RNNs WITH PRIVATE AND SHARED REPRESENTATIONS FOR SEMI-SUPERVISED SEQUENCE LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Training recurrent neural networks (RNNs) on long sequences using backpropagation through time (BPTT) remains a fundamental challenge. It has been shown that adding a local unsupervised loss term into the optimization objective makes the training of RNNs on long sequences more effective. While the importance of an unsupervised task can in principle be controlled by a coefficient in the objective function, the gradients with respect to the unsupervised loss term still influence all the hidden state dimensions, which might cause important information about the supervised task to be degraded or erased. Compared to existing semi-supervised sequence learning methods, this paper focuses upon a traditionally overlooked mechanism – an architecture with explicitly designed *private and shared hidden units* designed to mitigate the detrimental influence of the auxiliary unsupervised loss over the main supervised task. We achieve this by dividing RNN hidden space into a private space for the supervised task or a shared space for both the supervised and unsupervised tasks. We present extensive experiments with the proposed framework on several long sequence modeling benchmark datasets. Results indicate that the proposed framework can yield performance gains in RNN models where long term dependencies are notoriously challenging to deal with.

1 INTRODUCTION

Recurrent neural networks (RNNs) are widely considered the de facto tool for modeling sequences with a deep learning approach. Training RNNs usually relies on the use of backpropagation through time (BPTT). It is well known that unfortunately, it becomes difficult for BPTT to transmit gradients through very long computational graphs, as gradients tend to explode or vanish (Hochreiter et al., 2001). Figure 1-(a) gives an example in an oversimplified setting, where the hidden state at the first time-step does not receive gradients. To make the BPTT-based training more effective, architectures such as the long short-term memory (LSTM) (Hochreiter & Schmidhuber, 1997) RNN and gated recurrent unit (GRU) (Cho et al., 2014) RNNs, use parameterized gates which can make gradient flow more effective over long sequences.

Recently, strong evidence in (Trinh et al., 2018) suggests that simultaneously learning supervised and unsupervised tasks can also enhance an RNN's ability to capture long-term dependencies. By injecting unsupervised tasks locally along the sequence the unsupervised tasks can be harnessed to provide local and reliable gradients to more effectively optimize RNNs for long sequence learning tasks. Recent work using this strategy (Peters et al., 2018; Dai & Le, 2015; Trinh et al., 2018), could be characterized as employing semi-supervised learning architectures consisting of two distinct types of RNNs, one for the primary supervised task and another for the auxiliary unsupervised tasks which are injected locally along the sequence. More concretely, the RNN for an unsupervised task updates is instantiated periodically along the sequence and its hidden states are reinitialized occasionally; whereas, the RNN for the supervised tasks operates at every time-step. Figure 1-(b) shows how gradients flow in this architecture.

Despite the ability of these new semi-supervised architectures to mitigate the problem of long-distance BPTT, these approaches risk impairing the training of the main task by contaminating the entire representation-space with the unsupervised loss gradients.The challenge we address here is how to properly coordinate supervised and unsupervised tasks. Common wisdom for semi-supervised learning (Peters et al., 2018) typically follows one of the two procedures discussed below. The

first widely used approach is to weight supervised and unsupervised loss functions with varying coefficients empirically. However this method cannot radically address aforementioned problem since representations for supervised and unsupervised learning are still entangled in same space. It is true that the contribution of the unsupervised task can in principle be controlled by a coefficient in the objective function, but the gradients with respect to the unsupervised loss term still influence all the hidden state dimensions, which might cause important information about the supervised task to be erased accidentally. The second approach coordinates these two types of learning by specifying a training order and separating them into different learning phases. For example, these approaches usually first pre-train a model under unsupervised setting, then use the model for supervised learning (Radford et al., 2018).

While these methods can provide rich auxiliary knowledge which are potentially useful for the main task, there is no guarantee that this asynchronous learning fashion could let the main task utilize the auxiliary information well, and therefore long-term dependencies are still difficult to capture. It is thus crucial to ask: how exactly can auxiliary unsupervised tasks best serve the main supervised learning task for long-term dependencies learning?

On the other hand, it has been demonstrated that dividing an RNN's representational space into different groups is useful for modeling long-term dependencies. One such example is clockwork RNNs (Koutnik et al., 2014), where each group is responsible for a subset of hidden states and each processes input at different clock speeds. It is also possible to let each layer represent a group, and each group may run at different time scales (Schmidhuber, 1992; Chung et al., 2016).

With the above analysis in mind, we propose to solve the long-term dependency problem by enabling the two RNNs to have a shared feature space for both supervised and unsupervised tasks, and allowing an RNN to have a private space dedicated for the supervised task. The key insight is to associate different time-scale updating operations of distinct RNNs with different representational spaces. Through the shared feature space, the RNNs form an interface to exchange features useful for both of them with less inference. As a side-product, the proposed variant of RNNs trains and evaluates slightly faster since the architecture by design introduced an inductive bias that the modules for auxiliary tasks should have less parameters. Figure 1-(c) shows how the gradients flow through the hidden states during the backward pass of BPTT for the proposed architecture. It is clear that the lower (blue) space is not allowed to receive gradients from the unsupervised task.

Our primary contribution is introducing a private-and-shared feature space architecture for semi-supervised sequential learning tasks, which is motivated through the lens of gradient flows. While the modification is simple, its application on modeling long-term dependencies has shown significant improvement over other state-of-the-art algorithms to our knowledge, and thus we believe it will be of broad interest to the community. In Section 3, we describe the proposed method. In section 4, we present the experiments. In section 5, we give an analysis of our method and experiments.

2 Related work

(Bai et al., 2018) show that a generic temporal convolutional network (TCN) outperforms some RNN variants on several benchmark datasets. However, compared with TCNs, RNNs require lower memory for inference and can handle potential parameter change for a transfer of domain (Bai et al., 2018). Furthermore, (Trinh et al., 2018) show that RNNs with an auxiliary unsupervised loss still outperform TCNs in terms of accuracy on long sequence learning tasks. More importantly, RNNs can model, in principle, infinitely long dependencies with a finite number of parameters.

Unsupervised learning is often introduced in a pre-training fashion. For example, (Radford et al., 2018) show that for natural language understanding tasks, generative pre-training of a language model on a large amount of unlabeled text followed by discriminative fine-tuning leads to significant improvement. As another example, in (Peters et al., 2018), after pretraining the model with unlabeled data, the authors fix the weights and add additional task-specific model capacity, making it possible to leverage large, rich and universal representations for the downstream tasks. It should be noted that (Peters et al., 2018) utilizes additional datasets, whereas we do not.

(Trinh et al., 2018) propose RNN-AE (AutoEncoder) to form an auxiliary unsupervised task to aid RNNs in handling long sequencse, i.e. r-RNN (reconstruction) and p-RNN (prediction). The r-RNN approach tries to reconstruct the original input from the internal representation.

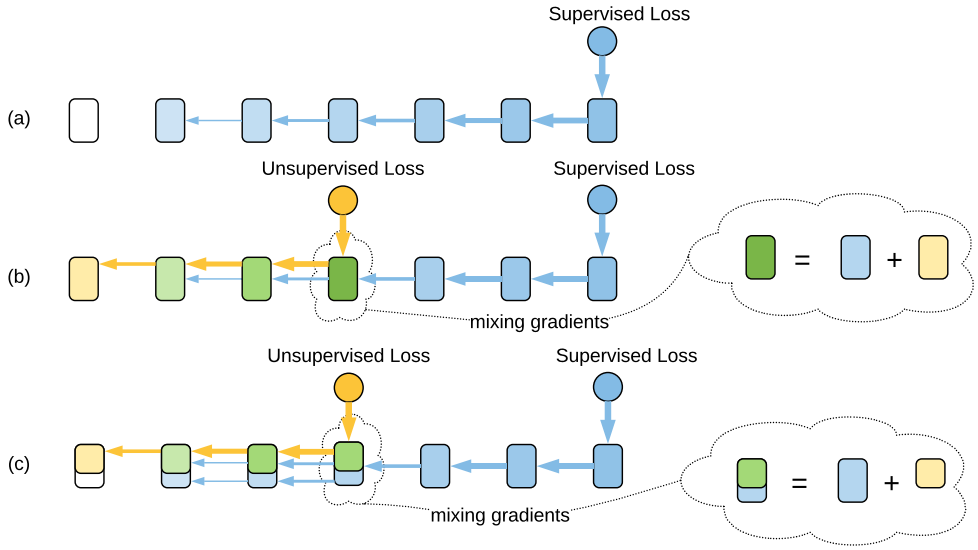


Figure 1: Schematic about gradient flows of different architectures during the backward pass. Each cube indicates a set of hidden states at a time-step. Shading means the signal strength of gradients – a darker color is associated with more reliable gradients. Arrows indicate gradient flows, and those thicker ones represent more reliable gradients. Blue filling indicates gradients w.r.t. the supervised loss. Yellow filling indicates gradients w.r.t. the unsupervised loss. Green filling represents a mixture of the supervised and unsupervised gradients. White filling means there is no gradient flowing through the hidden states. (a) Gradient flows of a typical RNN with a supervised loss. (b) Gradient flows of a semi-supervised RNN (Trinh et al., 2018). (c) Gradient flows of our proposed method, which consists of a shared and private spaces.

Skim-RNN (Seo et al., 2017) dynamically decides to update only a small fraction of the hidden state for relatively unimportant input tokens. A skim-RNN contains multiple RNNs that have a shared space and also have private spaces on which they operate. However a skim-RNN only considers supervised tasks and aims at accelerating the inference speed. In contrast, our method is specifically designed for long sequence learning problems with an unsupervised loss. Furthermore, a skim-RNN only uses one RNN at each time-step which is determined by a reinforcement learning agent, whereas ours always use multiple RNNs. As a side-effect of not relying on reinforcement learning algorithms, our method is easier to train in practice.

The hidden state of our proposed RNN also has multiple subsets, but they run at the same clock speed. Even more importantly, we introduce an inductive bias that different hidden sub-spaces should be responsible for different tasks.

3 Methodology

We will discuss these methods in detail. We first briefly explain our version of the RNN-AE and its key differences with that introduced by (Trinh et al., 2018); then, we dive into our method of inducing a private-and-shared structure.

3.1 Local Reconstruction and Prediction

Following (Trinh et al., 2018), we add the unsupervised tasks of local sequence reconstruction and prediction at various anchor points within the input sequence. We sample n anchors at locations $a_1, a_2, ..., a_n$ among the input sequence and at anchor a_i, we obtain an unsupervised loss. This loss is generated through local reconstruction and/or prediction within a neighbourhood of the input x_{a_i}. It uses an auxiliary RNN with GRUs, initialized by $f(\boldsymbol{h}_{a_i})$ which is a function of the hidden state at the anchor time step. In (Trinh et al., 2018), f is simply the identity function $f(x) = x$. This generates a

total unsupervised loss as the sum of auxiliary loss at each anchor location, and its gradient flows back to each anchor neighbourhood to improve long-term dependency.

There are some important differences in our implementation: Instead of randomly sampling the anchor locations over the entire sequence, we evenly divide the input into as many sub-sequences as there are anchors and only sample each anchor within its corresponding region. This is to ensure that reconstruction spans most if not all of the input sequence; this way, gradient flows back to a higher percentage of the input. During reconstruction, we ask the r-RNN to reconstruct the local sequence backward (as we found that backwards reconstruction worked better in practice). Furthermore, in various task setting, we include both unsupervised prediction and reconstruction instead of using only one of the two. And lastly, and most importantly, rather than using the entire hidden state of the anchor to do reconstruction and prediction, we only use a part of the state vector for these unsupervised tasks; this is the point we expand on next.

3.2 Private-and-Shared Structure

We propose to divide the hidden space of the main RNN into task-specific sub-spaces. More concretely, ignoring the time step suffix, we take the d-dimension state vector $\boldsymbol{h} = [h_1, h_2, ..., h_d]^\intercal$ and split it into $\boldsymbol{h}^s$ and $\boldsymbol{h}^p$ (shared and private state) where $\boldsymbol{h}^s = [h_1, h_2, ..., h_r]^\intercal$ and $\boldsymbol{h}^p = [h_{r+1}, ..., h_d]^\intercal$; aka. $\boldsymbol{h} = [\boldsymbol{h}^{s\intercal}||\boldsymbol{h}^{p\intercal}]^\intercal$ where $[\boldsymbol{a}||\boldsymbol{b}]$ denotes in the concatenation of $\boldsymbol{a}$ and $\boldsymbol{b}$. At each anchor step, when the main RNN state vector at that step is needed to initialize the auxiliary RNN, only $\boldsymbol{h}^s$ is sent, and $\boldsymbol{h}^p$ remains untouched. On the other hand, at the terminal time step, we use the entire state vector $\boldsymbol{h}$ to predict a class label. In this sense, $\boldsymbol{h}^s$ is the section of state vector that is shared by the reconstruction task and the final classification; $\boldsymbol{h}^p$ is private in the sense that it is the section of state vector that is reserved for only the supervised task.

The intuition here is that we want to disentangle the feature space so the learned features from the unsupervised tasks do not affect the entire state vector which is later used for the supervised task. In doing so, we create an uncontaminated region learned solely through the supervised task; naturally, it would capture more specific features for the supervised tasks than the shared region that's used both for the unsupervised and supervised task.

This way, we overcome the negative side effects of RNN-AE while retaining its ability to introduce gradient at all time steps. Thus we are able to facilitate the learning of long-term dependency without hindering the model's ability to perform supervised task.

4 Experiments

The experiments are designed to answer a key question: Since we divide hidden states into sub-spaces, compared with RNNs with a holistic hidden space, is our proposed method indeed more effective?

We evaluate our proposed methods on two benchmark tasks: image classification and sentiment analysis. For image classification, we used pixel-level sequential MNIST, Fashion-MNIST and CIFAR-10 as our datasets. For sentiment analysis, we use the IMDB movie reviews corpus. Detailed information about the datasets is given in Table 1.

In order to compare with state-of-the-art results fairly, we re-implement and re-run all the baseline methods. For all of our experiments, including the baseline and re-implementations, we grid search our hyperparameters using a validation set to find the optimal values. For faster convergence, we adopt SGDR (Loshchilov & Hutter, 2016) as the optimizer throughout this paper. We incorporate early stopping with patience of 50 epochs to avoid overfitting, which is also based on a validation set.

As shown in Table 3, our proposed method achieves better outcome on both MNIST and CIFAR-10 than previous competitive results. Table 2 provides a more comprehensive list of the experiments along with the hyperparameters used for each one. The top row(s) of each sections, the ones without parameters, are baseline results run with GRU. As the share proportion parameter reaches 100%, our model reduce to the RNN-AE model introduced in (Trinh et al., 2018), thus the rows with "100%" as the value for the "shared" hyperparameter are results for (Trinh et al., 2018). As previously mentioned, the hyperparameters for these cases are also grid searched to ensure fair comparison.

Dataset	Mean Length	# Classes	Train Set Size	Valid. Set Size
Sequential MNIST	784	10	50K	10K
Sequential CIFAR10	1024	10	45K	15K
Sequential Fashion-MNIST	784	10	50K	10K
IMDB	1430[a]	2	20K	2.2K

Table 1: Key statistics of the datasets used in this paper.

[a]We use one-hot character-level embedding and remove the inputs with length less than 500.

Tasks	Accuracy	Unsup. Type		Sharing Schema		Sampling Opts.		#Param.
		Pred.	Rec.	Shared	Private	#Anchor	Rec Len.	
MNIST	98.5%			-	-	-	-	90K
	98.8%		✓	100%	0%	1	50	1M
	98.5%		✓	100%	0%	20	30	1M
	98.8%		✓	30%	70%	1	50	76K
	98.9%	✓	✓	30%	70%	40	20	84K
F-MNIST	90.5%			-	-	-	-	11K
	90.4%		✓	100%	0%	5	32	12K
	90.8%		✓	60%	40%	20	30	12K
	90.4%	✓	✓	50%	50%	5	30	11K
CIFAR10	70.8%			-	-	-	-	3.1M
	68.9%			-	-	-	-	5.4M
	73.5%		✓	100%	0%	20	30	5.4M
	75.1%		✓	40%	60%	20	30	3.9M
	76.2%	✓	✓	60%	40%	10	64	4.7M
IMDB	84.4%			-	-	-	-	95K
	84.8%		✓	100%	0%	1	50	69K
	85.1%		✓	60%	40%	1	64	59K
	85.7%		✓	60%	40%	5	50	75K

Table 2: Performances of our models on MNIST, fashion-MNIST, CIFAR-10, and IMDB datasets. Note that when the shared proportion reaches 100%, it reduces to the model proposed in (Trinh et al., 2018).

In our experiments, we are able to achieve better accuracy with either less or comparable number of parameters. In the case of MNIST, we are able to do so with less than one-tenth of the parameters compared to (Trinh et al., 2018).

Overall, from these tables, we can study the effect of key hyperparameters such as private-vs-shared proportion, number of anchors and reconstruction length. we see how entirely sharing the hidden space is not optimal, which leads us to ask the follow up question: how much of the space should be shared? In the next section, we decribe our analysis on this very inquiry. 5.1.

5 Analysis

In this section, we present two crucial analyses on the private-shared structure. First, we examine how shared proportion affects our model outcomes. Then, we visualize both the private and shared hidden space learned by our model to highlight distinct features of the two.

	MNIST	CIFAR-10
iRNN (Le et al., 2015)	97%	N/A
uRNN (Arjovsky et al., 2015)	95.1%	N/A
RNN with Aux. Loss (Trinh et al., 2018)	98.4%	72.2%
RNN with private-shared space	**98.9%**	**76.2%**

Table 3: Comparing test accuracy on sequential MNIST and sequential CIFAR10.

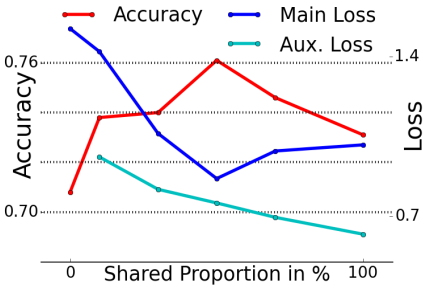


Figure 2: Accuracy (left axis) and Loss (right axis) vs Shared Proportion. We visualize the results from Table 4

Acc.	L Main	L Aux.	S.%	# Param.
70.8	1.523	-	0%	3.1M
73.8	1.422	0.960	10%	3.3M
74.0	1.063	0.818	30%	4M
76.1	**0.865**	0.758	50%	5.4M
74.6	0.986	0.695	70%	7.3M
73.1	1.013	**0.621**	100%	11M

Table 4: Experiments on shared proportions (S.%) performed on CIFAR. Accuracy is reported in percentage. L Main and L Aux. are the main loss and auxiliary loss respectively.

5.1 Effect of Shared Proportion

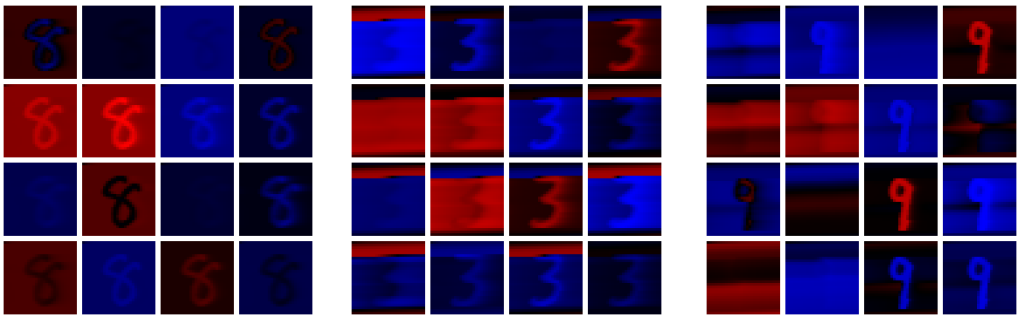
To observe exactly what effect different sharing proportions have on classification tasks, we run our model on CIFAR-10 with varying share percentages from 0% to 100% as listed in Table 4. The results are given in Figure 2.

With 0% shared, the model is essentially an RNN without unsupervised loss as no part of the hidden state is sent to the auxiliary RNN. In this setting, our model achieves the lowest accuracy of the group. As we slowly increase the shared percentage, the accuracy rises. From this, we confirm that the addition of auxiliary loss is indeed helpful in modeling long-term dependencies.

Remarkably, there is a pronounced peak at the 50% mark – where half of the state is shared by the auxiliary RNN and half is reserved for the classification task – after which, model performance, in terms of both cross entropy loss and classification accuracy begins to worsen. One may suspect that this is caused by overfitting since the number of parameters steadily increases. However, in this set of experiments, the capacity of the main RNN is fixed as the dimension of the hidden state remains the same. As we include more of the hidden state for unsupervised task, the auxiliary RNN increases in capacity, thus causing the overall parameter number to increase. Furthermore, the auxiliary test loss lowers progressively, suggesting that there is no overfitting in the auxiliary RNN either. In this case, the change in performance should attributed to the difference in shared proportion. This finding agrees with what we posited earlier: that the learned knowledge from the unsupervised task is not all helpful towards the supervised task, and that this information may actually hurt performance by overwriting knowledge that are important to the main task. What is somewhat surprising is that at 100% shared, the model's classification result is barely comparable to the 10% version. We did not expect such big drop in classification rate ($\sim$3%) at the maximum shared level. However, this further corroborates the importance of disentangling the hidden space through a private-shared framework.

5.2 Shared and Private Spaces

To better understand how the use of our framework influences representation learning, we analyze the hidden vectors of the main RNN. For this analysis, we retrain our MNIST model with 50% shared proportion and a hidden space size of only 16. In other words, the first 8 dimensions are shared,



(a) Hidden state response to an "8" at beginning of training

(b) Hidden state response to a "3" after 5 epochs

(c) Hidden state response to a "9" after 25 epochs

Figure 3: Hidden state evolution during training at different epochs.

and the last 8 are private. We hypothesize that the small representation-space would force the RNN to learn important features; it would also allow us to see how knowledge from the two tasks might contend with each other if there existed any competition.

To visualize the hidden space, we collect the RNN's hidden state vectors $\{\boldsymbol{h}_t \in \mathbb{R}^{d\times 1} | \ 1 \leq t \leq 784\}$ after it receives an MNIST image (784 pixels) as input. We concatenate them horizontally to obtain $\boldsymbol{H} = [\boldsymbol{h}_1||\boldsymbol{h}_2||...||\boldsymbol{h}_{784}] \in \mathbb{R}^{d\times 784}$. Then, we look at how the n^{th} dimension of the hidden state changes across time by picking out the n^{th} row of $\boldsymbol{H}$. We reshape each row into a 28×28 image to better compare the state elements.

Figure 3 shows 3 instances of visualization of the hidden state vector across time, which are generated using the same network and different input image at different stages of training. Respectively from left to right, the images are generated when training begins, progresses, and converges. Each small square in the image corresponds to a single dimension of the hidden state from $t = 1$ to $t = 784$. In this setting, the left 2 columns of each subplot contain the 8 shared dimensions and the right 2 columns corresponds to the 8 private dimensions.

Noticeably, as training progresses the hidden state changes how it responds to input. Moreover, there are dimensions in the shared region of the hidden state that originally display a response similar to those in the private region, only to be replaced later. For example, in the beginning of training, there are several similar responses in multiple dimensions of the state vector. In both the designated shared and private region, some dimensions seem to have a high correlation with the input image thus resulting in a readable number. As the model refines itself, the aforementioned response begins to fade from the shared representation; however, it still exists in the private representation. In the end, we see the a complete absence of such response in the shared representation, replaced by more abstract features.

One possible explanation would be that this type of response is more important to the supervised task than to the unsupervised one. By generating a large non-zero value when the input is non-zero, a dimension of the hidden state allows the RNN to propagate this information along to later times steps. This could be very helpful for digit classification, as we might need to know what is written in the beginning to decide which digit it is. However, it is not required for the auxiliary task which concerns with only local pixels and has less need to propagate a strong signal when a particular input is given.

6 Conclusion

In this paper, we have presented a semi-supervised RNN architecture with explicitly designed private and shared hidden representations. This architecture allows information transfer between the supervised and unsupervised tasks in a hitherto unexplored way. Compared with other similar semi-supervised RNN techniques, our experiments on widely used and competitive benchmark data sets suggest that our formulation indeed yields performance gains. We conjecture that these gains come from the desirable properties of both gradient and information flow in architectures with shared and private representations. As a side-product, our proposed architecture trains and evaluates faster

than the related alternatives that we have explored since the architecture introduces an inductive bias that the modules for auxiliary tasks should have fewer parameters.

REFERENCES

Martín Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *CoRR*, abs/1511.06464, 2015. URL http://arxiv.org/abs/1511.06464.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv:1803.01271*, 2018.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

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

Andrew M Dai and Quoc V Le. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, pp. 3079–3087, 2015.

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

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941, 2015. URL http://arxiv.org/abs/1504.00941.

Ilya Loshchilov and Frank Hutter. Sgdr: stochastic gradient descent with restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

Minjoon Seo, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. Neural speed reading via skim-rnn. *arXiv preprint arXiv:1711.02085*, 2017.

Trieu H Trinh, Andrew M Dai, Minh-Thang Luong, and Quoc V Le. Learning longer-term dependencies in rnns with auxiliary losses. 2018.