# Focusing on Fast Neural Network Inference

Anonymous ACL submission

## Abstract

This paper describe the submissions to the efficiency track for GPUs by members of the University of Edinburgh, Adam Mickiewicz University, Tilde and University of Alicante. We focus on efficient implementation of the recurrent deep-learning model as implemented in Amun, the fast inference engine for neural machine translation. We improve the performance with an efficient mini-batching and maxi-batching algorithm. Translation speed was also reduced by fusing the softmax operation with the beam seach algorithm.

## 1 Introduction

As neural machine translation (NMT) models have become the new state-of-the-art, the challenge is to make their deployment efficient and economical. This is the challenge that this shared task is shining a spotlight on.

It is tempting to use a general purpose deep-learning toolkit such as Tensorflow or Pytorch to train and do the the inference where faster translation could be obtained by tuning parameters within the toolkit, and choosing fast models. We believe this is the approach most other submissions have taken.

We take an opposing approach by using and enhancing a custom inference engine, Amun, which is developed on the premise that fast neural network inference is an issue that deserves dedicated tools which are not compromised by other objectives such as training or support for multiple models. As well as delivering on the useful goal of fast inference, it can serve as a test-bed for novel ideas on neural network inference, and is useful as a means to explore the upper limit of the possible speed for a particular model and hardware. That is, Amun is an inference-only engine that supports a limited number of NMT models that put fast inference on modern GPU above all other considerations.

We submitted two systems to this year's shared task for the efficient translation on GPU. Our first submission was tailored to be as fast as possible while being above the baseline BLEU score. Our second submission trade some of the speed of first submission to return good quality translation.

We will describe below our experimental setup as well as the enhancements to Amun that we feel enabled it to achieve its outstanding speed.

## 2 Improvements

We desribe the main enhancements to Amun since the original 2016 publication.

### 2.1 Batching

The use of mini-batching is critical for fast model inference. The size of the batch is determined by the number of inputs sentences to the encoder in an encoder-decoder model. However, the number of batches during decoding can vary as some sentences have completed translating or the beam search add more hypotheses to the batch.

It is tempting to ignore these considerations, for example, by always decoding with a constant batch and beam size and ignoring hypotheses which are not needed but this comes at a cost of lower translation speed due to wasteful processing.

The Amun engine implements an efficient batching algorithm that takes into account the actual number of hypotheses that needs to be decoded at each decoding step, Figure 1.

**Algorithm 1** Mini-batching

**procedure** BATCHING(encoded records $i$)
    Create batch $b$ from $i$
    **while** $b$ is not empty **do**
        $b' \leftarrow$ DecodeAndBeamSearch($b$)
        $b \leftarrow \emptyset$
        **for all** hypo $h$ in $b'$ **do**
            **if** $h \neq </s>$ **then**
                Add $h$ to $b$
            **end if**
        **end for**
    **end while**
**end procedure**

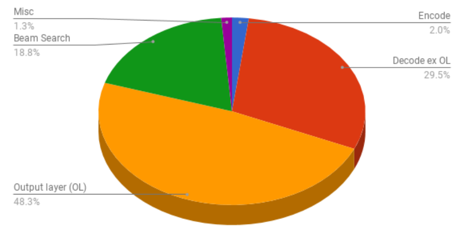


Figure 1: Proportion of time spent during translation (Europarl test set)

### 2.2 Softmax and Beam Search Fusion

Most NMT more predict a large number of classes in their output layer, corresponding to the number of words or subword units in their target 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 describe the steps to fuse the operations to improve efficiency.

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 $\mathrm{argmax}_i\, p_i$

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 leave step 1 for future work and focus on the last three steps, the outline for which are shown in Algorithm 2. 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.

Thirdly, the calculation of $max$ and $sum$ can be accomplished in one loop by adjusting $sum$ whenever a higher $max$ is found:

$$
\begin{aligned}
sum &= e^{a-max_b} + e^{b-max_b} \\
&= e^{a-max_a+\Delta} + e^{b-max_b} \\
&= e^{\Delta} e^{a-max_a} + e^{b-max_b}
\end{aligned}
$$

where $max_a$ is the original maximum value, $max_b$ is the new, higher maximum value, ie. $max_b > max_a$, and $\Delta = max_a - max_b$. The outline of our function is shown in Algorithm 3.

In fact, a well known optimization is to skip softmax altogether and calculate the argmax over the input vector, Algorithm 4. This is only possible for beam size 1 and when we are not interested in returning the softmax probabilities.

**Algorithm 2** 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 3** Fused softmax and beam search

**procedure** FUSED-KERNEL(vector $p$, bias vector $b$)
        ▷ add bias, calculate $max$ & $argmax$
    $max \leftarrow -\infty$
    $sum \leftarrow 0$
    **for all** $p_i$ in $p$ **do**
        $p'_i \leftarrow p_i + b_i$
        **if** $p'_i > max$ **then**
            $\Delta \leftarrow max - p'_i$
            $sum \leftarrow \Delta \times sum + 1$
            $max \leftarrow p'_i$
            $best \leftarrow i$
        **else**
            $sum \leftarrow sum + \exp(p'_i - max)$
        **end if**
    **end for**
    **return** $\frac{1}{sum}, best$
**end procedure**

**Algorithm 4** Find 1-best only

**procedure** FUSED-KERNEL-1-BEST(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**

These changes do not reduce the asymptotic runtime of our algorithm, which remains at $\mathcal{O}(|p|)$, but it does bring the number of iterations over $p$ down from 5 to 1. In practise, this has a significant affect on inference speed.

### 2.3 Half-Precision

////////////////////////////////////////////////////////////////

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. Target sentence lengths are unpredictable, leading to inefficiencies during batch processing. We provide solutions for each of these issues which together increase batched inference speed by up to 57% on modern GPUs without affecting model quality.

We 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.

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.

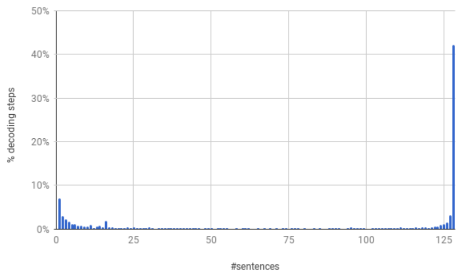


Figure 2: Actual batch size during decoding (Europarl test set)

## 3 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 slower, bigger models (Kim and Rush, 2016). Some models have been justified as more suited for the parallel architecture of GPUs (Vaswani et al., 2017) (Gehring 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 of batching models rather than 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, specifically the output layer, and the batching algorithm which was not touched on by the previous work. Our model is based on that of Cho et al. (2014) and Sennrich et al. (2016) but our work is applicable to other models and applications.

## 4 Proposal

### 4.1 Top-up Batching

The standard mini-batching algorithm is outlined in Algorithm 1.

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, Algorithm 5, 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.

**Algorithm 5** 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 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, Algorithm 6.

**Algorithm 6** 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
```

## 5 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.

| | 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

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.

## 6 Results

### 6.1 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 and 3. This led to an overall increase in translation speed of

| | Baseline | Fused |
|---|---|---|
| *Beam size 1* | | |
| Multiplication | 25.62 | 26.47 (+3.3%) |
| Add bias | 4.98 | |
| Softmax | 12.81 | 7.94 (-76.9%) |
| Beam search | 16.64 | |
| *Beam size 5* | | |
| Multiplication | 112.9 | 115.04 (+1.9%) |
| Add bias | 23.66 | |
| Softmax | 56.61 | 67.58 (-56.5%) |
| Beam search | 75.12 | |

Table 2: Time taken in sec (Europarl)

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

Table 3: Time taken in sec (OpenSubtitles)

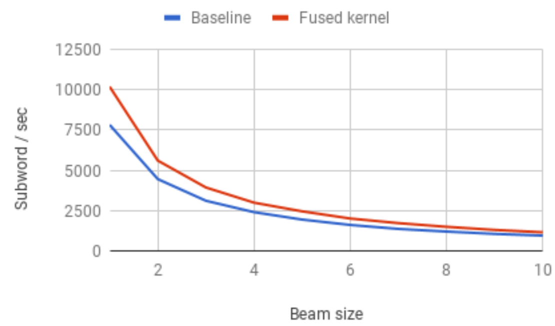


Figure 3: Speed using the fused kernel (Europarl)

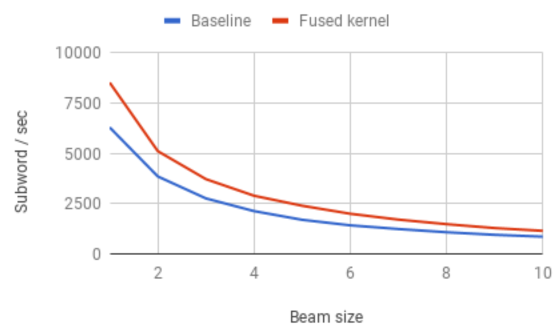


Figure 4: Speed using the fused kernel (OpenSubtitles)

up to 23% for the Europarl test set, Figure 3, and up to 41% for the OpenSubtitles test set, Figure 4.

The amount of time taken by the output layer dominates translation time for the OpenSubtitles test set, Figure 5, explaining the bigger overall translation speed with the fused kernel.

## 6.2 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. Therefore, using both methods together gives limited gains, in fact, using top-up batching with maxi-

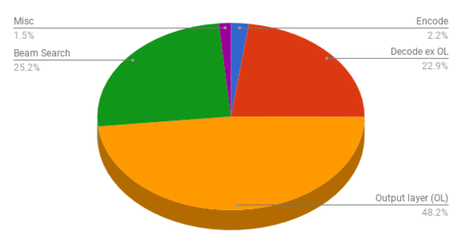


Figure 5: Proportion of time spent during translation (OpenSubtitles)

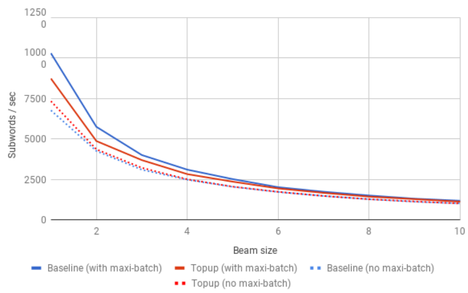


Figure 6: Speed using top-up batching (Europarl)

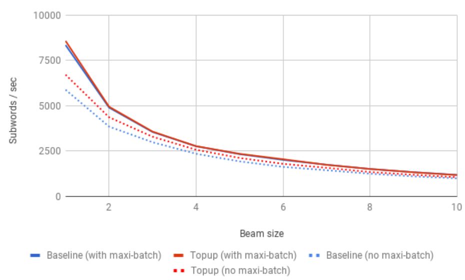


Figure 7: Speed using top-up batching (OpenSubtitles)

batch slows of between 2% to 18% when translating the Europarl test set due to the overhead of using the algorithm, Figure 6. 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 7. 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

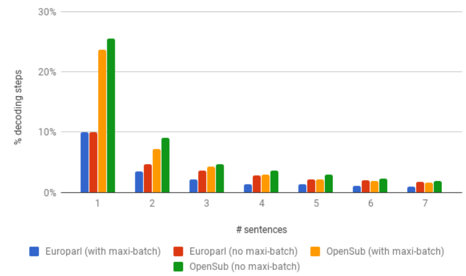


Figure 8: Actual batch size during decoding

mini-batching algorithm spend decoding with a very small number of sentences as this is does not fully utilize the GPU cores. From Figure 8, 20% of the decoding iterations have less than 8 sentences remaining in the batch when translating the Europarl test test with maxi-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.

### 6.3 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 9 and Figure 10. With maxi-batching, the speed was up to 41% and 21%.

## 7 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

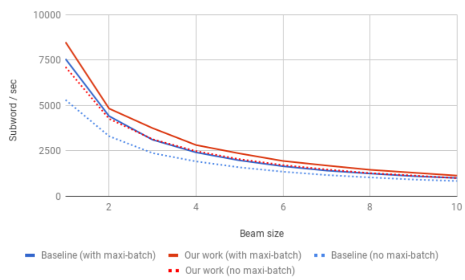


Figure 9: Cummulative results (Europarl

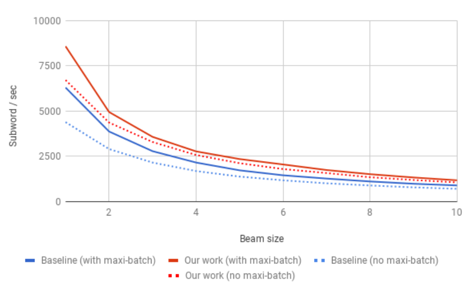


Figure 10: Cummulative results (OpenSubtitles

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.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional Sequence to Sequence Learning. *ArXiv e-prints*.

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.

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.