Transformer for language translation

Willem Johannes Van Der Merwe (2914429) COMS7047A

May 27, 2024

1 Introduction

As a tribute to large language models, I demonstrate the transformer model from the "Attention is all you need" paper by [4]. Transformers, aim to address and improve upon the Recurrent neural network structure by eliminating some of the runtime computational complexities of RNN's. The first of which is the fixed computation cost for long sequences [3]. Another issue is vanishing and exploding gradients, during backpropogation pass of the network as it is important to include each output and hidden state for each step [3]. Lastly it is difficult to accesses information encoded from long time ago [3].

Transformers effectively address the limitations of RNNs by using several adaptation but most prominently the "attention mechanism", which enable the model to focus on different parts of the input sequence, regardless of their positions. This feature allows transformers to consider the entire context of the input simultaneously, leading to a significant improvement in processing efficiency and model performance. By leveraging attention, transformers can dynamically weigh the relevance of each part of the input data, which is crucial for tasks like translation where the context and order of words are vital.

Additionally, this architecture alleviates the problems of vanishing and exploding gradients that are common in deep RNN structures, as it allows gradients to flow through the network more directly during training. This results in a model that not only learns faster but also has the capability to handle long-range dependencies more effectively, making it particularly powerful for language processing tasks where context from far earlier in the text can significantly influence the meaning and thus the correct translation of later text.

Objective: Preform a translation task that show the french translation for a input English sentence.

2 Model

2.1 Implementation of the Model

All of the implementation details of the model defined in this section is contained in the accompaning code file **model.py**

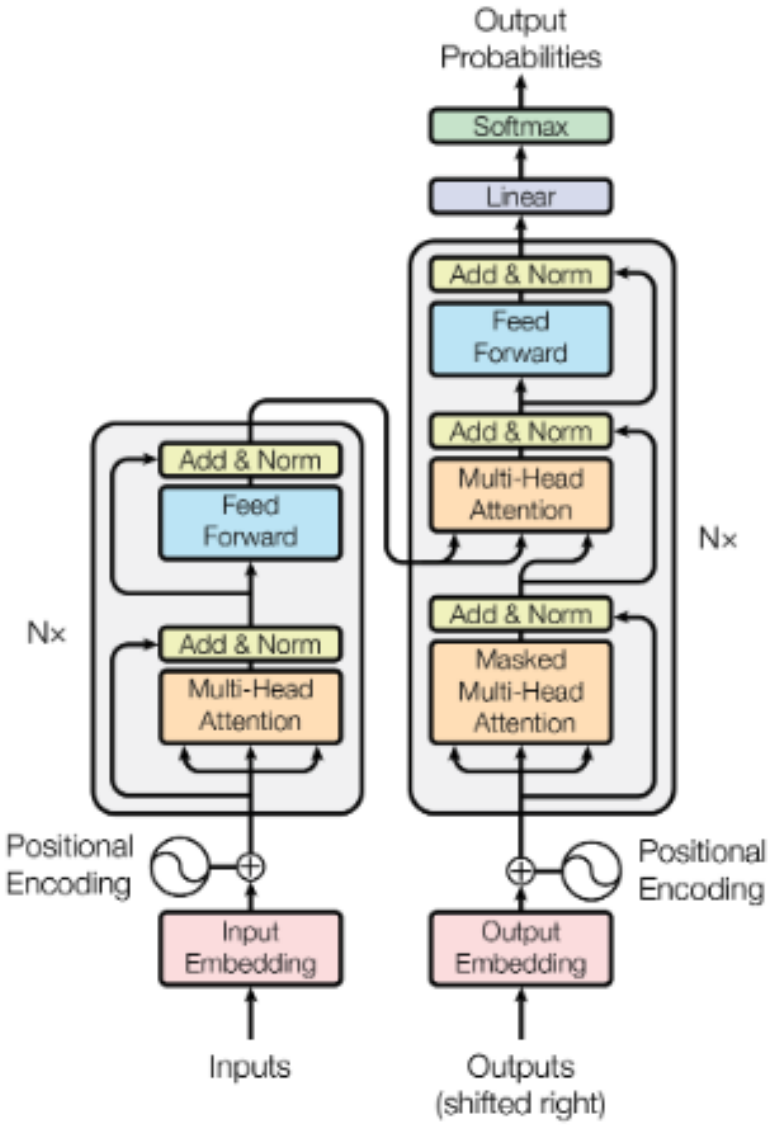

Figure 1: Transformer Architecture

2.1.1 Key components

For each of these as outlined by [4], I created classes that inherit from the PyTorch Module Class.

1. Input/Output Embeddings: Tensor representations of the words within a sentence. I utilize the PyTorch embedding layer here, and as [4] suggest apply the square root of the model size to the embedding.

```
return self.embeddings(x) * math.sqrt(self.size)
```

2. Positional Encoding: Adds information about the position of words in the sequence, as the Transformer itself is invariant to sequence order. The authors [4] add alternating sinusoidal functions. I ensure that the tensors are of the size of the model dimension and also already apply batch splitting.

```
position = torch.arange(0, seq_len, dtype=torch.float).
    unsqueeze(1)

# common denominator in the sinusoidal functions
div_term = torch.exp(torch.arange(0, size, 2).float() * (-math
    .log(10000.0) / size))

# odds and evens -> start even go forward 2, start odd go
    forward 2
pos_enc[:, 0::2] = torch.sin(position * div_term)
pos_enc[:, 1::2] = torch.cos(position * div_term)
```

3. Attention: The authors [4] use scaled dot product attention:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

- Self-Attention & Cross-Attention: shows the attention scores for each word in the input sentence relative to all other words, capturing dependencies regardless of their distance. With self attention we feed queries, keys and values from the same embedded inputs, where as the decoder cross attention mechanism accepts the queries and keys from the encoder.

```
attention_scores = torch.matmul(query, key.transpose(-2,
    -1)) / math.sqrt(vec)
attention_scores = attention_scores.softmax(dim=-1)
```

- Multi-Head Attention: Extends the self-attention mechanism to multiple attention heads, allowing the model to jointly attend to information from different representation subspaces, we can think of this as scaling the dimensions of the attention scores up and down to capture better information.

$$MultiHead(Q, K, V) = Concat(head_1 \dots head_h)W^O$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

```
queries = queries.view(queries.shape[0], queries.shape[1],
    self.heads, self.vec).transpose(1, 2)
keys = keys.view(keys.shape[0], keys.shape[1], self.heads,
    self.vec).transpose(1, 2)
values = values.view(values.shape[0], values.shape[1], self.
    heads, self.vec).transpose(1, 2)
```

```
x, self.attention_scores = MultiHeadAttention.attention(
    queries, keys, values, mask, self.dropout)
x = x.transpose(1, 2).contiguous().view(x.shape[0], -1, self.
    heads * self.vec)
```

4. Feed-Forward Neural Network: Applies a position-wise fully connected feed-forward network to the outputs of the attention layer.

```
return self.linear2(self.dropout(torch.relu(self.linear1(x))))
```

5. Layer Normalization: Normalizes the input to each sublayer to stabilize and speed up training. I simply extend the PyTorch layer normalization class.

6. Residual Connections: Adds the input of each sublayer to its output to ensure that gradients can flow through the entire network during back-propagation.

```
return x + self.dropout(connector(self.norm(x)))
```

7. Masking: [4] show that they adapt the self-attention mechanism in the decoder to ensure that each position only attends to previous positions. This is achieved by masking future positions and offsetting the output embedding by one position. As a result, the prediction at position i can only rely on the known outputs from positions before i .

8. Regularisation: Throughout there is use of dropout within the embeddings.

2.1.2 Architecture

I combined the required components within my build_transformer() method which produces a Class representation Transformer(nn.Module) with the necessary methods to invoking the encoder and decoder

- Encoder: The encoder reads the input tokens, applies the embedding and positional encoding, and the we run the first attention scores with residual connection to feed inputs into a normalisation and output addition, after which I apply the feed forward network with a residual connection of the inputs into a normalisation and output addition. This is inline with the architecture in 1.

- Decoder: The decoder also consists of stacked layers similar to the encoder. Each layer in the decoder has three sub-layers: a multi-head self-attention mechanism, a multi-head attention mechanism over the encoder's output, and a position-wise feed-forward network. Residual connections are employed around each of the sub-layers, followed by layer normalization. The self-attention sub-layer in the decoder is masked.

- Mapping Layer: The mapping layer, often referred to as the output layer, projects the final decoder outputs to the target vocabulary size. This is usually implemented as a linear layer followed by a softmax function to generate the probabilities for each token in the target vocabulary. The output of the softmax layer represents the model's prediction for the next token in the sequence.

2.1.3 Hyper-parameters

I chose the parameters used in the paper.

- Embedding Size (d_model): 512. This defines the size of the embedding vectors used in the model.

- Number of Layers: 6 layers for both the encoder and the decoder. This depth was found to be sufficient to capture complex patterns in the data.

- Number of Attention Heads: 8. Using multiple heads allows the model to attend to information from different representation subspaces simultaneously.
- Feed-Forward Network Size: 2048. This size was chosen to balance the capacity and computational efficiency.
- Dropout Rate: 0.1. Dropout was used to prevent over-fitting by randomly setting a fraction of input units to zero during training.
- Batch Size: 8. A batch size of 8 this is per attention head, was used to provide a balance between training speed and stability.
- Learning Rate: The initial learning rate was set to 10^{-4}, with a learning rate scheduler that decreases the learning rate as training progresses.

3 Dataset

All of the implementation details of the data processing defined in this section is contained in the accompanying code file **dataset.py**

I use the Opus_Books dataset, sourced from the Hugging Face Datasets. Dataset. It is a bilingual translation dataset commonly used for training and evaluating machine translation models. It consists of sentence pairs in two languages (source and target). Each entry in the dataset contains a "translation" key with sub-keys for the source and target languages, holding the respective sentences.

3.1 Pre-processing

1. Tokenization of source and target sentences. I used the Hugging face tokenizer library that contains prebuilt tokenization methods for splitting sentences into token representations. This includes, Tokenizer, WordLevel, WordLevelTrainer, Whitespace class methods.
2. Splitting the dataset into training and validation sets. I preform a 90/10 split and utilise the PyTorch helper method random_splits() to shuffle data.
3. Creating a PyTorch Dataset class to handle data loading. The Bilingual-Dataset class inherits from PyTorch's Dataset class and is designed to handle tokenization and padding of sentences.
    - Initialization: The class constructor initializes tokenizers, source and target languages, and sequence length. Special tokens (start of sequence [SOS], end of sequence [EOS], and padding [PAD]) are also initialized (`self.sos_token`, `self.eos_token`, `self.pad_token`).
    - Length: The `__len__` method returns the length of the dataset.
    - Get Item: The `__getitem__` method retrieves and processes individual data points:
        - Source and target texts are tokenized

```
enc_input_tokens = self.tokenizer_src.encode(src_text)
dec_input_tokens = self.tokenizer_tgt.encode(tgt_text)
```

        - Padding is added to ensure each sequence reaches the defined length (`self.seq_len`), with special tokens [SOS] and [EOS] added appropriately.
        - Masks for the encoder and decoder are created to handle padding during training.

4 Experiments

4.1 Training

All of the implementation details of the training defined in this sub-section is contained in the accompanying code file **train.py**. All of these experiments were run on a A100-80G gpu machine on Paperspace.

Experiment	Description	Training Duration
1	Invalid Attempt: The model had a design flaw that incorrectly overwrote padding tokens when inputs were longer than the allowed sequence length hyper-parameter.	40 mins : 2 Epochs
2	Invalid Attempt: I noticed that the model produced invalid predictions due to a flaw in the mapping layer that didn't map output embeddings correctly.	2.5 hours : 5 epochs
3	Main Results: final run produced "acceptable" results but due to time and cost constraints these results were the source of truth	7 hours : 15 epochs

The main training script was the invocation point of all the other python scripts I had; Below I show the basic structure of the script and the different components.

4.1.1 Data Loading and Pre-processing

Load dataset using load_dataset(); Build tokenizers for source and target languages; Split dataset into training and validation sets. Define custom collate_fn(); Create DataLoader for training and validation datasets; this was very simple due to using the nn.Dataset class inhertor from PyTorch.

4.1.2 Model Initialization

Build transformer model using build_transformer(); I found a useful PyTorch function, nn.init.xavier_uniform_(p) which calculates intial parameters for the nn.Module class. Initialize TensorBoard writer for logging, this was useful for tracking the training loss and evaluation metrics below. Set up weights optimizer (Adam). I set up a persistant storage for the weight files as to be able to go back in time and inferencing, this was also quite useful to load pre-trained model if specified and continue training in case of interruption.

4.1.3 Training Loop

Iterate through epochs; Clear CUDA cache; Set model to train mode; Loop through batches in DataLoader; Process input through encoder and decoder; Compute loss using nn.CrossEntropyLoss; Log loss to TensorBoard; Backpropagation preformed using nn.CrossEntropyLoss, and because the Transformer class inherits from the nn.Module, it is possible to pass the current Pytorch device to the loss function and and backpropagation is invoked automatically via the loss.backward() function, this is preformed on the Projection/Mapping layer weights; Update weights, with the Adam optimizer; Run validation at the end of each epoch, defined below; Save model checkpoint. I show the loss over time in 2.

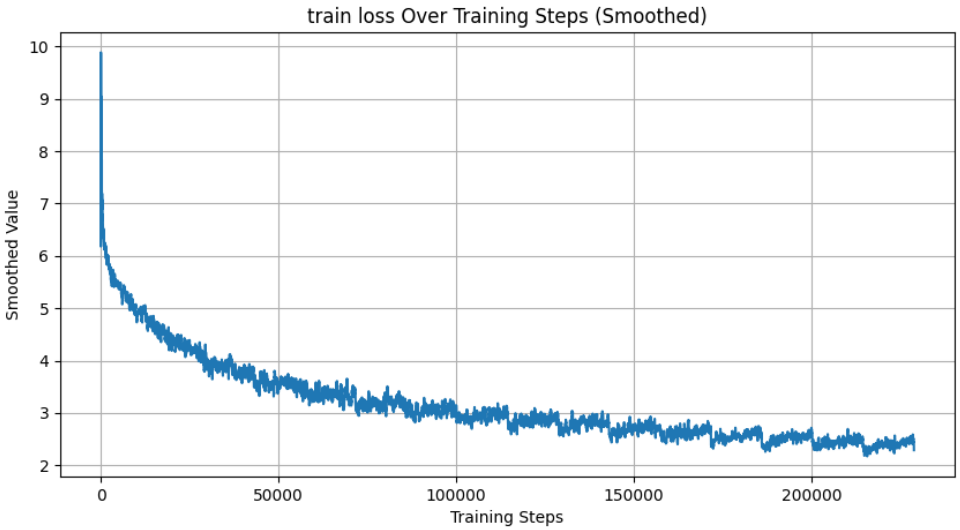

Figure 2: Training Loss

4.1.4 Attention Matrix Visualisation

I included a file **attention_maps.html** that can be viewed form the browser to represent the attention embeddings for a random input using the last weights file for experiment 3. It is too big to include in this report.

4.2 Evaluation

All of the implementation details of the validation and score visualisation defined in this subsection is contained in the accompanying code files **validation.py**, **visualisation.py**

4.2.1 BLEU Score

The BLEU (Bilingual Evaluation Understudy) score is a metric for evaluating the quality of text translated by machine translation (MT) systems against a set of reference translations [1]. BLEU compares the n-grams (unigrams, bigrams, trigrams, and four-grams) of the MT output with those of the reference translations. It calculates the precision of these n-grams, which is the proportion of n-grams in the MT output that appear in the reference texts [1]. To avoid giving undue credit to the overuse of certain words, BLEU applies a clipping mechanism. For each n-gram in the MT output, its count is clipped to the maximum number it appears in any single reference sentence. This prevents the score from being biased towards repetitive language in the MT output. The precision for each n-gram type is calculated by summing the clipped counts of matching n-grams in the MT output and dividing by the total number of n-grams in the MT output. To counter the limitation where shorter translations may artificially increase precision, BLEU includes a brevity penalty. If the total length of the MT output is shorter than the reference translations, a penalty is applied, calculated as [1]:

$$BP = e^{(1-r/c)}$$

Where r is the length of the reference translation and c is the length of the MT output. If the MT output is longer than the reference, the BP is set to 1 (no penalty). The overall BLEU score combines the geometric mean of the n-gram precisions with the brevity penalty. The formula is [1]:

$$\text{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Where w_n are weights assigned to each n-gram precision p_n, typically set to equal values summing to 1. The BLEU score ranges from 0 to 1, where 1 indicates a perfect match with the reference translations and 0 indicates no overlap. [1]

For The Transformer model, I struggled a bit during training to capture the BLEU, only after training completed I realised that the BLEU score was not saving correctly:

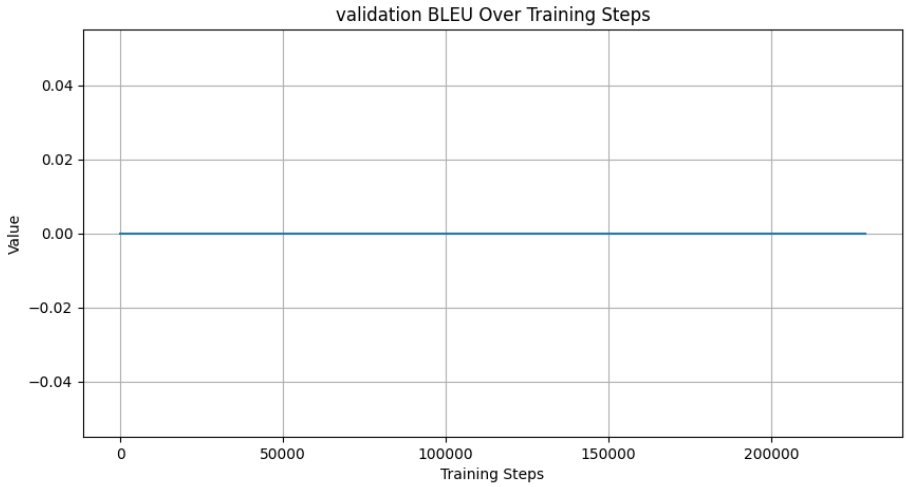

Figure 3: Invalid Bleu scores accumulated during last experiment

I managed to correct the issue after training, but due to time and cost constraints i ran the BLEU validation on my local machine with the last weight file to show that there was BLEU scores for some of the predictions. I ran a total of 100 phrases through the translate function in **translate.py** for each weights file to produce some output, which should suffice as sample representation of the score:

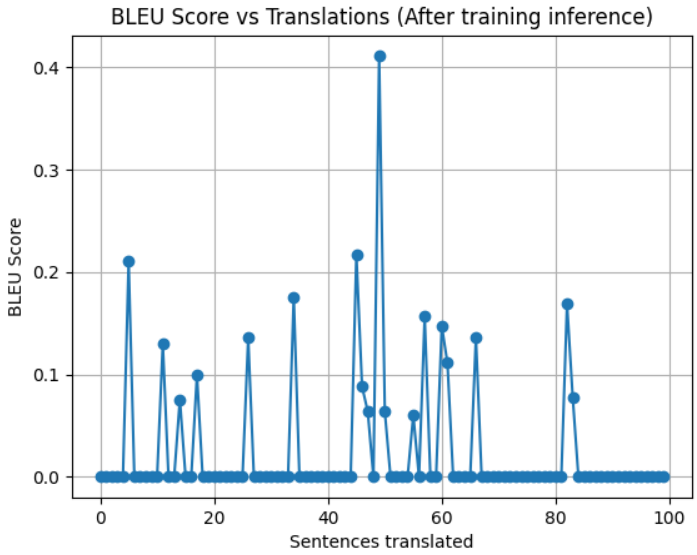

Figure 4: Bleu scores with trained weights

4.2.2 Word Error Rate

The Word Error Rate (WER) is a metric used to evaluate the performance of machine translation (MT) models by measuring the edit distance between the translated sentence and a reference sentence [1]. The edit distance is the number of edits (insertions, deletions, substitutions) required to change the translation into the reference. Here's a detailed explanation based on the provided text; Edit Types are Substitutions (Sub), Replacing a word in the MT output with a word from the reference; Deletions (Del), Removing words from the MT output that aren't in the reference; Insertions (Ines), Adding words from the reference into the MT output that weren't originally there. The WER starts by aligning the words in the MT output with the words in the reference sentence, usually using dynamic string alignment [1].

$$WER = \frac{Sub + Del + Ines}{N}$$

Here, N is the total number of words in the reference sentence, calculated as the sum of substitutions, deletions, and matched words (Sub + Del + M).

For The Transformer model I included the WER metric during training:

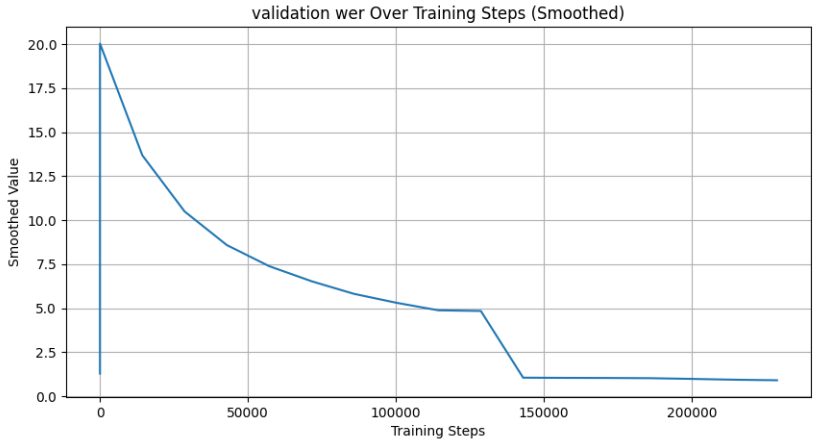

Figure 5: Word Error Rate

4.2.3 Character Error Rate

The Character Error Rate (CER) is another metric used in machine translation evaluation, similar in concept to the Word Error Rate (WER) but operating at the character level [1]. CER measures the edit distance between the MT output and the reference translation at the character level. This involves calculating the minimum number of character-level edits (insertions, deletions, substitutions) needed to transform the MT output into the reference [1].

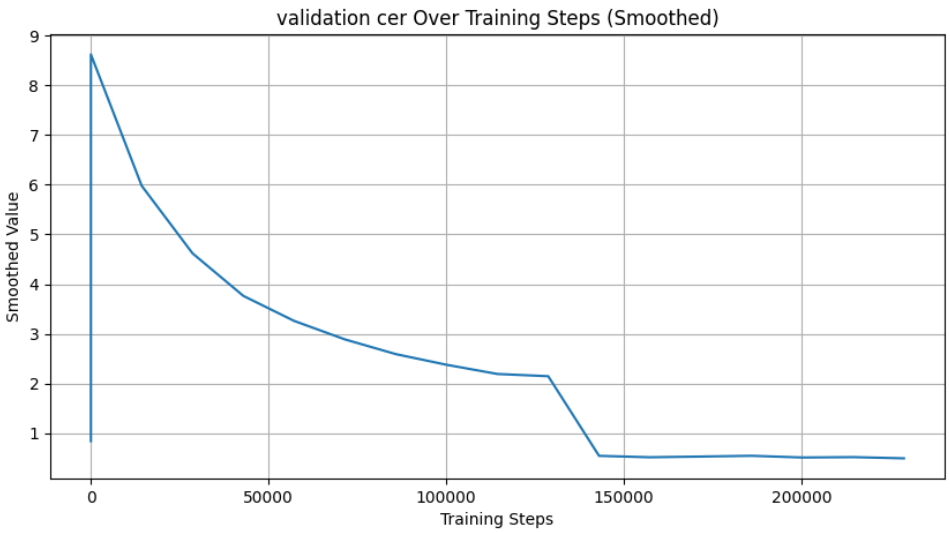

Figure 6: Character Error Rate

4.2.4 Inference Output

I created a inference function **translate.py** and **bleu.py**.

```
        ID: 445
   SOURCE: Two or three times, during the month of January and the first fortnight
   ↪  of February, I was roused out of my sleep in that way.
   TARGET: À deux ou trois reprises, durant le mois de janvier et la première
   ↪  quinzaine de février, je fus ainsi tiré de mon sommeil.
PREDICTED: Deux ou trois fois , durant le mois de janvier et deux mois de janvier
↪  , après , je fus long de février , je fus réveillé .  BLEU Score:
↪  tensor(0.2171)

       ID: 446
   SOURCE: Admiral Meaulnes was there, on foot, all equipped, his cloak on his
   ↪  back, ready to start, and every time, on the edge of that mysterious
   ↪  country into which he once already had ventured, he stopped, he hesitated.
   TARGET: Le grand Meaulnes était là, dressé, tout équipé, sa pèlerine sur le
   ↪  dos, prêt à partir, et chaque fois, au bord de ce pays mystérieux, où une
   ↪  fois déjà il s'était évadé, il s'arrêtait, hésitant.
PREDICTED: Le grand Meaulnes était là , à pied , tout équipé , tout équipé , il
↪  était là , prêt , le soir , tout armé , sur le soir , au milieu de cette
↪  campagne .  BLEU Score: tensor(0.0882)

       ID: 447
```

```
   SOURCE: At the moment of lifting the latch of the door to the stairs and of
   ↪  slipping off by the kitchen door, which he could easily have opened without
   ↪  being heard, he would shrink back once more . . . Then, during the long
   ↪  midnight hours he paced feverishly the disused lumber- rooms, lost in
   ↪  thought.
   TARGET: Au moment de lever le loquet de la porte de l'escalier et de filer par
   ↪  la porte de la cuisine qu'il eût facilement ouverte sans que personne
   ↪  l'entendît, il reculait une fois encore... Puis, durant les longues heures
   ↪  du milieu de la nuit, fiévreusement, il arpentait, en réfléchissant, les
   ↪  greniers abandonnés.
PREDICTED: Au moment où le loquet de la porte , le loquet de la porte de la porte
↪  de la porte de la cuisine , qu ' escalier , qu ' escalier , qu ' escalier , qu
↪  ' heure après l ' oreille , il aurait facilement , il aurait pu facilement ,
↪  il se  , il se  , il se  , il ne se  , il se  , il ne se  , il se  , il se  ,
↪  il se  , dans la  , il ne se  , il se  , il se  , il se  , il se  , il se  ,
↪  il se  , il se  , il se  , il se  , il ne se  , en  , en  , en  , en  , en
↪  était dans la porte , en  , en  , en était tout à la , en était encore .  BLEU
↪  Score: tensor(0.0639)

       ID: 448
   SOURCE: At last one night, towards the fifteenth of February, he woke me up by
   ↪  gently placing his hand on my shoulder.
   TARGET: Enfin une nuit, vers le 15 février, ce fut lui-même qui m'éveilla en me
   ↪  posant doucement la main sur l'épaule.
PREDICTED: Une fois , vers le soir , vers le quinzième de février , il m ' une me
↪  réveilla , il m ' épaule , il me traînant , il me  .  BLEU Score: tensor(0.)

       ID: 449
   SOURCE: The day had been very disturbed.
   TARGET: La journée avait été fort agitée.
PREDICTED: La journée avait été très agitée .  BLEU Score: tensor(0.4111)

       ID: 450
   SOURCE: Meaulnes, who had now entirely dropped out of all the games of his
   ↪  former comrades, had remained seated at his bench during the last
   ↪  recreation of the evening, busily sketching out a mysterious plan,
   ↪  following it with his finger and elaborately measuring it out on the atlas
   ↪  of the Cher.
   TARGET: Meaulnes, qui délaissait complètement tous les jeux de ses anciens
   ↪  camarades, était resté, durant la dernière récréation du soir, assis sur
   ↪  son banc, tout occupé à établir un mystérieux petit plan, en suivant du
   ↪  doigt, et en calculant longuement, sur l'atlas du Cher.
PREDICTED: Meaulnes , entièrement sorti entièrement au milieu de tous les
↪  dernières  de ses dernières récréation de la dernière récréation de son
↪  premier plan , Meaulnes , Meaulnes , était resté , Meaulnes , Meaulnes , s '
↪  après la dernière récréation du soir , tout en suivant , tout en suivant , en
↪  suivant un plan mystérieux , Meaulnes .  BLEU Score: tensor(0.0636)

       ID: 451
   SOURCE: There was an incessant going and coming between the playground and the
   ↪  classroom.
   TARGET: Un va-et-vient incessant se produisait entre la cour et la salle de
   ↪  classe.
PREDICTED: Il se après être rapidement à peu à peu à travers dans la cour , il y
↪  aller .  BLEU Score: tensor(0.)
```

5 Conclusion

The Output for the model seems to match at first glance, running some of these translations through a tool like google translate yields mixed results, sometimes closely representing meaning and other times definitely lacking in disposition and meaning, while word accuracy is mostly correct.

Take for example ID 449, being very good match almost identical in meaning has a lower score on the bleu due to the disposition of words.

```
SOURCE: The day had been very disturbed.
TARGET: The day had been very hectic.
PREDICTED: The day had been very hectic
```

I also noticed some examples get stuck in generation loops either printing invalid tokens or repeating certain words numerous times, some of these sentences have a non zero BLEU score which is concerning if these inputs could affect the loss positively it might become a problem.

The model does show a lack of training time and parameter refinements, in the final quality of the training, but the effort in building the model definitely out wayed the runtime. It is prominent however that although this model was at the time an improvement upon existing recurrence networks, that there is however to some degree new limitations with this model which became more prevalent during constructing the components. The fixed cost of interacting with inputs in outputs in a iterative way, albeit efficient due to shared attention parameter, is slow for longer input sizes. Interesting solutions for this do exist, example being models like MAMBA [2] which is a state space model which circumnavigates the fundamental limitations of the transformer architecture.

References

[1] Shweta Chauhan and Philemon Daniel. A Comprehensive Survey on Various Fully Automatic Machine Translation Evaluation Metrics. *Neural Processing Letters*, 55(9):12663–12717, December 2023.

[2] Albert Gu and Tri Dao. Mamba: Linear-Time Sequence Modeling with Selective State Spaces, December 2023. arXiv:2312.00752 [cs].

[3] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training Recurrent Neural Networks, 2012. Version Number: 2.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, August 2023. arXiv:1706.03762 [cs].