Learning from Sleep Data

Sizhang Zhao[1]
[1]**Georgia Institute of Technology, Atlanta, GA, USA**

Presentation link: `https://youtu.be/8nHf8Fly6Zg`

Abstract

Sleep is important to human health. Reduced sleep or abnormal sleep not only can make people suffer emotionally but also will bring severe consequences to human physical health. Manual workflow with recorded signals has been established to detemine the stages of sleep. However, classifying sleep quality is very labor-intensive and time-consuming, not to mention that recording signals can also be expensive as well. But being good at summarizing local structures and capturing time transition patterns correspondingly, CNN and RNN can privide a potential different but efficient way for handling the problem. In this paper, we will use public available sleep sensor data and apply deep learning methods above them to achieve automatic sleep staging. There are two different models being proposed in this paper, first one is to create hand made features and feed into Seq2Seq model, which can already achieve good results for major classes (sensitivity for major class can be as good as 80%) but it didn't perform well on minor classes. The other one is to use CNN to extract feature from the raw signals and then feed into Seq2Seq structure. We find the prediction power of minor classes got signicantly improved, the worst classes can achieve sensitivity as 50% with the model.

1 Introduction

To manually conduct sleep staging, sleep experts measure the quality of sleeps based on the signals recorded from sensors attached to human body. A set of signals from these sensors is called a polysomnogram (PSG), consisting of an electroencephalogram (EEG), an electrooculogram (EOG), an electromyogram (EMG), and an electrocardiogram (ECG). Those PSGs are recorded in 30s epochs. And based on the manual[9] from American Academy of Sleep Medicine (AASM), sleep experts will then classify sleep quality into 4 different categories, Rapid Eye Movement (stage R) sleep and 3 non-R stages, N1, N2 and N3. There will be 5 categories in total if we includes Wake quality as well. As we can tell, classifying sleep quality is very labor-intensive and time-consuming.

There have been many researches going on for to try to automate sleep stage scoring with machine learning or deep learning methods. Tsinalis et al.[8] firstly extracted time-frequency analysis based features and then feed the features into Stacked Sparse Autoencoder. To accomodate imbalanced class problem, they employed class balanced sampling with an ensemble of classiers, each one being trained on a dierent sample of the data. The model is trained with 20-fold cross validation and performance is compared against three exisiting studies. And it has been shown that the model has the best performance in the literature when classication is done across all ve sleep stages simultaneously using a single channel of EEG. However, the model is still based on hand-engineered features. Afterwards, Tsinalis et al.[1] proposed a CNN based methods where raw EEG signal without preprocessing as the input will be fed into the model. The model was trained and evaluated on a dataset of 20 healthy patients using 20-fold cross-validation. And they find the CNN based method can perform as good as a state-of-the-art hand-engineered feature approach. Similarly, Kaare et al.[3] proposed a transfer learning based CNN methods. Both EOG and EEG signals will be fed into the network. However, instead of try to best fitting the model, they firstly pretrained the model using at least 29 nights, and then this general model was presented with the rst-night from a single subject 10 times (meaning this additional training data consisted of 10 duplicates of the one training night). Finally, the model is tested on the second-night from the same subject. They eventually find the fine tuning methods generally increases model performance, particularly for difcult subjects. Being good at coping with sequence data, RNN has also been applied to the task. Huy et al.[2] utilized bidirectional recurrent neural networks (RNNs) with attention mechanism to encode EEG epoch divided sequences into high-level feature vector and then apply a linear SVM classifier on top of that, which demonstrates good performance on the Sleep-EDF dataset. Michielli et al.[5] proposed a two step RNN to better classify N1 and REM stages. They construct fifty-five time and frequency-domain features from EEG signal and extract useful features by feeding them into feature reduction algorithms. Then the constructed features are fed into two LSTM network, the first one performed 4class

classification (i.e. the five sleep stages with the merging of stages N1 and REM into a single stage) and the second one is targeted for 2-class classification (i.e. N1 vs REM). The overall percentage of correct classification for five sleep stages is found to be 86.7%. Not only CNN or RNN alone, the combination of them shows good performance as well. Akara et al.[4] proposed a hybrid methods with CNN and RNN. Firstly, two separate CNN with different kernal size are used to extract time-invariant features from raw singlechannel 30-s EEG epochs. Then, they employ two layers of bidirectional-LSTMs to learn temporal information such as stage transition rules. Residual network is used prior to the RNN part to help carry over time-invariant features directly to fully connected layers. They used a two step training to train their model as well, i.e. the CNN part is pretrained to directly predict class label before the complete model are fine tuned together. They used the model on two public datasets (MASS and Sleep-EDF) and the results showed that their model achieved similar overall accuracy and macro F1-score compared to the state-of-the-art methods on both datasets. Unsupervised learning methods are used for sleeping stages automatic detection as well. Martin et al.[7] used deep belief nets (DBNs) to replace hand feature construction steps. And the results are fed into HMM to accurately capture sleep stage switching. Traditional machine learning methods can also play a role in the game. Lajnef et al.[6] used models like a decision tree with multi-class Support Vector Machines. Unlike CNN and RNN can potentially construct feature from EEG signals, they have to hand engineer features based on expert knowledge for those models. The model was trained and tested on PSG data from 15 patients and performed better than standard multi-class procedures such as Linear Discriminant Analysis (LDA) and One-Against-All SVM.

In this paper, we focused the application of deep learning methods to automatic sleep staging. A LSTM based attention methods will be the base model as we can capture the time transition property of the signal over time. One model will construct manual features and feed them directly into the LSTM model similar as in Michielli et al's work[5]. The other model will feed the raw signal into CNN layers and the results will then be fed into LSTM model for the final prediction, similar as in Akara et al's work[4].

2 Problem Formulation and Infrastructure

Our goal is to predict the right stage label with the raw sensor signals. Local cluster with PySpark will be used to preprocess data. Edf data will be parsed to model readable format in this step. Hand made features will be constructed both in time domain and frequent domain in this preprocess step as well. Raw data will be fed into CNN feature net to build up neural net based features. Eventually, constructed features for each epoch will be fed into a LSTM model for Seq2Seq predictions. The pipeline is shown in Figure Main-1.

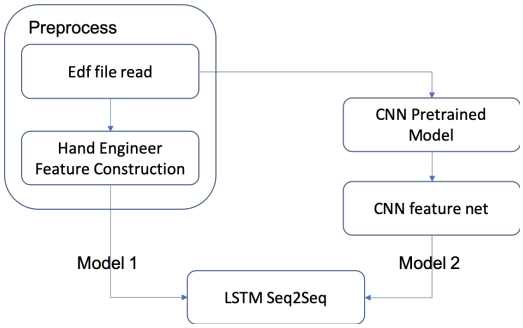

Figure Main-1: Model Architecture

3 Dataset

Sleep Cassette Study Data[10] from PhysioNet was used in the research. There are total 153 SC* files (SC = Sleep Cassette), which were obtained in a 1987-1991 study of age effects on sleep in healthy Caucasians aged 25-101, without any sleep-related medication. Each file corresponds to one night record for one individual. Two PSGs of about 20 hours each were recorded during two subsequent day-night periods at the subjects homes. The PSGs were manually annotated by sleep experts using 6 sleep stages: Wake, N1, N2, N3, N4 and REM. As mentioned in earlier session, there are multiple signals recorded, but for this research, we only utilize EEG signal(EEG Fpz-Cz and EEG

Pz-Oz). Due to the limited computational power I have, for manual feature model (model 1 in Figure Main-1), I choose 60% of the data as the population. And for CNN feature net based model (model 2 in Figure Main-1), only around 20 nights were chosen. Population are divided roughly 80% in training, 10% in validation and 10% in test set.

4 Exploratortory Data Analysis

4.1 Data Preprocessing

Two different datasets will be constructed from the raw data. Firstly, raw EDF data will be parsed into pandas dataframe as similar in the website[12]. For model 2, no actual processing will be needed in this step. But for model 1, we will also calculate manual features in the this step (more details in the following section). And then data will be divided into cohorts with given length, meaning continous several epochs will be grouped together to form a single sequence sample, which is suitable to be fed into LSTM structure. The length of the sequence is a tunable parameter, we will cover the experiment in the later section.

4.2 Class Distribution

As we mentioned earlier, there are 6 sleep stages altogether, but due to imbalanced data problem, we merge N3 and N4 together to form a new class, so we only have 5 different classes as our target class. Even after the merge, the classes are still very imbalaced, below are the summary of the class distribution for both models.

Sleep Stage	Training	% Training	Validation	% Validation	Testing	% Testing
W	131162	0.687	12251	0.659	10655	0.663
N1	7031	0.036	1830	0.098	1554	0.096
N2	31807	0.166	2831	0.152	3015	0.187
N3/N4	7831	0.041	538	0.028	158	0.010
REM	13096	0.068	1121	0.060	675	0.042
Sleep Stage	Training	% Training	Validation	% Validation	Testing	% Testing
W	28808	0.694	6527	0.640	7622	0.693
N1	1368	0.032	207	0.020	213	0.019
N2	6647	0.160	1664	0.163	2176	0.197
N3/N4	2051	0.049	909	0.089	271	0.024
REM	2625	0.063	888	0.087	715	0.065

Table 1: Class Distribution for both models, top is for manual RNN model, bottom is for CNN-RNN model

From the distribution, we can easily find the classes are highly imbalanced, we will use weighted sample later to construct the batch to try to alleviate the problem. Also we can find the distribution between training, validation and test are not exactly the same, especially on the small population class like N1 and N3/N4, this might bring harm to the generalization ability of our models.

4.3 Feature Construction

We need to manually calculate features from frequency and time domain for model 1. 10 different features are calculated similar as in website[11, 12], which brings 20 total features as we use 2 EEG signals. Time domain features: max, min, std, kurtosis, median of signals. Frequency domain features: alpha, beta, sigma, delta, theta spectual power band from power spectral density.

To better understand if the features are informative, we did the boxplot for all 20 features in training set in figure Main-2 (last page of the report). From the plots we can easily tell, all the features did have distribution difference across all the classes, frequency domain features are generally slightly more informative than time domain, and median is the less informative feature. Even though I have the choice to construct more and potentially more informative features, the plots show those features have the capability to distinguish between classes. I will continue modelling with those features.

5 Model Structure

5.1 Metric

Our problem is a typical classification problem, I will use CrossEntropy as the loss function to train and tune models. To better understand the performance of the model, confusion matrix will also be reported. However, we should note the speciality of RNN models when it comes to compute CrossEntropy loss. RNN can take variable length sequence as inputs of the model, and thus we need to compute our own padded version CrossEntropy here. More specifically, assume $P \in \mathbb{R}^{batch*max\_len*classes}$ as the output of our RNN model, we need to calculate:

$$P_{i,j} = \mathbb{1}_{C_{i,j}} log\_softmax(P_{i,j,:}) * \mathbb{1}_{M_{i,j}} \tag{1}$$

where i, j means the corresponding element in the matrix, $C_{i,j}$ is the real label of the data points, $M_{i,j}$ is the corresponding element in the mask matrix, where it will be one if the corresponding elements in P is the real datapoints other than padded datapoints. Sum of matrix P devided by number of real datapoints in the batch will give the padded CrossEntropy loss.

5.2 Seq2Seq Structure

As we mentioned in earlier sections, the base model of our structure will be Seq2Seq like one. So the raw batches will be fed into a encoder first and produce all the hidden vectors and final hidden vector. Then the final hidden vector will be fed into the decoder, where at each step, raw data will be the input. We will then calculate the attention score of the output hidden vector with the hidden vectors from encoder and calculate the weighted output vector based on the scores. The final projection layer will then project the output to have the same dimension as the classes. Detailed description of model structure can be seen below.

5.2.1 Encoder

Assume the max_len of the sequence is m and hidden size to be h. We will leave out batch size dimention in this section for simplicity. Bi-directional LSTM will be used here, at each timestamp, we will have,

$$h_i^{enc} = [\overrightarrow{h_i^{enc}}, \overleftarrow{h_i^{enc}}] \text{ where } h_i^{enc} \in \mathbb{R}^{2h\times 1}, \overrightarrow{h_i^{enc}}, \overleftarrow{h_i^{enc}} \in \mathbb{R}^{h\times 1} \qquad 1 \le i \le m \tag{2}$$

$$c_i^{enc} = [\overrightarrow{c_i^{enc}}, \overleftarrow{c_i^{enc}}] \text{ where } c_i^{enc} \in \mathbb{R}^{2h\times 1}, \overrightarrow{c_i^{enc}}, \overleftarrow{c_i^{enc}} \in \mathbb{R}^{h\times 1} \qquad 1 \le i \le m \tag{3}$$

where h are the hidden vectors and c are cell vectors.

5.2.2 Decoder

We will transform the final layer hidden and cell vectors from encoder as the starting cell and hidden states for decoder.

$$h_0^{dec} = W_h[\overrightarrow{h_i^{enc}}, \overleftarrow{h_i^{enc}}] \text{ where } h_0^{dec} \in \mathbb{R}^{h\times 1}, W_h \in \mathbb{R}^{h\times 2h} \tag{4}$$

$$c_0^{enc} = W_c[\overrightarrow{c_i^{enc}}, \overleftarrow{c_i^{enc}}] \text{ where } c_0^{dec} \in \mathbb{R}^{h\times 1}, W_c \in \mathbb{R}^{h\times 2h} \tag{5}$$

At each step of decoder, the input will be the concatenated vector from initial features $y_t \in \mathbb{R}^{f\times 1}$ (f denoted the number of features) and the output vector $o_{t-1} \in \mathbb{R}^{h\times 1}$ from the previous timestamp, the initial value of output vector o_0 will be zero vector. The concatenated input will be $\bar{y}_t \in \mathbb{R}^{(f+h)\times 1}$. And then at each timestamp, $\bar{y}_t$, h_{t-1}, c_{t-1} will be fed into the decoder (one directional LSTM).

$$h_t^{dec}, c_t^{dec} = \text{Decoder}(\bar{y}_t, h_{t-1}, c_{t-1}) \tag{6}$$

Next step will be the calculation of attention score, we are using multiplication methods to calculate distance and thus derive attention score between h_t^{dec} with all h_i^{enc}.

$$e_{t,i} = (h_t^{dec})^T W_{attr} h_i^{enc} \quad \text{where } e_t \in \mathbb{R}^{m\times 1}, W_{attr} \in \mathbb{R}^{h\times 2h} \qquad 1 \le i \le m \tag{7}$$

$$\alpha_t = \text{Softmax}(e_t) \quad \text{where} \quad \alpha_t \in \mathbb{R}^{m \times 1} \tag{8}$$

$$a_t = \sum_{i}^{m} \alpha_{i,t} h_i^{enc} \quad \text{where} \quad a_t \in \mathbb{R}^{2h \times 1} \tag{9}$$

To get the output, we will do additional output layer as below,

$$u_t = [a_t, h_t^{dec}] \quad \text{where} \quad u_t \in \mathbb{R}^{3h \times 1} \tag{10}$$

$$v_t = W_u u_t \quad \text{where} \quad v_t \in \mathbb{R}^{h \times 1}, W_u \in \mathbb{R}^{h \times 3h} \tag{11}$$

$$o_t = \text{Dropout}(\text{Tanh}(v_t)) \quad \text{where} \quad o_t \in \mathbb{R}^{h \times 1} \tag{12}$$

The final output with the same dimention as number of classes will be calculated as,

$$P_t = W_{final} o_t \quad \text{where} \quad P_t \in \mathbb{R}^{5 \times 1}, W_{final} \in \mathbb{R}^{5 \times h} \tag{13}$$

The output P_t is then ready to be fed into the padded CrossEntropy loss.

6 CNN Structure

This part is for the feature net with 1dCNN layers. Please note, due to the limited size of training data for CNN-RNN model and the depth of the model, we will firstly pretrain the CNN model to predict the classes and initialize the CNN part in CNN-RNN model with the pretrained model.

Two different kernel size will be used, both are tunable. A small one to try to detect local structure, a bigger one to detect longer structure. CNN layers are,

1. Small kernel convolution, with Relu activated following small kernel maxpool layer
2. Another small kernel convolution (2 * kernel size from previous step), with Relu activated following small kernel maxpool layer
3. Large kernel convolution, with Relu activated following large kernel maxpool layer
4. Another large kernel convolution (2 * kernel size from previous step), with Relu activated following large kernel maxpool layer
5. Concatenate results from 2 and 4, go through two fully connected layers and return outputs

For CNN-RNN model, the final output dimention will be the same as the input feature dimention as in RNN. For pretained model, there will be an additional layer to project the output to have the same dimention as the number of classes.

7 Results

7.1 Hand Feature RNN

7.1.1 Base Model

Base model will use 36 as hidden size, 0.001 as learning rate, 64 as batch size and 0.5 as dropout ratio. Each 50 of epochs will be grouped together as a training sequence. Results are shown in figure Main-3,

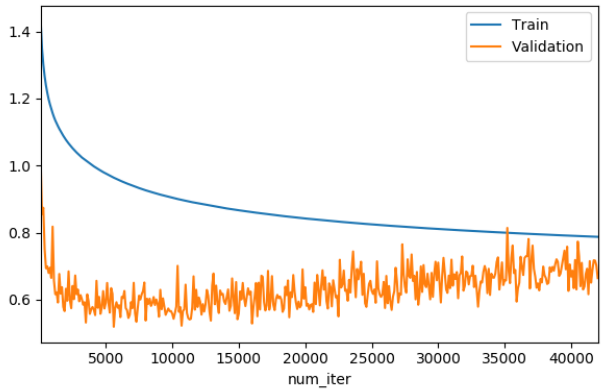

Figure 1: Train and Val loss for Base Model

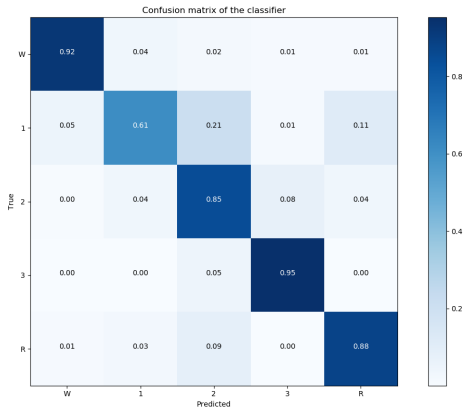

Figure 2: Training Confusion Matrix for base Model

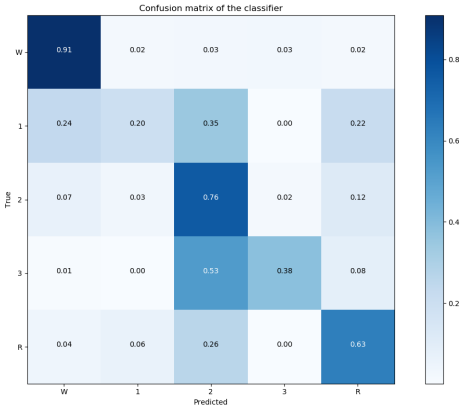

Figure 3: Validation Confusion Matrix for base Model

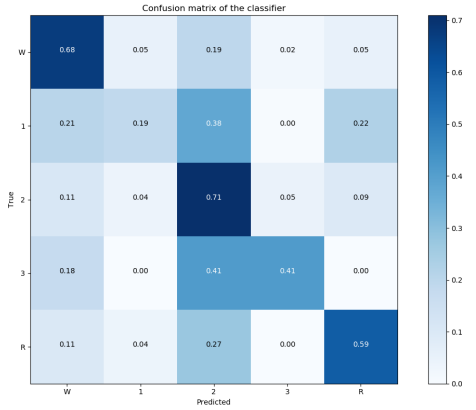

Figure 4: Testing Confusion Matrix for base Model

Figure Main-3: Loss and Confusion Matrix for Base Model

The base model can already achieve good performance. Please note I train the model up to 40000 iterations but for the confusion matrix, I took iteration 8000 as after that step, we tend to overfit the data from the loss chart that validation error started to increase (same methodology applied to all the following results unless specified differently). And the prediction on Wake, N2 and REM generally can generalize pretty well, but the predictions on N1, N3/N4 are still not good. That's probably because of the imbalanced classes problems as we discussed earlier. The weighted sample batch construction methods definately help with the training (as training accuracy are pretty high for almost all the classes), but it still doesn't help much for generalization to validation and test. We should also consider that our manual constructed features are very simple, so the performance of the model might also be constrained here.

Please note in the following sections, the images will be smaller to try to comply with the number of pages requirement, apologize in advance if it's hard to read. But we can tell the performance difference by color of confusion matrix.

7.1.2 Sequence Length

As we mentioned earlier, RNN can handle a sequence of obeservations. So how long of the sequence we will like to construct from epochs is also a good questions to answer. I use 50 as the default length as in the base model, another model with 300 as the sequence length is also constructed and trained, results are shown in figure Main-4.

We can easily tell the optimal loss becomes larger, and even though confusion matrix for training looks similar, the confusion matrix for validation and test are signicantly worse, especially for the two small population class.

There are 2 main reasons for that, firstly, when the sequence length becomes larger, we have less training data as the

total epochs are the same, which makes the model easier to overfit the data. Secondly, LSTM itself has limitation of how long sequence it is able to capture the trainsition mechanisms. If the sequence gets too long, the derivative will blur (exploding or vanishing) during backpropogation (LSTM is targeted to solve the problem but it still has limited capability). So for the next analysis, we will all use 50 as the sequence length.

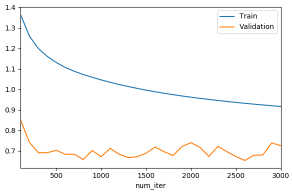

Figure 1: Train and Val loss

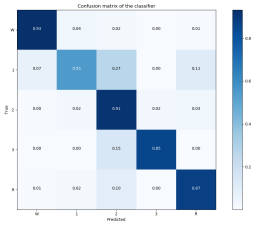

Figure 2: Training Confusion Matrix

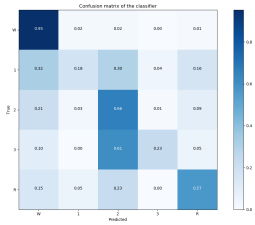

Figure 3: Validation Confusion Matrix

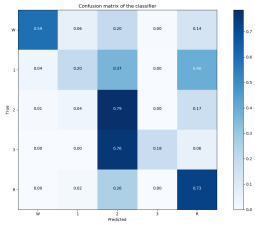

Figure 4: Testing Confusion Matrix

Figure Main-4: Loss and Confusion Matrix for Model with 300 sequence length

7.1.3 Hidden Size

Hidden size is an important parameter for our model, the defination of hidden size can be found in Model Strcuture section. In this section, we will change hidden size to be 72 and 18 and see how the model perform. Results for hidden size = 18 are shown in Figure Main-5 and results for hidden size 72 are shown in Figure Main-6.

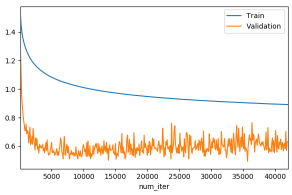

Figure 1: Train and Val loss

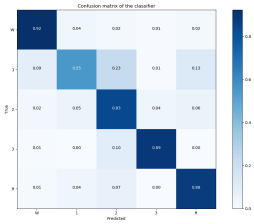

Figure 2: Training Confusion Matrix

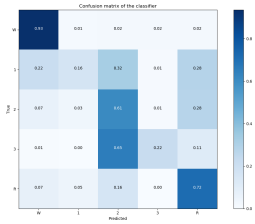

Figure 3: Validation Confusion Matrix

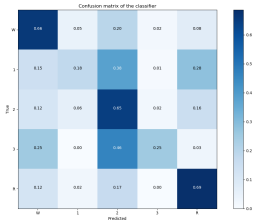

Figure 4: Testing Confusion Matrix

Figure Main-5: Loss and Confusion Matrix for Model with hidden size 18

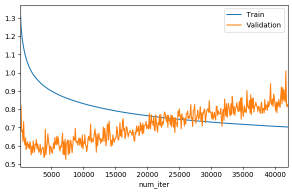

Figure 1: Train and Val loss

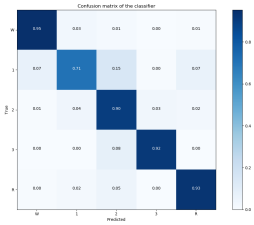

Figure 2: Training Confusion Matrix

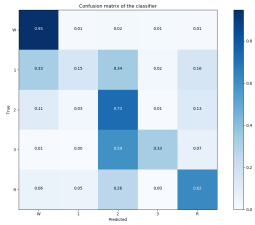

Figure 3: Validation Confusion Matrix

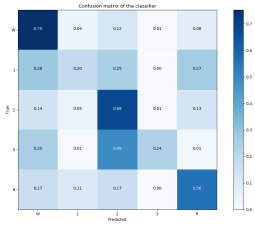

Figure 4: Testing Confusion Matrix

Figure Main-6: Loss and Confusion Matrix for Model with hidden size 72

Easy to notice, overfitting comes much quicker when hidden size is large, from the training perspective, seems hidden size = 72 is the best choice, but it didn't perform significantly better when it comes to validation or test sets. It's easy to understand that when hidden size becomes larger, the complexity of model increases and it has better ability to fit the data but it won't gurantee that the model will generailize well.

To better show the fitting power for different hidden size, we plot the training confusion matrix below at 40000's

iteration for hidden size 18, 36, 72 respectively in figure Main-7. We can easily find, the higher the hidden size, the more complex the model, and the better it will fit the training data.

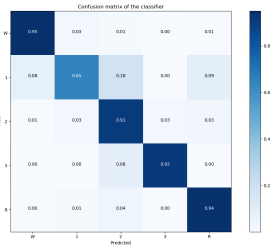

Figure 1: Hidden Size 18

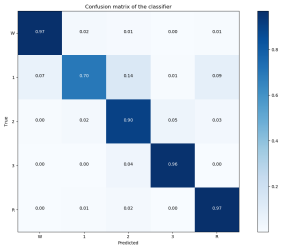

Figure 2: Hidden Size 36

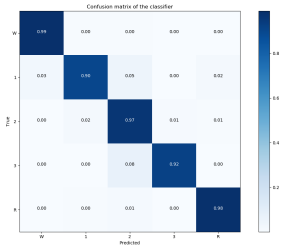

Figure 3: Hidden Size 72

Figure Main-7: Confusion Matrix for Model with different hidden size at 40000 iteration

From both the loss perspective and confusion matrix perspective, we can find hidden size = 18 can already perform quite well, to avoid overfitting, all of our following research will be based with hidden size as 18.

7.1.4 Learning Rate

Learning rate is also an important parameter for model tuning. In this section, we will change the default learning rate 0.001 to both 0.01 and 0.0001 and see how model perform. Results for learning rate 0.01 is shown in figure Main-8 and results for 0.0001 is shown in figure Main-9.

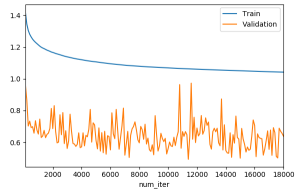

Figure 1: Train and Val loss

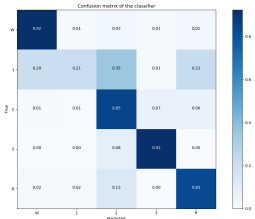

Figure 2: Training Confusion Matrix

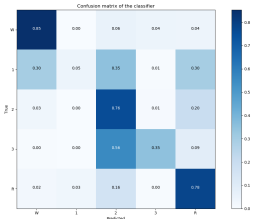

Figure 3: Validation Confusion Matrix

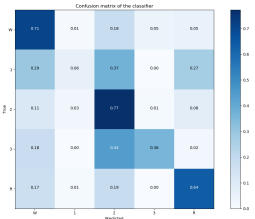

Figure 4: Testing Confusion Matrix

Figure Main-8: Loss and Confusion Matrix for Model with learning rate 0.01

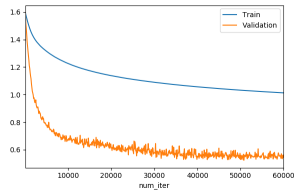

Figure 1: Train and Val loss

Figure 2: Training Confusion Matrix

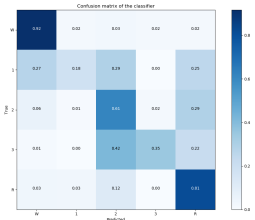

Figure 3: Validation Confusion Matrix

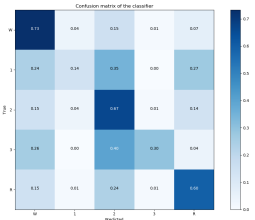

Figure 4: Testing Confusion Matrix

Figure Main-9: Loss and Confusion Matrix for Model with learning rate 0.0001

Easy to find when learning rate becomes larger, the convergence is quicker initially, but then the validation loss tend to fluctuate instead of converging. It makes sense that when gradient descent come near to the local optimal, if the learning rate is large, the model will actually diverge instead of converge. It can also be shown that the performance of the model becomes worse from all the confusion matrices. When learning rate is smaller, the convergence is smoother

but also slower. We can find even though we have come to the 60000 iteration, the training error is still above 1.0 and validation error is still around the range of learning rate = 0.001. It didn't perform signicantly better from the confusion matrix perspective either (it might perform slightly better for some of the classes though). So based on our research, we will still choose 0.001 as it gives the best tradeoff between convergence ability and speed.

We could have another option for decaying learning rate. But since our choice of lr as 0.001 already give out good results and the training converges well. So we didn't spend much time for it. It can potentially improve the convergence speed and optimal results, but coming up with a good decaying methods can also take long time to tune. Instead, we will move to pretraining of CNN and CNN-RNN model as the change of structure will potentially boost our performance more than tuning hyperparameters.

7.2 Pretrained CNN Model

In this section, we will start the pretraining of CNN model. Parameter are almost the same as previous section if the parameters are shared (like dropout ratio etc.). However, learning rate was adjusted to 0.005, as the time taken for one step is much longer. We make the learning rate larger so the convergence can becomes quicker. And specific parameter for CNN models are small kernal size is 4, larger kernal size is 16, channel number is 8 (stay constant over structure) and output size is 16.

And the performance is shown in figure Main-10. We can find the convergence of the model is very quick but since validation error didn't fluctuate that much initially, I think the learning rate 0.005 should be good enough. From the confusion matrix, we can find all of the classes other than N1 can be predicted quite well even in Validation and Test set. But N1 predictions are very bad that the model cannot even tell in the training set. And the lowest validation nearly corresponds to number of iteration 175. And the model parameter will be copied over to following CNN-RNN model when it's initialized.

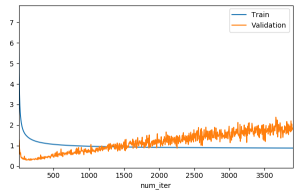

Figure 1: Train and Val loss

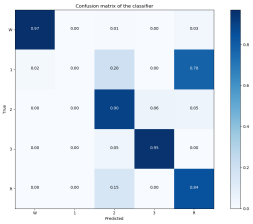

Figure 2: Training Confusion Matrix

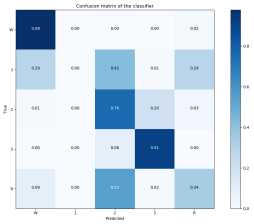

Figure 3: Validation Confusion Matrix

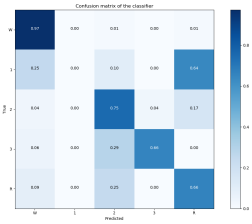

Figure 4: Testing Confusion Matrix

Figure Main-10: Loss and Confusion Matrix for Pretrained CNN model

7.3 CNN-RNN Model

We take our pretrained CNN model in and started to train CNN-RNN model. The parameter are the same as before unless specified. So because we care for the results to be accurate for the final CNN-RNN model and we also want the model to run faster, we choose learning rate to be 0.002 (0.005 is too large and might harm the final convergence and 0.001 is too small that it might take too long to converge). CNN structure is the same as the pretrained model and we only reseach the influence of hidden size in seq2seq in this section.

Hidden size for 18 is shown in figure Main-11. Hidden size 36 is shown in figure Main-12. And hidden size 72 is shown in figure Main-13.

We can easily find the hidden size 18 is not powerful enough and even though the fitting especially for class N1 got improved but only very little. And thus we tried with 36 and 72. Hidden size 36 gives out the most promising results, its performance on N1 got significantly improved, and not only for training set but also on validation and test set. And its performance is better compared to hand made feature RNN models too, especially on those two small population classes, N1 and N3/N4. This model which combined the CNN feature net to capture local strucures of signal and RNN to capture time transition structure, achieve the most superior performance and it generalizes pretty well. Hidden size

72 started to overfit the data already, based on the loss chart that validation error tends to be increasing and fluctuating, and also the confusion matrix on validation set and test set are both not as good as hidden size 36.

Based on the analysis so far, the CNN-RNN model with hidden size 36 seems to be the most promising model as it can not only fit the training data well but it can also generalize to validation and test set quite well too.

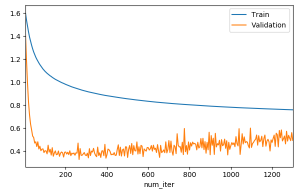

Figure 1: Train and Val loss

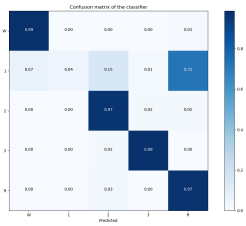

Figure 2: Training Confusion Matrix

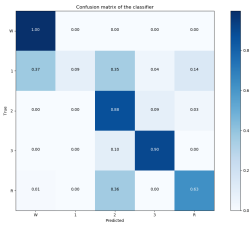

Figure 3: Validation Confusion Matrix

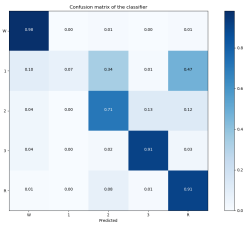

Figure 4: Testing Confusion Matrix

Figure Main-11: Loss and Confusion Matrix for CNN-RNN model with hidden size 18

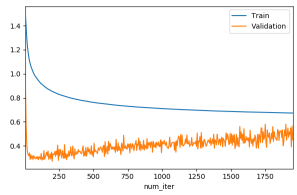

Figure 1: Train and Val loss

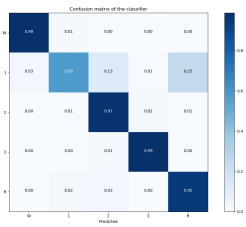

Figure 2: Training Confusion Matrix

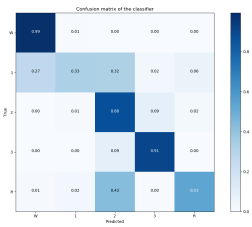

Figure 3: Validation Confusion Matrix

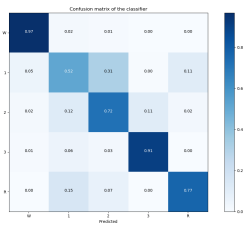

Figure 4: Testing Confusion Matrix

Figure Main-12: Loss and Confusion Matrix for CNN-RNN model with hidden size 36

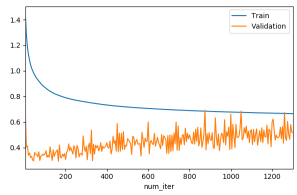

Figure 1: Train and Val loss

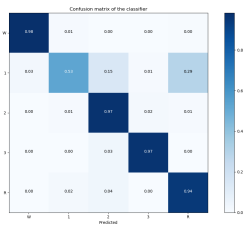

Figure 2: Training Confusion Matrix

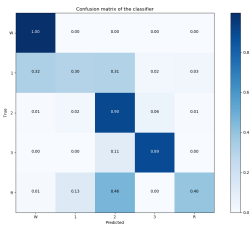

Figure 3: Validation Confusion Matrix

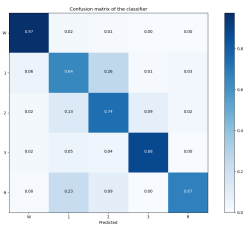

Figure 4: Testing Confusion Matrix

Figure Main-13: Loss and Confusion Matrix for CNN-RNN model with hidden size 72

7.4 Results Compared with Literature

From the above research, we can find we can already achieve good performance with CNN-RNN model, but how does our results perform against literatures? We will compare our results with work from Huy et al[2]. They used a similar structure as me but our design of RNN structure are almost different. The comparison of the sensitivities of results are shown in Figure 2. And results for test sets will be presented for my models. Huy's results are from mean of 20 fold cross validation. SeqSleepNet is his RNN-RNN structure, DeepSleepNet is CNN-RNN structure, ARNN is RNN only structure and CNN is CNN only structure.

Stage	Hand RNN	Pretrained CNN	RNN-CNN	SeqSleepNet	DeepSleepNet	ARNN	CNN
W	0.93	0.97	0.97	0.89	0.88	0.85	0.84
N1	0.16	0.0	0.52	0.61	0.57	0.37	0.41
N2	0.61	0.75	0.72	0.91	0.89	0.89	0.88
N3/N4	0.22	0.66	0.91	0.80	0.84	0.79	0.79
REM	0.72	0.66	0.77	0.93	0.90	0.94	0.93

Table 2: Class sensitivies for all models

From the comparison, we can easily find even compared to cutting edge results from literature, our model performed not bad at all. After switching to CNN-RNN model, our predictions sensitivities for N1, N3/N4, those two minor classes got signicantly improved and behave almost as good as literature results. However, we can still find out that our predictions of N2 and REM are consistently worse from literature. One possible reason is that the distribution of the raw data might differ between us and the literature as the performance for those two classes is consistent between all our models. Also Huy used a larger dataset with 200 subjects and he trained the model with 20 fold cross validation. Due to the limited computational power I have, I couldn't run my model to the scale as him (training of CNN-RNN model with my local GPU already take me almost an entire day for each hyperparameter set). So comparatively small dataset might also contribute to the lower performance. The different structure of models might be a reason as well, but based on the results, I can't draw a conclusion that our structure performs signicantly worse than him but on the contrary, our model performs better for some classes.

8 Conclusion

In this paper, we presented two different models to conduct automatic sleep staging. The base model is Seq2Seq to capture time transition structures. One model use hand made feature directly to feed into the model and the other model use CNN to extract features from raw signals before it got fed into the Seq2Seq model. We find the CNN-RNN model with hidden size 36 and CNN pretrained performed the best. Not only it fit the training data pretty well, but also it can give the best generalization abilities. And our models perform not bad even compared with literature results, it even peformed well for some of the classes than literature. The potential way to further improve the performance of the model is to either include more data for training or use ensemble methods as we already observed that different model or different parameter choice can give the models strengths in different area. Also we can tune the structure of CNN-RNN model more if we have enough computational power.

9 Challenges and Contributions

The first challenge for sure is data preprocessing, the raw data is encrypted and thus I need to build up PySpark pipeline to parse the data and process. It take me sometime to try to figure out how and what's worse, some of the files can't be parsed with my methods, so it brought additional challenge. The datasets is quite large, which also brought a lot trouble for local cluster, especially for CNN-RNN model when we need to save all the raw signals, the requirement of memory for driver node got increase signicantly for it and that's also the reason why we couldn't use larger datasets for the model as memory will leak.

The model part is quite smooth to be honest, as reading through literatures give me a lot of idea and the structure of the model fits the problem structure quite well. It did take me a lot of time to build up the pipeline and experimenting many choices only by myself is also challenging. But glad I came through and gained a lot of understanding in the jouney.

It is challenging to build up all the structure and conducting all the experiments only by myself. But I will say it's really a good experience and I learned a lot through the whole process.

References

1. O. Tsinalis, P. M. Matthews, Y. Guo, and S. Zafeiriou, Automatic sleep stage scoring with single-channel EEG using convolutional neural networks, arXiv:1610.01683, 2016.
2. H. Phan, F. Andreotti, N. Cooray, O. Y. Chen, and M. De Vos, Automatic sleep stage classification using single-channel eeg: Learning sequential features with attention-based recurrent neural networks, in Proc. EMBC, 2018, pp. 14521455.
3. K. Mikkelsen and M. De Vos, Personalizing deep learning models for automatic sleep staging, arXiv Preprint arXiv:1801.02645, 2018.
4. A. Supratak, H. Dong, C. Wu, and Y. Guo, DeepSleepNet: A model for automatic sleep stage scoring based on raw single-channel EEG, IEEE Trans. on Neural Systems and Rehabilitation Engineering, vol. 25, no. 11, pp. 19982008, 2017.
5. Michielli, N., Acharya, U. R., Molinari, F. (2019). Cascaded LSTM recurrent neural network for automated sleep stage classification using single-channel EEG signals. Computers in biology and medicine, 106, 71-81.
6. Lajnef T, Chaibi S, Ruby P, Aguera PE, Eichenlaub JB, Samet M, Kachouri A, Jerbi K. Learning machines and sleeping brains: automatic sleep stage classication using decision-tree multi-class support vector machines. Journal of Neuroscience Methods, 250, pp. 94-105, 2015
7. M.La ngkvist,L.Karlsson,andA.Loutfi, Sleepstageclassification using unsupervised feature learning, Advances in Artificial Neural Systems, vol. 2012, pp. 19, 2012.
8. O. Tsinalis, P. M. Matthews, and Y. Guo, Automatic sleep stage scoring using time-frequency analysis and stacked sparse autoencoders, Annals of Biomedical Engineering, vol. 44, no. 5, pp. 15871597, 2016.
9. Iber C, Ancoli-Israel A, Chesson A, Quan SF. The AASM manual for the scoring of sleep and associated events: rules, terminology and technical specications, Westchester, IL, American Association of Sleep Medicine, 2007
10. B Kemp, AH Zwinderman, B Tuk, HAC Kamphuisen, JJL Obery. Analysis of a sleep-dependent neuronal feedback loop: the slow-wave microcontinuity of the EEG. IEEE-BME 47(9):1185-1194 (2000).
11. https://raphaelvallat.com/bandpower.html
12. https://mne.tools/dev/auto_tutorials/sample-datasets/plot_sleep.html#sphx-glr-auto-tutorials-sample-datasets-plot-sleep-py%00

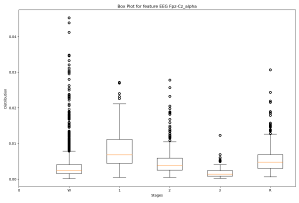

Figure 1: Fpz-Cz Alpha

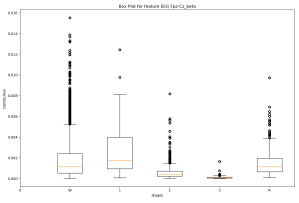

Figure 2: Fpz-Cz Beta

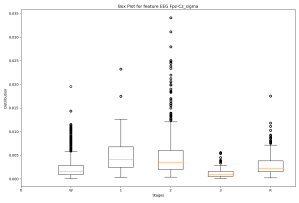

Figure 3: Fpz-Cz Sigma

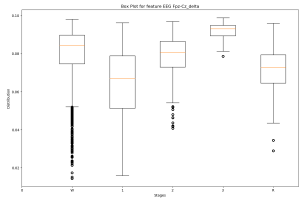

Figure 4: Fpz-Cz Delta

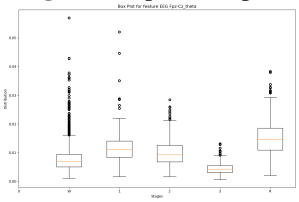

Figure 5: Fpz-Cz Theta

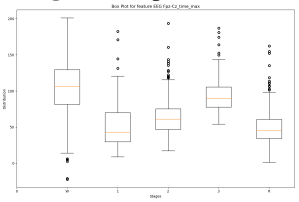

Figure 6: Fpz-Cz Max

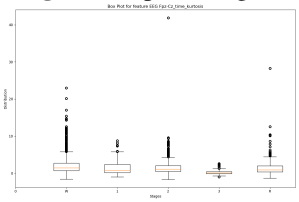

Figure 7: Fpz-Cz Kurtosis

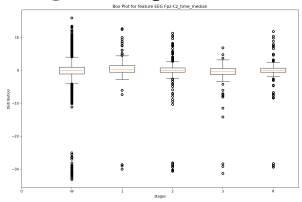

Figure 8: Fpz-Cz Median

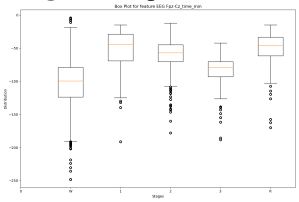

Figure 9: Fpz-Cz Min

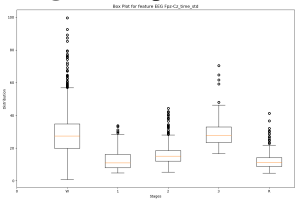

Figure 10: Fpz-Cz Std

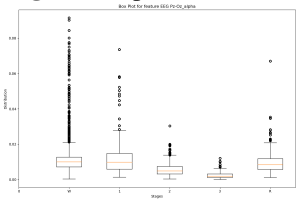

Figure 11: Pz-Oz Alpha

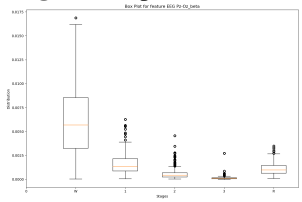

Figure 12: Pz-Oz Beta

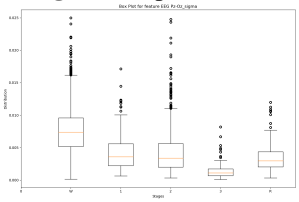

Figure 13: Pz-Oz Sigma

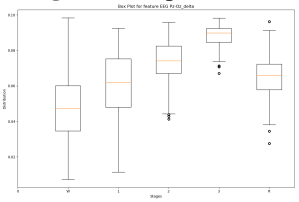

Figure 14: Pz-Oz Delta

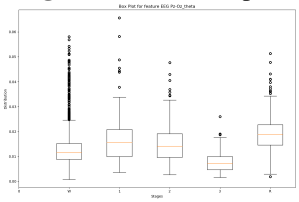

Figure 15: Pz-Oz Theta

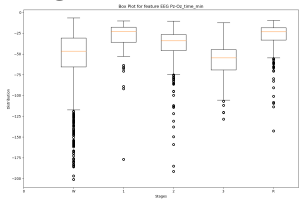

Figure 16: Pz-Oz Min

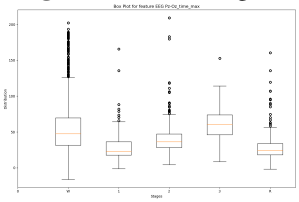

Figure 17: Pz-Oz Max

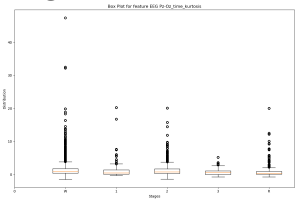

Figure 18: Pz-Oz Kurtosis

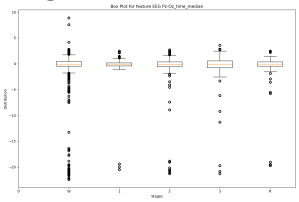

Figure 19: Pz-Oz Median

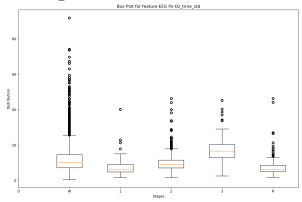

Figure 20: Pz-Oz Std

Figure Main-2: Box plots for all 20 features