EEG Mamba: Evaluating DL Architectures for EEG Data Classification

Krystof Latka
UCLA
latka@ucla.edu

Artin Kim
UCLA
artinkim@ucla.edu

Gregor MacDonald
UCLA
gregormacd6@ucla.edu

Konstantin Tzantchev
UCLA
tzantchev@ucla.edu

Abstract

Our goal in this paper is to explore novel architectures and their applications to classifying imagined motor function EEG signals. Among these architectures is Mamba, a selective state space model architecture introduced in late 2023. In this paper we compare Mamba's performance against more established techniques for classifying EEG signals, including CNNs and RNNs. Mamba is a state space model with strong empirical performances on long sequences and time series data. It utilizes elements from control theory, where a state space model is a mathematical representation of a physical system. Additionally Mamba's architecture has proven to be efficient and fast at inference time, which would be ideal for real time applications of EEG classification.

1. Introduction

We explore and compare different Deep Learning paradigms to determine which ones achieve the best performance on the imagined motor function EEG signal classification task.

1.1. State Space Models and Mamba

Mamba is a novel neural network architecture that leverages the mathematical concept of state spaces from control theory to model long sequences [5]. The state space model (SSM) can be defined by the simple equation

$$\begin{aligned} h'(t) &= \mathbf{A}h(t) + \mathbf{B}x(t) \\ y(t) &= \mathbf{C}h(t) \end{aligned} \tag{1}$$

which maps the one dimensional input signal $u(t)$ to a latent space, and then maps that latent representation to a one dimensional output signal $y(t)$. The matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are all learned parameters updated through backpropagation.

This formulation is applied to a discrete time sequence by mapping the input function $u(t)$ to a sequence of inputs $(u_0, \dots, u_k, \dots)$. Assuming a step size of δ then we can create the mapping $u_k = u(\delta k)$. We can then apply a bilinear method to the matrices of the state space model to get a discrete approximation.

$$\begin{aligned} \bar{\mathbf{A}} &= (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}(\mathbf{I} + \frac{\Delta}{2}\mathbf{A}) \\ \bar{\mathbf{B}} &= (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}\Delta\mathbf{B} \\ \bar{\mathbf{C}} &= C \end{aligned} \tag{2}$$

This new approximation can act as a sequence to sequence mapping from the input sequence to the latent space of the model [6]. Now the sequence $(x_0, x_1, \dots)$ represents a sequence of hidden states, effectively creating a recurrent neural network. As we know, this presents a major training bottle neck for the model due to the sequential calculations of the hidden states.

Fortunately this recurrence relation can be reformulated in to a discrete convolution over the input sequence. Within the original paper the notation is

$$y = x * \bar{\mathbf{K}} \tag{3}$$

where

$$\bar{\mathbf{K}} = (\mathbf{C}\bar{\mathbf{B}}, \mathbf{C}\bar{\mathbf{A}}\bar{\mathbf{B}}, \dots, \mathbf{C}\bar{\mathbf{A}}^k\bar{\mathbf{B}}) \tag{4}$$

this definition approximates the RNN definition of the SSM.

Mamba makes two major changes to the SSM. Firstly implementing a selection method to compress context from the input into a smaller representation. This creates a time varying state space model, which can no longer use the convolution above to speed calculations. To counteract this issue the authors introduce a hardware aware algorithm that exploits the memory hierarchy of modern systems to complete computations for the model efficiently. We trained multiple models in the Mamba paradigm to compare effectively with the other selected architectures.

1.2. Fully-Connected Networks

In addition to Mamba, we experimented with a simple fully connected neural network (FCNN). This method proved to be rather ineffective due to their capacity to overfit on the data, in spite of multiple regularization techniques applied to the network. We chose to train multiple models in this paradigm, with differing depths and regularizers, in order to create a baseline to compare with all other models.

1.3. Convolutional Neural Networks (CNNs)

Convolutional Neural Network take inspiration from the convolution operator in Signal Processing. They are widely used in Computer Vision because they can better capture spatial relationships while also involving substantially fewer parameters, compared to Fully-Connected Networks. This approach can also be applied to time-series datasets, using one-dimensional convolution. The properties of the cross-correlation operation that CNNs perform show that they will be able to preserve temporal relations in the data, similar to how spatial information is captured in two-dimensional data. We also briefly explored two-dimensional CNNs by transforming our datasets into spectrograms.

1.4. EEGNet

EEGNet is a compact CNN based architecture introduced in 2018 [2]. The main innovation over a simple CNN on the entire set of signals is that EEGNet processes each signal individually throughout the whole network. The model also utilizes depthwise convolutions in its blocks, before flattening and passing through a linear layer to generate a prediction. Depthwise convolutions have the impact of extracting EEG specific features from the input data for classification. This architecture compared favourably to deeper networks, and thus serves as an established baseline for us to compare to Mamba.

1.5. Recurrent Neural Networks

Another type of neural network that we tested was a recurrent neural network (RNN) architecture. Recurrent networks have properties that lend themselves to time series data, such as the use of a hidden state and the sequential nature of the network. We believed that this may lead to good performance on this dataset, and even if it didn't pass the performance threshold, we still viewed it as a relevant benchmark for comparison with Mamba and other architectures that we tested.

1.6. Transformers

The final architecture that we tested was transformers. These are sequence models built on the attention mechanism and an encoder decoder design [1]. They have been employed in industry to build large scale modeling systems in multiple fields, including text and image generation [3] [4]. Similar to the previous section on RNNs, we believed that their strong empirical performances on sequence based tasks and their ability to capture long range dependencies with attention would translate well to this particular task.

2. Data

The dataset is comprised of 2,115 trials for train and 443 for test. Each trial consists of 22 electrode channels each with 1,000 time bins. The label for each trial is one of four labels (presumably corresponding to directional cursor movement up, down, left, right). This means we have in total 46.5 million float data points to train off of. This leads to a difficult problem of overfitting when it comes to more complex model architecture which we discuss in later sections.

2.1. Data Augmentation

The dataset we work with is very limited, so we use several data augmentation techniques to increase the number of training samples. In particular, we applied the following to our training set:

- **Averaging with noise:** We took the data and averaged it in chunks of size two (equivalent to applying an AveragePool with a kernel of size 2 and a stride of 2). This divides the number of data points per sample by a factor of two.

- **Subsampling with noise:** We use subsampling with noise with a step size of two. This allows us to generate two new noisy subsets of the original (post-trim) training data, thus increasing the number of training samples.

This allowed us to essentially triple the number of samples in our training set.

In addition, the following operations were performed on all sets (training, validation, and testing) before any other operations or augmentations (except for MaxPooling when averaging with noise):

- **Trimming:** Upon manual inspection and visualization of the training data, it was apparent that data beyond t=500ms mostly consisted of noise, and would therefore hurt learning more than anything. We thus decided to train our model on the first 500ms of data only.

- **MaxPooling:** To further reduce the dimensionality of our training samples, we decided to perform one-dimensional MaxPooling to our data with a kernel size of 2 and a stride of 2. Note that MaxPooling was not applied when the data underwent averaging and noise addition, since the averaging already takes care of reducing the dimensionality.

This allowed the network to learn better while decreasing the dimensionality of our input data, allowing for faster training.

2.2. Transformations

In general, we did not apply extensive transformations to our input data beyond Trimming and MaxPooling. For our two-dimensional CNN experiment, we transformed our input into a 22-channel image by applying a Mel Spectrogram to each channel independently.

3. Results

Throughout our testing, we found that Mamba and CNN 1D + LSTM architectures achieved best performance. We did not include a table for the architecture of CNN 1D + LSTM, but this architecture can essentially be described by combining Tables 5 and 7, without the fully-connected block in Table 5. To achieve these results we applied all data augmentations and trained over the whole dataset. These findings demonstrate that mamba not only performs well in this use case, matching other state of the art approaches.

4. Discussion

4.1. Regularization techniques

The limited size of the dataset forced us to apply different techniques in order to increase our testing accuracy while avoiding overfitting. In particular, significant dropout (up to 0.65) was frequently used to regularize our model, and a large L2 regularization factor (up to 0.5) was used to keep our weight magnitudes relatively low. These techniques significantly reduced our models' overfitting and allowed us to keep reducing our validation loss over a larger number of epochs.

4.2. Performance

While exploring the transferability of training we trialled two approaches. Firstly we trained on a single subject, and attempted to generalize from that information to the whole dataset. This approach demonstrated poor generalization, presumably due to the smaller training set size causing overfitting to the characteristics of that person. When attempting the opposite approach, training over the whole set and testing on one person, we found that the model generalized very well.

Our team also explored several different trims to evaluate the effect the amount of data that we had over the time dimension had on training. Using different numbers of points from 250-1,000 we found that performance remained relatively high across the board. Empirically over the full dataset there was overfitting, making it the worst performer. This result could be influenced by the amount of unnecessary information in the final 250 points of the time series, which explains why our performance was higher in the other examples.

We also decided to find the impact of overfitting on our

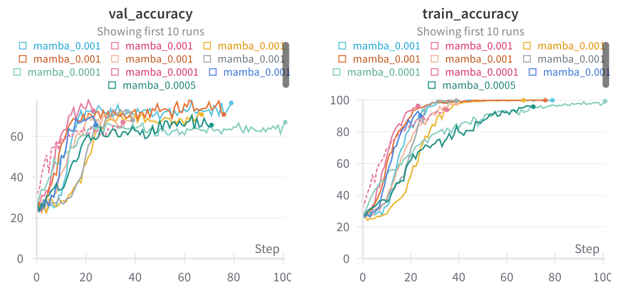

Figure 1. Accuracy of different mamba experiments

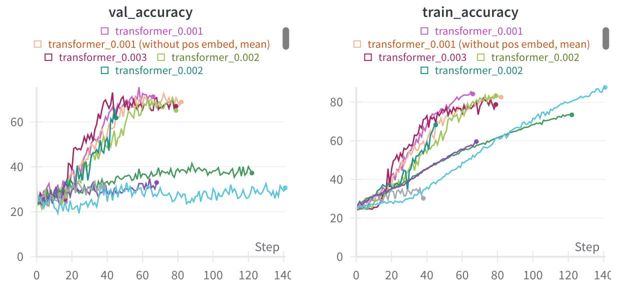

Figure 2. Accuracy of different transformer experiments

models. As we increased the number of epochs, almost every model overfit. Interestingly Mamba (and CNN 1d) appeared to perform well on the test set regardless of the train accuracy approaching 100%, unlike transformers, as can be seen in Figures 1, 2. However, this could be circumstantial, and there might exist certain setups where transformers exhibit the same behavior. With the data we collected, we believe that Mamba merits further exploration of its usefulness in the field.

References

[1] Ashish Vaswani et al. Attention is all you need, 2023. 2

[2] Lawhern et al. Eegnet: a compact convolutional neural network for eeg-based brain–computer interfaces. *Journal of Neural Engineering*, 15(5):056013, July 2018. 2

[3] OpenAI et al. Gpt-4 technical report, 2024. 2

[4] Patrick Esser et al. Scaling rectified flow transformers for high-resolution image synthesis, 2024. 2

[5] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2023. 1

[6] Alexander Rush and Sidd Karamcheti. The annotated s4. In *ICLR Blog Track*, 2022. https://iclr-blog-track.github.io/2022/03/25/annotated-s4/. 1

	Dataset				Accuracy		
Architecture	**Data Augmentations**	**Trim**	**Train + Val**	**Test**	**Train**	**Val**	**Test**
CNN 1D	All	1000	All 9 datasets	All 9 datasets	**100.0%**	60.85%	59.82%
CNN 1D	All	750	All 9 datasets	All 9 datasets	**100.0%**	72.64%	64.33%
CNN 1D	All	500	All 9 datasets	All 9 datasets	99.75%	**77.36%**	70.20%
CNN 1D	All	250	All 9 datasets	All 9 datasets	89.28%	71.70%	68.40%
CNN 2D	All + Spectrogram	500	All 9 datasets	All 9 datasets	92.28%	55.19%	55.53%
CNN 1D + LSTM	All	500	All 9 datasets	All 9 datasets	90.29%	71.22%	**70.43%**
CNN 1D + LSTM	All	500	Data from Person 1	All 9 datasets	96.41%	29.78%	38.60%
CNN 1D + LSTM	All	500	All 9 datasets	Data from Person 4	89.09%	64.00%	68.00%
LSTM	All	500	All 9 datasets	All 9 datasets	95.84%	46.70%	44.70%
GRU	All	500	All 9 datasets	All 9 datasets	98.62%	40.09%	39.28%
Transformer	All	500	All 9 datasets	All 9 datasets	76.48%	65.57%	64.46%
EEG Net	All	500	All 9 datasets	All 9 datasets	78.73%	66.51%	66.14%
Mamba	All	500	All 9 datasets	All 9 datasets	99.97%	76.41%	**70.43%**

Table 1. Best performance of various architectures

Training feature		**Notes**
Number of epochs		Varied, usually between 80-100
Batch size		Varied, [32, 64, 128, 256, 512]
Learning rate		Tuned, usually between 10^{-4} and 10^{-3}
Optimizer		AdamW
Data Augmentations	Averaging with noise	Window of 2, Noise scale of 0.5
	Subsampling with noise	Window of 2, Noise scale of 0.5
	Trimming	First 500 ms of data
	Max Pooling	Kernel size of 2, stride = 2

Table 2. Training notes

Layers	**Output Size**	**Hyperparameters**
Embedding	250×24	Hidden dim = 24
Dropout	250×24	p = 0.5
Mamba Block 1	250×24	Hidden dim = 24, Dim state = 16, Dim conv = 4, Expand = 25
LayerNorm	250×24	
Dropout	250×24	p = 0.5
Mamba Block 2	250×24	Hidden dim = 24, Dim state = 16, Dim conv = 4, Expand = 25
LayerNorm	250×24	
Linear	4	Input size 24, mean reduction applied before this layer

Table 3. Mamba Architecture

Layers		**Output Size**	**Hyperparameters**
Input FC Layer	Linear	24×250	Input size 22
Transformer	Encoder x 2	24×250	4 heads, pointwise 2048
Final FC Layer	Linear	4	Input size 24, mean reduction applied before this layer

Table 4. Transformer architecture

Layers		Output Size	Hyperparameters
Conv Block 1	Conv 1D	64×250	64 filters of size 3, stride = 1, padding = 1
	BatchNorm 1D	64×250	
	ELU	64×250	
	MaxPool 1D	64×125	Kernel size of 2, stride = 2
	Dropout	64×125	p = 0.65
Conv Block 2	Conv 1D	128×125	128 filters of size 5, stride = 1, padding = 1
	BatchNorm 1D	128×125	
	ELU	128×125	
	MaxPool 1D	128×61	Kernel size of 2, stride = 2
	Dropout	128×61	p = 0.65
Conv Block 3	Conv 1D	256×61	256 filters of size 3, stride = 1, padding = 1
	BatchNorm 1D	256×61	
	ELU	256×61	
	MaxPool 1D	256×28	Kernel size of 2, stride = 2
	Dropout	256×28	p = 0.65
Conv Block 4	Conv 1D	512×28	64 filters of size 3, stride = 1, padding = 1
	BatchNorm 1D	512×28	
	ELU	512×28	
	MaxPool 1D	512×11	Kernel size of 2, stride = 2
	Dropout	512×11	p = 0.65
FC Block 1	Linear	1024	
	ELU	1024	
	Dropout	1024	p = 0.65
Final FC Layer	Linear	4	

Table 5. CNN 1D architecture; this CNN was trained for 75 epochs.

Layers		Output Size	Hyperparameters
Block 1	Conv 2D	$8 \times 1 \times 251$	8 filters of kernel size (22, 64), padding = (0, 32)
	BatchNorm 2D	$8 \times 1 \times 251$	
	Depthwise Separable Conv2D	$16 \times 1 \times 252$	
	Batchnorm 2D	$16 \times 1 \times 252$	16 filters of kernel size (1, 62)
	ELU	$16 \times 1 \times 252$	
	AvgPool2D	$16 \times 1 \times 63$	Kernel size of (1, 4)
	Dropout	$16 \times 1 \times 63$	p = 0.5
Block 2	Depthwise Separable Conv2D	$16 \times 1 \times 64$	16 filters of kernel size (1, 16)
	Batchnorm 2D	$16 \times 1 \times 64$	
	ELU	$16 \times 1 \times 64$	
	AvgPool2D	$16 \times 1 \times 8$	Kernel size of (1, 8)
	Dropout	$16 \times 1 \times 8$	p = 0.5
Final FC Layer	Linear	4	Input size 128

Table 6. EEGNet architecture

Layers	Output Size	Hyperparameters
LSTM	250×128	hidden size of 128, 5 layers, dropout p = 0.5
Dropout	128	p = 0.4, mean reduction applied before this layer
Linear	4	Input size 24

Table 7. LSTM Architecture