Multi-Head Attention For Binary Classification

Marco Schwarz

1 Introduction

The task of this assignment was to built a binary classifier, which can differentiate between the digits one and eight in a provided dataset. It is a subset of the MNIST dataset, which contains handwritten digits with their corresponding labels. The digits are represented as an image with 28x28 pixels. In the dataset provided, these images are already flattened.

Since neither the type of classifier nor a target accuracy were specified, I chose to implement a deep neural network, utilising multi-head attention. This serves mainly as a practical introduction to attention mechanisms.

2 Attention

Attention has become increasingly present in modern deep learning. A very influential paper on this topic is *Attention Is All You Need* [2]. It shows an effective implementation of attention in form of a Transformer, which is used in language models. Part of the proposed Transformer is multi-head attention.
Common image classifiers are often built around convolution to extract features from images. The goal of this model is to replace them with a multi-head attention layer and investigate its behaviour.

In general, attention uses so called queries Q, keys K and values V to highlight certain areas of the input, which are important for the model. This can be specific words in a sentence or areas of an image. Computationally, V is scaled by a specific function of Q and K. In this case, for calculation the scaled dot-product attention is used.

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

d_k is the scaling factor, which represents the dimension of K. Softmax is a commonly used function in deep learning.

$$\mathit{softmax(x)} = \frac{e^x}{\sum e^x}$$

It normalizes its input and ensures that the sum of the output equals one. Furthermore, small values get reduced and big values get increased.

3 Architecture

The model is implemented in Python, using the Keras [1] Sequential API. Keras already provides multi-head attention as a layer building block.

The input gets reshaped and feeds into the attention layer. Its output gets flattened and directly passed to the last layer, which uses a sigmoid activation function, converting the output into values between zero and one, corresponding to eight and one respectively. This architecture is extremely shallow, putting the focus on the attention layer alone.

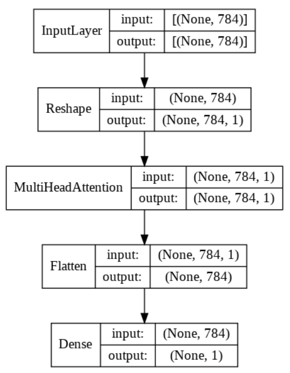

Figure 1: General layers of the implemented model.

Training From the dataset unwanted digits got removed and it got split into training and test data. The test data was roughly 20% of the dataset. As loss function binary cross-entropy was used and the optimizer was Adam [3] .

4 Hyper-Parameter Tuning

In order to find the optimal hyper-parameters for the model, I performed a Bayesian hyper-parameter optimization. The extend of this search was limited due to lack of computational power. The parameters consisted of the amount of attention heads, the presence of a skip connection, the drop-out rate and the learning rate of the optimizer. For each of those a small set of values was selected.

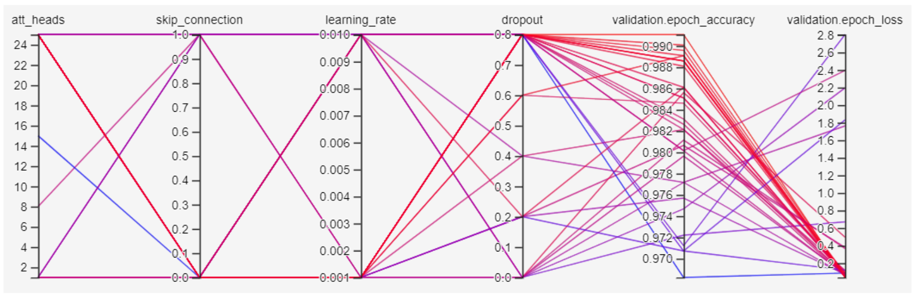

Figure 2: Parameter space of searched configurations with corresponding validation loss and accuracy. The color coding is based on accuracy. All configurations show over 95% validation accuracy.

Result The hyper-parameter search shows that an overwhelming amount of configurations performed very similar. More attention heads do provide a small improvement but the gains are marginal. This indicates that the classification task at hand does not require a complex model. Therefore, the finally selected configuration was focused on simplicity rather than accuracy.

5 Final Implementation

Since the hyper-parameter search does not show big differences, the selected parameters are the following:

Parameters	Values
Attention Heads	1
Drop-out Rate	0.0
Learning Rate	10^{-4}
Skip Connection	No

5.1 Performance

The accuracy was determined by the performance of the model on the test data. This data was initially split from the dataset and was only used for testing purposes. For the time-line during training, validation data is used. Similar to the test data, it is split off the training data and serves as accuracy measure after every training batch. Furthermore, the loss of the validation data was used to detect over-fitting, in which case early stopping was triggered. The accuracy on the test dataset is 97.9%.

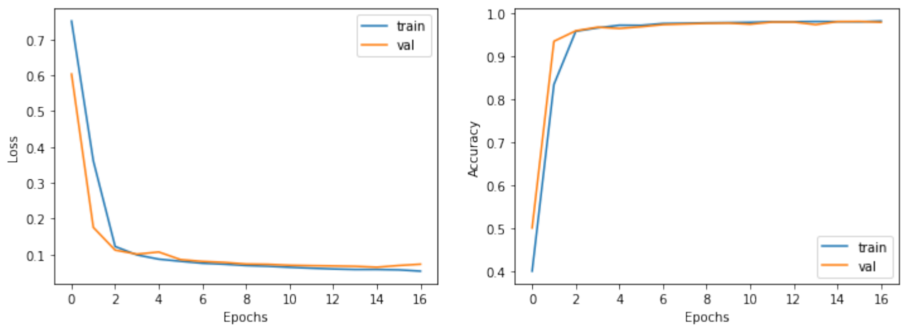

Figure 3: Accuracy and Loss during training

5.2 Layer Analysis

To get a better understanding of the inner workings of the attention layer, I looked at its learned weights and interaction with input data. More specifically, I used the internal methods from the Keras MultiHeadAttention class to reconstruct the computation steps of the layer.

The query, key and value vectors are calculated by multiplying the input with the query, key and value weights.

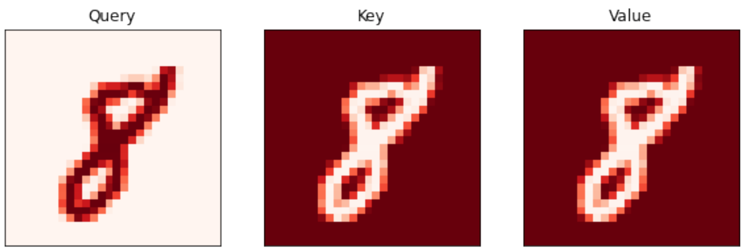

Figure 4: Computed query, key and value inside the attention layer.

From there, the attention scores get calculated. These represent the information each pixel receives from all the others and itself.

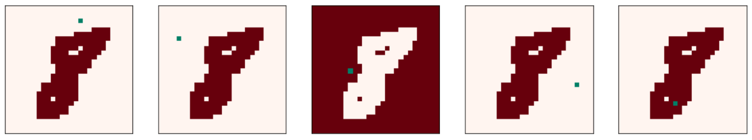

Figure 5: Attention scores within the layer show for a specific pixel where its information is aggregated from. The green pixel displays which specific one is investigated in the image.

By applying the attention scores to the value vector, the final attention output is calculated. Those steps are independently calculated in each head.

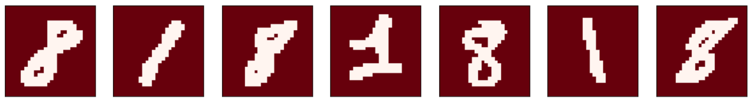

Figure 6: The final output of the attention layer as response to different inputs.

Conclusion The model seems to be focusing on the whole digit instead of subsections. This alone seems to be enough to reach a very high accuracy. I suspect that the classification task is too easy to require the attention layer to learn more complex patterns. This is also supported by the result of the hyper-parameter tuning, which shows that a complex model is only marginally better at this task.

6 Supplementary Model

Since the initial dataset only requires a very simple model, the behaviour of the attention layer could not be satisfyingly investigated. This second attempt uses image augmentation to increase the difficulty of the classification.

Image Augmentation In the first step the image data got artificially transformed in various ways. These included up to 180° rotation, directional translation, as well as stretching. This got achieved by using the Keras ImageDataGenerator class. It uses the existing images and adds transformed variants of them to the dataset.

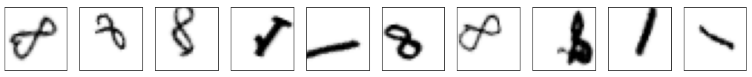

Figure 7: The images of the initial dataset got transformed in various ways.

Architecture The second model uses a significantly more complex architecture. More attention heads in particular, should increase the ability of the model to learn complex pattern in the image.

Parameters	Values
Attention Heads	24
Head Size	6
Drop-out Rate	0.0
Learning Rate	10^{-4}
Skip Connection	No

Training Due to the size of this model, the training took significantly longer. Attention is known to be computationally expensive, since it focuses on the whole image at the same time. Despite the increased classification difficulty, the model reaches 93% validation accuracy. Therefore, the learned weights should be appropriate to analyse.

Layer Analysis Again, the internal representation of a specific input after every mayor computation step is investigated.

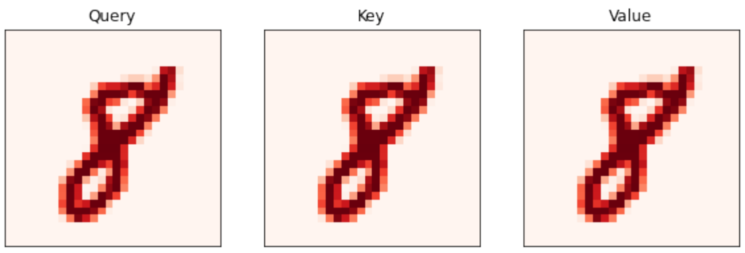

Figure 8: Query, key and value representation in the first attention head.

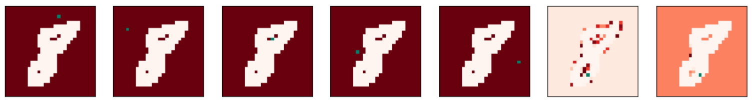

Figure 9: The attention scores in the first head do show some differences from the simple model. Some pixels show a very unique relationship to others.

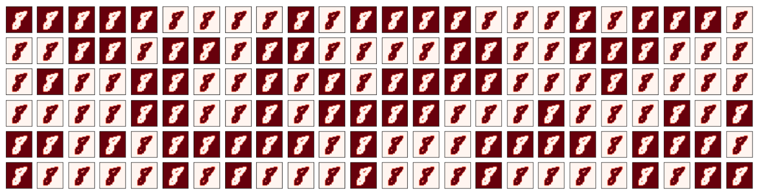

Figure 10: This figure shows the digit representation in all the layers (rows) for each attention head (columns). In this model as well, no interesting local patterns occur. Across all layers, only two distinct representations have been learned.

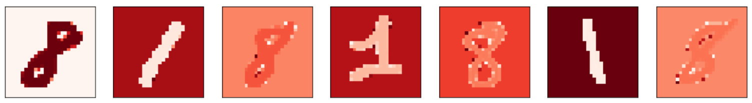

Figure 11: In this case, the output does produce more complex patterns, even though they do not seem very informative.

7 Conclusion

After implementing two different models, as well is manually trying a wide variety of hyper-parameters, the models do not seem to learn very local patterns. Instead, they mostly highlight the digit itself, which seems very uninformative. Regardless, the models are still achieving a high accuracy, indicating that these simple patterns are appropriate for the task at hand. For future experiments, the addition of background noise to the digits might give raise to more complexity.

8 Workflow

The Jupyter notebook containing the full workflow can be found on Google Colab.
`https://colab.research.google.com/drive/1srX7FIf-eqUywGB8MCEzCwIbrGIPZA_f?usp=sharing`

All relevant files, including the trained model, can be found on GitHub.
`https://github.com/xRetry/Multi-Head-Attention-For-Binary-Classification`

References

[1] Keras: Deep Learning Library,
`www.keras.io`

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, pages 5998– 6008, 2017.

[3] Diederik P. Kingma, Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. As *conference paper at the 3rd International Conference for Learning Representations, San Diego*, 2015.

9 Appendix

Results of Hyper-Parameter Search

att_heads	skip_connection	learning_rate	dropout	epoch_accuracy	epoch_loss
25	0	0.001	0.8	0.991	0.0389
25	0	0.001	0.8	0.99	0.058
25	0	0.001	0.8	0.99	0.0376
25	0	0.01	0.8	0.989	0.0419
25	0	0.001	0.6	0.989	0.0504
25	0	0.001	0.8	0.989	0.0398
25	0	0.001	0.8	0.989	0.0806
25	0	0.001	0.8	0.989	0.0473
25	0	0.001	0.8	0.988	0.0389
1	0	0.01	0.8	0.986	0.0459
25	0	0.01	0.2	0.986	0.0499
25	0	0.01	0	0.986	0.496
25	0	0.001	0.8	0.985	0.0826
25	0	0.001	0.8	0.985	0.102
25	0	0.001	0.6	0.985	0.0918
1	0	0.01	0.8	0.983	0.0617
25	0	0.001	0.8	0.983	0.0612
25	0	0.001	0.8	0.982	0.117
1	0	0.001	0.4	0.982	0.0481
25	1	0.001	0	0.981	0.366
8	1	0.001	0.2	0.981	0.378
25	0	0.001	0.8	0.98	0.069
1	1	0.01	0.8	0.98	2.4
1	0	0.001	0	0.98	0.0761
25	1	0.01	0.4	0.977	1.76
25	0	0.01	0	0.977	0.0658
1	0	0.001	0.2	0.976	0.0757
25	1	0.01	0	0.975	2.2
1	1	0.001	0	0.972	0.671
1	1	0.01	0.8	0.971	2.8
25	0	0.001	0.2	0.971	0.121
25	1	0.01	0.8	0.971	1.83
15	0	0.01	0.8	0.968	0.0872