Faster Neural Machine Translation Inference

Anonymous ACL submission

Abstract

We present two optimizations in areas where typical NMT models differ significantly from other deep-learning models. Firstly, we optimize the output layer which takes a significant amount of time for NMT models due to the large number of output classes, corresponding to the output vocabulary. Secondly, we present a novel batching algorithm which takes into account the differing output sentence lengths which leads to inefficiencies in current batch algorithms. Together, we increase inference speed by up to 57% on modern GPUs without affecting model quality.

1 Introduction

The number of classes in NMT models is typically in the tens or hundreds of thousands, 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 computationally expensive. Figure 1 shows the breakdown of amount of time during translation our NMT system; 73% of the time is involved in the output layer or beam search. Our first proposal will explicitly target this computation.

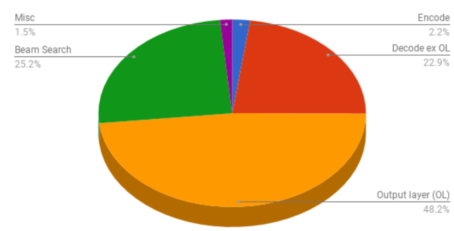

Figure 1: Proportion of time spent during translation

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 unacceptable delays in response time as inputs are not processed in the order they came, a particular concern for online applications.

Even with sequence-to-sequence NMT models, target sentence lengths will still differ even

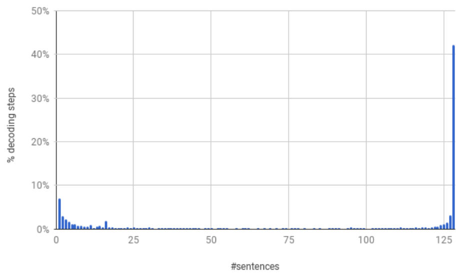

Figure 2: Actual batch size during decoding (Europarl test set)

for similar length inputs. 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.

2 Proposal

2.1 Softmax and Beam Search Fusion

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

1. multiplication of the weight matrix with the input vector $p = wx$
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 beam search for the best (or n-best) output classes $\operatorname{argmax}_i p_i$

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

We focus on the last three steps, the outline for which are shown in Algorithm 1. For brevity, we show the algorithm for 1-best, a beam search of the n-bests is a simple extension of this.

As can be seen, the matrix p is iterated over five times - once to add the bias, three times to calculate the softmax, and once to search for the best classes. We propose fusing the three functions into one kernel, a popular optimization technique (Guevara et al., 2009), making use of the following observations.

Firstly, softmax and $\exp$ are monotonic functions, therefore, we can move the search for the best class from FIND-BEST to SOFTMAX, at the start of the kernel.

Secondly, we are only interested in the probabilities of the best classes. Since they are now known at the start of the kernel, we compute softmax only for those classes. The outline of the our function is shown in Algorithm 2.

In fact, we are usually only interested in the best class during inference, not the probability. Since we now compute the best class before the softmax, we can skip softmax altogether. This is only possible for beam size 1; the comparison between softmax probabilities is required for larger beam sizes as the denominators are different for different hypotheses.

2.2 Top-up Batching

The standard mini-batching algorithm, Algorithm 3, encode the sentences for a batch, followed by decoding the batch. Decoding stop only once all sentences in the batch are completed.

Our proposal focuses on decoding as this is the more compute-intensive task, encoding and decoding asynchronously. The encoding step, Algorithm 4, is similar to the main loop of the standard algorithm but the results are added to

Algorithm 1 Original softmax and beam Search Algorithm

```
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 Find-Best(vector p)
    max ← −∞
    for all p_i in p do
        if p_i > max then
            max ← p_i
            best ← i
        end if
    end for
    return max, best
end procedure
```

Algorithm 2 Fused softmax and beam search

```
procedure Fused-Kernel(vector p, bias vector b)
                     ▷ add bias, calculate max & argmax
    max ← −∞
    for all p_i in p do
        if p_i + b_i > max then
            max ← p_i + b_i
            best ← i
        end if
    end for
                                ▷ calculate denominator
    sum ← 0
    for all p_i in p do
        if p_i > max then
            sum ← sum + exp(p_i − max)
        end if
    end for
    return 1/sum, best
end procedure
```

Algorithm 3 Mini-batching

```
procedure Mini-Batching
    while more input do
        Create batch b
        Encode(b)
        while batch is not empty do
            Decode(b)
            for all sentence s in b do
                if trans(s) is complete then
                    Remove s from b
                end if
            end for
        end while
    end while
end procedure
```

a queue to be consumed by the decoding step.

Algorithm 4 Encoding for top-up batching

```
procedure ENCODE
    while more input do
        Create encoding batch b
        Encode(b)
        Add b to queue q
    end while
end procedure
```

Rather than decoding a batch until completion, new sentences are added to the batch as old sentences completes, Algorithm 5.

Algorithm 5 Decoding for top-up batching

```
procedure DECODE
    create batch b from queue q
    while b is not empty do
        Decode(b)
        for all sentence s in b do
            if trans(s) is complete then
                Replace s with s' from q
            end if
        end for
    end while
end procedure
```

3 Experimental Setup

We train 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 are both set to 85,000 sub-words using byte-pair encoding (BPE), the hidden layer dimensions is 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.

	OpenSubtitles
# sentences	50,000
# sub-words	467,654
Avg sub-words/sent	9.4
Std dev subwords/sent	6.1

Table 1: Test sets

	Baseline	Fused
Beam size 1		
Multiplication	21.47	21.94 (+2.2%)
Add bias	3.29	5.70 (-78.1%)
Softmax	8.83	
Beam search	13.91	
Beam size 5		
Multiplication	79.66	80.01 (+4.4%)
Add bias	14.95	42.91 (-64.4%)
Softmax	36.86	
Beam search	68.80	

Table 2: Time taken in sec (OpenSubtitles)

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.

We trained a German-English with data from the Europarl corpus (Koehn, 2005), our test set consisted of a subset of the Open-Subtitles corpus, Table 1.

4 Results

4.1 Softmax and Beam Search Fusion

Fusing the last 3 steps led to a substantial reduction in the time taken, especially for beam size of 1 where the softmax probability does not actually have to calculated, Table 2. This led to an overall increase in translation speed of up to 41%, Figure 3.

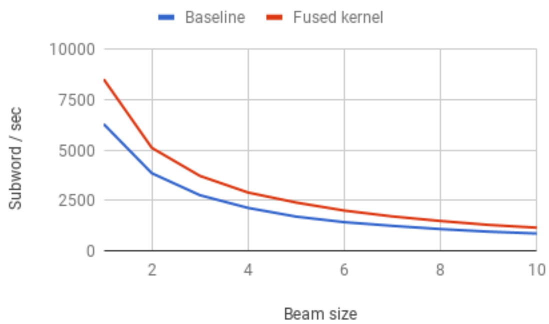

Figure 3: Speed using the fused kernel

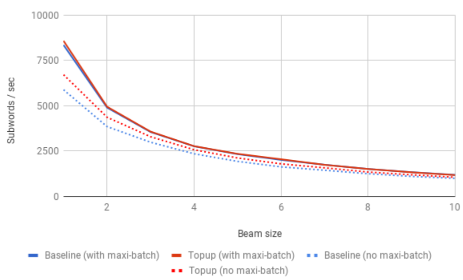

Figure 4: Speed using top-up batching

4.2 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 completed.

The top-up batching and maxi-batching have similar goals of maximizing efficiency when translating batches of different lengths. The top-up batching is of limited utility when used with maxi-batching but it increases translation speed by up to 12% when used alone, Figure 4.

4.3 Cummulative Results

Using both the fused kernel and top-up batching to translate led to a cummalative speed improvement of up to 57% when no maxi-batching is used, Figure 5. With maxi-batching, the speed was up to 41% faster.

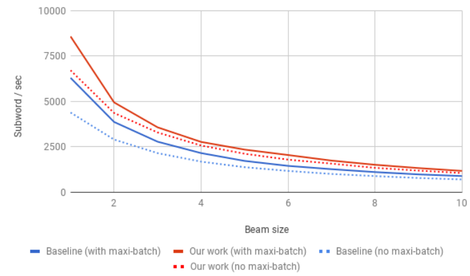

Figure 5: Cummulative results

5 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

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

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.

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

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.