Q-LSTMs: Quantized LSTM*

Sivakumar Chidambaram
McGill University
Montreal, QC, Canada
csivanitw@gmail.com

Ian Porada
McGill University
Montreal, QC, Canada
ian.porada@mail.mcgill.ca

Abstract

Implementing deep neural networks on power and space constrained devices such as embedded processors has been a big challenge. Generally, the parameters and the intermediate outputs in neural networks are represented as 32-bit floating point numbers; however, recent research has shown that such high precision representations are not always required. Specifically, quantization of parameters can reduce memory requirements and increase computational efficiency with only minor degradation in accuracy. In this work, we explore the effects of quantization as a regularizer on Long Short Term Memory (LSTM) models in the context of language modelling and neural machine translation tasks. Under the constraints of our experiments, we achieve competitive perplexities as compared to a state-of-the-art regularizer in language modelling, and find minor degradation of BLEU scores in neural machine translation.

1 Introduction

Most deep learning applications have been successful due to the availability of large data, advanced algorithms, and powerful hardware devices to run these algorithms in reasonable time. However, sometimes there is a need to run deep learning applications on power and space constrained devices such as mobile phones and robots. Quantization (Courbariaux et al., 2015), pruning (Han et al., 2015), and compression (Cheng et al., 2017) are some of the few techniques employed to implement Convolutional Neural Networks (CNNs) on such devices. Yet it is only very recently that these techniques are being applied to Recurrent Neural Networks (RNNs).

In this work, we focus specifically on quantization which is still in the early stages of being applied to natural language processing tasks (Guo, 2018). The three main advantages of quantization are:

- Acts as a regularizer and prevents overfitting
- Reduces required memory storage
- Increases computational efficiency

Our goal is to extend the quantization techniques used in CNNs to RNNs. We hypothesize that, following from the successes of quantized CNNs, quantization will result in only minor degradation of accuracy in sequence modelling tasks. To test this hypothesis, we implement a LSTM with quantization and compare its performance against a vanilla LSTM on two sequence modelling tasks: language modelling and neural machine translation.

For language modelling, we consider a subset of the Wikitext-2 dataset (Merity et al., 2016) and compare models by perplexity. We built our code on top of the vanilla (32-bit floating point) LSTM implementation included in Salesforce's LSTM Language Model Toolkit (Merity et al., 2017). As this toolkit includes an implementation of recurrent dropout (Semeniuta et al., 2016), a state-of-the-art RNN variant of DropConnect (Wan et al., 2013), we are able to compare our quantization method directly against the recurrent dropout regularizer.

For neural machine translation, our code is similarly built on top of the vanilla LSTM implementa-

*The code for this project is available at https://github.com/csk7/CS550-NLP-McGill

tion of OpenNMT (Klein et al., 2017). We compare translation performance by BLEU scores (Papineni et al., 2002) on the Multi30K dataset (Elliott et al., 2016). Both of our LSTM implementations are in PyTorch.

2 Related Work

BinaryConnect (Courbariaux et al., 2015) was the first quantized deep neural network to achieve near state-of-the-art results. In this method, weights are binarized in the forward pass by a simple deterministic rounding function:

$$x^b = \text{Sign}(x) = \begin{cases} +1 & x \geq 0, \\ -1 & \text{otherwise} \end{cases} \quad (1)$$

While these binarized weights are used to calculate outputs, the original 32-bit weights are also stored to be used for parameter updates in the backward pass. Stochastic Gradient Descent (SGD) is dependent on small, noisy updates and thus requires high resolution parameters in order to work. Furthermore, the gradient of the rounding function is zero at almost every point (except for when $x = 0$ where it is undefined), so BinaryConnect uses a a straight-through estimator (STE) (Hinton et al., 2012) in the backward pass to avoid these zero gradients. Our method is largely based on this work, and we provide a deeper explanation of STE in Section 3. BinaryNet (Courbariaux et al., 2016) extended BinaryConnect to quantize activations in addition to weights.

XNOR-Net (Rastegari et al., 2016) introduced a scaling layer architecture, wherein they quantized the weights and activations, and used a scaling layer to scale the intermediate outputs. The use of a scaling layer was motivated by forming quantization as an optimization problem. This technique has been applied to LSTMs for language modelling on Penn Treebank (Zheng and Tang, 2016) where perplexity was shown to be slightly worse than that of a vanilla LSTM. While these experiments only considered binary quantization, in our experiments we consider multiple levels of quantization on a more recent dataset. We also consider how quantization performs relative to recurrent dropout, a state-of-the-art regularization technique.

Further work has compared 2, 3, and 4-bit quantization of an LSTM model on Penn Treebank (Hubara et al., 2016). They found that 4-bit quantization performs similarly to the full 32-bit vanilla LSTM. This work does not, however, consider higher levels of quantization nor does it provide specific details as to the implementation of quantizing the LSTM cell gates and interlinks.

Binarized LSTM models were first tested on large vocabulary language modelling in (Liu et al., 2018). This work is closely related to ours, but again we consider higher levels of quantization on a more recent dataset.

More complicated quantized LSTMs have been applied to language modelling: parameter dependent adaptive thresholds (He et al., 2016), which outperformed previous quantization techniques on Penn Treebank, and empirically motivated thresholding (Alom et al., 2018). In our work, we focus only on quantization by uniform thresholding.

(Xu et al., 2018) is the only attempt other than ours to test quantization of RNNs on Wikitext-2. This method compares optimization approaches to quantizing LSTM, again in contrast to our simple uniform quantization approach. In addition, we consider our quantization performance relative to pruning regularizers.

To the best of our knowledge, there is no previous work applying quantization to neural machine translation tasks.

3 Models

In this section, we detail the modification to the existing binarization algorithm, propagation of gradients through a straight through estimator, and how the vanilla implementations are adapted for the chosen datasets.

3.1 Quantized LSTMs

In each LSTM cell the following equations are performed:

$$f_t, i_t, o_t = \sigma_g(W_{f,i,o}x_t + U_{f,i,o}h_{t-1} + b_{f,i,o}) \quad (2)$$

$$c_t = f_t \bullet c_{t-1} + i_t \bullet \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \quad (3)$$

$$h_t = o_t \bullet \sigma_h(c_t) \quad (4)$$

where σ is the sigmoid activation, t is the time step, W are the weights, and b are the biases. As we can see, the majority of the operations in LSTMs

are matrix multiplications between the weights and inputs/hidden outputs. So, we would like to quantize the weights, inputs, and/or hidden outputs while maintaining the overall accuracy of the model. Binarization (equation 1) was extended to n-bit uniform quantization in as shown in the figure 1.

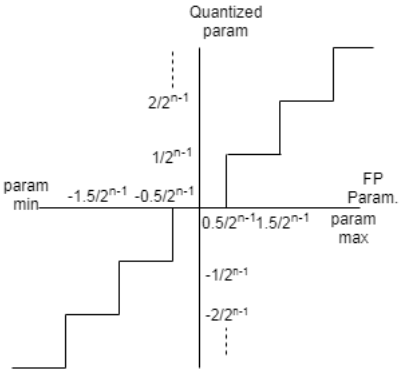

Figure 1: Uniform quantization to n bits

To calculate the gradients in the quantization operation, we use the a version of straight through estimator proposed in (Bengio et al., 2013) for binarization. Suppose g_q is the estimator of the gradient $\frac{\partial C}{\partial q}$ (C is the cost function) then the straight through estimator $\frac{\partial C}{\partial Q}$ is:

$$g_r = g_q 1_{|r|<1} \tag{5}$$

This ensures that the gradient information is preserved and also prevents exploding gradients. Essentially, in the forward pass we threshold the weights to achieve quantization, but this threshold function will have a zero gradient almost everywhere. To avoid this zero gradient in the backward pass, this straight through estimator approach calculates the gradients of our thresholding as an identity when $|r| < 1$ (equation 5). The gradient is still zero when $|r| \geq 1$.

Another technique introduced in the Gated-XNOR architecture (Deng et al., 2017) (for CNNs) involves multiplying the gradient g_q with smoothed delta-dirac functions. However, we found this technique to converge more slowly.

3.2 Language Modelling

For language modelling, the vanilla LSTM network consists of an embedding layer (200 embeddings per word) and 3 hidden layers (550 cells each) with recurrent dropout. These hyperparameters follow from the original tuning (Merity et al., 2017), although we decrease the embedding size and number of cells to half of the original implementation due to limited GPU memory. For the same reason, the model was trained on half of the Wikitext-2 dataset.

Our LSTM class with quantization is a custom implementation, inheriting some attributes from original PyTorch LSTM class. A fully connected layer is used as a classifier, followed by splitcross entropy loss. The word vectors are tied to the word classifier as explained by (Inan et al., 2016) that facilitates better language modelling as it reduces the number of trainable parameters and utilizes all the information effectively.

We tested modifications to our LSTM models such as using pretrained word embeddings or applying Layer Normalization (Ba et al., 2016), but these did not result in a noticable improvement.

3.3 Neural Machine Translation

Language translations tasks are much more complex than language modelling as they are multi-input, multi-output systems. The vanilla network contains a decoder, encoder, attention network, and generator. We use the Multi30K translation from English to German. The preprocessing step creates dictionaries of 10,841 entries for the English text and 18,563 entries for the German text. Sub word units are learned using Byte pair encoding. The Encoder contains an embedding layer and an LSTM layer with 500 LSTM cells. The decoder contains an embedding layer and stacked LSTMs of two layers with 500 cells each. The attention network is made of a fully connected layer. The generator contains a fully connected layer and softmax layer.

For, quantized LSTM we use our same LSTM class used in language modelling. In addition, we create our own class for quantized fully connected layers for the global attention and generator networks. We store the original 32-bit floating point weights for the weight update step. The quantized weights are only used in the forward pass and gradients calculation (backward pass). We use BLEU scores to compare the results.

4 Results

4.1 Language Modelling Task

From table 1 it can be observed that by quantizing the network parameters (embeddings, weight parameters, and hidden outputs) to 4-bits, the model achieves lower valid set perplexity than the vanilla network (with a recommended weight dropout = 0.5) and the test perplexity is higher by 2 points. However, by reducing the representations to 4-bits the we gain 4 times the storage space and also the computational complexity is reduced. By increasing the precision to 8-bits, the perplexity decreases marginally. This suggests that 4-bits of representation is not enough to fit the data. As we increase the precision to 16-bits, the test perplexity is found to be lower than that of the vanilla implementation.

Model	Valid	Test
Vanilla (dropout = 0.5)	161.2	145.8
Q-LSTM 4 bits	157.84	147.90
Q-LSTM 8 bits	157.30	147.55
Q-LSTM 16 bits	155.13	143.7

Table 1: Perplexity of subset of Wikitext-2 task

In table 2, we measure the effect of quantizing each parameter. It can seen that by just quantizing the weights, we achieve the best perplexity, even less than vanilla network with dropout. Quantizing the embeddings and hidden outputs increase the perplexity marginally (as compared to quantizing the weight parameters alone), but reduces the computational complexity. This is similar to results achieved in (Courbariaux et al., 2016), where binarizing the weights resulted in better performance, but by binarizing the activations the accuracy dropped. Quantizing the embedding and the hidden outputs is a design decision to be taken considering the tradeoff between computational complexity and accuracy.

Model	Weights	Hidden	Inputs	Test
Vanilla	$float$	$float$	$float$	145.8
Q-LSTM	$4bits$	$float$	$float$	145.4
Q-LSTM	$4bits$	$4bits$	$float$	147.1
Q-LSTM	$4bits$	$4bits$	$4bits$	147.9

Table 2: Effect of quantizing each parameter

4.2 Neural Machine Translation

Since the neural machine translation task consists of various modules (Encoder, Decoder, Attention, Generator), the effect of quantization on each was observed individually (Table 3). The drop in BLEU score was the highest in the case of decoder as it has the most parameters. It was observed that by quantizing the embeddings and the hidden outputs, the BLEU scores drop to 10.85. Hence in this task we only quantize the weights and biases.

In table 3, it can be seen that using dropout the BLEU score improves, which suggests that the model had been overfitting. However, while quantizing the weight parameters, it was observed that the in all the 3 cases (4, 8, and 16-bit) the results were not better than dropout. Hence, quantization does not always work. But, it makes the model computationally less expensive when implemented on a processor.

5 Discussion and conclusion

In this work, we considered a simple method of LSTM quantization as evaluated on Wikitext-2 language modelling and Multi30K neural machine translation. Our results confirm our initial hypothesis that quantization causes only minor performance degradation as compared to a vanilla LSTM. In fact, in language modelling quantization actually outperforms a vanilla LSTM even when the vanilla LSTM is regularized with state-of-the-art recurrent dropout.

In both tasks, we only considered performance on a single dataset, so it is not necessarily true that our findings will generalize for all language modelling or neural machine translation tasks. Our choice of metric also limits our findings, as comparable BLUE scores does not necesarily mean Q-LSTM translations are of the same quality as a vanilla LSTM.

In the case of language modelling, our LSTM model was limited by the memory constraints of our GPU, so we only trained on a subset of the Wikitext-2 dataset with a model size smaller than that which has led to state-of-the-art results. Consequently, our final perplexities are not comparable to other evaluations on Wikitext-2. While we have shown that quantized LSTM can outperform vanilla LSTM on this constrained dataset with a limited model size, it does not immediately follow that quantized LSTM

Model	Encoder	Decoder	Attention	Generator	BLEU
Vanilla	$float(NoDropout)$				30.38
Vanilla	$float(RecommendedDropout = 0.3)$				30.95
Q-LSTM	$float$	$float$	$8bits$	$8bits$	30.17
Q-LSTM	$float$	$8bits$	$float$	$float$	28.24
Q-LSTM	$8bits$	$float$	$float$	$float$	29.00
Q-LSTM	$8bits$	$8bits$	$8bits$	$8bits$	25.60

Table 3: Effect of quantizing the params. in each module of NMT

would outperform vanilla LSTM on the full dataset with looser model constraints. This case could be explored in future work.

In both tasks our choice of hyperparameters was justified by vanilla LSTM performance, and our Q-LSTM model followed the same hyperparameters. It is possible that independent tuning on Q-LSTM may lead to even higher performance for quanitzed networks.

Future work could also consider the performance of quantized Gated Recurrent Units (GRUs) on neural machine translation tasks which is still largely unexplored. As a primary goal of quantization is to decrease memory usage, and GRUs have less parameters than LSTM models, this seems like a natural area to explore.

6 Statement of contributions

Chidambaram implemented Q-LSTM as used in both tasks, including quantization for LSTM and fully connected layers. Porada implemented pretrained word embeddings and layer normalization. Both contributed to the writing of this report.

References

[Alom et al.2018] Md. Zahangir Alom, Adam T. Moody, Naoya Maruyama, Brian C. Van Essen, and Tarek M. Taha. 2018. Effective quantization approaches for recurrent neural networks. *CoRR*, abs/1802.02615.

[Ba et al.2016] Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

[Bengio et al.2013] Yoshua Bengio, Nicholas Leonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv: 1308.3432*.

[Cheng et al.2017] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. 2017. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*.

[Courbariaux et al.2015] Matthieu Courbariaux, Yoshua Bengio, and Jean Pierre David. 2015. Binaryconnect: Training deep neural networks with binary weights during propagations. *Neural Information Processing Systems*.

[Courbariaux et al.2016] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *Neural Information Processing Systems*.

[Deng et al.2017] Lei Deng, Peng Jiao, Jing Pei, Zhenzhi Wu, and Guoqi Li. 2017. Gated xnor networks: Deep neural networks with ternary weights and activations under a unified discretization framework. *arXiV preprint*, 1705.09283.

[Elliott et al.2016] Desmond Elliott, Stella Frank, Khalil Sima'an, and Lucia Specia. 2016. Multi30k: Multilingual english-german image descriptions. In *Proceedings of the 5th Workshop on Vision and Language*, pages 70–74. Association for Computational Linguistics.

[Guo2018] Yunhui Guo. 2018. A survey on methods and theories of quantized neural networks.

[Han et al.2015] Song Han, Huizi Mao, and William J. Dally. 2015. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *arXiv preprint arXiv: 1510.00149*.

[He et al.2016] Qinyao He, He Wen, Shuchang Zhou, Yuxin Wu, Cong Yao, Xinyu Zhou, and Yuheng Zou. 2016. Effective quantization methods for recurrent neural networks. *CoRR*, abs/1611.10176.

[Hinton et al.2012] Geoffrey Hinton, Nitsh Srivastava, and Kevin Swersky. 2012. Neural networks for machine learning. coursera, video lectures, 264.

[Hubara et al.2016] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Quantized neural networks: Training neural networks with low precision weights and activations. *CoRR*, abs/1609.07061.

[Inan et al.2016] Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying word vectors and word

classifiers: A loss framework for language modeling. *International Conference on Learning Representations*.

[Klein et al.2017] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*.

[Liu et al.2018] Xuan Liu, Di Cao, and Kai Yu. 2018. Binarized LSTM language model. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2113–2121.

[Merity et al.2016] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *CoRR*, abs/1609.07843.

[Merity et al.2017] Stephen Merity, Nitish Shirish Keskar, and Richard Socher. 2017. Regularizing and optimizing LSTM language models. *CoRR*, abs/1708.02182.

[Papineni et al.2002] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 311–318, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Rastegari et al.2016] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*.

[Semeniuta et al.2016] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. 2016. Recurrent dropout without memory loss. *CoRR*, abs/1603.05118.

[Wan et al.2013] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. 2013. Regularization of neural networks using dropconnect. *International Conference on Machine Learning*.

[Xu et al.2018] Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha. 2018. Alternating multi-bit quantization for recurrent neural networks. *CoRR*, abs/1802.00150.

[Zheng and Tang2016] Weiyi Zheng and Yina Tang. 2016. Binarized neural networks for language modeling. Technical Report cs224d, Stanford University.