Exploring Hyper-Parameter Optimization for Neural Machine Translation on GPU Architectures

Robert Lim¹*, Kenneth Heafield²†, Hieu Hoang³‡, Mark Briers³§, Allen Malony¹¶

¹ University of Oregon, Eugene, OR, U.S.A.
² University of Edinburgh, Edinburgh, Scotland, U.K.
³ The Alan Turing Institute, London, England, U.K.

Abstract

Neural machine translation (NMT) has been accelerated by deep learning neural networks over statistical-based approaches, due to the plethora and programmability of commodity heterogeneous computing architectures such as FP-GAs and GPUs and the massive amount of training corpuses generated from news outlets, government agencies and social media. Training a learning classifier for neural networks entails tuning hyper-parameters that would yield the best performance. Unfortunately, the number of parameters for machine translation include discrete categories as well as continuous options, which makes for a combinatorial explosive problem. This research explores optimizing hyper-parameters when training deep learning neural networks for machine translation. Specifically, our work investigates training a language model with Marian NMT. Results compare NMT under various hyper-parameter settings across a variety of modern GPU architecture generations in single node and multi-node settings, revealing insights on which hyper-parameters matter most in terms of performance, such as words processed per second, convergence rates, and translation accuracy, and provides insights on how to best achieve high-performing NMT systems.

Motivation

The rapid adoption of neural network (NN) based approaches to machine translation (MT) has been attributed to the massive amounts of datasets, the affordability of high-performing commodity computers, and the accelerated progress in fields such as image recognition, computational systems biology and unmanned vehicles. Research activity in NN-based machine translation has been taking place since the 1990s, but statistical machine translation (SMT) soared along with the successes of machine learning. SMT incorporates a rule-based, data driven approach, and includes language models such as word based (n-grams), phrased-based, syntax-based and hierarchical based approaches. Neural machine translation (NMT), on the other hand, does not require predefined rules, but learns linguistic rules from statistical models, sequences and occurrences from large corpuses. Models trained using NNs produce even higher accuracy than existing SMT approaches, but training time can take anywhere from days to weeks to complete. Suboptimal strategies are often difficult to find, given the dimensionality and its effect on parameter exploration.

One of the main difficulties of training neural networks is the millions of parameters that need to be estimated. These parameters are estimated by optimization methods, such as stochastic gradient descent, where the solver seeks to identify the global optima. Due to the combinatorial search space, local optimization in many cases is sufficient to generalize beyond the training set (Goodfellow, Bengio, and Courville 2016) (Ch. 8). Thus, the tuning of hyper-parameters is paramount in accelerating training of neural networks.

In neural machine translation, modeling and training are crucial in achieving high performing systems. A combination of hyper-parameter optimization methods to train a NMT system is investigated in this work. Specifically, this work examines the stability of different optimization parameters in discovering local minima, and how a combination of hyper-parameters can lead to faster convergence.

The following contributions are made in this work:

• We identify which hyper-parameters matter most in contributing to the learning trajectory of NMT systems.
• We analyze our findings for translation performance, training stability, convergence speed, and tuning cost.
• We tie in systems execution performance with hyper-parameters.

Related Work

Hyper-parameter optimization has been an unsolved problem since the inception of machine learning, and becomes even more crucial in training the millions of parameters in neural networks. The past work has investigated techniques for hyper-parameter tuning and search strategies, such as Bergstra, et. al., concluding that random search outperforms grid search (Bergstra et al. 2011). Likewise, the authors in (Shahriari et al. 2016), Snoek, Larochelle, and Adams
take a Bayesian approach toward parameter estimation and optimization. However, these efforts apply their strategies on image classification tasks.

In relation to NMT, Britz, et. al. massively analyze neural network architectures and its variants \cite{Britz2017}. Their approach incorporates a 2-layer bidirectional encoder/decoder with a multiplicative attention mechanism as a baseline architecture, with a 512-unit GRU and a dropout of 0.2 probability. Their model parameters remained fixed and the studies varied the architecture, including depth layer, unidirectional vs bidirectional encoder/decoder, attention mechanism size, and beam search strategies. Likewise, Bahar et. al. compare various optimization strategies for NMT by switching to a different optimizer after 10k iterations, and found that Adam combined with other optimizers, such as SGD or annealing, increased the BLEU score by 2.4 \cite{Bahar2017}. However, these approaches study a standard NMT system. In addition, Wu, et. al. \cite{Wu2016} utilized the combination of Adam and SGD, where Adam ran for a fixed number of iterations with a 0.0002 learning rate, and switched to SGD with a 0.5 learning decay rate to slow down training, but did not perform hyper-parameter optimization.

To the best of our knowledge, there has not been any work comparing different hyper-parameter optimization strategies for NMT. Moreover, our optimization strategies are demonstrated on a production-ready NMT system and explores parameter selection tradeoffs, in terms of performance and stability.

Background

Machine translation involves model design and model training. In general, learning algorithms are viewed as a combination of selecting a model criterion, defined as a family of functions, training, defined as parameterization, and a procedure for appropriately optimizing this criterion. The next subsections discuss how sentences are represented with a neural network and the optimization objectives used for training a model for a translation system.

Machine Translation

This subsection discusses how neural networks can model language translation from a source to a target sequence.

Recurrent Neural Networks

Recurrent neural networks (RNN) are typically employed for neural machine translation because of its ability to handle variable length sequences. RNNs capture unbounded context dependencies typical in natural language comprehension and speech recognition systems.

For inputs $x_t$ and $y_t$, connection weight matrices $W_{ih}$, $W_{hh}$, $W_{ho}$, indicating input-to-hidden, hidden-to-hidden and hidden-to-output, respectively, and activation function $f$, the recurrent neural network can be described as follows:

\begin{align}
\text{hidden state} & = f_h(W_{ih}x_t + W_{hh}h_{t-1}) \\
\text{output} & = f_o(W_{ho}h_t).
\end{align}

\footnote{https://devblogs.nvidia.com/introduction-neural-machine-translation-gpus-part-2/}

![Figure 1: RNN encoder-decoder, illustrating a sentence translation from English to French. The architecture includes a word embedding space, a 1-of-K coding and a recurrent state on both ends.]

RNNs learn a probability distribution over a sequence by being trained to predict the next symbol in a sequence. The output at each timestep $t$ is the conditional probability distribution $p(x_t|x_{t-1}, \ldots, x_1)$.

**RNN Encoder-Decoder**

A RNN encoder-decoder (pictured in Fig. 1) encodes a variable-length sequence into a fixed vector representation, and decodes the fixed vector representation into a variable-length sequence \cite{Cho2014}. The RNN encoder-decoder are separate neural networks that are jointly trained to maximize the conditional log-likelihood, defined as

\[
\arg \max_\theta \frac{1}{N} \sum_{n=1}^{N} \log p(y_n|s_n),
\]

where $\theta$ represents the set of model parameters, each $s_n$, $t_n$ is a pair of input and output sequences from a parallel text corpus training set, and the output of the decoder from the encoder is differentiable. A trained RNN encoder-decoder can generate a target sequence given an input sequence.

Neural Machine Translation

Neural machine translation is defined as maximizing the conditional probability, arg max$_t$ $p(t|s) \propto p(s|t)p(t)$, for a source s and target t sequence, where $p(s|t)$ represents the translation model, and $p(t)$ represents the language model \cite{Sutskever2014}. \cite{Bahdanau2014}.

Taking the log linear of $p(t|s)$ yields,

\[
\log p(t|s) = \sum_{n=1}^{N} w_n t_n + \log \lambda(s),
\]

where $t_n$ and $w_n$ are the $n^{th}$ feature and weight, and $\lambda(s)$ is a normalization constant. The BLEU score provides a measure for optimizing weights during training.
### Table 1: Stochastic gradient descent and its variants.

| Optimizer | Operations | Description |
|-----------|------------|-------------|
| SGD       | $g_t \leftarrow \nabla_\theta J(\theta_t)$ | $\theta_{t+1} \leftarrow \theta_t - \eta g_t$, $\eta$ - learning rate, $\theta$ parameters |
| AdaGrad   | $g_t \leftarrow \nabla_\theta J(\theta_t)$ | $\eta_t \leftarrow \eta_{t-1} + \frac{g_t^2}{\gamma t + \epsilon}$, $\theta_{t+1} \leftarrow \theta_t - \frac{\eta_t}{\sqrt{\eta_{t-1} + \epsilon}} g_t$, divided by previous gradients, handles sparse data well |
| Adam      | $g_t \leftarrow \nabla_\theta J(\theta_t)$ | $\eta_t \leftarrow \eta_{t-1} + (1 - \gamma) g_t^2$, $\tilde{\eta} \leftarrow \frac{\eta_t}{1 - \gamma t}$, $m_t \leftarrow \mu m_{t-1} + (1 - \mu) g_t$, $\tilde{m}_t \leftarrow \frac{m_t}{1 - \mu}$, $\hat{m}_t \leftarrow \eta t - \frac{\eta_t}{\sqrt{\eta_{t-1} + \epsilon}} \tilde{m}_t$, $m_\mu$, decay mean of past gradients, $\tilde{m}_\mu$, $\hat{m}_\mu$ - bias correct terms that avoid zero initialization, $\gamma = 0.9$, $\mu = 0.999$, $\epsilon = 10^8$ |

### Table 2: Activation units for RNN.

| Activation | Operations | Description |
|------------|------------|-------------|
| tanh       | $s_t \leftarrow (e^x - e^{-x})/(e^x + e^{-x})$ | hyperbolic tangent |
| LSTM       | $i \leftarrow \sigma(x_t U^i + s_{t-1} W^i)$, $f \leftarrow \sigma(x_t U^f + s_{t-1} W^f)$, $o \leftarrow \sigma(x_t U^o + s_{t-1} W^o)$, $g \leftarrow \tan(x_t U^g + s_{t-1} W^g)$, $c_t \leftarrow c_{t-1} \circ f + g \circ i$, $s_t \leftarrow \tan(c_t) \circ o$, $o$ output, 2 tanh |
| GRU        | $z \leftarrow \sigma(x_t U^z + s_{t-1} W^z)$, $r \leftarrow \sigma(x_t U^r + s_{t-1} W^r)$, $h \leftarrow \tan(x_t U^h + (s_{t-1} \circ r) W^h)$, $s_t \leftarrow (1 - z) \circ h + z \circ s_{t-1}$, 2 gates, no internal memory, no output gates, 1 tanh |

### Optimization Objectives

The following subsections describe the tuning of hyperparameters that affect the performance of training a NMT system. In particular, this work focuses on the optimizers, activation functions, and dropout.

#### SGD Optimizers

Stochastic gradient descent (SGD), commonly used to train neural networks, updates a set of parameters $\theta$, where $\eta$ is the learning rate and $g_t$ represents the gradient cost function, $J(\cdot)$. AdaGrad is an adaptive-based gradient method, where $\eta$ is divided by the square of all previous gradients, $\eta_t$, plus $\epsilon$, a smoothing term to avoid dividing by zero. As a result, larger gradients have less frequent updates, whereas smaller gradients have more frequent updates. AdaGrad handles sparse data well and does not require manual tuning of $\eta$. Adaptive moment estimation (Adam) accumulates the decaying mean of past gradients, $m_t$, and the decaying average of past squared gradients, $\eta_t$, referred to as the first and second moments, respectively. The moments, $\tilde{m}_t$, $\hat{m}_t$, are biased corrected terms that avoid initializing to zero. $\gamma$ is usually set to 0.9, with $\mu = 0.999$, and $\epsilon = 10^8$. Table 1 displays SGD, AdaGrad and Adam optimizers.

#### Activation Functions

Activation functions serve as logic gates for recurrent neural networks that computes the hidden states, and include the hyperbolic tangent, long short term memory (LSTM) [Hochreiter and Schmidhuber 1997], and gated recurrent unit (GRU) [Cho et al. 2014]. Table 2 displays the hyperbolic tangent, LSTM and GRU activation functions.

To address the vanishing gradients problem associated with learning long-term dependencies in RNNs, LSTMs and GRUs employ a gating mechanism when computing the hidden states. For LSTMs, note that the input $i$, forget $f$ and hidden $h$ gates are the same equations except with different parameter matrices. $g$ is a hidden state, based on the current input and previous hidden state. $c_t$ serves as the internal memory, which is a combination of the previous memory, $c_{t-1}$, multiplied by the input gate. The hidden state, $s_t$, is calculated by multiplying $c_t$ and the output gate. On the other hand, a GRU employs a reset gate $r$ and an update gate $u$. The reset gate $r$ determines how to combine the new input with the previous memory, whereas the update gate $u$ defines how much of the previous memory to retain. If the reset gates were set to 1’s and the update gates to 0’s, this would result in a vanilla RNN.

The differences between the approaches to compute hidden units are that GRUs have 2 gates, whereas LSTMs have 3 gates. GRUs do not have an internal memory and output gates, compared with LSTM which uses $c$ as its internal memory and $o$ as an output gate. The GRU input and forget gates are coupled by an update gate $z$, and the reset gate $r$ is applied directly to the previous hidden state. Also, GRUs do
not have a 2nd non-linearity operation, compared to LSTMs, which uses two hyperbolic tangents.

**Dropout** In a fully-connected, feed-forward neural network, dropout randomly retains connections within hidden layers while discarding others (Srivastava et al. 2014). Table 3 displays a standard hidden update function on the top, whereas a version that decides whether to retain a connection is displayed on the bottom. $\hat{y}^{(l)}$ is the thinned output layer, and retaining a network connection is decided by a Bernoulli random variable $r^{(l)}$ with probability $p(\cdot) = 1$.

### Combination of Optimizers

Since the learning trajectory significantly affects the training process, it is required to select and tune the proper types of hyper-parameters to yield good performance. The construction of the RNN cell with activation functions, the optimizer and its learning rate, and the dropout rates all have an affect on how the training progresses, and whether good accuracy can be achieved.

#### Marian NMT

Marian (Junczys-Dowmunt et al. 2018) is an efficient NMT framework written in C++, with support for multi-node and multi-GPU training and CPU/GPU translation capabilities. Marian is currently being developed and deployed by the Microsoft Translator team. Table 7 displays parameters involved with tuning a neural machine translation system, categorized by model, training and validation, with values and types in brackets, and its default value, if any. The types of models in Marian include RNNs and Transformers (Vaswani et al. 2017).

The translation system evaluated in this study is a sequence-to-sequence model with single layer RNNs for both the encoder and decoder. The RNN in the encoder is bi-directional and the decoder is sequence-to-sequence. Depth, also referred to as *deep transitions* (Koehn 2017), is achieved by stacking activation blocks, resulting in tall RNN cells for every recurrent step. The encoder consists of four activation blocks per cell, whereas the decoder consists of eight activation blocks, with an attention mechanism placed between the first and second block. Word embedding sizes were set at 512, the RNN state size was set to 1024, and layer normalization was applied inside the activation blocks and the attention mechanism.

### Experiments

The experiments were carried out on the WMT 2016 (Junczys-Dowmunt and Grundkiewicz 2016) translation tasks for the Romanian and German languages in four directions: EN $\rightarrow$ RO, RO $\rightarrow$ EN, EN $\rightarrow$ DE, and DE $\rightarrow$ RO.
| Model    | dimensions | vocab [vect] |
|----------|------------|--------------|
| RNN      | embed [int] |
|          | dim [int]  |
|          | type [bi-dir, bi-unidir, s2s] |
|          | cell: type [gru, lstm, tanh], depth [1, 2, ...], transition cells [1, 2, ...] |
|          | skip [bool] |
|          | layer norm [bool] |
|          | tied embeddings [src, trg, all] |
|          | dropout [float] |
| transformer | heads [int] |
|          | no projection |
|          | tied layers [vector] |
|          | guided alignment layer |
|          | preproc, postproc, post-emb [dr, add, norm] |
|          | dropout [float] |
| Training | cost       | [ce-mean, ce-mean, words, ce-sum, perplexity] |
|          | after-epochs | [∞] |
|          | max length  | [int=50] |
|          | system      | GPUs, threads |
|          | mini-batch  | size, words, fit, fit-step [int, int, bool, uint] |
|          | optimizer   | [sgd, adgrad, adam] |
|          | learn rate  | decay: strategy [epochs, stalled, epoch + batches, ep+stalled], start, frequency, repeat warmup, inverse sqrt, warmup |
|          | label smoothing | [bool] |
|          | clip norm   | [float=1] |
|          | exponential smoothing | [float=0] |
|          | guided alignment | cost [ce, mean, mult], weight [float=0.1] |
|          | data weighting | type [sentence, word] |
|          | embedding   | vectors, norm, fix-src, fix-trg |
| Validation | frequency metrics | [ce, ce-words, perplexity, valid-script, translation, bleu, bleu-detok] |
|          | early stopping | [int=10] |
|          | beam size    | [int=12] |
|          | normalize    | [float=0] |
|          | max-length-factor | [float=3] |
|          | word penalty | [float] |
|          | mini-batch   | [int=32] |
|          | max length   | [int=1000] |
EN. The datasets and its characteristics used in the experiments are listed in Table 4, with number of sentence examples in parenthesis. Table 4 shows that for WMT 2016 EN → RO and RO → EN, the training data consisted of 2.6M English and Romanian sentence pairs, whereas for WMT 2016 EN → DE and DE → EN, the training corpus consisted of approximately 4.5M German and English sentence pairs. Validation was performed on 1000 sentences of the newsdev2016 corpus for RO, and on the newstest2014 corpus for DE. The newstest2016 corpus consisted of 1999 sentences for RO and 2999 sentences for DE, and was used as the test set. We evaluated and saved the models every 10K iterations and stopped training after 500K iterations.

All experiments used bilingual data without additional monolingual data. We used the joint byte precision encoding (BPE) approach (Sennrich, Haddow, and Birch 2015) in both the source and target sets, which converts words to a sequence of subwords. For all four tasks, the number of joint-BPE operations were 20K. All words were projected on a 512-dimensional embedding space, with vocabulary dimensions of 66000 × 50000. The mini-batch size was determined automatically based on the sentence length that was able to fit in GPU global memory, set at 13000 MB for each GPU.

Beam search was used for decoding, with the beam size set to 12. The translation portion consisted of recasing and detokenizing the translated BPE chunks. The trained models compared different hyper-parameter strategies, including the type of optimizer, the activation function, and the amount of dropout applied. The number of parameters were initialized with the same random seed. The systems were evaluated using the case-sensitive BLEU score computed by Moses SMT (Koehn et al. 2007).

We compared models trained on two different types of GPUs (P100 Pascal, V100 Volta), listed on Table 5. The corresponding CPUs are listed on Table 6. Each ran with four GPUs. The dataset was partitioned across 4 GPUs, and a copy of the model was executed on each GPU.

### Analysis

This section analyzes the results of the evaluated NMT systems in terms of translation quality, training stability, convergence speed and tuning cost.

### Translation Quality

Table 8 shows BLEU scores calculated for four translation directions for the validation sets (top) and the test sets (bottom), comparing learning rates, activation functions and GPUs. Note that entries with n/a means that no results were available, whereas entries with dnf indicates training time that did not complete within 24 hours. For the validation sets, LSTMs were able to achieve higher accuracy rates, whereas in the test set GRUs and LSTMs were about the same. Also, note that the best performing learning rates were usually at a lower value (e.g. 1e-3). The type of hidden unit

| cell | learn-rt | ro→en | de→en | en→ro | de→en |
|------|----------|-------|-------|-------|-------|
|      |          | P100  | V100  | P100  | V100  | P100  | V100  | P100  | V100  |
| GRU  | 1e-3     | 35.53 | 35.43 | 19.19 | 19.28 | 28.00 | 27.84 | 20.43 | 20.61 |
|      | 5e-3     | 34.37 | 34.05 | 19.07 | 19.16 | 26.05 | 22.16 | n/a   | 19.01 |
|      | 1e-4     | 35.47 | 35.46 | 19.45 | 19.49 | 27.37 | 27.81 | dnf   | 21.41 |
| LSTM | 1e-3     | 34.27 | 35.61 | 19.29 | 19.64 | 28.62 | 28.83 | 21.70 | 21.69 |
|      | 5e-3     | 35.05 | 34.99 | 19.48 | 19.43 | n/a   | 24.36 | 18.53 | 18.01 |
|      | 1e-4     | 35.41 | 35.28 | 19.43 | 19.48 | n/a   | 28.50 | dnf   | dnf   |
| GRU  | 1e-3     | 34.22 | 34.17 | 19.42 | 19.43 | 33.03 | 32.55 | 26.55 | 26.85 |
|      | 5e-3     | 33.13 | 32.74 | 19.31 | 19.87 | 31.04 | 26.76 | n/a   | 26.02 |
|      | 1e-4     | 33.67 | 34.24 | 18.98 | 19.69 | 33.15 | 33.12 | dnf   | 28.43 |
| LSTM | 1e-3     | 33.00 | 33.95 | 19.56 | 19.08 | 33.10 | 33.89 | 28.79 | 28.84 |
|      | 5e-3     | 33.10 | 33.52 | 19.13 | 19.51 | n/a   | 29.16 | 24.12 | 24.12 |
|      | 1e-4     | 33.29 | 32.92 | 19.14 | 19.23 | n/a   | 33.44 | dnf   | dnf   |

### Table 8: BLEU scores for validation (top) and test (bottom) datasets.

| cell | dropout | ro→en | de→en |
|------|---------|-------|-------|
|      |         | P100  | V100  |
| GRU  | 0.0     | 34.47 | 6:29  |
|      | 0.2     | 35.53 | 8:48  |
|      | 0.3     | 35.36 | 12.21 |
|      | 0.5     | 34.50 | 16:33 |
| LSTM | 0.0     | 34.34 | 6:29  |
|      | 0.2     | 34.71 | 8:10  |
|      | 0.3     | 35.67 | 15:13 |
|      | 0.5     | 34.50 | 15:13 |

### Table 9: Dropout rates, BLEU scores and total training time for test set, comparing systems.
Figure 2: BLEU scores as a function of training time (seconds), comparing GPUs (color), activation units (sub-columns), learning rates and translation directions.
mechanism (e.g. LSTM vs GRU) and the learning rate can affect the overall accuracy achieved, as demonstrated by Table 8.

Table 9 displays various dropout rates applied for translation directions RO → EN and DE → EN, comparing hidden units, GPUs and overall training time. The learning rate was evaluated at 0.001, the rate that achieved the highest BLEU score, as evident in Table 8. Generally speaking, increasing the dropout rates also increased training time. This may be the result of losing network connections when applying the dropout mechanism, but at the added benefit of avoiding overfitting. This is evident in Table 9 where applying some form of dropout will result in a trained model achieving higher accuracies. The best performance can be seen when the dropout rate was set at 0.2 to 0.3. This confirms that some form of skip connection mechanism is necessary to prevent the overfitting of models under training.

Figure 2 shows BLEU score results as a function of training time, comparing GPUs, activation units, learning rates and translation directions. Note that in most cases a learning rate of 0.001 achieves the higher accuracy in most cases, at the cost of higher training time. Also, note the correlation between longer training time and higher BLEU scores in most cases. In some cases, the models were able to converge at a faster rate (e.g. Fig. 2 upper left, RO → EN, GRU with learning rate of 0.005 vs 0.001).

Training Stability

Figure 3 shows the cross-entropy scores for the RO → EN and EN → RO translation tasks, comparing different activation functions (GRU vs. LSTM) during validation also performed similarly across GPUs and was also highly correlated with the translation direction. Cross-entropy scores for the EN → RO translation direction were more or less the same. However, for RO → EN, a LSTM that executed on a P100 converged the earliest by one iteration.

Figure 4 shows the same comparison of cross-entropy scores over epochs for DE → EN and EN → DE translation tasks. Note that the behavior of this translation task was wildly different for all systems. Not only did it take more epochs to converge compared to Fig 3, but also how well the system progressed also varied, as evident in the cross-entropy scores during validation. When comparing hidden units, LSTMs outperformed GRUs in all cases. When comparing GPUs, the V100 performed better than the P100 in terms of cross-entropy, but took longer to converge in some cases (e.g. v100-deen-lstm, v100-ende-lstm). Also, note that the behavior of the translation task EN → DE for a GRU hidden unit never stabilized, as evident in both the high cross-entropy scores and the peaks toward the end. The LSTM was able to achieve a better cross-entropy score overall, with nearly a 8 point difference for DE → EN, compared with the GRU.

Convergence Speed

Figure 5 shows the average words-per-second for the RO → EN translation task, comparing systems. The average words-per-second executed remained consistent across epochs. The system that was able to achieve the most words-per-second was v100-roen-gru-0.001, whereas the one that achieved the least words-per-second was the v100-roen-gru-0.005. Surprisingly, the best and worst performer was the v100-roen-gru, depending on its learning rate, with the sweet spot at 0.001. This confirms 0.001 as the best learn rate that can execute a decent number of words-per-second and achieve a fairly high accuracy, as evident in previous studies, across all systems.

Table 10 also displays words-per-second and validation, comparing activation units, learning rates and GPUs.
Figure 4: Cross-entropy over the number of epochs for DE → EN and EN → DE, comparing activation functions and GPUs.

Table 10: Words-per-second (average) and number of epochs, comparing activation units, learning rates and GPUs.

| cell | learn-rt | words-per-sec validation | | words-per-sec validation | |
|------|----------|--------------------------|----------------|--------------------------|----------------|
|      |          | P100 V100                |               | P100 V100                |               |
|      |          | validation                |               |                          |               |
|      |          | p100-deen-gru             |               | v100-deen-gru            |               |
|      |          | p100-deen-lstm            |               | v100-deen-lstm           |               |
|      |          | p100-ende-gru             |               | v100-ende-gru            |               |
|      |          | p100-ende-lstm            |               | v100-ende-lstm           |               |
| GRU  | 1e-3     | 33009.23 45762.54         | 18000 18000    | 29969.14 42746.15        | 15000 15000    |
|      | 5e-3     | 32965.23 24253.14         | 19000 8000     | 30223.89 23144.62        | 17000 10000    |
|      | 1e-4     | 32828.61 24341.96         | 44000 16000    | 29959.34 23277.51        | 25000 14000    |
| LSTM | 1e-3     | 29412.87 40534.06         | 15000 16000    | 27282.54 38131.13        | 14000 14000    |
|      | 5e-3     | 29536.65 40598.24         | 16000 16000    | 27245.42 37384.46        | 19000 21000    |
|      | 1e-4     | 29478.51 41441.37         | 40000 35000    | 27002.60 38118.79        | 25000 25000    |

Table 11: Total training time for four translation directions, comparing systems.

| cell | learn-rt | ro→en | en→ro | de→en | en→de |
|------|----------|-------|-------|-------|-------|
|      |          | P100 V100 | P100 V100 | P100 V100 | P100 V100 |
|      |          | validation | validation | validation | validation |
|      |          | P100 V100 | P100 V100 | P100 V100 | P100 V100 |
| GRU  | 1e-3     | 8:48 6:21 | 7:47 5:26 | 18:47 19:40 | 9:26 6:41 |
|      | 5e-3     | 9:41 4:52 | 8:38 6:02 | 23:57 4:36 | n/a 10:56 |
|      | 1e-4     | 21:58 9:43 | 12:33 8:59 | 23:50 21:09 | dnf 23:58 |
| LSTM | 1e-3     | 8:10 6:34 | 7:49 5:36 | 16:33 13:39 | 13:50 12:24 |
|      | 5e-3     | 9:02 6:34 | 10:44 8:32 | n/a 5:12 | 9:37 4:35 |
|      | 1e-4     | 22:29 14:05 | 13:46 9:45 | n/a 23:57 | dnf dnf |
fixing learning rate, the V100 was able to execute more
words-per-second than the P100, and was able to converge
at an earlier iteration. When comparing hidden units, GRUs
were able to execute higher words per second on a GPU and
converge at a reasonable rate (at 18000 iterations) for most
learning rates, except for 5e-3. When looking at LSTMs,
words-per-second executed on a V100 was similar, although
at a higher learning rate it was able to converge at 42000 it-
erations, but at the cost of longer training time and slower
convergence (35000 iterations).

Table [11] shows the corresponding total training time for
the four translation directions, comparing GPUs, activation
units, and learning rates. The dropout rate was set at 0.2,
which was the best performer in most cases (Tab 9). Table [11]
shows that the training time increased as the learning rates
were decreased. In general, Romanian took a fraction of the
time to complete training (usually under 10 hours), whereas
German took 18-22 hours to complete training.

Cost of Tuning a Hyper-Parameter

Table [12] displays the average time spent per epoch for the
Romanian ↔ English translation task, and Table [13] displays
the average time spent per epoch for the German ↔ English
translation task, comparing learning rates, activation cells,
and GPUs. The mean is displayed in each cell, with the stan-
dard deviation in parenthesis and the number of epochs ex-
cuted in brackets. For both tasks, dropout was set to 0.2.
Surprisingly, GRUs take longer on the V100 on average with
larger learning rates (5e-3, 1e-4) vs the P100, whereas for
LSTMs, the V100s clearly speeds up execution per epoch.
Note also that the learning rate does not have a significant
change in the average time spent per epoch, except for the
case with GRUs executing on the V100 with large learning
rates. The learning rate does have an effect on the number
of epochs executed, as seen in brackets as the learning rate
increases. Table [13] reports on the German ↔ English trans-
lation tasks. The same observation can be made for this task,
where GRUs spend less time per epoch compared to LSTMs,
and that the average time spent per epoch remains fixed as
the learnignrate increases.

Summarize Findings
This work reveals the following, with respect to tuning
hyper-parameters:
• Dropout is necessary to avoid overfitting. The recom-
   mended probability rate is 0.2 to 0.3.
• LSTMs take longer than GRUs per epoch, but achieves
   better accuracy.
• Although the average time spent per epoch remains fixed
   as learning rates increase, the total number of epochs ex-
cuted per training run increases as the learning rates in-
crease.
• Tensor core GPUs, particularly the V100, provide more
   words that can be processed per second, compared to non-
tensor core GPUs, such as the Pascal P100.

Discussion
The variation in the results, in terms of language translation,
hyer-parameters, words-per-second executed and BLEU
scores, in addition to the hardware the training was executed
on demonstrates the complexity in learning the grammatical
structure between the two languages. In particular, the learn-
ing rate set for training, the hidden unit selected for the ac-
tivation function, the optimization criterion and the amount
of dropout applied to the hidden connections all have a dras-
tic effect on overall accuracy and training time. Specifically,
we found that a lower learning rate achieved the best per-
formance in terms of convergence speed and BLEU score.
Also, we found that the V100 was able to execute more
words-per-second than the P100 in all cases. When looking
Table 12: Average time spent per iteration for RO → EN and EN → RO translation directions, comparing systems, with standard deviation in parenthesis and epochs in brackets.

| cell | learn-rt | P100   | V100   | P100   | V100   |
|------|----------|--------|--------|--------|--------|
|      |          | ro→en  |        | en→ro  |        |
|      |          | 1807.362941 | 1304.076471 | 1824.721059 | 1581.300893 |
|      |          | (142.43) [17] | (102.67) [17] | (166.06) [14] | (117.63) [14] |
| GRU | 1e-3     | 1814.640556 | 2472.531429 | 1816.642500 | 2385.243333 |
|      |          | (140.01) [18] | (11.16) [7] | (165.40) [16] | (15.08) [9] |
|      | 5e-3     | 1823.828837 | 2466.306429 | 1839.624500 | 2369.436923 |
|      |          | (129.08) [43] | (11.29) [14] | (167.28) [24] | (13.79) [23] |
|      | 1e-4     | 1860.640556 | 2512.531429 | 1862.642500 | 2415.243333 |
|      |          | (129.08) [43] | (11.29) [14] | (167.28) [24] | (13.79) [23] |
|      |          | 2032.362857 | 1470.278000 | 2010.199231 | 1438.945385 |
|      |          | (155.58) [14] | (108.79) [15] | (146.74) [13] | (107.76) [13] |
|      | LSTM     | 2018.048000 | 1469.054000 | 2014.716667 | 1474.787500 |
|      | 1e-3     | 2018.048000 | 1469.054000 | 2014.716667 | 1474.787500 |
|      |          | (148.21) [15] | (110.05) [15] | (144.41) [18] | (100.57) [20] |
|      | 5e-3     | 2026.976154 | 1445.585882 | 2037.517083 | 1443.758333 |
|      |          | (147.46) [39] | (106.30) [34] | (140.28) [24] | (99.68) [24] |

Table 13: Average time spent per iteration for DE → EN and EN → DE translation directions, comparing systems, with standard deviation in parenthesis and epochs in brackets.

| cell | learn-rt | P100   | V100   | P100   | V100   |
|------|----------|--------|--------|--------|--------|
|      |          | de→en  |        | en→de  |        |
|      |          | 3430.330526 | 2555.738148 | 3432.534444 | 2535.11 (88.61) |
|      |          | (124.58) [19] | (95.76) [27] | (128.70) [9] | (99.68) [9] |
| GRU | 5e-3     | 3450.174167 | 4898.036667 | 3859.903750 | 2886.194000 |
|      |          | (133.13) [24] | (47.79) [3] | (167.48) [8] | (122.26) [5] |
|      | 1e-4     | 3425.231600 | 4907.070667 | 2814.689000 | n/a     |
|      |          | (129.98) [25] | (51.24) [15] | (118.66) [30] | n/a     |
|      |          | (164.183) [15] | (129.37) [16] | (167.48) [8] | (122.26) [5] |
|      | LSTM     | 3855.21 (162.27) | 2852.335000 | 3859.903750 | 2886.194000 |
|      | 1e-3     | 3887.889333 | 2988.554375 | 3840.552500 | 2761.088125 |
|      |          | (124.58) [19] | (95.76) [27] | (128.70) [9] | (99.68) [9] |
|      | 5e-3     | 3855.21 (162.27) | 2852.335000 | 3859.903750 | 2886.194000 |
|      |          | (133.13) [24] | (47.79) [3] | (167.48) [8] | (122.26) [5] |
|      | 1e-4     | n/a     | 2814.689000 | n/a     | n/a     |
at accuracy as a whole, LSTM hidden units outperformed GRUs in all cases. Lastly, the amount of dropout applied on a network in all cases prevented the model from overfitting and achieve a higher accuracy.

The multidimensionality of hyper-parameter optimization poses a challenge in selecting the architecture design for training NN models, as illustrated by the varying degrees of behavior across systems and its performance outcome. This work investigated how the varying design decisions can affect training outcome and provides neural network designers how to best look at which parameters affect performance, whether accuracy, words processed per second, and convergence expectation. Coupled with massive datasets for parallel text corpuses and commodity heterogeneous GPU architectures, the models trained were able to achieve WMT grade accuracy with the proper selection of hyper-parameter tuning.

**Conclusion**

We analyzed the performance of various hyper-parameters for training a NMT, including the optimization strategy, the learning rate, the activation cell, and the GPU across various systems for the WMT 2016 translation task in four translation directions. Results demonstrate that a proper learning rate and a minimal amount of dropout is able to prevent overfitting as well as achieve high training accuracy.

Future work includes developing optimization methods to evaluate how to best select hyper-parameters. By statically analyzing the computational graph that represents a NN in terms of instruction operations executed and resource allocation constraints, one could derive execution performance for a given dataset without running experiments.

**References**

[Bahar et al. 2017] Bahar, P.; Alkhouri, T.; Peter, J.-T.; Brix, C. J.-S.; and Ney, H. 2017. Empirical investigation of optimization algorithms in neural machine translation. *The Prague Bulletin of Mathematical Linguistics* 108(1):13–25.

[Bahdanau, Cho, and Bengio 2014] Bahdanau, D.; Cho, K.; and Bengio, Y. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

[Bergstra et al. 2011] Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems (NIPS)*, 2546–2554.

[Britz et al. 2017] Britz, D.; Goldie, A.; Luong, T.; and Le, Q. 2017. Massive exploration of neural machine translation architectures. *arXiv preprint arXiv:1703.03906*.

[Cho et al. 2014] Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1079*.

[Goodfellow, Bengio, and Courville 2016] Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press, http://www.deeplearningbook.org.

[Hochreiter and Schmidhuber 1997] Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

[Junczys-Dowmunt and Grundkiewicz 2016] Junczys-Dowmunt, M.; and Grundkiewicz, R. 2016. Log-linear combinations of monolingual and bilingual neural machine translation models for automatic post-editing. In *ACL*, 751–758. Berlin, Germany: Association for Computational Linguistics.

[Junczys-Dowmunt et al. 2018] Junczys-Dowmunt, M.; Grundkiewicz, R.; Grundkiewicz, T.; Hoang, H.; Heafield, K.; Neckermann, T.; Seide, F.; Germann, U.; Aji, A. F.; Bogoychev, N.; et al. 2018. Marian: Fast Neural Machine Translation in C++. *arXiv preprint arXiv:1804.00344*.

[Koehn et al. 2007] Koehn, P.; Hoang, H.; Birch, A.; Callison-Burch, C.; Federico, M.; Bertoldi, C.; Cowan, B.; Shen, W.; Moran, C.; Zens, R.; et al. 2007. Moses: Open source toolkit for statistical machine translation. In *ACL*, 177–180. Association for Computational Linguistics.

[Koehn 2017] Koehn, P. 2017. Neural machine translation. *arXiv preprint arXiv:1709.07809*.

[Shahriari et al. 2016] Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; and De Freitas, N. 2016. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE* 104(1):148–175.

[Snoek, Larochelle, and Adams 2012] Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, 2951–2959.

[Srivastava et al. 2014] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

[Vaswani et al. 2017] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

[Wu et al. 2016] Wu, Y.; Schuster, M.; Chen, Z.; Le, Q. V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; et al. 2016. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.