Faster Neural Machine Translation Inference

Anonymous ACL submission

Abstract

As neural machine translation (NMT) models have become the new state-of-the-art, the challenge is to make their deployment efficient and economical. We have identified two areas where typical NMT models differ significantly from other deep-learning models that impact on their inference speed. Firstly, NMT models usually contains a large number of output 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 or sub-words in the target sentence. The sentence lengths are unpredictable, leading to inefficiencies during batch processing. We provide solutions for each of these issues which can increase batched inference speed by up to 57% on modern GPUs without affecting model quality.

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 sub-word 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 account variable sentence lengths which decrease the actual 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

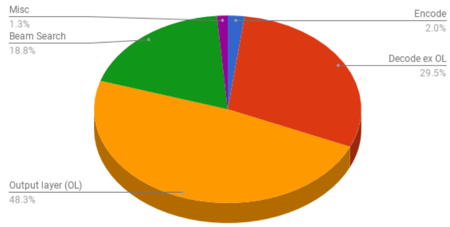

Figure 1: Proportion of time spent during translation (Europarl test set)

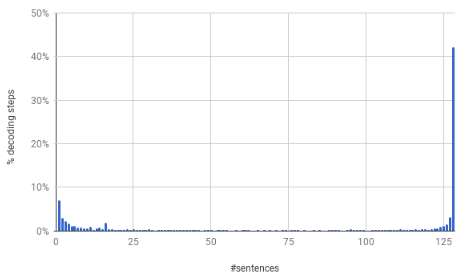

Figure 2: Actual batch size during decoding (Europarl test set)

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

Deep-learning models has 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. The computational resources required in order to run these models has been challenging but, fortunately, there has been a lot work to meet this challenge.

Hardware accelerators such as GPUs are very popular but other specialized hardware such as custom processors (Jouppi et al., 2017) and FPGA (Lacey et al., 2016) have been used. Hardware-supported reduced precision is also an ideal way to speed up model inference (Micikevicius et al., 2017).

Simpler, faster models have been created that are as good, or almost as good, as more slower, more complex models (Bahdanau et al., 2014). Research have also gone into smaller, faster models that can approximate the original, slower, bigger model (Kim and Rush, 2016). Some models have been justified as more suited for the parallel architecture of GPUs (Vaswani et al., 2017).

The speed of the softmax layer when used with large vocabularies have been looked at in (Grave et al., 2016) but there have been many other attempts at faster softmax, for example (Mikolov et al., 2013), (Zoph et al., 2016).

There has been surprisingly little work on batching algorithms, considering its critical importance for efficient deep-learning training and inference. (Neubig et al., 2017) describe an novel batching algorithm, however, its aim is to alleviate the burden on developers in batching of new models, not faster batching.

Our paper follows most closely on from (Devlin, 2017) which achieved faster NMT inference mainly by novel implementation of an existing model. However, we differ by focusing on GPU implementation, and the output layer and batching algorithm which were not touched on by the previous work.

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.

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.

```
procedure ADDBIAS(vector p, bias vector b)
    for all p_i in p do
        p_i ← p_i + b_i
    end for
end procedure

procedure SOFTMAX(vector p)
                        ▷ calculate max for softmax stability
    max ← −∞
    for all p_i in p do
        if p_i > max then
            max ← p_i
        end if
    end for
                                    ▷ calculate denominator
    sum ← 0
    for all p_i in p do
        sum ← sum + exp(p_i − max)
    end for
                                      ▷ calculate softmax
    for all p_i in p do
        p_i ← (exp(p_i) − max) / sum
    end for
end procedure

procedure FINDBEST(softmax vector p)
    max ← −∞
    for all p_i in p do
        if p_i > max then
            max ← p_i
            best ← i
        end if
    end for
end procedure
```

Figure 3: Original Softmax and Beam Search Algorithm

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 ?? to Figure ?? 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 4.

procedure FUSED KERNEL(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 $\frac{1}{sum}, best$
end procedure

Figure 4: Fused softmax and argmax

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 4 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 4.

procedure FIND-1-BEST(activation vector p, bias vector b)
  $max \leftarrow -\infty$
  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
  return $best$
end procedure

Figure 5: Find 1 best

Unlike the 1-best case, however, the softmax calculation cannot be skipped as the denominator differs for each input.

3.2 Top-up Batching

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

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 ameliorated by maxi-batching.

Our proposed top-up batching algorithm encode and decode asynchronously. The encoding step, Figure ??, 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 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 **??**.

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 sub-words 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
# sub-words	787,908	467,654
Avg sub-words/sent	26.3	9.4
Std dev subwords/sent	14.9	6.1

Table 1: Test sets

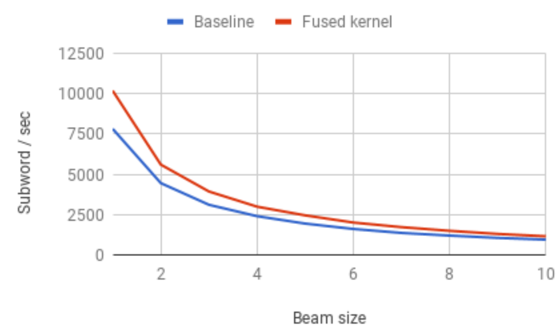

Figure 6: Speed using the fused kernel (Europarl)

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 6. Translation speed improved by up to 41% when translating the OpenSubtitles dataset, Figure 7. 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 8.

5.2 Top-up Batching

After some experimentation, we decided to top-up the decoding batch only when it is at least half empty, rather than whenever a sentence has

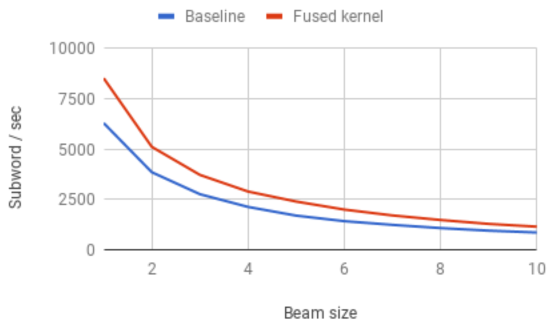

Figure 7: Speed using the fused kernel (OpenSubtitles)

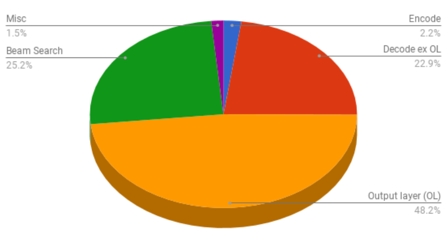

Figure 8: Proportion of time spent during translation (OpenSubtitles)

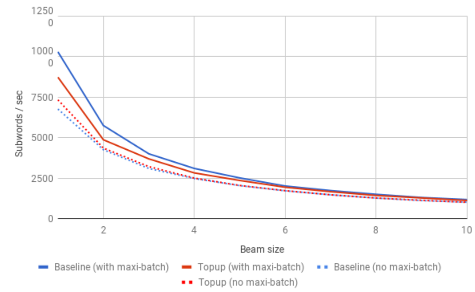

Figure 9: Speed using top-up batching (Europarl)

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 translating the Europarl test set due to the overhead of using the algorithm, Figure 9. 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 10. The top-up batching does not harm performance when used with maxi-batching, even helping a little for small beams. However, top-up batching increases translation speed by up to 12% when used alone.

The gain from top-up batching is partially dependent on how much time the standard mini-batching algorithm spend decoding with a very small number of sentences as this is does not fully utilize the GPU cores. From Figure 11, 20% of the decoding iterations have less than 8 sentences remaining in the batch when translating the Europarl test test with maxi-

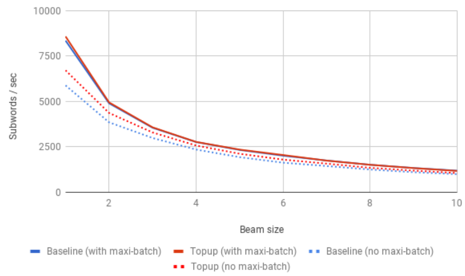

Figure 10: Speed using top-up batching (OpenSubtitles)

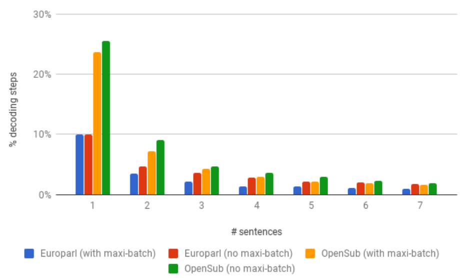

Figure 11: Actual batch size during decoding

batching. This figure is 50% for the OpenSubtitles test set with no maxi-batching. The OpenSubtitles test set has a higher variance of sentence lengths relative to its average sentence length, which forces the mini-batch algorithm to continue decoding with a small number of sentences while the other sentences have already completed.

5.3 Cummulative Results

Using both the fused kernel and top-up batching to translate led to a cummalative speed improvement of up to 57% and 34% for the OpenSubtitles and Europarl test set, respectively, when no maxi-batching is used, Figure 12 and Figure 13. With maxi-batching, the speed was up to 41% and 21%.

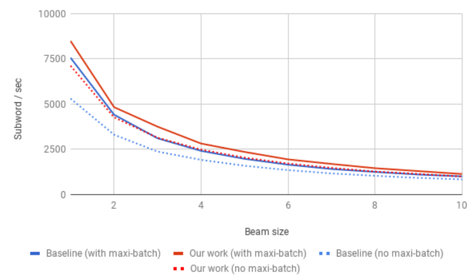

Figure 12: Cummulative results (Europarl

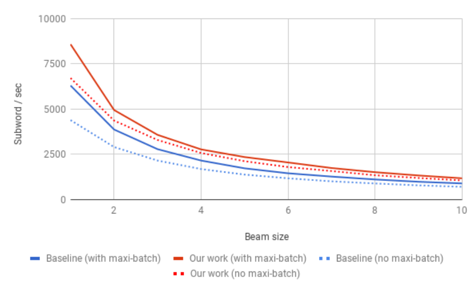

Figure 13: Cummulative results (OpenSubtitles

6 Conclusion and Future Work

We have presented two methods for faster deep-learning inference, targeted at neural machine translation.

The first method focused on output layer of the neural network which accounts for a large part of the running time of the NMT model. By fusing the output layer with the beam search, we are able to increase translation speed by up to 41%.

The second method replaces the mini-batching algorithm with one that avoids decoding with a small number of sentences, maximizing the parallel processing potential of GPUs. For certain scenarios, this increases translating speed by up to 12%.

For future work, we would like to apply our optimization for other NLP and deep-learning tasks. We are also interested in further optimization of the output layer in NMT, specifically the matrix multiplication which still takes up a significant proportion of the translation time.

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.

Grave, E., Joulin, A., Cissé, M., Grangier, D., and Jégou, H. (2016). Efficient softmax approximation for gpus. *CoRR*, abs/1609.04309.

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.

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.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Neubig, G., Goldberg, Y., and Dyer, C. (2017). On-the-fly operation batching in dynamic computation graphs.

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.

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.

Zoph, B., Vaswani, A., May, J., and Knight, K. (2016). Simple, fast noise-contrastive estimation for large rnn vocabularies.