Recurrent Batch Normalization

Tim Cooijmans **Nicolas Ballas** **César Laurent**

Aaron Courville
MILA - Université de Montréal
firstname.lastname@umontreal.ca

Abstract

We propose a reparameterization of LSTM that brings the benefits of batch normalization to recurrent neural networks. Whereas previous works only apply batch normalization to the input-to-hidden transformation of RNNs, we demonstrate that it is both possible and beneficial to batch-normalize the hidden-to-hidden transition, thereby reducing internal covariate shift between time steps.

We evaluate our proposal on various sequential problems such as sequence classification, language modeling and question answering. Our empirical results show that our batch-normalized LSTM consistently leads to faster convergence and improved generalization.

1 Introduction

Recurrent neural network architectures such as LSTM [10] and GRU [6] have recently exhibited state-of-the-art performance on a wide range of complex sequential problems including speech recognition [1], machine translation [3] and image and video captioning [26, 27]. Top-performing models, however, are based on very high-capacity networks that are computationally intensive and costly to train. Effective optimization of recurrent neural networks is an active area of study [22, 18, 20].

It is known that for deep feed-forward neural networks, covariate shift [23, 11] degrades the efficiency of training. Covariate shift is a change in distribution of the inputs to a model. This occurs continuously during training of feed-forward neural networks, where changing the parameters of a layer affects the distribution of the inputs to all layers above it. As a result, the upper layers are continually adapting to the shifting input distribution and unable to learn effectively. This *internal* covariate shift [11] may play an especially important role in recurrent neural networks, which resemble very deep feed-forward networks.

Batch normalization [11] is a recently proposed technique for controlling the distributions of feed-forward neural network activations, thereby reducing internal covariate shift. It involves standardizing the activations going into each layer, enforcing their means and variances to be invariant to changes in the parameters of the underlying layers. This effectively decouples each layer's parameters from those of other layers, leading to a better-conditioned optimization problem. Indeed, deep neural networks trained with batch normalization converge significantly faster and generalize better.

Although batch normalization has demonstrated significant training speed-ups and generalization benefits in feedforward networks, it has proven difficult to apply in recurrent architectures [14, 1]. It has found limited use in stacked RNNs, where the normalization is applied "vertically", i.e. to the input of each RNN, but not "horizontally" between timesteps. RNNs are deepest in the time direction, and as such batch normalization would be most beneficial when applied horizontally. However, it has been hypothesized [14] that applying batch normalization in this way hurts training because of exploding gradients due to repeated rescaling.

Our findings run counter to this hypothesis. We show that it is both possible and highly beneficial to apply batch normalization in the hidden-to-hidden transition of recurrent models. In particular, we describe a re-parameterization of LSTM that involves batch normalization and demonstrate that doing so speeds up optimization and improves generalization. In addition, we empirically analyze the gradient backpropagation and show that proper initialization of the batch normalization parameters is crucial to avoiding vanishing gradient.

We evaluate our proposal on several sequential problems and show that our LSTM reparameterization consistently outperforms the LSTM baseline across tasks, in terms of both time to convergence and performance. In particular, we achieve state-of-the-art performance on Sequential MNIST [15] and character-level language modeling [19] on Penn Treebank [17] and text8 [16].

Section 2 introduces RNNs and batch normalization in detail. In Section 3, we discuss batch normalization in the recurrent setting. Section 4 investigates the impact of the batch normalization scale parameter on the gradient back-propagation for recurrent models . We show in Section 5 our evaluations of the proposed re-parameterization on a variety of tasks.

2 Prerequisites

2.1 LSTM

Long Short-Term Memory (LSTM) networks are an instance of a more general class of recurrent neural networks (RNNs), which we shall briefly review. Given an input sequence $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ...\mathbf{x}_T)$, an RNN defines a sequence of hidden states h_t according to

$$\mathbf{h}_t = \phi(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}), \tag{1}$$

where $\mathbf{W}_h \in \mathbb{R}^{d_h \times d_h}, \mathbf{W}_x \in \mathbb{R}^{d_x \times d_h}, \mathbf{b} \in \mathbb{R}^{d_h}$ and the initial state $\mathbf{h}_0 \in \mathbb{R}^{d_h}$ are model parameters. A popular choice for the activation function ϕ is $\tanh$.

RNNs are popular in sequence modeling because of their natural ability to process variable-length sequences. Training RNNs using first-order stochastic gradient descent (SGD) however is notoriously difficult due to the well-known problem of exploding/vanishing gradients [5, 9, 22]. Gradient vanishing occurs when states h_t are not influenced by small changes in much earlier states h_τ, $t \ll \tau$, preventing learning of long-term dependencies in the input data. While the long-term dependencies problem is unsolvable in absolute [5], it has been demonstrated that different RNN re-parameterizations, such as the LSTM [10], GRU [6] or i/u-RNN [15, 2] can help mitigate the vanishing gradient problem.

In what follows, we focus on the LSTM architecture [10] with recurrent transition given by

$$\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} = \mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b} \tag{2}$$

$$\mathbf{c}_t = \sigma(\tilde{\mathbf{f}}_t) \odot \mathbf{c}_{t-1} + \sigma(\tilde{\mathbf{i}}_t) \odot \tanh(\tilde{\mathbf{g}}_t) \tag{3}$$

$$\mathbf{h}_t = \sigma(\tilde{\mathbf{o}}_t) \odot \tanh(\mathbf{c}_t), \tag{4}$$

where $\mathbf{W}_h \in \mathbb{R}^{d_h \times 4d_h}, \mathbf{W}_x \mathbb{R}^{d_x \times 4d_h}, \mathbf{b} \in \mathbb{R}^{4d_h}$ and the initial states $\mathbf{h}_0 \in \mathbb{R}^{d_h}, \mathbf{c}_0 \in \mathbb{R}^{d_h}$ are model parameters. σ is the logistic sigmoid function, and the $\odot$ operator denotes the Hadamard product.

The LSTM differs from simple RNNs in that it has an additional memory *cell* $\mathbf{c}_t$ whose update is nearly linear which allows the gradient to flow back through time more easily. In addition, unlike the RNN which overwrites its content at each time-step, the update of the LSTM cell is regulated by a set of gates. The forget gate $\mathbf{f}_t$ determines the extent to which information is carried over from the previous time-step, and the input gate $\mathbf{i}_t$ controls the flow of information from the current input $\mathbf{x}_t$. The output gate $\mathbf{o}_t$ allows the model to read from the cell. This carefully controlled interaction with the cell is what allows the LSTM to robustly retain information for long periods of time.

2.2 Batch Normalization

Covariate shift [23] is a phenomenon in machine learning where the features presented to a model change in distribution during the course of training. In order for learning to succeed in the presence of covariance shift, the model's parameters must be adjusted not just to learn the concept at hand but also to adapt to the changing distribution of the inputs. In deep neural networks, this problem manifests as *internal covariance shift* [11], where changing the parameters of a layer affects the distribution of the inputs to all layers above it.

Batch Normalization [11] is a recently proposed network re-parameterization that aims to reduce internal covariate shift. It does so by standardizing the activations using statistical estimates of their means and standard deviations. However, it does not decorrelate the activations as the matrix inversion involved would be too expensive.

The batch normalizing transform is as follows:

$$\mathrm{BN}(\mathbf{h};\gamma,\beta) = \beta + \gamma \frac{\mathbf{h} - \widehat{\mathbb{E}}(\mathbf{h})}{\sqrt{\widehat{\mathrm{Var}}(\mathbf{h}) + \epsilon}} \tag{5}$$

where $\mathbf{h} \in \mathbb{R}^d$ is the vector of (pre)activations to be normalized, $\gamma \in \mathbb{R}^d, \beta \in \mathbb{R}^d$ are model parameters that determine the mean and standard deviation of the normalized activation, and $\epsilon \in \mathbb{R}$ is a regularization hyperparameter. The division should be understood to proceed elementwise.

At training time, the statistics $\mathbb{E}(\mathbf{h})$ and $\mathrm{Var}(\mathbf{h})$ are estimated by the sample mean and sample variance of the current minibatch. This allows for backpropagation through the statistics, preserving the convergence properties of stochastic gradient descent. During inference, the statistics are typically estimated based on the entire training set, so as to produce a deterministic prediction.

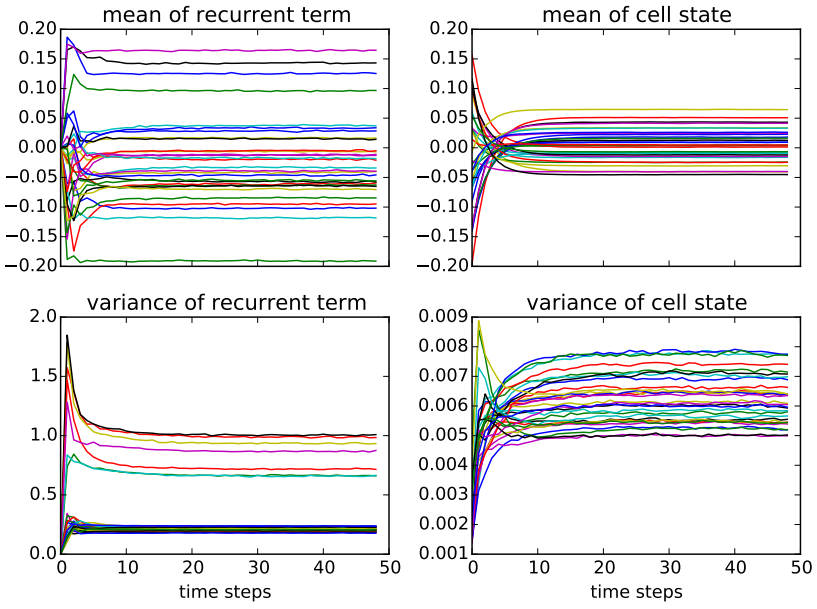


Figure 1: Convergence of population statistics to stationary distributions on the Penn Treebank task. Only a random subset of population statistics is shown.

3 Batch-Normalized LSTM

This section introduces a re-parameterization of LSTM that takes advantage of batch normalization. Contrary to previous work [14, 1], we leverage batch normalization in both the input-to-hidden *and*

the hidden-to-hidden transformations. We introduce the batch-normalizing transform $\mathrm{BN}(\,\cdot\,;\gamma,\beta)$ into the LSTM as follows:

$$\begin{pmatrix} \tilde{\mathbf{f}}_t \\ \tilde{\mathbf{i}}_t \\ \tilde{\mathbf{o}}_t \\ \tilde{\mathbf{g}}_t \end{pmatrix} = \mathrm{BN}(\mathbf{W}_h\mathbf{h}_{t-1};\gamma_h,\beta_h) + \mathrm{BN}(\mathbf{W}_x\mathbf{x}_t;\gamma_x,\beta_x) + \mathbf{b} \tag{6}$$

$$\mathbf{c}_t = \sigma(\tilde{\mathbf{f}}_t) \odot \mathbf{c}_{t-1} + \sigma(\tilde{\mathbf{i}}_t) \odot \tanh(\tilde{\mathbf{g}}_t) \tag{7}$$

$$\mathbf{h}_t = \sigma(\tilde{\mathbf{o}}_t) \odot \tanh(\mathrm{BN}(\mathbf{c}_t;\gamma_c,\beta_c)) \tag{8}$$

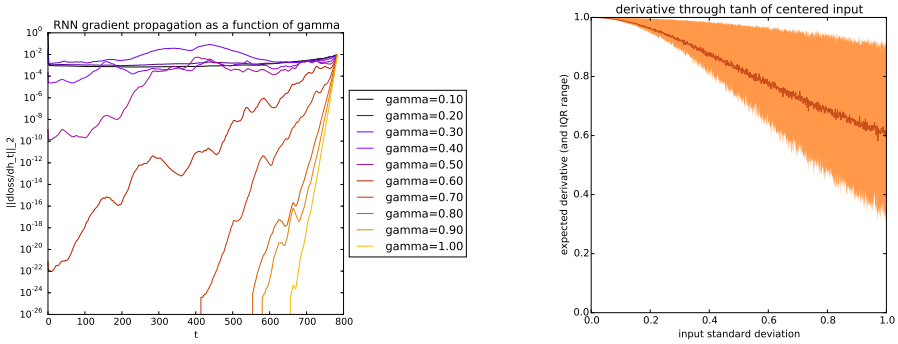
In our formulation, we normalize the recurrent term $\mathbf{W}_h\mathbf{h}_{t-1}$ and the input term $\mathbf{W}_x\mathbf{x}_t$ separately. Normalizing these terms individually gives the model better control over the relative contribution of the terms using the γ_h and γ_x parameters. We set $\beta_h = \beta_x = \mathbf{0}$ to avoid unnecessary redundancy, instead relying on the pre-existing parameter vector $\mathbf{b}$ to account for both biases. In order to leave the LSTM dynamics intact and preserve the gradient flow through $\mathbf{c}_t$, we do not apply batch normalization on the cell states.

The batch normalization transform relies on batch statistics to standardize the LSTM activations. It would seem natural to share the statistics that are used for normalization across time, just as recurrent neural networks share their parameters over time. However, we have found that simply averaging statistics over time severely degrades performance. Although LSTM activations do converge to a stationary distribution, we have empirically observed that their statistics during the initial transient differ significantly as figure 1 shows. Consequently, we recommend using separate statistics for each timestep to preserve information of the initial transient phase in the activations.

Generalizing the model to sequences longer than those seen during training is straightforward thanks to the rapid convergence of the activations to their steady-state distributions (cf. figure 1). For our experiments we estimate the population statistics separately for each timestep $1,\ldots,T_{max}$ where T_{max} is the length of the longest training sequence. When at test time we need to generalize beyond T_{max}, we use the population statistic of time T_{max} for all time steps beyond it.

During training we estimate the statistics across the minibatch, independently for each timestep. At test time we use estimates obtained by averaging the minibatch estimates over the training set.

4 Initializing γ for Gradient Flow



(a) Gradient flow through a batch-normalized $\tanh$ RNN as a function of γ. High variance causes vanishing gradient.

(b) Empirical expected derivative of $\tanh$ nonlinearity as a function of input variance. High variance causes saturation, which decreases the expected derivative.

Figure 2: Influence of pre-activation variance on gradient propagation.

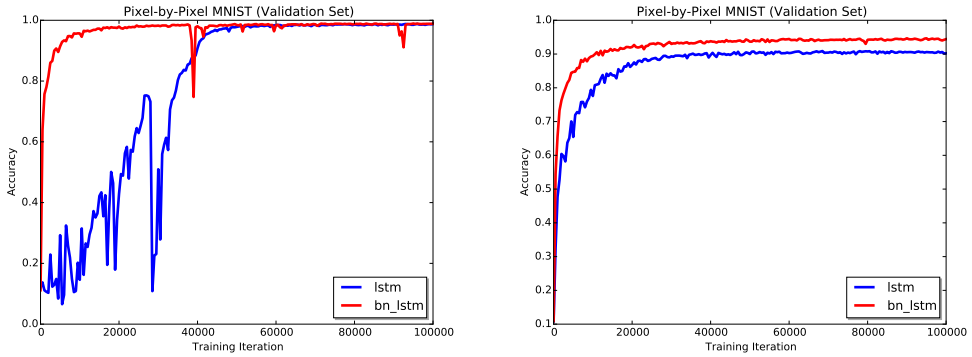


Figure 3: Accuracy on the validation set for the pixel by pixel MNIST classification tasks. The batch-normalized LSTM is able to converge faster relatively to a baseline LSTM. Batch-normalized LSTM also shows some improve generalization on the permuted sequential MNIST that require to preserve long-term memory information.

Although batch normalization allows for easy control of the pre-activation variance through the γ parameters, common practice is to normalize to unit variance. We suspect that the previous difficulties with recurrent batch normalization reported in the literature [14, 1] are largely due to improper initialization of the batch normalization parameters, and γ in particular. In this section we demonstrate the impact of γ on gradient flow.

In Figure 2(a) we show how the pre-activation variance impacts gradient propagation in a simple RNN on the sequential MNIST task described in Section 5.1. Since backpropagation operates in reverse, the plot is best read from right to left. The quantity plotted is the norm of the gradient of the loss with respect to the hidden state at different time steps. For large values of γ, the norm quickly goes to zero as gradient is propagated back in time. For small values of γ the norm is nearly constant.

Figure 2(b) shows empirically how the expected derivative of the $\tanh$ nonlinearity changes with the variance of the argument. When the input variance is low, the input tends to be close to the origin where the derivative is close to 1. As the variance increases, the expected derivative decreases as the input is more likely to be in the saturation regime. At unit variance, the expected derivative is much smaller than 1.

We conjecture that this is what causes the gradient to vanish, and recommend initializing γ to a small value. In our trials we found that values of 0.01 or lower caused instabilities during training. Our choice of 0.1 seems to work well across tasks.

5 Experiments

This section presents an empirical evaluation of the proposed batch-normalized LSTM on four different tasks. Results show that batch-normalized LSTM achieves convergence faster than a baseline LSTM on all the tasks, and, can also lead to better generalization.

Note that for all the experiments, we initialize the batch normalization scale and shift parameters γ and β to 0.1 and 0 respectively.

5.1 Sequential MNIST

We evaluate our batch-normalized LSTM on a sequential version of the MNIST classification task [15]. The model processes each image one pixel at a time and finally predicts the label. We consider both sequential MNIST tasks, MNIST and permuted MNIST (pMNIST). In MNIST, the pixels are processed from left to right, top to bottom. In pMNIST the pixels are processed in a fixed random order.

Model	MNIST	pMNIST
TANH-RNN [15]	35.0	35.0
iRNN [15]	97.0	82.0
uRNN [2]	95.1	91.4
sTANH-RNN [28]	98.1	94.0
LSTM (ours)	98.9	90.2
BN-LSTM (ours)	**99.0**	**95.4**

Table 1: Accuracy obtained on the test set for the pixel by pixel MNIST classification tasks

Model	Penn Treebank
HF-MRNN [19]	1.41
Norm-stabilized LSTM [13]	1.39
ME n-gram [19]	1.37
LSTM (ours)	1.38
BN-LSTM (ours)	**1.32**

Table 2: Bits-per-character on the Penn Treebank test sequence.

Model	text8
td-LSTM [28]	1.63
HF-MRNN [19]	1.54
skipping RNN [21]	1.48
BN-LSTM (ours)	**1.39**

Table 3: Bits-per-character on the text8 test sequence.

Our baseline consists of an LSTM with 100 hidden units, with a softmax classifier to produce a prediction from the final hidden state. We use orthogonal initialization for all weight matrices, except for the hidden-to-hidden weight matrix which we initialize to be the identity matrix, as this yields better generalization performance on this task for both models. The model is trained using RMSProp [24] with learning rate of 10^{-3} and 0.9 momentum. We apply gradient clipping at 1 to avoid exploding gradients.

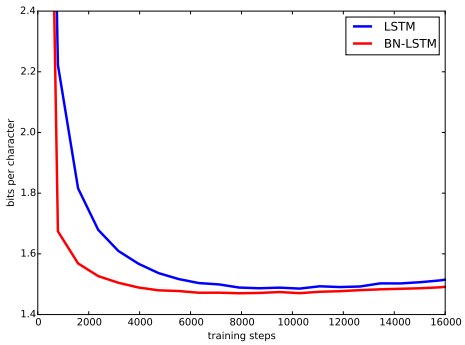
The in-order MNIST task poses a unique problem for our model: the input for the first hundred or so timesteps is constant across examples since the upper pixels are almost always black. This causes the variance of the hidden states to be exactly zero for a long period of time. Normalizing these zero-variance activations involves division by a small number at many timesteps, which causes the gradient to explode. We work around this by injecting noise into the initial hidden states. Although the normalization amplifies the noise to signal level, we find that it does not hurt performance compared to data-dependent ways of initializing the hidden states.

In Figure 3 we show the validation accuracy while training for both LSTM and batch-normalized LSTM (BN-LSTM). BN-LSTM converges faster than LSTM on both tasks. Additionally, we observe that BN-LSTM generalizes significantly better on pMNIST. It has been highlighted in [2] that pMNIST contains many longer term dependencies across pixels than in the original pixel ordering, where a lot of structure is local. A recurrent network therefore needs to characterize dependencies across varying time scales in order to solve this tasks. Our results suggest that BN-LSTM is better able to capture these long-term dependencies.

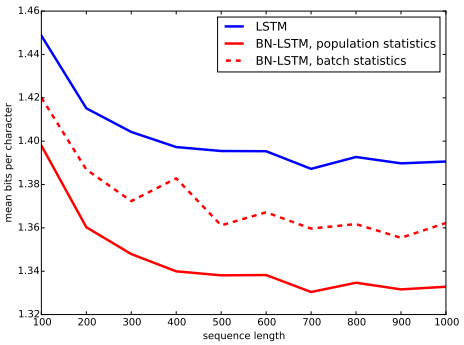
Table 1 reports the test set accuracy of the early stop model for LSTM and BN-LSTM using the population statistics. Recurrent batch normalization leads to better test score, especially for pMNIST where models have to leverage long-term temporal depencies. In addition, Table 1 shows that our batch-normalized LSTM achieves state of the art on both MNIST and pMNIST.

5.2 Character-level Penn Treebank

We evaluate our model on the task of character-level language modeling on the Penn Treebank corpus [17] according to the train/valid/test partition of [19]. For training we segment the training sequence into examples of length 100. The training sequence does not cleanly divide by 100, so for each epoch we randomly crop a subsequence that does and segment that instead.



(a) Performance in bits-per-character on length-100 subsequences of the Penn Treebank validation sequence during training.



(b) Generalization to longer subsequences of Penn Treebank using population statistics. The subsequences are taken from the test sequence.

Figure 4: Penn Treebank evaluation

Our baseline is an LSTM with 1000 units, trained to predict the next character using a softmax classifier on the hidden state h_t. We use stochastic gradient descent on minibatches of size 64, with gradient clipping at 1.0 and step rule determined by Adam [12] with learning rate 0.002. We use orthogonal initialization for all weight matrices. The setup for the batch-normalized LSTM is the same in all respects except for the introduction of batch normalization as detailed in 3.

We show the learning curves in figure 4(a). BN-LSTM converges faster and generalizes better than the LSTM baseline. Figure 4(b) shows the generalization of our model to longer sequences. We observe that using the population statistics improves generalization performance, which confirms that repeating the last population statistic (cf. section 3) is a viable strategy. In table 2 we report the performance of our best models (early stopped on validation performance) on the Penn Treebank test sequence and compare them to previous work.[1] It shows that our BN-LSTM model compares favorably to state-of-art models.

5.3 Text8

We evaluate our model on a second character-level language modeling task on the text8 dataset [16]. This dataset is derived from Wikipedia and consists of a sequence of 100M characters including only alphabetical characters and spaces. We follow previous authors [19, 28] and use the first 90M characters for training, the next 5M for validation and the final 5M characters for testing. We train on nonoverlapping sequences of length 180.

Our model is a BN-LSTM with 2000 units, trained to predict the next character using a softmax classifier on the hidden state h_t. We use stochastic gradient descent on minibatches of size 64, with gradient clipping at 1.0 and step rule determined by Adam [12] with learning rate 0.01. All weight matrices were initialized to be orthogonal.

We early-stop on validation performance and report the test performance of the resulting model in table 3. Our model achieves state of the art.

5.4 Teaching Machines to Read and Comprehend

To demonstrate the generality and practical applicability of our re-parameterization, we explore the use of batch-normalized LSTM in a unidirectional Attentive Reader Model [8].[2] We evaluate two variants. The first variant, referred to as BN-LSTM, consists of the vanilla Attentive Reader model with the LSTM simply replaced by our BN-LSTM re-parameterization. We further introduce

[1] We do not compare with [7] as they evaluate in a different setting.

[2] We make use of the existing implementation available at `https://github.com/caglar/Attentive_reader`.

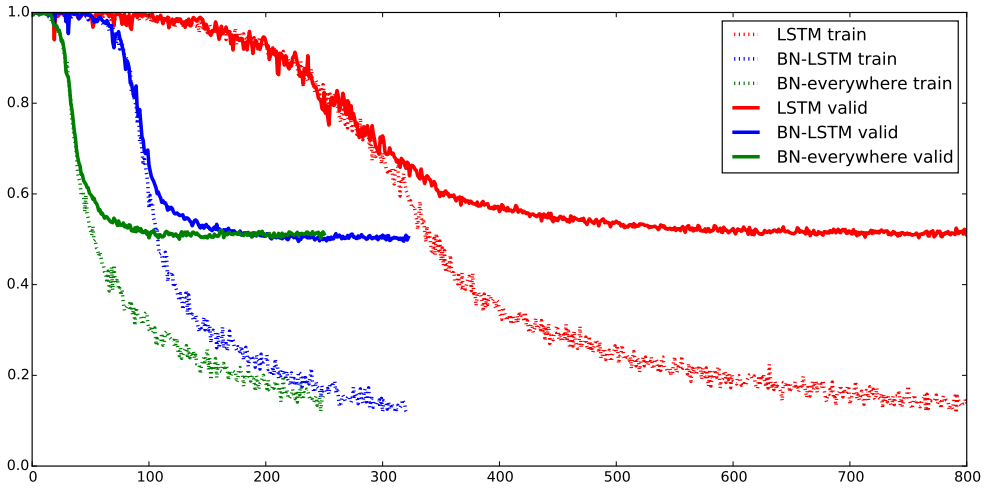


Figure 5: Error rate on the validation set for the Attentive Reader models on a variant of the CNN QA task [8]. BN-LSTM converges much faster than the baseline LSTM. Applying batch normalization in the attention computations as well ("BN-everywhere") converges faster yet. As detailed in the text, the theoretical lower bound on the error rate on this task is 43%.

batch normalization into the attention computations, normalizing each term going into the $\tanh$ nonlinearities, to obtain another variant which we term BN-everywhere.

All three variants are trained using the exact same hyperparameters. The hidden state is composed of 240 units. We use stochastic gradient descent on minibatches of size 64, with gradient clipping at 10 and step rule determined by Adam [12] with learning rate 8×10^{-5}.

We evaluate the Attentive Reader Model along with our batch-normalized variants on the question answering task using the CNN corpus, with placeholders to replace the named entities. We follow a similar preprocessing pipeline to [8]. However contrary to [8], we limit the number of sentences in each passage to 4 in order to save computation. We choose which sentences to keep by performing a string matching between the question and the passage. We rank the sentences in the text according to the string matching score and use only the top 4 sentences for each passage in the dataset. The training, validation and test sets are all preprocessed using this same procedure. After this procedure, the validation set has a recall of 57% – the passage contains the answer in only 57% of the passage/question pairs. This puts an upper bound on the accuracy that can be achieved.

During the training, we randomly sample the examples with replacement and shuffle the order of the placeholders in each text inside the minibatch. We use the whole vocabulary for the input and the answer which consists of 65829 unique words.

Figure 5 shows the learning curves for the different variants of the attentive reader. BN-LSTM trains dramatically faster than the LSTM baseline. BN-everywhere, applying batch normalization in the attention computations as well, in turn shows a significant improvement over BN-LSTM. In addition, both BN-LSTM and BN-everywhere show a generalization benefit over the baseline. We emphasize that these are preliminary results on the validation set; full results are forthcoming.

6 Conclusion

Contrary to previous findings [14, 1], we have demonstrated that batch-normalizing the hidden states of recurrent neural networks greatly improves optimization. Indeed, doing so yields benefits similar to those of batch normalization in feed-forward neural networks: our proposed BN-LSTM trains faster and generalizes better on a variety of tasks including language modeling and question-answering. We have argued that proper initialization of the batch normalization parameters is crucial, and suggest that previous difficulties [14, 1] were due in large part to improper initialization.

Acknowledgements

The authors would like to acknowledge the support of the following agencies for research funding and computing support: NSERC, Calcul Québec, Compute Canada, the Canada Research Chairs and CIFAR. We would also like to thank the developers of Theano [4] and the Blocks and Fuel [25] libraries, for developing such powerful tools for scientific computing. We thank Çağlar Gülçehre for sharing his implementation of Attentive Reader and for helping us with experiments, and David Krueger, Saizheng Zhang, Ishmael Belghazi and Yoshua Bengio for discussions and suggestions.

References

[1] Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. *arXiv preprint arXiv:1512.02595*, 2015.

[2] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. *arXiv preprint arXiv:1511.06464*, 2015.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[5] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on*, 1994.

[6] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[7] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[8] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1684–1692, 2015.

[9] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Master's thesis, Institut fur Informatik, Technische Universitat, Munchen*, 1991.

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

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[12] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[13] David Krueger and Roland Memisevic. Regularizing rnns by stabilizing activations. *arXiv preprint arXiv:1511.08400*, 2015.

[14] César Laurent, Gabriel Pereyra, Philémon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. *arXiv preprint arXiv:1510.01378*, 2015.

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

[16] Matt Mahoney. Large text compression benchmark. *URL: http://www. mattmahoney. net/text/text. html*, 2009.

[17] Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313–330, June 1993.

[18] James Martens and Ilya Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040, 2011.

[19] Tomáš Mikolov, Ilya Sutskever, Anoop Deoras, Hai-Son Le, Stefan Kombrink, and J Cernocky. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 2012.

[20] Yann Ollivier. Persistent contextual neural networks for learning symbolic data sequences. *CoRR*, abs/1306.0514, 2013.

[21] Marius Pachitariu and Maneesh Sahani. Regularization and nonlinearities for neural language models: when are they needed? *arXiv preprint arXiv:1301.5650*, 2013.

[22] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063*, 2012.

[23] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 2000.

[24] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[25] Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. Blocks and fuel: Frameworks for deep learning. *CoRR*, abs/1506.00619, 2015.

[26] Kelvin Xu, Jimmy Ba, Ryan Kiros, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

[27] Li Yao, Atousa Torabi, Kyunghyun Cho, Nicolas Ballas, Christopher Pal, Hugo Larochelle, and Aaron Courville. Describing videos by exploiting temporal structure. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4507–4515, 2015.

[28] Saizheng Zhang, Yuhuai Wu, Tong Che, Zhouhan Lin, Roland Memisevic, Ruslan Salakhutdinov, and Yoshua Bengio. Architectural complexity measures of recurrent neural networks. *arXiv preprint arXiv:1602.08210*, 2016.