ECE 239AS Final Project: GAN + AE + LSTM + CNN for EEG classification

Wen-Dyan Hsu
UCLA
wenhsu@g.ucla.edu

Hua-En Li
UCLA
tonyli99@g.ucla.edu

Abstract

First, we used GAN to generate 10 times of new data. Second, we trained an Autoencoder with the new and old data to get a great encoder for dimension reduction. Lastly, we cascaded the pre-trained encoder from the Autoencoder with the classifier, which is LSTM + 2-layer CNN. With this architecture, we can get a great accuracy of 40% when we tested on the first subject.

1. Introduction

Considering the dataset is time series data, we thought that RNN architecture should be appropriate to tackle this task since RNN can capture the time-dependency property of the data by its recurrent connections. The initial architecture we tried was LSTM followed by 2-layer FC-net. However, the performance was not good enough. We inferred that maybe the given data amount was too small. Then we decided to build the GANs to generate a bunch of new data and fed it into the classifier. Hopefully the augmented dataset can increase the performance of our classifier. Also, we built an autoencoder before the classifier to reduce the dimension of time step axis of the dataset because autoencoder can compress the data and select fewer and more important features for the training process. This compression step can reduce the training time significantly. With this huge and dimension-reduced dataset, we tried more different architectures of the classifier. Finally, we found the LSTM followed by 2-layer Conv1D architecture can provide the best testing accuracy.

1.1. Preprocessing

First, we excluded the last three electrodes (EOG) data from the dataset, which are not strongly related to the EEG data. Second, we isolated each subject from the total training and validation datasets to train individual subject GANs and use those GANs to generate new data. Third, we transposed each data point by its axis 1 and 2 to match the input shape of the Keras API of LSTM, whose dimensions would become (batch_size, 1000, 22). Furthermore, we insert a new axis into the data at the second dimension to match the input shape of the autoencoder since there are Conv2D and Conv2D Transpose layers in our encoder and decoder respectively. With the autoencoder, we reduced the time step dimension by a factor of 2, which can reduce the training time hugely and select better features for training. Finally, the input shape of the LSTM + Conv classifier will become (batch_size, 1, time_steps/2, 22).

When it comes to the investigation of varied time period, we truncated the last 200, 400, 600 and 800 time steps of the real and generated data then went through the same preprocessing steps described above.

1.2. Data Augmentation

Before we fed the data into the autoencoder, we split the data of each subject into 4 groups according to their labels and trained different GANs for each group. Therefore, there are 9x4=36 different GANs for each subject and label group. We will provide detailed architectures of those GANs in the section below when we investigate further into GANs. With those 36 GANs, we generated a bunch of new data whose sizes are 10 times the original dataset. Hopefully, our classifier can benefit from the augmented dataset.

1.3. GAN

We introduced GAN after we found the original dataset is quite small, which has only 2115 data points. Considering there are 9 subjects and 4 classes for each subject, we split the training dataset into 36 groups and fed them into 36 separate GANs with the same architecture. As for the architecture of GAN, since the EEG data has time dependency, we were invoked by the idea from this paper[1] that we should introduce LSTM layers in the first layer of generator and discriminator of the GAN to maintain and capture the order and time dependency of the features. After introducing LSTM as the first layer in both generator and discriminator, we cascaded the LSTM with TimeDistributed FC-net in Keras API in the generator. On the other hand, we cascaded the last output of LSTM with normal FC-net in the discriminator. Here we introduced

batch normalization with momentum=0.8 in both generator and discriminator after their first FC-layer. Activations used are tanh in the generator and LeakyReLU (alpha=0.2) in the discriminator. Dropout was only introduced in the LSTM layer with input_dropout=0.2 and recurrent_dropout=0.2, not introduced in the following FC-net. Optimizer is Adam with learning_rate=1e-4, beta_1=0.9 and beta_2=0.999. With this architecture of GAN, we trained these GANs for 15 epochs. In the end, every GAN got a beautiful result with discriminator_accuracy around 50% and converged. Therefore, we have strong confidence that the generated data is fairly similar to the real data and can be used to train our autoencoder and the following classifier.

1.4. Autoencoder

Compared to the feature size=22, the dimension of time steps=1000 in this task is much bigger. Therefore, we started to consider dimension reduction and feature selection to extract the most significant features and reduce the training time of this task. Here we applied one convolution layer and one transpose-convolution layer in encoder and decoder respectively of the autoencoder. The motivation of applying convolution and transpose-convolution along the time step axis is that we can maintain the time dependency of the original data.

We trained the whole autoencoder with the GAN-augmented dataset, later on isolated the trained encoder from the autoencoder and cascaded it with the classifier. Then we will train the pre-trained-encoder + classifier architecture in our next step.

1.5. Classifier: LSTM + CNN

As we mentioned many times above, considering the time dependency of the dataset, we need to introduce LSTM in the first layer of the classifier. Then we applied 2 Conv1D layers after LSTM. The motivation of Conv1D here is also to capture the time dependency property of the features. In the end, we fed the features into a 4-neuron FC-layer to get the score of the data. Loss function used here is softmax function. We will elaborate more on how we arrived at this model, which has the best testing accuracy during our exploration of different architectures of the classifier. To see the detailed architecture and hyperparameters of the classifier, please refer to Fig. 5.

2. Result

We trained 14 models in total including models trained with the specific subject data only, and all the subjects data with different time steps. At first we believed that the model will perform better if we train each model with specific subject and use that model for that subject only. Also, we think that the more time steps we have, the better result we can get since it contains more information. The test accuracies are shown in Table 1. and Table 2. and more detailed results are shown below.

2.1. Classifier of individual subject

First, we tried to use the subject 1 data from GAN and real data to train a RNN model with the autoencoder to get the best RNN architecture. After numerous trials, we found the classifier architecture in Fig. 5 produces the best test accuracy. Then we input all the subjects data separately to train 9 models for each subject using the best architecture. The result in Table 1. shows that the architecture does not perform well on different subjects. The accuracy varies from 40% to 16%, which indicates that signal distribution among different subjects are quite different. Besides, we trained the model using all the data together and we got the test accuracy 27.5%. This also backs up our prior belief that the EEG signal varies so much among subjects that we should train models with different architecture for each subject.

2.2. Classifier across all subjects

After numerous trials, we chose to use the same RNN architecture to train across all the subjects with all the real data and GAN data. The test accuracy is around 27.5%, which is not so good. However, we noticed that the validation loss is still dropping drastically when epoch = 5, and maybe the model can learn much better if the number of epochs increase to 50 and give the model more time to train. Beside, since we have around 18000 data for training, the model is less prone to overfitting. Thus, increase the number of epochs will probably lead to better result instead of overfitting the training data.

2.3. Classifiers of varied time period across all subjects

As we can see in the Table 2., the model performs the best when time step = 600. Also, in the Fig 3. both the validation and training loss are still dropping, which means the model is very likely to become better when we increase the number of epochs. In Fig 4., compared to the model with time step = 600, the model with time step = 200 seems very desperate. Its validation loss remains the same through each epoch. We think this is because that first 200 time step signal does not contain enough information for model to classify the data. Also, the model performs worse when the time step is too large. As we can see in the Fig 2., the accuracy reaches the peak at time = 600 and start to going down after that. We believe it's because there are too many irrelevant time step data making it more difficult to learn the signal pattern. To sum up, the accuracy does not increase as we have data over longer period of time and it is better to use only first 600 time step signals to train the model.

3. Discussion

3.1 GAN

In our first version of GAN, we didn't introduce any RNN layer into the model, using only FC-net and convolutional layers. However, no matter how many epochs we trained this GAN model, it never converged to the result with discriminator_accuracy around 50%. It meant the data generated from this GAN model would not be real enough to fool the discriminator. Therefore, we started to think that maybe we should introduce the RNN structure into the first stage of both generator and discriminator of GAN to capture the time dependency of the data[1]. With this modification, we arrived at a GAN that can converge to discriminator_accuracy around 50% no matter which subject or class the training dataset belonged to. Then we isolated the generator from the converged GAN model and used the generator to create 10 times number of new data compared to the original dataset for each subject and class. With the huge new dataset, we found that the classifier started to learn much better than before. The classifier now can have a smoother curve of training and validation loss. Most importantly, it outperforms the previous model which didn't have a huge dataset in the sense of testing accuracy.

3.2. LSTM + CNN

The first architecture we tried is a vanilla 3-layer FC-net. The motivation is to get a sense of this dataset and setup the baseline performance of this task. However, the testing accuracy of this vanilla FC-net is below 20%. Therefore, we realized we must introduce RNN architecture to capture the time dependency property of the features since feeding the data directly into FC-net would make the model lose this property.

To capture this time dependency property, we introduced LSTM as our first layer of the classifier. We've tried different depth of LSTM such as 2-layer and 3-layer LSTM. However, due to the given small dataset, we found it was fairly easy to overfit the training set and get a bad performance in the testing set if we applied a deep and wide architecture. Therefore, we concluded that using only 1-layer LSTM in the first stage of the classifier is enough.

Later we tried LSTM + 2-layer FC-net architecture and found the performance was still not good enough. The most obvious reason we came up with was that FC-net would destroy the time dependency of data in the second stage, and it would make the model lose a lot of information of the dataset. Therefore, we decided to apply convolutional layers in the classifier. We've tried 1, 2, 3 and 4 layers of Conv1D in Keras and found that 2-layer Conv1D can outperform the others. The reason is also obvious that we can not apply deep convolutional architecture on this task since the dataset is small.

As for the size of each dimension in every layer of above architecture, we applied a decreasing approach for the number of hidden units in LSTM and filters in CNN. With this decreasing approach, the model can capture more meaningful features as it feeds forward the data along each layer. Then we arrived at our best architecture of classifier. To view the detailed architecture and hyperparameters of the classifier, please refer to Fig. 5.

3.2. Future work

Other aspects we can explore for this task are preprocessing, model architecture and well-tuned hyperparameters.

3.2.1 Preprocessing

We can do Fourier transform to the time-domain data and use the frequency responses as the features. Also, we can downsample the time-domain data first and normalize it to a dataset with mean=0 and variance=1.

3.2.2. Model architecture

We can try different orders of the layers of the classifier. For example, putting the CNN before the LSTM layer or even an interleaving mix of CNN and LSTM might be a promising model.

3.2.3. Hyperparameters

Due to the limited computing resources we have, we cannot explore too many different combinations of hyperparameters. Therefore, the hyperparameters we chose in our model may not be the optimal hyperparameters.

References

[1] Olof Mogren. C-RNN-GAN: Continuous recurrent neural networks with adversarial training. arXiv:1611.09904v1

Table 1. Test accuracies of models trained only with one particular subject data, time step = 1000

	S_0	S_1	S_2	$S3$	S_4	S_5	S_6	S_7	S_8
Test Acc	0.400	0.340	0.220	0.280	0.362	0.306	0.180	0.160	0.255

Table 2. Test accuracies of models trained with all the data but different period of time

	$Time_{200}$	$Time_{400}$	$Time_{600}$	$Time_{800}$	$Time_{1000}$
Test Acc	0.246	0.264	0.336	0.312	0.275

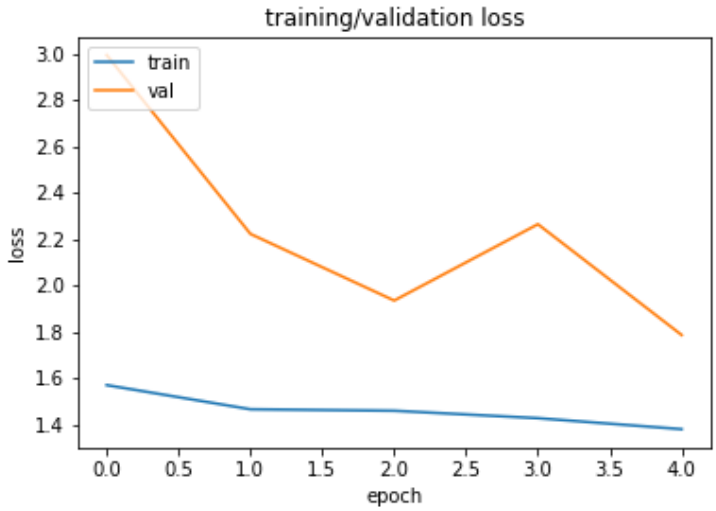

Figure 1. Loss plot for subject 0, time step = 1000

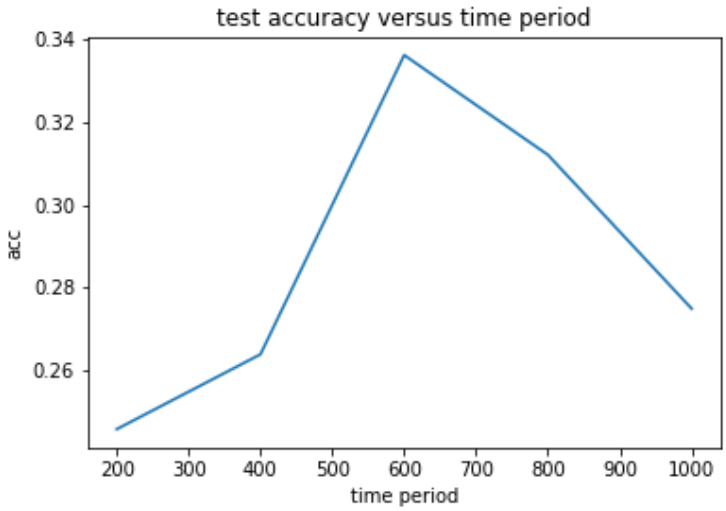

Figure 2. Test accuracy versus time period

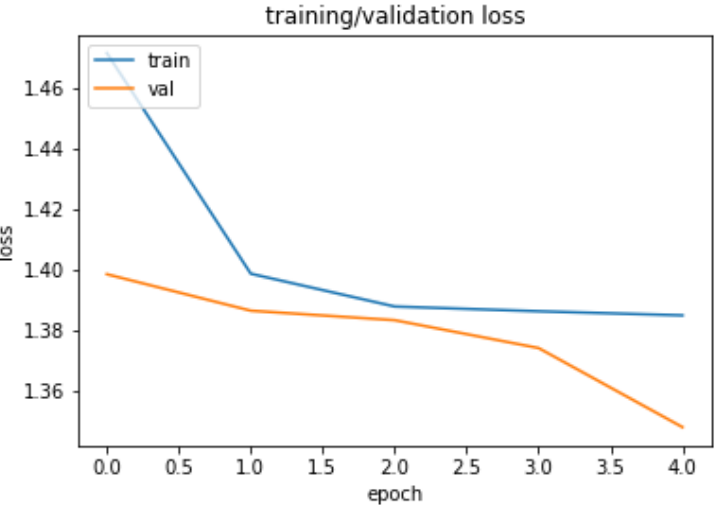

Figure 3. Loss plot for all the data, time step = 600

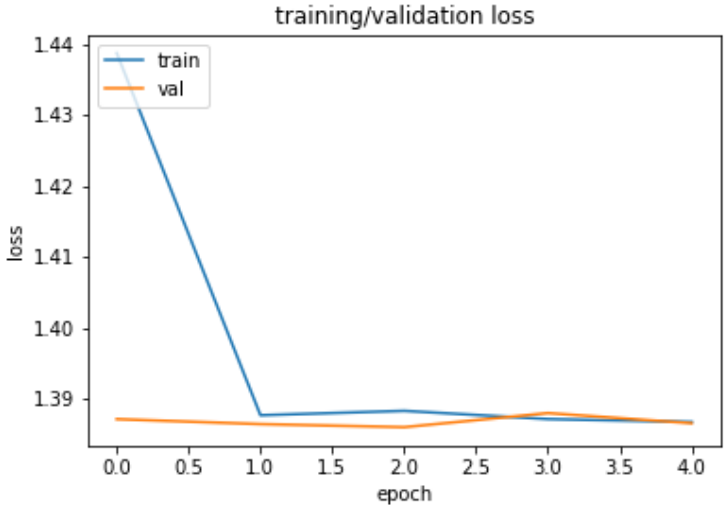

Figure 4. Loss plot for all the data, time step = 200

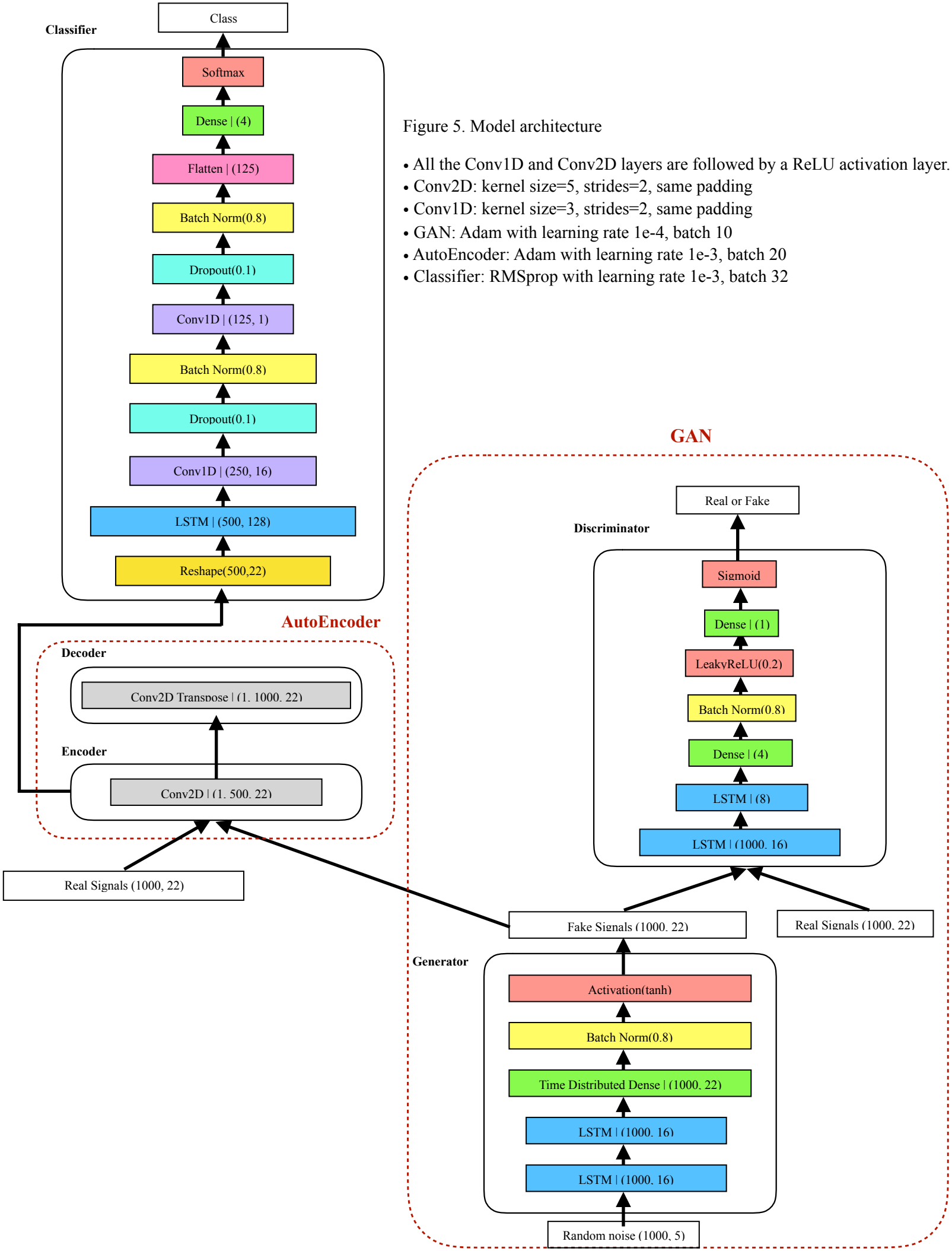

Figure 5. Model architecture

- All the Conv1D and Conv2D layers are followed by a ReLU activation layer.
- Conv2D: kernel size=5, strides=2, same padding
- Conv1D: kernel size=3, strides=2, same padding
- GAN: Adam with learning rate 1e-4, batch 10
- AutoEncoder: Adam with learning rate 1e-3, batch 20
- Classifier: RMSprop with learning rate 1e-3, batch 32