# Deep Learning Ex3

Or Shkuri, Nadav Cohen

July 2024

## 1 Shakespeare data

For the Shakespeare data, we ran our model using $50k$ batches, which is $1.6M$ training sequences. Our model contains $3.61M$ parameters. After training the model, we got to a loss value of 0.2877.

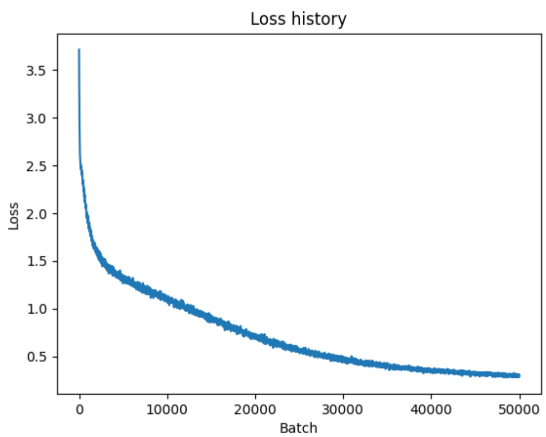



Fig. 1: Training history plot showing the loss value for each iteration.

## 2 Hyperparameters

For this model, we used the following hyperparameters:

| **Hyperparameter** | **Value** |
|---|---|
| Sequence Length | 128 |
| Batch Size | 64 |
| Number of Layers | 8 |
| Number of Heads | 8 |
| Embedding Size | 192 |
| Learning Rate | $5 \times 10^{-4}$ |
| Gradient Clipping | 1.0 |
| Weight Decay | $1 \times 10^{-4}$ |
| Temperature | 0.6 |

Table 1: Hyperparameters Used for Training

## 3 Hyperparameters modifications

We tried using different learning rates, however $5e-4$ worked the best for us. Also, we added a weight decay of $1e-4$ and a temperature of 0.6 to make the model more concise while sampling words using the softmax distribution. We tried using a Dropout with a rate of 0.1, however, we got poor results and decided to "drop" it. Moreover, we changed the number of layers and heads to 8, to make our model more expressive. Of course, in a real project, we would optimize our hyperparameters using a Grid Search.

## 4 Hebrew data results

For the Hebrew data, we used the same hyperparameters that we achieved from the Shakespeare data, and we ran our model with the same amount of batches and sequences. Finally, we achieved a loss value of 0.1691.

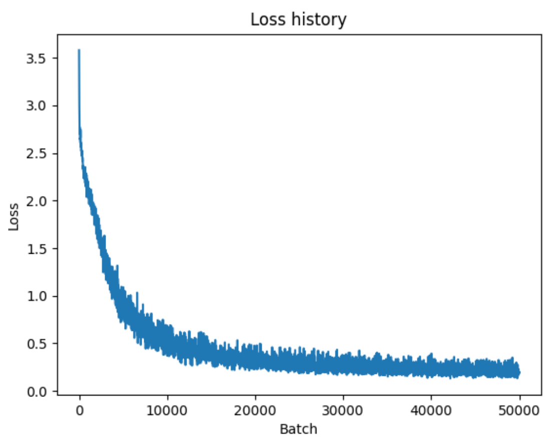
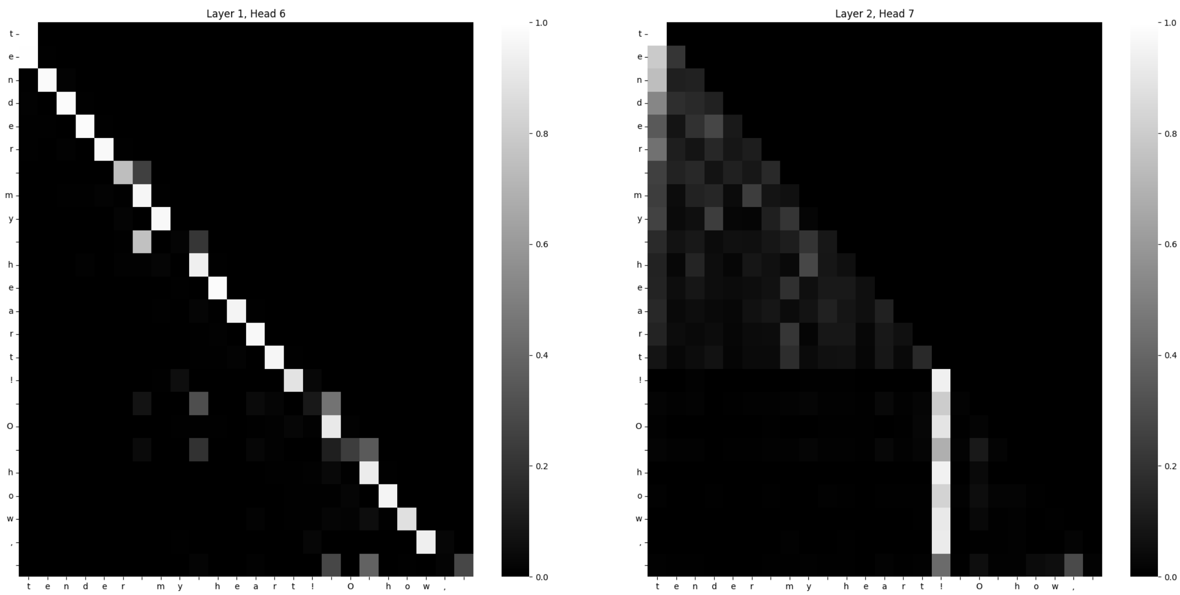



Fig. 2: Training history plot showing the loss value for each iteration.

One observation is that although our loss is quite low and the model generates mostly correct Hebrew words, the sentences themselves do not make sense. It is probably because the model generates letters and not words so we can use fewer parameters. In a real-case scenario, we would generate the words themselves and add more parameters and layers, and use way more train data.

## 5 Interpretability

In this part, we aim to interpret the attention weights on the Shakespeare model. We used the prefix "Hello there, " to generate a sentence of 24 tokens (see Fig. 3).




(a) Attention matrix for layer 1 and head 6

(b) Attention matrix for layer 2 and head 2

Fig. 3

In the left attention matrix, we can observe that almost every letter attends mostly to its next token without giving attention to other tokens. Another interesting observation is that this behavior is more frequent for tokens within words and not spaces. However, in the right attention matrix, the letters after the symbol "!", attend mostly to it. Therefore, we can conclude that the symbol "!" greatly impacts the next generated tokens in this specific attention matrix.

# 6 A brief description of your experience with the project

In this project, we implemented a transformer-decoder model both using Shakespeare data and Hebrew data. We had the experience of training the model and playing with different hyperparameters. Also, we interpreted the results by looking at the attention weights and understood the structure of the transformer model and the way it works using batches and the use of the mask matrix.