Faster Neural Machine Translation Inference

Hieu Hoang hieu@hoang.co.uk

Tomasz Dwojak ???
Adam Mickiewicz University

Kenneth Heafield ???
University of Edinburgh, Scotland

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 look at two areas that are critical to fast NMT inference where the models in NMT differ significantly from those in other applications. These areas have been overlooked by the general deep-learning community, we aim to improve their efficient for NMT-specific tasks.

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, the best German-English NMT model at WMT14 contains 85,000 classes (Rico ???). This makes the output layer of NMT models very computationally expensive. ??? shows the breakdown of amount of time during translation using RNN model similar to (Rico ???); over 40% of the time is involved in calculating the activation, softmax and the beam search. We will look at optimizations which explicitly target the output layer.

Secondly, mini-batching is often used to increase the speed of deep-learning applications, including NMT systems. (??? graph of batch v. speed) shows using batching can increase speed by 17 times. However, mini-batching does not take into acccount the variable length of NMT inputs and outputs, creating computational issues and efficiency challenges. The computational issues are often solved by masking unwanted calculations. Translation speed can often be improved by employing max-batching whereby the input set is pre-sorted by sentence length before mini-batching, creating mini-batches with similar length input sentences. However, the target sentence lengths will still differ even for similar length inputs but the standard

mini-batchin algorithm must continue to process the batch until all target sentences have been completed. (??? batchsize size v. iterations) shows the number of sentences still being processed for each decoding iteration, for a specific batch. The number of sentences still to be processed decreases, reducing the effectiveness of mini-batching. We will propose an alternative to the mini-batching algorithm.

We based our model on the sequence-to-sequence model of (??? Cho) for machine translation models, but unlike (??? Devlin), we avoid solutions for the specific model. Therefore, our solution should be applicable to other models, architectures and task which have the similar characteristics. We envisage that our solutions would be of value to models used in text summarization, chatbot or image captioning. We also choose to focus on the use of GPU, rather than CPU as pursued in (??? Devlin).

2 Prior Work

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 (??? Student-Teacher model). 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. (??? Kalshammer) 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). (???) significantly improved translation quality by using LSTM was used in each RNN node. (??? Cho) used GRU that has the required properties of LSTM but is computationally cheaper. (??? Badhannau) 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. (??? Attention is all you need) 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 (???). Other solutions include specialized hardware, the most popular being graphical processing units (GPU) but other hardware such as custom processors (??? TPU), FPGA (??? Intel) 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) 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. activation - multiplication of the input with the weight matrix for that layer
2. addition of a bias term to the resulting scores

3. activation function - in the output layers, this is commonly softmax or a variant, eg. log-softmax.
4. a search for the argmax output class, and probability is necessary.

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 the matrix multiplication for the next proposal and future work, and concentrate on the last three steps, the outline of which are shown in Figures 1 to 3.

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

Figure 1: Add Bias Term

Require: activation 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**
    $sum \leftarrow sum + e^{p_i}$
  end for
  {calculate softmax}
  for all p_i in p **do**
    $p_i \leftarrow \frac{e^{p_i}}{sum}$
  end for
  return p

Figure 2: Calculate softmax

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 fused the three functions and combine the matrix access into two iterations, reducing memory bandwidth. Kernel fusion is a well practised optimization technique (???).

Secondly, we make use of the fact that softmax and exp are monotonic functions therefore we can avoid the search for the argmax in Figure 3 as it will be the same as that in Figure 2.

Thirdly, we are only interested in the probability of the best class. Since we have precalculated the best class in Figure 2, we shall only calculate softmax for this one class, avoiding the computational and memory bandwith of computing the probability for all classes.

In fact, we are usually only interested in the argmax class during inference, not the probability. Therefore, by using the argmax calculated in Figure 2, we can avoid computing the softmax altogether. The outline of the our function is shown in Figure 4.

```
Require: softmax vector p
  max ← −∞
  for all p_i in p do
    if p_i > max then
      max ← p_i
      argmax ← i
    end if
  end for
  return max, argmax
```

Figure 3: Find best

```
  while more input do
    Create encoding batch
  end while
```

Figure 4: Fused Kernel

The algorithm was extended from argmax to beam search for n-best classes.

3.2 Top-up Batching

The standard mini-batching algorithm is outlined in Figure 5.

```
  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 5: Mini-batching

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 6, 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 7.

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

Figure 6: Encoding for top-up batching

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 7: Decoding for top-up batching

4 Experimental Setup

5 Results

6 Conclusion

Acknowledgments

This work is sponsored by the Air Force Research Laboratory, prime contract FA8650-11-C-6160. The views and conclusions contained in this document are those of the authors and should not be interpreted as representative of the official policies, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

References