Recurrent Neural Networks and Hybrid of Variational Autoencoder and CNN approaches on classification of electroencephalography data

Hengda Shi
University of California, Los Angeles
hengda.shi@cs.ucla.edu

Gaohong Liu
University of California, Los Angeles
FILL YOUR EMAIL

Mengyao Shi
University of California, Los Angeles
mengyao.shi@cs.ucla.edu

Steven Zhiying Li
University of California, Los Angeles
zhiyingli@g.ucla.edu

Abstract

We implemented multiple neural networks and hybrid of networks to classify EEG data and achieved 65% accuracy on test dataset as best performance. Convolutional neural network, a variational auto-encoder network and a hybrid-CNN-LSTM neural network were implemented. The vanilla convolutional neural network performs fairly well and can achieve around 65.0% accuracy on test dataset. The hybrid-CNN-LSTM neural network can also achieve around 61.4% accuracy on test dataset.

1. Introduction

Electroencephalography (EEG) data are signals that get constantly recorded by electrodes. In this EEG dataset, there are 9 subjects that get recorded by 22 EEG electrodes. The user tries to imagine performing one of the four actions. Therefore, in this classification task, all the data points will belong to one of the four classes. There are a total of 2115 trials completed and each train has corresponding EEG data from 22 electrodes for 1000 times pins, i.e. the signals are likely recorded in a periodic fashion. These signals are recorded near a non-invasive scalp electrodes and therefore have less accurate resolution compared to other approaches. Classification on these types of dataset is challenging because there are many noises in the dataset. Our approach is to apply deep learning technique ,including CNN, RNN, and VAE, to this dataset and find out whether non-linearity introduced by the deep neural networks can nicely capture the signal meaning. We will explain our motivation behind choosing each network and discuss further detail on implementing these networks.

1.1. Motivation of RNN Approach

As we can observe in the dataset, there are temporal information stored in the data. Given the data, since recurrent network is a type of internal dynamical system that it has both feed-forward connections and feedback connections. Therefore we believe it is preferable to apply this model to the dataset changing across different timesteps. In particular, due to the vanishing and exploding gradient problem in training the recurrent neural network, we use more sophisticated model, the long short-term memory (LSTM).

Also, it is inspired that CNNs are always combined in use the RNNs in the case of processing the sequential data, and such model is called convolutional recurrent neural network (CRNN). Essentially, we choose our hybrid-CNN-LSTM model to be have one convolutional layer and 3 LSTM for the purpose of temporal feature extraction.

Details with regarding to the hybrid-CNN-LSTM model is referred at Appendix A.3.

1.2. Motivation of Variational Auto-encoder and CNN hybrid approach

Variational autoencoder(VAE) is designed to capture latent variables which could be used to reconstruct input while keeping its main features [4]. EEG data could be noisy, and useful information could be weak considering effects caused by eyes blinking and muscle movement, breathing and so on. We looked at raw EEG data as 22 by 1000 images and realized VAE could be a good solution peeling off noises and only keep important features [1]. Intuitively it uses latent variables, and one or more settings of these variables could be used to generate output which is highly alike to input. To generate a sample from a dataset $X = \boldsymbol{x}^{(i)}{}_{i=1}^{N}$, made of N samples. The VAE framework follows two steps: (1) a value of latent variable $\boldsymbol{z}^{(i)}$ is generated using some probability distribution func-

tion(PDF) $p_{\theta^*}(\boldsymbol{z})$. (2) value $\boldsymbol{x}^{(\boldsymbol{i})}$ is generated from some conditional distribution $p_{\theta^*}(\boldsymbol{x}|\boldsymbol{z})$. Our goal is to optimize our θ such that with high probability, generated image will be like $\boldsymbol{x}$'s in our dataset. And this could translate to that we aim to maximize the integral of the marginal likelihood $p_\theta(\boldsymbol{x}) = \int p_\theta(\boldsymbol{z})p_\theta(\boldsymbol{x}|\boldsymbol{z})d\boldsymbol{z}$. Because the true values θ^* and $\boldsymbol{z}^{(\boldsymbol{i})}$ are unknown, and the above marginal likelihood is intractable, we therefore consider variational lower bound on above likelihood instead. And this goal could be realized by minimizing a loss function made of two terms. The so-called KL divergence measures approximation from the true posterior, another term corresponds to reconstruction loss [4].

Our VAE neural network follows the design in Figure 1. It features an encoder which compresses information into latent variables distribution; then decoder that generates output which carries main features of the input. Notice in the raw design shown on the left plot, sampling operation is non-differentiable, therefore in order to make backpropagation work, VAE uses a "reparameterization" trick. Instead of sampling latent variable directly from $\mathcal{N}(\mu(X), \Sigma(X))$, it samples an ϵ from a normal distribution first and multiply to $\Sigma(X)$ then plus $\mu(X)$. By performing reparameterization, we could manage to do backpropagation of the network.

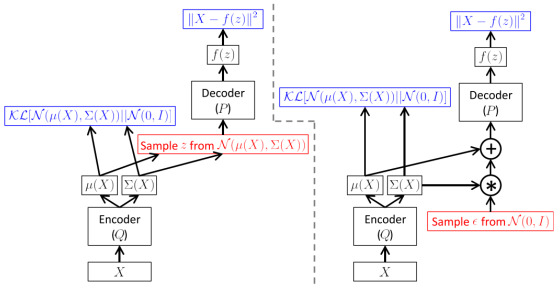

Figure 1. VAE implementation as a neural network and its reparameterization trick [2].

Convolutional Neural Network, CNN, is designed to extract the features with spatial relation. It works very well for the data, such as image, which have internal spatial relation. Using VAE as a filter to reduce the noise such as eyes blink, CNN could classify EEG signal better due to improved purity of data. We have two CNN models. One is ResNet[3], and other one is DeepConvNet[6]. ResNet is the first and one of the greatest deep convolutional neural network and DeepConvNet is a convolutional neural network which is specialized for EEG. We tried these two models based on our intuition.

ResNet Our ResNet is based on the architecture in the paper shown in Figure 2 However, ResNet is specifically designed

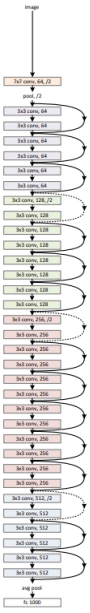

Figure 2. ResNet Achitecture

for Image Recognition and the convolution layers and batch normalization are all 2D. To deal with this situation, we modify the all the 2D layers back to 1D layers. Besiders, there is bottleneck layer inside the ResNet, we remove the bottleneck layer to increase the speed of training. Besides we have VAE model to filter the noise, we think it is okay to remove the bottleneck layer to reduce the complexity of the model.

DeepConvNet is a convolutional neural network specifically designed for EEG which is similar to ResNet. Its architecture is shown in Figure 3 DeepConvNet is a combination of residual network and ConvNet. It surveys a combinations of activation layers and regularization techniques. It achieves our best result across all the method.

In summary, we would like to combine VAE and CNN because it is intuitive to to combine them as our first trial. VAE can filter the noise and CNN can extract spatial relation between data.

2. Results

We implemented an vanilla convolutional neural network that was proposed in the paper [5], a variational auto-encoder and a hybrid-CNN-LSTM neural network. The vanilla convolutional neural network performs fairly well and can achieve around 63% accuracy on test dataset. The hybrid-CNN-LSTM neural network can also achieve around 60% accuracy on test dataset. We aimed to stack variational auto-encoder with vanilla convolutional neural network or hybrid-CNN-LSTM neural network because variational auto-encoder can denoise noisy dataset and act as a technique of data augmentation. However, due to lack

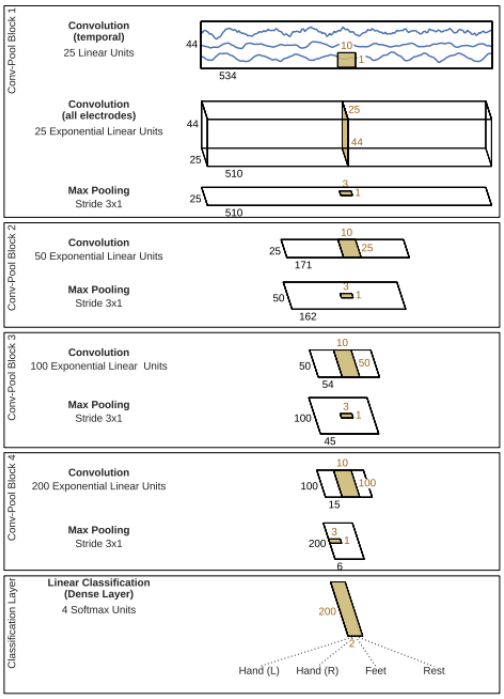

Figure 3. DeepConvNet Architecture

of regularization and proper metric to assess the effectiveness of our auto-encoder. It did not help us improve test accuracy over test dataset.

3. Discussion

Designing a complicated neural network is easy, but it will easily be prone to over-fitting and causes tremendous troubles. Various problems occurred while developing recurrent neural network. The first problem is vanishing gradient. We had a problem where the loss decreases down to zero in less than 10 epochs. There are several possible reasons that led to this problem. One of them is that the batch size was too small, and the network will quickly learn all the details in a mini-batch and easily overfit the data and then cause the network stop to learn because the loss decreases to zero. Another problem is that there were no regularization applied to the loss function. Without regularization, the data will also be easily over-fitted.

Printing out training loss and accuracy is also extremely important because they can both reflect part of the different underlying cost of the problem. Training accuracy was not properly printed while we were developing our neural networks. In fact, even though the loss is still decreasing, the training accuracy stays at 1.0 which indicates how bad the overfitting was. We applied k-fold cross validation while training the network, and that helped the overfitting problem quite a bit.

While training our RNN, several optimizers are proposed such as adam, rmsprop, and lbfgs. LBFGS is a notable optimizer because it uses second order optimization instead of gradient descent. However, the biggest problem with second order optimization is that it is extremely memory intense. Even though we trained our model on GPU, GPU's memory is not enough to perform second order optimization.

We also compared the proformance of RNN, LSTM, and GRU. It turns out that even we add weight regularization to RNN (because it can use ReLU). It still performs worse than GRU. LSTM performs the best among these three. However, it requires 4x more parameters to train and can therefore be memory intensive.

After we combined the convolutional layer and LSTM, the training and validation accuracy was extremely unstable. It was due to the problem that because there was not early stopping mechanism in our implementation, the gradient started jumping around around. It might also due to the fact that our RNN is a shallow network which means that there are only a few layers. Our deep convolutional neural network does not have this problem while training.

One last note about the dataset is that it is easy to get overfit because there is no enough data. This problem can be solved bu upsampling the time interval or use data augmentation technique. One of the data augmentation techique is to use auto-encoder to generate new data. Although we did not successfully gain benefit from implementing VAE, it is worth to learn and possibly use it in the future. Another technique that we applied is upsampling. We upsample the data by a factor of 2 and found immediate improvement over our network. Unfortunately, upsampling only works with the hybrid CNN+LSTM network at this moment because some of the network parameters in other networks are hard-coded and need to be recalculated after upsampling.

Variational autoencoder is supposed to suppress noise and therefore improve the later on CNN or RNN performances. We tested various latent variable dimensions and encoding/decoding dimensions as well as learning rate. I could find intermediate results meaningful, as noises are indeed suppressed. But the results of either VAE+RNN or VAE+CNN was not good as without VAE. But maybe a lack of experience of knowing what is considered as "noise" of EEG data made evaluation difficult.

Figure 4. Reconstructed VAE results. The upper half corresponds to part of original input, lower half corresponds to reconstructed result. Notice it did suppress some noise. But maybe a lack of experience of knowing what is considered as "noise" of EEG data made evaluation difficult.

References

[1] M. Dai, D. Zheng, R. Na, S. Wang, and S. Zhang. Eeg classification of motor imagery using a novel deep learning framework. *Sensors*, 19(3), 2019.

[2] C. Doersch. Tutorial on variational autoencoders. *ArXiv*, abs/1606.05908, 2016.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[4] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[5] 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.

[6] 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 eeg decoding and visualization. *Human Brain Mapping*, 38(11):5391–5420, Aug 2017.

Appendix

A. Methods

Preprocessing:

- Standardize dataset
- k-fold cross validation
- upsampling if needed
- shuffling dataset

A.1. CNN Models

A.1.1 ConvNet

Layers	Note
Conv2d	in_chan=1, out_chan=25, kernel=(1, 20)
BatchNorm2d	n_features=25
Activation	ELU
Conv2d	in_chan=25, out_chan=25, kernel=(22, 1)
BatchNorm2d	n_features=25
Activation	ELU
MaxPool2d	kernel=(1, 6), stride=3
Dropout	0.25
Conv2d	in_chan=25, out_chan=50, kernel=(1, 20)
BatchNorm2d	n_features=50
Activation	ELU
MaxPool2d	kernel=(1, 6), stride=3
Dropout	0.25
Conv2d	in_chan=50, out_chan=100, kernel=(1, 20)
BatchNorm2d	n_features=100
Activation	ELU
MaxPool2d	kernel=(1, 6), stride=3
Dropout	0.25
Conv2d	in_chan=100, out_chan=200, kernel=(1, 20)
BatchNorm2d	n_features=200
Activation	ELU
MaxPool2d	kernel=(1, 6), stride=3
Dropout	0.25
FC layer	in=200, out=4

Table 1. CNN Structure

A.2. Hybrid VAE and CNN

A.3. Hybrid LSTM and CNN Models

Layers	Note
VAE	
Linear layer fc1	dimension (22000, 2000)
Activation	Relu (h1)
Linear layer fc21	dimension (2000, 20) as mu
Linear layer fc22	dimension (2000, 20) as logvar
Reparameterization	$z = reparameterize(mu, logvar)$
Linear layer fc3	dimension(20, 2000) decode
Activation	Relu (h3)
Linear layer fc4	dimension(2000, 22000) decode
CNN	

Table 2. Hybrid VAE and CNN Structure

Layers	Note
Conv1d	in_chan=22, out_chan=40, kernel=20, stride=4
Activation	ReLU
Dropout	0.5
BatchNorm1d	40
MaxPool1d	pool=4, stride=4
RNN/LSTM/GRU	in=40, hidden=64, n_layers=3, dropout=0.5, bidirectional=True
BatchNorm1d	features=num_dir * 64
Linear 1	int=num_dir * 64 * (need calculate), out=64
Linear 2	int=64, out=4

Table 3. Combined Structure

B. Performance

Note: All performance are across all subjects.

Algorithm	Training Accuracy (%)	Validation Accuracy (%)	Testing Accuracy (%)
CNN.	89.70%	91.40%	65.00%
CNN+LSTM	81.85%	81.56%	61.40%