ECE 239AS Final Report
Classifying EEG Data Using CNNs and RNNs

Lawrence Chen
UCLA
Los Angeles, CA 90095
lawrencechen98@gmail.com

Vignesh Sairaj
UCLA
Los Angeles, CA 90095
vignesh.sairaj@cs.ucla.edu

Aditya Joglekar
UCLA
Los Angeles, CA 90095
adivj123@g.ucla.edu

Sean Derman
UCLA
Los Angeles, CA 90095
seandermanyang@gmail.com

Abstract

In this paper, we explore the effectiveness of three convolutional neural network architectures alongside one recurrent neural network architecture. We discovered that the performance of a CNN architecture, even before exhausting hyperparameter tuning, outshone the RNN architecture. This was due to the fact that RNNs can only operate on the inputs of sequences one step at a time where the CNNs can convolve over the temporal space and build useful temporal hierarchical features on top of it. Recent research from neurological research also suggests that architectures modeling the spatio-temporal information hierarchically do significantly better than simple recurrent models.

1. Introduction

We had experimented with four different architectures: three CNN architectures, and one RNN. They will be referenced as CNN-22, CNN-25, CNN-Spatio-Temporal, and the RNN architectures.

The CNN-Spatio-Temporal architecture was inspired from A. Joglekar et al's paper. This basic idea is to convert the spatio-temporal EEG data, which has one spatial and one temporal dimension, into a data form more suited to the CNN without sacrificing subtle discriminatory information. The above paper uses this idea to process fMRI data into 2d image-like data followed by a shallow CNN (LeNet). This architecture uses a Deep Trans Layer to compute cross-correlations across the regions converting the 22 x time data into 22 x 22 matrices. Once these matrices are obtained, a standard CNN is trained to classify the activity.

The second CNN, CNN-22 architecture we had tried closely resembles the shallow CNN proposed by Tobias et. al. This architecture has two consecutive convolution layers across time and space. The convolution across space is really just a weighted linear combination across electrodes as the filter size is equal to the number of electrodes. This layer flattens the electrode-dimension. After the 2 convolution layers is a mean-pooling layer that reduces the dimension across time. There is, then, a fully-connected layer that outputs the scores for the softmax layer. The paper suggested using ELU activations with batch-normalization and dropout. We were able to get a validation accuracy of around 60% with this approach.

The third CNN, CNN-25, we trained was a deeper CNN architecture with more convolutional layers and fully connected layers. While Tobias et. al's deep CNN had a convolution across the filter-dimension, when we tried to reproduce those results, we did not get very good accuracies. We hypothesized that these subpar results were due to a lack of knowledge concerning the spatial orientation of the electrodes. Specifically, convolving over a small window, as opposed to a full-sized filter, would not give us the results that we would have achieved if we understood the local structure in the data to a greater extent. We decided instead to treat each electrode as a channel and performed 1-dimensional convolutions across time only. We also used dropout and 2 fully connected layers at the end to achieve much higher accuracies.

For the RNN implementation, we used two LSTM layers with larger dropout on the second layer. We discovered that this yielded better performance than conducting high dropout on both layers, while not using dropout at all would make the model overly sensitive to noise. We did, however, utilize dropout on the recurrent state for both

LSTM layers, which proved to be a good regularization technique.

2. Results

With regard to CNN-Spatio-Temporal, we observed that we could take advantage of the temporal patterns but not of the spatial. This is mainly because unlike fMRI data, EEG data is spatially lacking in information but temporally rich. Hence, the spatio-temporal CNN method which averages out over time and focuses on analyzing spatial correlations across regions may not perform very well. Hence, we progressed onto CNN architectures which exploit the valuable information caught by the 1D time convolutions.

Further, the spatio-temporal CNN which computes an average 1-d convolution across time and then applies spatial filters does not get any spatial learning. Ultimately, the validation accuracy sat at around 35% only, despite a training accuracy of about 70%, indicating significant over-fitting.

When we attempted tuning CNN-25, we had used all 25 electrodes and had achieved a validation accuracy of 70%. Since we were unaware when training this model that we were to use only the first 22 electrodes, we did not have enough time to tune the hyperparameters for the 22 channel model. We got a validation accuracy of around 57% for CNN-22.

Finally, the RNN model with 22 electrodes proved the hardest to tune, as the model was extremely sensitive to hyperparameter modifications and the training itself took longer with more model parameters to train. We spent the majority of the project trying variations of our RNN architecture. We explored using two fully connected layers with loss curves shown in Figure 1, but it was not effective, so we changed the model to only one fully connected layer. We also augmented the data by resampling the sequences to remove some noise and reduced dropout rate in the input layer with loss curve (results shown in Figure 2). Our final model achieved a validation accuracy of about 60%, while our test accuracy was 58.4%.

3. Discussion

Ultimately, it seems like RNN's are disadvantaged to CNNs for EEG data. The RNN was more difficult to train and only resulted in comparable performance to CNNs at best, with much more intensive hyperparameter tuning. RNNs lack the ability to look for high-level patterns as CNN's do due to their convolutions.

RNN's take in data as a one-directional format which doesn't take advantage of the certain more "visual" patterns that a CNN may be able to pick up that are present in the EEG data. Hence, if important temporal events (e.g peaks, troughs) happen far away across different channels, that might be completely missed by the RNN. Feeding the RNN a time step at a time makes the RNN generate states as it processes a sequence, thus it becomes sensitive to noise that affects the state too much to recover from. Compare this with CNNs, which convolve across the temporal dimension, and thus can look for general high-level features, and not be easily influenced by slight noises or fluctuations. We hypothesize that this is one major handicap limiting the effectiveness of RNNs for the problem at hand.

We also noticed that reducing the dropout rate on input data but increasing the dropout rate in the second layer improves our model's performance. We hypothesize that this is due to the dropout on the input of the data (to the first LSTM) not simply nulling out particular elements, but rather providing misleading noise that prevents the RNN model from learning patterns in the data. When we dropout a time step value, it creates plausible signal fluctuations in our data that are similar to other features. This is unlike image data, in which zeroing out a single pixel does not compromise a data's overall integrity, features, or general patterns, but simply provides general noise. Thus, we reduce the rate of dropout on input but increase dropout in later layers to still retain the regularization functionality.

We realized in training the first CNN, that the data at 1000 timesteps was very noisy. In order to alleviate this, we subsampled every 4 seconds/steps this, in turn, increased the dataset's number of trials by 4x to 250 timesteps. Additionally, we applied SciPy's resample method to apply Fourier Transform resampling to downsample the data to 100 timesteps. For the RNN, we augmented our data even further, because we noticed the architecture was sensitive to such noise. We subsampled the data at every 10 seconds, to 10x our dataset, now with 100 timesteps. We then applied the same resample Fourier method to create sequences of 40 time steps.

Additionally, the spatio-temporal CNN was not very suitable for EEG data unlike fMRI data, which the original architecture was built for. The training showed signs of significant overfitting with training accuracy never surpassing 70% and test accuracy maxing at 35%. To alleviate this, we attempted using Xavier Initialization, batch normalization, regularization, dropout, Adam SGD and tuned hyperparameters. However, this architecture is fundamentally not suited to EEG data. These findings lead us to CNN-25, as described in [1] as the 1-dimensional convolutions across time make the most of the temporal information and makes up for the lack of (learned) spatial information therein.

4. References

1. R. T. Schirrmeister et al, Convolutional neural networks in EEG analysis, 2018, in arXiv:1703.05051v5
2. A. Joglekar et al, Deep Transformation Method for Discriminant Analysis of Multi-channel Resting State fMRI, 2019, AAAI

Methods Appendix Found On Next Page

5. Methods

5.1 RNN-22 Architecture Details and References

Layer (type)	Output Shape	Param #
lstm_15 (LSTM)	(None, 40, 30)	6360
batch_normalization_15 (Batc	(None, 40, 30)	120
lstm_16 (LSTM)	(None, 16)	3008
batch_normalization_16 (Batc	(None, 16)	64
dropout_8 (Dropout)	(None, 16)	0
dense_8 (Dense)	(None, 4)	68

Total params: 9,620
Trainable params: 9,528
Non-trainable params: 92

Table 1: The model summary for our final RNN-22 model.

Optimizer: ADAM

Data augmentation: Subsampled sequences of size 1000 every 10 timesteps, resulting in sequences of size 100. Then resampled using scipy.resample( ) to get sequences of size 40.

LSTM layer 1: recurrent_dropout = 0.25, L2 kernel_regularizer = 0.01, default Keras LSTM activations

LSTM layer 2: dropout = 0.50, recurrent_dropout = 0.50, L2 kernel_regularizer = 0.03, default Keras LSTM activations

Dropout layer: dropout = 0.60

Dense layer: L2 regularizer = 0.06, softmax activation

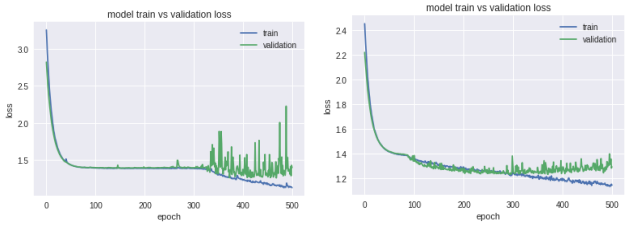


Figure 1: Example training and validation loss curves for the RNN models with two fully-connected layers. As we can see, the model not only overfits really quickly, but also results in very noisy validation error.

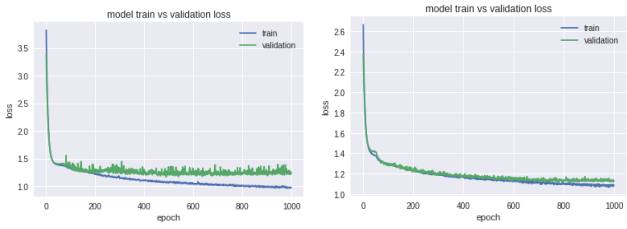


Figure 2: Example training and validation loss curves for the RNN models with two LSTM layers and one fully-connected layer. Dataset sequences resampled to 100 time steps per sequence (left). Dataset sequences resampled to 50 time steps per sequence, with decreased input dropout and increased second layer dropout (right).

5.2 CNN-22 Architecture Details and References

CNN - 22

Input shape = (-1, 22)

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 18, 22, 976]	468
Conv2d-2	[-1, 18, 1, 976]	7,146
BatchNorm2d-3	[-1, 18, 1, 976]	36
ELU-4	[-1, 18, 1, 976]	0
AvgPool2d-5	[-1, 18, 1, 60]	0
Dropout-6	[-1, 18, 1, 60]	0
Linear-7	[-1, 4]	4,324

Total params: 11,974
Trainable params: 11,974
Non-trainable params: 0

Input size (MB): 0.08
Forward/backward pass size (MB): 3.37
Params size (MB): 0.05
Estimated Total Size (MB): 3.50

Table 2: The model summary for our final CNN-22 model.

- Conv layer 1: kernel_size=(13, 1), stride=(1, 1), padding=(6, 0)
- Conv layer 2: kernel_size=(22, 1), stride=(1, 1)
- Mean Pool: kernel_size=(1, 32), stride=(1, 16), padding=0

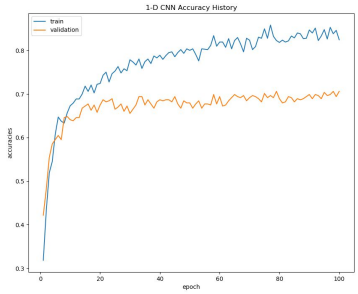


Figure 3: Example training and validation loss curves for the CNN-22 model with 1 fully-connected layer.

5.3 CNN-25 Architecture Details and References

CNN - 25

Input shape = (-1, 25)

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 50, 1000, 1]	16,300
AvgPool2d-2	[-1, 50, 120, 1]	0
ELU-3	[-1, 50, 120, 1]	0
Dropout2d-4	[-1, 50, 120, 1]	0
Conv2d-5	[-1, 35, 120, 1]	5,285
MaxPool2d-6	[-1, 35, 60, 1]	0
ELU-7	[-1, 35, 60, 1]	0
Dropout2d-8	[-1, 35, 60, 1]	0
Linear-9	[-1, 400]	840,400
BatchNorm1d-10	[-1, 400]	800
Dropout-11	[-1, 400]	0
ELU-12	[-1, 400]	0
Linear-13	[-1, 4]	1,604

Total params: 864,389
Trainable params: 864,389
Non-trainable params: 0

Table 3: The model summary for our final CNN-25 model.

Model Parameters:

- Conv layer 1: kernel_size=(13, 1), stride=(1, 1), padding=(6, 0)
- Mean Pool: kernel_size=(48, 1), stride=(8, 1), padding=0)
- Conv layer 2: kernel_size=(3, 1), stride=(1, 1), padding=(1, 0)
- Max Pool: kernel_size=(2, 1), stride=(2, 1), padding=0

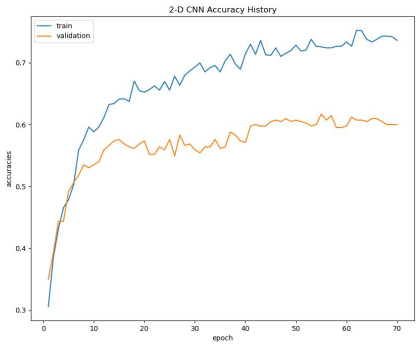
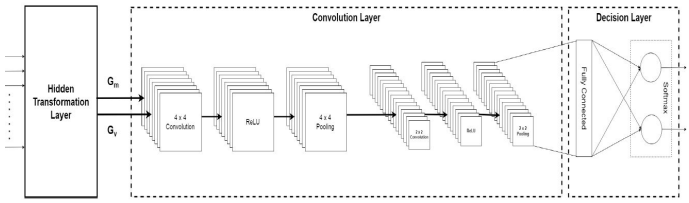


Figure 4: Example training and validation loss curves for the CNN-25 model with 2 fully-connected layers.

5.4 Spatio-Temporal CNN Architecture Details and References



We use the above architecture for the spatio-temporal CNN. In a nutshell, the architecture comprises of the 1. Hidden Transformation layer followed by the 2. standard LeNet-5 shallow CNN.

Details for the spatio-temporal CNN

Optimizer: ADAM

Data augmentation: Subsampled sequences of size 1000 every 10 timesteps, resulting in sequences of size 100. Then resampled using scipy.resample( ) to get sequences of size 40.

The HTL layer:

It transforms the 22 x time data into 22 x 22 image-like matrices using the following equations.

$$S_{(i,j)} = \frac{Y_i[t]Y_j[t-d]'}{trace(Y_i[t]Y_j[t-d]')}$$

The above equation can be simplified as

$$\widehat{S}_{(i,j)} = \frac{W_i[t]X_i[t]X_j[t-d]'W_j[t-d]'}{trace(W_i[t]X_i[t]X_j[t-d]'W_j[t-d]')}$$

$$Y = WX$$

LeNet component:

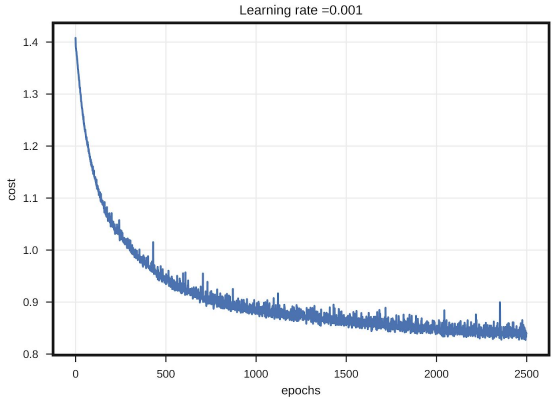
The below figure summarizes the computational graph for the CNN part of the architecture

```
############################ forward propogation of the proposed DTM
# CONV2D: stride of 1, padding 'SAME'

Z1 = tf.nn.conv2d(X, W1, strides=[1, 1, 1, 1], padding='SAME')
# RELU
A1 = tf.nn.leaky_relu(Z1, alpha=0.2)
# A1 = tf.nn.relu(Z1)
# MAXPOOL: window 8x8, sride 8, padding 'SAME'
P1 = tf.nn.max_pool(A1, ksize=[1, 8, 8, 1], strides=[1, 8, 8, 1], padding='SAME')

# CONV2D: filters W2, stride 1, padding 'SAME'
Z2 = tf.nn.conv2d(P1, W2, strides=[1, 1, 1, 1], padding='SAME')  # change P should be conv now
# RELU
A2 = tf.nn.leaky_relu(Z2, alpha=0.2)
# A2 = tf.nn.relu(Z2)
# MAXPOOL: window 4x4, stride 4, padding 'SAME'
P2 = tf.nn.max_pool(A2, ksize=[1, 4, 4, 1], strides=[1, 4, 4, 1], padding='SAME')

# FLATTEN
P2 = tf.contrib.layers.flatten(P2)
# FULLY-CONNECTED without non-linear activation function as required by tf api
# 2 neurons in output layer representing the class probabilities P(Y=1|X) and P(Y=0|X).
Z3 = tf.contrib.layers.fully_connected(P2, 4, activation_fn=None, weights_regularizer=regularizer) # CHANGE 4 classes in fc out
Z3_drop = tf.contrib.layers.dropout(Z3, dropout)  # adding dropout and returning that, CHANGE 8/4
```

Weight initialization: Xavier **initialization**,

Regularization: Batch norm, L2 regularization= 0.01-0.001, dropout= 0.4, learning rate= 0.001 , batch size= 64, # epochs = 2500

Training curve:



5.5 Performance Summary

Architecture	Datasets tested	Performance (test accuracy)
RNN-22	BCI Dataset 2a with 22-EEG	58.4%
CNN-22	BCI Dataset 2a with 22-EEG	58.69%
CNN-25	BCI Dataset 2a with 22-EEG and 3-EOG	69.977%
Spatio-Temporal CNN	BCI Dataset 2a with 22-EEG	35%

Table 5: Performance summary for all algorithms tested.