Faster Neural Machine Translation Inference

Anonymous ACL submission

Abstract

Deep learning models are widely deployed for machine translation. In comparison on many other deep learning models, there are two characterisitics of machine translation which have not been adequately addressed by most software implementations, leading to slow inference speed. Firstly, as opposed to standard binary class models, machine translation models are used to discriminate between a large number of classes corresponding to the output vocabulary. Secondly, rather than a single label, the output from machine translation models is a sequence of class labels that makes up the words in the target sentence. The sentence lengths are unpredictable, leading to inefficiencies during batch processing. We provide solutions for these issues which together, increase batched inference speed by up to ??? on modern GPUs without affecting model quality. For applications where the use of maxi-batching introduces unacceptable delays in response time, our work speed up inference speed by up to ???. Our work is applicable to other language generation tasks and beyond.

1 Introduction

We will examine two areas that are critical to fast NMT inference where the models differ significantly from other deep-learning models. We believe the optimizations of these models have been overlooked by the general deep-learning community.

Firstly, the number of classes in many deep-learning applications is small. However, the number of classes in NMT models is typically in the tens or hundreds of thousands, corresponding to the vocabulary size of the output language. For example, (Sennrich et al., 2016) experimented with target vocabulary sizes of 60,000 and 90,000 subword units. This makes the output layer of NMT models very computationally expensive. Figure 1 shows the breakdown of amount of time during translation our NMT system; nearly 70% of the time is involved in the output layer. We will look at optimizations which explicitly target the output layer.

Secondly, the use of mini-batching is critical for fast model inference. However, mini-batching does not take into acccount variable sentence lengths which decrease the ac-

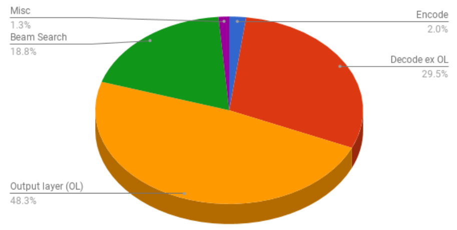

Figure 1: Proportion of time spent during translation (Europarl test set)

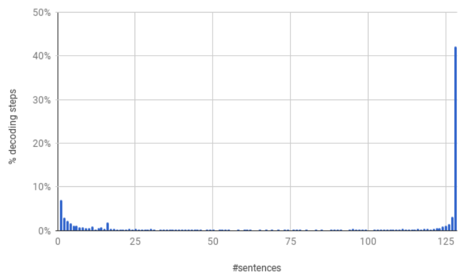

Figure 2: Actual batchsize during decoding (Europarl test set)

tual number of input or output sentences that are processed in parallel, negating the benefit of mini-batching. This can be partially reduced in the encoding with *maxi-batching*, i.e. pre-sorting sentences by length before creating mini-batches with similar length source sentences. However, maxi-batching can introduce unacceptable delay in response for online applications as the input are not processed in the order they came. It is also not appropriate for applications where only the output lengths varies, for example, image captioning.

Even with sequence-to-sequence NMT models, target sentence lengths will still differ even for similar length inputs, compromising decoding performance. Figure 2 shows the actual batch size during decoding with a maximum batch size of 128; the batch was full in only 42% of decoding iterations.

We will propose an alternative batching algorithm to increase the actual batch size during decoding without the problems associated with mini-batching and maxi-batching.

We base our work on the sequence-to-sequence model of (Cho et al., 2014). We also choose to focus on the using NMT inference on GPUs.

2 Prior Work

REDO

Deep learning models have been successfully used on many machine learning task in recent years in areas as diverse as computer vision and natural language processing. This success have been followed by the rush to put these model into production. However, the computational resources required in order to run these models have ben chanllenging. One possible solution involves creating faster models that can approximate the original model (Kim and Rush, 2016). Another solution is tackling specific computationally intensive functions by approximating them (??? softmax).

Most current neural MT models follow an encoder-decoder architecture. The encoder calculates the word and sentence embeddings while the decoder generates the output sentence. The architecture within the encoder is still a subject of much research. (Kalchbrenner and Blunsom, 2013) used a convolution to encode the input and a recurrent neural network (RNN) for the decoder. RNNs was used for both encoding and decoding in (??? who used RNN 1st). (Sutskever et al., 2014) significantly improved translation quality by using LSTM was used in each RNN node. (Cho et al., 2014) used GRU that has the required properties of LSTM but is computationally cheaper. (Bahdanau et al., 2014) added attention model which improved translation at a cost of slower speed.

There has been recent attempts to move away from the sequence-to-sequence models of RNN encoder-decoder. Some justify their architecture on faster inference as well as better translation results. (Vaswani et al., 2017) and (??? course-to-fine) has been proposed as a fast alternatives to RNN-based models as it is possible to process more of the units in parallel.

It has been noticed that half precision arithmetic can be used for deep learning model without significan loss of model quality (Micikevicius et al., 2017). Other solutions include specialized hardware, the most popular being graphical processing units (GPU) but other hardware such as custom processors TPU (Jouppi et al., 2017), FPGA (Lacey et al., 2016) have been used.

Many of these optimization are general purpose improvements for deep learning models, regardless of the task it is being used in. (Devlin, 2017) is a noticeable machhine translation-specific optimization which describe the speed improvements that can be obtained by techniques such as the use of half-precision, model changes and pre-computation.

3 Proposal

3.1 Softmax and Beam Search Fusion

The output layer of most deep learning models consist of the following steps

1. multiplication of the input with the weight matrix $p = wx$
2. addition of a bias term to the resulting scores $p = p + b$
3. applying the activation function, most commonly softmax $p_i = \exp(p_i)/\sum \exp(p_i)$
4. a search for the best output class, and probability if necessary $\text{argmax}_i \, p_i$

In models with a small number of classes such as binary classification, the computational effort required is trivial and fast. However, this is not the case for large number of classes such as those found in NMT models.

We shall leave step 1 for future work and focus on the last three steps, the outline for which are shown in Figures 3 to 5.

```
Require: vector $p$, bias vector $b$
  for all $p_i$ in $p$ do
    $p_i \leftarrow p_i + b_i$
  end for
```

Figure 3: Add Bias Term

```
Require: vector $p$
  {calculate max for softmax stability}
  $max \leftarrow -\infty$
  for all $p_i$ in $p$ do
    if $p_i > max$ then
      $max \leftarrow p_i$
    end if
  end for
  {calculate denominator}
  $sum \leftarrow 0$
  for all $p_i$ in $p$ do
    $sum \leftarrow sum + \exp(p_i - max)$
  end for
  {calculate softmax}
  for all $p_i$ in $p$ do
    $p_i \leftarrow \frac{\exp(p_i) - max}{sum}$
  end for
  return $p$
```

Figure 4: Calculate softmax

```
Require: softmax vector $p$
  $max \leftarrow -\infty$
  for all $p_i$ in $p$ do
    if $p_i > max$ then
      $max \leftarrow p_i$
      $best \leftarrow i$
    end if
  end for
  return $max, best$
```

Figure 5: Find best

As can be seen, the operations iterate over the matrix p five times - once to add the bias, three times to calculate the softmax, and once to search for the best class. We shall use a popular method, kernel fusion (Guevara et al., 2009), to optimize the output layer.

Secondly, we make use of the fact that softmax and exp are monotonic functions therefore we can move the search for the best class from Figure 5 to Figure 4 while max is being sought.

Thirdly, we are only interested in the best class. Since the best class is known, we can avoid calculating softmax for all classes. The outline of the our function is shown in Figure 6.

In fact, we are usually only interested in the best class during inference, not the probability. Therefore, we can skip the second iteration over p in Figure 6 and avoid computing the softmax altogether.

It has been shown (Koehn and Knowles, 2017) that using beam search to use the n-best number of classes, rather than just the best class, improves translation quality. This is a simple extension to the algorithm of Figure 6. Unlike the 1-best case, however, the softmax calculation cannot be skipped as the denominator differs for each input.

```
Require: vector $p$, bias vector $b$
  {add bias, calculate $max$ & $argmax$}
  $max \leftarrow -\infty$
  for all $p_i$ in $p$ do
    if $p_i + b_i > max$ then
      $max \leftarrow p_i + b_i$
      $best \leftarrow i$
    end if
  end for
  {calculate denominator}
  $sum \leftarrow 0$
  for all $p_i$ in $p$ do
    if $p_i > max$ then
      $sum \leftarrow sum + \exp(p_i - max)$
    end if
  end for
  return $\frac{1}{sum}, best$
```

Figure 6: Fused softmax and argmax

3.2 Top-up Batching

The standard mini-batching algorithm is outlined in Figure 7.

This algorithm encode the sentences for a batch, followed by decoding the batch. The decoding stop once all sentences in the batch are completed. This is a potential inefficiency as the number of remaining sentences may not be optimal.

We will focus on decoding as this is the more compute-intensive step, and issues with differing sentence sizes in encoding can partly be ameriorated by maxi-batching.

Our proposed top-up batching algorithm encode and decode asynchronously. The encoding step, Figure 8, is similar to the main loop of the standard algorithm but the results are added to a queue to be consumed by the decoding step later.

Rather than decoding the same batch until all

```
while more input do
  Create batch
  Encode
  while batch is not empty do
    Decode batch
    for each sentence in batch do
      if translation is complete then
        Remove sentence from batch
      end if
    end for
  end while
end while
```

Figure 7: Mini-batching

```
while more input do
  Create encoding batch
  Encode
  Add to queue
end while
```

Figure 8: Encoding for top-up batching

sentences in the batch are completed, the decoding step processing the same batch continuously. New sentences are added to the batch as old sentences completes, Figure 9.

```
create decoding batch $b$ from queue
while $b$ is not empty do
  Decode $b$
  Replace completed sentences with new sentence from queue
end while
```

Figure 9: Decoding for top-up batching

4 Experimental Setup

We trained a sequence-to-sequence, encoder-decoder NMT system similar to that described in (Sennrich et al., 2016). This uses recurrent neural networks with gated recurrent units. The input and output vocabulary size were both set to 85,000 subwords using byte-pair encoding (BPE) to adjust the vocabulary to the desired size. The hidden layer dimensions was set to 512.

For inference, we used and extend Amun (Junczys-Dowmunt et al., 2016), the fastest open-source inference engine we are aware of for the model used in this paper. We used a beam size of 5, mini-batch of 128 sentences and maxi-batch 1280, unless otherwise stated.

The hardware used in all experiments was an Nvidia GTX 1060 GPU on a host containing 8 Intel hypercores running at 2.8Ghz, 16GB RAM and SSD hard drive.

Our training data consisted of the German-English parallel sentences from the Europarl corpus (Koehn, 2005). To test inference speed, we used two test sets with differing characteristics:

1. a subset of the Europarl training data, which contains mostly long sentences, and is, of course, in the same domain as the training data

2. a subset of the German-English data from the Open-Subtitles corpus, consisting of mostly short, out-of-domain sentences.

Table 1 gives further details of the test sets.

	Europarl	OpenSubtitles
# sentences	30,000	50,000
# subwords	787,908	467,654
Avg subwords/sent	26.3	9.4

Table 1: Test sets

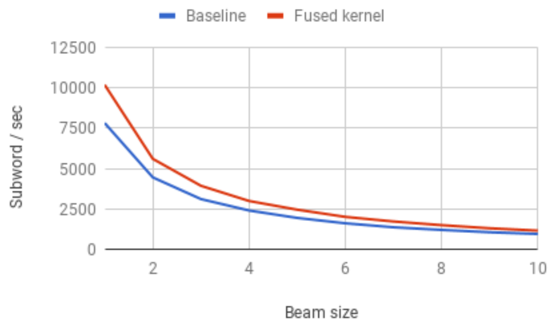

Figure 10: Translation speed for Europarl test set when using the fused kernel

5 Results

5.1 Softmax and Beam Search Fusion

Fusing the softmax and beam search increase the speed of the output layer by 43% for beam size 1, decreasing to 25% for a beam of 10 when translating the Europarl dataset. This led to an overall increase in translation speed of up to 23%, Figure 10. Translation speed improved by up to 41% when translating the Opensubtitles datset, Figure 11. Compared to translating the Europarl test st, Figure 1 in Section 1, the amount of time taken by the output layer dominates translation time for the OpenSubtitles test set, Figure 12.

5.2 Top-up Batching

After some experimentation, we decided to fully fill the decoding batch only when it is at least half empty, rather than whenever a sentence has completed.

The top-up batching and maxi-batching have similar goals of maximizing efficiency when translating batches of different lengths. Therefore, using both methods together gives limited gains, in fact, using top-up batching with maxi-batch slows of between 2% to 18% when trans-

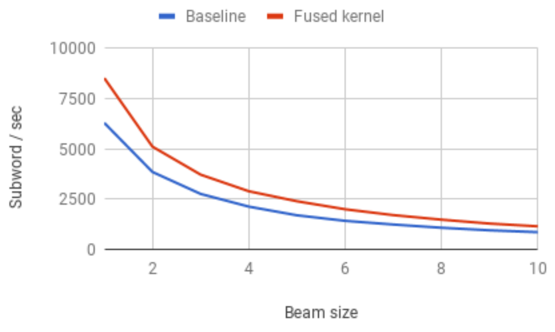

Figure 11: Translation speed for Opensubtitles test set when using the fused kernel

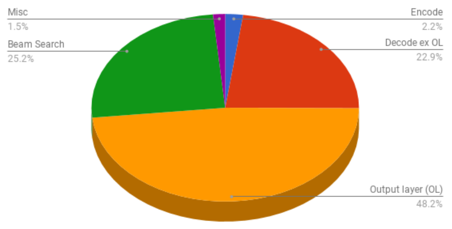

Figure 12: Proportion of time spent during translation (Opensubtitles test set)

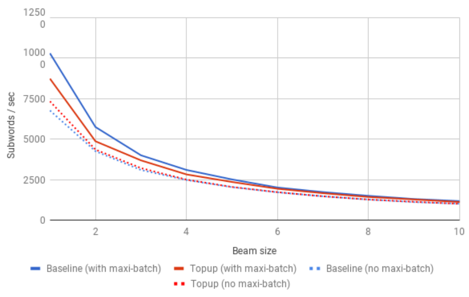

Figure 13: Translation speed for Europarl test set using top-up batching

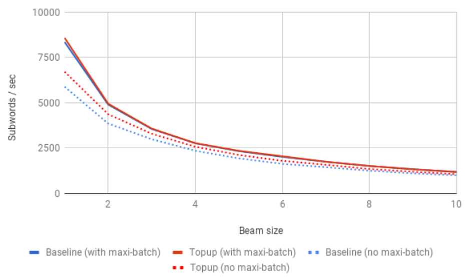

Figure 14: Translation speed for OpenSubtitles test set using top-up batching

lating the Europarl test set due to the overhead of using the algorithm, Figure 13. When maxi-batching is inappropriate, using top-up batching alone matches the performance of maxi-batching, even being slightly faster when a small beam is used.

The results are better when translating the OpenSubtitles test set, Figure 14. The top-up batching does not harm performance when used with maxi-batching, even helping a little for small beams. However, it increases translation speed by up to 12% when used alone.

6 Conclusion

References

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics.

Devlin, J. (2017). Sharp models on dull hardware: Fast and accurate neural machine translation decoding on the CPU. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 2820–2825.

Guevara, M., Gregg, C., Hazelwood, K. M., and Skadron, K. (2009). Enabling task parallelism in the cuda scheduler.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, R. C., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., and Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit. *CoRR*, abs/1704.04760.

Junczys-Dowmunt, M., Dwojak, T., and Hoang, H. (2016). Is neural machine translation ready for deployment? a case study on 30 translation directions. In *Proceedings of the 9th International Workshop on Spoken Language Translation (IWSLT)*, Seattle, WA.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. Seattle. Association for Computational Linguistics.

Kim, Y. and Rush, A. M. (2016). Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 1317–1327.

Koehn, P. (2005). Europarl: A parallel corpus for statistical machine translation. In *Proceedings of the Tenth Machine Translation Summit (MT Summit X)*, Phuket, Thailand.

Koehn, P. and Knowles, R. (2017). Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39, Vancouver. Association for Computational Linguistics.

Lacey, G., Taylor, G. W., and Areibi, S. (2016). Deep learning on fpgas: Past, present, and future. *CoRR*, abs/1602.04283.

Micikevicius, P., Narang, S., Alben, J., Diamos, G. F., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. (2017). Mixed precision training. *CoRR*, abs/1710.03740.

Sennrich, R., Haddow, B., and Birch, A. (2016). Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.