Binarizing Neural Machine Translation

Akshat Shrivastava, Kevin Bi, Sarah Yu
Paul G. Allen School of Computer Science
akshats@, kevinb22@, sarahyu@ (cs.washington.edu)

Abstract

We explore ways to reduce computation and model size for neural machine translation. With the development of binary weight networks and Xnor networks in vision, we attempt to extend that work to machine translation. In particular, we evaluate how binary convolutions can be used in machine translation and their effects.

1 Introduction

Various work has been done in the field of computer vision to lower the computational resources required to run these deep neural networks. Binarized Weight Networks and Xnor networks showed that it was possible to reduce the size of the model and computation by replaces all the weights with binary values in $\{1, +1\}$. Some work has been done to binarize different components of language models, however in this project we tackle binarizing the encoder and decoder layers.

With the development of ConvS2S by FAIR - a convolutional based encoder decoder model - we build a similar model, but binarize the convolutions to create a binary weight network and binarize the inputs to create an xnor network. We compare these binary models against various recurrent architectures on both translation performance and computation resources used.

2 Implementation

For this project, we implemented our own framework for machine translation and analyzing different computational performance metrics of these models. For some of the datasets we were using, we built on top of torchtext, a Pytorch library for NLP. However this library uploads datasets into memory at once, so we also implemented our own dataloaders and vocabularies to allow training on some of the larger datasets.

The models we analyze have all been implemented before, and are cited in our references. In spite of this, we reimplement each of these models to analyze the performance of these models further and in more detail. The convolutional model in particular is originally implemented assuming that the input to all the modules was of shape (time, batch, channel) and uses a custom convolution to operate on such an input (torch.convTBC).

We reimplement this to work with the normal Pytorch convolution module and our binarized version. This binarization framework has been implemented by AI2 in torch and another part in Pytorch for 2d convolutional model. We rewrote this to work for 1d convolutions and refactor the convolutional model to be able to use the binary convolutions either with binary input or with only binary weights.

3 Data

We initially experimented and implemented various training pipelines for different translations tasks with sizes. WMT 14 English to French with 40 million training pairs, IWSLT English to German with 200,000 training pairs, and Multi30k with 30,000 training pairs. We use the Multi30k dataset, mainly for its size; it is the smallest dataset.

4 Training

We train our models using an Adam optimizer. We also employ a learning rate scheduler with two parameters. The scheduler reduces the learning rate by a factor gamma after every step size epochs. Most our experiments employ a step size of 5 and a gamma of 0.1, meaning our learning rate is reduced by a factor of 10 every 5 epochs. We also clip our gradients to stop the exploding gradients issue.

In terms of the model itself, we evaluate the model predictions with the cross entropy loss compared to the ground truth predictions. We also use teacher forcing and feed the correct inputs at each time step to the decoders.

5 Models

We evaluate 6 different models: 4 baseline models and 2 variants of binarized networks. We compare various metrics on translation performance, runtime, and model size.

In this section, we list descriptions of the models along with the optimal hyper parameters we found for each of them.

5.1 Baseline Models

Simple LSTM: An encoder decoder model, that encodes the source language with an LSTM, then presents the final hidden state to the decoder. The decoder uses the final hidden state to decode the output.

- Tuning: Learning rate: 0.001, Batch size: 32, Epochs: 50, Gradient Clipping Threshold: 5, Scheduler step size: 5, Scheduler gamma: 0.1
- Model Parameters: 512 embedding dimensions, 512 hidden dimensions, 0.2 dropout, 4 layer encoder, 4 layer decoder.

Attention RNN: An encoder decoder model, similar to the last but at every decoder step applies an attention mechanism over all the encoder outputs conditioned on the current hidden state.

- Tuning: Learning rate: 0.001, Batch size: 32, Epochs: 50, Gradient Clipping Threshold: 5, Scheduler step size: 5, Scheduler gamma: 0.1
- Model Parameters: 512 embedding dimensions, 512 hidden dimensions, 0.2 dropout, 2 layer encoder, 2 layer decoder

Attention QRNN: The same model as above, but using QRNN (Quasi Recurrent Neural Network developed by Salesforce Research) instead of LSTMs. QRNN should be much faster since the rely on lower level convolutions and can be parallelized further than Attention RNN.

Tuning: Learning rate: 0.0003125, Batch size: 32, Epochs: 50, Gradient Clipping Threshold: 5, Scheduler step size: 1, Scheduler gamma: 0.1 Model Parameters: 512 embedding dimensions, 512 hidden dimensions, 0.2 dropout, 4 layer encoder, 4 layer decoder

ConvSeq2Seq: This model (implemented by FAIR) rather than using RNNs, creates a series of convolutional layers that are used for the encoder, and decoder along with attention.

- Tuning: Learning rate: 0.001, Batch size: 128, Epochs: 50, Gradient Clipping Threshold: 5, Scheduler step size: 5, Scheduler gamma: 0.1
- Model Parameters: 1024 positional embedding layer, 256 embedding dimension, encoder and decoder with: 4 convolutional layers with 256 in channels,512 out channels, kernel size of 3, and residual connections to the previous layer. Each convolution also applies the Gated Linear Unit non-linearity.

5.2 Binarized Models

ConvSeq2Seq Binarized Weight Network: This model is the same as the one implemented above, with one key difference. All the weights are represented as a binary tensor β, and a normalization vector such that $W \approx \beta \cdot \alpha$. The benefit here is that a convolution can be estimated as $(I \cdot \beta) \cdot \alpha$

- Tuning: Learning rate: 0.001, Batch size: 128, Epochs: 50, Gradient Clipping Threshold: 5, Scheduler step size: 5, Scheduler gamma: 0.1
- Model Parameters: 1024 positional embedding layer, 256 embedding dimension, encoder and decoder with: 4 convolutional layers with 256 in channels,512 out channels, kernel size of 3, and residual connections to the previous layer. Each convolution also applies the Gated Linear Unit non-linearity.

ConvSeq2Seq Xnor: This model extends upon the binarized weight network. The input is binarized as well so the convolutions can be estimated as $(sign(I) \cdot sign(\beta)) \cdot \alpha$.

- Tuning: Learning rate: 0.001, Batch size: 128, Epochs: 50, Gradient Clipping Threshold: 5, Scheduler step size: 5, Scheduler gamma: 0.1
- Model Parameters: 1024 positional embedding layer, 256 embedding dimension, encoder and decoder with: 4 convolutional layers with 256 in channels,512 out channels, kernel size of 3, and residual connections to the previous layer. Each convolution also applies the Gated Linear Unit non-linearity.

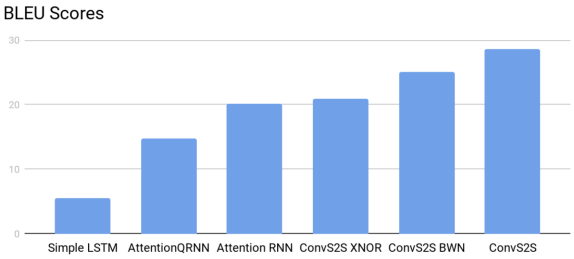
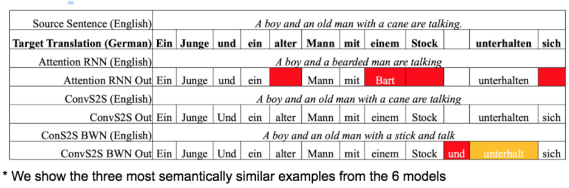


Figure 1: Bleu Scores by Model



* We show the three most semantically similar examples from the 6 models

Figure 2: Generated Translations by Model Against Target Sentence and Translation

6 Results

6.1 Translation Performance

To compare models performance on the Multi30k task, we generate output translations through greedy decoding and measure 4 metrics: the perplexity of the model, the BLEU score for the model, the smoothed BLEU score, and the train loss. We provide the full results in Appendix A and focus on BLEU scores in Figure 1.

- Perplexity: Allows us to measure how well the model is doing rather quickly, and is very useful for tuning the models
- BLEU: this is a more expensive score since it requires us to generate our output then compare it. We implemented a greedy search for decoding.
- Smoothed BLEU: Because our sentences are rather small, some of the issues that come up in BLEU are due to the sentence not being long enough to compare, and result in much lower scores. Smoothed BLEU helps give a better estimate of how the model is doing with variable length sentences. We implement BLEU and a smoothed BLEU using Chen Cherry Smoothing method number 4, which is for translation of short sentences.

In Figure 1 we see that the convolutional sequence model we tuned performs the best. We can also see how varying degrees of binarization affect translation by looking at the Biniary Weight Network and Xnor models. We provide translations generated by the models in Figure 2.

6.2 Model Impact on System

In order to measure and compare model performance, we poll several metrics during the train phase of the models. The metrics we focus on are

- size (in bits) of the model
- CPU utilization of the model
- GPU utilization of the model.

These three variables allow us to roughly estimate the impact of the model on the system. For example, we predicted that the ConvSeq2Seq model would be the smallest in terms of size (in bits) because both the weights and input were binary rather than floating point. All the parameters for each of the models were tested on the same EC2 instance to guarantee consistency. While each model ran, we poll the systems every minute.

6.2.1 Calculating Model Size

We calculate model size with respect to 16-bit, 32-bit, and 64-bit floating point values. For each model we unpack and calculate the total number of model parameters and then multiply the number by the the number n where n is the number of bits used to represent a floating point value (16-bit, 32-bit, 64-bit) to get the model size in terms of bits.

For binary networks we compute the same calculation with a modification - rather than the convolution weights being represented as a floating point tensor of size: (out channel, in channel, kernel size), we represent the tensor as bit tensors of (out channel, in channel, kernel size) and a floating point vector of (out channel) size.

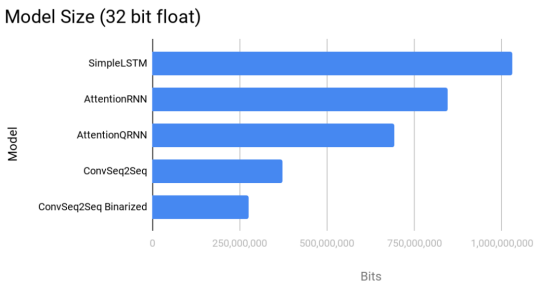


Figure 3: Model Size in Bits

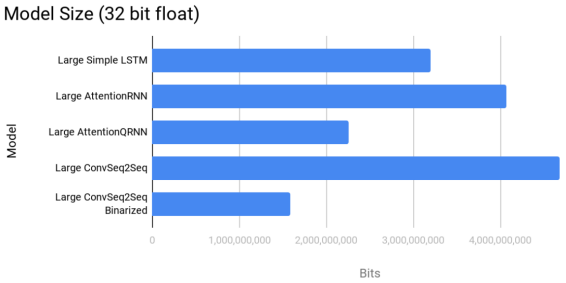


Figure 4: Large Models Size in Bits

Substituting this calculation for the one above, we get the results in Figures 3 and 4.

6.2.2 Calculating CPU/GPU Statistics

We infer the the CPU and GPU usage of a model by observing the process training the model. We used linux commands to get the values:

- ps -ux for CPU usage
- gpustat for GPU usage

To ensure the best possible accuracy and consistency, we run one model on the system at a time. Each model also uses the same hyperparameters as well. We see their performances in Figures 5 and 6.

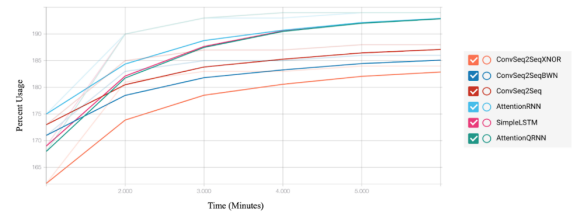


Figure 5: CPU Percentage

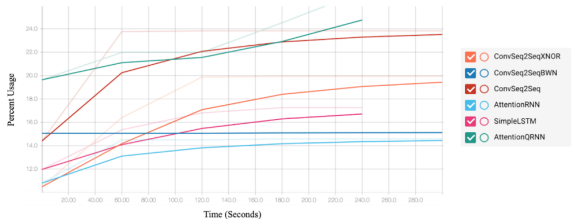


Figure 6: GPU Percentage

Note: We run these statistics on model training rather than model inference, since our framework implements a $O(n^2)$ decoder for the convolutional models and a $O(n)$ decoder for the RNN models. While we would have liked to compare on model inference, the time complexity difference of the algorithms led us to believe that computing training seemed like a more consistent option to measure system statistics. (We ensure that the models being trained have the same batch size and vocabulary features).

7 Conclusion

In this project we analyze the performance of various different neural architectures for machine translation on both GPU and CPU . In addition, we compute size (in bits) of each architecture to show that binarizing a translation network can result in a much smaller model size, with minor reductions in translation performance. The results of the project also show that translation models can become much quicker with Xnor convolutions, at a higher price of accuracy.

In potential future work on this project, we hope to examine how binarized embedding layers and linear layers can affect these models as well as the binarized convolutional layers. We also did not implement XNOR or binary weight convolutions, which doesnt allow us to analyze the speed increase of these networks. We hope to analyze that next as well.

References

Papers

1. XNOR Net: https://arxiv.org/abs/1603.05279
2. Multi bit quantization networks: https://arxiv.org/pdf/1802.00150.pdf
3. Binarized LSTM Language Model: http://aclweb.org/anthology/N18-1192
4. Fair Seq Convolutinal Sequence Learning: https://arxiv.org/pdf/1705.03122.pdf
5. Quasi Recurrent Networks: https://arxiv.org/abs/1611.01576
6. WMT 14 Translation Task: http://www.statmt.org/wmt14/translation-task.html
7. Attention is all you need: https://arxiv.org/abs/1706.03762
8. Imagination improves multimodal translation: https://arxiv.org/abs/1705.04350

9. Multi30k dataset:
https://arxiv.org/abs/1605.00459

10. IWSLT:
http://workshop2017.iwslt.org/downloads/O1-2-Paper.pdf

11. A Systematic Comparison of Smoothing Techniques for Sentence-Level BLEU:
http://acl2014.org/acl2014/W14-33/pdf/W14-3346.pdf

Githubs and Code links

1. Pytorch MT Seq2Seq:
https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

2. XNOR-net AI2:
https://github.com/allenai/XNOR-Net

3. Annotated Transformer (Harvard NLP):
http://nlp.seas.harvard.edu/2018/04/03/attention.html

4. Salesforce QRNN Pytorch:
https://github.com/salesforce/pytorch-qrnn

5. Fair Seq: https://github.com/pytorch/fairseq

6. Torchtext: https://github.com/pytorch/text

7. XNOR NET Pytorch:
https://github.com/jiecaoyu/XNOR-Net-PyTorch

Appendix A: Model Translation Performance on Multi30k

Appendix B: Model Size in Bits

Model	**Valid BLEU**	Valid BLEU Smooth	Valid Perplexity	Train Loss
Simple LSTM	**5.46**	21.3	17.75962	2.62
Attention RNN	**20.2**	32.6	7.14925	1.15
Attention QRNN	**14.8**	28.4	9.54268	1.41
ConvS2S	**28.6**	38.6	5.14285	1.07
ConvS2S BWN	**25.1**	35.7	6.69483	1.71
ConvS2S XNOR	**20.9**	32.5	8.17928	2.06

Table 1: Model Translation Performance on Multi30k

Model	16 bit float	32 bit float	64 bit float
SimpleLSTM	515,043,648	1,030,087,296	2,060,174,592
AttentionRNN	422,547,776	845,095,552	1,690,191,104
AttentionQRNN	346,952,000	693,904,000	1,387,808,000
ConvSeq2Seq	186,393,216	372,786,432	745,572,864
ConvSeq2Seq Binarized	139,207,296	275,268,864	547,392,000
Large Simple LSTM	1,600,352,576	3,200,705,152	6,401,410,304
Large AttentionRNN	2,036,707,648	4,073,415,296	8,146,830,592
Large AttentionQRNN	1,130,148,160	2,260,296,320	4,520,592,640
Large ConvSeq2Seq	2,343,793,280	4,687,586,560	9,375,173,120
Large ConvSeq2Seq Binarized	841,757,312	1,583,375,616	3,066,612,224

Table 2: Model Size in Bits