An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

Shaojie Bai [1] **J. Zico Kolter** [2] **Vladlen Koltun** [3]

Abstract

For most deep learning practitioners, sequence modeling is synonymous with recurrent networks. Yet recent results indicate that convolutional architectures can outperform recurrent networks on tasks such as audio synthesis and machine translation. Given a new sequence modeling task or dataset, which architecture should one use? We conduct a systematic evaluation of generic convolutional and recurrent architectures for sequence modeling. The models are evaluated across a broad range of standard tasks that are commonly used to benchmark recurrent networks. Our results indicate that a simple convolutional architecture outperforms canonical recurrent networks such as LSTMs across a diverse range of tasks and datasets, while demonstrating longer effective memory. We conclude that the common association between sequence modeling and recurrent networks should be reconsidered, and convolutional networks should be regarded as a natural starting point for sequence modeling tasks. To assist related work, we have made code available at http://github.com/locuslab/TCN.

1. Introduction

Deep learning practitioners commonly regard recurrent architectures as the default starting point for sequence modeling tasks. The sequence modeling chapter in the canonical textbook on deep learning is titled "Sequence Modeling: Recurrent and Recursive Nets" (Goodfellow et al., 2016), capturing the common association of sequence modeling and recurrent architectures. A well-regarded recent online course on "Sequence Models" focuses exclusively on recurrent architectures (Ng, 2018).

[1]Machine Learning Department, Carnegie Mellon University, Pittsburgh, PA, USA [2]Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA [3]Intel Labs, Santa Clara, CA, USA. Correspondence to: Shaojie Bai <shaojieb@cs.cmu.edu>, J. Zico Kolter <zkolter@cs.cmu.edu>, Vladlen Koltun <vkoltun@gmail.edu>.

On the other hand, recent research indicates that certain convolutional architectures can reach state-of-the-art accuracy in audio synthesis, word-level language modeling, and machine translation (van den Oord et al., 2016; Kalchbrenner et al., 2016; Dauphin et al., 2017; Gehring et al., 2017a;b). This raises the question of whether these successes of convolutional sequence modeling are confined to specific application domains or whether a broader reconsideration of the association between sequence processing and recurrent networks is in order.

We address this question by conducting a systematic empirical evaluation of convolutional and recurrent architectures on a broad range of sequence modeling tasks. We specifically target a comprehensive set of tasks that have been repeatedly used to compare the effectiveness of different recurrent network architectures. These tasks include polyphonic music modeling, word- and character-level language modeling, as well as synthetic stress tests that had been deliberately designed and frequently used to benchmark RNNs. Our evaluation is thus set up to compare convolutional and recurrent approaches to sequence modeling on the recurrent networks' "home turf".

To represent convolutional networks, we describe a generic temporal convolutional network (TCN) architecture that is applied across all tasks. This architecture is informed by recent research, but is deliberately kept simple, combining some of the best practices of modern convolutional architectures. It is compared to canonical recurrent architectures such as LSTMs and GRUs.

The results suggest that TCNs convincingly outperform baseline recurrent architectures across a broad range of sequence modeling tasks. This is particularly notable because the tasks include diverse benchmarks that have commonly been used to evaluate recurrent network designs (Chung et al., 2014; Pascanu et al., 2014; Jozefowicz et al., 2015; Zhang et al., 2016). This indicates that the recent successes of convolutional architectures in applications such as audio processing are not confined to these domains.

To further understand these results, we analyze more deeply the memory retention characteristics of recurrent networks. We show that despite the theoretical ability of recurrent architectures to capture infinitely long history, TCNs exhibit

substantially longer memory, and are thus more suitable for domains where a long history is required.

To our knowledge, the presented study is the most extensive systematic comparison of convolutional and recurrent architectures on sequence modeling tasks. The results suggest that the common association between sequence modeling and recurrent networks should be reconsidered. The TCN architecture appears not only more accurate than canonical recurrent networks such as LSTMs and GRUs, but also simpler and clearer. It may therefore be a more appropriate starting point in the application of deep networks to sequences.

2. Background

Convolutional networks (LeCun et al., 1989) have been applied to sequences for decades (Sejnowski & Rosenberg, 1987; Hinton, 1989). They were used prominently for speech recognition in the 80s and 90s (Waibel et al., 1989; Bottou et al., 1990). ConvNets were subsequently applied to NLP tasks such as part-of-speech tagging and semantic role labelling (Collobert & Weston, 2008; Collobert et al., 2011; dos Santos & Zadrozny, 2014). More recently, convolutional networks were applied to sentence classification (Kalchbrenner et al., 2014; Kim, 2014) and document classification (Zhang et al., 2015; Conneau et al., 2017; Johnson & Zhang, 2015; 2017). Particularly inspiring for our work are the recent applications of convolutional architectures to machine translation (Kalchbrenner et al., 2016; Gehring et al., 2017a;b), audio synthesis (van den Oord et al., 2016), and language modeling (Dauphin et al., 2017).

Recurrent networks are dedicated sequence models that maintain a vector of hidden activations that are propagated through time (Elman, 1990; Werbos, 1990; Graves, 2012). This family of architectures has gained tremendous popularity due to prominent applications to language modeling (Sutskever et al., 2011; Graves, 2013; Hermans & Schrauwen, 2013) and machine translation (Sutskever et al., 2014; Bahdanau et al., 2015). The intuitive appeal of recurrent modeling is that the hidden state can act as a representation of everything that has been seen so far in the sequence. Basic RNN architectures are notoriously difficult to train (Bengio et al., 1994; Pascanu et al., 2013) and more elaborate architectures are commonly used instead, such as the LSTM (Hochreiter & Schmidhuber, 1997) and the GRU (Cho et al., 2014). Many other architectural innovations and training techniques for recurrent networks have been introduced and continue to be actively explored (El Hihi & Bengio, 1995; Schuster & Paliwal, 1997; Gers et al., 2002; Koutnik et al., 2014; Le et al., 2015; Ba et al., 2016; Wu et al., 2016; Krueger et al., 2017; Merity et al., 2017; Campos et al., 2018).

Multiple empirical studies have been conducted to evaluate the effectiveness of different recurrent architectures. These studies have been motivated in part by the many degrees of freedom in the design of such architectures. Chung et al. (2014) compared different types of recurrent units (LSTM vs. GRU) on the task of polyphonic music modeling. Pascanu et al. (2014) explored different ways to construct deep RNNs and evaluated the performance of different architectures on polyphonic music modeling, character-level language modeling, and word-level language modeling. Jozefowicz et al. (2015) searched through more than ten thousand different RNN architectures and evaluated their performance on various tasks. They concluded that if there were "architectures much better than the LSTM", then they were "not trivial to find". Greff et al. (2017) benchmarked the performance of eight LSTM variants on speech recognition, handwriting recognition, and polyphonic music modeling. They also found that "none of the variants can improve upon the standard LSTM architecture significantly". Zhang et al. (2016) systematically analyzed the connecting architectures of RNNs and evaluated different architectures on character-level language modeling and on synthetic stress tests. Melis et al. (2018) benchmarked LSTM-based architectures on word-level and character-level language modeling, and concluded that "LSTMs outperform the more recent models".

Other recent works have aimed to combine aspects of RNN and CNN architectures. This includes the Convolutional LSTM (Shi et al., 2015), which replaces the fully-connected layers in an LSTM with convolutional layers to allow for additional structure in the recurrent layers; the Quasi-RNN model (Bradbury et al., 2017) that interleaves convolutional layers with simple recurrent layers; and the dilated RNN (Chang et al., 2017), which adds dilations to recurrent architectures. While these combinations show promise in combining the desirable aspects of both types of architectures, our study here focuses on a comparison of generic convolutional and recurrent architectures.

While there have been multiple thorough evaluations of RNN architectures on representative sequence modeling tasks, we are not aware of a similarly thorough comparison of convolutional and recurrent approaches to sequence modeling. (Yin et al. (2017) have reported a comparison of convolutional and recurrent networks for sentence-level and document-level classification tasks. In contrast, sequence modeling calls for architectures that can synthesize whole sequences, element by element.) Such comparison is particularly intriguing in light of the aforementioned recent success of convolutional architectures in this domain. Our work aims to compare generic convolutional and recurrent architectures on typical sequence modeling tasks that are commonly used to benchmark RNN variants themselves (Hermans & Schrauwen, 2013; Le et al., 2015; Jozefowicz et al., 2015; Zhang et al., 2016).

3. Temporal Convolutional Networks

We begin by describing a generic architecture for convolutional sequence prediction. Our aim is to distill the best practices in convolutional network design into a simple architecture that can serve as a convenient but powerful starting point. We refer to the presented architecture as a temporal convolutional network (TCN), emphasizing that we adopt this term not as a label for a truly new architecture, but as a simple descriptive term for a family of architectures. (Note that the term has been used before (Lea et al., 2017).) The distinguishing characteristics of TCNs are: 1) the convolutions in the architecture are causal, meaning that there is no information "leakage" from future to past; 2) the architecture can take a sequence of any length and map it to an output sequence of the same length, just as with an RNN. Beyond this, we emphasize how to build very long effective history sizes (i.e., the ability for the networks to look very far into the past to make a prediction) using a combination of very deep networks (augmented with residual layers) and dilated convolutions.

Our architecture is informed by recent convolutional architectures for sequential data (van den Oord et al., 2016; Kalchbrenner et al., 2016; Dauphin et al., 2017; Gehring et al., 2017a;b), but is distinct from all of them and was designed from first principles to combine simplicity, autoregressive prediction, and very long memory. For example, the TCN is much simpler than WaveNet (van den Oord et al., 2016) (no skip connections across layers, conditioning, context stacking, or gated activations).

Compared to the language modeling architecture of Dauphin et al. (2017), TCNs do not use gating mechanisms and have much longer memory.

3.1. Sequence Modeling

Before defining the network structure, we highlight the nature of the sequence modeling task. Suppose that we are given an input sequence $x_0, \ldots, x_T$, and wish to predict some corresponding outputs $y_0, \ldots, y_T$ at each time. The key constraint is that to predict the output y_t for some time t, we are constrained to only use those inputs that have been previously observed: $x_0, \ldots, x_t$. Formally, a sequence modeling network is any function $f : \mathcal{X}^{T+1} \to \mathcal{Y}^{T+1}$ that produces the mapping

$$\hat{y}_0, \ldots, \hat{y}_T = f(x_0, \ldots, x_T) \tag{1}$$

if it satisfies the causal constraint that y_t depends only on $x_0, \ldots, x_t$ and not on any "future" inputs $x_{t+1}, \ldots, x_T$. The goal of learning in the sequence modeling setting is to find a network f that minimizes some expected loss between the actual outputs and the predictions, $L(y_0, \ldots, y_T, f(x_0, \ldots, x_T))$, where the sequences and outputs are drawn according to some distribution.

This formalism encompasses many settings such as autoregressive prediction (where we try to predict some signal given its past) by setting the target output to be simply the input shifted by one time step. It does not, however, directly capture domains such as machine translation, or sequence-to-sequence prediction in general, since in these cases the entire input sequence (including "future" states) can be used to predict each output (though the techniques can naturally be extended to work in such settings).

3.2. Causal Convolutions

As mentioned above, the TCN is based upon two principles: the fact that the network produces an output of the same length as the input, and the fact that there can be no leakage from the future into the past. To accomplish the first point, the TCN uses a 1D fully-convolutional network (FCN) architecture (Long et al., 2015), where each hidden layer is the same length as the input layer, and zero padding of length $(\text{kernel size} - 1)$ is added to keep subsequent layers the same length as previous ones. To achieve the second point, the TCN uses *causal convolutions*, convolutions where an output at time t is convolved only with elements from time t and earlier in the previous layer.

To put it simply: TCN = 1D FCN + causal convolutions.

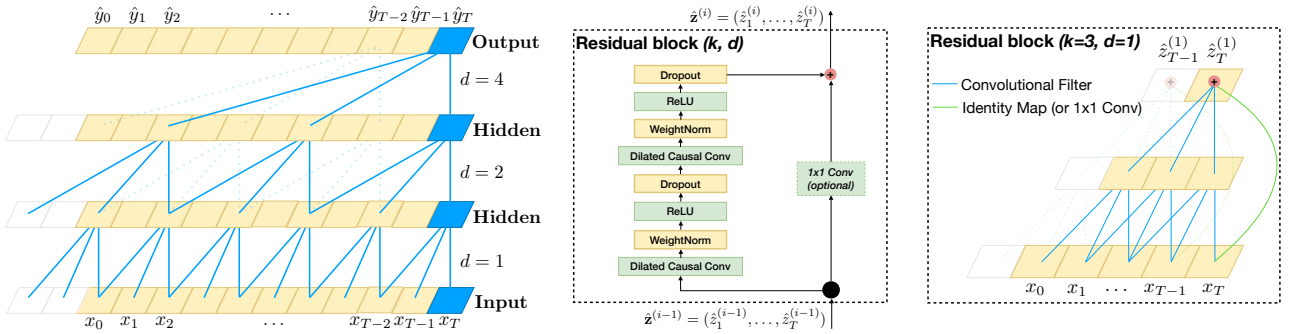
Note that this is essentially the same architecture as the time delay neural network proposed nearly 30 years ago by Waibel et al. (1989), with the sole tweak of zero padding to ensure equal sizes of all layers.

A major disadvantage of this basic design is that in order to achieve a long effective history size, we need an extremely deep network or very large filters, neither of which were particularly feasible when the methods were first introduced. Thus, in the following sections, we describe how techniques from modern convolutional architectures can be integrated into a TCN to allow for both very deep networks and very long effective history.

3.3. Dilated Convolutions

A simple causal convolution is only able to look back at a history with size linear in the depth of the network. This makes it challenging to apply the aforementioned causal convolution on sequence tasks, especially those requiring longer history. Our solution here, following the work of van den Oord et al. (2016), is to employ dilated convolutions that enable an exponentially large receptive field (Yu & Koltun, 2016). More formally, for a 1-D sequence input $\mathbf{x} \in \mathbb{R}^n$ and a filter $f : \{0, \ldots, k-1\} \to \mathbb{R}$, the dilated convolution operation F on element s of the sequence is defined as

$$F(s) = (\mathbf{x} *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot \mathbf{x}_{s-d \cdot i} \tag{2}$$



(a) (b) (c)

Figure 1. Architectural elements in a TCN. (a) A dilated causal convolution with dilation factors $d = 1, 2, 4$ and filter size $k = 3$. The receptive field is able to cover all values from the input sequence. (b) TCN residual block. An 1x1 convolution is added when residual input and output have different dimensions. (c) An example of residual connection in a TCN. The blue lines are filters in the residual function, and the green lines are identity mappings.

where d is the dilation factor, k is the filter size, and $s - d \cdot i$ accounts for the direction of the past. Dilation is thus equivalent to introducing a fixed step between every two adjacent filter taps. When $d = 1$, a dilated convolution reduces to a regular convolution. Using larger dilation enables an output at the top level to represent a wider range of inputs, thus effectively expanding the receptive field of a ConvNet.

This gives us two ways to increase the receptive field of the TCN: choosing larger filter sizes k and increasing the dilation factor d, where the effective history of one such layer is $(k-1)d$. As is common when using dilated convolutions, we increase d exponentially with the depth of the network (i.e., $d = O(2^i)$ at level i of the network). This ensures that there is some filter that hits each input within the effective history, while also allowing for an extremely large effective history using deep networks. We provide an illustration in Figure 1(a).

3.4. Residual Connections

A residual block (He et al., 2016) contains a branch leading out to a series of transformations $\mathcal{F}$, whose outputs are added to the input $\mathbf{x}$ of the block:

$$o = \text{Activation}(\mathbf{x} + \mathcal{F}(\mathbf{x})) \tag{3}$$

This effectively allows layers to learn modifications to the identity mapping rather than the entire transformation, which has repeatedly been shown to benefit very deep networks.

Since a TCN's receptive field depends on the network depth n as well as filter size k and dilation factor d, stabilization of deeper and larger TCNs becomes important. For example, in a case where the prediction could depend on a history of size 2^{12} and a high-dimensional input sequence, a network of up to 12 layers could be needed. Each layer, more specifically, consists of multiple filters for feature extraction. In our design of the generic TCN model, we therefore employ a generic residual module in place of a convolutional layer.

The residual block for our baseline TCN is shown in Figure 1(b). Within a residual block, the TCN has two layers of dilated causal convolution and non-linearity, for which we used the rectified linear unit (ReLU) (Nair & Hinton, 2010). For normalization, we applied weight normalization (Salimans & Kingma, 2016) to the convolutional filters. In addition, a spatial dropout (Srivastava et al., 2014) was added after each dilated convolution for regularization: at each training step, a whole channel is zeroed out.

However, whereas in standard ResNet the input is added directly to the output of the residual function, in TCN (and ConvNets in general) the input and output could have different widths. To account for discrepant input-output widths, we use an additional 1x1 convolution to ensure that element-wise addition $\oplus$ receives tensors of the same shape (see Figure 1(b,c)).

3.5. Discussion

We conclude this section by listing several advantages and disadvantages of using TCNs for sequence modeling.

- **Parallelism**. Unlike in RNNs where the predictions for later timesteps must wait for their predecessors to complete, convolutions can be done in parallel since the same filter is used in each layer. Therefore, in both training and evaluation, a long input sequence can be processed as a whole in TCN, instead of sequentially as in RNN.
- **Flexible receptive field size**. A TCN can change its receptive field size in multiple ways. For instance, stacking more dilated (causal) convolutional layers, using larger dilation factors, or increasing the filter size are all viable options (with possibly different interpretations). TCNs thus afford better control of the model's memory size, and are easy to adapt to different domains.
- **Stable gradients**. Unlike recurrent architectures, TCN has a backpropagation path different from the temporal direction of the sequence. TCN thus avoids the problem

of exploding/vanishing gradients, which is a major issue for RNNs (and which led to the development of LSTM, GRU, HF-RNN (Martens & Sutskever, 2011), etc.).

- **Low memory requirement for training**. Especially in the case of a long input sequence, LSTMs and GRUs can easily use up a lot of memory to store the partial results for their multiple cell gates. However, in a TCN the filters are shared across a layer, with the backpropagation path depending only on network depth. Therefore in practice, we found gated RNNs likely to use up to a multiplicative factor more memory than TCNs.
- **Variable length inputs**. Just like RNNs, which model inputs with variable lengths in a recurrent way, TCNs can also take in inputs of arbitrary lengths by sliding the 1D convolutional kernels. This means that TCNs can be adopted as drop-in replacements for RNNs for sequential data of arbitrary length.

There are also two notable disadvantages to using TCNs.

- **Data storage during evaluation**. In evaluation/testing, RNNs only need to maintain a hidden state and take in a current input x_t in order to generate a prediction. In other words, a "summary" of the entire history is provided by the fixed-length set of vectors h_t, and the actual observed sequence can be discarded. In contrast, TCNs need to take in the raw sequence up to the effective history length, thus possibly requiring more memory during evaluation.
- **Potential parameter change for a transfer of domain**. Different domains can have different requirements on the amount of history the model needs in order to predict. Therefore, when transferring a model from a domain where only little memory is needed (i.e., small k and d) to a domain where much longer memory is required (i.e., much larger k and d), TCN may perform poorly for not having a sufficiently large receptive field.

4. Sequence Modeling Tasks

We evaluate TCNs and RNNs on tasks that have been commonly used to benchmark the performance of different RNN sequence modeling architectures (Hermans & Schrauwen, 2013; Chung et al., 2014; Pascanu et al., 2014; Le et al., 2015; Jozefowicz et al., 2015; Zhang et al., 2016). The intention is to conduct the evaluation on the "home turf" of RNN sequence models. We use a comprehensive set of synthetic stress tests along with real-world datasets from multiple domains.

The adding problem. In this task, each input consists of a length-n sequence of depth 2, with all values randomly chosen in $[0, 1]$, and the second dimension being all zeros except for two elements that are marked by 1. The objective is to sum the two random values whose second dimensions are marked by 1. Simply predicting the sum to be 1 should give an MSE of about 0.1767. First introduced by Hochreiter & Schmidhuber (1997), the adding problem has been used repeatedly as a stress test for sequence models (Martens & Sutskever, 2011; Pascanu et al., 2013; Le et al., 2015; Arjovsky et al., 2016; Zhang et al., 2016).

Sequential MNIST and P-MNIST. Sequential MNIST is frequently used to test a recurrent network's ability to retain information from the distant past (Le et al., 2015; Zhang et al., 2016; Wisdom et al., 2016; Cooijmans et al., 2016; Krueger et al., 2017; Jing et al., 2017). In this task, MNIST images (LeCun et al., 1998) are presented to the model as a 784×1 sequence for digit classification. In the more challenging P-MNIST setting, the order of the sequence is permuted at random (Le et al., 2015; Arjovsky et al., 2016; Wisdom et al., 2016; Krueger et al., 2017).

Copy memory. In this task, each input sequence has length $T + 20$. The first 10 values are chosen randomly among the digits $1, \ldots, 8$, with the rest being all zeros, except for the last 11 entries that are filled with the digit '9' (the first '9' is a delimiter). The goal is to generate an output of the same length that is zero everywhere except the last 10 values after the delimiter, where the model is expected to repeat the 10 values it encountered at the start of the input. This task was used in prior works such as Zhang et al. (2016); Arjovsky et al. (2016); Wisdom et al. (2016); Jing et al. (2017).

JSB Chorales and Nottingham. JSB Chorales (Allan & Williams, 2005) is a polyphonic music dataset consisting of the entire corpus of 382 four-part harmonized chorales by J. S. Bach. Each input is a sequence of elements. Each element is an 88-bit binary code that corresponds to the 88 keys on a piano, with 1 indicating a key that is pressed at a given time. Nottingham is a polyphonic music dataset based on a collection of 1,200 British and American folk tunes, and is much larger than JSB Chorales. JSB Chorales and Nottingham have been used in numerous empirical investigations of recurrent sequence modeling (Chung et al., 2014; Pascanu et al., 2014; Jozefowicz et al., 2015; Greff et al., 2017). The performance on both tasks is measured in terms of negative log-likelihood (NLL).

PennTreebank. We used the PennTreebank (PTB) (Marcus et al., 1993) for both character-level and word-level language modeling. When used as a character-level language corpus, PTB contains 5,059K characters for training, 396K for validation, and 446K for testing, with an alphabet size of 50. When used as a word-level language corpus, PTB contains 888K words for training, 70K for validation, and 79K for testing, with a vocabulary size of 10K. This is a highly studied but relatively small language modeling dataset (Miyamoto & Cho, 2016; Krueger et al., 2017; Merity et al., 2017).

Wikitext-103. Wikitext-103 (Merity et al., 2016) is almost

Table 1. Evaluation of TCNs and recurrent architectures on synthetic stress tests, polyphonic music modeling, character-level language modeling, and word-level language modeling. The generic TCN architecture outperforms canonical recurrent networks across a comprehensive suite of tasks and datasets. Current state-of-the-art results are listed in the supplement. ^{h} means that higher is better. $^\ell$ means that lower is better.

Sequence Modeling Task	Model Size ($\approx$)	Models			
		LSTM	GRU	RNN	**TCN**
Seq. MNIST (accuracy^h)	70K	87.2	96.2	21.5	**99.0**
Permuted MNIST (accuracy)	70K	85.7	87.3	25.3	**97.2**
Adding problem T=600 (loss$^\ell$)	70K	0.164	**5.3e-5**	0.177	**5.8e-5**
Copy memory T=1000 (loss)	16K	0.0204	0.0197	0.0202	**3.5e-5**
Music JSB Chorales (loss)	300K	8.45	8.43	8.91	**8.10**
Music Nottingham (loss)	1M	3.29	3.46	4.05	**3.07**
Word-level PTB (perplexity$^\ell$)	13M	**78.93**	92.48	114.50	88.68
Word-level Wiki-103 (perplexity)	-	48.4	-	-	**45.19**
Word-level LAMBADA (perplexity)	-	4186	-	14725	**1279**
Char-level PTB (bpc$^\ell$)	3M	1.36	1.37	1.48	**1.31**
Char-level text8 (bpc)	5M	1.50	1.53	1.69	**1.45**

110 times as large as PTB, featuring a vocabulary size of about 268K. The dataset contains 28K Wikipedia articles (about 103 million words) for training, 60 articles (about 218K words) for validation, and 60 articles (246K words) for testing. This is a more representative and realistic dataset than PTB, with a much larger vocabulary that includes many rare words, and has been used in Merity et al. (2016); Grave et al. (2017); Dauphin et al. (2017).

LAMBADA. Introduced by Paperno et al. (2016), LAMBADA is a dataset comprising 10K passages extracted from novels, with an average of 4.6 sentences as context, and 1 target sentence the last word of which is to be predicted. This dataset was built so that a person can easily guess the missing word when given the context sentences, but not when given only the target sentence without the context sentences. Most of the existing models fail on LAMBADA (Paperno et al., 2016; Grave et al., 2017). In general, better results on LAMBADA indicate that a model is better at capturing information from longer and broader context. The training data for LAMBADA is the full text of 2,662 novels with more than 200M words. The vocabulary size is about 93K.

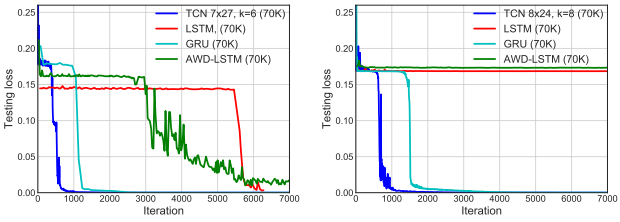
text8. We also used the text8 dataset for character-level language modeling (Mikolov et al., 2012). text8 is about 20 times larger than PTB, with about 100M characters from Wikipedia (90M for training, 5M for validation, and 5M for testing). The corpus contains 27 unique alphabets.

5. Experiments

We compare the generic TCN architecture described in Section 3 to canonical recurrent architectures, namely LSTM, GRU, and vanilla RNN, with standard regularizations. All experiments reported in this section used exactly the same TCN architecture, just varying the depth of the network n and occasionally the kernel size k so that the receptive field covers enough context for predictions. We use an exponential dilation $d = 2^i$ for layer i in the network, and the Adam optimizer (Kingma & Ba, 2015) with learning rate 0.002 for TCN, unless otherwise noted. We also empirically find that gradient clipping helped convergence, and we pick the maximum norm for clipping from $[0.3, 1]$. When training recurrent models, we use grid search to find a good set of hyperparameters (in particular, optimizer, recurrent drop $p \in [0.05, 0.5]$, learning rate, gradient clipping, and initial forget-gate bias), while keeping the network around the same size as TCN. No other architectural elaborations, such as gating mechanisms or skip connections, were added to either TCNs or RNNs. Additional details and controlled experiments are provided in the supplementary material.

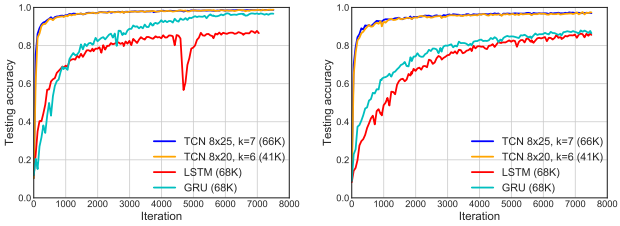
5.1. Synopsis of Results

A synopsis of the results is shown in Table 1. Note that on several of these tasks, the generic, canonical recurrent architectures we study (e.g., LSTM, GRU) are not the state-of-the-art. (See the supplement for more details.) With this caveat, the results strongly suggest that the generic TCN architecture *with minimal tuning* outperforms canonical recurrent architectures across a broad variety of sequence modeling tasks that are commonly used to benchmark the performance of recurrent architectures themselves. We now analyze these results in more detail.



(a) $T = 200$ (b) $T = 600$

Figure 2. Results on the adding problem for different sequence lengths T. TCNs outperform recurrent architectures.



(a) Sequential MNIST (b) P-MNIST

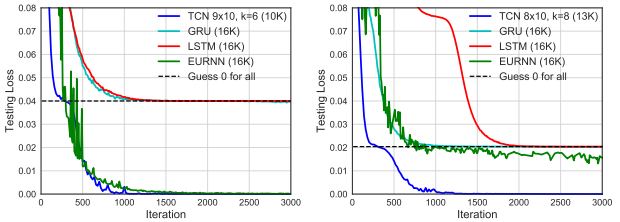
Figure 3. Results on Sequential MNIST and P-MNIST. TCNs outperform recurrent architectures.

5.2. Synthetic Stress Tests

The adding problem. Convergence results for the adding problem, for problem sizes $T = 200$ and 600, are shown in Figure 2. All models were chosen to have roughly 70K parameters. TCNs quickly converged to a virtually perfect solution (i.e., MSE near 0). GRUs also performed quite well, albeit slower to converge than TCNs. LSTMs and vanilla RNNs performed significantly worse.

Sequential MNIST and P-MNIST. Convergence results on sequential and permuted MNIST, run over 10 epochs, are shown in Figure 3. All models were configured to have roughly 70K parameters. For both problems, TCNs substantially outperform the recurrent architectures, both in terms of convergence and in final accuracy on the task. For P-MNIST, TCNs outperform state-of-the-art results (95.9%) based on recurrent networks with Zoneout and Recurrent BatchNorm (Cooijmans et al., 2016; Krueger et al., 2017).

Copy memory. Convergence results on the copy memory task are shown in Figure 4. TCNs quickly converge to correct answers, while LSTMs and GRUs simply converge to the same loss as predicting all zeros. In this case we also compare to the recently-proposed EURNN (Jing et al., 2017), which was highlighted to perform well on this task. While both TCN and EURNN perform well for sequence length $T = 500$, the TCN has a clear advantage for $T = 1000$ and longer (in terms of both loss and rate of convergence).



(a) $T = 500$ (b) $T = 1000$

Figure 4. Result on the copy memory task for different sequence lengths T. TCNs outperform recurrent architectures.

5.3. Polyphonic Music and Language Modeling

We now discuss the results on polyphonic music modeling, character-level language modeling, and word-level language modeling. These domains are dominated by recurrent architectures, with many specialized designs developed for these tasks (Zhang et al., 2016; Ha et al., 2017; Krueger et al., 2017; Grave et al., 2017; Greff et al., 2017; Merity et al., 2017). We mention some of these specialized architectures when useful, but our primary goal is to compare the generic TCN model to similarly generic recurrent architectures, before domain-specific tuning. The results are summarized in Table 1.

Polyphonic music. On Nottingham and JSB Chorales, the TCN with virtually no tuning outperforms the recurrent models by a considerable margin, and even outperforms some enhanced recurrent architectures for this task such as HF-RNN (Boulanger-Lewandowski et al., 2012) and Diagonal RNN (Subakan & Smaragdis, 2017). Note however that other models such as the Deep Belief Net LSTM perform better still (Vohra et al., 2015); we believe this is likely due to the fact that the datasets are relatively small, and thus the right regularization method or generative modeling procedure can improve performance significantly. This is largely orthogonal to the RNN/TCN distinction, as a similar variant of TCN may well be possible.

Word-level language modeling. Language modeling remains one of the primary applications of recurrent networks and many recent works have focused on optimizing LSTMs for this task (Krueger et al., 2017; Merity et al., 2017). Our implementation follows standard practice that ties the weights of encoder and decoder layers for both TCN and RNNs (Press & Wolf, 2016), which significantly reduces the number of parameters in the model. For training, we use SGD and anneal the learning rate by a factor of 0.5 for both TCN and RNNs when validation accuracy plateaus.

On the smaller PTB corpus, an optimized LSTM architecture (with recurrent and embedding dropout, etc.) outperforms the TCN, while the TCN outperforms both GRU and vanilla RNN. However, on the much larger Wikitext-103 corpus and the LAMBADA dataset (Paperno et al., 2016), without any hyperparameter search, the TCN outperforms

the LSTM results of Grave et al. (2017), achieving much lower perplexities.

Character-level language modeling. On character-level language modeling (PTB and text8, accuracy measured in bits per character), the generic TCN outperforms regularized LSTMs and GRUs as well as methods such as Norm-stabilized LSTMs (Krueger & Memisevic, 2015). (Specialized architectures exist that outperform all of these, see the supplement.)

5.4. Memory Size of TCN and RNNs

One of the theoretical advantages of recurrent architectures is their unlimited memory: the theoretical ability to retain information through sequences of unlimited length. We now examine specifically how long the different architectures can retain information in practice. We focus on 1) the copy memory task, which is a stress test designed to evaluate long-term, distant information propagation in recurrent networks, and 2) the LAMBADA task, which tests both local and non-local textual understanding.

The copy memory task is perfectly set up to examine a model's ability to retain information for different lengths of time. The requisite retention time can be controlled by varying the sequence length T. In contrast to Section 5.2, we now focus on the accuracy on the last 10 elements of the output sequence (which are the nontrivial elements that must be recalled). We used models of size 10K for both TCN and RNNs.

The results of this focused study are shown in Figure 5. TCNs consistently converge to 100% accuracy for all sequence lengths, whereas LSTMs and GRUs of the same size quickly degenerate to random guessing as the sequence length T grows. The accuracy of the LSTM falls below 20% for $T < 50$, while the GRU falls below 20% for $T < 200$. These results indicate that TCNs are able to maintain a much longer effective history than their recurrent counterparts.

This observation is backed up on real data by experiments on the large-scale LAMBADA dataset, which is specifically designed to test a model's ability to utilize broad context (Paperno et al., 2016). As shown in Table 1, TCN outperforms LSTMs and vanilla RNNs by a significant margin in perplexity on LAMBADA, with a substantially smaller network and virtually no tuning. (State-of-the-art results on this dataset are even better, but only with the help of additional memory mechanisms (Grave et al., 2017).)

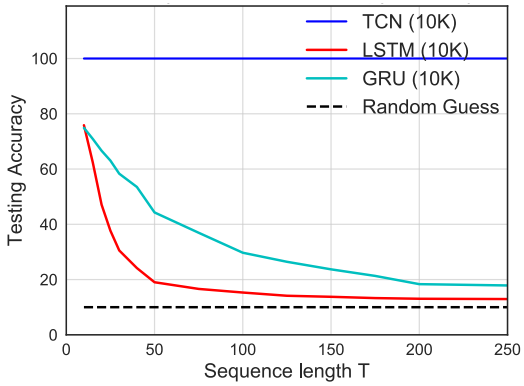


Figure 5. Accuracy on the copy memory task for sequences of different lengths T. While TCN exhibits 100% accuracy for all sequence lengths, the LSTM and GRU degenerate to random guessing as T grows.

6. Conclusion

We have presented an empirical evaluation of generic convolutional and recurrent architectures across a comprehensive suite of sequence modeling tasks. To this end, we have described a simple temporal convolutional network (TCN) that combines best practices such as dilations and residual connections with the causal convolutions needed for autoregressive prediction. The experimental results indicate that TCN models substantially outperform generic recurrent architectures such as LSTMs and GRUs. We further studied long-range information propagation in convolutional and recurrent networks, and showed that the "infinite memory" advantage of RNNs is largely absent in practice. TCNs exhibit longer memory than recurrent architectures with the same capacity.

Numerous advanced schemes for regularizing and optimizing LSTMs have been proposed (Press & Wolf, 2016; Krueger et al., 2017; Merity et al., 2017; Campos et al., 2018). These schemes have significantly advanced the accuracy achieved by LSTM-based architectures on some datasets. The TCN has not yet benefitted from this concerted community-wide investment into architectural and algorithmic elaborations. We see such investment as desirable and expect it to yield advances in TCN performance that are commensurate with the advances seen in recent years in LSTM performance. We will release the code for our project to encourage this exploration.

The preeminence enjoyed by recurrent networks in sequence modeling may be largely a vestige of history. Until recently, before the introduction of architectural elements such as dilated convolutions and residual connections, convolutional architectures were indeed weaker. Our results indicate that with these elements, a simple convolutional architecture is more effective across diverse sequence modeling tasks than recurrent architectures such as LSTMs. Due to the comparable clarity and simplicity of TCNs, we conclude that convolutional networks should be regarded as a natural starting point and a powerful toolkit for sequence modeling.

References

Allan, Moray and Williams, Christopher. Harmonising chorales by probabilistic inference. In *NIPS*, 2005.

Arjovsky, Martin, Shah, Amar, and Bengio, Yoshua. Unitary evolution recurrent neural networks. In *ICML*, 2016.

Ba, Lei Jimmy, Kiros, Ryan, and Hinton, Geoffrey E. Layer normalization. *arXiv:1607.06450*, 2016.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

Bengio, Yoshua, Simard, Patrice, and Frasconi, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2), 1994.

Bottou, Léon, Soulie, F Fogelman, Blanchet, Pascal, and Liénard, Jean-Sylvain. Speaker-independent isolated digit recognition: Multilayer perceptrons vs. dynamic time warping. *Neural Networks*, 3(4), 1990.

Boulanger-Lewandowski, Nicolas, Bengio, Yoshua, and Vincent, Pascal. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv:1206.6392*, 2012.

Bradbury, James, Merity, Stephen, Xiong, Caiming, and Socher, Richard. Quasi-recurrent neural networks. In *ICLR*, 2017.

Campos, Victor, Jou, Brendan, Giró i Nieto, Xavier, Torres, Jordi, and Chang, Shih-Fu. Skip RNN: Learning to skip state updates in recurrent neural networks. In *ICLR*, 2018.

Chang, Shiyu, Zhang, Yang, Han, Wei, Yu, Mo, Guo, Xiaoxiao, Tan, Wei, Cui, Xiaodong, Witbrock, Michael J., Hasegawa-Johnson, Mark A., and Huang, Thomas S. Dilated recurrent neural networks. In *NIPS*, 2017.

Cho, Kyunghyun, Van Merriënboer, Bart, Bahdanau, Dzmitry, and Bengio, Yoshua. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv:1409.1259*, 2014.

Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv:1412.3555*, 2014.

Chung, Junyoung, Ahn, Sungjin, and Bengio, Yoshua. Hierarchical multiscale recurrent neural networks. *arXiv:1609.01704*, 2016.

Collobert, Ronan and Weston, Jason. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, 2008.

Collobert, Ronan, Weston, Jason, Bottou, Léon, Karlen, Michael, Kavukcuoglu, Koray, and Kuksa, Pavel P. Natural language processing (almost) from scratch. *JMLR*, 12, 2011.

Conneau, Alexis, Schwenk, Holger, LeCun, Yann, and Barrault, Loïc. Very deep convolutional networks for text classification. In *European Chapter of the Association for Computational Linguistics (EACL)*, 2017.

Cooijmans, Tim, Ballas, Nicolas, Laurent, César, Gülçehre, Çağlar, and Courville, Aaron. Recurrent batch normalization. In *ICLR*, 2016.

Dauphin, Yann N., Fan, Angela, Auli, Michael, and Grangier, David. Language modeling with gated convolutional networks. In *ICML*, 2017.

dos Santos, Cícero Nogueira and Zadrozny, Bianca. Learning character-level representations for part-of-speech tagging. In *ICML*, 2014.

El Hihi, Salah and Bengio, Yoshua. Hierarchical recurrent neural networks for long-term dependencies. In *NIPS*, 1995.

Elman, Jeffrey L. Finding structure in time. *Cognitive Science*, 14 (2), 1990.

Gehring, Jonas, Auli, Michael, Grangier, David, and Dauphin, Yann. A convolutional encoder model for neural machine translation. In *ACL*, 2017a.

Gehring, Jonas, Auli, Michael, Grangier, David, Yarats, Denis, and Dauphin, Yann N. Convolutional sequence to sequence learning. In *ICML*, 2017b.

Gers, Felix A, Schraudolph, Nicol N, and Schmidhuber, Jürgen. Learning precise timing with lstm recurrent networks. *JMLR*, 3, 2002.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. *Deep Learning*. MIT Press, 2016.

Grave, Edouard, Joulin, Armand, and Usunier, Nicolas. Improving neural language models with a continuous cache. In *ICLR*, 2017.

Graves, Alex. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2012.

Graves, Alex. Generating sequences with recurrent neural networks. *arXiv:1308.0850*, 2013.

Greff, Klaus, Srivastava, Rupesh Kumar, Koutník, Jan, Steunebrink, Bas R., and Schmidhuber, Jürgen. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10), 2017.

Ha, David, Dai, Andrew, and Le, Quoc V. HyperNetworks. In *ICLR*, 2017.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *CVPR*, 2016.

Hermans, Michiel and Schrauwen, Benjamin. Training and analysing deep recurrent neural networks. In *NIPS*, 2013.

Hinton, Geoffrey E. Connectionist learning procedures. *Artificial Intelligence*, 40(1-3), 1989.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural Computation*, 9(8), 1997.

Jing, Li, Shen, Yichen, Dubcek, Tena, Peurifoy, John, Skirlo, Scott, LeCun, Yann, Tegmark, Max, and Soljačić, Marin. Tunable efficient unitary neural networks (EUNN) and their application to RNNs. In *ICML*, 2017.

Johnson, Rie and Zhang, Tong. Effective use of word order for text categorization with convolutional neural networks. In *HLT-NAACL*, 2015.

Johnson, Rie and Zhang, Tong. Deep pyramid convolutional neural networks for text categorization. In *ACL*, 2017.

Jozefowicz, Rafal, Zaremba, Wojciech, and Sutskever, Ilya. An empirical exploration of recurrent network architectures. In *ICML*, 2015.

Kalchbrenner, Nal, Grefenstette, Edward, and Blunsom, Phil. A convolutional neural network for modelling sentences. In *ACL*, 2014.

Kalchbrenner, Nal, Espeholt, Lasse, Simonyan, Karen, van den Oord, Aäron, Graves, Alex, and Kavukcuoglu, Koray. Neural machine translation in linear time. *arXiv:1610.10099*, 2016.

Kim, Yoon. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Koutnik, Jan, Greff, Klaus, Gomez, Faustino, and Schmidhuber, Juergen. A clockwork RNN. In *ICML*, 2014.

Krueger, David and Memisevic, Roland. Regularizing RNNs by stabilizing activations. *arXiv:1511.08400*, 2015.

Krueger, David, Maharaj, Tegan, Kramár, János, Pezeshki, Mohammad, Ballas, Nicolas, Ke, Nan Rosemary, Goyal, Anirudh, Bengio, Yoshua, Larochelle, Hugo, Courville, Aaron C., and Pal, Chris. Zoneout: Regularizing RNNs by randomly preserving hidden activations. In *ICLR*, 2017.

Le, Quoc V, Jaitly, Navdeep, and Hinton, Geoffrey E. A simple way to initialize recurrent networks of rectified linear units. *arXiv:1504.00941*, 2015.

Lea, Colin, Flynn, Michael D., Vidal, René, Reiter, Austin, and Hager, Gregory D. Temporal convolutional networks for action segmentation and detection. In *CVPR*, 2017.

LeCun, Yann, Boser, Bernhard, Denker, John S., Henderson, Donnie, Howard, Richard E., Hubbard, Wayne, and Jackel, Lawrence D. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4), 1989.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998.

Long, Jonathan, Shelhamer, Evan, and Darrell, Trevor. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

Marcus, Mitchell P, Marcinkiewicz, Mary Ann, and Santorini, Beatrice. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2), 1993.

Martens, James and Sutskever, Ilya. Learning recurrent neural networks with Hessian-free optimization. In *ICML*, 2011.

Melis, Gábor, Dyer, Chris, and Blunsom, Phil. On the state of the art of evaluation in neural language models. In *ICLR*, 2018.

Merity, Stephen, Xiong, Caiming, Bradbury, James, and Socher, Richard. Pointer sentinel mixture models. *arXiv:1609.07843*, 2016.

Merity, Stephen, Keskar, Nitish Shirish, and Socher, Richard. Regularizing and optimizing LSTM language models. *arXiv:1708.02182*, 2017.

Mikolov, Tomáš, Sutskever, Ilya, Deoras, Anoop, Le, Hai-Son, Kombrink, Stefan, and Cernocky, Jan. Subword language modeling with neural networks. *Preprint*, 2012.

Miyamoto, Yasumasa and Cho, Kyunghyun. Gated word-character recurrent language model. *arXiv:1606.01700*, 2016.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted Boltzmann machines. In *ICML*, 2010.

Ng, Andrew. Sequence Models (Course 5 of Deep Learning Specialization). *Coursera*, 2018.

Paperno, Denis, Kruszewski, Germán, Lazaridou, Angeliki, Pham, Quan Ngoc, Bernardi, Raffaella, Pezzelle, Sandro, Baroni, Marco, Boleda, Gemma, and Fernández, Raquel. The LAMBADA dataset: Word prediction requiring a broad discourse context. *arXiv:1606.06031*, 2016.

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. In *ICML*, 2013.

Pascanu, Razvan, Gülçehre, Çaglar, Cho, Kyunghyun, and Bengio, Yoshua. How to construct deep recurrent neural networks. In *ICLR*, 2014.

Press, Ofir and Wolf, Lior. Using the output embedding to improve language models. *arXiv:1608.05859*, 2016.

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

Schuster, Mike and Paliwal, Kuldip K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45 (11), 1997.

Sejnowski, Terrence J. and Rosenberg, Charles R. Parallel networks that learn to pronounce English text. *Complex Systems*, 1, 1987.

Shi, Xingjian, Chen, Zhourong, Wang, Hao, Yeung, Dit-Yan, Wong, Wai-Kin, and Woo, Wang-chun. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *NIPS*, 2015.

Srivastava, Nitish, Hinton, Geoffrey E, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*, 15(1), 2014.

Subakan, Y Cem and Smaragdis, Paris. Diagonal RNNs in symbolic music modeling. *arXiv:1704.05420*, 2017.

Sutskever, Ilya, Martens, James, and Hinton, Geoffrey E. Generating text with recurrent neural networks. In *ICML*, 2011.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. In *NIPS*, 2014.

van den Oord, Aäron, Dieleman, Sander, Zen, Heiga, Simonyan, Karen, Vinyals, Oriol, Graves, Alex, Kalchbrenner, Nal, Senior, Andrew W., and Kavukcuoglu, Koray. WaveNet: A generative model for raw audio. *arXiv:1609.03499*, 2016.

Vohra, Raunaq, Goel, Kratarth, and Sahoo, JK. Modeling temporal dependencies in data using a DBN-LSTM. In *Data Science and Advanced Analytics (DSAA)*, 2015.

Waibel, Alex, Hanazawa, Toshiyuki, Hinton, Geoffrey, Shikano, Kiyohiro, and Lang, Kevin J. Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3), 1989.

Werbos, Paul J. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10), 1990.

Wisdom, Scott, Powers, Thomas, Hershey, John, Le Roux, Jonathan, and Atlas, Les. Full-capacity unitary recurrent neural networks. In *NIPS*, 2016.

Wu, Yuhuai, Zhang, Saizheng, Zhang, Ying, Bengio, Yoshua, and Salakhutdinov, Ruslan R. On multiplicative integration with recurrent neural networks. In *NIPS*, 2016.

Yang, Zhilin, Dai, Zihang, Salakhutdinov, Ruslan, and Cohen, William W. Breaking the softmax bottleneck: A high-rank RNN language model. *ICLR*, 2018.

Yin, Wenpeng, Kann, Katharina, Yu, Mo, and Schütze, Hinrich. Comparative study of CNN and RNN for natural language processing. *arXiv:1702.01923*, 2017.

Yu, Fisher and Koltun, Vladlen. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016.

Zhang, Saizheng, Wu, Yuhuai, Che, Tong, Lin, Zhouhan, Memisevic, Roland, Salakhutdinov, Ruslan R, and Bengio, Yoshua. Architectural complexity measures of recurrent neural networks. In *NIPS*, 2016.

Zhang, Xiang, Zhao, Junbo Jake, and LeCun, Yann. Character-level convolutional networks for text classification. In *NIPS*, 2015.

An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

Supplementary Material

A. Hyperparameters Settings

A.1. Hyperparameters for TCN

Table 2 lists the hyperparameters we used when applying the generic TCN model on various tasks and datasets. The most important factor for picking parameters is to make sure that the TCN has a sufficiently large receptive field by choosing k and d that can cover the amount of context needed for the task.

As discussed in Section 5, the number of hidden units was chosen so that the model size is approximately at the same level as the recurrent models with which we are comparing. In Table 2, a gradient clip of N/A means no gradient clipping was applied. In larger tasks (e.g., language modeling), we empirically found that gradient clipping (we randomly picked a threshold from $[0.3, 1]$) helps with regularizing TCN and accelerating convergence.

All weights were initialized from a Gaussian disitribution $\mathcal{N}(0, 0.01)$. In general, we found TCN to be relatively insensitive to hyperparameter changes, as long as the effective history (i.e., receptive field) size is sufficient.

A.2. Hyperparameters for LSTM/GRU

Table 3 reports hyperparameter settings that were used for the LSTM. These values are picked from hyperparameter search for LSTMs that have up to 3 layers, and the optimizers are chosen from {SGD, Adam, RMSprop, Adagrad}. For certain larger datasets, we adopted the settings used in prior work (e.g., Grave et al. (2017) on Wikitext-103). GRU hyperparameters were chosen in a similar fashion, but typically with more hidden units than in LSTM to keep the total network size approximately the same (since a GRU cell is more compact).

B. State-of-the-Art Results

As previously noted, the generic TCN and LSTM/GRU models we used can be outperformed by more specialized architectures on some tasks. State-of-the-art results are summarized in Table 4. The same TCN architecture is used across all tasks. Note that the size of the state-of-the-art model may be different from the size of the TCN.

C. Effect of Filter Size and Residual Block

In this section we briefly study the effects of different components of a TCN layer. Overall, we believe dilation is required for modeling long-term dependencies, and so we mainly focus on two other factors here: the filter size k used by each layer, and the effect of residual blocks.

We perform a series of controlled experiments, with the results of the ablative analysis shown in Figure 6. As before, we kept the model size and depth exactly the same for different models, so that the dilation factor is strictly controlled. The experiments were conducted on three different tasks: copy memory, permuted MNIST (P-MNIST), and Penn Treebank word-level language modeling. These experiments confirm that both factors (filter size and residual connections) contribute to sequence modeling performance.

Filter size k. In both the copy memory and the P-MNIST tasks, we observed faster convergence and better accuracy for larger filter sizes. In particular, looking at Figure 6a, a TCN with filter size ≤ 3 only converges to the same level as random guessing. In contrast, on word-level language modeling, a smaller kernel with filter size of $k = 3$ works best. We believe this is because a smaller kernel (along with fixed dilation) tends to focus more on the local context, which is especially important for PTB language modeling (in fact, the very success of n-gram models suggests that only a relatively short memory is needed for modeling language).

Residual block. In all three scenarios that we compared here, we observed that the residual function stabilized training and brought faster convergence with better final results. Especially in language modeling, we found that residual connections contribute substantially to performance (See Figure 6f).

D. Gating Mechanisms

One component that had been used in prior work on convolutional architectures for language modeling is the gated activation (van den Oord et al., 2016; Dauphin et al., 2017). We have chosen not to use gating in the generic TCN model. We now examine this choice more closely.

Table 2. TCN parameter settings for experiments in Section 5.

TCN SETTINGS							
Dataset/Task	**Subtask**	k	n	**Hidden**	**Dropout**	**Grad Clip**	**Note**
The Adding Problem	$T = 200$	6	7	27	0.0	N/A	
	$T = 400$	7	7	27			
	$T = 600$	8	8	24			
Seq. MNIST	-	7	8	25	0.0	N/A	
		6	8	20			
Permuted MNIST	-	7	8	25	0.0	N/A	
		6	8	20			
Copy Memory Task	$T = 500$	6	9	10	0.05	1.0	RMSprop 5e-4
	$T = 1000$	8	8	10			
	$T = 2000$	8	9	10			
Music JSB Chorales	-	3	2	150	0.5	0.4	
Music Nottingham	-	6	4	150	0.2	0.4	
Word-level LM	PTB	3	4	600	0.5	0.4	Embed. size 600
	Wiki-103	3	5	1000	0.4		Embed. size 400
	LAMBADA	4	5	500			Embed. size 500
Char-level LM	PTB	3	3	450	0.1	0.15	Embed. size 100
	text8	2	5	520			

Dauphin et al. (2017) compared the effects of gated linear units (GLU) and gated tanh units (GTU), and adopted GLU in their non-dilated gated ConvNet. Following the same choice, we now compare TCNs using ReLU and TCNs with gating (GLU), represented by an elementwise product between two convolutional layers, with one of them also passing through a sigmoid function $\sigma(x)$. Note that the gates architecture uses approximately twice as many convolutional layers as the ReLU-TCN.

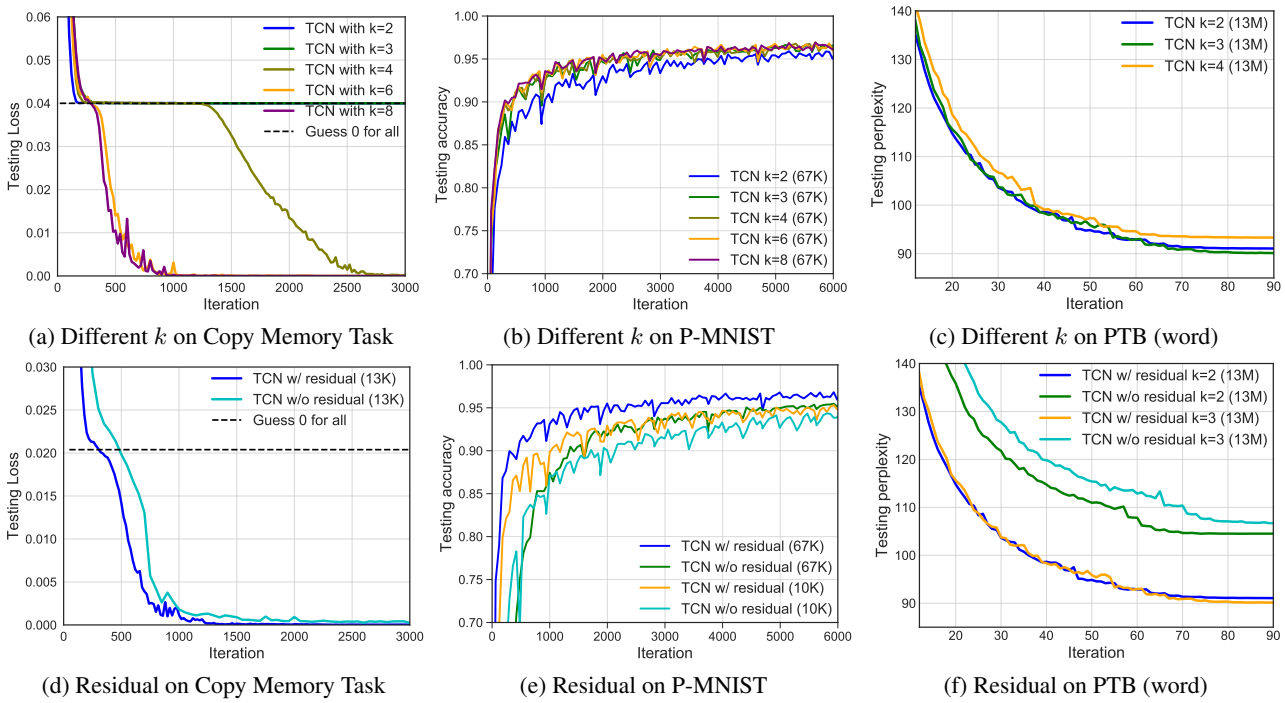
The results are shown in Table 5, where we kept the number of model parameters at about the same size. The GLU does further improve TCN accuracy on certain language modeling datasets like PTB, which agrees with prior work. However, we do not observe comparable benefits on other tasks, such as polyphonic music modeling or synthetic stress tests that require longer information retention. On the copy memory task with $T = 1000$, we found that TCN with gating converged to a worse result than TCN with ReLU (though still better than recurrent models).

Table 3. LSTM parameter settings for experiments in Section 5.

LSTM SETTINGS (KEY PARAMETERS)							
Dataset/Task	**Subtask**	n	**Hidden**	**Dropout**	**Grad Clip**	**Bias**	**Note**
The Adding Problem	$T = 200$	2	77	0.0	50	5.0	SGD 1e-3
	$T = 400$	2	77		50	10.0	Adam 2e-3
	$T = 600$	1	130		5	1.0	-
Seq. MNIST	-	1	130	0.0	1	1.0	RMSprop 1e-3
Permuted MNIST	-	1	130	0.0	1	10.0	RMSprop 1e-3
Copy Memory Task	$T = 500$	1	50	0.05	0.25	-	RMSprop/Adam
	$T = 1000$	1	50		1		
	$T = 2000$	3	28		1		
Music JSB Chorales	-	2	200	0.2	1	10.0	SGD/Adam
Music Nottingham	-	3	280	0.1	0.5	-	Adam 4e-3
		1	500		1	-	
Word-level LM	PTB	3	700	0.4	0.3	1.0	SGD 30, Emb. 700, etc.
	Wiki-103	-	-	-	-	-	Grave et al. (2017)
	LAMBADA	-	-	-	-	-	Grave et al. (2017)
Char-level LM	PTB	2	600	0.1	0.5	-	Emb. size 120
	text8	1	1024	0.15	0.5	-	Adam 1e-2

Table 4. State-of-the-art (SoTA) results for tasks in Section 5.

TCN VS. SOTA RESULTS					
Task	**TCN Result**	**Size**	**SoTA**	**Size**	**Model**
Seq. MNIST (acc.)	99.0	21K	99.0	21K	Dilated GRU (Chang et al., 2017)
P-MNIST (acc.)	97.2	42K	95.9	42K	Zoneout (Krueger et al., 2017)
Adding Prob. 600 (loss)	5.8e-5	70K	5.3e-5	70K	Regularized GRU
Copy Memory 1000 (loss)	3.5e-5	70K	0.011	70K	EURNN (Jing et al., 2017)
JSB Chorales (loss)	8.10	300K	3.47	-	DBN+LSTM (Vohra et al., 2015)
Nottingham (loss)	3.07	1M	1.32	-	DBN+LSTM (Vohra et al., 2015)
Word PTB (ppl)	88.68	13M	47.7	22M	AWD-LSTM-MoS + Dynamic Eval. (Yang et al., 2018)
Word Wiki-103 (ppl)	45.19	148M	40.4	>300M	Neural Cache Model (Large) (Grave et al., 2017)
Word LAMBADA (ppl)	1279	56M	138	>100M	Neural Cache Model (Large) (Grave et al., 2017)
Char PTB (bpc)	1.31	3M	1.22	14M	2-LayerNorm HyperLSTM (Ha et al., 2017)
Char text8 (bpc)	1.45	4.6M	1.29	>12M	HM-LSTM (Chung et al., 2016)



(a) Different k on Copy Memory Task (b) Different k on P-MNIST (c) Different k on PTB (word)

(d) Residual on Copy Memory Task (e) Residual on P-MNIST (f) Residual on PTB (word)

Figure 6. Controlled experiments that study the effect of different components of the TCN model.

Table 5. An evaluation of gating in TCN. A plain TCN is compared to a TCN that uses gated activations.

Task	**TCN**	**TCN + Gating**
Sequential MNIST (acc.)	**99.0**	**99.0**
Permuted MNIST (acc.)	**97.2**	96.9
Adding Problem $T = 600$ (loss)	**5.8e-5**	**5.6e-5**
Copy Memory $T = 1000$ (loss)	**3.5e-5**	0.00508
JSB Chorales (loss)	**8.10**	8.13
Nottingham (loss)	**3.07**	3.12
Word-level PTB (ppl)	88.68	**87.94**
Char-level PTB (bpc)	1.31	**1.306**
Char text8 (bpc)	**1.45**	1.485