Exploring Batch Normalization in Recurrent Neural Networks

Jay Whang
Stanford SCPD
jaywhang@stanford.edu

Akihiro Matsukawa
Stanford SCPD
akihiro@stanford.edu

Abstract

Batch Normalization has been shown to have significant benefits for feed-forward networks in terms of training time and model performance. In this project, we explore the application of Batch Normalization to recurrent neural networks for the task of language modeling. Recently, some early success of applying Batch Normalization to Long-Short Term Memory (LSTM) networks has been reported in [3]. We elaborate on a few practical tricks in order to successfully apply batch normalization to LSTMs, show that it serves as a regularizer, and note that most mini-batch statistics can be shared across time steps. We also show that a similar Batch Normalization construction can be successfully applied to Gated Recurrent Units.

1 Introduction

Covariate shift [11] is the phenomenon in which the distributions of input data change between training and validation/production time, thus hurting the performance of the trained model. To alleviate this effect, it is a common practice before training a machine learning model to normalize and de-correlate the features to have zero mean and unit variance, known as feature "whitening." [8]

Ioffe et. al observe in [6] that since neural network models consist of stacked layers, covariate shift happens internally between layers during training. As the parameters of a model are updated by backpropagation, the distributions of a lower layer's activations change over time and cause "internal covariate shift." Thus the parameters of the upper layer must constantly adapt to this change in the distribution of incoming activations, which slows training convergence and model performance. To alleviate this problem, the authors propose Batch Normalization (BN), which normalizes each unit of a layer using the mean and variance calculated from the mini batch (but does not de-correlate them for computational efficiency). They report that batch normalization reduces the number of steps required to train feed-forward models by 10x and outperforms their baseline models in prediction.

Recurrent neural networks (RNN) such as Long-Short Term Memory (LSTM) [4] and Gated Recurrent Units (GRU) [1] have shown state-of-the-art performance on sequence modeling tasks, but they are known to be very difficult to train. Due to variable-length recurrent connections, applying batch normalization to a recurrent neural network is also not straightforward. There has been some early success as demonstrated in [3], and we continue to explore the effects of batch normalization on RNNs using a character-level language modeling task on the Penn Tree Bank (PTB) dataset [9].

2 Background

2.1 Recurrent Neural Network

A recurrent neural network (RNN) is a neural network with a feedback loop for a sequence of inputs $\boldsymbol{X} = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_T\}$ defined as

$$\boldsymbol{h}_t = f\left(\boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b}\right)$$

where f is a nonlinear function. However, this simple form of the RNN is difficult to train due to exploding and vanishing gradients from the recurrent layer. Typically exploding gradients are dealt with by gradient clipping, which bounds the norm of the gradient [10]. However, vanishing gradients are more difficult identify, and thus architectures such as LSTM and GRU were created to mitigate this problem.

2.1.1 Long-Short Term Memory

LSTM [4] defines gates that control the contributions from previous memory and current input to a linearly combined memory cell, as well as the contribution from the new memory cell to the output state. This finer grained control allows the LSTM to retain relevant information over longer time steps and avoid the vanishing gradient problem.

$$\begin{aligned}
\begin{pmatrix} \boldsymbol{f}_t \\ \boldsymbol{i}_t \\ \boldsymbol{o}_t \\ \widetilde{\boldsymbol{c}}_t \end{pmatrix} &= \boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b} \\
\boldsymbol{c}_t &= \sigma\left(\boldsymbol{f}_t\right) \circ \boldsymbol{c}_{t-1} + \sigma\left(\boldsymbol{i}_t\right) \circ \tanh\left(\widetilde{\boldsymbol{c}}_t\right) \\
\boldsymbol{h}_t &= \sigma\left(\boldsymbol{o}_t\right) \circ \tanh\left(\boldsymbol{c}_t, \gamma_c, \beta_c\right)
\end{aligned}$$

2.1.2 Gated Recurrent Unit

Gated Recurrent Unit [1] also utilizes *input* and *update* gates to control the contributions from the current input and the previous state. Unlike LSTM, however, it has no cell memory that is passed to the next time step separately from the output hidden state.

$$\begin{aligned}
\begin{pmatrix} \boldsymbol{r}_t \\ \boldsymbol{z}_t \end{pmatrix} &= \boldsymbol{W}\boldsymbol{h}_{t-1} + \boldsymbol{U}\boldsymbol{x}_t + \boldsymbol{b} \\
\tilde{\boldsymbol{h}}_t &= \boldsymbol{r}_t \circ \boldsymbol{W}^{(\boldsymbol{h})}\boldsymbol{h}_{t-1} + \boldsymbol{U}^{(\boldsymbol{h})}\boldsymbol{x}_t \\
\boldsymbol{h}_t &= \boldsymbol{z}_t \circ \boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t) \circ \tanh\left(\bar{\boldsymbol{h}}_t\right)
\end{aligned}$$

2.2 Batch Normalization

Batch normalization [6] attempts to alleviate the problem of internal covariate shift by approximately normalizing the activations x_i of a layer to zero mean and unit variance using statistics calculated from the mini-batch, $\boldsymbol{x}_i$.

$$BN\left(x_i, \boldsymbol{x}_i, \gamma_i, \beta_i\right) = \beta_i + \gamma_i \frac{x_i - \widehat{\mathbf{E}\left[\boldsymbol{x}_i\right]}}{\sqrt{\widehat{\mathbf{Var}\left[\boldsymbol{x}_i\right]} + \epsilon}}$$

Batch normalization is an element-wise operation. Note that activations are not de-correlated because that would involve an expensive computation of the covariance matrix.

By scaling and shifting the activations by trained parameters γ and β, we allow the network to learn the optimal distribution while maintaining the expressiveness of the original network. If appropriate,

the batch normalization operation can be completely "undone" by setting $\beta = \mathbf{E}[\boldsymbol{x}_i]$ and $\gamma = \mathbf{Var}[\boldsymbol{x}_i]$.

During training, the means and variances are calculated and tracked from the mini-batch. Cooijmans et. al [3] advocate tracking separate means and variances for each time step. Note that this makes the activation of each data point a function of all other data points in the mini-batch, and errors must be backpropagated accordingly. During prediction, means and variances estimated during training are used rather than mini-batch statistics.

3 Batch Normalized Recurrent Neural Networks

3.1 Batch Normalized LSTM

Following [3], we apply batch normalization to the hidden states and input separately, but not the cell state, to preserve memory. Also rather than learning two separate bias terms, we let the bias term $\boldsymbol{b}$ serve as the total bias from both batch normalization operations for the gates and cell input.

We subscript BN by time (BN_t) to indicate that each time step tracks its own mean and variance. In practice, we track these statistics as they change over the course of training using an exponential moving average. Note, however, the γ and β parameters are shared across time steps and do not have the t subscript.

$$\begin{pmatrix} \boldsymbol{f}_t \\ \boldsymbol{i}_t \\ \boldsymbol{o}_t \\ \bar{\boldsymbol{c}}_t \end{pmatrix} = BN_t\left(\boldsymbol{W}\boldsymbol{h}_{t-1}, \gamma_h\right) + BN_t\left(\boldsymbol{U}\boldsymbol{x}_t, \gamma_x\right) + \boldsymbol{b}$$

$$\begin{aligned} \boldsymbol{c}_t &= \sigma\left(\boldsymbol{f}_t\right) \circ \boldsymbol{c}_{t-1} + \sigma\left(\boldsymbol{i}_t\right) \circ \tanh\left(\bar{\boldsymbol{c}}_t\right) \\ \boldsymbol{h}_t &= \sigma\left(\boldsymbol{o}_t\right) \circ \tanh\left(BN_t\left(\boldsymbol{c}_t, \gamma_c, \beta_c\right)\right) \end{aligned}$$

3.2 Batch Normalized GRU

We use a similar construction to batch normalize GRU, where the input and hidden terms are normalized separately:

$$\begin{pmatrix} \boldsymbol{r}_t \\ \boldsymbol{z}_t \end{pmatrix} = BN_t\left(\boldsymbol{W}\boldsymbol{h}_{t-1}, \gamma_h\right) + BN_t\left(\boldsymbol{U}\boldsymbol{x}_t, \gamma_x\right) + \boldsymbol{b}$$

$$\begin{aligned} \tilde{\boldsymbol{h}}_t &= \sigma\left(\boldsymbol{r}\right) \circ BN_t(\boldsymbol{W^{(h)}}\boldsymbol{h}_{t-1}, \gamma_h^{(h)}) + BN_t(\boldsymbol{U^{(h)}}\boldsymbol{x}_t, \gamma_x^{(h)}) \\ \boldsymbol{h}_t &= \sigma\left(\boldsymbol{z}_t\right) \circ \boldsymbol{h}_{t-1} + \left(1 - \sigma\left(\boldsymbol{z}_t\right)\right) \circ \tanh\left(BN_t\left(\tilde{\boldsymbol{h}}_t, \gamma_m, \beta_m\right)\right) \end{aligned}$$

4 Experiments

We explore the performance of these batch normalized models on the problem of character level language modeling on the PTB dataset [9]. The dataset consists of 42K training, 3K validation, and 3K test sentences. We kept only the 10K most common words, and turned other tokens into an <UNK> token. For the purposes of this experiment, we only consider sequences of 100 characters as is done in [3]. It also shows some preliminary result on a generalization of this scheme to longer sequences, and we will attempt to explain the reason why this model works for longer sequences by examining the means and variances more closely.

4.1 Batch Normalized LSTM

The networks were trained using 500 hidden units, batch size of 64, orthogonal initialization of matrices, global gradient clipping to the L2-norm of 1.0, and Adam [7] optimizer with learning rates shown below.

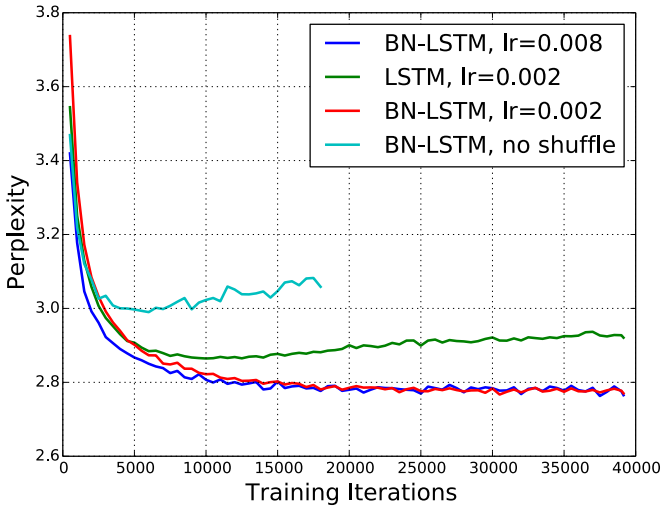

Figure 1: LSTM vs. BN-LSTM validation error. Observe the effect of shuffling and different learning rates.

Figure 1 shows the validation performance of LSTM and BN-LSTM for this problem. We note a few hyper-parameters that we found to be important for successful training:

- BN-LSTM reaches a lower validation error than LSTM when using the same learning rate, and it is also able to tolerate a much higher (4x) learning rate of 0.008, while maintaining model performance. This property is also observed in feed-forward networks in [6].
- The algorithm performed poorly if the batch size was too small. This is intuitive as a large enough sample must be present to accurately estimate mean and variance statistics. We found that the batch size of at least 64 was necessary.
- [6] observes shuffling the data between mini-batches and before each training epoch gave a 1% validation improvement. For LSTMs, we found a more significant impact. BN showed significantly poorer performance and overfitting if data was not shuffled.

4.2 BN-LSTM and Dropout

We see in Figure 1 that the vanilla LSTM begins to exhibit overfitting, while the BN-LSTM does not. We investigate this behavior of batch normalization as a regularizer by comparing it to dropout.

Figure 2 (left) shows the interaction of Dropout [12] (probability 0.5) with batch normalization. We find that while Dropout successfully prevents overfitting in LSTM, it hurts the performance of BN-LSTM, and that the BN-LSTM without Dropout performed the best in validation perplexity. This result agrees with [6], which also observed that Dropout was not necessary for a batch normalized network. We believe this is caused by the fact that Dropout interferes with estimated statistics from the mini-batch.

4.3 Sharing Batch Statistics

It is somewhat unnatural that we keep independent batch statistics at each time step in the LSTM when all other parameters are shared. In fact, [3] proposes not sharing the means and variances and only sharing the γ and β parameters, which is intended to allow the model to learn appropriate means and variances. The success of this model with untied statistics but tied γ and β suggests that the statistics must be at least fairly similar.

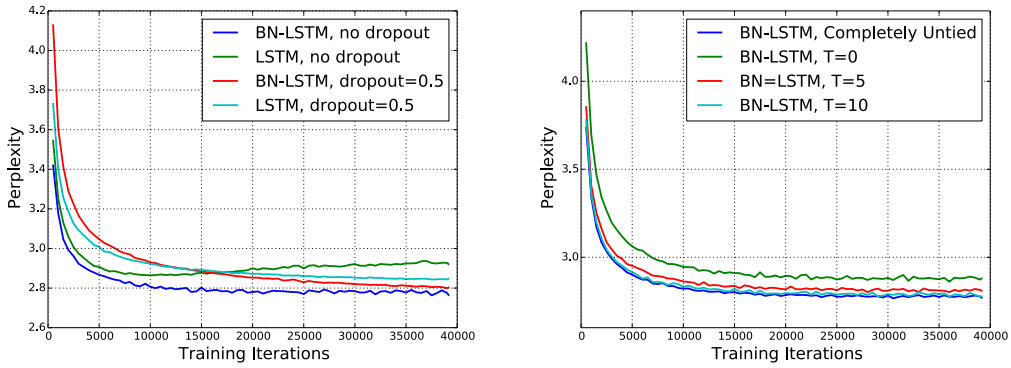

Figure 2: (Left) LSTM vs. BN-LSTM validation perplexity with Dropout. (Right) BN-LSTM validation perplexity, sharing statistics past time step $> T$.

Figure 3 shows quantiles of means at time steps 0, 2, 49, and 99 obtained via Tensorboard. We quantitatively observed that the means and variances are significantly different between time steps 0 and 2, but seem to converge for larger T. We believe that the initial fluctuation is caused by the zero vector that is used as the starting state before inputs are seen, which causes a "burn in" period before the statistics stabilize.

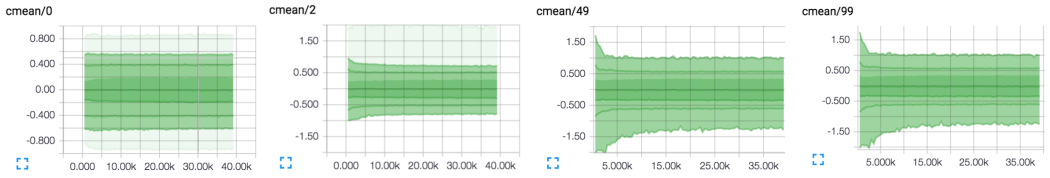

Figure 3: Means of memory c at time steps. Lines show quantiles of 500-dimensional vectors.

Figure 2 (right) shows the effect of sharing the mean and variance beyond time step T. We see that performance indeed degrades if statistics are shared across all time steps ($T = 0$). However, we can obtain comparable performance by sharing the statistics beyond time step 10.

This is an intuitive explanation for the demonstrated ability in [3] of the BN-LSTM to generalize to sequences that are longer than the ones seen in training using the the statistics from the last time step it knows of.

4.4 Batch Normalized GRU

For GRUs, we used the same hyper-parameters as LSTM experiments.

Figure 4 (left) shows the performance of batch normalized GRU (BN-GRU) on validation set. Notice that the vanilla GRU quickly reaches its optimal point and starts overfitting. In comparison, BN-GRU not only achieves better performance, but also avoids overfitting entirely. This behavior is consistent with our observation of BN-LSTM. Unlike for BN-LSTM, however, batch normalization slows down the initial convergence rate for BN-GRU.

As shown in Figure 4 (right), applying Dropout caused significant performance degrade for both GRU and BN-GRU. This is likely due to the same reason as for BN-LSTM where mini-batch statistics become inaccurate due to Dropout.

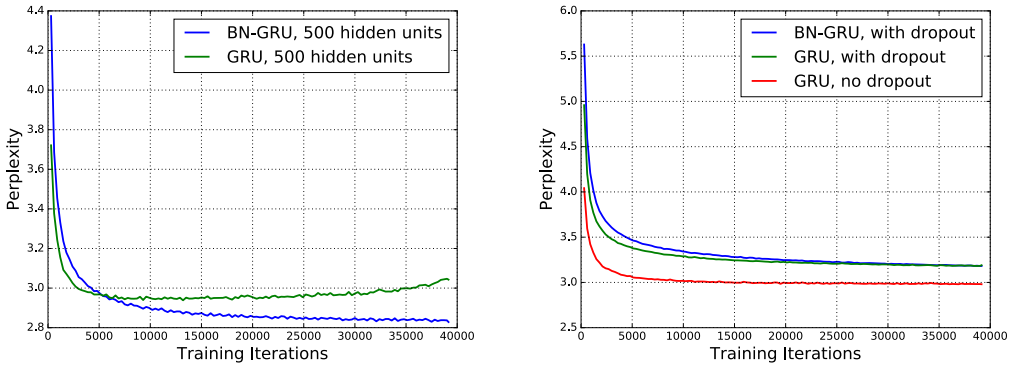

Figure 4: (Left) GRU vs. BN-GRU validation perplexity. (Right) GRU performance with and without Dropout.

4.5 Test Performance

Finally, we present the test performance of the models mentioned in this section in Table 1.

Model	Test Perplexity
LSTM	2.82
Dropout LSTM	2.77
BN LSTM	**2.69**
BN LSTM T=10	2.70

Table 1: Test set performance. BN LSTM T=10 is where statistics are shared beyond T=10.

5 Conclusion

In this work, we empirically showed various properties of batch normalized LSTM and GRU. Notably, we observed that BN is an effective regularizer for both LSTM and GRU, often better than Dropout. Also, BN allows the model to be trained with higher learning rates.

Our investigation into convergence of batch statistics suggests future work in hybrid models where BN is selectively applied, or using estimated statistics rather than calculating them from the mini-batch for backpropagation efficiency in later epochs of training.

References

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

[2] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[3] T. Cooijmans, N. Ballas, C. Laurent, and A. Courville. Recurrent batch normalization. *CoRR*, abs/1603.09025, 2016.

[4] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):17351780, 1997.

[5] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

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

[7] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv*, arXiv:1412.6980, 2014.

[8] Y. Lecun, L. Bottou, G.Orr, and K. Muller. *Neural Networks: Tricks of the trade*. Springer, Reading, Massachusetts, 1998.

[9] M. Mitchell, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of english: The penn treebank. *Comput. Linguist.*, 19(2):313330, 1993.

[10] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *CoRR*, abs/1211.5063, 2012.

[11] H. Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227244, 2000.

[12] W. Zaremba, I. Sutskever, and O. Vinyals. Recurrent neural network regularization. *arXiv*, arXiv:1409.2329, 2014.