Binarized Neural Networks for Language Modeling

Weiyi Zheng
SCPD Student
Stanford University
Stanford, CA 94305
weiyi@zhengwy.com

Yina Tang
SCPD Student
Stanford University
Stanford, CA 94305
yn.tang1@gmail.com

Abstract

Inspired by recent research on binarized convolutional neural networks, in this project, we tried binarization of recurrent neural network (LSTM) in two different models, one with binarized weights only and the other binarizing both the weight and the input embedding matrix (XNOR). Our results showed comparable performance with binarized weight on both Penn Tree Bank and Stanford Sentiment Treebank datasets compared to the original floating point models, while the XNOR model showed significant loss in model performance and is yet to be improved. We also implemented a custom CUDA kernel for the XNOR function and observed speedup of 1.6x on large matrices compared to CUBLAS.

1 Introduction

Recently there has been a lot of work involving reducing the size and computation cost of neural networks through quantization/binarization while maintaining the performance of the model at approximately the same level. For natural language processing for deep learning, it usually involves training language models of huge corpus and the training speed is slow as well. The Penn Treebank dataset contains 7 million words and training usually takes days. If we are able to binarize a network without losing much of its accuracy, we might also enable a wider adoption of NLP programs in various mobile devices.

In this project, we looked into applying the binarization of the LSTM cell by applying similar techniques as in [1] with the goal of achieve a similar accuracy and boost in speed. Our baseline model is the single large regularized LSTM model from Zaremba et.al [2], and we applied two different binarization techniques: binarizing the weights only and binarizing both weights and activations. We evaluated the performance of this binarized LSTM network with Perplexity as the intrinsic metric on the Penn Tree Bank [3], and accuracy of sentiment classification as the extrinsic metric using Stanford Sentiment Treebank [4].

2 Background/Related Work

The idea of binarizing weight in convolutional neural network was first explored by Bengio et al [12]. Later this idea was extended to binarized activations (inputs) as well with the goal of reducing the number of floating-point multiplications[13]. They investigated the possibility of speeding up the training of convolutional neural network by binarizing the input and weights and

perform bit count and XNOR instead of matrix multiplication. They showed that binarization acts as a powerful regularizer of the network and results on small networks such as CIFAR-10 were competitive, but not good enough for large networks such as ImageNet. Recently Rastegari, Mohammad et al.[1] extended this idea by adding a scalar to the binarized version of the matrix. They were able to achieve ~ 32x memory saving with only ~12% of accuracy loss on Alexnet. The model size also reduced by 95%. All these improvements made running a state-of-art neural network on mobile and low power processors possible.

3 Approach

3.1 Binarizing LSTM

Here is a brief overview of the internal structure of an LSTM. x_t is the input at timestep t, and h_{t-1} is the hidden state from timestep t-1.

$$d_t = [x_t \; h_{t-1}] \tag{1}$$

$$I_t, F_t, O_t, G_t = W d_t + B \tag{2}$$

$$\{i_t, f_t, o_t\} = sigmoid(\{I_t, F_t, O_t\})$$

$$g_t = tanh(G_t)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ g_t$$

$$h_t = tanh(c_t) \circ o_t$$

$$y^{hat} = Uh_t + V$$

Our modification focus on binarizing W in equation (2). We used the method described in XNOR-net [1] to produce the binarized W and its coefficient α. A single coefficient was producing good enough result for the binary weight LSTM. We apply the same technique to binarize the bias term B as well.

$$W_b^* = sign(W)$$

$$\alpha^* = \frac{1}{n}\|W\|_{l1}$$

For XNOR LSTM that binaries both the weight and the input, we rewrite equation (2) to separate the weight W into W_x and W_h .

$$I_t, F_t, O_t, G_t = W_x x_t + W_h h_{t-1} + B \tag{3}$$

This allows us to introduce 2 separate coefficient α_{wx} and α_{wh} into the equation, which improves the performance.

3.2 Binarizing Fully Connected Layer

We considered the possibility of binarizing fully connected layers in both language modelling tasks and sentiment analysis tasks, but didn't proceed with the experiment. The main reason is the matrix size. Given the language modelling tasks, with a hidden size of 1500 and the embedding size of 1500, the weight matrix in 2 layers of LSTM has a size of 3000 x 1500 x 4 x2, which makes up ~70% of the entire model. Looking at the 5-label sentiment analysis model, a hidden size of 168 plus an embed size of 300 will make the size of LSTM weights 468 x 300 x 4, which makes up ~92% of the entire model. Binary LSTM is efficient then binarizing fully connected layers.

The second reason is speed. Consider the speed up in experiment section 4.1, XNOR kernel at current implementation benefits the most when the matrix size are large. A smaller matrix multiplication actually negatively impact the performance.

3.3 Gradients Propagation

When calculating gradients for the binarize operation, we use the straight-through estimator describe by Courbariaux et. al [5]. Consider the sign function

$$q = sign(r)$$

Assume g_r is an estimator of the gradient $\frac{\partial C}{\partial q}$. The gradient $\frac{\partial C}{\partial r}$ is

$$g_r = g_q 1_{|r| \leq 1}$$

This equation preserves the gradient information, but also cancels the gradient when r is too large.

Following the method mentioned in [1], the gradients update are performed on the original floating point (real value) version of the weight, not on the binarized version. Because the gradients during updates are tiny, and can't be reflected on the binarized weight. [1,5,6] all use this technique to perform gradient updates.

4 Experiments

4.1 XNOR-bitcount Performance

We first investigated the ~32x speed up claimed by XNOR-net [1] by implementing custom CPU and GPU kernels in Tensorflow to compare the run time for different matrix sizes. Though our implementation is certainly not the most efficient, it should still serve as a baseline to show some performance comparison.

Table 4.1.1 CPU XNOR Performance Comparison

Speed / Matrix size	**10 x 300, 300 x 4500**	**4096 x 4096, 4096 x 4096**
Tensorflow Matmul	1x	1x
XNOR on CPU	**0.58x**	**0.28x**

Table 4.1.2 GPU XNOR Performance Comparison

Speed / Matrix size	10 x 300, 300 x 4500	4096 x 4096, 4096 x 4096
cuBLAS	1x	1x
XNOR Kernel	**0.3x**	**1.6x**

The cpu implementation is fairly poor due to the highly optimized modern matrix multiplication assembly instructions. On GPU we were only able to achieve speedup against cuBLAS for larger matrix. The reason is likely due to the unoptimized block/thread dimension when launching configuring GPU kernels. Due to the interest of time, we didn't delve too far into optimizing this kernel.

4.2 Language Modeling on Penn Treebank

We evaluated the perplexity of word level language modeling on the Penn Treebank dataset [3].The real-value model was pretrained following the example provided by Tensorflow and used to initialized the binarized LSTM. On Binary Weight LSTM, our model's performance was comparable to the real-value model.

Table 4.2.1 Comparison for Binary Weight LSTM

(hidden size = 1500, timesteps = 35, dropout = 0.65, learning rate = 0.001)

Model	Test Perplexity
Reference Model[2]	78.29
Reference Model (ours)	81.532
Binary Weight LSTM	**80.461**

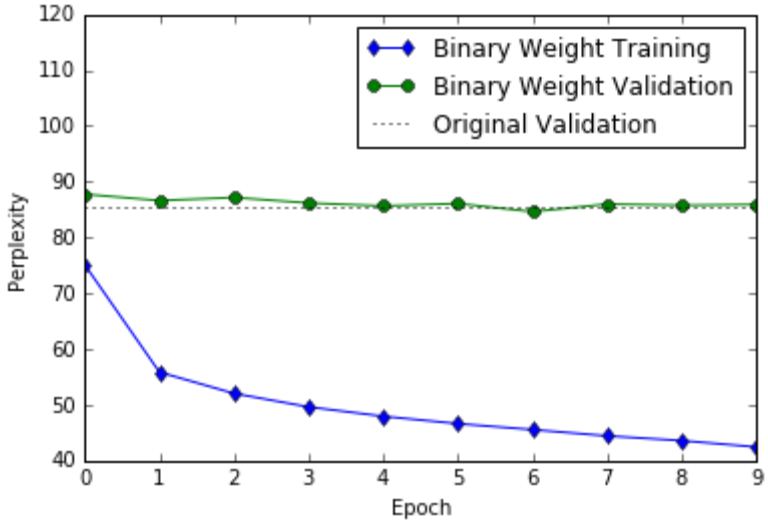

Fig. 4.2.2. Training and Validation Perplexity of the Binary Weight LSTM Model on Penn Treebank

Using a binary weight network and pretrain actually gave us about 1 perplexity advantage over the original model. We think this is because binarization is similar to a form of regularization, by reducing precision, the model was able to generalize better. This effect was very pronounced when

we were experimenting with a smaller network at the beginning.

Table 4.2.3: Comparison for XNOR LSTM

(hidden size = 200, timesteps = 20, dropout = 0.3)

Model	Test Perplexity
Reference Model [2]	117.18
XNOR network	137.34
XNOR with Batch Normalization	139.5

Binarizing both inputs and the weights proved to be difficult. The model lost 20 perplexity even after pretraining. The test perplexity also started to be bigger comparing with the validation perplexity, which we didn't observe in Binary Weight LSTM. Due to time constraints, we were not able to perform a similar experiment on a larger network. But we can expect the result to be poor.

4.3 Sentiment Analysis on Stanford Sentiment Treebank

For this part of the evaluation, we chose the 5-class classification task on the Stanford Sentiment Treebank dataset. The network architecture is shown in Fig 4.3.1. We take the input word indices from the input sentence for each word, go through the embedding layer, and then feed the word vectors into the binarized LSTM one by one. At the end of the sentence, the final hidden state of the LSTM is fed into a fully connected layer and sentiment is predicted via softmax. We utilized Tensorflow's dynamic RNN [1] to handle the different lengths of the sentences.

We chose the basic LSTM module in [4] as the reference model and replaced the LSTM cell with our binarized and XNOR versions. During optimization, we used Adam Optimizer [9] instead of the RMSprop mentioned in [4] due to its relative lack of support in the current version of Tensorflow. We adjusted the learning rate and learning rate decay for Adam Optimizer, as it was found to converge very quickly and and tends to bounce around the local minimum once it's converged. Decaying the learning rate after a few epochs showed improvement for our model.

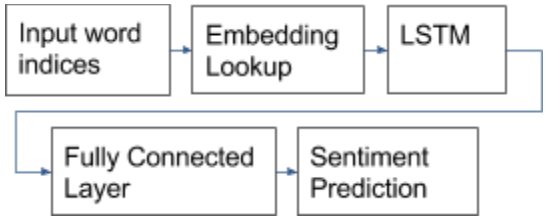

Fig. 4.3.1. Network architecture of sentiment analysis for one time step. Prediction is made on the final hidden state of the sentence.

[1] https://github.com/tensorflow/tensorflow/blob/07db1806460cbebbbf3abe9174a65eb52c8a63e0/tensorflow/python/ops/rnn.py

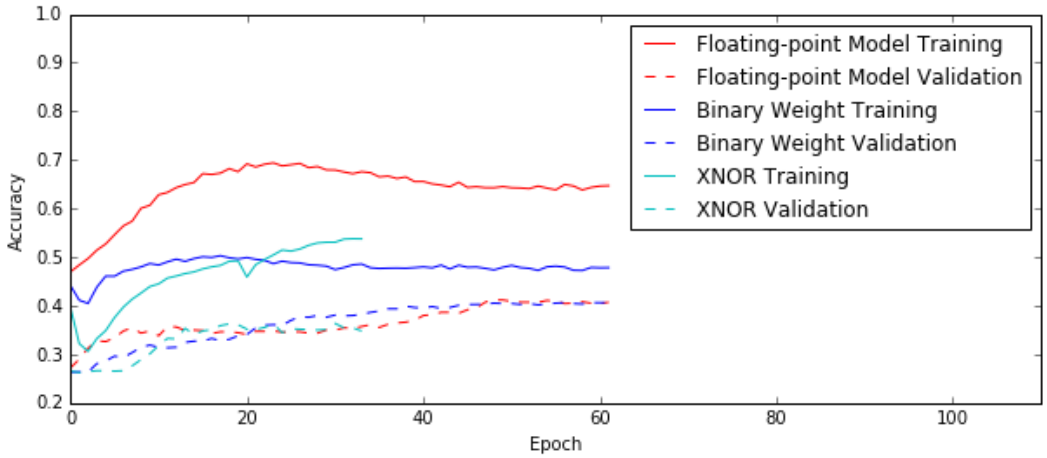

Fig. 4.3.2. Sentiment analysis accuracy across epochs for 3 different models: floating point, binary weight, and XNOR network all trained with Dh = 168, Dx = 300, lr = 0.0001, dropout = 0.5, l2 = 0.0001, batch_size = 5, lr_decay = 0.9 after 3 epochs with Adam Optimizer. Embedding is initialized with GloVe[10] vectors. For the XNOR network, dropout = 1.

Table 4.3.3. Train, validation and test accuracies for 3 different models using the Stanford Sentiment Treebank

Accuracy (%)/Model	Train	Validation	Test
Full-precision LSTM	64.7	40.2	44.4
Binary weight LSTM	47.8	40.6	41.8
XNOR LSTM	53.75	34.8	30.4

Our result in Fig 4.3.2 showed that training accuracy actually dropped significantly for the binarized LSTM model compared to the reference model, and we noticed a drop in the training accuracy at the beginning of the training that we think is due to the binarization of weights killing off benefits from initialization the embedding matrix with glove vectors. However, at the end of the training the binarized LSTM was able to achieve similar validation accuracy and the test accuracy dropped 2.6% compared to the reference model. This is similar performance degradation compared to [1]'s attempt to binarize convolutional neural networks.
For XNOR network, we were not able to train further epochs due to validation loss started to increase again. This means the model has already overfitted with the training data. And the 4% drop between validation and test accuracy confirmed this overfitting too. A simple dropout would not work with XNOR network because all the dropout that got turned to 0 become -1 after the sign() function, which effectively means adding random noise in the input data, instead of turning part of the network off. We still need to investigate further on how to avoid overfitting without dropout.

5 Conclusion

Using the technique described in XNOR-net[1], we experimented with binary weight LSTM and XNOR LSTM. Binary weight LSTM showed comparable performance with floating point LSTM. During inference, the weights only needs to be stored 1 bit per parameter, which results in a 32x size reduction from floating point. XNOR LSTM failed to recover its performance after re-training, and heavy overfitting occurred. The traditional dropout layer does not really work with binarized input. We think some possible techniques to explore in the future includes batch normalization[11] and shift-based batch normalization [5].

Also XNOR kernel does not guarantee speed ups. It depends on the user case and the hardware architecture. Only by benchmarking against the specific use case can the user tell whether this technique is worth using or not.

6. References

[1] Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks, 1–17. Retrieved from http://arxiv.org/abs/1603.05279

[2] Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent Neural Network Regularization. Neural and Evolutionary Computing. Retrieved from http://arxiv.org/abs/1409.2329

[3] Taylor, A., Marcus, M., & Santorini, B. (2003). Treebanks: Building and Using Parsed Corpora. In A. Abeillé (Ed.), (pp. 5–22). Dordrecht: Springer Netherlands. http://doi.org/10.1007/978-94-010-0201-1_1

[4] Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks, 10. http://doi.org/10.1515/popets-2015-0023

[5] Courbariaux, M., Bengio, Y.: Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. CoRR (2016)

[6] Courbariaux, M., Bengio, Y., David, J.P.: Binaryconnect: Training deep neural networks with binary weights during propagations. In: Advances in Neural Information Processing Systems. (2015) 3105–3113

[7] Recurrent Neural Networks. https://www.tensorflow.org/versions/r0.8/tutorials/recurrent/index.html

[8] Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., & Courville, A. (2016). Recurrent Batch Normalization, 1–10. Retrieved from http://arxiv.org/abs/1603.09025

[9] Diederik K., Jimmy B. (2014): Adam: A Method for Stochastic Optimization

[10] Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 1532–1543. http://doi.org/10.3115/v1/D14-1162

[11] Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., & Courville, A. (2016). Recurrent Batch Normalization, 1–10. Retrieved from http://arxiv.org/abs/1603.09025

[12] M. Courbariaux, Y. Bengio, et al., BinaryConnect: Training Deep Neural Networks with binary weights during propagations, 2014, Retrieved from http://arxiv.org/pdf/1511.00363v3.pdf

[13] Z. Lin, M. Courbariaux, Y. Bengio, et al., Neural Networks with Few Multiplications, 2015, Retrieved from https://arxiv.org/pdf/1510.03009.pdf