

Learning Longer-term Dependencies in RNNs with Auxiliary Losses

Trieu H. Trinh¹ Andrew M. Dai Thang Luong Quoc V. Le
 {thtrieu, adai, thangluong, qvl}@google.com

Abstract

Despite recent advances in training recurrent neural networks (RNNs), capturing long-term dependencies in sequences remains a fundamental challenge. Most approaches use backpropagation through time (BPTT), which is difficult to scale to very long sequences. This paper proposes a simple method that improves the ability to capture long term dependencies in RNNs by adding an unsupervised auxiliary loss to the original objective. This auxiliary loss forces RNNs to either reconstruct previous events or predict next events in a sequence, making truncated backpropagation feasible for long sequences and also improving full BPTT. We evaluate our method on a variety of settings, including pixel-by-pixel image classification with sequence lengths up to 16 000, and a real document classification benchmark. Our results highlight good performance and resource efficiency of this approach over competitive baselines, including other recurrent models and a comparable sized Transformer. Further analyses reveal beneficial effects of the auxiliary loss on optimization and regularization, as well as extreme cases where there is little to no back-propagation.

1. Introduction

Many important applications in artificial intelligence require the understanding of long term dependencies between events in a sequence. For example, in natural language processing, it is sometimes necessary to understand relationships between distant events described in a book to answer questions about it. Typically, this is achieved by gradient descent and BPTT (Rumelhart et al., 1986) with recurrent networks. Learning long term dependencies with gradient descent, however, is difficult because the gradients computed by BPTT tend to vanish or explode during training (Hochreiter et al., 2001). Additionally, for BPTT to work, one needs to store the intermediate hidden states

¹Work done as a member of the Google Brain Residency program (g.co/brainresidency.)

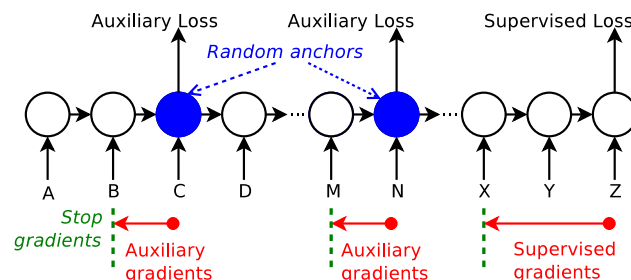


Figure 1. An overview of our method. The auxiliary loss improves the memory of the recurrent network such that the number of steps needed for the main task’s BPTT is small.

in the sequence. The memory requirement is therefore proportional to the sequence length, making it difficult to scale to large problems.

Several promising approaches have been proposed to alleviate the aforementioned problems. First, instead of using the vanilla recurrent network, one can use Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997), which is designed to improve gradient flow in recurrent networks. In addition, one can also use gradient clipping (Pascanu et al., 2013) to stabilize the training of the LSTM. Finally, to reduce the memory requirement, one can either store the hidden states only periodically (Gruslys et al., 2016; Chen et al., 2016), use truncated BPTT, or use synthetic gradients (Jaderberg et al., 2017).

Convolutional neural networks also mitigate the problem of long-term dependencies since large kernel sizes and deep networks such as ResNets (He et al., 2016) allow long-term dependencies to be learnt across distant parts of an image. However, this is a fundamentally different kind of architecture that has other tradeoffs. For example, the entire input (an image or sequence) and the intermediate activations of the model must be stored in memory during training. At inference time, typical CNNs also need $O(n)$ storage where n is the size of the input.² The Transformer (Vaswani et al., 2017) has a similar issue, though somewhat magnified since computation for training and inference requires ran-

²A convolutional layer is often followed by a reduction layer to reduce the input size by a constant factor.

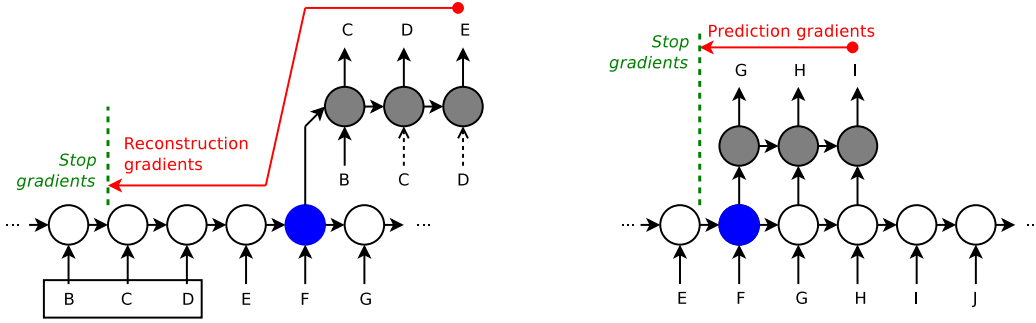


Figure 2. An overview of our methods. For each random anchor point, say F, we build an auxiliary loss at its position. **Left:** We predict a random subsequence BCD that occurs before F. B is inserted into a decoder network to start the reconstruction, while C and D is optionally fed. **Right:** We predict the subsequence GHI by stacking an auxiliary RNNs on top of the main one. Gradients from auxiliary loss is truncated in both cases to keep the overall cost of BPTT constant.

dom access to storage that is $O(n)$.

RNNs therefore have the advantage where assuming a fixed BPTT length of l , training requires $O(l)$ storage. This is commonly the case when training language models on the PTB dataset (Marcus et al., 1994), where the state is never reset over the entire 1 million token sequence. Therefore, in theory the RNN can learn relationships across this extremely long distance. Furthermore, inference in RNNs also requires $O(1)$ storage since RNNs do not need to ‘look back’.

In this paper, we propose an orthogonal technique to further address the weakness of recurrent networks purely relying on BPTT. Our technique introduces an unsupervised auxiliary loss to the main supervised loss that reconstructs/predicts a random segment in the sequence before/after an anchor point. This enables learning with only need a few BPTT steps from the supervised loss.

Our results show that unsupervised auxiliary losses significantly improve optimization and generalization of LSTMs. Moreover, using this technique, one does not have to perform lengthy BPTT during training to obtain good results. Our method, therefore, lends itself to very long sequences where vanishing/exploding gradients as well as the cost of lengthy BPTT become critical bottlenecks.

In our experiments where sequences of up to 16000 elements is processed, LSTMs with auxiliary losses can train much faster and with less memory usage, while training LSTMs with full backprop becomes very difficult.

2. Related works

As learning long term dependencies with recurrent networks is an important problem in machine learning, many approaches have been proposed to tackle this challenge. Well known approaches include recurrent networks with special structures (El Hihhi & Bengio,

1996; Sperduti & Starita, 1997; Frasconi et al., 1998; Socher et al., 2011; Chan et al., 2016), Long Short-Term Memory Networks (Hochreiter & Schmidhuber, 1997; Gers et al., 1999; Graves, 2013), Gated Recurrent Unit Networks (Cho et al., 2014; Chung et al., 2014), multiplicative units (Wu et al., 2016), specialized optimizers (Martens & Sutskever, 2011; Kingma & Ba, 2014), identity initialization and connections (Mikolov et al., 2014; Le et al., 2015; He et al., 2016), highway connections (Zilly et al., 2017), orthogonal or unitary-constrained weights (White et al., 2004; Henaff et al., 2016; Arjovsky et al., 2016), dilated convolutions (Salimans et al., 2017), connections (Koutnik et al., 2014) and attention mechanisms (Bahdanau et al., 2015; Luong et al., 2015; Vaswani et al., 2017). A more recent approach is to skip input information at certain steps (Yu et al., 2017; Seo et al., 2018; Campos et al., 2018). As training very long recurrent networks is memory-demanding, many techniques have also been proposed to tackle this problem (Chen et al., 2016; Gruslys et al., 2016; Jaderberg et al., 2017). We propose methods that are orthogonal to these approaches, and can be used in combination with them to improve RNNs.

Our work is inspired by recent approaches in pretraining recurrent networks (Dai & Le, 2015; Ramachandran et al., 2017) with sequence autoencoders or language models. Their work, however, focuses on short sequences, and using pretraining to improve generalization of these short recurrent networks. In contrast, our work focuses on longer sequences, and studies the effects of auxiliary losses in learning long term dependencies.

Combining auxiliary losses and truncated BPTT is also described in the context of online learning (Schmidhuber, 1992), where the main network learns to predict the concatenation of its next input token, the target vector, and distilled knowledge from an auxiliary network. The auxiliary network only predicts the sequence of tokens that is

not predicted correctly by the main network. This shorter sequence is termed the *compressed history* and is argued to suffice for good classification.

3. Methodology

An overview of our methods is shown in Figure 1. Let us suppose that the goal is to use a recurrent network to read a sequence and classify it. We propose to randomly sample one or multiple anchor positions, and insert an unsupervised auxiliary loss at each of them.

3.1. Reconstruction auxiliary loss

In reconstructing past events, we sample a subsequence before the anchor point, and insert the first token of the subsequence into a decoder network; we then ask the decoder network to predict the rest of the subsequence. The whole process is illustrated in Figure 2-left.

Our intuition is that if the events to be predicted are close enough to the anchor point, the number of BPTT steps needed for the decoder to reconstruct past events can be quite small. Furthermore, with this training, the anchor points serve as a temporary memory for the recurrent network to remember past events in the sequence. If we choose enough anchor points, the memory is built over the sequence such that when we reach sequence end, the classifier remembers enough about the sequence and can do a good job at classifying it. Consequently, the classifier only needs a few backpropagation steps to fine-tune the LSTM’s weights, since good embeddings of the input sequence have been learnt by optimizing the auxiliary objective.

3.2. Prediction auxiliary loss

Another auxiliary loss of consideration is analogous to Language Modelling loss, illustrated in Figure 2-right. In this case, we ask the decoder network to predict the next token given the current one sequentially, over a subsequence starting from the anchor point. This type of unsupervised auxiliary loss is first examined by Dai & Le (2015), where it is applied over the whole input sequence. In our experiments, however, we are interested in scalable schemes of learning long term dependencies, we therefore only apply this loss on a subsequence after the random anchor point.

3.3. Training

We name the former method *r*-LSTM, the later *p*-LSTM (which respectively stand for *reconstruct*- and *predict*-LSTM) and train them in two phases. The first is pure unsupervised pretraining where only the auxiliary loss is minimized. In the second phase, semi-supervised learning is performed where we minimize the sum of the main objec-

tive loss $L_{\text{supervised}}$ and our auxiliary loss $L_{\text{auxiliary}}$. The auxiliary LSTM that performs reconstruction is trained with Scheduled Sampling (Bengio et al., 2015a).

3.4. Sampling frequency and subsequence length

By introducing the auxiliary losses over subsequences of the input, one inevitably introduces extra hyper-parameters. One of them indicates how frequently one should sample the reconstruction segments, the others indicate how long each segment should be. Denoting the former n , and the later $\{l_i\}_{i=1}^n$, we obtain the auxiliary loss as follows:

$$L_{\text{auxiliary}} = \frac{\sum_{i=1}^n L_i}{\sum_{i=1}^n l_i} \quad (1)$$

Where L_i denotes the loss evaluated on the i^{th} sampled segment, and is calculated by summing losses on all predicted tokens (TokenLoss_t) in that segment.

$$L_i = \sum_{t=1}^{l_i} \text{TokenLoss}_t \quad (2)$$

For sequences of characters, each TokenLoss_t is the cross-entropy loss between the ground truth one-hot vector and the predicted distribution over the vocabulary produced by our decoder network. For other types of input, we treat each token as a continuous, multi-dimensional real vector and perform L_2 distance minimization.

Tuning hyper-parameters is known to be very expensive, especially so when training RNNs on very long sequences. We therefore set all sampled segments to the same length: $l_i = l \forall i$, and sample at frequency $n = 1$ in most experiments. Tuning these hyper-parameters is also explored in cases where sequence length is relatively short. In later experiments, we show that the tuned values generalize well to much longer input sequences.

4. Experiments

To evaluate the effectiveness of our models, we consider a wide variety of datasets with sequences of varying lengths from 784 to 16384. Our first benchmark is a *pixel-by-pixel image classification* task on MNIST in which pixels of each image are fed into a recurrent model sequentially before a prediction is made. This dataset was proposed by Le et al. (2015) and has now become the most popular benchmark for testing long term dependency learning.³

Beside MNIST, we also explore pMNIST, a harder version, where each pixel sequence is permuted in the same way. Permuting pixels breaks apart all local structures and

³No symbol was added to indicate the end of each row of pixels.

Table 1. Datasets and average sequence length.

Dataset	Mean length	# classes	Train set size
MNIST	784	10	60K
pMNIST	784	10	60K
CIFAR10	1024	10	50K
StanfordDogs ⁴	1600 – 16384	120	150K
DBpedia	300	14	560K

creates even more complex dependencies across various time scales. To test our methods on a larger dataset, we include pixel-by-pixel CIFAR10 (no permutation). Additionally, to perform control experiments with several scales of sequence lengths, we use the StanfordDogs dataset (Khosla et al., 2011) which contains large images categorized to 120 dog breeds. All images are scaled down to 8 different sizes from 40×40 to 128×128 before being flattened into sequences of pixels without permutation. This setup results in sequences of lengths up to 16 000, which is over 20 times longer than any previously used benchmark of this flavor.

Lastly, we explore how well truncated BPTT and the auxiliary losses can perform on a real language task, where previous RNN variants have already reported remarkable accuracy. For this task, the DBpedia character level classification task is chosen as it is a large-scale dataset (with 560K training examples) and has been well benchmarked by Dai & Le (2015). We follow the procedure suggested in Zhang et al. (2015) to normalize the dataset.

A summary of all datasets being used is presented in Table 1.

4.1. Model Setup

We use a single-layer LSTM with 128 cells and an embedding size of 128 to read the input sequence. For the supervised loss, the final state of the main LSTM is passed through a two-layer feedforward network (FFN) with 256 hidden units, before making a prediction. We apply drop-connect (Wan et al., 2013) with probability 0.5 on the second layer. For the auxiliary losses, we use a two-layer LSTM in which the bottom layer is initialized from the current state of the main classification LSTM, while the top one starts with zero state. When reconstructing image pixels, a two-layer FFN (256 units, drop-connect 0.5 on second layer) is applied on top of the auxiliary LSTM per timestep.

Our RNNs are trained using the RMSProp opti-

⁴We follow the procedure suggested in Sermanet et al. (2014) to obtain a larger training set, while keeping the same test set. All images are scaled down to 8 different sizes from 40×40 to 128×128 before being flattened into sequences of pixels.

mizer (Tieleman & Hinton, 2012) with batch size of 128. Unsupervised pretraining is done in 100 epochs with initial learning rate of 0.001, which is halved to 0.0005 halfway through pretraining. For the semi-supervised phase, the same learning rate is halved every 300 epochs until training reaches 1000 epochs. Scheduled sampling for auxiliary LSTMs is annealed linearly to zero after 100 000 training steps.

As we scale our methods to various input lengths, we make sure that backpropagation cost is constant regardless of the input length. Specifically, gradients are truncated to 300 time steps for both the supervised and auxiliary losses.⁵ For the auxiliary losses, we choose the simplest setup of sampling $n=1$ segment of length $l=600$ per training example. In Section 5.2, we will explore different values for n and l .

As a complement to results from purely recurrent models, in Section 4.3, we will also compare our models with Transformer (Vaswani et al., 2017). Transformer is a completely different paradigm of processing sequences that sidesteps the difficulties of BPTT through the use of self-attention. Such advantage is achieved at the cost of $O(n)$ working memory during both training and inference compared to $O(1)$ for RNNs. Even though our main interest is to improve over recurrent models, we include these results to study how scalable the self-attention mechanism is.

We use Tensor2Tensor⁶ to train Transformer models with an off-the-shelf configuration that has a comparable number of parameters as our RNNs (0.5M weights)⁷. A simple setting for classification is adopted where the Transformer output vectors is average-pooled and fed into a two-layer FFN before making predictions, as done in our RNNs.

4.2. Main results

4.2.1. MNIST, PMNIST, AND CIFAR10

We first explore sequences of length no longer than 1000 on MNIST, pMNIST and CIFAR10. Besides results from previous works on pixel MNIST and permuted MNIST (pMNIST) such as Le et al. (2015); Arjovsky et al. (2016), we evaluated a fully trained LSTM and an LSTM trained with only 300 steps of BPTT as the main baselines to see how much disadvantage truncating classification gradients might cause. At this stage, it is also affordable to include test accuracies from both truncated and fully-trained r -LSTM and p -LSTM for a more complete result.

An overview of Table 2 shows that our proposed auxil-

⁵All models are implemented in TensorFlow (Abadi et al., 2015). Truncated gradients are achieved using the built-in `tf.stop_gradient` op.

⁶<https://github.com/tensorflow/tensor2tensor>

⁷`transformer_tiny`.

Table 2. Test accuracy (%) on MNIST, pMNIST, and CIFAR10.

	MNIST	pMNIST	CIFAR10
iRNN (Le et al., 2015)	97.0	82.0	N/A
uRNN (Arjovsky et al., 2016)	95.1	91.4	N/A
LSTM Full BP	98.3	89.4	58.8
LSTM Truncate300	11.3	88.8	49.0
<i>r</i> -LSTM Truncate300	96.4	92.8	65.9
<i>p</i> -LSTM Truncate300	95.4	92.5	64.7
<i>r</i>-LSTM Full BP	98.4	95.2	72.2
<i>p</i> -LSTM Full BP	98.0	92.8	67.6

ary losses produce gradually larger improvements moving from MNIST to pMNIST and CIFAR10. On pixel-by-pixel MNIST, our truncated LSTM baseline is nearly untrainable, with only 11.3% accuracy. This is due to the fact that gradients back-propagated from the loss can only reach largely non-informative solid pixels near the end of the sequence. Despite this detrimental effect of gradient truncation, the proposed unsupervised losses bring *r*-LSTM and *p*-LSTM on par with fully trained RNNs like uRNN and LSTM.

On permuted pMNIST where more complex long-range dependencies is put to the test, *r*-LSTM and *p*-LSTM easily outperform the fully trained LSTM baselines as well as a fully trained uRNN, while using less than half the number of gradients from the classification loss.

On CIFAR10, we observe an even greater discrepancy, where *r*-LSTM is followed closely by *p*-LSTM in accuracy, while a fully trained LSTM is more than 7% lower in absolute accuracy.

With fully backpropagated gradients from classification loss, we obtain the best accuracy across all datasets against other recurrent models. Notably on the two harder benchmarks pMNSIT and CIFAR10, *r*-LSTM outperforms a fully-trained LSTM by a large margin.

4.2.2. STANFORDDOGS

So far, our experiments give hints that *r*-LSTM and *p*-LSTM scale better in performance when input sequences get longer and more complex. Next, we present how this trend elaborates when input sequences extend up to an order of magnitude higher – over 10 000 steps. As presented earlier, we use the dataset StanfordDogs resized down to 8 levels of sequence lengths and test the models on all levels.

As this range, training is expensive in terms of time and computational resources, especially so with LSTMs where parallelization over the time dimension is not possible. We therefore restrict each training session to the same amount of resource (a single Tesla P100 GPU) and report infeasible whenever a mini-batch of one training example can no longer fit into memory.

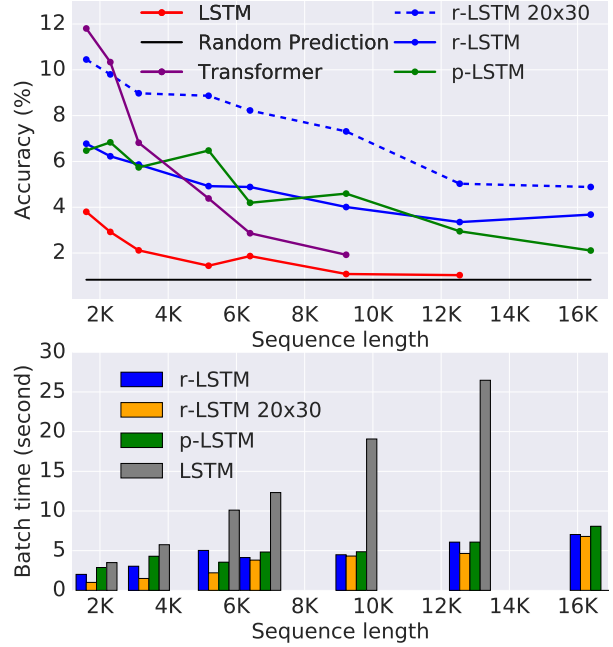


Figure 3. Top: Test accuracy on StanfordDogs resized to 8 levels of sequence length. Bottom: Time to run a single mini batch of 128 training examples, measured in second.

In Figure 3-top, we report test accuracy from a fully back-propagated LSTM baseline, *r*-LSTM, and *p*-LSTM on all levels. Since StanfordDogs is an even more challenging classification problem compared to CIFAR10, pursuing useful accuracy with non-convolutional models is not our main goal. We instead examine the relative robustness of different methods when input sequences get longer. All models are evaluated after 100 epochs of training, with an additional 20 epochs of pretraining for models with auxiliary loss.

Using the unsupervised auxiliary losses, we are able to obtain much better results compared to other methods. Figure 3-top shows that both *r*-LSTM and *p*-LSTM exhibit the strongest resistance to the growing difficulty, while an LSTM trained with full backpropagation is slow to improve and produces no better than random predictions when the input sequence length reaches the 9 000 mark. After the 12 000 mark, memory constraint is exceeded for this model. At the same time, there is virtually no accuracy loss in *r*-LSTM going from 12 000 to 16 000 element long sequences.

The gradient truncation in *r*-LSTM and *p*-LSTM also offers a much greater computational advantage as sequence length gets arbitrarily large. Figure 3-bottom illustrates the time to finish one training step for each recurrent model. LSTM takes 4 seconds at the 1600 mark and quickly stretches out to 26 seconds at the 12 000 mark. With the

same computational resource, our proposed methods stay under 3 seconds and grow up to only around 8 seconds at the end, processing a batch of sequences with lengths more than 16 000.

4.3. Comparing with Transformer

In this set of experiments, we explore how well our proposed recurrent models fare with those that utilize a self-attention mechanism. As noted in the introduction, these models require random access to the entire sequence at inference time, so are very quick to become infeasible as sequences get longer (such as the PTB LM dataset).

4.3.1. MNIST, PMNIST, AND CIFAR10

On MNIST and pMNIST, Transformer outperforms our best model as shown in Table 3. On CIFAR10, however, Transformer performance drops significantly - worse than most recurrent models on this dataset.

Table 3. Test accuracy (%) on MNIST, pMNIST and CIFAR10.

	MNIST	pMNIST	CIFAR10
<i>r</i> -LSTM Full BP	98.4	95.2	72.2
Transformer	98.9	97.9	62.2

We additionally evaluate results from T-DMCA (Liu et al., 2018), though strictly speaking, this is an unfair comparison since the T-DMCA adds convolutions at each self-attention layer and thus does additional $O(n)$ processing at inference time. Compared to the Transformer, T-DMCA is more memory efficient as it utilizes local-attention and memory-compressed attention.⁸ Results indicate that T-DMCA performs better than Transformer on MNIST (99.3%) and CIFAR10 (73.0%). On pMNIST where there is no spatial locality to be exploited by convolution, T-DMCA achieves 97.6% accuracy, slightly worse than that of Transformer.

4.3.2. STANFORDDOGS

Similar to the previous section, we transfer the same hyper-parameter settings of Transformer to much longer sequences, created using StanfordDogs dataset. As shown in Figure 3, Transformer starts with almost twice the accuracy of *r*-LSTM or *p*-LSTM, but this performance quickly degrades at a much higher rate as input sequences get longer. Specifically, Transformer performs worse than our methods after the 3000 mark and end up only slightly better than random prediction around the 9200 mark. Its training using the

⁸In our experiments, Transformer-DMCA consists of 5 alternating layers of unmasked local attention and memory compressed attention, with all hidden sizes and filter sizes set to 128.

same resource also becomes infeasible after this point.

Note that our proposed method is orthogonal to most models that process sequences. Incorporating our technique to any scalable Transformer variant will therefore likely result in significant improvements. Our current work, however, focuses on improving recurrent networks and therefore leaves this option for future exploration.

4.4. Classifying DBpedia documents at character level

We explore how well truncated BPTT and the auxiliary losses can do on sequences of discrete data (text), where previous methods already reported remarkable accuracy. For this task, the DBpedia dataset is chosen as it provides a large and carefully curated set of clean Wikipedia texts and no duplication. In our experiments, each document in the dataset is processed at character level (Zhang et al., 2015). This makes the average sequence length 300, with 99% of the training examples are under 600 elements long.

Table 4. Test error rate (%) on character-by-character DBpedia.

	Test error
LSTM Full Backprop (Dai & Le, 2015)	13.64
LM-LSTM Truncate100	4.04
SA-LSTM Truncate100	3.89
<i>r</i> -LSTM Truncate100	4.02
<i>r</i> -LSTM 20x15 Truncate100	3.84
<i>p</i>-LSTM Truncate100	2.85

To explore how well auxiliary losses can help with limited backpropagation, supervised gradients are truncated to only 100 time steps, while anchored subsequences are sampled with length $l = 300$. Similar to Dai & Le (2015), we did not perform joint-training since it slightly degrades performance on this large dataset, all other hyper-parameters are reused. We also test *r*-LSTM with the 20-sample setting, a full BPTT trained LSTM baseline and truncated LM-LSTM and SA-LSTM (Dai & Le, 2015) baselines⁹.

As can be seen in Table 4, auxiliary losses with truncated BPTT can significantly outperform the LSTM baseline by more than 10% absolute accuracy. Our methods also have better results than truncated LM-LSTM and SA-LSTM. We conjecture that this comes from the combination of more randomness and truncation in our training process.

⁹There are stronger but not directly comparable baselines on the DBpedia dataset, such as full BPTT SA-LSTM or LM-LSTM.

5. Analysis

5.1. Shrinking supervised BPTT length

Given the clear trend demonstrated in previous sections, it is natural to ask the question of how much longer the input has to grow before r -LSTM and p -LSTM becomes untrainable. To simulate this effect without growing sequence length indefinitely, we instead keep the input sequence length fixed, while truncating backpropagation incrementally. We perform experiments on CIFAR10 and start shrinking the BPTT length from 300 down to 1 - where gradients from the classification loss have minimal impact on the main LSTM.

Results in Figure 4 shows that r -LSTM and p -LSTM can afford a reduction of another 200 BPTT steps, while still being able to generalize better than a fully trained LSTM. Moreover, by applying gradients on only 50 steps – less than 5% of the total input steps, r -LSTM and p -LSTM’s accuracy can still approximate their fully trained counterpart. At the extreme point of one-step backpropagation, both r -LSTM (46.1%) and p -LSTM (47.0%) perform commendably well.

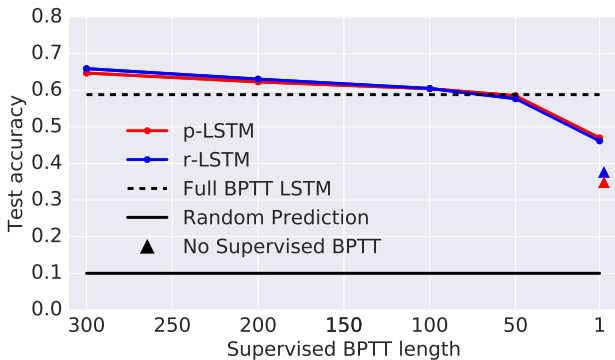


Figure 4. Effects of shrinking supervised BPTT length.

Going one step further, we prevent classification gradients from updating the main LSTM, thereby training it completely unsupervised. By doing so, we attempt to address the question of why the human brain can understand long sequences of events, even though BPTT is argued biologically implausible (Bengio et al., 2015b). Results from Figure 4 indicate that both r -LSTM (37.7%) and p -LSTM (34.9%) can still classify unseen data with far-from-random accuracy.

5.2. Multiple reconstructions with fixed BPTT cost

So far, we only adopt the simplest setting of $n = 1$ reconstruction sample per sequence. One can also tune this hyper-parameter for even better results. We explore this option to improve one and zero step supervised BPTT.

To keep the total cost of backpropagation approximately the same with previous experiments, we gradually increase n and shrink each subsequence length l proportionately. We also set the unsupervised BPTT truncation to be l . In Table 5, we report results obtained with five different sampling frequencies, ranging from 10 to 200 samples.

Table 5. Classification test accuracy (%) on CIFAR10 with varying sample frequency and fixed backpropagation cost.

n	l	Supervised BPTT length	
		1	0
10	60	46.0	40.6
20	30	48.0	41.6
50	12	47.7	41.0
100	6	47.2	40.1
200	3	46.2	37.9

We indeed observed accuracy gain on the test set across almost all frequencies of sampling. Interestingly, there is a peak at 20 samples per sequence and the accuracy gain starts decaying from this point in both directions. In other words, it is harmful to sample too little, or sample too many at the cost of very little backpropagation. Comparing these two extremes, we observe slightly better accuracy with many small reconstructions than one big reconstruction.

At sampling frequency 20, for single time step backpropagation, we obtain an increase of 2.0%. For completely unsupervised training (no backpropagation on the main LSTM), there is a remarkable increase of 4.0%. This increase implies that r -LSTM has great potential to improve on long sequences, with relatively few supervised gradients, as long as one is able to afford tuning extra hyper-parameters.

We explore this potential on StanfordDogs by retraining r -LSTM with sampling frequency 20 (r -LSTM 20×30) on all 8 levels. As shown in Figure 3-top, this best performing setting found on CIFAR10 generalizes to all difficulty levels of StanfordDogs. Namely, r -LSTM 20×30 closes the gap with Transformer on shorter sequences, and stays at this top position throughout, outperforming all other recurrent models as well as Transformer by a large margin starting from the 3000 mark.

Furthermore, by independently sampling several segments of equal length, one can batch them to utilize data parallelism and subsequently speed up the training process even more. This is illustrated in Figure 3-bottom, where single-batch training time of r -LSTM 20×30 consistently stays lower than that of any other recurrent method.

5.3. Regularization and Optimization Advantages from Unsupervised Losses

With a significant gap between r -LSTM/ p -LSTM and a fully-trained LSTM on almost all benchmarks, it is important to understand why. In particular, we ask whether it is regularization or optimization advantage that is added by truncated BPTT and our auxiliary losses. At any point during training, we identify optimization advantage when training accuracy with auxiliary losses are much better than that of the baseline, while the corresponding improvement on test set is not as significant. On the other hand, if our models generalize better while being harder or insignificantly easier to train, the improvement comes from regularization.

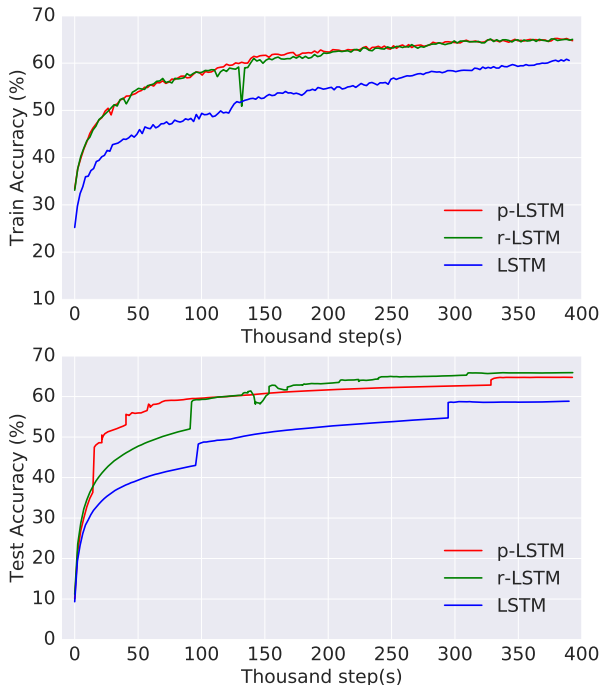


Figure 5. Effects of auxiliary losses on training and testing accuracy.

Figure 5 shows the training/testing accuracy of r -LSTM, p -LSTM and an LSTM during training. r -LSTM and p -LSTM training curves trace each other almost identically throughout, while r -LSTM gives better result on testing data. This implies that r -LSTM regularizes better than p -LSTM.

Comparing to the LSTM baseline, r -LSTM and p -LSTM start off with much higher training accuracy while having the same testing accuracy (10%). This reveals the significant improvement from unsupervised pretraining for both r -LSTM and p -LSTM’s optimization. Gradually throughout the training process, this optimization gap with the baseline becomes smaller, while the corresponding difference in test

accuracy becomes relatively bigger.

We therefore conclude that both types of pretraining bring optimization advantages at early stages of training. Later on, minimizing the semi-supervised loss creates a regularization effect that quickly takes over until the end.

5.4. Ablation Study

In this section, we evaluate the relative contribution of different factors to r -LSTM’s performance. Here we test each factor by turning it off and retraining the model from scratch on CIFAR10, using the same random seed. Firstly, as reported in Table 2, eliminating the auxiliary loss and leaving the main LSTM with a truncation of 300 BPTT steps cause a loss of nearly 17% in test accuracy. With the auxiliary loss in effect, Figure 6 shows the results when turning off other parts from the original full setting.

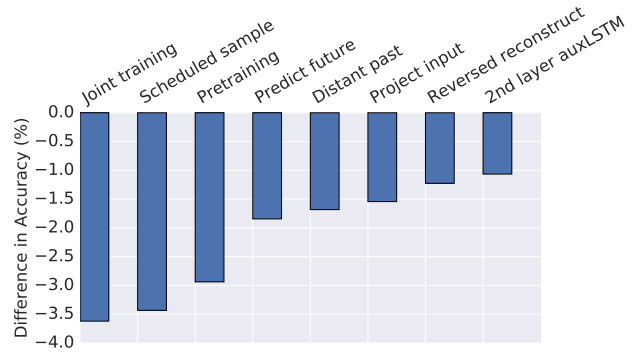


Figure 6. Ablation analysis of r -LSTM performance.

Jointly training unsupervised and supervised loss is most important, with a corresponding loss of more than 3.6% reported. As long as joint training is in effect, pre-training is slightly less important than applying Scheduled Sampling for the auxiliary LSTMs.

More randomness is better. Instead of only reconstructing the immediate past, allowing reconstruction segments to be randomly sampled in distant past gives almost a 2% accuracy gain. Allowing a part of the sampled segment to spread over to the anchor point’s future also gives a boost.

Other improvements come from embedding input pixels to the same dimensionality as the LSTM’s hidden size, reversing the order of reconstruction and stacking a second layer on the LSTM that receives outputs from the anchor point.

6. Conclusion

In this paper, we have presented a simple approach to improve the learning of long-term dependencies in RNNs. An auxiliary loss was added to the main supervised loss to offer two main benefits. First, it induces a regularization ef-

fect, allowing our models to generalize well to very long sequences, up to length 16 000. Second, it provides computational advantages as the input sequence gets very long, so that one only needs to backpropagate for a small number of time steps to obtain competitive performance. In the extreme cases where there is little to no backpropagation, our models perform far better than random predictions.

On a comprehensive set of benchmarks, ranging from pixel-by-pixel image classification (MNIST, pMNIST, CIFAR10, StanfordDogs) to character-level document classification (DBpedia), our models have demonstrated competitive performance over strong recurrent baselines (iRNN, uRNN, LM-LSTM, SA-LSTM) and non-recurrent ones such as Transformer. For long sequences, our results are superior despite using much fewer resources.

We anticipate that this simple technique will be widely applicable to any system that processes unusually long sequences.

References

- Abadi, Martín, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S., Davis, Andy, Dean, Jeffrey, Devin, Matthieu, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
- Arjovsky, Martin, Shah, Amar, and Bengio, Yoshua. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128, 2016.
- Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations*, 2015.
- Bengio, Samy, Vinyals, Oriol, Jaitly, Navdeep, and Shazeer, Noam. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 1171–1179, 2015a.
- Bengio, Yoshua, Lee, Dong-Hyun, Bornschein, Jörg, and Lin, Zhouhan. Towards biologically plausible deep learning. *CoRR*, abs/1502.04156, 2015b.
- Campos, Víctor, Jou, Brendan, Giró-i Nieto, Xavier, Torres, Jordi, and Chang, Shih-Fu. Skip RNN: Learning to skip state updates in recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2018.
- Chan, William, Jaitly, Navdeep, Le, Quoc, and Vinyals, Oriol. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pp. 4960–4964. IEEE, 2016.
- Chen, Tianqi, Xu, Bing, Zhang, Chiyuan, and Guestrin, Carlos. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.
- Cho, Kyunghyun, Van Merriënboer, Bart, Gulcehre, Caglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.
- Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.
- Dai, Andrew M. and Le, Quoc V. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, pp. 3079–3087, 2015.
- El Hihi, Salah and Bengio, Yoshua. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in Neural Information Processing Systems*, pp. 493–499, 1996.
- Frasconi, Paolo, Gori, Marco, and Sperduti, Alessandro. A general framework for adaptive processing of data structures. *IEEE transactions on Neural Networks*, 9(5):768–786, 1998.
- Gers, Felix A, Schmidhuber, Jürgen, and Cummins, Fred. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 1999.
- Graves, Alex. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.
- Gruslys, Audrunas, Munos, Rémi, Danihelka, Ivo, Lanctot, Marc, and Graves, Alex. Memory-efficient backpropagation through time. In *Advances in Neural Information Processing Systems*, pp. 4125–4133, 2016.
- He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- Henaff, Mikael, Szlam, Arthur, and LeCun, Yann. Recurrent orthogonal networks and long-memory tasks. In *International Conference on Machine Learning*, 2016.

- Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- Hochreiter, Sepp, Bengio, Yoshua, Frasconi, Paolo, and Schmidhuber, Jürgen. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In *A field guide to dynamical recurrent neural networks*. IEEE Press, 2001.
- Jaderberg, Max, Czarnecki, Wojciech Marian, Osindero, Simon, Vinyals, Oriol, Graves, Alex, and Kavukcuoglu, Koray. Decoupled neural interfaces using synthetic gradients. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.
- Khosla, Aditya, Jayadevaprakash, Nityananda, Yao, Bangpeng, and Fei-Fei, Li. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, Colorado Springs, CO, June 2011.
- Kingma, Diederik P and Ba, Jimmy. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.
- Koutnik, Jan, Greff, Klaus, Gomez, Faustino, and Schmidhuber, Jürgen. A clockwork RNN. In *International Conference on Machine Learning*, pp. 1863–1871, 2014.
- Le, Quoc V., Jaitly, Navdeep, and Hinton, Geoffrey E. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.
- Liu, Peter J., Saleh, Mohammad, Pot, Etienne, Goodrich, Ben, Sepassi, Ryan, Kaiser, Lukasz, and Shazeer, Noam. Generating Wikipedia by summarizing long sequences. In *International Conference on Learning Representations*, 2018.
- Luong, Minh-Thang, Pham, Hieu, and Manning, Christopher D. Effective approaches to attention-based neural machine translation. In *Association of Computational Linguistics*, 2015.
- Marcus, Mitchell, Kim, Grace, Marcinkiewicz, Mary Ann, MacIntyre, Robert, Bies, Ann, Ferguson, Mark, Katz, Karen, and Schasberger, Britta. The Penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology, HLT '94*, pp. 114–119, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics. ISBN 1-55860-357-3.
- Martens, James and Sutskever, Ilya. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning*, pp. 1033–1040. Citeseer, 2011.
- Mikolov, Tomas, Joulin, Armand, Chopra, Sumit, Mathieu, Michael, and Ranzato, Marc’Aurelio. Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*, 2014.
- Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013.
- Ramachandran, Prajit, Liu, Peter J, and Le, Quoc V. Unsupervised pretraining for sequence to sequence learning. In *Empirical Methods in Natural Language Processing*, 2017.
- Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pp. 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-68053-X.
- Salimans, Tim, Karpathy, Andrej, Chen, Xi, and Kingma, Diederik P. PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications. In *International Conference on Learning Representations (ICLR)*, 2017.
- Schmidhuber, Jürgen. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.
- Seo, Minjoon, Min, Sewon, Farhadi, Ali, and Hajishirzi, Hannaneh. Neural speed reading via skim-rnn. In *International Conference on Learning Representations (ICLR)*, 2018.
- Sermanet, Pierre, Frome, Andrea, and Real, Esteban. Attention for fine-grained categorization. *CoRR*, abs/1412.7054, 2014.
- Socher, Richard, Lin, Cliff, Ng, Andrew Y., and Manning, Christopher D. Parsing natural scenes and natural language with recursive neural networks. In *International Conference on Machine Learning*, 2011.
- Sperduti, Alessandro and Starita, Antonina. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8(3):714–735, 1997.
- Tieleman, T. and Hinton, G. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSE: Neural Networks for Machine Learning, 2012.

- Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N, Kaiser, Łukasz, and Polosukhin, Illia. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 6000–6010, 2017.
- Wan, Li, Zeiler, Matthew, Zhang, Sixin, Le Cun, Yann, and Fergus, Rob. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pp. 1058–1066, 2013.
- White, Olivia L, Lee, Daniel D, and Sompolinsky, Haim. Short-term memory in orthogonal neural networks. *Physical review letters*, 92(14):148102, 2004.
- Wu, Yuhuai, Zhang, Saizheng, Zhang, Ying, Bengio, Yoshua, and Salakhutdinov, Ruslan R. On multiplicative integration with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 2856–2864, 2016.
- Yu, Adams Wei, Lee, Hongrae, and Le, Quoc V. Learning to skim text. In *Association of Computational Linguistics*, 2017.
- Zhang, Xiang, Zhao, Junbo, and LeCun, Yann. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28*, 2015.
- Zilly, Julian Georg, Srivastava, Rupesh Kumar, Koutník, Jan, and Schmidhuber, Jürgen. Recurrent highway networks. In *International Conference on Machine Learning*, 2017.