Neural Machine Translation: English-Afrikaans

Jean Lucien Randrianantenaina
Applied Mathematics, Machine Learning and Artificial Intelligence
Stellenbosch University

***Abstract*—Language is the**

I. Introduction

Effective communication across languages is crucial in today's globalized world. Machine translation (MT) bridges language barriers by automatically converting text from one language to another. Traditional MT approaches often relied on complex linguistic rules and struggled to capture the nuances of natural language. Neural machine translation (NMT) offers a powerful alternative which leverages deep learning techniques to translate languages directly, achieving higher accuracy and fluency compared to traditional methods. Developing such model is not easy due to the data requirement and computational cost, so to have an insight through them, we will develop some simplistic model from scratch and fine-tuning existing model to translate engineering assessments from English to Afrikaans, with a small dataset. For that purpose we present briefly the basic architecture used in NMT in section II. Then, the section III focus on the methodology and finally present and discuss the result of our experimentation in the section IV.

II. Neural Machine Translation

Translation task consist into converting a text $\mathbf{x} = x_{1:T}$ from a language A into a text $\mathbf{y} = y_{1:S}$ of a language B, that have the same meaning and sense.

Compared to human, computer does not understand text and even words. But, with the appropriate modelling and tools it can perform such tasks. For Neural Machine Translation (NMT), it assigns a probability $P_\theta(y|x)$ for any possible translation y of x then choose the the one with the highest probability. In that case the probability of the whole translation is given by :

$$P(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{S} P_\theta(y_t|y_{1:t-1}, \mathbf{x}) \tag{1}$$

where θ is the model parameters. To implement this model, as indicated by the name NMT use neural network, specifically Encoder-Decoder architecture, illustrated in Figure 1 and 2b. In the Encoder part, the input are compressed into a smaller representation, which is feed into the Decoder part to produce an input with a specific target.

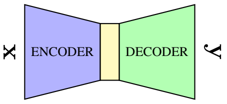
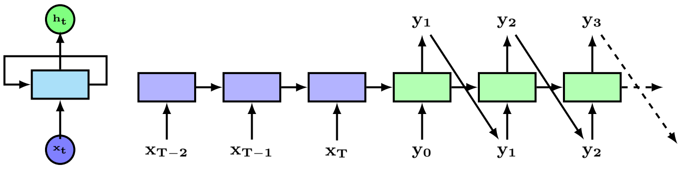


Fig. 1: Illustration of an encoder-decoder architecture

Treating, text data as a sequence, we use Recurrent Neural Networks (RNNs, Figure 2a), which predict a single word at a time, and the model is often call seq2seq (sequence-to-sequence) model. The particularity, of Recurrent neuron is to depend on its previous outputs or states.



(a) An RNN (b) A seq2seq model

Fig. 2: RNN and Encoder-Decoder architecture

The major problem of the basic of the RNNs are the vanishing and exploding gradient. This is solved by using more advanced architecture like Gated Recurrent Unit (GRU), and Long Short-Term Memory (LSTM). However, these architectures can be hard to train and computationally slow due to their recurrence. In addition, these model may have poor performance when dealing with long sequence.

Inspired by how human translate a sentence, researcher introduce attention mechanism. This technique allow to the model to keep track on the long term dependency, and not relaying only on the compressed version from the encoder, but with the output of the encoder at each time step.

Furthermore, they come up wit the self-attention mechanism by producing the layer n as a form of weighted sum of the previous layer $n-1$. This method, does not need any more of the currency between each term of the sequence. Lastly, attention and self-attention are on of the main components of the so-called transformer block found at the state-of-the-art of many machine learning models today.

The optimal weights of the neural networks (parameters) θ is found by minimizing the per-word (sequence) negative log likelihood:

$$J(\theta) = -\frac{1}{S}\sum_{i=1}^{S} \log P_\theta(y_t|y_{1:t-1}, \mathbf{x}) \tag{2}$$

Which can be done by using gradient based optimisation.

III. Methodology

This section present different part of the methodology in our experimentation and the reasons behind them.

A. Dataset

The initial dataset is a relatively small parallel English-Afrikaans from engineering assessment of the Stellenbosch University. This dataset is then augmented with a data from the Tatoeba project [**?**]. As, the assessment comes from several latex file, we combine all the mathematical environment by removing spaces to treat them as a single word. We also add space on delimiter characters such as "(),[],{}" to treat them as a single word, the removing extra spaces. Finally we switched it into lower case to be more efficient for our from-scratch implementation, and special character token line for start of sentence and end of sentence are used to replace "!, ? and ." in the text.

But, in the second part of the experimentation we just handle the latex mathematical environment without changing the case, and use the tokenizer provided by the pre-trained model that we use.

B. From Scratch Implementation

We principally focus on the fundamental basis of the implementation with PyTorch, so we use the same hyper-parameter for each implemented model as in the Table I.

Parameter	Value
Embedding Size	256
Hidden Size	1024
Number of Layers	2

TABLE I: Hyperparameters used in the from-scratch models.

Thus, with these parameter we implement the following model:

- Vanilla: RNNs, GRU, LSTM
- With scaled dot-product attention: RNNs, GRU, LSTM

Scaled dot product is defined is defined by :

$$a(\mathbf{q}, \mathbf{v}) = \frac{\mathbf{q}^\top \mathbf{v}}{\sqrt{D}} \tag{3}$$

where $\mathbf{q}$ is the current output of the decoder, $\mathbf{v}$ an output of the decoder at some time step, and D is the dimension of $\mathbf{q}$ (and $\mathbf{v}$ as well because they must have the same dimension).These value are passed through a $\mathrm{softmax}$ layer to have a vector formed by some $\alpha[0, 1]$, then we compute the context vector given by :

$$\mathbf{c}_s = \sum_{i=1}^{T} \alpha_i \mathbf{v}_i \tag{4}$$

This later is then concatenated with $\mathbf{q}_s$ to produce the final output of the decoder at the step s.

Finally we train each model with at most 50 epochs, using the NAdam optimizer with 10^{-3} as learning rate. Note that we did not perform any explicit hyper-parameter tuning for these model. We use a sentence to observe the evolution of the the model during training and stop it once it have stay output a a perfect prediction, or there is no more evolution in the loss value during training (manually ...). This is done to avoid over-fitting on the training set.

C. Using a Pre-trained Model

A pre-trained model is model which is already trained on some data set for a specific task. So, Leveraging such model can be a highly efficient approach, and allows us to save substantial time and resources. Usually trained on massive amounts of data, therefore it contains already an important amount of information.

For this task, we will perform a fine-tuning, e.g, we will re-train the model on our data-set to give the model a more specific domain and vocabulary as it may have not seen data such engineering assessment. This practice is also known as domain shifting.

Since we aim to translate English to Afrikaans, we fine-tune the `opus-mt-en-af` model which can already perform the given task but not yet adapted into our requirement. Based on the Marian-MT framework [7], this model is part of a larger collection of an open source high performant Neural Machine Translation (NMT) models [8].

The opus-mt-en-af model was initially trained on a public dataset from the Open Parallel Corpora [3], which contain a strong foundation. But it is highly probably to not contain the some specific vocabulary from the engineering domain. Thus by fine-tuning the model on our dataset, we expect to improve its performance in handling these specialized terms and phrases. Here we only use the engineering assessment data set because, the Tatoeba data set is part of the training data of the model.

To implement this task, we use the `transformer` library that allows us to obtain the weights of the model from Hugging Face [2] using PyTorch. It also allows us to tokenize our data set, using the tokenized provided by opus-mt which is based on `sentencepiece` [5] of Google (with the de-tokenizer as well).

Then for the parameter optimization of the model, AdamW [4] as suggested in the Hugging Face NLP course for the translation model fine-tuning.

D. Model evaluation

To evaluate all the models by computing the bilingual evaluation understudy (BLEU) metric, which allows us to measure how similar the results of the machine translation are to human translation (references).

For the from-scratch implementation, we evaluate the augmented dataset and the engineering assessment only.

For the finetuning, we evaluate it before the training and after training on the validation set, this is done to see if we effectively able to improve the model on the new domain.

IV. Results and Discussion

After training our model, we present in this section the results and discussion related to our experimentation.

A. From-Scratch Implementation

Figure 3 shows the loss per epoch for each model. We stopped the training at specific epochs to avoid overfitting on the training set once we had several epochs where the model predicted our monitoring sentence perfectly. We also note that we did not perform any hyperparameter tuning.

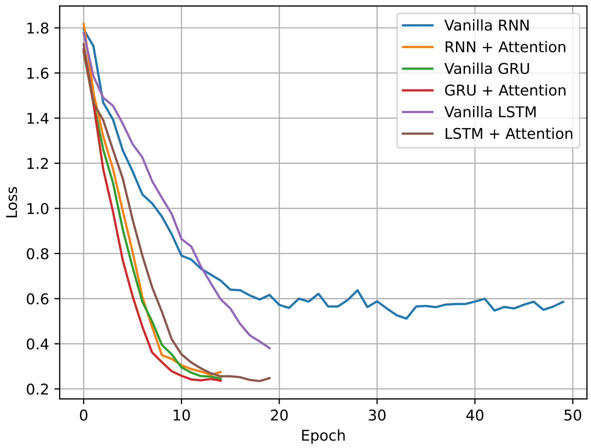


Fig. 3: Epoch/Loss for from-scratch implementation

As we can observe, the vanilla RNN got stuck during the training process, which is confirmed in Table II. The vanilla LSTM converged to optimal weights but with poor performance, even with the attention mechanism, compared to the other models. On the other hand, the other models converged quickly and had better performance on the training set.

		Validation Set 1		Validation Set 2	
Model	#Epochs	BLEU G	BLEU B	BLEU G	BLEU B
RNN	50	0.05	0.02	0.05	0.02
GRU	15	0.38	0.35	0.14	0.12
LSTM	20	0.28	0.24	0.12	0.10
RNN+Att.	15	0.39	0.33	0.18	0.17
GRU+Att.	15	0.31	0.32	0.13	0.16
LSTM+Att.	20	0.28	0.30	0.11	0.13

TABLE II: BLEU scores for each from-scratch model on the validation set

Table II provides the BLEU scores for each model on two different validation sets. Validation Set 1 is the augmented dataset and Validation Set 2 is the engineering assessment only. G indicates that we used the Greedy search method to predict the translation, and B indicates beam search with width 3.

- RNN: The vanilla RNN performed poorly, with BLEU scores of 0.05 and 0.02 on both validation sets, indicating that it failed to learn effectively.
- GRU: The GRU model showed significantly better performance, particularly on Validation Set 1, with BLEU scores of 0.38 and 0.35. However, its performance dropped on Validation Set 2.
- LSTM: The LSTM model performed better than the RNN but was still outperformed by the GRU and attention-based models. Its BLEU scores were 0.28 and 0.24 on Validation Set 1, and 0.12 and 0.10 on Validation Set 2.
- RNN+Att.: The addition of an attention mechanism to the RNN improved its performance significantly, achieving BLEU scores of 0.39 and 0.33 on Validation Set 1, and 0.18 and 0.17 on Validation Set 2.
- GRU+Att.: The GRU model with attention achieved slightly lower scores than the vanilla GRU on Validation Set 1 but showed a slight improvement on Validation Set 2.
- LSTM+Att.: The LSTM with attention had similar performance to the vanilla LSTM, indicating that the attention mechanism did not significantly improve its performance.

These result

B. *Fine-Tuning*

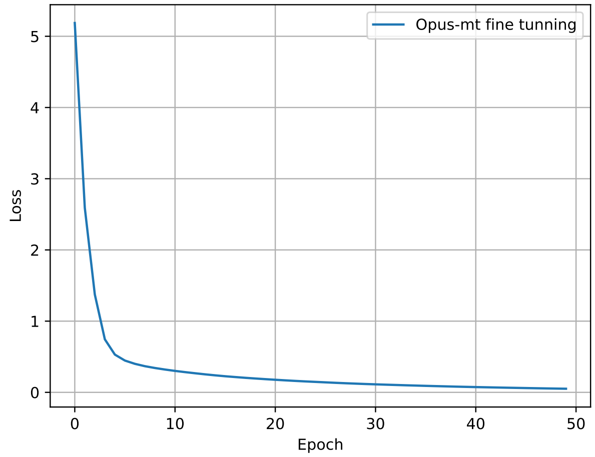


Fig. 4: Epoch/Loss for fine-tuning implementation

Model	#Epochs	LR	BLEU
opus-mt-en-af	N/A	N/A	0.28
Fine-tuned opus-mt-en-af	50	2×10^{-5}	0.37

TABLE III: BLEU score of the original and fine-tuned opus-mt model on the validation set

1) Interpretation and Discussion: The results from the fine-tuning experiment are shown in Figure 2 and Table 2. The original opus-mt-en-af model achieved a BLEU score of 0.28 on the validation set, whereas the fine-tuned version of the model, trained for 50 epochs with a learning rate of 2×10^{-5}, achieved a BLEU score of 0.37.

- Original opus-mt-en-af: The pre-trained opus-mt-en-af model provided a baseline BLEU score of 0.28, indicating a reasonable performance without any fine-tuning.
- Fine-tuned opus-mt-en-af: The fine-tuned version of the model showed a significant improvement with a BLEU score of 0.37. This suggests that the fine-tuning process effectively adapted the pre-trained model to the specific characteristics of our dataset, improving its translation quality.

2) Discussion of Results: The results indicate that fine-tuning a pre-trained model can considerably enhance its performance on a specific task. The improvement in BLEU score from 0.28 to 0.37 highlights the benefits of adapting a general model to a particular domain or dataset.

Comparing the from-scratch models and the fine-tuned models, several points emerge:

- Training Efficiency: The from-scratch models required careful monitoring to avoid overfitting and significant computational resources to converge. In contrast, fine-tuning a pre-trained model was more efficient and resulted in better performance.
- Model Performance: The best from-scratch model (RNN+Att) achieved a BLEU score of 0.39, which is close to the fine-tuned model's score of 0.37. This demonstrates that while from-scratch models can achieve competitive performance, the effort and resources required are substantially higher.
- Transfer Learning Benefits: The fine-tuned model's performance underscores the effectiveness of transfer learning. By leveraging a pre-trained model, we could achieve high performance with relatively fewer epochs and without extensive hyperparameter tuning.

V. Conclusion

In conclusion, while training models from scratch can yield high-performing models, fine-tuning pre-trained models provides a more efficient and often equally effective alternative. This approach is especially advantageous when computational resources are limited or when a quick turnaround is needed for model deployment.

References

[1] Herman Kamper. *NLP817*. https://www.kamperh.com/nlp817/, 2022–2024.

[2] Hugging Face https://huggingface.co/

[3] Open Parallel Corpora https://opus.nlpl.eu/

[4] AdamW https://pytorch.org/docs/stable/generated/torch.optim.AdamW.html

[5] Sentencepiece https://github.com/google/sentencepiece

[6] Helsinki NLP https://blogs.helsinki.fi/language-technology/, https://huggingface.co/Helsinki-NLP

[7] Junczys-Dowmunt, M., Grundkiewicz, R., Dwojak, T., Hoang, H., Heafield, K., Neckermann, T., Seide, F., Germann, U., Aji, A.F., Bogoychev, N. and Martins, A.F., 2018. Marian: Fast neural machine translation in C++. *arXiv preprint arXiv:1804.00344.*

[8] Jörg Tiedemann and Santhosh Thottingal. 2020. OPUS-MT - Building open translation services for the World. *In Proceedings of the 22nd Annual Conference of the European Association for Machine Translation, pages 479–480, Lisboa, Portugal. European Association for Machine Translation.*