Assignment 4

Name: Zhe Lin(linz38)

001422116

Content:

Q1. 2

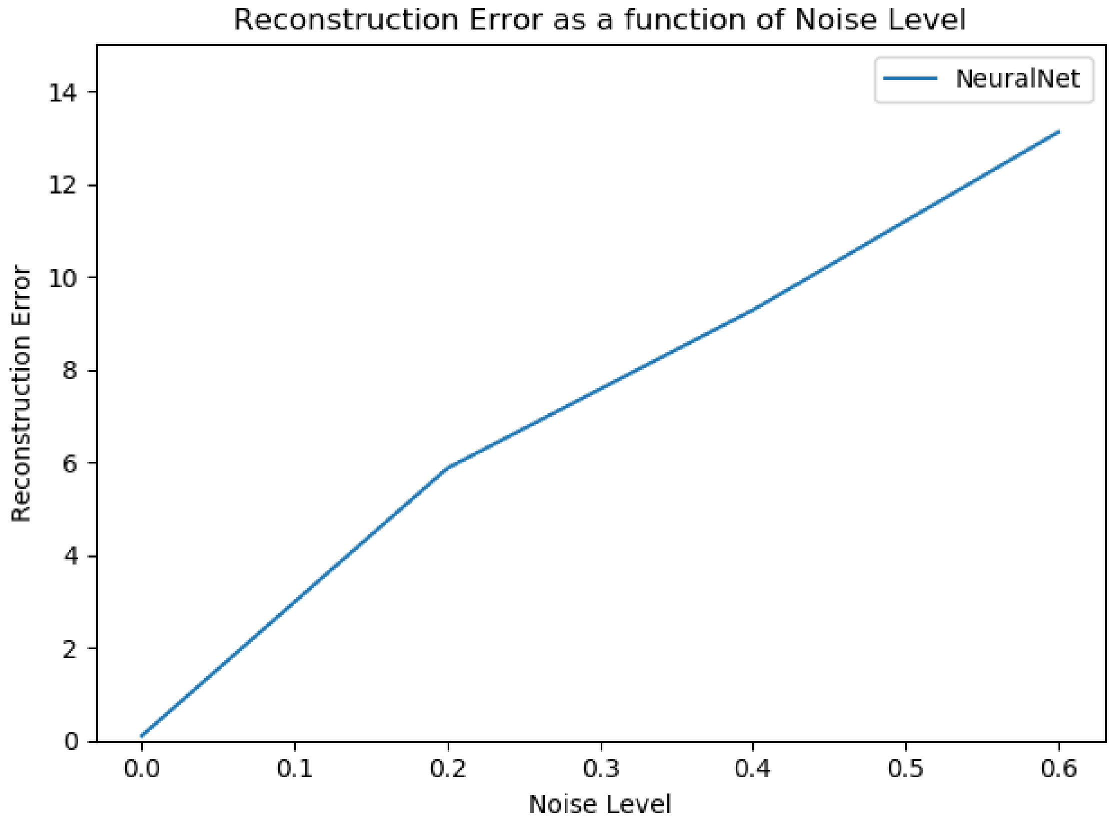
Q2. 3

Q3. 4

Q4. 5

Q5(Bonus) 7

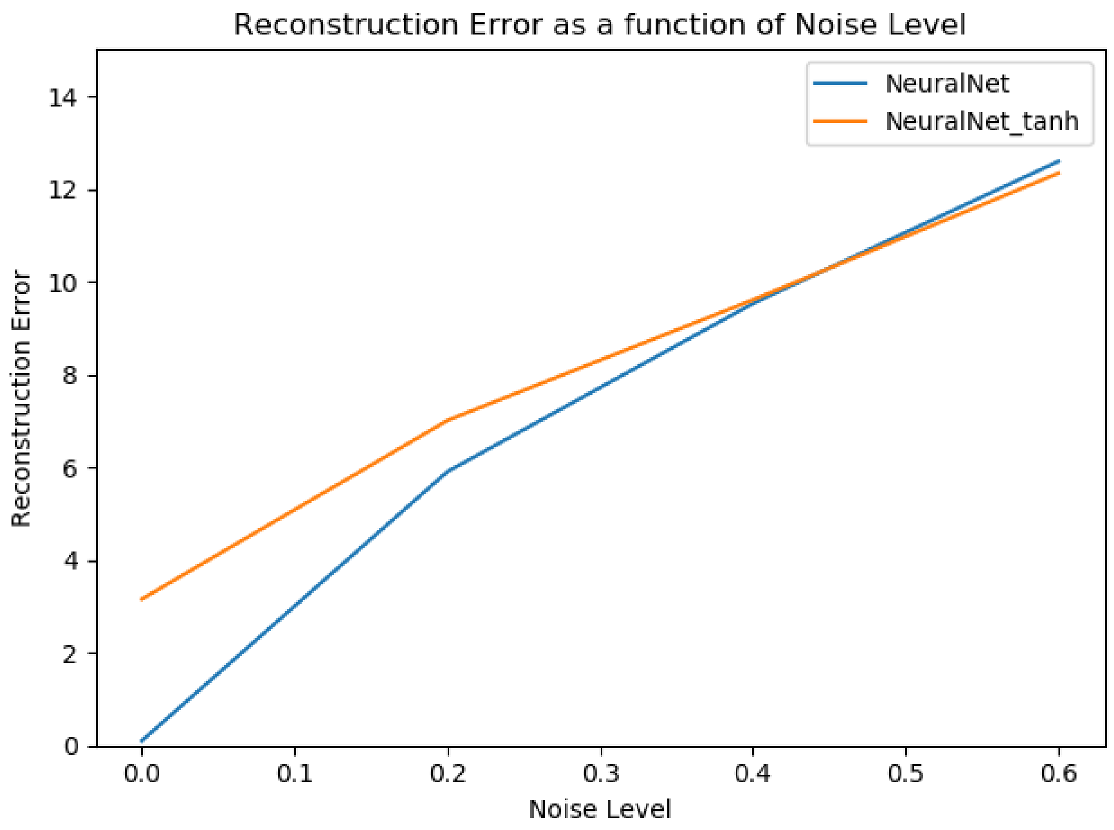
Q1.



For the denoising method, we create a neural work with MLPRegressor, which execute a multi-layer-perceptron. It training multi-layer neural network by using backpropagation, and the output doesn’t have activation function. It uses square error as loss function, then it updates the weight of output and hidden layer, respectively. It will continuous iterating and calculating with new weight. Consequently, we train the neural network and consider the noise digits as input to regress digits as output.

From previous assignment, in case of accuracy, we know that the higher score is better. However, in the loss(error) function, the lower score is better. If we want to handle the accuracy and losses in the same way, it needs return the negative. Therefore, we apply the scoring = ‘neg_mean_squared’. In the output graph, it shows that the greater is not better, which is the opposite way of accuracy and it has upward trend, it means the error is increasing with the increasing noise level. We also can see two slops for the line, it indicates the neural network by using backpropagation to improve the output by using previous output.

Q2.

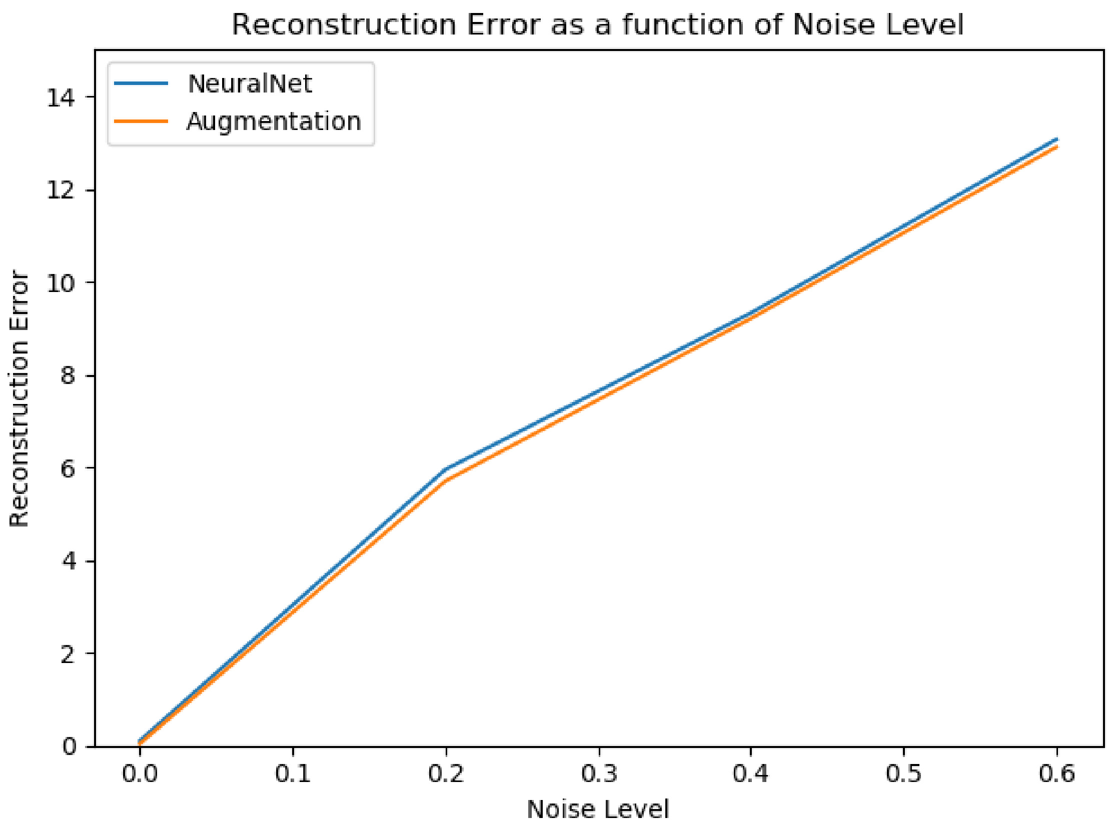


I saw our regressor which the most part of the activation function of the neural network function ‘tanh’ is above the activation function of the neural network function ‘telu’, which means the activation function of ‘relu’ is better than the activation function of ‘tanh’.

Q3.

Obviously, for activation function of ‘relu’, it works better than the activation function of ‘tanh’. Comparing with tanh, it doesn’t have exp and save more calculation so that it can greatly accelerate the convergence. The ‘relu’ makes some parts’ output of neuron is 0, it causes network sparsity, and reduces the interdependence of parameters and helps the problem of over fitting. The sparse model can better extract related features and fit training data.

Q4.



```python
# This is our MyAugmentedNeuralNetRegressor

class MyAugmentedNeuralNetRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, demo_param='demo'):
        self.demo_param = demo_param
        return

    # Make Noise function
    def fit(self, X, Y, drop_probability):
        self.X_ = X
        self.Y_ = Y

        # Create four additional of Y
        self.y1 = np.copy(Y)
        self.y2 = np.copy(Y)
        self.y3 = np.copy(Y)
        self.y4 = np.copy(Y)

        #create four additional copies of X
        self.x1 = np.copy(X)
        self.x2 = np.copy(X)
        self.x3 = np.copy(X)
        self.x4 = np.copy(X)

        # Combine y and Y
        self.combine_y = np.concatenate((Y, self.y1, self.y2, self.y3, self.y4), axis=0)

       # Combine x and X and make it noise
        self.original_x = np.concatenate((self.x1, self.x2, self.x3, self.x4), axis=0)
        self.original_x_noise = np.multiply(self.original_x, np.random.choice([0, 1], size=self.original_x.shape,
                                                                              p=[drop_probability,
                                                                                 1 - drop_probability]))
        # Combine original_x_noise and X
        self.combine_x = np.concatenate((X, self.original_x_noise), axis=0)

        self.NeuralNet = MLPRegressor(solver='adam', random_state=1, \
                                      hidden_layer_sizes=(100, 100), activation='relu', \
                                      alpha=1e-5, learning_rate_init=0.001, \
                                      max_iter=1000, early_stopping=True, validation_fraction=0.1)

        self.NeuralNet.fit(self.combine_x, self.combine_y)

        # Return the classifier
        return self

    def predict(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['X_', 'Y_'])
        # Input validation
        X = check_array(X)

        return self.NeuralNet.predict(X)
```

For our new regressor, we can see the output that is below the original approach (no data augmentation), which means that increasing the size of data, it can helps improve regressor.

Q5(Bonus)

```python
# My Bonus question

class MyBonusNeuralNetRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, demo_param='demo'):
        self.demo_param = demo_param
        return

    def fit(self, X, Y):
        self.X_ = X
        self.Y_ = Y

        self.param_grid = {'hidden_layer_sizes': [(i * 20,) * i for i in range(5, 10)]}

        # Look at https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
        # to understand each parameter. In particular, we are using two hidden layers, each having 100 neurons.
        self.NeuralNet = MLPRegressor(solver='adam', random_state=1, \
                                      hidden_layer_sizes=(100,), activation='relu', \
                                      alpha=1e-5, learning_rate_init=0.001, \
                                      max_iter=1000, early_stopping=True, validation_fraction=0.1)

        # GridSearchCV
        self.grid = GridSearchCV(estimator=self.NeuralNet, param_grid=self.param_grid,
                                 cv=10, scoring='accuracy')
        self.grid.fit(X, Y)

        self.NeuralNet = MLPRegressor(solver='adam', random_state=1, \
                                      hidden_layer_sizes=self.grid.best_params_['hidden_layer_size'], activation='relu', \
                                      alpha=1e-5, learning_rate_init=0.001, \
                                      max_iter=1000, early_stopping=True, validation_fraction=0.1)
        self.NeuralNet.fit(X, Y)

        # Return the classifier
        return self

    def predict(self, X):
        # Check is fit had been called
        check_is_fitted(self, ['X_', 'Y_'])
        # Input validation
        X = check_array(X)

        return self.NeuralNet.predict(X)
```

I’m trying to apply the GriSearchCV to search the best combination of hidden layer, then we train the MLPRegressor with the new combination of hidden layer.