The background of the slide is a blue-tinted photograph of the Seoul National University Hospital building. The building is a large, multi-story structure with a modern architectural style, featuring a curved glass facade on the lower levels and a more traditional grid-like facade on the upper levels. The text is overlaid on this image.

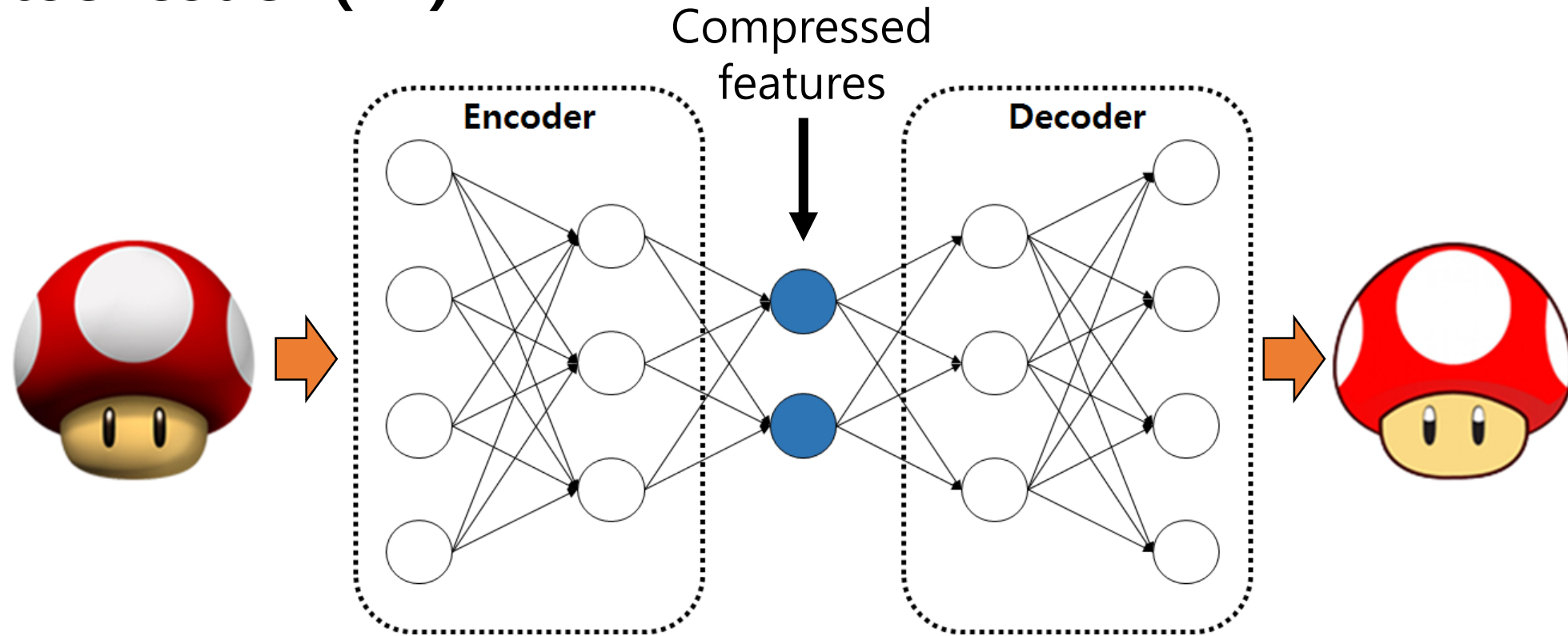
2019 Summer School of Biomedical Engineering

Deep Learning based Biosignal Processing: Autoencoders & Recurrent Neural Networks

2019-8-24

Joonnyong Lee PhD
Biomedical Research Institute
Seoul National University Hospital

Autoencoder (AE)



- Autoencoders are neural networks trained to output the input
- The input is reduced/compressed, and the key features are learned
 - Any spurious noise/data are eliminated during feature reduction

Autoencoder (AE) 실습

- ① Validation mini-batch loop을 만드세요 (line 205)
- ② Optimizer를 변경하고 학습 시간을 비교하세요 (line 158)
- ③ Loss function을 L1으로 변경하고 학습 결과 (loss)를 비교하세요 (line 154)
- ④ Output layer에 activation function을 추가하고 결과를 비교하세요 (line 136)
- ⑤ Hidden layer에 노드수와 activation function을 변경하고 결과를 비교하세요 (line 127)
- ⑥ Autoencoder을 hidden layer 수를 3으로 변경하고 결과를 비교하세요 (line 124)
- ⑦ 학습된 autoencoder에 0으로 된 데이터를 넣어서 나오는 결과를 확인하세요 (line 239)

1) Validation mini-batch loop

Training

```
# create a variable to hold training loss value after each epoch
train_loss = 0
# set a for-loop to go through the training set
for i in range(total_batch):

    # set index for training data batch
    batch_index = i * batch_size

    # extract training data batch
    batch = train_input[batch_index:batch_index + batch_size]
    label = train_output[batch_index:batch_index + batch_size]

    # run optimizer and calculate loss
    _, loss1 = sess.run([optimizer, loss], feed_dict={X: batch, Y: label, prob: 0.5})

    # add loss at end of batch to the total training loss for this epoch
    train_loss += loss1/total_batch

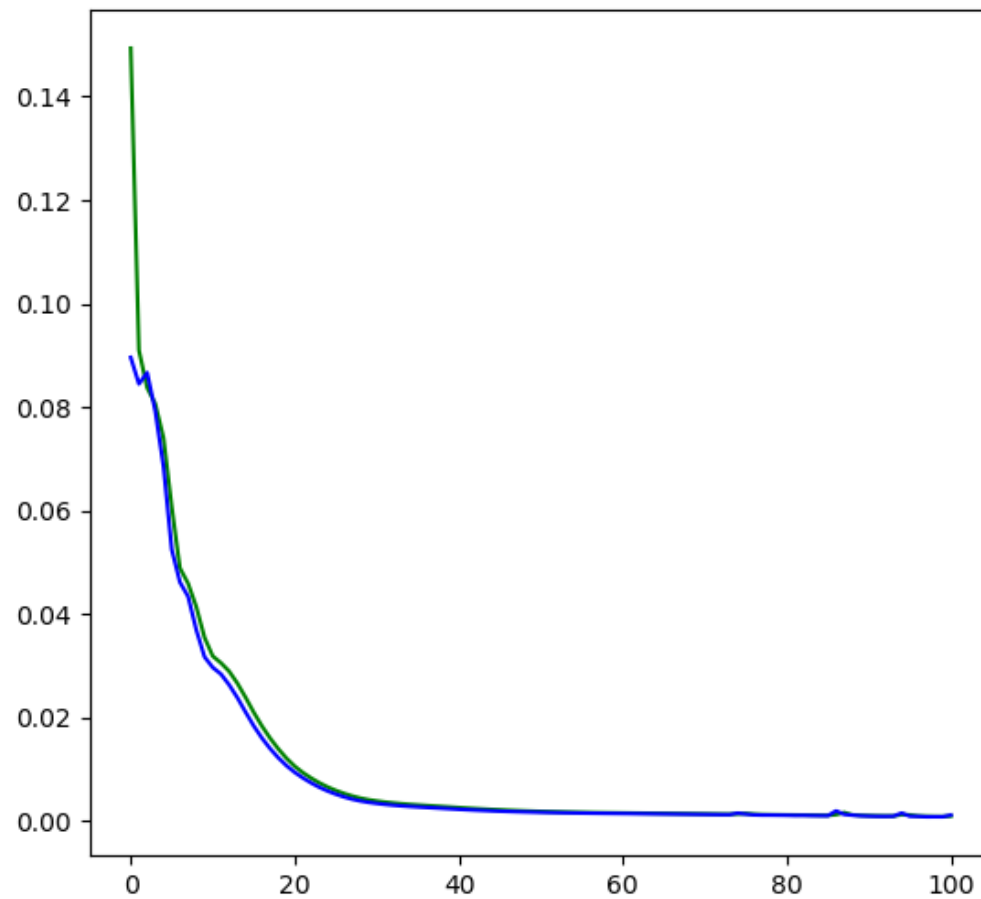
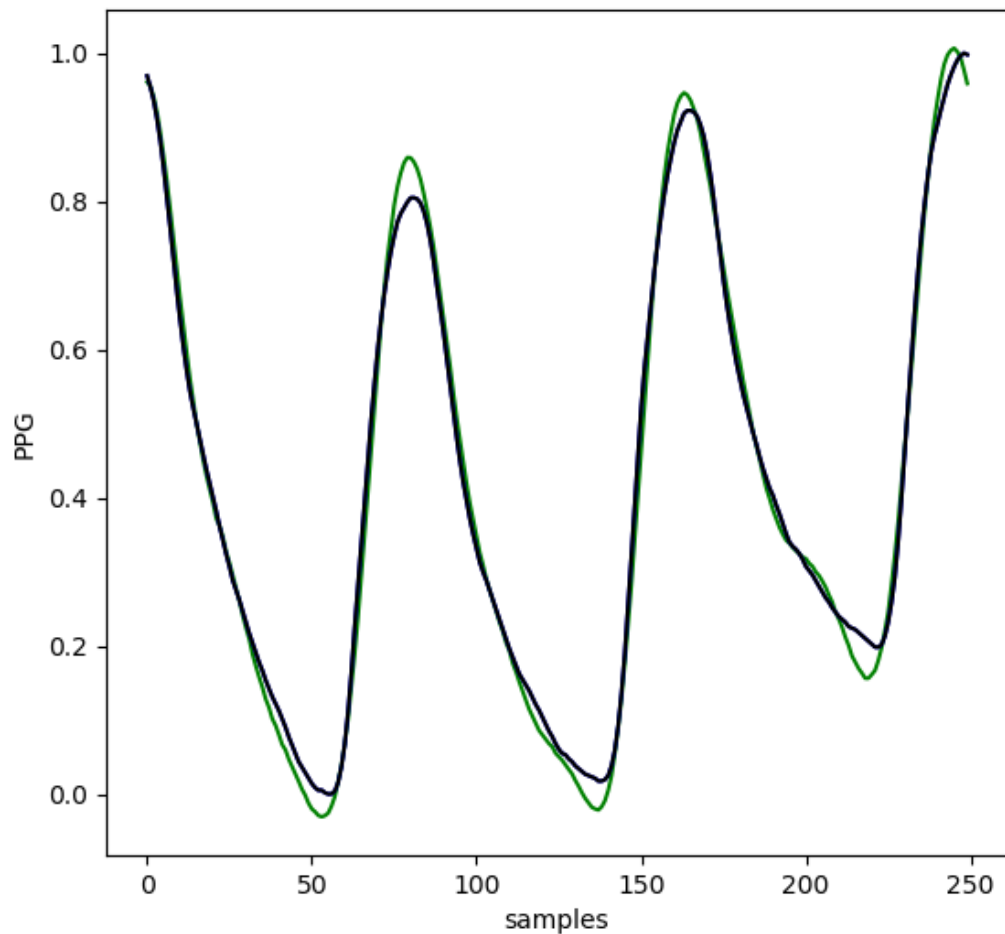
train_loss_epochs.append(train_loss)
```

Validation

2) Optimizer 변경

Adam

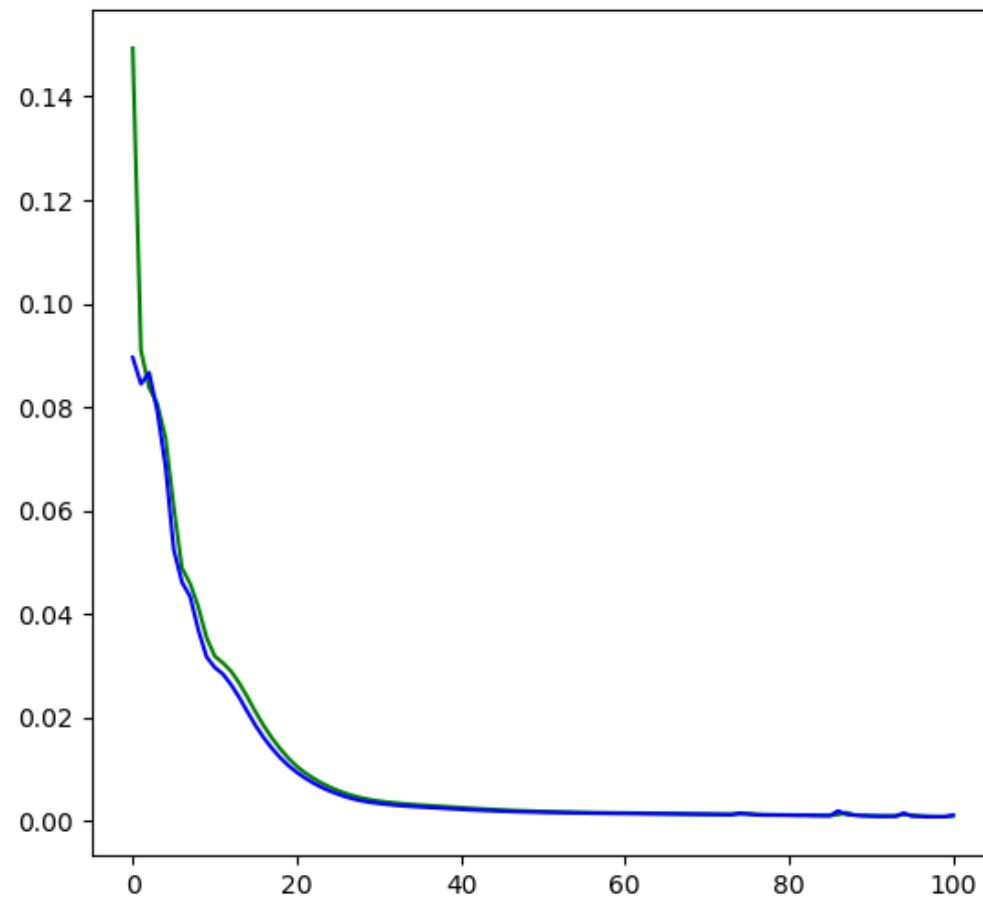
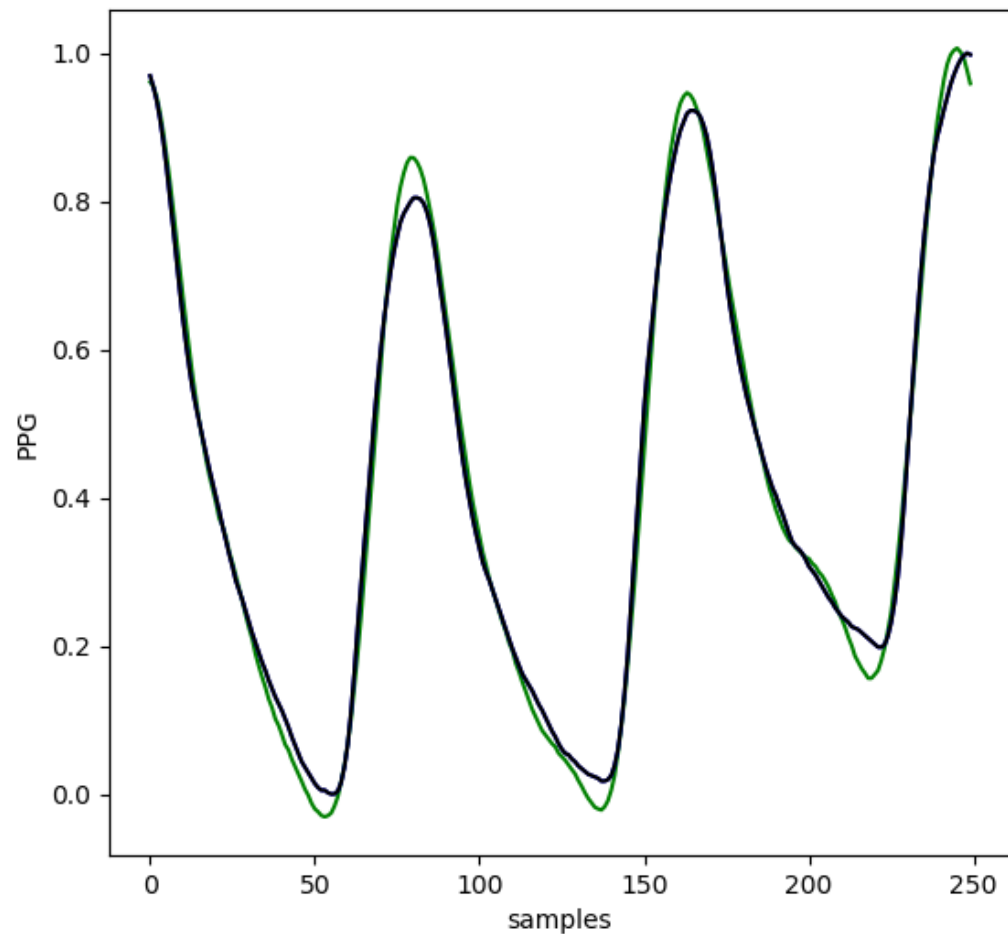
```
optimizer = tf.train.AdamOptimizer(learning_rate=0.01).minimize(loss)
```



3) Loss Function 변경

L2 loss

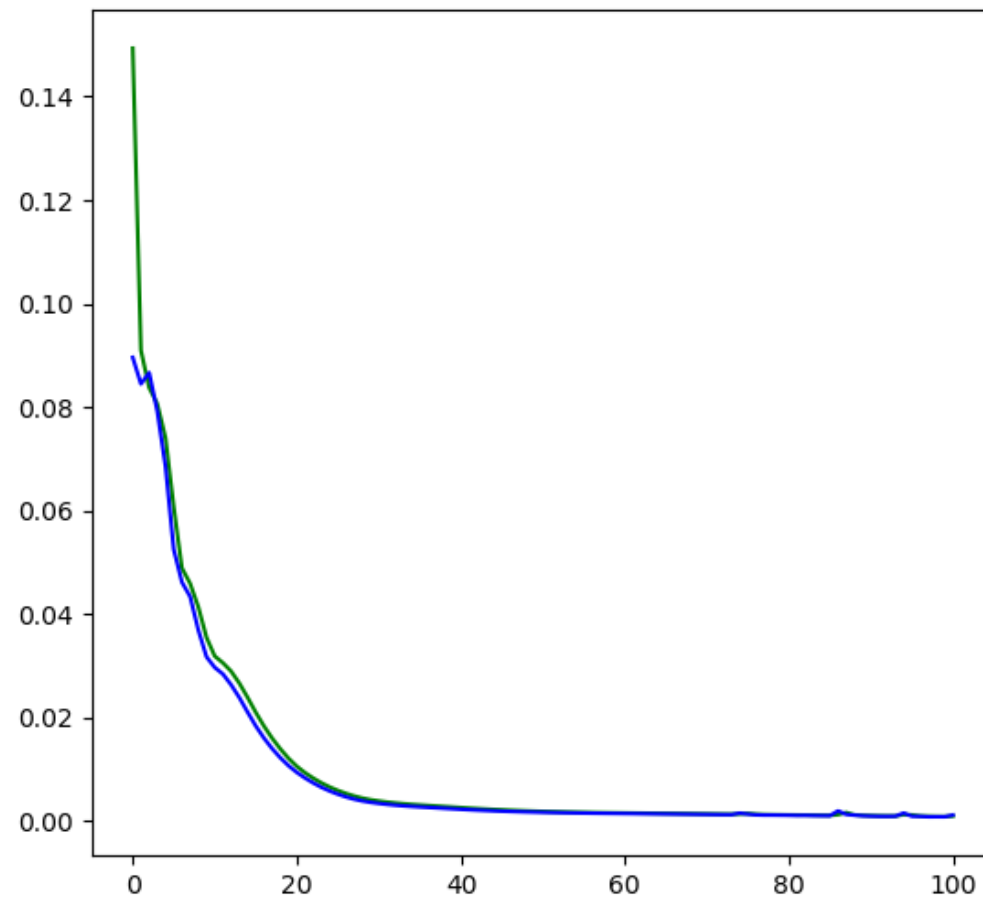
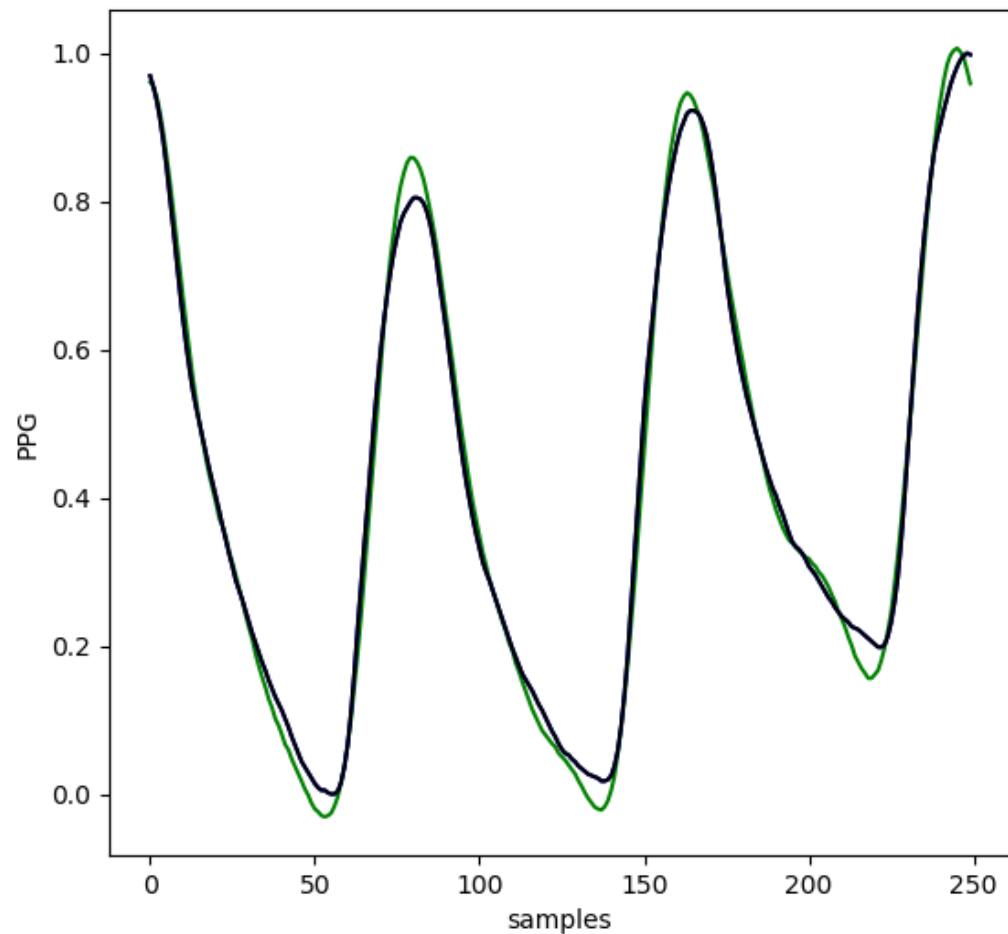
```
loss = tf.losses.mean_squared_error(Y, prediction)
```



4) Output Activation 추가

No Activation

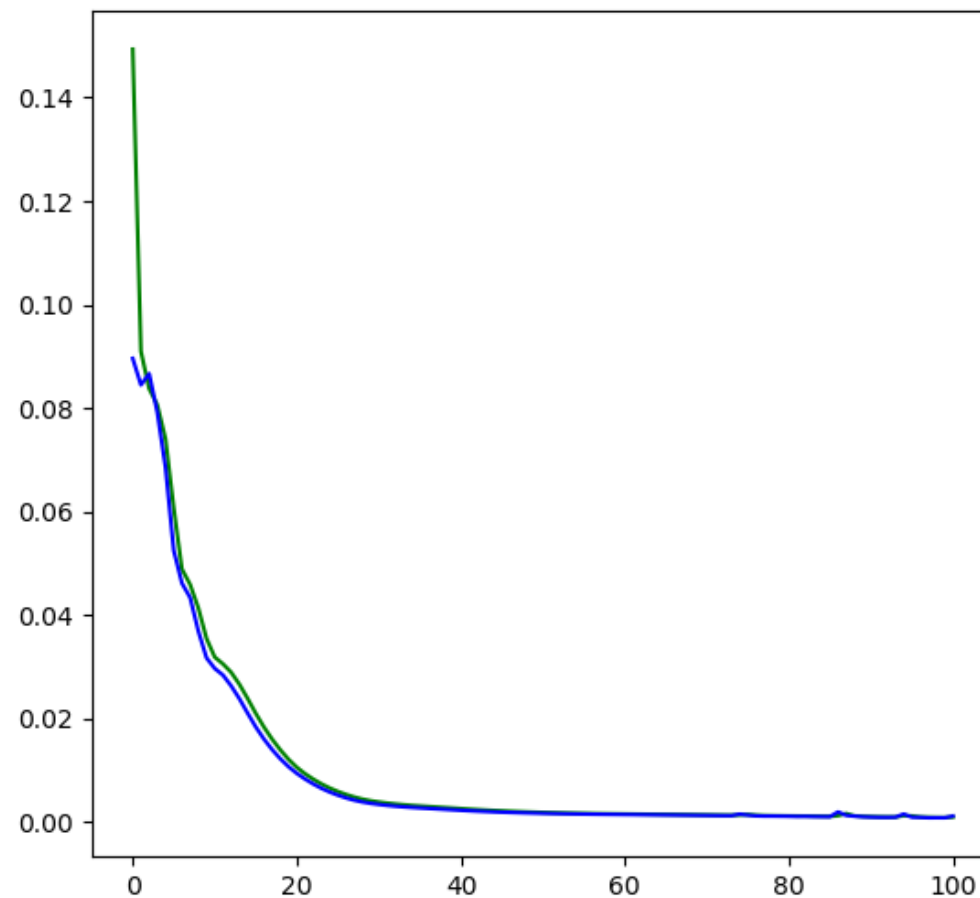
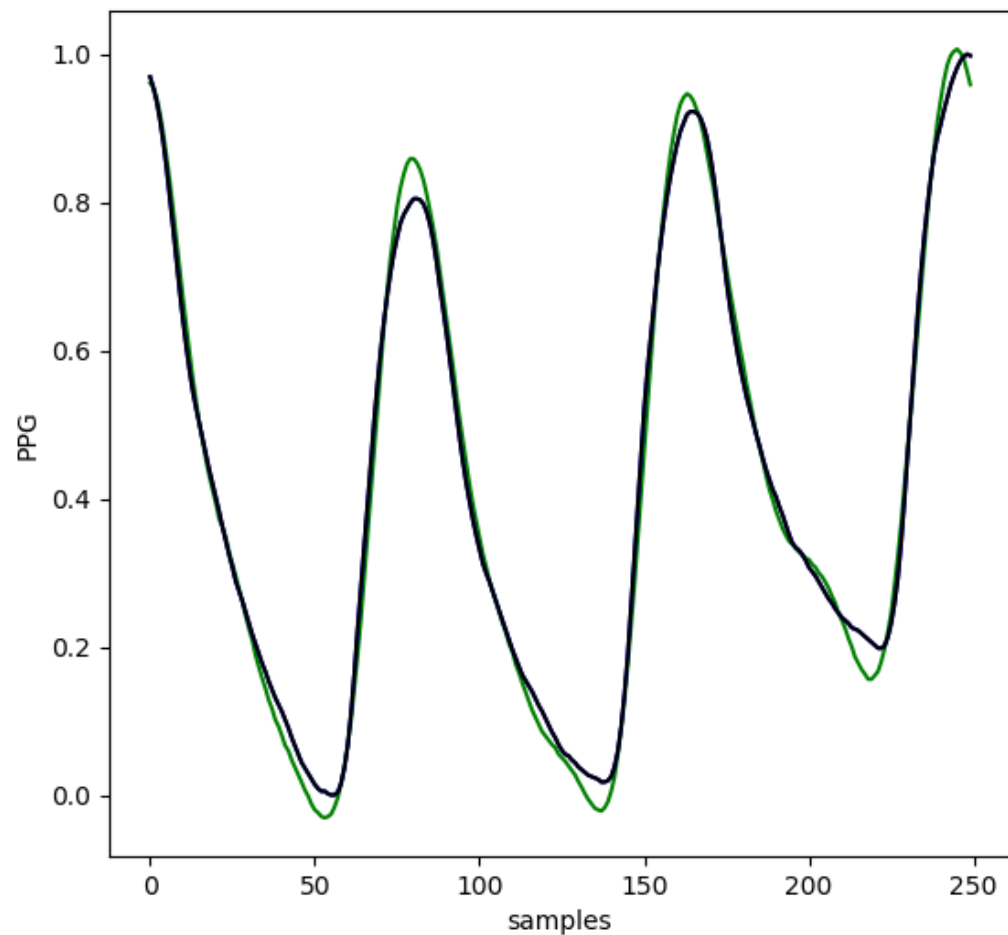
```
logit = tf.layers.dense(encoded, signal_length, activation=None,
```



5) Hidden Layer 변경

기존 hidden layer

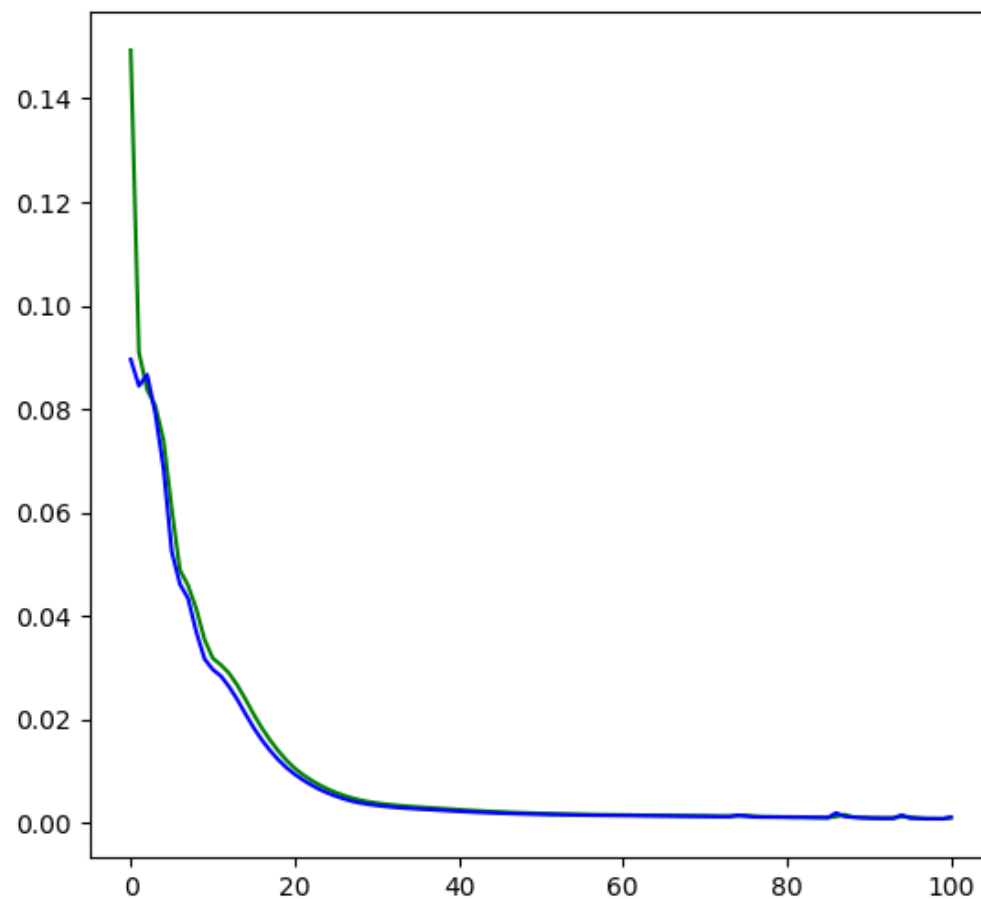
```
encode = tf.layers.dense(x, num_hidden_nodes, activation=tf.nn.sigmoid,
```



6) Hidden Layer 구조 변경

기존 hidden layer

```
encode = tf.layers.dense(x, num_hidden_nodes, activation=tf.nn.sigmoid,  
                        use_bias=True, name='encoding_layer',  
                        kernel_initializer=tf.truncated_normal_initializer(stddev=0.01),  
                        bias_initializer=tf.ones_initializer())  
encode = tf.layers.dropout(encoded, rate=probability)
```



7) 정수 데이터 autoencoding 결과 확인

np.ones 함수 사용:

np.ones((size), datatype)

예) np.ones((1,5), float) → [1.0 1.0 1.0 1.0 1.0]

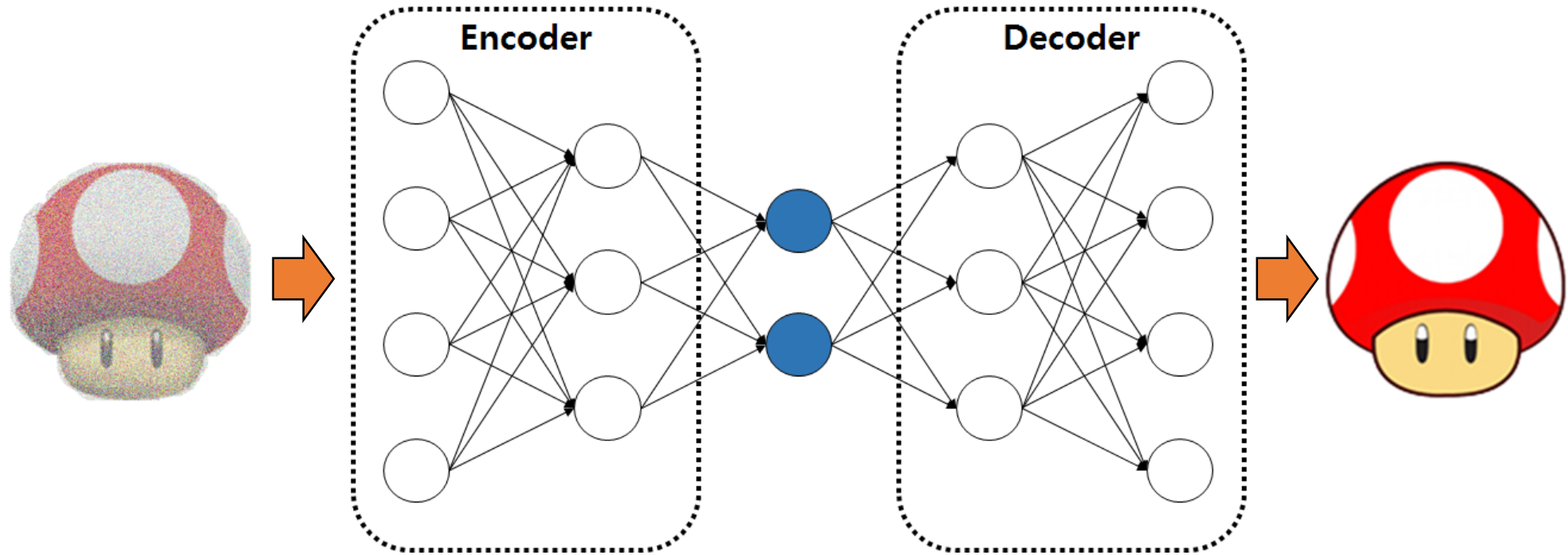
예) np.ones((2,5), float) → [[1.0 1.0 1.0 1.0 1.0] , [1.0 1.0 1.0 1.0 1.0]]

예) 0.5*np.ones((2,5), float) → [[0.5 0.5 0.5 0.5 0.5] , [0.5 0.5 0.5 0.5 0.5]]

가공된 데이터로 출력 보기:

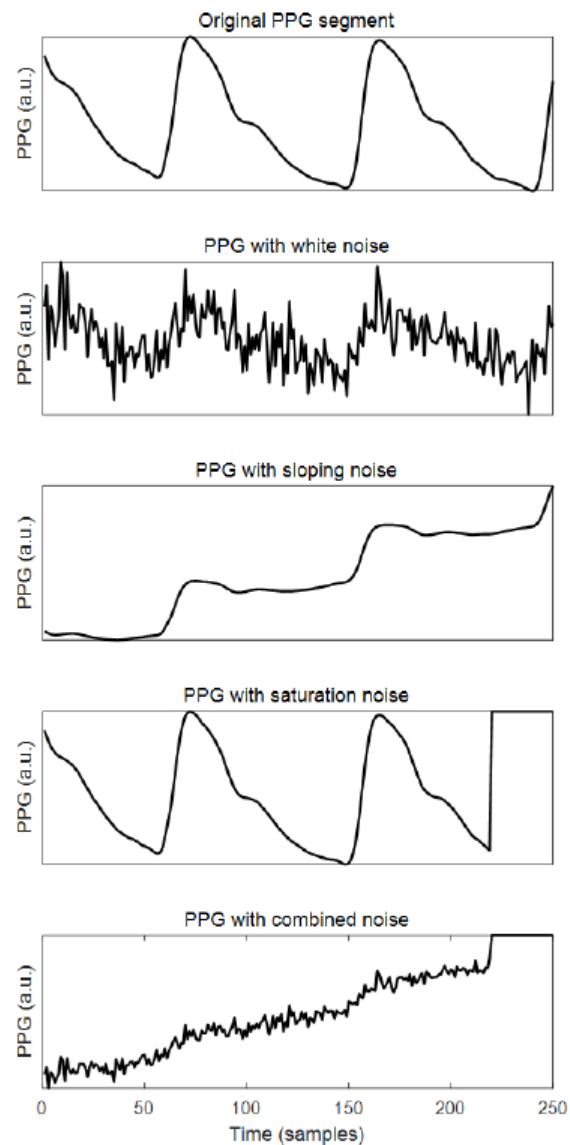
output = sess.run(prediction, feed_dict={X: 가공데이터})

Denoising Autoencoder (DAE)



- Inputs have added noise
- The compression of features in the input eliminates any spurious noise that does not belong to reconstruction of the noise-free input

Training AE for PPG Denoising



Answer

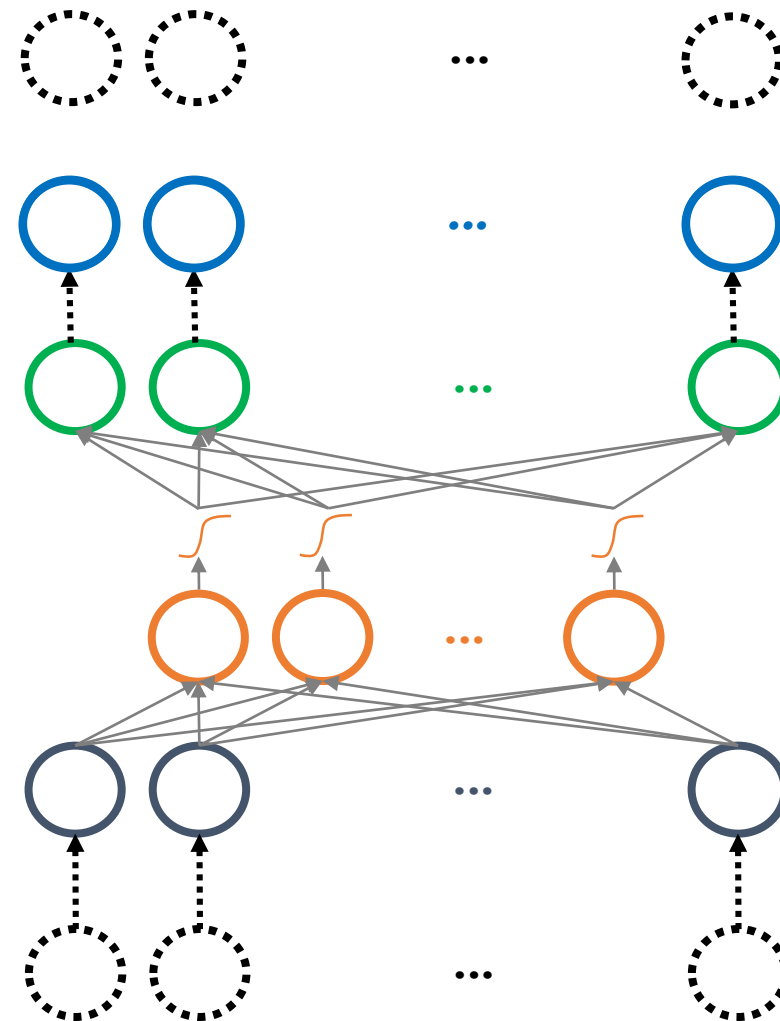
Prediction

Logit layer

Encoding layer

Input layer

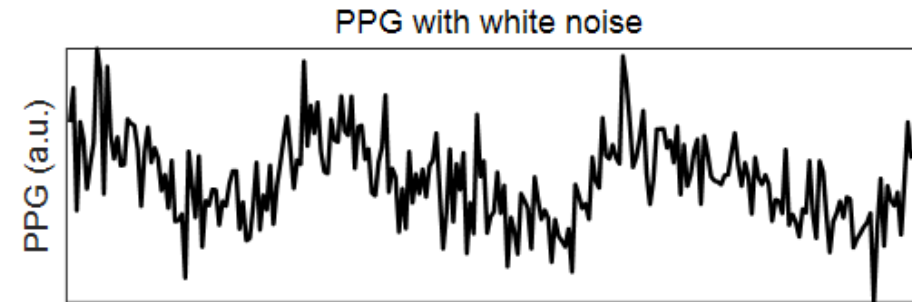
Input



Noise Generation

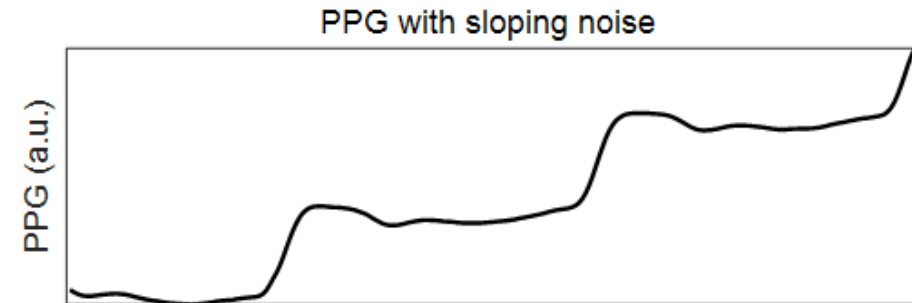
White noise:
(high frequency)

$$c_i^1 = s_i + \frac{1}{3} \cdot N(0, 1)$$



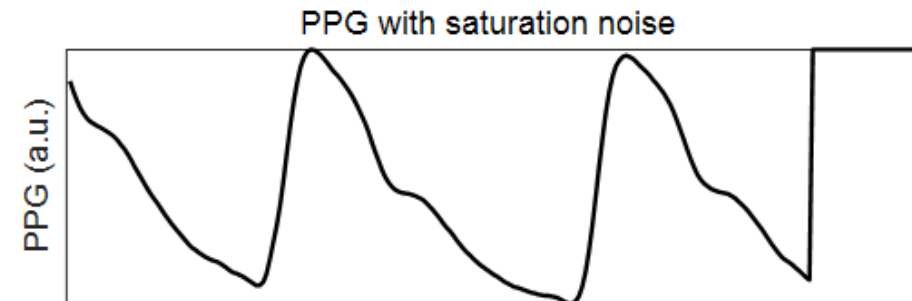
Sloping noise:
(low frequency)

$$c_i^2 = s_i + \frac{2}{250} \cdot i \cdot U(-1, 1)$$



Saturation noise:

$$c_i^3 = \begin{cases} 0 \text{ or } 1 & \text{if } x_1 < i \leq x_2 \\ s_i & \text{otherwise} \end{cases}$$

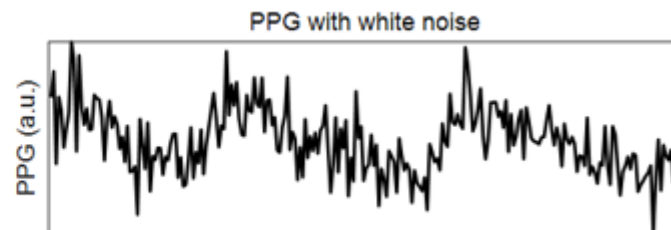


Denoising autoencoder (DAE) 실습

- ① 각각 노이즈 모델을 numpy를 사용해서 만들어보세요

White noise:
(high frequency)

$$c_i^1 = s_i + \frac{1}{3} \cdot N(0, 1)$$



Random normal 함수

- `np.random.randn` 함수 사용
- `np.random.randn(size)`

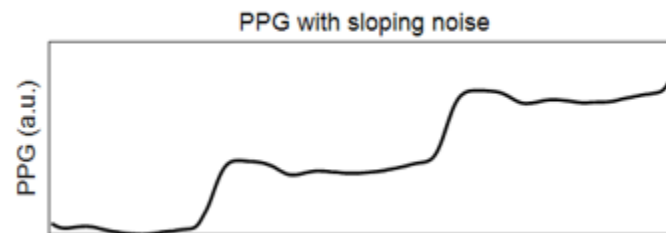
- 예) `np.random.randn(5)` → [1.42976 -0.89163182 0.47766052 -0.30909519 0.07349453]

Denoising autoencoder (DAE) 실습

① 각각 노이즈 모델을 numpy를 사용해서 만들어보세요

Sloping noise:
(low frequency)

$$c_i^2 = s_i + \frac{2}{250} \cdot i \cdot U(-1, 1)$$



Random uniform함수

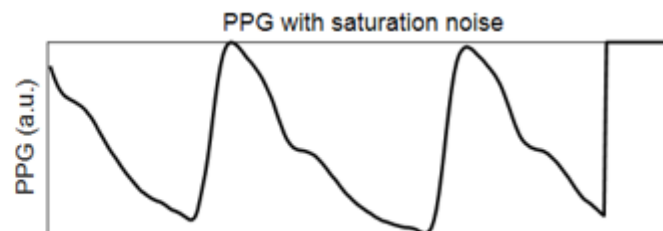
- np.random.rand 함수 사용해서 랜덤한 slope 생성
- for loop 활용해서 각 샘플의 데이터 포인트에 slope 더하기
- np.random.rand(size): 0~1사이 uniform distribution
- 예) np.random.rand(1) → [0.90283792]

Denoising autoencoder (DAE) 실습

① 각각 노이즈 모델을 numpy를 사용해서 만들어보세요

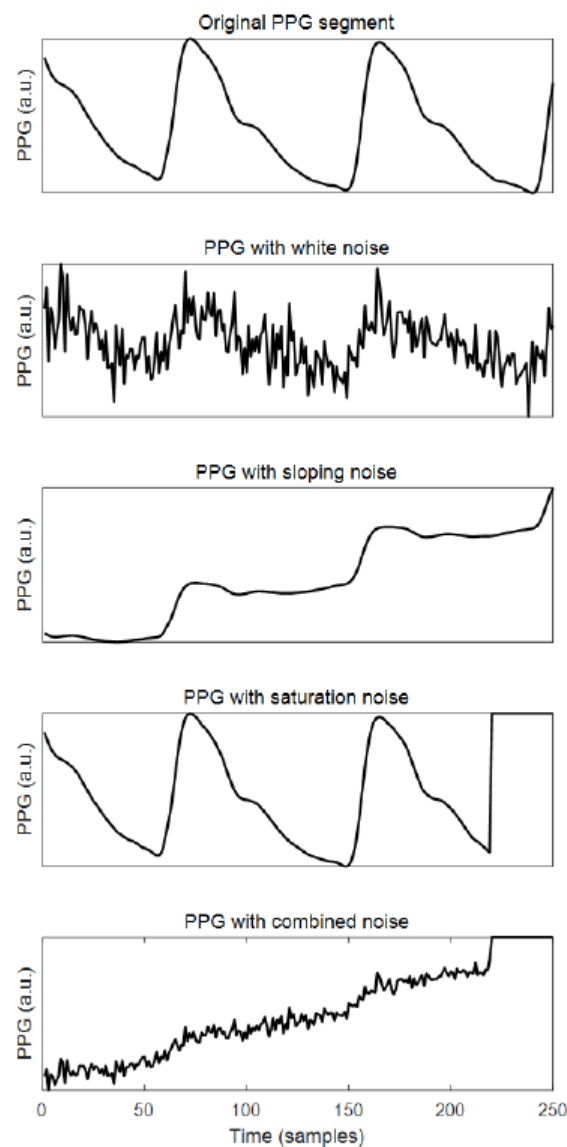
Saturation noise:

$$c_i^3 = 0 \text{ or } 1 \text{ if } x_1 < i \leq x_2 \\ c_i^3 = s_i \text{ otherwise}$$



- **np.random.rand 함수 사용해서 saturation 시작점 설정**
 - 시작점이 float인 경우 indexin을 위해 int으로 변경
 - 예) `int(102.723433)` → 102
- **np.random.rand 함수 사용해서 saturation 종료점 설정**
- **np.random.rand 함수 사용해서 saturation 값 0,1 설정**
- **np.ones 사용해서 saturation 구간의 정수 벡터 생성**

Training BRAE for PPG Denoising



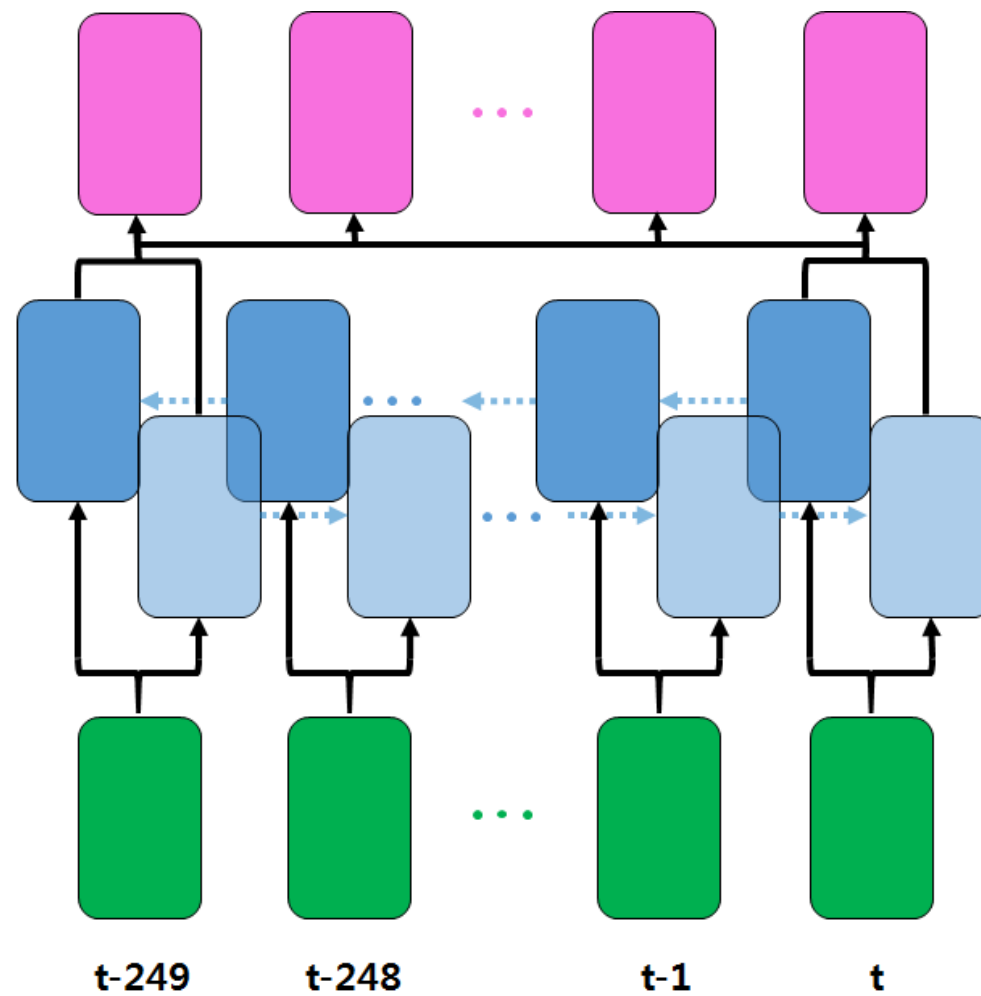
Output layer

Backward Layer

Forward Layer

Input layer

Time Steps



Tensorflow Recurrent AE

데이터와 placeholder의 dimension에
number of inputs dimension을 추가함

unstack 함수를 사용하여 time step
을 첫 dimension으로 설정해줌

LSTM cell의 구조를 설정함

Many-to-one 구조를 사용하여 맨 마지막
timestep에 output만 사용하고, 그 후에 fully
connected layer로 원 데이터의 사이즈 구축

```
print(train_input[0])
train_input = np.reshape(train_input, [len(train_input), signal_length, 1])
print(train_input[0])
val_input = np.reshape(val_input, [len(val_input), signal_length, 1])
x = tf.placeholder("float", [None, signal_length, 1])

def RAE(x, probability, num_hidden_nodes):
    # Unstack to get a list of 'timesteps' tensors of shape (batch_size, n_input)
    # before unstacking shape = batch_size, timesteps, num_input
    x = tf.unstack(tf.transpose(x, perm=[1, 0, 2]))
    # x = tf.unstack(x, signal_length, 1)
    # after unstacking shape = timesteps, batch_size, num_input

    lstm_fw_cell = rnn.LSTMCell(num_hidden_nodes, forget_bias=1.0)
    lstm_fw_cell = rnn.DropoutWrapper(cell=lstm_fw_cell, output_keep_prob=probability)

    outputs, _ = rnn.static_rnn(lstm_fw_cell, x, dtype=tf.float32)
    print(outputs.get_shape())

    # Linear activation, using rnn inner loop last output
    logit = tf.layers.dense(outputs[-1], signal_length, activation=None,
                             use_bias=True, name='output_layer',
                             kernel_initializer=tf.truncated_normal_initializer(stddev=0.01),
                             bias_initializer=tf.ones_initializer())
    print(logit.get_shape())

    return logit
```

Tensorflow Bidirectional Recurrent AE

Recurrent AE와 똑같지만 forward cell과 backward cell을 둘 다 설정해주고 rnn.static_bidirectional_rnn 함수로 bidirectional 구조 설정

```
def BRAE(x, probability, num_hidden_nodes):
    # Unstack to get a list of 'timesteps' tensors of shape (batch_size, n_input)
    # before unstacking shape = batch_size, timesteps, num_input
    x = tf.unstack(tf.transpose(x, perm=[1, 0, 2]))
    # x = tf.unstack(x, signal_length, 1)
    # after unstacking shape = timesteps, batch_size, num_input

    lstm_fw_cell = rnn.LSTMCell(num_hidden_nodes, forget_bias=1.0)
    lstm_fw_cell = rnn.DropoutWrapper(cell=lstm_fw_cell, output_keep_prob=probability)
    lstm_bw_cell = rnn.LSTMCell(num_hidden_nodes, forget_bias=1.0)
    lstm_bw_cell = rnn.DropoutWrapper(cell=lstm_bw_cell, output_keep_prob=probability)

    outputs, _, _ = rnn.static_bidirectional_rnn(lstm_fw_cell, lstm_bw_cell, x, dtype=tf.float32)
    print(outputs.get_shape())

    # Linear activation, using rnn inner loop last output
    logit = tf.layers.dense(outputs[-1], signal_length, activation=None,
                             use_bias=True, name='output_layer',
                             kernel_initializer=tf.truncated_normal_initializer(stddev=0.01),
                             bias_initializer=tf.ones_initializer())
    print(logit.get_shape())

    return logit
```

- ① RAE_practice_solution.py 파일을 변경해서 bidirectional RAE (BRAE)를 만들어 보세요
- ② RAE와 BRAE의 outputs.get_shape() 차이를 확인해보세요

실습: 학습된 RAE 모델을 활용한 PPG Denoising

기존
(without restoring checkpoint)

```
166 with tf.Session() as sess:
167     # run the initializer
168     sess.run(init)
169     # Writer
170     writer = tf.summary.FileWriter(FILEWRITER_PATH)
171     # Saver
172     saver = tf.train.Saver(max_to_keep=200)
173
174     # pick a goal for the loss value
175     old_loss = 0.001
```

학습된 모델 checkpoint 설정

```
23 os.makedirs(FILEWRITER_PATH)
24 CHECKPOINT_PATH = './RAE_tensorboard/checkpoints'
25 if not os.path.isdir(CHECKPOINT_PATH):
26     os.makedirs(CHECKPOINT_PATH)
```

학습된 모델 복원
(with restoring checkpoint)

```
166 with tf.Session() as sess:
167     # run the initializer
168     sess.run(init)
169     # Writer
170     writer = tf.summary.FileWriter(FILEWRITER_PATH)
171     # Saver
172     saver = tf.train.Saver(max_to_keep=200)
173     # restore checkpoint =====
174     latest_ckpt = tf.train.latest_checkpoint(CHECKPOINT_PATH)
175     print("loading the latest checkpoint: " + latest_ckpt)
176     saver.restore(sess, latest_ckpt)
177
178     # pick a goal for the loss value
```

실습: 학습된 모델을 활용한 PPG Denoising

- ① 학습된 RAE 모델을 복구해서 결과를 출력해보세요 (RAE_apply.py 파일 사용)
- ② 학습된 BRAE 모델을 복구하고 연장으로 학습하는 코드를 작성해보세요
 - 학습된 BRAE 모델 checkpoint 설정 필요
 - RAE 구조를 BRAE 구조로 변경 필요
- ③ 다른 데이터 (external_val1.txt, ... external_val10.txt)를 로딩해서 학습된 BRAE에 적용하는 코드를 작성해보세요