Zaid Abdulrehman

Professor Jesse Thomason

CSCI 499 Natural Language Processing

10 November 2022

Hw#3 seq2seq

How To Run

https://github.com/zabdulre/CSCI499_NaturalLanguageforInteractiveAI/tree/main/hw3

Please note, you may select/unselect the transformer model using the –transformer flag.

```
python train.py --in_data_fn=lang_to_sem_data.json --model_output_dir=experiments/s2s
--batch_size=1000 --num_epochs=15 --val_every=1 --force_cpu --transformer
--embedding_size=64 --encoder_size=128
```

Report

In this coding assignment, I have successfully implemented a sequence to sequence model, with an optional transformer component. The implementation learns successfully, and is able to process the entire dataset within reasonable training time. As a sequence to sequence model, an encoder-decoder structure is utilized. Teacher-forcing and greedy-decoding is supported. Furthermore, additional metrics have been implemented in order to illustrate the model's learning progress.

Implementation Decisions

There have been multiple design iterations for this project, each informing the current implementation details. In the first iteration of this model, the encoder was finalized. Tokenized Data is fed through the encoder in a batch-wise manner. Text preprocessing groups instructions by episode, and one-hot encodes each word in an episode. The maximum length of an encoding is set to the maximum number of instructions in an episode, and the maximum number of words in an instruction is set to two standard deviations above the mean. As part of the encoding, all episodes have a start and end token as well as padding tokens, both for the labels and features. The encoder consists of an embedding layer and an LSTM. In the first design iteration, problems with the decoder became apparent. Firstly, the decoder was using the encoder output as *input* to its internal LSTM. However, the decoder also took action and target embeddings as input in subsequent decoding steps. This led to a less efficient implementation, in which the output of the embedding layer for the decoder required further processing before it could be used as input for the LSTM. The main issue which I encountered in the first implementation was a break in the computation graph. This issue became apparent when, during training, the training and validation loss did not increase or decrease at all. I suspected this issue was either likely due to the one-batch-at-a-time loop which I had implemented for decoding. Furthermore, running the decoder one-batch-at-a-time had led to a very slow implementation.

In the second iteration, I addressed performance issues and had the first working example. In this iteration, decoding was done for all batches, one action-target pair at a time. This allowed for a more concise and much faster implementation. Greedy decoding was implemented using the same method; predicted tokens are calculated simply by using the maximum. The training loop was completed. As part of training, two instances of cross entropy are used, one for targets and one for actions. Both losses are added together. The Adam optimizer is used based on its quicker convergence than stochastic gradient descent.

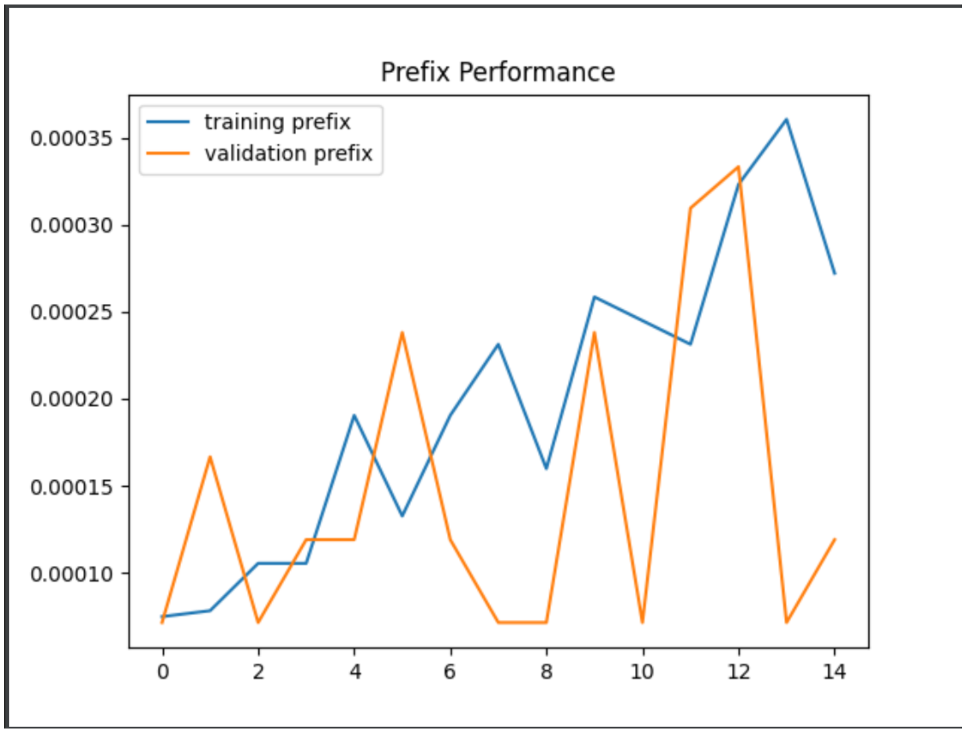
As part of the current implementation iteration, I added a transformer component as well as metrics. To incorporate a transformer in this architecture, I utilized a transformer encoder layer for the embedding layer in the encoder. It is my understanding that there is no requirement to use a transformer-only encoder or decoder; the requirement is to use a transformer. I chose to use a transformer for embedding, as it was most compatible with the decoder; output shapes did not change. Finally, I modified the training loop to calculate the prefix match score, the target accuracy and loss, and the action accuracy and loss. Due to compute constraints, the accuracy and prefix match scores are calculated every 10 batches each epoch. The accuracy score counts the number of correct guesses, except where the original token is a pad token.

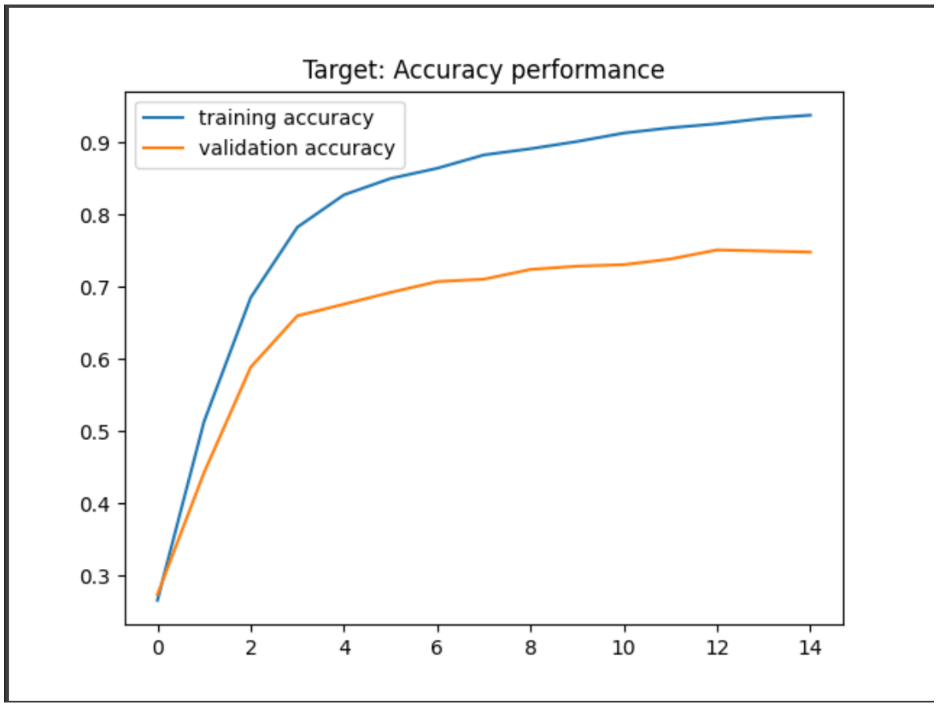
Model Configuration

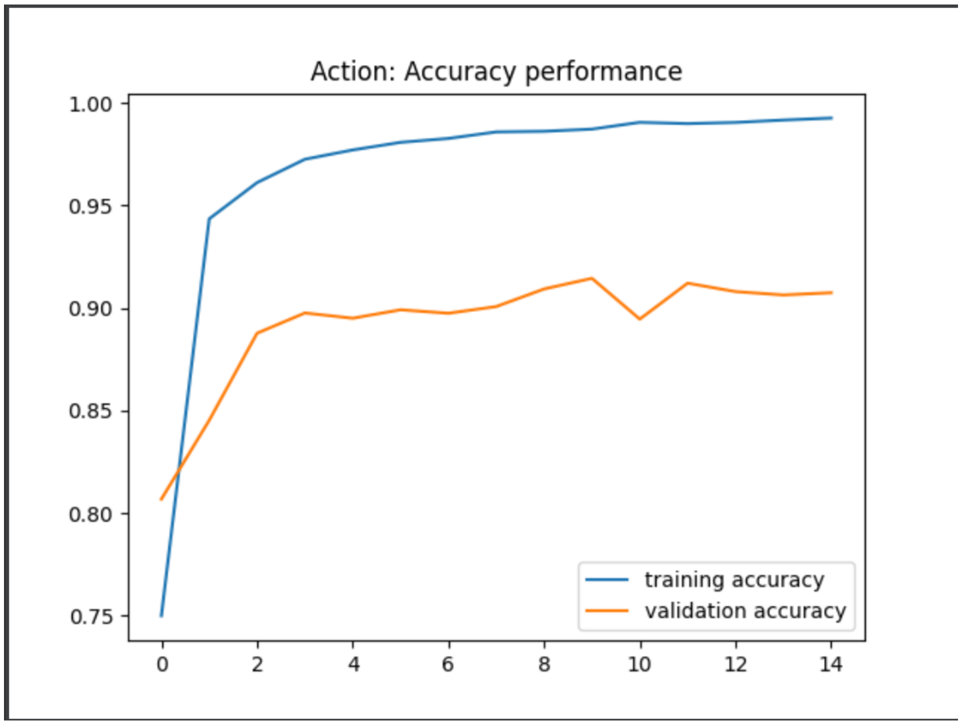
The model is trained using two instances of cross entropy loss. The Adam optimizer is used. Using the recommended run command provided at the beginning of this document, the model completes training on the entire dataset in approximately one hour. An embedding size of 64 and encoder/decoder hidden size of 128 is recommended based on training time. However, it is likely that increasing the embedding size and possibly increasing the encoder and decoder hidden size may marginally improve accuracy. The current configuration is able to showcase the model’s learning trajectory and performance in using greedy and teacher-forced decoding.

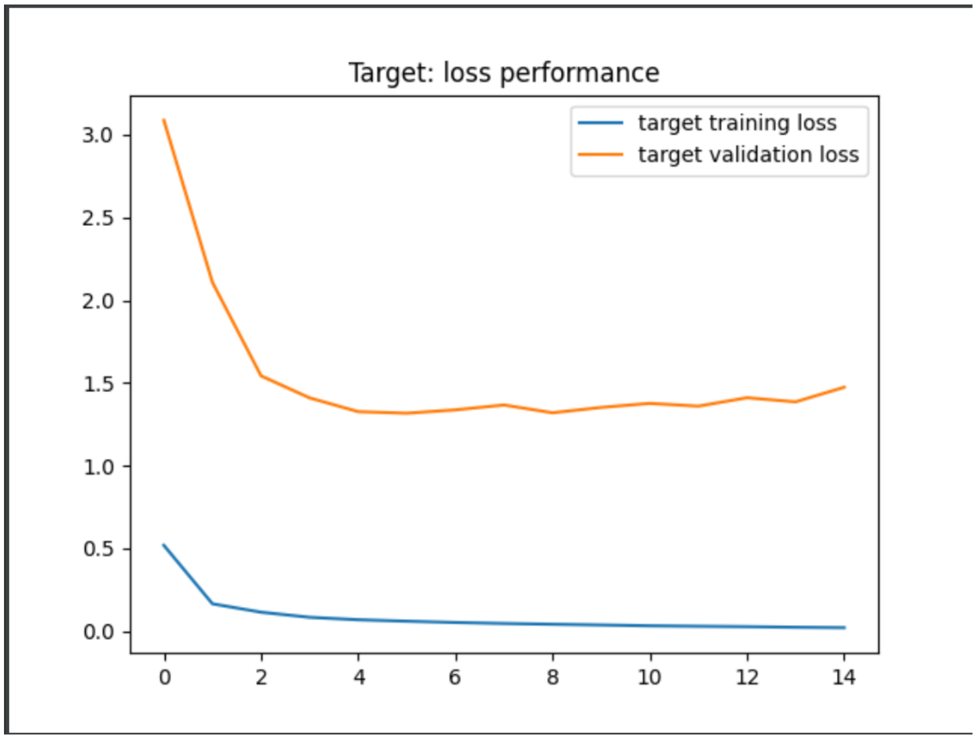
Results

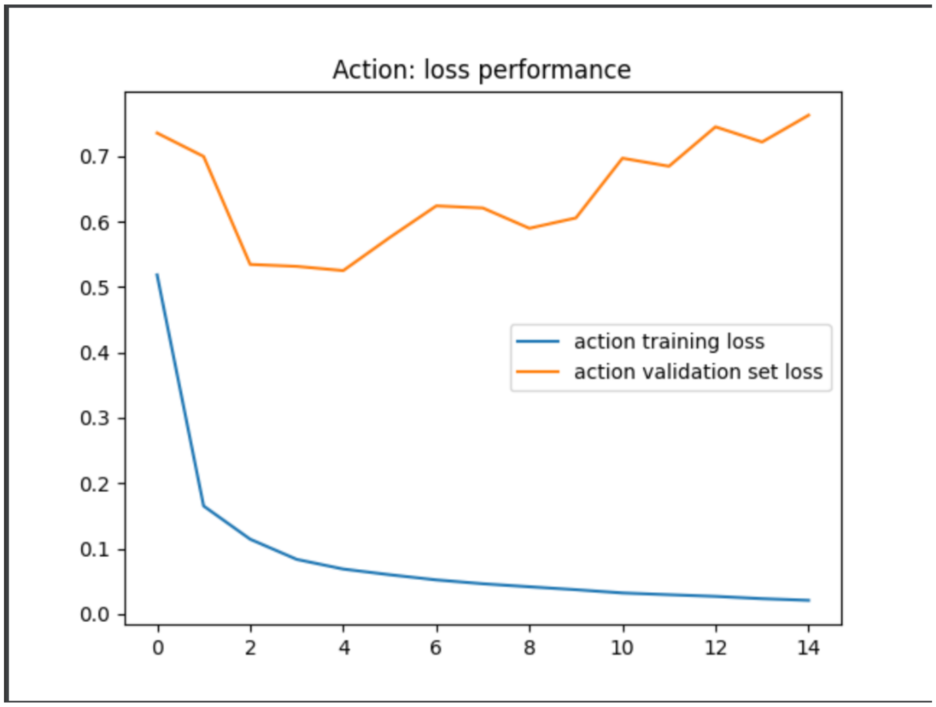
Results running on the recommended configuration *without* the transformer component:



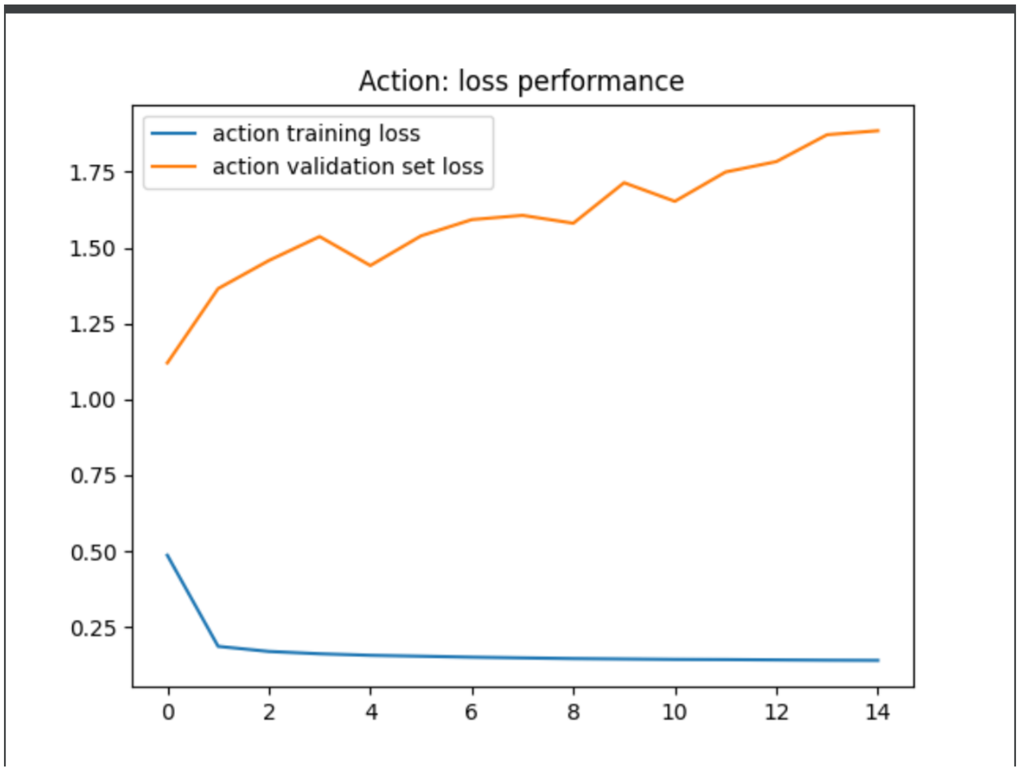


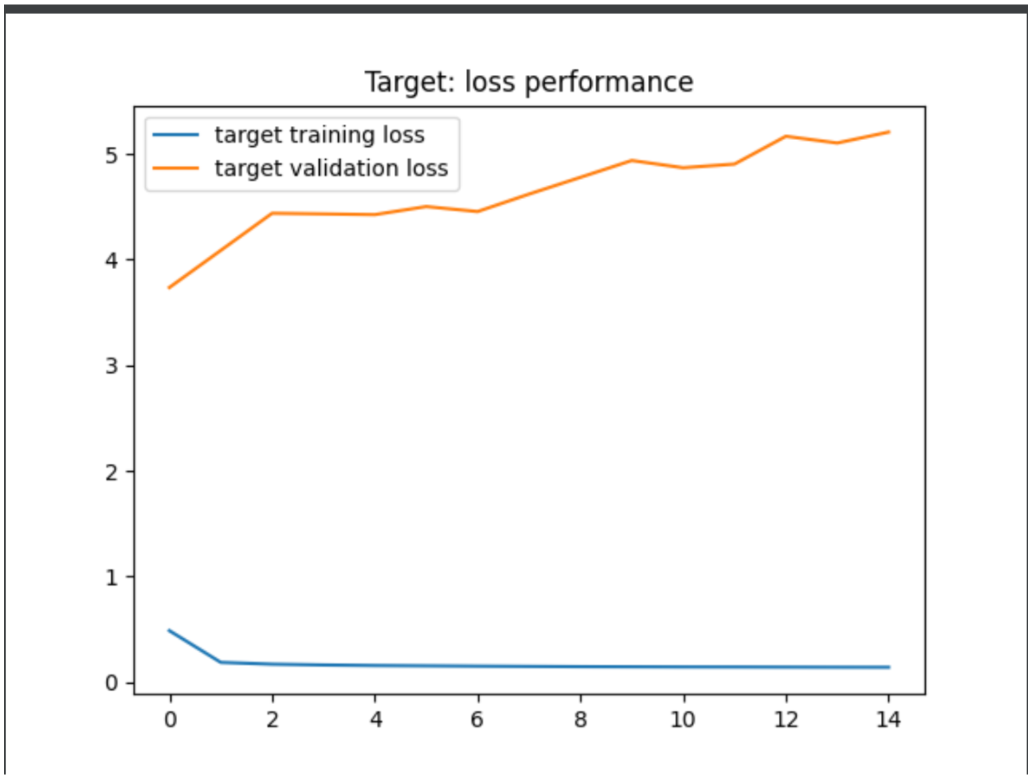

Action: Accuracy performance
1.00
0.95
0.90
0.85
0.80
0.75
0 2 4 6 8 10 12 14
training accuracy
validation accuracy

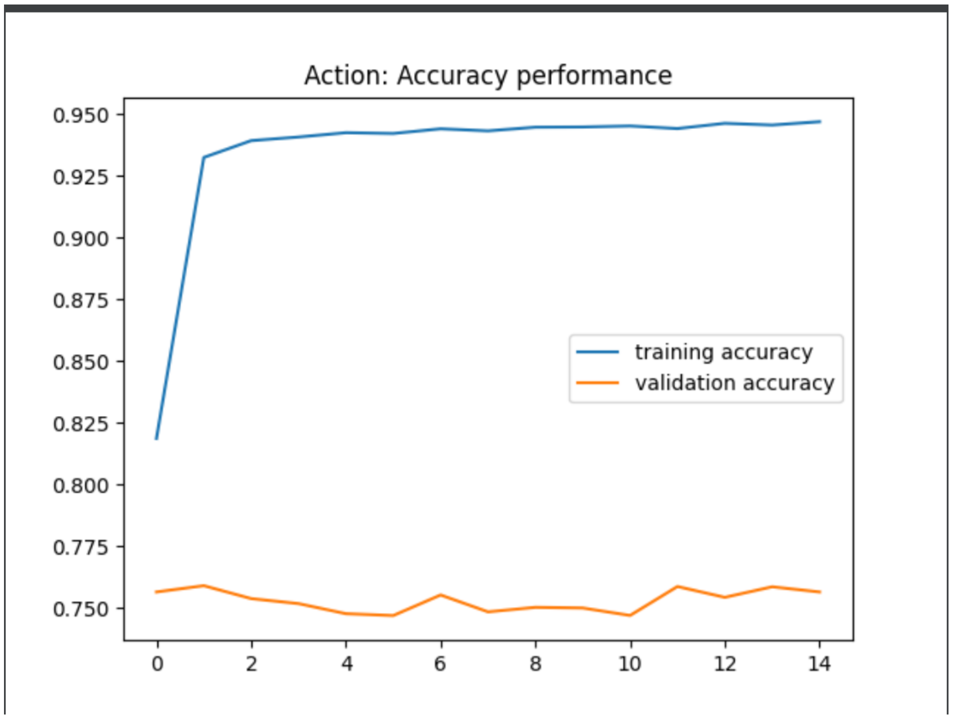

Target: loss performance
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0 2 4 6 8 10 12 14
target training loss
target validation loss

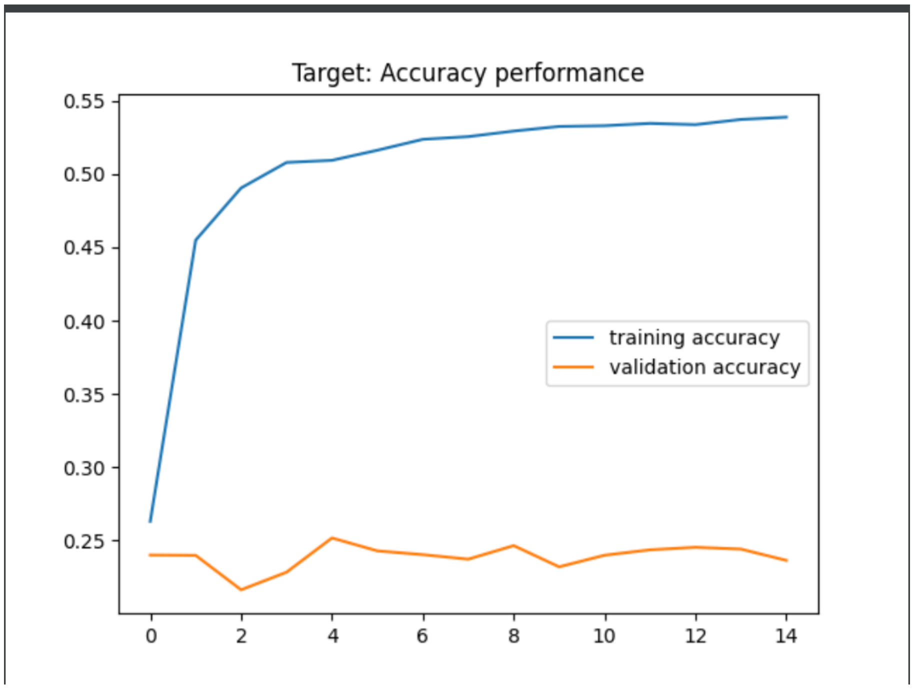


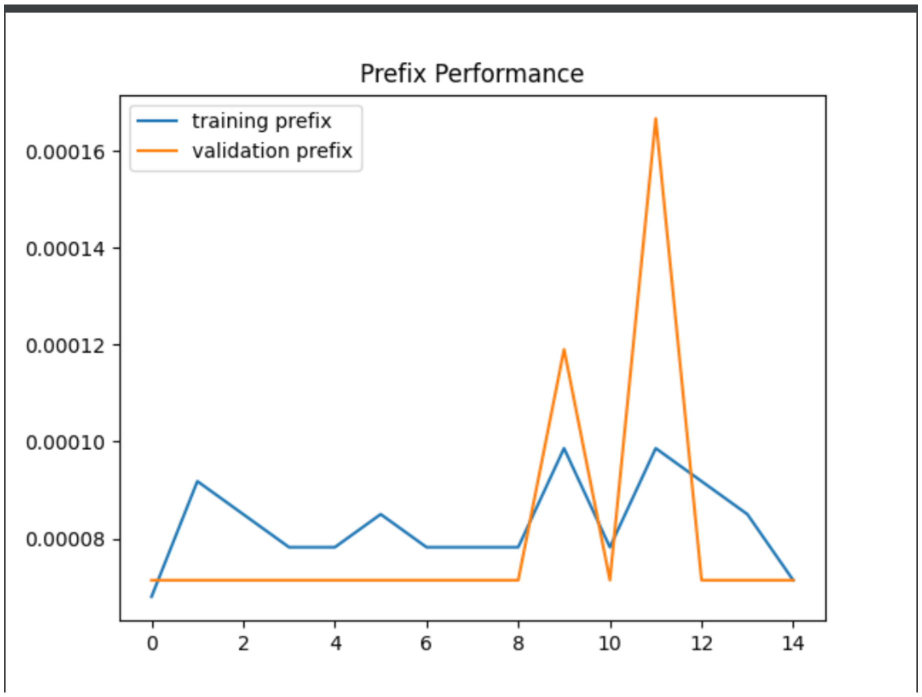
Results running on the recommended configuration *with* the transformer component:




Target: loss performance
target training loss
target validation loss
0
1
2
3
4
5
0
2
4
6
8
10
12
14


Action: Accuracy performance
training accuracy
validation accuracy
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
0
2
4
6
8
10
12
14


Target: Accuracy performance
training accuracy
validation accuracy
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0
2
4
6
8
10
12
14


Prefix Performance
training prefix
validation prefix
0.00016
0.00014
0.00012
0.00010
0.00008
0
2
4
6
8
10
12
14

Analysis

Both models achieved over 50% validation accuracy for targets. Both models achieved a validation accuracy over 75% for actions. In both metrics, the model without the transformer performed better.

In the model without the transformer component, peak validation accuracy continued to improve well into training. The validation action loss, however, began to increase while the training action loss began to decrease near the end of training. This indicates that the model was likely plateauing in terms of action performance. Further training is likely to lead to overfitting for the action category. For targets, however, training and validation loss remained approximately decreasing. As a result, it is likely that the model may benefit from increased embedding dimension size or training time in terms of target prediction. The prefix match performance of the model varied significantly between epochs. However, a strong upward trend is apparent in training epochs and a light upward trend is present in validation epochs. This is likely due to the difference in teach-forced and greedy decoding predictions. The model in training performed significantly better in all categories, especially prefix matching. So, having the correct previous token as input to the decoder is very likely an important factor in model performance. Based on the results, two possible areas of improvement for the model may be; an increase in embedding size in order to better fit to target outputs, and possibly a separating token between instructions in order to make the model more likely to predict tokens correctly in succession.

The model with the transformer component generally performed worse. The model has a significantly lower action and target accuracy. Furthermore, the prefix match accuracy is lower on the validation set. The results indicate that the model is hindered by having only one transformer embedding layer. It is possible that adding more layers may improve model performance. Both action validation loss and target validation loss increase as training loss remains relatively the same. This behavior emerges early in the training phase, indicating that the model is hindered by the architecture rather than a lack of data, for example. Despite this, the model is able to learn adequately. Consistently achieving over 75% action accuracy and ~25% target accuracy. Based on the results, the transformer-based model can likely be improved by either increasing the number of transformer layers, or by increasing the number of transformer output heads.