ECE C147/247 Final Project Paper

Hui Lan
hlan@g.ucla.edu

Galen Wong
wonggalen1999@gmail.com

Kevin Zhang
kevin.zhang.13499@gmail.com

Abstract

This project classifies electroencephalographs(EEG) data provided by the Brain-Computer Interaction Competition [1] using neural networks. The data is split into four motor tasks. Our project compares the performance of different networks including CNNs and RNNs in an effort to correctly classify the data. We found that CNN architecture performs well. A combination of CNN and LSTM architecture also works well but takes longer to train. Subsampling can help increase generalization performance in a CNN+LSTM architecture.

1. Introduction

In our project, we used 2 main machine learning architectures in fitting the EEG data set, RNN and CNN. We used CNN since literature before has shown that CNN can attain a high accuracy [2]. RNN is considered because the data has a temporal axis in nature, therefore RNN might be a good choice of target. The combination of RNN and CNN are also used to utilize the information extraction capability of the convolutional layers, and the temporal data processing power of the recurrent layers. For the RNN layers, LSTM and GRU are used to prevent the problem of vanishing and exploding back-propagation gradient. In terms of data processing, data subsampling is used to increase generalization accuracy. For regularization techniques, we use batch normalization and dropout.

For the hardware setup, Google Colabotory is used since it offers free GPU to speed up training and it allows easy sharing of code and result through the power of Google Drive.

1.1. Pure CNN

The first attempt is to recreate the result from Schirrmeister et. al[2]. Their pure CNN architecture was able to attain 83% accuracy. Their Deep ConvNet architecture is used but the data being passed in is the raw data without any type of processing.

1.2. Pure CNN with subsampling

In the original paper, Schirrmeister et. al used the cropping technique, which is to select a window of continuous data to train on[2]. This effectively increase the number of training examples and can improve generalization accuracy. We took a different approach which is to subsample the data, which means to generate training example by sampling every f time step. The raw data is 1000 time unit long. With a subsampling rate of $f = 2$, each example gives 2 example with 500 time units. This also increases the amount of training data. This should theoretically increase the performance of the CNN architecture since it see more data over time in a convolutional filter.

1.3. Pure LSTM

The motivation for a LSTM network is that the data is temporal in nature, with which a recurrent network should perform well.

1.4. LSTM after convolutional layers

In developemtn, a pure LSTM architecture does not perform well on the data. Therefore, we decided to combine a CNN architecture with a LSTM layer to leverage the power of both constructs. We simply take parts of the CNN layers and appended it with LSTM layers.

We varies the subsampling rate $f = 1, 2, 3 \cdots$ to evaluate the LSTM performance with different sampling rate. Different sampling rate allow the model to see more data over time, which might improve generalization performance.

1.5. Vanilla LSTM vs Bidirectional LSTM

We also compared the difference between a bidirectional and a vanilla LSTM layers. Bidirectional layers allow the model to see the signals from the future, which can improve performance. However, bidirectional LSTM are usually used for image processing and EEG data is in nature time dependent. So, it is cannot be claimed whether one can perform better than the other.

1.6. GRU after convolution layers

GRU is an alternative to LSTM, and empirically performs better than LSTM. We compared the differences between a vanilla and bidirectional GRU and to see the internal differences and the differences compared to the other networks.

1.7. FFT preprocessing before LSTM or CNN

EEG data might have interesting features in the frequency domain, and it has been reported that higher accuracies can be obtained by training deep learning network on the Fourier transform of the data[1]. In this paper, we performed one-dimensional Fourier transform the data from each channel before training the network. We also applied a 0.5 - 100 Hz bandpass filter as in BCI 4-class motor imagery dataset description. The resulting frequency domain data has 794 datapoints per channel.

2. Results

The accuracy for each architectures are shown in table 1. We see that CNN seems to have the best generalization results. The next best architecture is a vanilla LSTM with a sampling rate of 5.

A notable result is the poor performance of a pure LSTM network. This is the motivation of the CNN + Recurrent network.

3. Discussion

3.1. Performance of CNN

The pure CNN has a good performance since its first two conv layers are special. It first does a convolution over time only but not over the 22 channels. It effectively projects the 2D data (channels, times) to 3D (filters, channels, time). Then, a convolution is done over the 3D data but only over the channel, not the time. The reason why they are separated is to increase regularization by splitting the linear transformation into two conv layers [2].

Since this has good performance, we took the first conv layers and attached them to recurrent layer, which is discussed in Section 3.2.

3.2. Pure RNN vs. CNN+RNN

At first, we attempted to train a pure recurrent network with just LSTM layers. However, a pure recurrent network easily overfit on the training data, and it takes too long to train. It takes about 200 seconds per epoch and it convergence requires a large number of epochs (more than 30).

The large amount of training time can be attributed to the fact that the training data has high dimensionality and it takes a lot of iteration for the LSTM to compute each time step. By using convolution, we can reduce the dimensionality and convolution is a faster operation which can be parallelized better compared to LSTM time step iterations. To give an example, on the subsampled data with $f = 5$ (time steps = 200). After a temporal, spatial conv layer, and a max pool layer, we reduce the data from dimension of $(22, 200)$ to $(63, 25)$, which speed up training. More detail can be found in the appendix.

We also believe that convolutional layers can also improve performance. We can view the internal of LSTM as another fully connected network. Conv layers extract spatial information across the 22 channels better than the LSTM through sparse connection and shared filter.

3.3. Effect of Subsampling

We extensively used subsampling as a way to improve our accuracy and made comparisons of no subsampling, sampling every 3, sampling every 4, or sampling every 5 on our bidirectional LSTM implementation. We found a clear indication of a positive correlation between the subsampling and accuracy. We postulated that this is because sampling higher causes a decrease in overfitting. Since there is less data, there is less data for the model to fit onto variance and individual noise of the specific dataset so the overall overfitting is decreased and this is shown in the increase in validation and test accuracy. However, between 5 samples and 4 samples, we see a leveling off of the increase in accuracy; we decided to stop the training here since we conjectured that this is evidence that the data is becoming to sparse to train effectively.

3.4. Bidirectional vs. Vanilla Recurrent Layers

We observed comparable test and validation accuracies for the bidirectional and vanilla LSTM models - the bidirectional model had a 1.2% higher validation accuracy, while the vanilla model had a 1.2% higher test accuracy. This result is expected as the EEG data is time dependent so we do not expect the bidirectionality of the network to lead to a boost in the accuracy. Since the bidirectional model is also trained with the reversed inputs, the training time is generally longer than the vanilla model. In our setup, the bidirectional network had an average epoch time of 170 seconds while the vanilla network had an average epoch time of 50 seconds.

3.5. Number of units in RNN layers

Number of units in the LSTM layers directly impacts the capacity of the model. We tried 2 different number of units, 64 and 25. We suspect that having more units can increase generalization performance.

For bidirectional LSTM, there is a boost in performance by increasing the number of units. In the case of subsampling $f = 3$, increase in units resulted in a 3% increase in

test accuracy. In case of $f = 5$, it only increased by 0.6%. With lower subsampling rate, the LSTM units have to see more time steps, which means more units give a better increase in capacity. However, in higher subsampling rate, the model is better generalized since we are effectively seeing more data overtime, which means an increase in capacity provided not much performance gain.

For unidirectional LSTM, however, an increase in number of units decreases the test accuracy. We suspect that since the data is temporal in nature, since time only flows forward but not backward. The unidirectional LSTM is able to capture is information well in 25 units. However, by using 64 units, the increased in capacity overfitted the training data, which resulted in a lower test accuracy.

3.6. GRU vs LSTM

Both the GRU and LSTM are layers that control the problem of vanishing gradient, however, there are small differences in the implementation that manifest in both empirical training differences and training time differences. One of the main advantages of using a GRU over an LSTM that we saw, outside of the general empirical advantage, is the advantage of the 2 gate GRU vs a 3 gate LSTM. Although the GRU does not have a "cell state" gate, the bidirectional gate has a testing accuracy improvement of about 4%. However, there's also a more drastic improvement in the training time, where average epoch time for LSTM is almost 1.5x faster when comparing the regular vanilla GRU and the vanilla LSTM training with the same number of units. More specifically, the average epoch time is 45 seconds for the GRU but 59 seconds for the LSTM. This is due to the fact that there is one less gate to calculate and backpropagate through.

3.7. Effects of FFT preprocessing

For both the CNN and the bidirectional LSTM model, we observed lower validation and test accuracies when trained on the Fourier transform instead of the raw data. This may suggest that the particular tasks that is performed for this dataset, the motor imagery tasks, did not induce a big change in the signals in the frequency domain. Prior to training the models, we anticipated a higher accuracy from the CNN model since the Fourier transform of the EEG data is no longer time dependent. However, in practice, the FFT & CNN model was only able to achieve 39% test accuracy while the FFT & LSTM model was able to achieve 53%. We speculate that this might be because the CNN model does not utilize the inherent orderliness for the frequency domain data, while in the LSTM model, the values at each frequencies are analysed in a sequential manner.

3.8. Speed of Training

In training, we discovered that a CNN trains faster than a LSTM model. One epoch on CNN takes 1 second per epoch, but RNN takes 60 seconds per epoch on Google Colab. CNN takes more than 40 epochs to converge, CNN+LSTM takes only 25 epochs to converge (See figure 1 and 2). However, due to the immense time difference in one epochs. CNN is still take way less time to converge. Therefore, CNN has a better performance in terms of speed. The reason for why LSTM has slower training has been explained in Section 3.2.

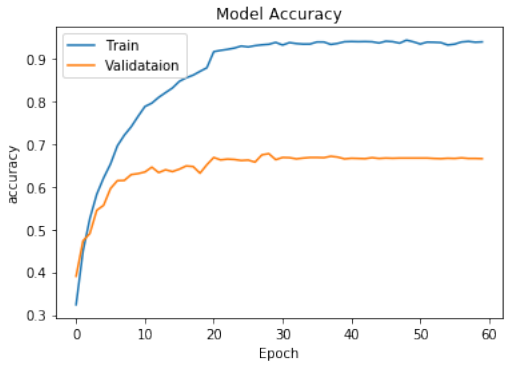

Figure 1. Loss of CNN + LSTM

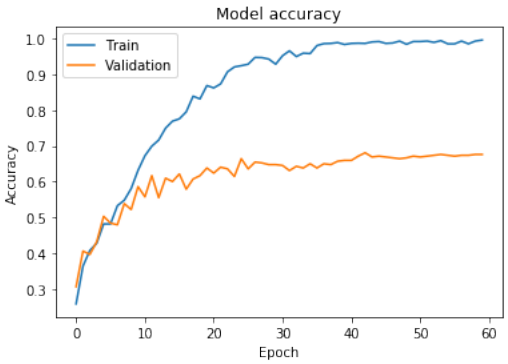

Figure 2. Loss of Pure CNN

4. References

References

[1] A. Craik, Y. He, and J. L. Contreras-Vidal. Deep learning for electroencephalogram (EEG) classification tasks: a review. *Journal of Neural Engineering*, 16(3):031001, apr 2019.

[2] R. T. Schirrmeister, J. T. Springenberg, L. D. J. Fiederer, M. Glasstetter, K. Eggensperger, M. Tangermann, F. Hutter, W. Burgard, and T. Ball. Deep learning with convolutional neural networks for brain mapping and decoding of movement-related information from the human EEG. *CoRR*, abs/1703.05051, 2017.

5. Result Summary

This page is the summary of result. It includes the data modification description.

Model	Validation Accuracy	Test Accuracy
CNN, no subsampling	67.60%	**68.40**%
CNN + Bidirectional LSTM (25 units), subsampling = 5	68.70%	63.90%
CNN + Bidirectional LSTM (64 units), subsampling = 5	67.70%	64.40%
CNN + Bidirectional LSTM (25 units), subsampling = 4	68.20%	61.40%
CNN + Bidirectional LSTM (64 units), subsampling = 4	68.91%	62.35%
CNN + Bidirectional LSTM (25 units), subsampling = 3	58.20%	60.80%
CNN + Bidirectional LSTM (64 units), subsampling = 3	64.40%	64.40%
CNN + Bidirectional LSTM (25 units), no subsampling	41.10%	45.80%
CNN + Unidirectional LSTM (25 units), subsampling = 5	66.20%	65.60%
CNN + Unidirectional LSTM (64 units), subsampling = 5	61.10%	60.40%
Bidirectional GRU (64 units), subsampling = 5	65.10%	64.20%
LSTM 25 units (no CNN)	40%	42.90%
FFT & Bidirectional LSTM	51.10%	52.80%
FFT & CNN	39.50%	39.10%

Table 1. Accuracy of different architectures

6. Model Description

6.1. CNN

Our CNN model uses the following structure. All the conv layers have padding 0 and stride 1 unless otherwise specified. Note that we used 2D and 3D convolution with some dimension being 1, which is effectively convolution in 1D and 2D. However, we did it so that the output dimension works out. This CNN architecture is the same as the one from the paper.

The input size to this model can vary by subsampling rate, but it is always controlled by the FC layers to make sure it outputs the right dimension.

Layer	Parameter Description
CONV2D	$25 \times 1 \times 10$
CONV3D	$25 \times 25 \times 22 \times 1$
BATCHNORM	NA
ELU	NA
MAXPOOL2D	3×1
DROPOUT	$p = 0.25$
CONV2D	$50 \times 25 \times 10$
BATCHNORM	NA
ELU	NA
MAXPOOL2D	3×1 (stride: (3×1))
DROPOUT	$p = 0.25$
CONV2D	$100 \times 50 \times 10$
BATCHNORM	NA
ELU	NA
MAXPOOL2D	3×1(stride: (3×1))
DROPOUT	$p = 0.25$
CONV2D	$200 \times 100 \times 10$
BATCHNORM	NA
ELU	NA
MAXPOOL2D	3×1(stride: (3×1))
DROPOUT	$p = 0.25$
FC LAYER	256
DROPOUT	$p = 0.5$
BATCHNORM	NA
RELU	NA
FC LAYER	4
SOFTMAX	NA

Table 2. The layers used in the CNN model. In the representation of the filter size, the first value always refer to the number of filters.

6.2. Pure RNN

Out pure LSTM model has a batchnorm layer at the beginning to speed up training convergence. Turns out it is too effective that the model overfits to training data too easily.

Layer	Parameter Description
BATCHNORM	NA
Bidirectional RNN	25
SOFTMAX	NA

Table 3. The layers used in each RNN model where we only normalize and use the regular RNN layer w/o convolution.

6.3. CNN+RNN

Our CNN + RNN model. The CNN part is up to the MAXPOOL layer. Again, as mentioned before in section 6.1, the input shapes can be different depending on subsampling rate, the final output has the correct dimension ensured by the FC layer at the end.

We varies the type of the RNN layer with the following combinations:

- Direction: Bidirectional, Unidirectional
- Type: GRU, LSTM
- number of units: 25, 64

Layer	Parameter Description
CONV2D	$25 \times 1 \times 10$
CONV3D	$25 \times 22 \times 1 \times 25$
BATCHNORM	NA
ELU	NA
MAXPOOL2D	3×1 (stride: (3×1))
DROPOUT	$p = 0.25$
Bidirectional RNN	64
FC Layer	4
SOFTMAX	NA

Table 4. The layers used in each CRNN model where we subsample and convolve first to get the inputs in the correct dimensions before passing the filter through an RNN layer

6.4. Model Training

Since our dataset is small and the inputs don't require a huge number of epochs to train, we chose to use ELU instead of regular RELU as our activation. We used softmax for our output data probabilities. For each model, we tried two different optimizers including RMSProp and Adam and ended up choosing Adam because of the versatility. We chose a learning rate of 5e(-4) and a decay of 1e(-5) and included learning rate decrease on a accuracy plateau to help the model train better. We also used certain dataset modifications including shuffling, subsampling, and data transformation to help with processing.