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

Czech Technical University in Prague
Faculty of Informatics

Radek Bartyzal

March 5, 2019

Contents

1 Prerequisites

The following text assumes that the reader is familiar with these terms:

- Neural Networks: Chapter 1 of [1]
- Convolutional Neural Networks (CNN): [2]
- Generic Recurrent Neural Networks (RNN): [3]
- Long Short Term Memory Networks (LSTM): [3]

2 Sequence Modeling

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 step. The key constraint, called **causal constraint**, is that to predict the output y_t for some time step t, we are constrained to only use those inputs that have been previously observed: $x_0, \ldots, x_t$. In other words the function producing y_t cannot depend on 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 **auto-regressive 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 Temporal Convolutional Networks (TCN)

TCN describe a family of network architectures. The distinguishing characteristics of TCNs are:

- The convolutions in the architecture are causal, meaning that there is no information "leakage" from future to past. This is achieved by **Causal convolutions**, convolutions where an output at time t is convolved only with elements from time t and earlier in the previous layer.

- 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. This is achieved by using **1D fully-convolutional network (FCN)** architecture where each hidden layer is the same length as the input layer, and zero padding of length (kernel size $-$ 1) is added to keep subsequent layers the same length as previous ones.

Beyond this, TCNs are able 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 connections** for easier convergence.

- **Dilated convolutions** to create exponentially larger receptive field with every added layer.

3.1 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 it on sequence tasks requiring longer history.

Dilated convolution enables an exponentially large receptive field. More formally, for a 1-D sequence input $X \in \mathbb{R}^n$ and a filter $f : \{0, \ldots, k-1\} \to \mathbb{R}$, the dilated convolution operation F on element at index s of the sequence is defined as:

$$F(s) = \sum_{i=0}^{k-1} f(i) X_{s-d\cdot i} \tag{1}$$

where d is the dilation factor, k is the filter size, and $s-d\cdot i$ represents the direction into 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.

There are two ways to increase the receptive field of the TCN

- Choosing larger filter sizes k.
- Increasing the dilation factor d.

The effective history of one such layer is $(k-1)d$.

The d is increased 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. An illustration is in Figure 1a.

3.2 Residual Connections

A residual block contains a branch leading out to a series of transformations F, whose outputs are added to the input x of the block:

$$o = \text{Activation}(x + F(x))$$

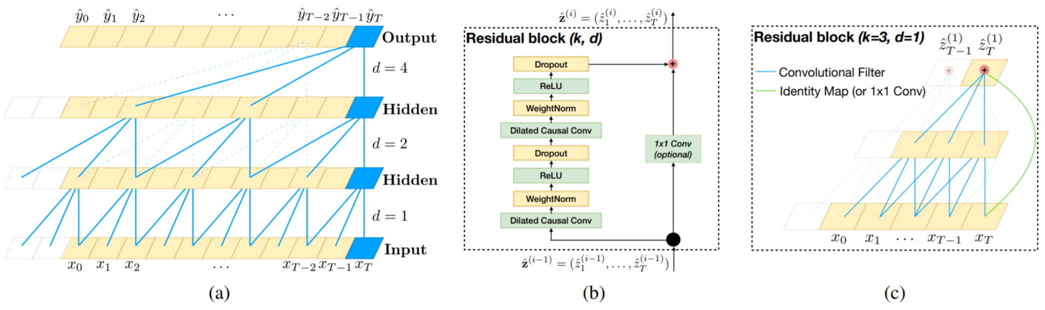


Figure 1: a) Dilated causal convolution with $k = 3$, $d = [2^0, 2^1, 2^2]$. The receptive field is able to cover all values from the input sequence. b) TCN residual block. An 1x1 convolution is added when residual input and output have different dimensions. c) An example of residual connection in a TCN.

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.

A generic residual module shown in Figure 1b is used in place of a standard convolutional layer to help convergence. It uses spatial dropout for regularization: at each training step, a whole channel is zeroed out.

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, an additional 1x1 convolution is used to ensure that the element-wise addition receives tensors of the same shape.

3.3 Advantages and disadvantages

3.3.1 Advantages

- **Parallelism**: Kernels are reused = one layer can be calculated in parallel.
- **Flexible receptive field size**: Changing receptive field by: changing: d, k, number of layers.
- **Stable gradients**: Backpropagation path different from the temporal direction of the sequence = the error propagates only through layers not through time.

- **Low memory requirement for training**: Kernels are reused = less memory than LSTMs that keep partial results for various gates.
- **Variable length inputs**: TCNs can also take in inputs of arbitrary lengths by sliding the 1D convolutional kernels.

3.3.2 Disadvantages

- **Data storage during evaluation**: LSTMs need only one hidden state to predict. TCNs need whole input sequence.
- **Potential parameter change for a transfer of domain**: If transfering to a domain with longer history dependencies, the k and d might need to be changed.

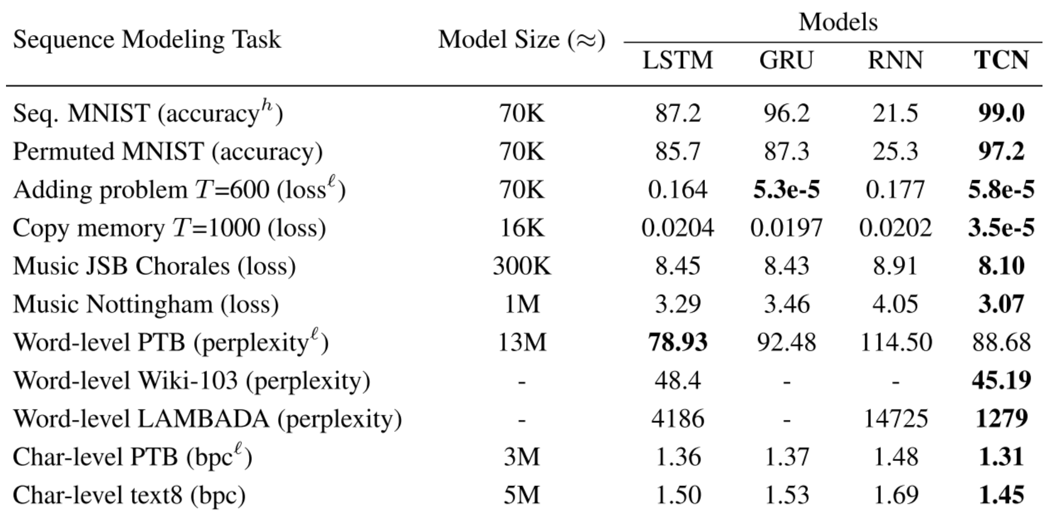
4 Results

The datasets are explained well in the paper. The results comparing TCN with standard RNN are shown in Figure 2. The comparison with SOTA is shown in Figure 3.

5 Conclusion

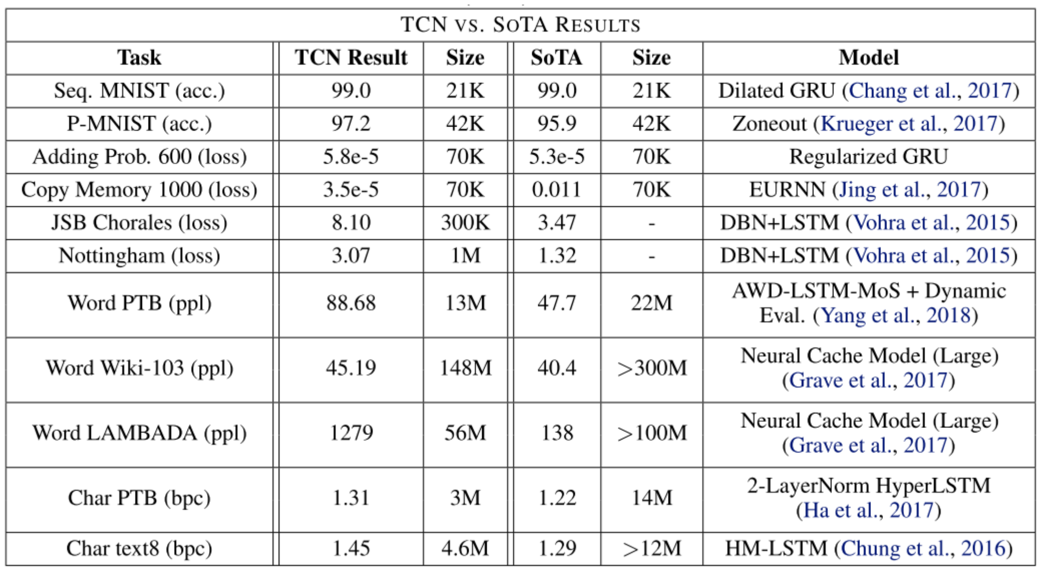
The results strongly suggest that the generic TCN architecture with minimal tuning outperforms canonical recurrent architectures across a broad variety of sequence modeling tasks that are commonly used to benchmark the performance of recurrent architectures themselves.

TCN excel at the adding and copy tasks where they achieve perfect accuracy. The ability to memorize information from distant past has been evaluated on the copy task with increasing length of the sequence between the input that needs to be remembered and the output where it needs to be reproduced. TCNs consistently converged to 100% accuracy for all sequence lengths, whereas LSTMs and GRUs of the same size quickly degenerated to random guessing as the sequence length T grows.



Sequence Modeling Task	Model Size (≈)	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**

Figure 2: Comparison of TCN and standard RNN. The TCNs had tuned k, d, number of layers, gradient clipping. They did a grid search over optimizer, recurrent drop, learning rate, gradient clipping, and initial forget-gate bias. Both TCN and RNN should have similar number of parameters in the comparisons.



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)

Figure 3: Comparison of TCN and State-of-the-art results on the datasets.

References

[1] Nielsen, Michael A. Neural networks and deep learning. Vol. 25. USA: Determination press, 2015.

[2] "CS231n Convolutional Neural Networks for Visual Recognition" `http://cs231n.github.io/convolutional-networks/`

[3] Olah, Christopher. "Understanding lstm networks." (2015). `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

[4] Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling." arXiv preprint arXiv:1803.01271 (2018). `https://arxiv.org/abs/1803.01271`

Code: `https://github.com/locuslab/TCN`