Language Modeling with Gated Convolutional Networks

Yann N. Dauphin [1]   **Angela Fan** [1]   **Michael Auli** [1]   **David Grangier** [1]

Abstract

The pre-dominant approach to language modeling to date is based on recurrent neural networks. Their success on this task is often linked to their ability to capture unbounded context. In this paper we develop a finite context approach through stacked convolutions, which can be more efficient since they allow parallelization over sequential tokens. We propose a novel simplified gating mechanism that outperforms Oord et al. (2016b) and investigate the impact of key architectural decisions. The proposed approach achieves state-of-the-art on the WikiText-103 benchmark, even though it features long-term dependencies, as well as competitive results on the Google Billion Words benchmark. Our model reduces the latency to score a sentence by an order of magnitude compared to a recurrent baseline. To our knowledge, this is the first time a non-recurrent approach is competitive with strong recurrent models on these large scale language tasks.

1. Introduction

Statistical language models estimate the probability distribution of a sequence of words by modeling the probability of the next word given preceding words, i.e.

$$P(w_0, \ldots, w_N) = P(w_0) \prod_{i=1}^{N} P(w_i | w_0, \ldots, w_{i-1}),$$

where w_i are discrete word indices in a vocabulary. Language models are a critical part of systems for speech recognition (Yu & Deng, 2014) and machine translation (Koehn, 2010).

Recently, neural networks (Bengio et al., 2003; Mikolov et al., 2010; Jozefowicz et al., 2016) have been shown to

---

[1]Facebook AI Research. Correspondence to: Yann N. Dauphin <ynd@fb.com>.

outperform classical *n-gram* language models (Kneser & Ney, 1995; Chen & Goodman, 1996). These classical models suffer from data sparsity, which makes it difficult to represent large contexts and thus, long-range dependencies. Neural language models tackle this issue by embedding words in continuous space over which a neural network is applied. The current state of the art for language modeling is based on long short term memory networks (LSTM; Hochreiter et al., 1997) which can theoretically model arbitrarily long dependencies.

In this paper, we introduce new gated convolutional networks and apply them to language modeling. Convolutional networks can be stacked to represent large context sizes and extract hierarchical features over larger and larger contexts with more abstractive features (LeCun & Bengio, 1995). This allows them to model long-term dependencies by applying $\mathcal{O}(\frac{N}{k})$ operations over a context of size N and kernel width k. In contrast, recurrent networks view the input as a chain structure and therefore require a linear number $\mathcal{O}(N)$ of operations.

Analyzing the input hierarchically bears resemblance to classical grammar formalisms which build syntactic tree structures of increasing granuality, e.g., sentences consist of noun phrases and verb phrases each comprising further internal structure (Manning & Schütze, 1999; Steedman, 2002). Hierarchical structure also eases learning since the number of non-linearities for a given context size is reduced compared to a chain structure, thereby mitigating the vanishing gradient problem (Glorot & Bengio, 2010).

Modern hardware is well suited to models that are highly parallelizable. In recurrent networks, the next output depends on the previous hidden state which does not enable parallelization over the elements of a sequence. Convolutional networks, however, are very amenable to this computing paradigm since the computation of all input words can be performed simultaneously (§2).

Gating has been shown to be essential for recurrent neural networks to reach state-of-the-art performance (Jozefowicz et al., 2016). Our gated linear units reduce the vanishing gradient problem for deep architectures by providing a linear path for the gradients while retaining non-linear capabilities (§5.2).

We show that gated convolutional networks outperform other recently published language models such as LSTMs trained in a similar setting on the Google Billion Word Benchmark (Chelba et al., 2013). We also evaluate the ability of our models to deal with long-range dependencies on the WikiText-103 benchmark for which the model is conditioned on an entire paragraph rather than a single sentence and we achieve a new state-of-the-art on this dataset (Merity et al., 2016). Finally, we show that gated linear units achieve higher accuracy and converge faster than the LSTM-style gating of Oord et al. (2016; §4, §5).

2. Approach

In this paper we introduce a new neural language model that replaces the recurrent connections typically used in recurrent networks with gated temporal convolutions. Neural language models (Bengio et al., 2003) produce a representation $\mathbf{H} = [\mathbf{h}_0, \ldots, \mathbf{h}_N]$ of the context for each word $w_0, \ldots, w_N$ to predict the next word $P(w_i|\mathbf{h}_i)$. Recurrent neural networks f compute $\mathbf{H}$ through a recurrent function $\mathbf{h}_i = f(\mathbf{h}_{i-1}, w_{i-1})$ which is an inherently sequential process that cannot be parallelized over i.[1]

Our proposed approach convolves the inputs with a function f to obtain $\mathbf{H} = f * w$ and therefore has no temporal dependencies, so it is easier to parallelize over the individual words of a sentence. This process will compute each context as a function of a number of preceding words. Compared to recurrent networks, the context size is finite but we will demonstrate both that infinite contexts are not necessary and our models can represent large enough contexts to perform well in practice (§5).

Figure 1 illustrates the model architecture. Words are represented by a vector embedding stored in a lookup table $\mathbf{D}^{|\mathcal{V}| \times e}$ where $|\mathcal{V}|$ is the number of words in the vocabulary and e is the embedding size. The input to our model is a sequence of words $w_0, \ldots, w_N$ which are represented by word embeddings $\mathbf{E} = [\mathbf{D}_{w_0}, \ldots, \mathbf{D}_{w_N}]$. We compute the hidden layers $h_0, \ldots, h_L$ as

$$h_l(\mathbf{X}) = (\mathbf{X} * \mathbf{W} + \mathbf{b}) \otimes \sigma(\mathbf{X} * \mathbf{V} + \mathbf{c}) \qquad (1)$$

where m, n are respectively the number of input and output feature maps and k is the patch size, $\mathbf{X} \in \mathbb{R}^{N \times m}$ is the input of layer h_l (either word embeddings or the outputs of previous layers), $\mathbf{W} \in \mathbb{R}^{k \times m \times n}$, $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{V} \in \mathbb{R}^{k \times m \times n}$, $\mathbf{c} \in \mathbb{R}^n$ are learned parameters, σ is the sigmoid function and $\otimes$ is the element-wise product between matrices.

When convolving inputs, we take care that $\mathbf{h}_i$ does not contain information from future words. We address this by shifting the convolutional inputs to prevent the kernels

---

[1]Parallelization is usually done over multiple sequences instead.

```
input:[batch,seq,feas]=[batch,seq,m]
kernel:[height,width,out_channels]=[k,m,n]
```
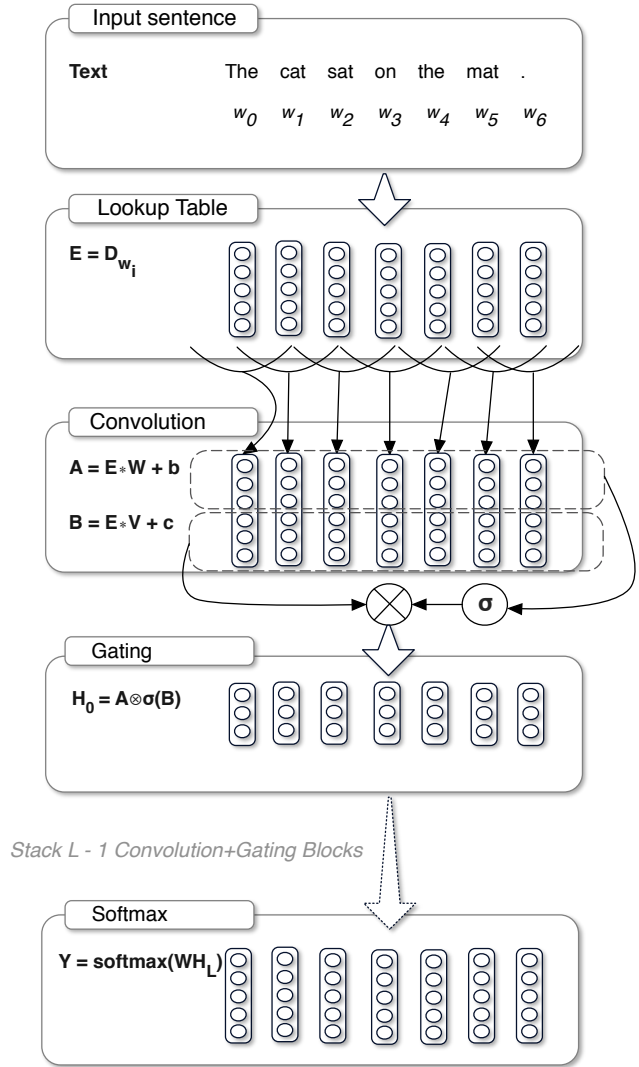

Figure 1. Architecture of the gated convolutional network for language modeling.

from seeing future context (Oord et al., 2016a). Specifically, we zero-pad the beginning of the sequence with $k-1$ elements, assuming the first input element is the beginning of sequence marker which we do not predict and k is the width of the kernel.

The output of each layer is a linear projection $\mathbf{X} * \mathbf{W} + \mathbf{b}$ modulated by the gates $\sigma(\mathbf{X} * \mathbf{V} + \mathbf{c})$. Similar to LSTMs, these gates multiply each element of the matrix $\mathbf{X} * \mathbf{W} + \mathbf{b}$ and control the information passed on in the hierarchy. We dub this gating mechanism Gated Linear Units (GLU). Stacking multiple layers on top of the input $\mathbf{E}$ gives a representation of the context for each word $\mathbf{H} = h_L \circ \ldots \circ h_0(\mathbf{E})$. We wrap the convolution and the gated linear unit in a pre-activation residual block that adds the input of the block to
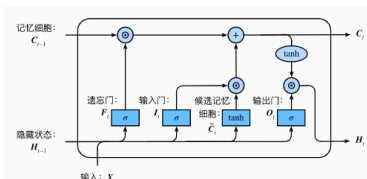
the output (He et al., 2015a). The blocks have a bottleneck structure for computational efficiency and each block has up to 5 layers.

The simplest choice to obtain model predictions is to use a softmax layer, but this choice is often computationally inefficient for large vocabularies and approximations such as noise contrastive estimation (Gutmann & Hyvärinen) or hierarchical softmax (Morin & Bengio, 2005) are preferred. We choose an improvement of the latter known as *adaptive softmax* which assigns higher capacity to very frequent words and lower capacity to rare words (Grave et al., 2016a). This results in lower memory requirements as well as faster computation at both training and test time.

3. Gating Mechanisms

Gating mechanisms control the path through which information flows in the network and have proven to be useful for recurrent neural networks (Hochreiter & Schmidhuber, 1997). LSTMs enable long-term memory via a separate cell controlled by *input* and *forget* gates. This allows information to flow unimpeded through potentially many timesteps. Without these gates, information could easily vanish through the transformations of each timestep. In contrast, convolutional networks do not suffer from the same kind of vanishing gradient and we find experimentally that they do not require forget gates.

Therefore, we consider models possessing solely output gates, which allow the network to control what information should be propagated through the hierarchy of layers. We show this mechanism to be useful for language modeling as it allows the model to select which words or features are relevant for predicting the next word. Parallel to our work, Oord et al. (2016b) have shown the effectiveness of an LSTM-style mechanism of the form $\tanh(\mathbf{X} * \mathbf{W} + \mathbf{b}) \otimes \sigma(\mathbf{X} * \mathbf{V} + \mathbf{c})$ for the convolutional modeling of images. Later, Kalchbrenner et al. (2016) extended this mechanism with additional gates for use in translation and character-level language modeling.

Gated linear units are a simplified gating mechanism based on the work of Dauphin & Grangier (2015) for non-deterministic gates that reduce the vanishing gradient problem by having linear units coupled to the gates. This retains the non-linear capabilities of the layer while allowing the gradient to propagate through the linear unit without scaling. The gradient of the LSTM-style gating of which we dub gated tanh unit (GTU) is

$$\nabla[\tanh(\mathbf{X}) \otimes \sigma(\mathbf{X})] = \tanh'(\mathbf{X})\nabla\mathbf{X} \otimes \sigma(\mathbf{X})$$
$$+ \sigma'(\mathbf{X})\nabla\mathbf{X} \otimes \tanh(\mathbf{X}). \quad (2)$$

Notice that it gradually vanishes as we stack layers because of the downscaling factors $\tanh'(\mathbf{X})$ and $\sigma'(\mathbf{X})$. In con-

trast, the gradient of the gated linear unit

$$\nabla[\mathbf{X} \otimes \sigma(\mathbf{X})] = \nabla\mathbf{X} \otimes \sigma(\mathbf{X}) + \mathbf{X} \otimes \sigma'(\mathbf{X})\nabla\mathbf{X} \quad (3)$$

has a path $\nabla\mathbf{X} \otimes \sigma(\mathbf{X})$ without downscaling for the activated gating units in $\sigma(\mathbf{X})$. This can be thought of as a multiplicative skip connection which helps gradients flow through the layers. We compare the different gating schemes experimentally in Section §5.2 and we find gated linear units allow for faster convergence to better perplexities.

4. Experimental Setup

4.1. Datasets

We report results on two public large-scale language modeling datasets. First, the Google Billion Word dataset (Chelba et al., 2013) is considered one of the largest language modeling datasets with almost one billion tokens and a vocabulary of over 800K words. In this dataset, words appearing less than 3 times are replaced with a special unknown symbol. The data is based on an English corpus of $30,301,028$ sentences whose order has been shuffled. Second, WikiText-103 is a smaller dataset of over 100M tokens with a vocabulary of about 200K words (Merity et al., 2016). Different from GBW, the sentences are consecutive which allows models to condition on larger contexts rather than single sentences. For both datasets, we add a beginning of sequence marker $<$S$>$ at the start of each line and an end of sequence marker $<$/S$>$ at the end of each line. On the Google Billion Word corpus each sequence is a single sentence, while on WikiText-103 a sequence is an entire paragraph. The model sees $<$S$>$ and $<$/S$>$ as input but only predicts the end of sequence marker $<$/S$>$. We evaluate models by computing the perplexity $e^{\frac{1}{N}\sum_i^N -\log p(w_i|...,w_{i-1})}$ on the standard held out test portion of each dataset.

4.2. Training

We implement our models in Torch (Collobert et al., 2011) and train on Tesla M40 GPUs. The majority of our models are trained on single GPU, as we focused on identifying compact architectures with good generalization and efficient computation at test time. We trained larger models with an 8-GPU setup by copying the model onto each GPU and dividing the batch such that each worker computes 1/8th of the gradients. The gradients are then summed using Nvidia NCCL. The multi-GPU setup allowed us to train models with larger hidden units.

We train using Nesterov's momentum (Sutskever et al., 2013). While the cost in terms of memory is storing another vector of the size of the parameters, it increases the speed of convergence significantly with minimal additional

Name	GCNN-13	GCNN-14B	GCNN-9	GCNN-8B	GCNN-8	GCNN-14
Dataset	Google Billion Word				wikitext-103	
Lookup	128				280	
Conv1	$[4, 1268] \times 1$	$[5, 512] \times 1$	$[4, 807] \times 1$	$[1, 512] \times 1$	$[4, 900] \times 1$	$[6, 850] \times 3$
Conv2.x	$\begin{bmatrix} 4, 1268 \\ 4, 1268 \end{bmatrix} \times 12$	$\begin{bmatrix} 1, 128 \\ 5, 128 \\ 1, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 4, 807 \\ 4, 807 \end{bmatrix} \times 4$	$\begin{bmatrix} 1, 128 \\ 5, 128 \\ 1, 512 \end{bmatrix} \times 3$	$[4, 900] \times 7$	$[1, 850] \times 1$
Conv3.x		$\begin{bmatrix} 1, 512 \\ 5, 512 \\ 1, 1024 \end{bmatrix} \times 3$		$\begin{bmatrix} 1, 256 \\ 5, 256 \\ 1, 512 \end{bmatrix} \times 3$		$[5, 850] \times 4$
Conv4.x		$\begin{bmatrix} 1, 1024 \\ 5, 1024 \\ 1, 2048 \end{bmatrix} \times 6$		$\begin{bmatrix} 1, 1024 \\ 1, 1024 \\ 1, 2048 \end{bmatrix} \times 1$		$[1, 850] \times 1$
Conv5.x		$\begin{bmatrix} 1, 1024 \\ 5, 1024 \\ 1, 4096 \end{bmatrix} \times 1$				$[4, 850] \times 3$
Conv6.x						$[4, 1024] \times 1$
Conv7.x						$[4, 2048] \times 1$
AdaSoftmax	10k,40k,200k		4k,40k,200k		2k,10k,50k	10k,20k,200k

Table 1. Architectures for the models. The residual building blocks are shown in brackets with the format $[k, n]$. "B" denotes bottleneck architectures.

computation compared to standard stochastic gradient descent. The speed of convergence was further increased with gradient clipping (Pascanu et al., 2013) and weight normalization (Salimans & Kingma, 2016).

Pascanu et al. (2013) argue for gradient clipping because it prevents the gradient explosion problem that characterizes RNNs. However, gradient clipping is not tied to RNNs, as it can be derived from the general concept of trust region methods. Gradient clipping is found using a spherical trust region

$$\Delta\theta^* = \operatorname*{argmin}_{\text{s. t. } \|\Delta\theta\| \leq \epsilon} f(\theta) + \nabla f^T \Delta\theta$$
$$= -\max(\|\nabla f\|, \epsilon) \frac{\nabla f}{\|\nabla f\|}. \tag{4}$$

Empirically, our experiments converge significantly faster with the use of gradient clipping even though we do not use a recurrent architecture.

In combination, these methods led to stable and fast convergence with comparatively large learning rates such as 1.

4.3. Hyper-parameters

We found good hyper-parameter configurations by cross-validating with random search on a validation set. For model architecture, we select the number of residual blocks between $\{1, \ldots, 10\}$, the size of the embeddings with $\{128, \ldots, 256\}$, the number of units between $\{128, \ldots, 2048\}$, and the kernel width between $\{3, \ldots, 5\}$.

In general, finding a good architecture was simple and the rule of thumb is that the larger the model, the better the performance. In terms of optimization, we initialize the layers of the model with the Kaiming initialization (He et al., 2015b), with the learning rate sampled uniformly in the interval $[1., 2.]$, the momentum set to 0.99, and clipping set to 0.1. Good hyper-parameters for the optimizer are quite straightforward to find and the optimal values do not change much between datasets.

5. Results

LSTMs and recurrent networks are able to capture long term dependencies and are fast becoming cornerstones in natural language processing. In this section, we compare strong LSTM and RNN models from the literature to our gated convolutional approach on two datasets.

We find the GCNN outperforms the comparable LSTM results on Google billion words. To accurately compare these approaches, we control for the same number of GPUs and the adaptive softmax output model (Grave et al., 2016a), as these variables have a significant influence on performance. In this setting, the GCNN reaches 38.1 test perplexity while the comparable LSTM has 39.8 perplexity (Table 2).

Further, the GCNN obtains strong performance with much greater computational efficiency. Figure 2 shows that our approach closes the previously significant gap between models that use the full softmax and models with the usually less accurate hierarchical softmax. Thanks to the adap-

Model	Test PPL	Hardware
Sigmoid-RNN-2048 (Ji et al., 2015)	68.3	1 CPU
Interpolated KN 5-Gram (Chelba et al., 2013)	67.6	100 CPUs
Sparse Non-Negative Matrix LM (Shazeer et al., 2014)	52.9	-
RNN-1024 + MaxEnt 9 Gram Features (Chelba et al., 2013)	51.3	24 GPUs
LSTM-2048-512 (Jozefowicz et al., 2016)	43.7	32 GPUs
2-layer LSTM-8192-1024 (Jozefowicz et al., 2016)	30.6	32 GPUs
BIG GLSTM-G4 (Kuchaiev & Ginsburg, 2017)	23.3*	8 GPUs
LSTM-2048 (Grave et al., 2016a)	43.9	1 GPU
2-layer LSTM-2048 (Grave et al., 2016a)	39.8	1 GPU
GCNN-13	38.1	1 GPU
GCNN-14 Bottleneck	31.9	8 GPUs

Table 2. Results on the Google Billion Word test set. The GCNN outperforms the LSTMs with the same output approximation.
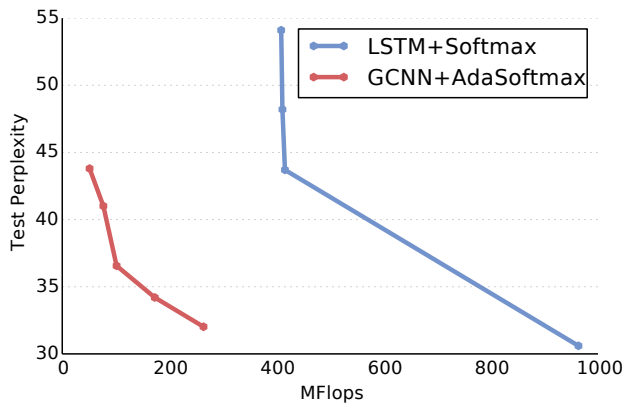

Figure 2. In comparison to the state-of-the-art (Jozefowicz et al., 2016) which uses the full softmax, the adaptive softmax approximation greatly reduces the number of operations required to reach a given perplexity.

Model	Test PPL	Hardware
LSTM-1024 (Grave et al., 2016b)	48.7	1 GPU
GCNN-8	44.9	1 GPU
GCNN-14	37.2	4 GPUs

Table 3. Results for single models on the WikiText-103 dataset.

lion Word, the average sentence length is quite short — only 20 words. We evaluate on WikiText-103 to determine if the model can perform well on a dataset where much larger contexts are available. On WikiText-103, an input sequence is an entire Wikipedia article instead of an individual sentence - increasing the average length to 4000 words. However, the GCNN outperforms LSTMs on this problem as well (Table 3). The GCNN-8 model has 8 layers with 800 units each and the LSTM has 1024 units. These results show that GCNNs can model enough context to achieve strong results.

We evaluated on the Gigaword dataset following Chen et al. (2016) to compare with fully connected models. We found that the fully connected and convolutional network reach respectively 55.6 and 29.4 perplexity. We also ran preliminary experiments on the much smaller Penn tree bank dataset. When we score the sentences independently, the GCNN and LSTM have comparable test perplexity with 108.7 and 109.3 respectively. However, it is possible to achieve better results by conditioning on previous sentences. Unlike the LSTM, we found that the GCNN overfits on this quite small dataset and so we note the model is better suited to larger scale problems.

5.1. Computational Efficiency

Computational cost is an important consideration for language models. Depending on the application, there are a number of metrics to consider. We measure the *throughput*
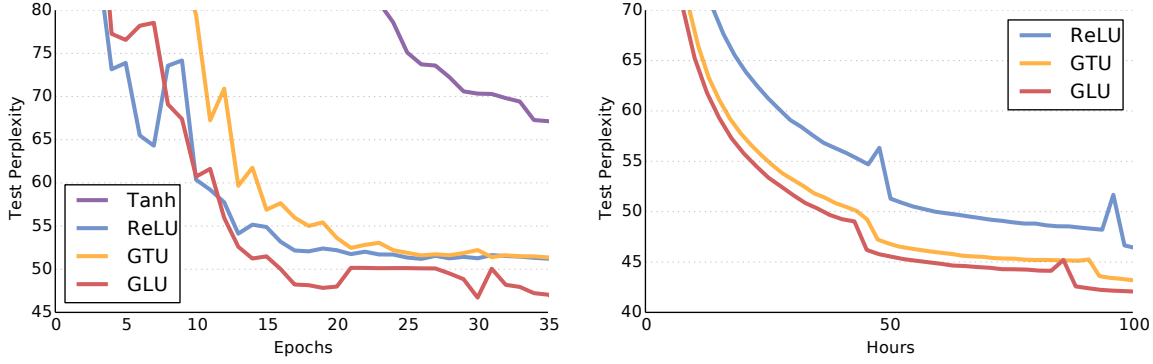
tive softmax, the GCNN only requires a fraction of the operations to reach the same perplexity values. The GCNN outperforms other single model state-of-the-art approaches except the much larger LSTM of Jozefowicz et al. (2016), a model which requires more GPUs and the much more computationally expensive full softmax. In comparison, the largest model we have trained reaches 31.9 test perplexity compared to the 30.6 of that approach, but only requires training for 2 weeks on 8 GPUs compared to 3 weeks of training on 32 GPUs for the LSTM. Note that these results can be improved by either using mixtures of experts (Shazeer et al., 2017) or ensembles of these models.

Another relevant concern is if the GCNN's fixed context size can thoroughly model long sequences. On Google Bil-

---

*appeared after submission

Figure 3. Learning curves on WikiText-103 (left) and Google Billion Word (right) for models with different activation mechanisms. Models with gated linear units (GLU) converge faster and to a lower perplexity.

	Throughput		Responsiveness
	(CPU)	(GPU)	(GPU)
LSTM-2048	169	45,622	2,282
GCNN-9	121	29,116	29,116
GCNN-8 Bottleneck	**179**	**45,878**	**45,878**

Table 4. Processing speed in tokens/s at test time for an LSTM with 2048 units and GCNNs achieving 43.9 perplexity on Google Billion Word. The GCNN with bottlenecks improves the responsiveness by 20 times while maintaining high throughput.

of a model as the number of tokens that can be processed per second. Throughput can be maximized by processing many sentences in parallel to amortize sequential operations. In contrast, *responsiveness* is the speed of processing the input sequentially, one token at a time. Throughput is important because it indicates the time required to process a corpus of text and responsiveness is an indicator of the time to finish processing a sentence. A model can have low responsiveness but high throughput by evaluating many sentences simultaneously through batching. In this case, such a model is slow in finishing processing individual sentences, but can process many sentences at a good rate.

We evaluate the throughput and responsiveness for models that reach approximately 43.9 perplexity on the Google Billion Word benchmark. We consider the LSTM with 2048 units in Table 2, a GCNN-8Bottleneck with 7 Resnet blocks that have a bottleneck structure as described by (He et al., 2015a) and a GCNN-8 without bottlenecks. A bottleneck block wedges a $k > 1$ convolution between two $k = 1$ layers. This designs reduces computational cost by reducing and increasing dimensionality with the $k = 1$ layers so that the convolution operates in a lower dimensional space. Our results show that the use of bottleneck blocks is important to maintaining computational efficiency.

The throughput of the LSTM is measured by using a large batch of 750 sequences of length 20, resulting in 15,000 tokens per batch. The responsiveness is the average speed to process a sequence of 15,000 contiguous tokens. Table 4 shows that the throughput for the LSTM and the GCNN are similar. The LSTM performs very well on GPU because the large batch size of 750 enables high parallelization over different sentences. This is because the LSTM implementation has been thoroughly optimized and uses cuDNN, whereas the cuDNN implementation of convolutions is not been optimized for the 1-D convolutions we use in our model. We believe much better performance can be achieved by a more efficient 1-D cuDNN convolution. Unlike the LSTM, the GCNN can be parallelized both over sequences as well as across the tokens of each sequence, allowing the GCNN to have 20x higher responsiveness.

5.2. Gating Mechanisms

In this section, we compare the gated linear unit with other mechanisms as well as to models without gating. We consider the LSTM-style gating mechanism (GTU) $\tanh(\mathbf{X} * \mathbf{W} + \mathbf{b}) \otimes \sigma(\mathbf{X} * \mathbf{V} + \mathbf{c})$ of (Oord et al., 2016b) and networks that use regular ReLU or Tanh activations. Gating units add parameters, so for fair comparison, we carefully cross-validate models with a comparable number of parameters. Figure 3 (left) shows that GLU networks converge to a lower perplexity than the other approaches on WikiText-103. Similar to gated linear units, the ReLU has a linear path that lets the gradients easily pass through the active units. This translates to much faster convergence for both the ReLU and the GLU. On the other hand, neither Tanh nor GTU have this linear path, and thus suffer from the vanishing gradient problem. In the GTU, both the inputs as well as the gating units can cut the gradient when the units saturate.
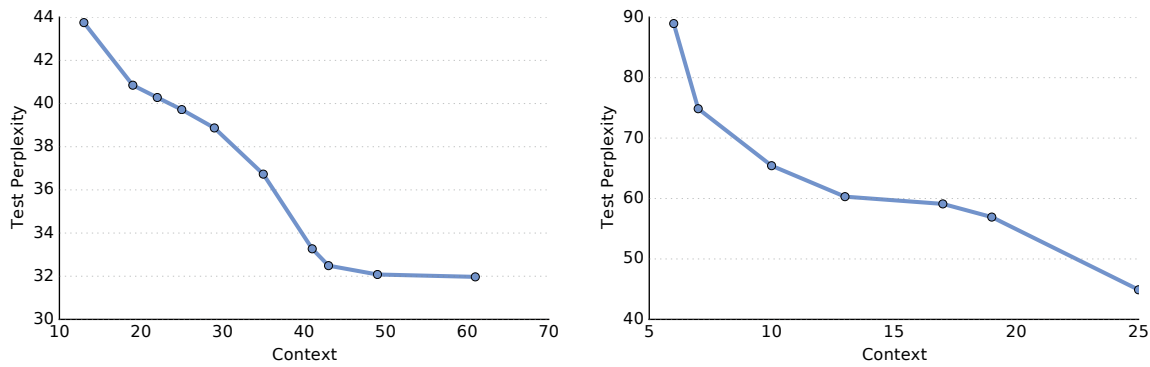
Comparing the GTU and Tanh models allows us to measure

Figure 4. Test perplexity as a function of context for Google Billion Word (left) and Wiki-103 (right). We observe that models with bigger context achieve better results but the results start diminishing quickly after a context of 20.

the effect of gating since the Tanh model can be thought of as a GTU network with the sigmoid gating units removed. The results (Figure 3, left) show that the gating units make a vast difference and provide useful modeling capabilities, as there is a large difference in the performance between GTU and Tanh units. Similarly, while ReLU unit is not an exact ablation of the gating units in the GLU, it can be seen as a simplification $\text{ReLU}(\mathbf{X}) = \mathbf{X} \otimes (\mathbf{X} > 0)$ where the gates become active depending on the sign of the input. Also in this case, GLU units lead to lower perplexity.

In Figure 3 (right) we repeat the same experiment on the larger Google Billion Words dataset. We consider a fixed time budget of 100 hours because of the considerable training time required for this task. Similar to WikiText-103, the gated linear units achieve the best results on this problem. There is a gap of about 5 perplexity points between the GLU and ReLU which is similar to the difference between the LSTM and RNN models measured by (Jozefowicz et al., 2016) on the same dataset.

5.3. Non-linear Modeling

The experiments so far have shown that the gated linear unit benefits from the linear path the unit provides compared to other non-linearities. Next, we compare networks with GLUs to purely linear networks and networks with bilinear layers in order to measure the impact of the non-linear path provided by the gates of the GLU. One motivation for this experiment is the success of linear models on many natural language processing tasks (Manning & Schütze, 1999). We consider deep *linear* convolutional networks where the layers lack the gating units of the GLU and take the form $h_l(\mathbf{X}) = \mathbf{X} * \mathbf{W} + \mathbf{b}$. Stacking several layers on top of each other is simply a factorization of the model which remains linear up to the softmax, at which point it becomes log-linear. Another variation of GLUs are bilinear layers (Mnih & Hinton, 2007) which take the form

$$h_l(\mathbf{X}) = (\mathbf{X} * \mathbf{W} + \mathbf{b}) \otimes (\mathbf{X} * \mathbf{V} + \mathbf{c}).$$
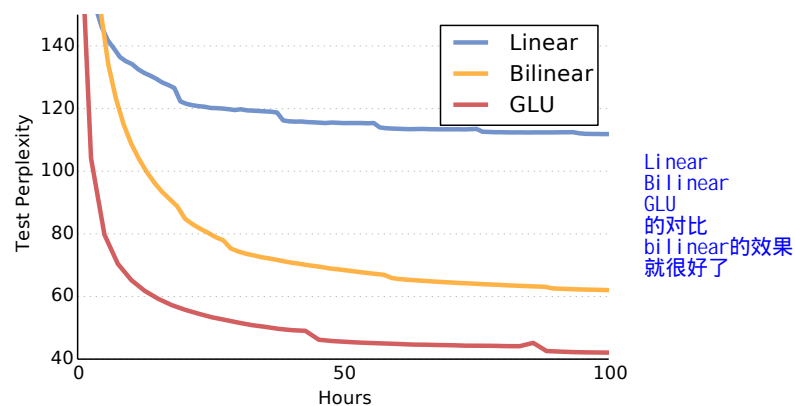

Figure 5. Learning curves on Google Billion Word for models with varying degrees of non-linearity.

Figure 5 shows that GLUs perform best, followed by bilinear layers and then linear layers. Bilinear layers improve over linear ones by more than 40 perplexity points, and the GLU improves another 20 perplexity points over the bilinear model. The linear model performs very poorly at perplexity 115 even compared to 67.6 of a Kneser-Ney 5-gram model, even though the former has access to more context. Surprisingly, the introduction of the bilinear units is enough to reach 61 perplexity on Google Billion Word, which surpasses both Kneser-Ney 5-gram models and the non-linear neural model of (Ji et al., 2015).

5.4. Context Size

Figure 4 shows the impact of context size for the gated CNN. We tried different combinations of network depth and kernel widths for each context size and chose the best performing one for each size. Generally, larger contexts

improve accuracy but returns drastically diminish with windows larger than 40 words, even for WikiText-103 where we may condition on an entire Wikipedia article. This means that the unlimited context offered by recurrent models is not strictly necessary for language modeling. Furthermore, this finding is also congruent with the fact that good performance with recurrent networks can be obtained by truncating gradients after only 40 timesteps using truncated back propagation through time. Figure 4 also shows that WikiText-103 benefits much more from larger context size than Google Billion Word as the performance degrades more sharply with smaller contexts. WikiText-103 provides much more context than Google Billion Word where the average sentence size is 20. However, while the average size of the documents is close to 4000 tokens, we find that strong performance can be achieved with a context size as low as 30 tokens.

5.5. Training

In this section, we perform an ablation study of the impact of weight normalization and gradient clipping. We separately cross-validate the hyper-parameters of each configuration to make the comparison fair. Due to the high cost of each of these experiments, we only consider a single iteration over the training data. Figure 6 shows that both methods significantly speed up convergence. Weight normalization in particular improves the speed by over two times. This speedup is partly due to the ability to use much larger learning rates (1 instead of 0.01) than would otherwise be possible. Both clipping and weight normalization add computational overhead, but it is minor compared to the large gains in convergence speed.
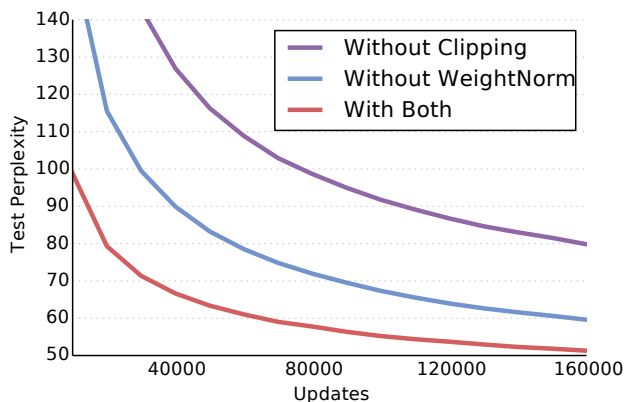

Figure 6. Effect of weight normalization and gradient clipping on Google Billion Word.

6. Conclusion

We introduce a convolutional neural network for language modeling with a novel gating mechanism. Compared to recurrent neural networks, our approach builds a hierarchical representation of the input words that makes it easier to capture long-range dependencies, similar in spirit to the tree-structured analysis of linguistic grammar formalisms. The same property eases learning since features are passed through a fixed number of layers and non-linearities, unlike for recurrent networks where the number of processing steps differs depending on the position of the word in the input. The results show that our gated convolutional network achieves a new state of the art on WikiText-103. On the Google Billion Word benchmark, we show competitive results can be achieved with significantly fewer resources.

Acknowledgments

References

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

Chelba, Ciprian, Mikolov, Tomas, Schuster, Mike, Ge, Qi, Brants, Thorsten, Koehn, Phillipp, and Robinson, Tony. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*, 2013.

Chen, Stanley F and Goodman, Joshua. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pp. 310–318. Association for Computational Linguistics, 1996.

Chen, Wenlin, Grangier, David, and Auli, Michael. Strategies for training large vocabulary neural language models. *CoRR*, abs/1512.04906, 2016.

Collobert, Ronan, Kavukcuoglu, Koray, and Farabet, Clement. Torch7: A Matlab-like Environment for Machine Learning. In *BigLearn, NIPS Workshop*, 2011. URL http://torch.ch.

Dauphin, Yann N and Grangier, David. Predicting distributions with linearizing belief networks. *arXiv preprint arXiv:1511.05622*, 2015.

Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. *The handbook of brain theory and neural networks*, 2010.

Grave, E., Joulin, A., Cissé, M., Grangier, D., and Jégou, H. Efficient softmax approximation for GPUs. *ArXiv e-prints*, September 2016a.

Grave, E., Joulin, A., and Usunier, N. Improving Neural Language Models with a Continuous Cache. *ArXiv e-prints*, December 2016b.

Gutmann, Michael and Hyvärinen, Aapo. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015a.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015b.

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

Ji, Shihao, Vishwanathan, SVN, Satish, Nadathur, Anderson, Michael J, and Dubey, Pradeep. Blackout: Speeding up recurrent neural network language models with very large vocabularies. *arXiv preprint arXiv:1511.06909*, 2015.

Jozefowicz, Rafal, Vinyals, Oriol, Schuster, Mike, Shazeer, Noam, and Wu, Yonghui. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Kalchbrenner, Nal, Espeholt, Lasse, Simonyan, Karen, van den Oord, Aaron, Graves, Alex, and Kavukcuoglu, Koray. Neural Machine Translation in Linear Time. *arXiv*, 2016.

Kneser, Reinhard and Ney, Hermann. Improved backing-off for m-gram language modeling. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pp. 181–184. IEEE, 1995.

Koehn, Philipp. *Statistical Machine Translation*. Cambridge University Press, New York, NY, USA, 1st edition, 2010. ISBN 0521874157, 9780521874151.

Kuchaiev, Oleksii and Ginsburg, Boris. Factorization tricks for LSTM networks. *CoRR*, abs/1703.10722, 2017. URL http://arxiv.org/abs/1703.10722.

LeCun, Yann and Bengio, Yoshua. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Manning, Christopher D and Schütze, Hinrich. Foundations of statistical natural language processing, 1999.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer Sentinel Mixture Models. *ArXiv e-prints*, September 2016.

Mikolov, Tomáš, Martin, Karafiát, Burget, Lukáš, Cernocký, Jan, and Khudanpur, Sanjeev. Recurrent Neural Network based Language Model. In *Proc. of INTERSPEECH*, pp. 1045–1048, 2010.

Mnih, Andriy and Hinton, Geoffrey. Three new graphical models for statistical language modelling. In *Proceedings of the 24th international conference on Machine learning*, pp. 641–648. ACM, 2007.

Morin, Frederic and Bengio, Yoshua. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pp. 246–252. Citeseer, 2005.

Oord, Aaron van den, Kalchbrenner, Nal, and Kavukcuoglu, Koray. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016a.

Oord, Aaron van den, Kalchbrenner, Nal, Vinyals, Oriol, Espeholt, Lasse, Graves, Alex, and Kavukcuoglu, Koray. Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*, 2016b.

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. In *Proceedings of The 30th International Conference on Machine Learning*, pp. 1310–1318, 2013.

Salimans, Tim and Kingma, Diederik P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

Shazeer, Noam, Pelemans, Joris, and Chelba, Ciprian. Skip-gram language modeling using sparse non-negative matrix probability estimation. *arXiv preprint arXiv:1412.1454*, 2014.

Shazeer, Noam, Mirhoseini, Azalia, Maziarz, Krzysztof, Davis, Andy, Le, Quoc V., Hinton, Geoffrey E., and Dean, Jeff. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR*, abs/1701.06538, 2017. URL http://arxiv.org/abs/1701.06538.

Steedman, Mark. The syntactic process. 2002.

Sutskever, Ilya, Martens, James, Dahl, George E, and Hinton, Geoffrey E. On the importance of initialization and momentum in deep learning. 2013.

Yu, Dong and Deng, Li. *Automatic Speech Recognition: A Deep Learning Approach*. Springer Publishing Company, Incorporated, 2014. ISBN 1447157788, 9781447157786.