similar to [17]. We combine data from several datasets. We use two datasets for training, the Lang-8 Corpus [22] and the CoNLL 2014 Shared Task training set. The Lang-8 Corpus is a set of language learner text from the language exchange site

Lang-8. We specifically use the English learner corpus which contains 100K training and 1K test entries with 550K and 5K parallel sentences. We use 5K sentences from the training set for validation. CoNLL 2014 contains 57K training and 1K test sentences from essays from English learners with corresponding corrected sentences.

When training we use a special domain token to indicate if the sentence is from Lang-8 or CoNLL 2014 and when decoding on CoNLL 2013 we use the domain token corresponding to the CoNLL 2014 training sentences.

Model	Valid	P	R	$F_{0.5}$
RNN (Xie et al. 2016)		**0.49**	0.10	0.28
Causal Conv	1.285	0.41	0.12	0.27
Causal Conv + Attn	1.272	0.41	0.15	**0.31**
Causal Conv + Attn + ARA	1.272	0.34	**0.16**	0.28

Table 4: CoNLL 2013 development set performance.

For these tests again we see that attention seems to help while autoregressive attention seems to hurt the results slightly. We do obtain comparable results to [17] whose RNN model obtained $F_{0.5}$ of 0.28 showing that this model is a viable option for natural language correction.

5 Conclusion

We plan to apply similar models to larger datasets for language modeling, such as the WikiText-103 character-level dataset, as well as the Hutter Prize Wikipedia task. Additionally, we hope to test the model on more tasks requiring long-term dependencies such as the LAMBADA dataset.

Moreover, there have been recent further advancements in using convolutional architectures for sequence modeling (namely version 2 of [1]), which we hope to pull ideas from to enhance our own models. These include different blocks (Residual Units for MT, Multiplicative Units for LM), using dilation, layer normalization [9], and more.

Finally, based on our observations, the information that the autoregressive attention mechanism may not be fully utilized by the model in its current formulation. We hope to explore many different avenues of applying autoregressive attention such as applying it between multiple different layers, or even allowing higher layers to attend to lower layers.

Acknowledgments

We thank Ziang Xie for helping us with the code for the project, Hao Sheng for applying autoregressive attention to LSTMs, and Chris Manning and Richard Socher for their advice on the project.

References

[1] Aaron van den Oord, Alex Graves, Koray Kavukcuoglu. Neural Machine Translation in Linear Time. *arXiv:1610.10099, 2016.*

[2] David Ha, Andrew Dai, Quoc V. Le. HyperNetworks. *arXiv:1609.09106, 2016.*

[3] James Bradbury, Stephen Merity, Caiming Xiong, Richard Socher. Quasi-Recurrent Neural Networks. *arXiv:1611.01576, 2016.*

[4] Rupesh Kumar Srivastava, Klaus Greff, Jurgen Schmidhuber. Highway Networks. *arXiv:1505.00387, 2015.*

[5] Yann N. Dauphin, Angela Fan, Michael Auli, David Grangier. Language Modeling with Gated Convolutional Networks. *arXiv:1612.08083, 2016.*

[6] Arron van den Oord, Nal Kalchbrenner, Koray Kavukcuoglu. Pixel Recurrent Neural Network. *arXiv:1601.06759, 2016*

[7] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu. Wavenet: A Generative Model for Raw Audio. *arXiv:1609.03499, 2016.*

[8] Armen Aghjanyan. Convolution Aware Initialization. *arXiv:1702.06295, 2017.*

[9] Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton. Layer Normalization. *arXiv:1607.06450, 2016.*

[10] Gao Huang, Zhuang Liu, Kilian Q. Weinberger, Laurens van der Maaten. Densely Connected Convolutional Networks. *arXiv:1608.06993, 2016.*

[11] Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush. Character-Aware Neural Language Models. *arXiv:1508.06615, 2015.*

[12] Dzmitry Bahdanau, Kyunghyun Cho, Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473, 2014.*

[13] Minh-Thang Luong, Hieu Pham, Christopher D. Manning. Effective Approaches to Attention-based Neural Machine Translation. *arXiv:1508.04025, 2015.*

[14] Razvan Pascanu, Tomas Mikolov, Yoshua Bengio. On the difficulty of training Recurrent Neural Networks. *arXiv:1211.5063, 2013.*

[15] Julian Georg Zilly, Rupesh Kumar Srivastava, Jan Koutník, Jürgen Schmidhuber. Recurrent Highway Networks. *arXiv:1607.03474, 2017.*

[16] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, Chris Pal. Zoneout: Regularizing RNNs by Randomly Preserving Hidden Activations. *arXiv:1606.01305, 2017.*

[17] Ziang Xie, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, Andrew Y. Ng. Neural Language Correction with Character-Based Attention. *arXiv:1603.09727, 2016.*

[18] Kyuyeon Hwang, Wonyong Sung. Character-Level Language Modeling with Hierarchical Recurrent Neural Networks. *arXiv:1609.03777, 2017.*

[19] Xiang Zhang, Junbo Zhao, Yann LeCun. Character-level Convolutional Networks for Text Classification. *arXiv:1509.01626, 2016.*

[20] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, Yoshua Bengio. A Structured Self-attentive Sentence Embedding. *arXiv:1703.03130, 2017.*

[21] Stephen Merity, Caiming Xiong, James Bradbury, Richard Socher. Pointer Sentinel Mixture Models. *arXiv:1609.07843, 2016.*

[22] Toshikazu Tajiri, Mamoru Komachi, Yuji Matsumoto. Tense and Aspect Error Correction fro ESL Learners Using Global Context. In *Association for Computational Linguistics: Short Papers, 2012.*

[23] Sepp Hochreiter, Juergen Schmidhuber. Long Short-Term Memory. *Neural Computation, 1997.*

[24] Yann LeCun, Yoshua Bengio. Convolutional Networks for Images, Speech and Time Series. *The Handbook of Brain Theory and Neural Networks, 1995.*

[25] Noam Shazeer, Azalia Miroseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *arXiv:1701.06538, 2017.*

[26] M. Cettolo, C. Girardi, M. Federico. 2012. Inventory of Transcribed and Translated Talks. In *Proc. of EAMT, pp. 260-268*, Trento, Italy.

[27] Mitchell Marcus, Beatrice Santorini, Maryann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics, 19(2).*

[28] Edouard Grave, Armand Joulin, Nicolas Usunier. Improving Neural Language Models with a Continuous Cache. *arXiv:1612.04426, 2016.*

[29] Stephen Merity, Caiming Xiong, James Bradbury, Richard Socher. Pointer Sentinel Mixture Models. *arXiv:1609.07843, 2016.*

A Appendix

A.1 Penn TreeBank Error Analysis

The following error analysis is performed on the best 5.5M parameter model without autoregressive attention on PTB. First we show a grid search of hyperparameters centered around the hyperparameters of this model in Figure 4. Notice that performance (measured in bits-per-character) worsens around these settings in every case except for the number of blocks, where the performance plateaus at 7 blocks. The performance of block layers and feature maps likely demonstrate this behavior due to suboptimal regularization or optimization.

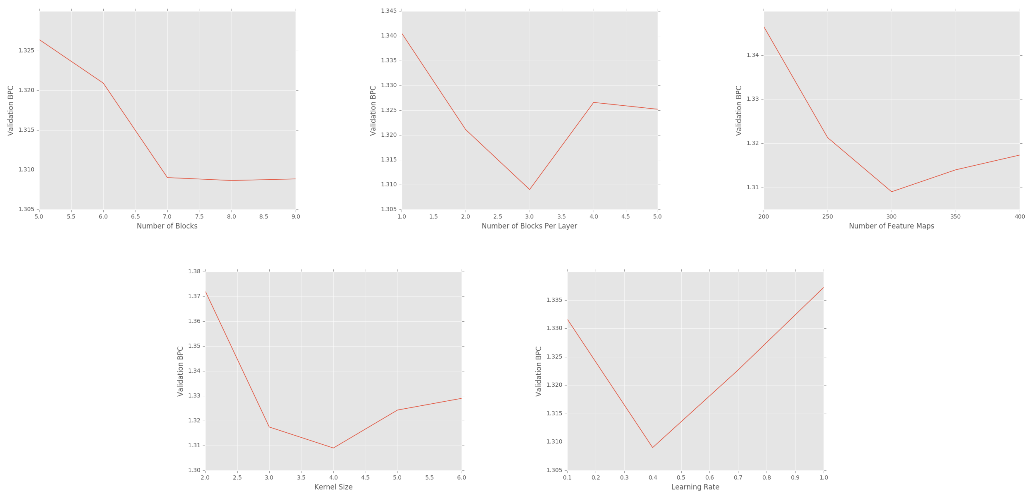

Figure 4: Grid search among various hyperparameters. Plots show feature changes against validation bits-per-character (lower is better) on the Penn TreeBank dataset. Starting from the top left: number of blocks, number of block layers, number of feature maps per layer, kernel size, learning rate used in SGD.

Next, we examine the confusion matrix of this model on the validation set in Figure 5. The model commonly predicts `t`, `a`, `s`, `c`, `<`, `\n` instead of the correct character. It almost always gets `>` correctly, meaning it is able to identify `<unk>` correctly once it has seen `<unk`. ADD MORE ANALYSIS

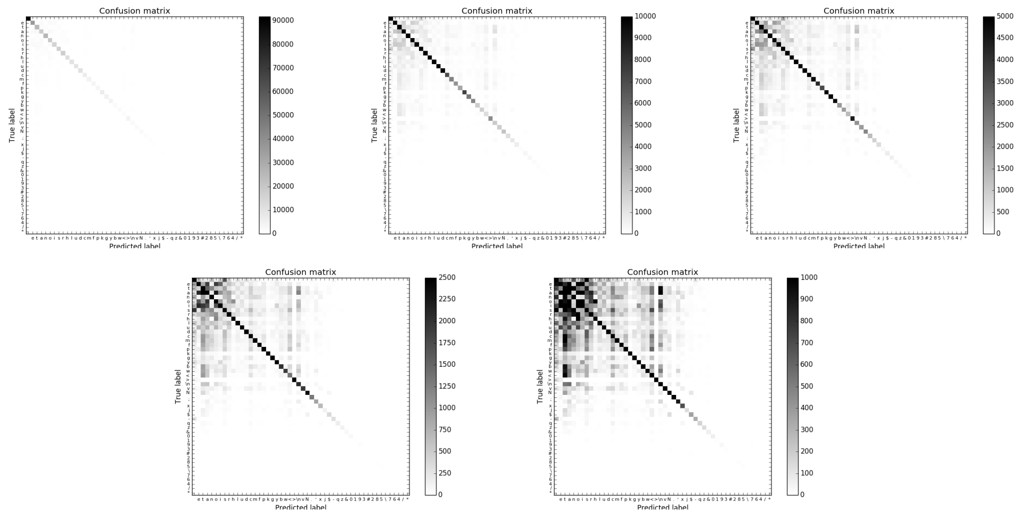

Figure 5: Confusion matrix on the validation set of the best 5.5M parameter model without autoregressive attention on PTB. Each plot shows the same confusion matrix with number of examples in each square cut off to a different amount. Starting from the top left: no cutoff, 10000, 5000, 2500, 1000.

We also plot the confidences over time (square root probabilities at each timestep) of the best model on three examples in the validation set in Figure 6. We take the square root in order to slightly exaggerate the low probabilities in order to better illustrate model uncertainty. We clearly see that the model tends to be uncertain after spaces, and often makes an incorrect prediction on these instances. This may be an indication that the model is failing to fully incorporate context in some cases.

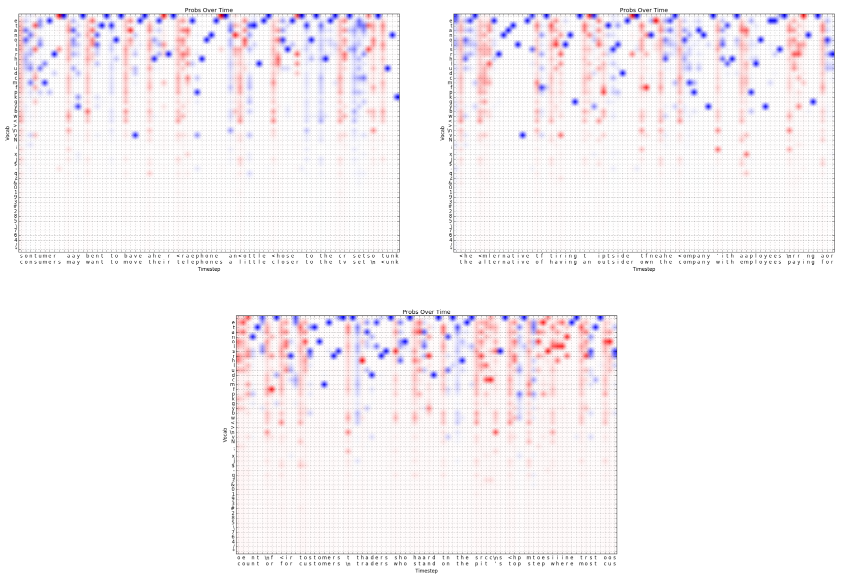

Figure 6: Square root of probabilities over time of the best 5.5M parameter model without autoregressive attention on PTB. Probabilities on three different sequences in the validation set are shown. The points are colored red if the prediction (argmax) in that timestep was incorrect, and blue if the prediction in that timestep was correct.