Motivating Mamba Model

Project within the scope of the course "Implementing ANN's with TensorFlow" at the University of Osnabrueck

Alina Griesel
agriesel@uni-osnabrueck.de

Stella Hoyos Trueba
shoyostrueba@uni-osnabrueck.de

Madleen Uecker
mauecker@uni-osnabrueck.de

I. Introduction

Natural Language Processing and Large Language Models (LLMs) can nowadays be encountered everywhere in the form of chatbots and virtual assistants, text summarization and translation of languages. The majority of these models is based on the Transformer architecture and its attention mechanism. LLMs have achieved great results with increasing computational resources that are needed to train these models. Some famous LLMs include the GPT family from OpenAI used by Microsoft and Duolingo, Gemini and BERT from Google and Claude from Anthropic. To analyze textual input, LLMs are restricted by the maximal sequence length they are able to process, which is often referred to as the context window. Larger context windows allow accessing more information. This should enable models to recall information better and draw connections to earlier information. Currently, the largest context windows are provided by Claude 3 with a context window of 200k tokens [1] and by GPT-4 Turbo with 128k tokens [2]. One issue with attention-based models is that the sequence length cannot be extended arbitrarily, since the computational complexity scales quadratically with the input sequence length.

In the paper "Mamba: Linear-Time Sequence Modeling with Selective State Spaces" Tri Dao and Albert Gu [3] present a new model architecture called Mamba. This model scales linearly in computational complexity with sequence length while allegedly performing as good as other prominent sequence-to-sequence models. It thereby specifically promotes large context windows that could contribute to enhancing LLMs as well as other areas using deep learning such as genomics.

This project report is structured as follows: At first an extensive overview of relevant concepts, papers and mechanisms is given. Afterwards, the implementation of a simplified Mamba model is explained, followed by the experiments conducted with the implementation. Last but not least, the Mamba model and implementation are discussed with regard to future prospects.

II. Related Work

In order to understand why Mamba is an interesting architecture and worth exploring we need to take a closer look at the architecture and its inherent mechanisms. To do this we examine the original paper and the state space mechanisms it builds on and improves. Further, to appreciate the Mamba model as an alternative to the prominent Transformer model we also need to understand the relevant parts of the Transformer's architecture, like the attention mechanism.

A. Mamba Model

Gu and Dao [3] put forward a selective state space model (SSM) called Mamba opposing the well-known Transformer model introduced by Vaswani et al. [4]. The Mamba model offers an alternative to the Transformer by picking up on its drawbacks including modeling long-term dependencies and quadratic scaling of computational complexity with input sequence length [3] [5]. Furthermore, Mamba expands earlier state space models such as the structured state space model (S4) [6] by adding a selection mechanism [3]. A Mamba block consists of linear projections, sequence transformations such as one-dimensional convolutions and the SSM, as well as a residual connection (figure 1).

To address the issue of computational efficiency and relevant context filtering Gu and Dao [3] make use of a hardware-aware algorithm and selective filtering.

Gu and Dao [3] show within several different experiments counting language model pretraining, DNA sequence pretraining and audio waveform pretraining, that the Mamba model can achieve similar and sometimes better results than Transformer models.

The Mamba model has amongst other things already been used to build a bidirectional mamba block as vision backbone to account for global visual context [7] and to build a segmentation model for 3D medical images [8].

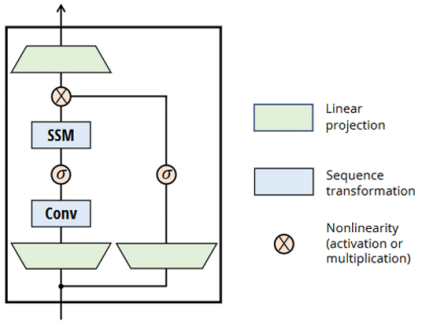

Fig. 1. Architecture of Mamba Block [3]

B. Encoder-Decoder Recurrent Models

The general idea of compressing an input and then reconstructing the input as the output of the model was introduced by Kramer [9] [10]. According to this idea, the term Autoencoder was coined by Hinton and Salakhutdinov [11] to describe an architecture that is composed of two distinct neural network sub-models. The first model, called the encoder, compresses the input while the second model, called the decoder, reconstructs the original input as output of the model.

Sutskever et al. [12] as well as Cho et al. [13] designed the idea of autoencoders with recurrent structures in the two sub-models for modeling sequence-to-sequence tasks [14].

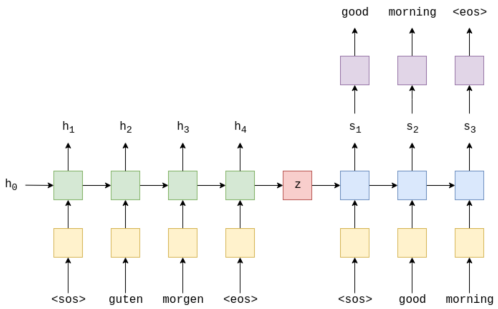

Fig. 2. Architecture of Mamba Block [15]

The encoder sequentially receives an input sequence and learns to encode this information in a single-dimensional hidden vector h [16]. The hidden vector h_j of time step j is built by some function g of the previous time step's hidden state h_{j-1} and the current time steps input x_j:

$$h_j = g(h_{j-1}, x_j). \tag{1}$$

The decoder learns to decode the input information that is compressed in the last encoder's time steps hidden state to generate an output [16]. The decoder receives the encoder's last time steps hidden state h_{T_x}, denoted as z in figure 2, as its first hidden state s_1. Analogously to the calculation of an encoder hidden state, the decoder hidden state at time step i is built as follows:

$$s_i = f(s_{i-1}, x_i). \tag{2}$$

Recurrent Neural Networks (RNNs) are fast at inference because the computational complexity scales only linearly with the input sequence length [17]. This is due to each hidden state only needing the hidden state of the previous time step and the input of the current time step for the calculation of this time step's hidden state. Unfortunately, this means that the training of the model is not parallelizable but has to be performed sequentially [17], resulting in slower training. A bigger problem concerning the performance of encoder-decoder RNNs is that they tend to forget inputs with increasing path length [18]. This means that the more time steps are between a decoder time step i and an encoder time step j, the weaker the information from time step j is encoded in the hidden state when arriving at time step i. Bahdanau et al. [18] showed that with increasing sequence length the BLEU score, which indicates how well neural machine translation models perform, decreases [18]. So, with increasing sequence length the encoder-decoder RNNs decrease in their performance.

C. Attention Mechanism

To tackle the problem of forgetting tokens with sequence length that is faced by RNNs, Bahdanau et al. [18] came up with the idea to indirectly connect every time step in the encoder to every time step in the decoder.

To recall: in encoder-decoder models with recurrent structure every decoder hidden state s_i is computed as described in equation 2. Bahdanau et al. [18] propose to additionally include a context vector c_i composing the decoder hidden state s_i according to this description:

$$s_i = f(s_{i-1}, x_i, c_i). \tag{3}$$

The context vector c_i of the decoder time step i is a by scalar α_{ij} weighted sum over each encoder time step's hidden state vector h_j (4).

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{4}$$

The higher α_{ij} the more important the encoder hidden state h_j is for the decoder time step i. If α_{ij} has the highest value among all time steps j that means indirectly that the input to the encoder at time step j is more important for predicting the output at decoder time step i than any other encoder hidden state. The α_{ij} are also called attention weights and are derived from learnable parameters. The attention weights are the output of a softmax function (6) where its input is the output of a common Dense layer, often referred to as attention layer a whose weights trainable parameters (not to confuse with the attention weights α) (7). The input for this layer is the decoder hidden state h_{i-1} from the previous time step and the encoder hidden state h_j from the time step j.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \tag{5}$$

$$e_{ij} = a(s_{i-1}, h_j) \tag{6}$$

The learnable parameters in the attention layer are the same for all time steps (weight sharing across the time dimension). They are learned through backpropagation simultaneously with the training of the whole model. Yet, the attention weight α_{ij} differs for each encoder-decoder time step pair ij since the attention layer receives different inputs depending on the time step. Accordingly, for each time step pair ij the attention weight has to be computed. This results in quadratic computational complexity for the attention mechanism as depicted in figure 3. For each decoder time step i there is an attention score calculated with each encoder time step j. So the brighter the intersecting area the bigger the alignment between the corresponding input and output time step. For example,

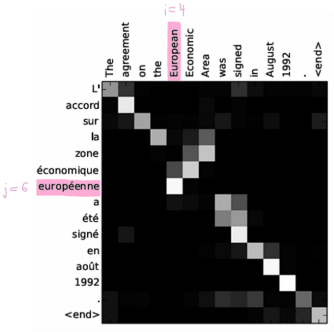

Fig. 3. Attention weights for input-output pairs indicating their alignment [18]

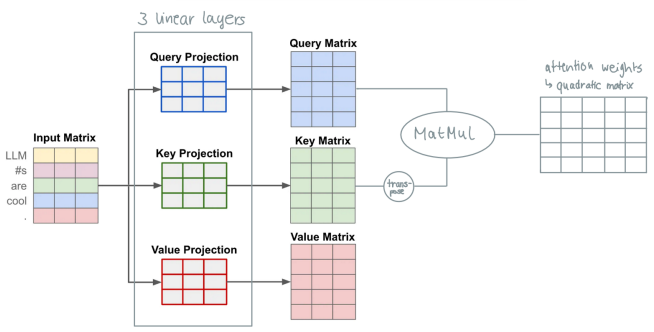

Fig. 4. Scaled Dot-Product Attention [19] with extensions by us

encoder time step $j = 6$ is very important for predicting decoder time step $i = 4$.

This attention mechanism is the reason why the computational complexity of attention-based models scales quadratically with their sequence length. Since for every time step i in the decoder the attention weights α_{ij} to every time step j in the encoder have to be computed [17]. So for, each decoder time step j there is a vector specifying the attention weights to each encoder time step i. The concatenation of these vectors yields the attention matrix (figure 3).

D. Transformer Model

Vaswani et al. [4] utilize the idea of Bahdanau Attention that every token of the input sequence is connected to every token in the output sequence, but realize the attention mechanism as so-called Scaled Dot-Product Attention. Instead of just one attention layer, they instantiate three linear layers, which are commonly referred to as query (Q), key (K), and value (V) (figure 4) [4]. Each layer receives an input (depending on the model structure it often is the same vector) and is thought to model different functions. The dot-product of the query with all keys results in a quadratic matrix of size sequence length as depicted in figure 4. These are the attention weights [19]. Therefore, again Scaled Dot-Product Attention results in quadratic computational complexity with the input sequence length of the model. Thus, the inference is slow [17]. The attention weight for each pair of sequence tokens can be computed simultaneously independent of the position of the token in the input and output sequence. So, in contrast to encoder-decoder recurrent structures, the attention mechanism in transformers is highly parallelizable and therefore enables faster training [17].

The Scaled Dot-Product Attention builds a core building block of the famous transformer architecture: the Multi-Head Attention block (figure 5). Transformers need positional encodings to be able to consider sequential information. Otherwise, Transformers would only operate on sets of the input sequence. Together with linear layers, the input and output embeddings, layer normalization and residual connections, the positional encodings, and the Multi-Head attention layers compose the transformer architecture (figure 5).

We will not go into more detail with the other building blocks of the transformer because this does not further support understanding the Mamba model. The important part for the motivation of the Mamba model is that exactly like Bahdanau Attention, the computational complexity of Scaled Dot-Product Attention scales quadratically with the sequence length [19], resulting in slow inference [17].

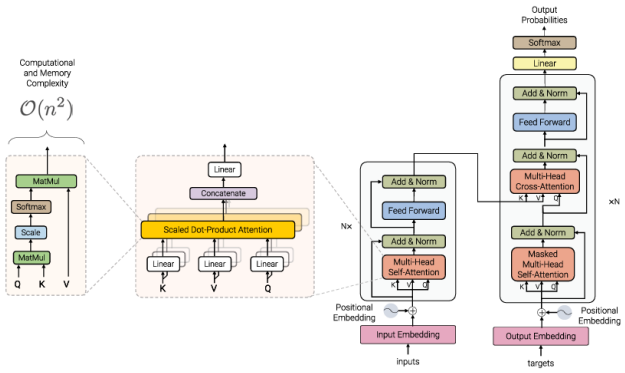

Fig. 5. Scaled Dot-Product Attention and Architecture of Transformers [20]

Other versions of the transformer architecture propose adjustments to reduce the computational complexity at inference time [21] [22] [23] [24] [20]. However, their performance does not compare to the performance of transformers with Scaled Dot-Product Attention [3].

E. State Space Models

SSMs are used to represent a dynamic system as a first-order differential equation with a continuous input (x), output (y), and latent state representation (h) [17], [25]. For linear systems, the state (1) and output (2) equations are defined as follows:

$$h'(t) = \mathbf{A}h(t) + \mathbf{B}x(t) \tag{7}$$

$$y(t) = \mathbf{C}h(t) + \mathbf{D}x(t) \tag{8}$$

The state equation (7) describes the way the state representation $h(t)$ changes with the input $x(t)$. The transformation of the state $h(t)$ into the output $y(t)$ and the effect of input $x(t)$

on the output $y(t)$ are described by the output equation (8) [17].

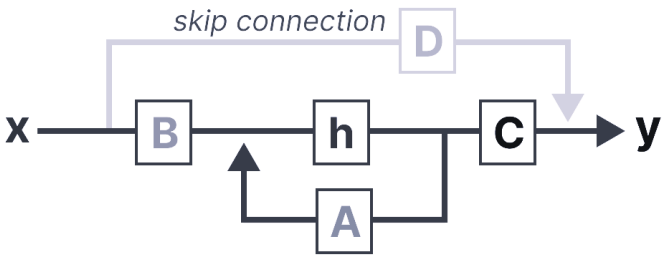

Fig. 6. Continuous, time-invariant SSM [17]

The step-by-step process is as follows: First, the input $x(t)$ is multiplied with matrix B, which represents the impact of the input on the state [17]. Additionally, matrix A, which learns details on the connection between the states, is multiplied with the state representation $h(t)$. As depicted in figure 6 matrix A is used before the state representation $h(t)$ is created and updated afterwards. Next, matrix C is multiplied with the state representation $h(t)$ to translate the state into an output. Matrix D is thought of as a skip-connection, as it provides a direct connection from the input to the output, therefore D is often omitted from the equation [17].

Considering that the input typically consists of discrete values, it is necessary to discretize the SSM [17]. This is achieved through the technique zero-order-hold [17]. In this method, each time a signal is received the value is held until a new signal is received. This ensures that the SSM receives a continuous signal. The duration of the hold is set by a learnable parameter Δ, called the step size [17].

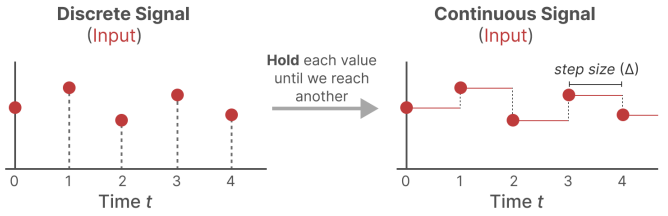

Fig. 7. Transformation of Discrete Input into Continuous Input Signal [17]

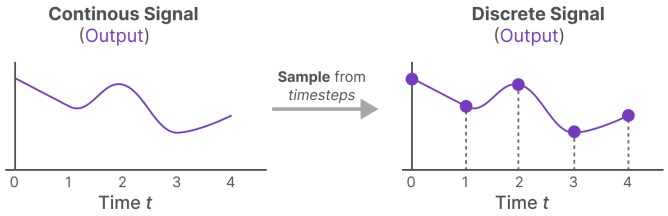

Fig. 8. Transformation of Continuous Output into Discrete Output Signal [17]

To obtain a discrete output, the SSM's continuous output is sampled based on the step size Δ [17]. Mathematically, the zero-order hold is defined through [17]:

$$\bar{A} = exp(\Delta A) \tag{9}$$

$$\bar{B} = (\Delta A)^{-1}(exp(\Delta A)) - I \cdot \Delta B \tag{10}$$

Matrices A and B are now discretized parameters of the SSM, still, the continuous valued matrix A is used during training [17]. With these two equations, the SSM is now depicted as a sequence-to-sequence model rather than a function-to-function one.

F. Linear State-Space Layer (LSSL)

The state equation (7) has become a recurrence, making it possible to represent the SSM as a RNN [17]. This way the SSM has all the advantages and disadvatages of a RNN, mainly efficient inference and inability to be parallelized. Moreover, discretized SSMs can be portrayed using a Convolutional Neural Network. The kernel used in the convolutional representation of an SSM is defined as seen in figure 9 and the output is computed by multiplying the input x with the kernel $\bar{K}$ (11) [17].

$$y = x \cdot \bar{K} \tag{11}$$

Fig. 9. Kernel of Convolutional SSM representation [17]

The convolutional representation allows for parallelized training but slower inference [17]. The Linear State-Space Layer (LSSL) proposed by Gu et al. [26] uses this to its advantage: Because of its higher efficiency due to its parallelizability, the convolutional representation is used during training, and the recurrent representation is used at inference [17]. A significant disadvantage of this model is its higher memory requirement compared to RNNs or CNNs of similar size [6].

G. Structured State Space Model (S4)

The backbone of a Structured State Space Model is a SSM but additionally, it discretizes and initializes the parameter A according to High-order Polynomial Projection Operators (HiPPO) [6]. Finding an appropriate initialization for matrix A is crucial, as in the recurrent representation this matrix stores information about the previous state, which is then used to construct the new state [17]. Therefore, matrix A has a significant impact on the model's ability to remember past tokens, making the difference between remembering only a few tokens and capturing every token so far. By using HiPPO, matrix A stores a state representation that effectively captures recent tokens while reducing the effects of earlier tokens [17]. Mathematically, it can be expressed as follows [6]:

$$A_{nk} = -\begin{cases} (2n+1)^{1/2}(2k+1)^{1/2}, & \text{if } n > k. \\ n+1, & \text{if } n = k. \\ 0, & \text{if } n < k \end{cases} \tag{12}$$

With the addition of HiPPO, the S4 model is characterized by the ability to handle long text sequences as well as efficient memory storage [17].

H. Selective State Space Model (S6)

State Space Models and S4 can be used to model textual sequences, however, they still underperform on important language modeling and generation tasks [17]. For example, selective copying tasks where certain parts of an input sequence should be copied and returned in order, e.g. extracting all nouns in a sentence [17]. Since SSMs are linear time-invariant, as matrices A, B, and C are identical for every token generated by the SSM, SSMs are unable to accomplish content-aware reasoning [17]. Additionally, SSMs are time-invariant and static, which makes it hard for them to replicate patterns in a sequence. Transformers, on the other hand, perform well on these kinds of tasks, as they are able to shift their attention dynamically based on the input [17].

With the Selective State Space Model (S6) Gu et al. [3] aim to overcome these shortcomings by introducing a selective scan algorithm, that filters relevant information, and a hardware-aware algorithm for efficient storage of results via parallel scan, kernel fusion, and recomputation [17].

The selective scan algorithm makes it possible to selectively retain information by selectively compressing information into the state [17]. To do so matrices B and C and the step size Δ are made input-dependent by including the input's batch size and sequence length as dimensions. This way the SSM is content-aware, as every token has different matrices B and C. With the matrices being dynamic it is not possible to use the convolutional representation of a SSM, as it requires a fixed kernel, and parallelization is lost. To combat that the parallel scan algorithm is introduced [17]. In the case of recurrency, the output gets calculated by summing up the prior state multiplied by matrix A and the current input multiplied by matrix B, which can be easily done with a for loop but not in parallel. However, S6 assumes that it is irrelevant in which order the operations are performed because of the associate property. Therefore, the sequence can be calculated in portions and merged iteratively [17].

The second addition to the SSM proposed for the S6 model is the hardware-aware algorithm [17]. A big limitation in current GPUs is the slow transfer speed between SRAM and DRAM, since transferring information often between SRAM and DRAM becomes a bottleneck. To combat this the S6 model uses kernel fusion to prevent writing of intermediate results [17]. Another component of the hardware-aware algorithm is recomputation. This means intermediate states are recomputed during the backwards step but not saved in DRAM [17].

The overview table in figure 11 compares Transformers, RNNs, and Mamba with respect to training and inference performance.

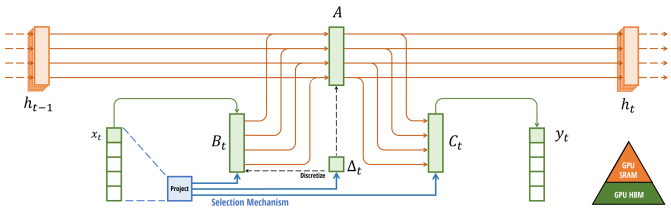

Fig. 10. Selective State Space Model Architecture [3]

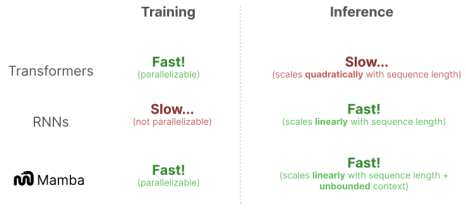

Fig. 11. Overview of the Efficiency of Neural Network Architectures [17]

III. Implementation

The implementation of a simplified version of the Mamba model in TensorFlow is largely based on Pei's PyTorch implementation [27] and Grootendorst's visualizations of the architecture [17]. Furthermore, the hardware-aware algorithm is not implemented in our version. We are aware that the hardware-aware realization of Mamba and the selective scan algorithm are an essential part of why the architecture shows such promising results. Since the hardware-aware algorithm mainly promotes computational efficiency it seems reasonable to implement an easier version of Mamba without it as this goes beyond our available resources. We will therefore provide proof of concept rather than solving a specific task. The following classes are all implemented using the TensorFlow framework in Python and use the model subclassing API for custom models and layers.

A. Mamba Block

The Mamba block as described in [3] and as shown in figure 1. comprises linear projections, sigmoid-weighted linear unit (SiLU) activations, transformations in the form of convolutions and the SSM. Our implementation consists of Dense layers for the linear projections, a Conv1D layer for the convolutional transformation, a custom SSM class for the S6 model and a skip connection described in [17] and is, therefore, called a MambaResBlock. Additionally, we use layer normalization in the beginning and a Dropout layer with 0.2 at the end. For the activation function we use, as proposed in [3], a SiLU activation. The MambaResBlock is initialized with the input sequence length.

B. SSM

The S6 model is implemented as a custom class as well with a call and selective scan function. The SSM is initialized with a states parameter and the projection dimension of the MambaResBlock. Parameter A and D are initialized as a

matrix of shape (projection dimension x states) and a vector filled with ones with length of the projection dimension, respectively. Matrix A is implemented as a HiPPO matrix [28] in the original implementation [3]. Since parameter B, C and Δ are input dependent in the S6 model they are initialized as Dense layers. Parameter B and C have units equaling the projection dimension and the parameter Δ has units according to the number of states. The selective scan method is implemented according to [29] and [27]. Heinsen [29] describes a way to implement parallel computation of otherwise sequential calculations using the logarithmic cumulative sum of the exponent to the euler base. The SSM is initialized in the MambaResBlock.

C. Mamba Model

The final MambaModel class combines the above classes into one coherent model. The input to the MambaModel class is firstly embedded and then fed through the number of MambaResBlocks that were specified in the initialization. The output is again passed through a Dense layer projection with class number depending on the task as well as the activation function.

IV. Experiments

To test our implementation we have conducted experiments in two different domains of natural language processing, namely next-token prediction and sentiment analysis. Next-token prediction means that the model is learning by means of a sentence to predict the next word or specifically token. A model trained on next-token prediction enables generation of arbitrarily long sentences. Sentiment analysis on the other hand uses the sentence input to classify the whole sequence into one of the sentimental classes. For a binary sentiment analysis the classes could simply be positive and negative. Due to (lack of) computational resources and complexity of the Mamba model we have carried out several smaller experiments showing that the implementation works in theory but is not yet solving the tasks efficiently or with great results.

A. Next-Token Prediction

We have executed two experiments on next-token prediction differing in dataset size, batch size, epochs and model complexity. The other configurations were the same for both experiments. To train the Mamba model for next-token prediction we have used a text file containing the Bible. The dataset cleaning includes removing special characters and converting all letters to lowercase. To preprocess the data for training the dataset is tokenized using the self-trained sentencepiece tokenizer with a vocabulary size of 2000. Afterwards, the tokenized data is split into input sequence and target token using the sliding window function with a window size of 128+1. The input sequence has length 128 and the target has length 1 and is one-hot encoded. The data is split into a train and a validation set with a ratio of 0.7 and 0.3 respectively during the preprocessing.

To see what happens as a first experiment we initialize the model with a single MambaResBlock and $\frac{1}{14}$ of the Bible dataset, which is provided in batches of 32. To compile the model we use the Adam optimizer with a learning rate of 0.001 and the categorical cross-entropy loss function. The model is trained for 10 epochs using train and validation data.

For the second experiment we used the configurations mentioned in the appendix of [3]. This time we use $\frac{1}{8}$ of the original dataset and provide the data in batches of 16 items. The model is initialized with three mamba blocks and compiled using Adam with a learning rate of 0.002. The model is trained for only five epochs this time.

Results of the first experiment (figure 12) show very strong overfitting on the training data right after the first epoch and poor results on the validation set with the loss increasing and the accuracy declining. Increasing the dataset size and the model complexity in the second experiment shows less overfitting than the first experiment (figure 13). Gu and Dao [3] describe strong overfitting for a speech generation task that went through 200k training steps. Hence, rather poor results on our side of the text generation approach do not seem too surprising.

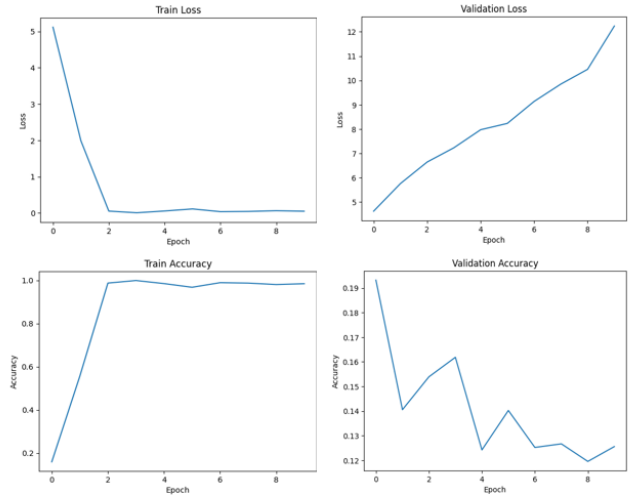

Fig. 12. Accuracy and Loss of experiment 1 for next-token prediction

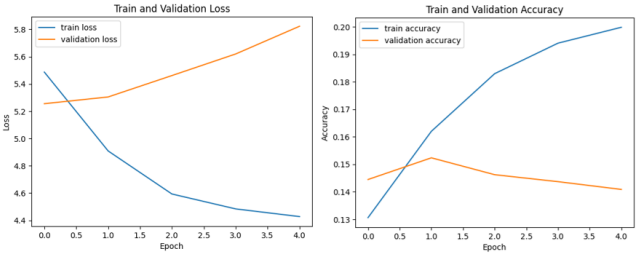

Fig. 13. Accuracy and Loss of experiment 2 for next-token prediction

B. Sentiment Analysis

For the binary sentiment analysis we have also conducted two simple experiments. As a dataset we have chosen the Stanford Sentiment Treebank that provides labeled movie reviews. The dataset is already split into a train, validation and test set containing 8544, 1101 and 2210 samples respectively. In order to preprocess the data the input is tokenized using the pretrained "bert-base-cased" tokenizer and the targets are rounded to 0 and 1 since they were originally float values representing the degree of positivity of the movie review. The input has a sequence length of 267 in accordance with the longest sentence in the dataset.

For the first experiment the model is initialized with a single Mamba block and compiled with the Adam optimizer and a learning rate of 0.001 and the binary cross-entropy as a loss function. The model is trained for 10 epochs. In the second experiment the model has three Mamba blocks and otherwise the same configurations as the first experiment.

The results of the first experiment depicted in figure 14 show a decreasing train accuracy and a stable slightly above chance validation accuracy. The validation accuracy being stable and about 0.5 hints to the model not being complex enough to learn the differentiation of positive and negative reviews. The loss is also practically stable for both training and validation. The second experiment on the other hand already shows within the 10 epochs of training a better tendency of learning how to differentiate the reviews as seen in figure 15. The model still overfits on the training data but the validation accuracy shows a tendency to increase. The validation accuracy staying at roughly 70 percent indicates that the model is not complex enough with 3 blocks to correctly classify the data with more than 70 percent accuracy. As a next experiment one could try adding more Mamba blocks and train for a higher number of epochs.

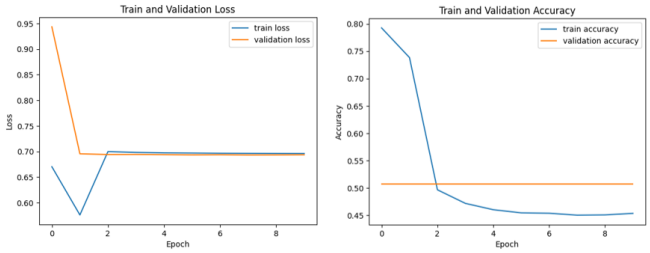

Fig. 14. Accuracy and Loss of experiment 1 for sentiment analysis

V. Discussion

The Mamba architecture offers an alternative to the prevailing Transformer model and its attention mechanism. Mamba shows promising results in the original paper by Gu and Dao [3] and is therefore worth taking a closer look at. In order to understand why Mamba is a reasonable alternative to the Transformer model, we examined Mamba's inherent mechanism and structures. Afterwards, we applied our knowledge to

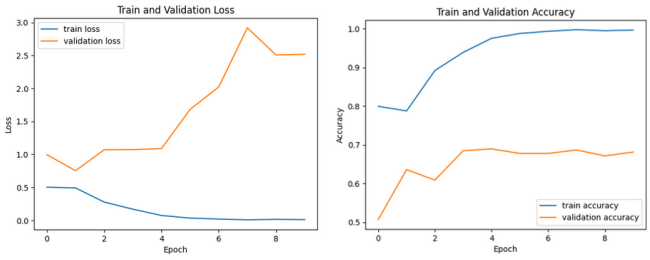

Fig. 15. Accuracy and Loss of experiment 2 for sentiment analysis

implement the Mamba architecture for next-token prediction and sentiment analysis.

Even though we were not able to derive reasonable results testing our simplified Mamba version, examined literature demonstrated that others were able to confirm the promising results of the Mamba architecture. Zhu et al. [7] make use of Mamba's ability to capture long-range dependencies to propose a Vision Mamba Model that integrates a bidirectional SSM for global visual context modeling. They conduct experiments on image classification, semantic segmentation, object detection, and instance segmentation. Their results compared to a Transformer model for vision tasks achieved similar accuracies for image classification with a lot less parameters.

Furthermore, other areas that could profit from the linear scaling of the computational complexity with the sequence length in the Mamba model are genomic studies. Shen et al. [30] describe different genomic studies that have already made use of deep learning methods for predicting sequence specificities of DNA and RNA binding proteins and also predicting effects of noncoding variants. Two disadvantages that came up from the above mentioned genomic studies are the inability to capture long-range dependencies and also to handle longer sequence context [30]. Therefore, the Mamba model could be a great new approach for genomic studies where longer sequences are relevant.

On the other hand, longer sequences might not necessarily be a solution for better model performance per se. The recently published paper "Lost in the Middle: How Language Models Use Long Context" by Liu et al. [31] questions how well information provided by longer sequences is actually used by the model. Their experiments showed that the position of the relevant information in the sequence matters for model performance [31]. While being able to extend the sequence length without increasing the computational complexity too much provides promising possibilities, it is important to consider how well the relevant information can actually be retrieved from the sequence.

Understanding the Mamba architecture by reviewing related work and implementing a Mamba version ourselves, motivated us to engage with several deep learning concepts. These include Transformers, SSMs, and recurrent and convolutional structures.

REFERENCES

[1] Anthropic. (2024) Introducing the next generation of claude. [Online]. Available: https://www.anthropic.com/news/claude-3-family

[2] OpenAI. (2024) Gpt-4 turbo and gpt-4. [Online]. Available: https://platform.openai.com/docs/models/continuous-model-upgrades

[3] A. Gu and T. Dao, "Mamba: Linear-time sequence modeling with selective state spaces," *arXiv preprint arXiv:2312.00752*, 2023.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[5] A. Fan, T. Lavril, E. Grave, A. Joulin, and S. Sukhbaatar, "Addressing some limitations of transformers with feedback memory," *arXiv preprint arXiv:2002.09402*, 2020.

[6] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021.

[7] L. Zhu, B. Liao, Q. Zhang, X. Wang, W. Liu, and X. Wang, "Vision mamba: Efficient visual representation learning with bidirectional state space model," *arXiv preprint arXiv:2401.09417*, 2024.

[8] Z. Xing, T. Ye, Y. Yang, G. Liu, and L. Zhu, "Segmamba: Long-range sequential modeling mamba for 3d medical image segmentation," *arXiv preprint arXiv:2401.13560*, 2024.

[9] M. A. Kramer, "Autoassociative neural networks," *Computers & chemical engineering*, vol. 16, no. 4, pp. 313–328, 1992.

[10] ——, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.

[11] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[12] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[13] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[14] J. Brownlee. (2019) Decoder-only transformers: The workhorse of generative llms. [Online]. Available: https://machinelearningmastery.com/encoder-decoder-recurrent-neural-network-models-neural-machine-translation/

[15] B. Trevett. (2018) pytorch-seq2seq. [Online]. Available: https://github.com/bentrevett/pytorch-seq2seq

[16] L. Weng. (2018) From autoencoder to beta-vae. [Online]. Available: https://lilianweng.github.io/posts/2018-08-12-vae/

[17] M. Grootendorst. (2024) A visual guide to mamba and state space models. [Online]. Available: https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

[18] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," 2016.

[19] C. R. Wolfe. (2024) Decoder-only transformers: The workhorse of generative llms. [Online]. Available: https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse

[20] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, "Efficient transformers: A survey," *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–28, 2022.

[21] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, "Linformer: Self-attention with linear complexity," *arXiv preprint arXiv:2006.04768*, 2020.

[22] I. Beltagy, M. E. Peters, and A. Cohan, "Longformer: The long-document transformer," *arXiv preprint arXiv:2004.05150*, 2020.

[23] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020.

[24] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser *et al.*, "Rethinking attention with performers," *arXiv preprint arXiv:2009.14794*, 2020.

[25] T. Watanabe, "Chapter 1 - background: Dexterity in robotic manipulation by imitating human beings," in *Human Inspired Dexterity in Robotic Manipulation*, T. Watanabe, K. Harada, and M. Tada, Eds. Academic Press, 2018, pp. 1–7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128133859000017

[26] A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré, "Combining recurrent, convolutional, and continuous-time models with linear state space layers," *Advances in neural information processing systems*, vol. 34, pp. 572–585, 2021.

[27] Y. R. Pei. (2024) "mamba tiny". [Online]. Available: https://github.com/PeaBrane/mamba-tiny

[28] A. Gu, I. Johnson, A. Timalsina, A. Rudra, and C. Ré, "How to train your hippo: State space models with generalized orthogonal basis projections," *arXiv preprint arXiv:2206.12037*, 2022.

[29] F. A. Heinsen, "Efficient parallelization of an ubiquitous sequential computation," *arXiv e-prints*, pp. arXiv–2311, 2023.

[30] X. Shen, C. Jiang, Y. Wen, C. Li, and Q. Lu, "A brief review on deep learning applications in genomic studies," *Frontiers in Systems Biology*, vol. 2, p. 877717, 2022.

[31] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, "Lost in the middle: How language models use long contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024.