Language Modelling with Transformers

Roger Bukuru (BKRROG001)

University of Cape Town
Natural Language Processing (CSC5035Z)
Assignment Report

1 Introduction

In this assignment, we explored language modelling with Transformers, specifically we explored character-level language modelling. We worked with data from the NCHLT monolingual corpora, where we utilized the isiXhosa language data set for our modelling and evaluation. Basic cleaning had already been performed. We proceeded by performing additional data pre-processing, implementing a transformer language model, fine-tuning it, implementing two intermediate and a single advanced extension and finally testing the model on a test data set to assess it's character-level language modelling performance.

2 Data Processing

Before processing our data through the transformer we first implemented the following necessary pre-processing steps.

Character tokenization: We split the text data into individual characters.

Input Embedding: To create our initial input embeddings we proceeded to first create a vocabulary by mapping from unique characters to integer IDs, thereafter we converted the characters in a given dataset to their corresponding integer IDs.

Sequence Preparation: In order to define the input and target pairs for the language model, we split the data into sequences of a fixed length (plus 1). This means that the dataset is reshaped into sequences of a fixed length $sequence_length + 1$ where the last character in each sequence serves as the target for prediction.

Batching: We grouped the data into batches for parallel processing.

Data Shuffling: We shuffled the data to ensure that the model does not learn patterns specific to the order of the data.

Masking: The last technique we implemented was generating attention masks to ensure the model looks only at past tokens during training.

3 Language Model Training

We implemented a standard transformer decoder layer as per [1]. Some key considerations regarding our implementation was the inclusion of dropouts. This was implemented after the Multi-Head Attention, The Feed Forward Layer and finally on the final output before calculating the probabilities.

We then proceeded to train our language model on this architecture. Training was conducted by evaluating each training batch across 50 epochs with a baseline Adam optimizer. For efficient training we combined this with our hyperparameter tuning which we describe next and the parameters that were evaluated are shown in table 1 below.

Hyperparameter	Values	Type	Description
Batch Size	32, 64	Optimization	Number of sequences per batch
Sequence Length	64, 128	Architecture	Length of each input sequence
d_m	128, 256	Architecture	Model dimension
Number of Heads	4, 8	Architecture	Number of attention heads
Number of Layers	2, 4	Architecture	Number of transformer layers
FF Widening Factor	4, 8	Architecture	Widening factor for feedforward layers
Learning Rate (LR)	1×10^{-3}, 2×10^{-3}	Optimization	Learning rate for optimizer
Dropout Rate	0.1, 0.2	Architecture	Dropout rate applied during training

Tab. 1: Hyperparameter tuning ranges for the character-level language model.

4 Hyperparameter Tuning and Evaluation

As highlighted training was conducted in conjunction with hyperparameter tunining, the parameter values as well as what they were intended to tune is shown in table 1, this was manually executed on a single *T4 GPU* on Google Colab.

The goal of the hyperparameter tuning process was to identify the best set of hyperparameters that minimize the Bits-Per-Character (BPC) on a validation dataset, with the aim of improving the model's performance on the test dataset. For each combination of hyperparameters, the model was trained over a specified number of epochs, in this case 50 epochs (computation was too expensive to go beyond this). During each epoch, the model's performance was evaluated by calculating both the training loss and BPC on the training dataset, as well as the validation loss and BPC on a validation dataset. The tuning was conducted without any intermediate or advanced extensions. Based on the limited computational resources at our disposal, the optimal parameters obtained are shown in table 2 below:

Hyperparameter	Best Value
Batch Size (BS)	32
Sequence Length (Seq_Len)	128
Model Dimension (DM)	256
Number of Heads (NH)	4
Number of Layers (NL)	2
Feedforward Widening Factor (FF_WF)	4
Learning Rate (LR)	0.001
Dropout Rate (DR)	0.2
Training Metrics	**Values**
Train Loss	1.2425
Train BPC	2.9130
Validation Metrics	**Values**
Validation Loss	6.8579
Validation BPC	10.4591

Tab. 2: Best Hyperparameters and Corresponding Training and Validation Metrics.

We observe that additional training and hyperparameter tuning is required in order to improve the validation BPC as it was considerably higher than 1, this would require additional computation.

Finally using the identified hyperparameters, the model was re-initialized with these hyperparameters and its associated network parameters and evaluated on the test dataset. The test dataset was processed batch by batch and the average test loss and BPC were calculated across all batches. The final results are shown in table 3 below:

Metric	Value
Test Loss	9.1150
Test BPC	13.2339

Tab. 3: Test Performance Metrics

We observe that the results were sub-par, this was primary the result of limited training resources.

5 Model Extensions

Our model extension can be categorized into two categories, namely intermediate and advanced extensions. We highlight our work for each category.

5.1 Intermediate Extensions

In the intermediate category we explored the following:

1. Compare doing layer normalization before or after the attention and feedforward layers.
2. Investigate the effect of weight tying between the embedding and output layers.

Given we had limited resources, obtaining new hyperparameters was not feasible, so we conducted both extensions by using the best hyperparameters obtained earlier and then re-trained the model with the proposed extension. With the newly obtained parameters we evaluated the performance of the extensions and show the results in the table 4 below:

Extension	Train Loss	Train BPC	Test Loss	Test BPC
Baseline(Standard architecture with no Weight Tying)	1.2425	2.9130	10.28	14.9200
Layer Norm Before Attention and FF	1.1195	1.6281	10.9623	15.8224
Weight Tying	1.1394	1.6499	8.1933	11.8207

Tab. 4: Performance Metrics for Different Model Extensions

Evaluating the results shown in 4, we observed that our the weight tying extension model although subpar outperformed both our baseline model and the other extension, as it had a lower Test BPC based on the same architecture. We do however note that given additional computation and more robust training i.e hyperparameter tuning on the extensions, and then testing may yield different results.

5.2 Advanced Extensions

In terms of advanced extensions we explored advanced optimization methods or learning rate schedules that have been proposed for training Transformers, specifically we looked at using our standard optimizer (Adam) with and without a learning rate schedule and then extended this by using an advanced optimization variation called AdamW. For this extension we also assessed it's performance with and without a learning rate schedule.

Similary due to limited resources we analyzed the performance of the above optimizers and learning rate schedules using the best hyperparameters we obtain earlier. We show are results in table 5 below:

Configuration	Train Loss	Train BPC	Test Loss	Test BPC
Adam (No LR Schedule)	1.0989	1.8061	9.1150	13.2339
Adam (With LR Schedule)	3.2755	4.7496	6.8204	9.8592
AdamW (No LR Schedule)	1.1064	1.6651	7.2807	10.5589
AdamW (With LR Schedule)	3.2772	4.7511	6.7852	9.8072

Tab. 5: Performance Metrics for Different Optimization Configurations

The results show that for both optimizers, optimizing with a learning rate schedule proved to be more optimal, this shows that potential further exploration in this direction may yield better results in the aim of achieving better convergence and stabilizing our training.

6 Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.