TP3 - Machine Translation with Sequence Processing Units

Paulo Victor Correia - 2167525

March 23, 2023

1 Introduction

This report displays the experimental results obtained for the translation model with Vanilla RNN, GRU RNN and self-attention transformers for the TP3 project. It consists of a translation task from English to French by using the architectures aforementioned.

The data consists of sentences in English on one side and their respective translation on the other side. Overall, we have around 188,000 training sentences and 20,000 validation sentences, while we have an English vocabulary of size 11,709 and a French vocabulary of size 17,645.

To take into account the position of the tokens, a transformer must use a positional encoding layer to associate each token to a specific position in the sentence. For this instance, we utilized a modified version of the positional encoding from the original paper [1]. We claim that this modification also makes the transformers able to perform the task.

In order to understand the capacities of the models analyzed, we performed a hyperparameter search over the number of stacked layers in each architecture. The reason for running only through the number of layers comes from computational resource scarcity and the slow convergence of some models, which would output minimally decent results after at least 10 epochs of training. Finally, we will display the results for each type of model according to their size.

The purpose of these experiments is to confirm whether or not the architecture used to translate affects the final performance of the task. If so, which are the best and in what conditions do they perform better than the other. With our background knowledge, we hypothesize that the transformers will perform better, followed by the GRU and the Vanilla architectures.

The report has the following structure: section 2 will display experimental configurations as well as the positional encoding modification; section 3 displays the results of the experiments; and we close the article in section 4 with the conclusion.

2 Experimental Configuration

2.1 Positional Encoding Modification

Transformers require positional encoding to inject information about the relative and absolute position of the words in a sentence. They are necessary because the transformer's architecture has no clue of the order of the words in the sequence fed to the model. Therefore, the authors of "Attention is All You Need" propose a positional encoding to make the transformer aware of the position of the words. Hence, they do this by encoding words on even and odd positions of the sentence with the following equations:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

which d_{model} stands for the embedding dimension, pos is the position and i is the dimension. We use the first equation on words in even positions and the second on words in odd positions.

However, we made a small modification based on Alladin Person's implementation [2], in which he creates a vector from zero to the sequence length and expand it to have the shape (batch_size, sequence_length). After that, we feed this vector to a positional embedding layer and sum it to the actual sequence embedding layer.

2.2 Training and Validation Datasets

After downloading the data for both languages, we have the option to filter the data to reduce the size and match the computational resources available. We have the parameter *Max Sequence Size* which limits the sizes of the sequence in both languages and the *Min Token Frequency* which removes sentences that have words that appeared less than 2 times in the dataset.

For this project, we used the following parameters for training and validation dataset generation.

- Max Sequence Size: 60
- Min Token Frequency: 2

2.3 Training Parameters

The training parameters consist of the parameters used on the optimizer function of the synaptic weights. We used the Adam optimizer from pytorch with the following parameters:

- Learning Rate: 0.001

- Batch size: 256
- Epochs: 20

All the other parameters were set to the default values as stated in the framework.

2.4 Hyperparameter Search Space And Computational Resources

We had to buy Google Colab Pro to perform the experiments as specified with the training parameters because these models demand a lot of computational power, unavailable in the free version of Colab. Therefore, we couldn't perform more refined experiments by changing the hidden size and embedding layer and we performed and had to search through the number of layers.

Having said that, the experiments were performed in Google Colab Pro GPUs, either NVIDIA T4 Tensor core or V100 and A100 premium GPUs. The hyperparameters we searched for were only the number of layers in the recurrent networks, and the number of stacked encoders and decoders for the transformer.

Vanilla-RNN and GRU-RNN configuration:

- Number of layers: [1, 2, 3]
- Hidden size: 256
- Embedding size: 120

Transformer's configuration:

- Number of layers: [1, 2, 3]
- Hidden size: 256
- Embedding size: 120
- Number of attention heads: 8

3 Results And Discussion

We will present the results for each network architecture as well as the hyperparameters used in each one of them. At the end we will present the charts obtained through weights and biases [3] and discuss the results displayed in the tables and in the charts.

3.1 Vanilla RNN

Tables 1 and 2 present the final accuracy scores for 1, 2 and 3 layers for the training and test data respectively of the Vanilla RNN models. We measure the performance by using top-k accuracy, which means that the correct word to

compose the translated sequence is within the k most probable words according to the translator.

Train - top-1	Train - top-10	Train - top-5	
1	0.46	0.67	0.75
2	0.42	0.64	0.73
3	0.43	0.64	0.73

Table 1: Train performance of the Vanilla RNN model.

num. layers	Test - top-1	Test - top-10	Test - top-5
1	0.45	0.66	0.74
2	0.42	0.63	0.72
3	0.44	0.65	0.73

Table 2: Validation performance of the Vanilla RNN models.

3.2 GRU RNN

Tables 3 and 4 present the final accuracy scores for 1, 2 and 3 layers for the training and test data respectively of the GRU RNN models.

num. layers	Train - top-1	Train - top-10	Train - top-5
1	0.55	0.76	0.82
2	0.60	0.82	0.87
3	0.63	0.84	0.89

Table 3: Validation performance of the GRU RNN models.

num. layers	Test - top-1	Test - top-10	Test - top-5
1	0.56	0.76	0.82
2	0.62	0.82	0.87
3	0.65	0.85	0.89

Table 4: Validation performance of the GRU RNN models.

Comparing the results with the Vanilla RNNs, GRUs are more powerful for the translation task than Vanilla RNNs. We can assume that one of the reasons is the GRU's architecture, which allows the network to have more parameters and make stronger associations in time.

However, since we could train for only 20 epochs, we could not see how powerful the network could become. But given enough resources to train, GRUs can output really robust models.

3.3 Transformers

Tables 5 and 6 present the final accuracy scores for 1, 2 and 3 layers for the training and test data respectively of the Transformer models.

num. layers	Train - top-1	Train - top-10	Train - top-5
1	0.74	0.91	0.94
2	0.71	0.88	0.92
3	0.41	0.62	0.71

Table 5: Train performance of the Transformer models.

num. layers	Test - top-1	Test - top-10	Test - top-5
1	0.76	0.91	0.93
2	0.73	0.89	0.92
3	0.42	0.62	0.71

Table 6: Train performance of the Transformer models.

Transformers play a completely different game on the translation task. For just a single encoder-decoder translator, it outperformed both RNN architectures by a comfort amount. It also had a good margin for improvement given enough resources.

Figure 1 and 2 display the loss for the training and validation obtained throughout the optimization process respectively.

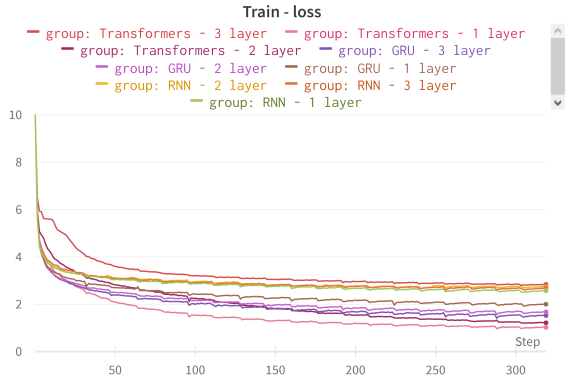

Figure 1: Train loss along the 300 steps of optimization.

The plots indicate to us that the transformer with one and two layers obtains the best loss function values over the other two. However, the transformer with three layers performs worse than the worse Vanilla RNN model. This bad performance indicates that our training setting may not fit that larger model

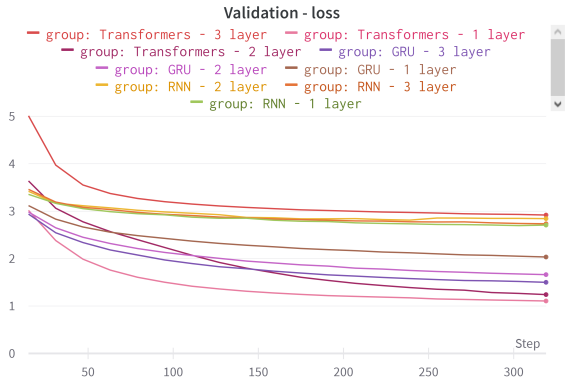

Figure 2: Validation loss along the 300 steps of optimization.

and that it slowed down the convergence of that model because it has much more parameters than the other 2.

Figures 3 and 4 present the accuracy of the most likely prediction for the next word in the 300 steps of training. These graphs endorse our hypothesis that the transformers perform better on average, followed by GRU and Vanilla RNNs.

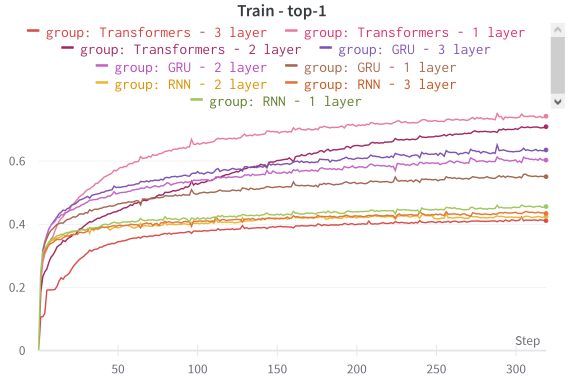

Figure 3: Accuracy of the top 1 prediction of the next word in a sentence during translation for the training data.

4 Conclusion

After thoroughly performing experiments, we conclude that Transformers outperform recurrent neural networks in translation. While the RNN's sequential cascade processing introduces the risk of gradient vanishing or exploding, or

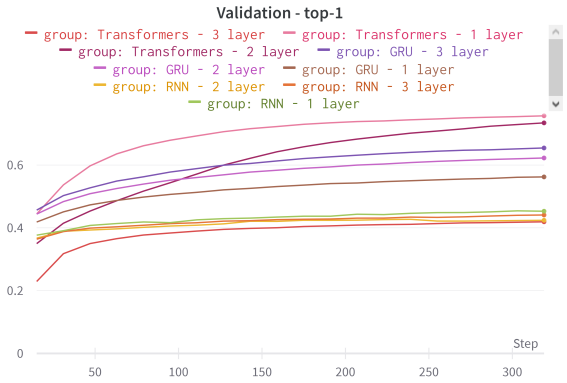

Figure 4: Accuracy of the top 1 prediction of the next word in a sentence during translation for the training data.

even losing the context of the sentence, the transformer's architecture can compute the whole sequence at once without this risk. Even by the necessity of positional embedding encoding to make the model aware of the word's position in a sentence, transformers produced a more robust model with a faster convergence over the other models.

Other experiments that we could have performed include a more refined hyperparameter search with more epochs to assess the model's performance, but it requires a lot of computational resources that are unavailable.

For future work, we propose to perform a more fine grid search for hyperparameters. Search for different values of embedding size and even hidden size show a significant difference in the final performance, according to initial experiments.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[2] Aladdin Persson. Machine-Learning-Collection. `https://github.com/aladdinpersson/Machine-Learning-Collection/blob/master/ML/Pytorch/more_advanced/transformer_from_scratch/transformer_from_scratch.py`, 2023.

[3] Weights and biases.

A Appendix

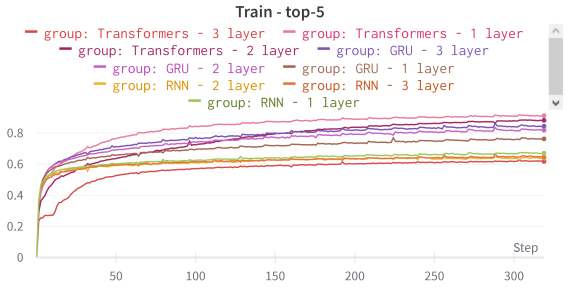

Figure 5: Accuracy of the top 5 predictions of the next word in a sentence during translation for the training data.

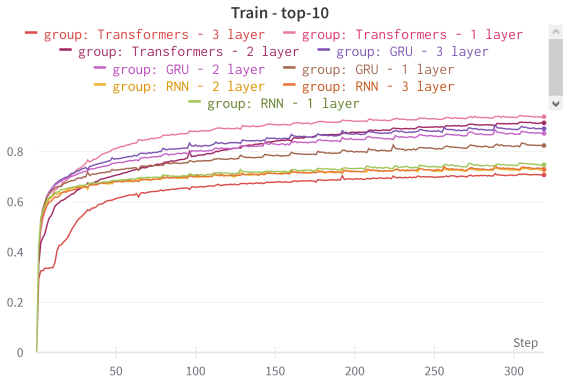

Figure 6: Accuracy of the top 10 predictions of the next word in a sentence during translation for the training data.

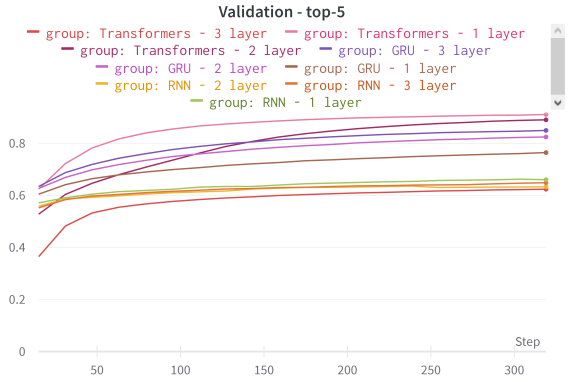

Figure 7: Accuracy of the top 5 predictions of the next word in a sentence during translation for the validation data.

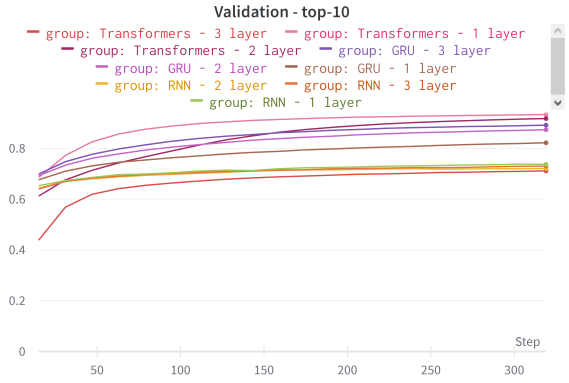

Figure 8: Accuracy of the top 10 predictions of the next word in a sentence during translation for the validation data.