Advanced Topics in Climate Dynamics
Homework 1

L. Vulto
4685784
l.vulto@students.uu.nl

May 25, 2023

(a) **Use 50%, 25% and 25% for the training, validation and testing data set and randomly select these data from the total data set. Make a scatter plot (x versus sin(x)) of these three data sets.**

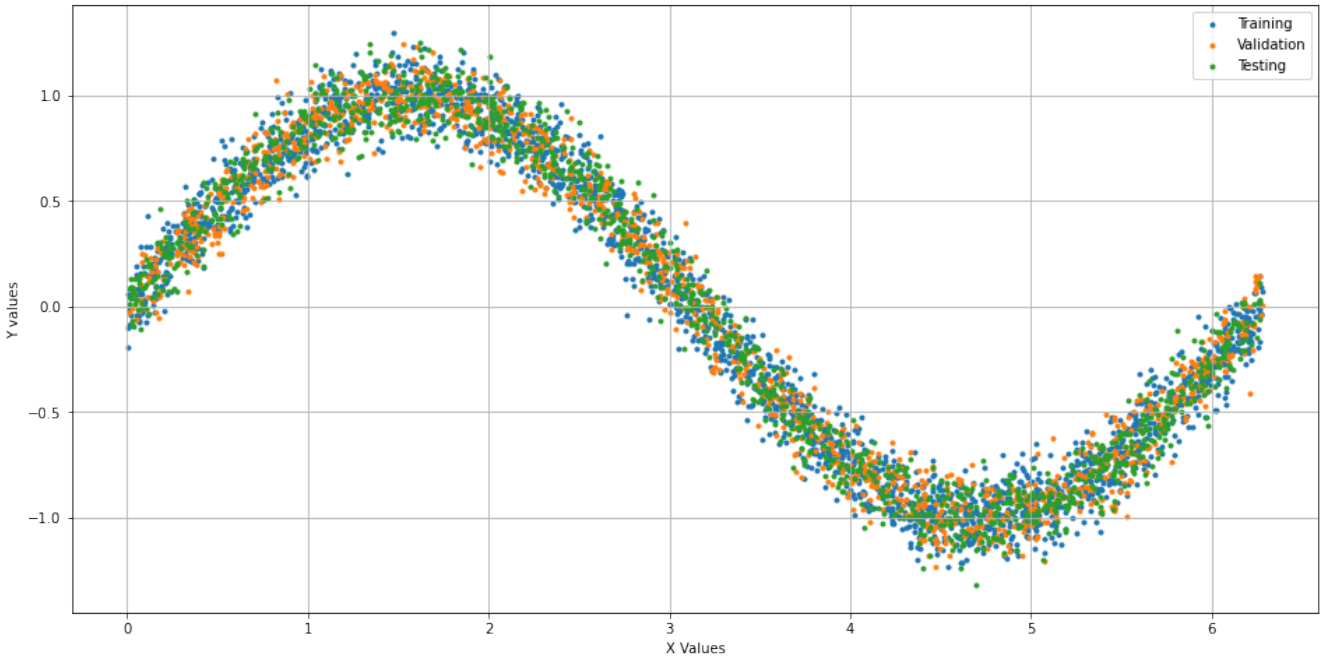

Figure 1: Training, validation and testing data, randomly selected from the total dataset

In figure 1 is the training data, testing data and validation data plotted. In order to test if the right amount of datapoints are used, the following code is used:

```
1 X_train_val, X_test, y_train_val, y_test = train_test_split(
    X_values, sin_wave, test_size=0.25, random_state=42)
2
3 X_train, X_val, y_train, y_val = train_test_split(X_train_val,
    y_train_val, test_size=(1/3), random_state=42)
4
5
6 print('Train data is size: ',np.shape(y_train)[0], 'Test data is
    size: ', np.shape(y_test)[0],'Validation data is size: ',(np.
    shape(y_val)[0]))
```

Output: Train data is size: 2500 Test data is size: 1250 Validation data is size: 1250

Thus, our data selection has been successful.

(b) **Plot the training and validation error (e.g. the mean squared error) as a function of the number of epochs.**

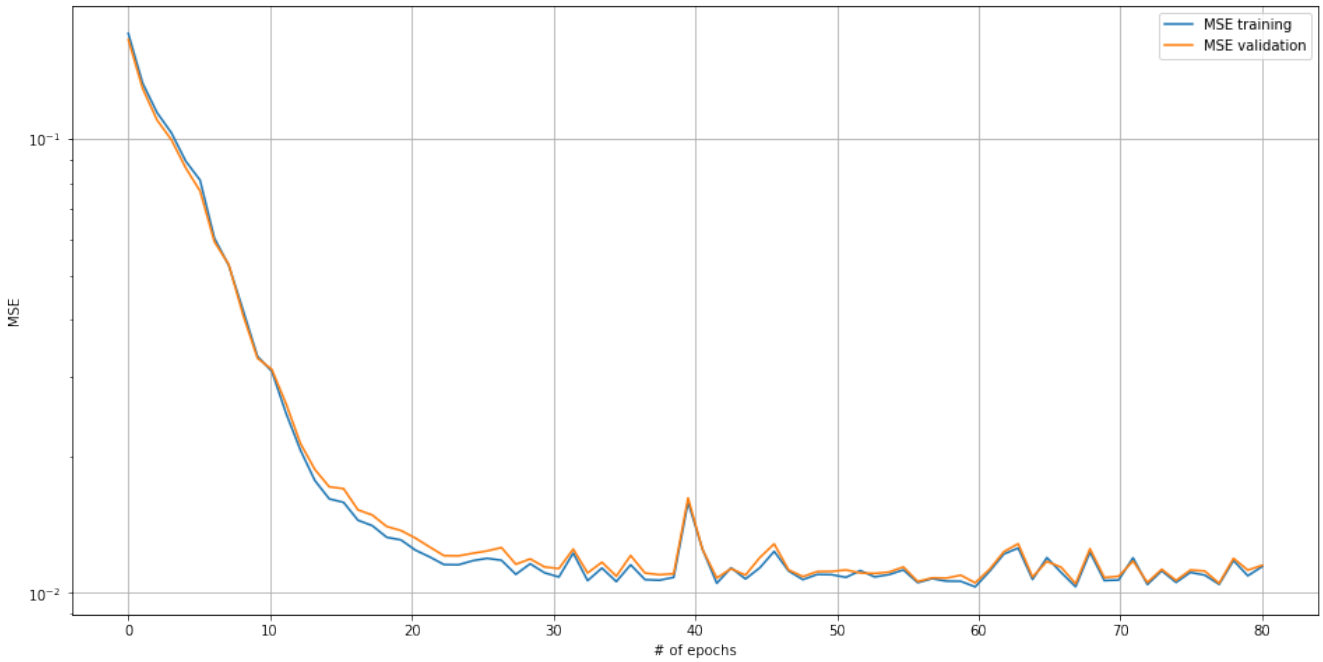

Figure 2: Number of epochs as a function of the Mean Squared Error(MSE) for training data and validation data.

The MSE drastically decreases for an increase in number of epochs as can be seen from figure 2. This decrease stagnates after approximately 30 epochs.

Furthermore, in figure 2 it can be seen that the MSE of training and validation decrease very similarly. The reason for this extreme similarity is that the data is chosen from the same (relatively) large random pool of data, namely the perturbed sine wave, and therefore it is expected to be so closely related.

If the MSE of the training and and validation are very similar this means that the model is able to make accurate predictions on both the training data and the validation data, suggesting it is generalizing well to new data.

(c) **Describe the effect of the learning rate on the behaviour of the loss function. What is the optimal value of this hyper parameter?** The learning rate is a hyperparameter of the system. The hyperparameter is a parameter that is set before the training. This is not the same as a model parameter, which are learned from the data during training. Other hyperparameters are for example the number of hidden layers. The learning rate hyperparameter controls how much the weights of the model are updated with respect to the gradient loss of the function. If we choose a too high learning rate the loss function could start to diverge, and a low learning rate could mean that the function converges very slowly, so the model will take long before it reaches a good solution. Or, if there is a local minimum, a low learning rate can also mean that the loss function cannot escape this local minimum and you end up with bad results. The learning rate that was provided was $\eta = 0.005$.

I chose a scalar between 0.01-50 with steps of 1 and multiplied this with η to find a range of η's to compare the loss function and MSE. The MSE of both the training and validation as well as the Cost function is plotted in figure 3.

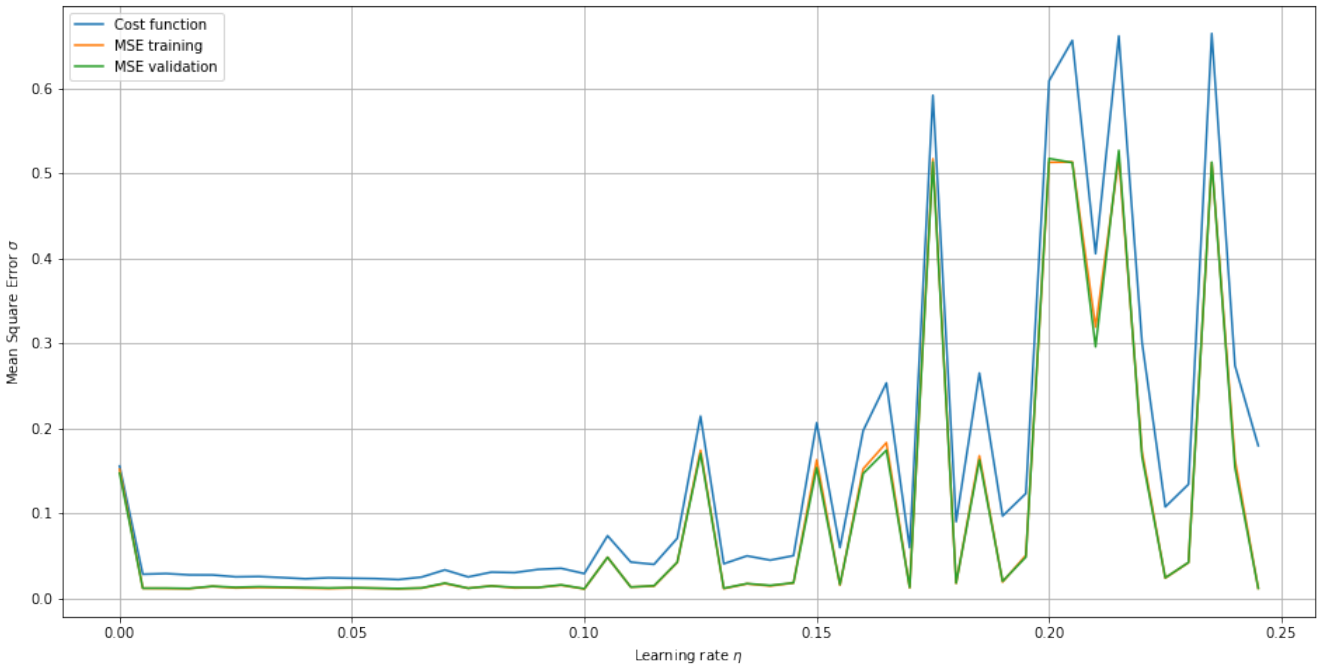

Figure 3: Learning rate η versus mean square error σ

The cost function and the MSE are very closely related because of they share the following relation:

```
cost = np.sum(np.square(output-y))/output.shape[0] + L2_term
```

The L2 term thus explains the linear shift between the cost function and the MSE.

The reason that the MSE of both training and validation is quite high for a very small learning rate can be because there are not enough epochs for the loss function to be minimized. For a greater learning rate, the MSE also starts to rapidly increase. This can be because with this learning rate, the FNN 'jumps' over the minima. For some η it can be seen that it does find a relatively low value, but this can be explained that the algorithm randomly, or with luck, finds a local minimum.

It can be seen that the minimum of the loss function is for a relatively small η. Therefore I zoomed in in the range of $\eta = 0$ to $\eta = 0.10$ with 200 steps. In fig 4 with a new scalar between 0.01 and 10 with 200 steps.

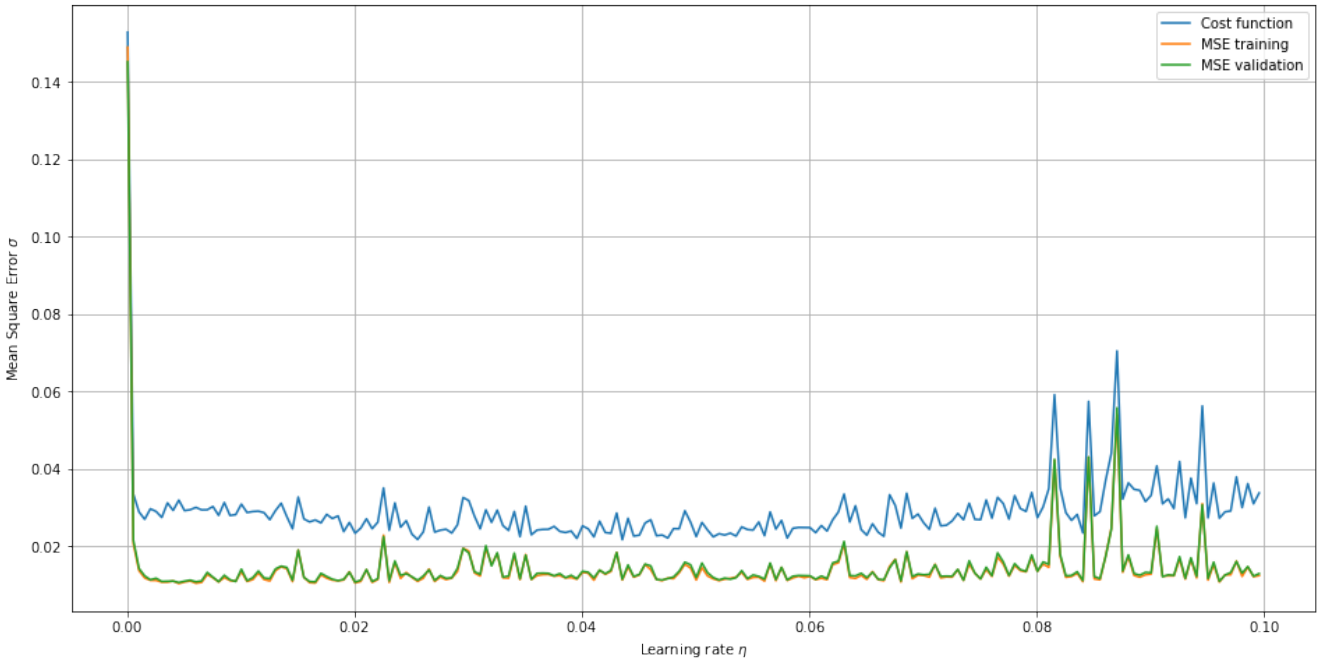

Figure 4: Learning rate η versus mean square error σ , smaller domain

This gave a minimum in the MSE at $\eta = 0.00455$. Here I looked at the minimum for the validation, because validation data is used to evaluate the performance of the model during training. The validation data is a separate dataset from the training data and is not used to update the model's weights. Instead, it's used to provide an unbiased estimate of the model's performance on unseen data.

(d) **Determine the performance of the FNN on the test data. What is the accuracy of the predictions for the chosen hyper parameters?** If we use the self.eval() function for this learning rate η we find a mean squared error of $\sigma = 0.01069$. For the reasons stated above, this is also the MSE of the validation data. This is quite a small error, since our data variances between 0 and $|1.25|$ roughly. The error imposed by the random perturbations in the beginning can be calculated, to check how well the model does. This, since the randomness was used using python's ramdom.rand module which draws numbers from a Gaussian distribution. This prediction limit to the mean squared error is very close to 0.01, and therefore our best estimate for σ is very close to the prediction limit .

The predicted Y values are plotted in the original dataset in figure 5.

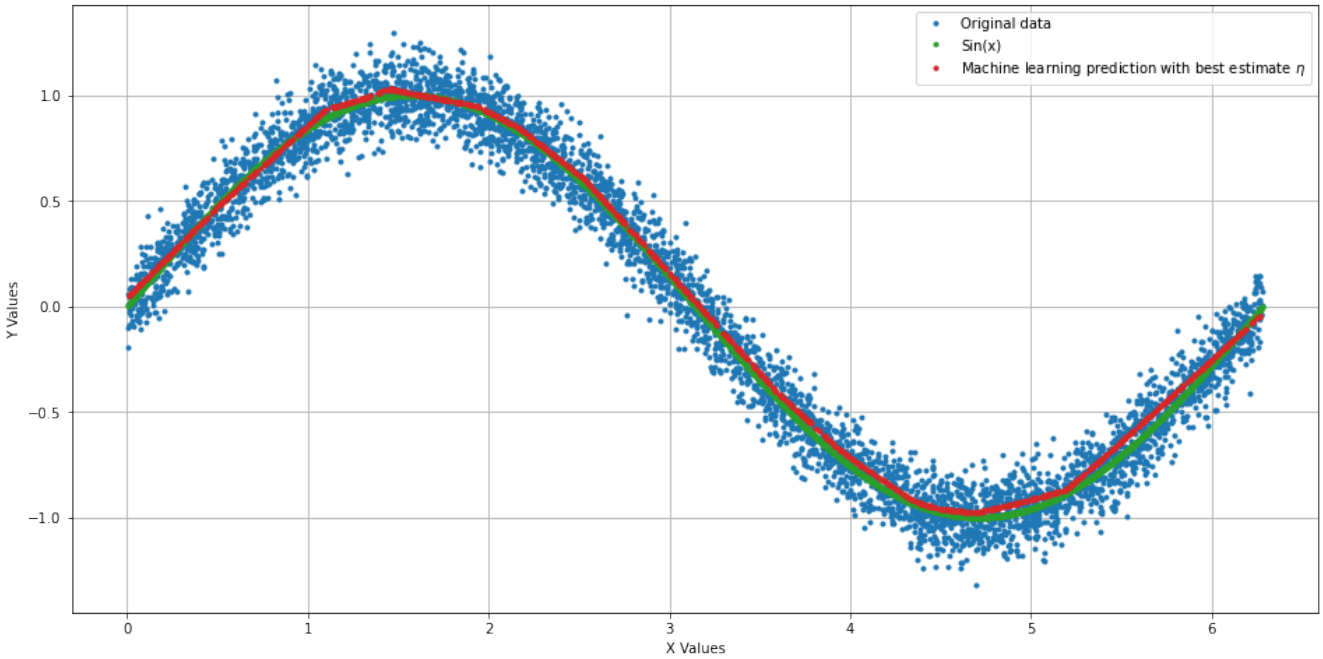

Figure 5: Predicted y values from the FNN with $\eta = 0.0045$

It can be seen that the predicted values very closely resemble a perfect sinusoidal wave.

Furthermore, we also will use the R squared value, or the coefficient of determination. This is a statistical measure that represents the proportion of the variance in the dependent variable that is predictable from the independent variable. In other words, it also measures how well the model fits the data.

The R-squared value ranges from 0 to 1, with higher values indicating a better fit. A value of 1 means that the model can perfectly explain all the variability in the data, while a value of 0 means that the model cannot explain any of the variability.

For the optimal value of η we find a R-squared value of 0.9793. This indicates a very good fit.

It's possible for a model to have a high R-squared value and a low σ but still produce inaccurate predictions if it is over fitting the data, but with use of figure 5 we can see that this is not the case and thus a good fit.

The FNN predicts a sinusoidal wave, and it does not try to predict the randomness that we implemented in the beginning. This would be the case if the data was overfitted.

Appendix

```
n_samples=5000
x_values=np.random.uniform(low=0,high=2*np.pi,size=n_samples)
sin_wave=np.sin(x_values)+(0.1*np.random.randn(x_values.shape[0]))
plt.plot(x_values,sin_wave,'.')
plt.show()

class MLP():

    def _compute_cost(self, y, output):

        squared_weights_sum=0
        weights=np.array(self.mlp.coefs_)

        for layer_weights in weights:
            squared_weights_sum+=np.sum(layer_weights**2)

        L2_term = (self.l2 * (squared_weights_sum))

        cost = np.sum(np.square(output-y))/output.shape[0] + L2_term
        return cost


    def __init__(self,n_hidden=100,seed=2,l2 = 0.001,epochs = 80,eta
       = 0.005,shuffle = True, minibatch_size = 100):

        self.n_hidden=n_hidden
        self.seed=seed
        self.l2 = l2
        self.epochs = epochs
        self.eta = eta
        self.shuffle = shuffle
        self.minibatch_size = minibatch_size
        self.mlp=None

    def train(self,X_train,y_train,X_validation,y_validation):

        loss_curve_training=[]
        loss_curve_validation=[]

        epoch_strlen = len(str(self.epochs))
        self.eval_ = {'cost': [], 'train_mse': [], 'valid_mse': []}

        self.mlp=MLPRegressor(hidden_layer_sizes=self.n_hidden,
           activation='relu',max_iter=self.epochs,solver='adam',
```

```
                                    shuffle=self.shuffle,
                                    batch_size=self.minibatch_size,
                                       learning_rate_init=self.eta,
                                       verbose=False,alpha=self.l2,
                                    n_iter_no_change=self.epochs)

        for i in range(self.epochs):

            self.mlp.partial_fit(X_train,y_train)
            y_pred_training=self.predict(X_train)
            y_pred_validation=self.predict(X_validation)
            mse_training=np.mean(np.square(y_train-y_pred_training))
            mse_validation=np.mean(np.square(y_validation-
               y_pred_validation))

            loss_curve_training.append(mse_training)
            loss_curve_validation.append(mse_validation)

            cost = self._compute_cost(y=y_train,
                                      output=y_pred_training)


            sys.stderr.write('\r%0*d/%d | Cost: %.2f '
                             '| Train/Valid MSE.: %.2f/%.2f ' %
                            (epoch_strlen, i+1, self.epochs, cost,
                             mse_training, mse_validation))
            sys.stderr.flush()

            self.eval_['cost'].append(cost)
            self.eval_['train_mse'].append(mse_training)
            self.eval_['valid_mse'].append(mse_validation)


    def evaluate(self,X,y):

        y_pred=self.predict(X)
        mse=np.mean(np.square(y-y_pred))
        print("mean squared error:{}".format(mse))

    def predict(self,X):

        if(self.mlp==None):
            print("model not trained")
        else:
            predictions=self.mlp.predict(X)

        return predictions
```

```
87
88      def printer(self):
89          return self.eval_['train_mse'],self.eval_['valid_mse']
90
91      def printer_train(self):
92          return self.eval_['cost']
93
94
95
96  import numpy as np
97  from sklearn.model_selection import train_test_split
98
99
100 X_values = x_values.reshape(-1, 1)
101 # sin_wave = sin_wave.reshape(-1, 1)
102
103
104 X_train_val, X_test, y_train_val, y_test = train_test_split(X_values,
        sin_wave, test_size=0.25, random_state=42)
105 X_train, X_val, y_train, y_val = train_test_split(X_train_val,
        y_train_val, test_size=(1/3), random_state=42)
106
107
108 mlp = MLP()
109 mlp.train(X_train, y_train, X_val, y_val)
110 mlp.evaluate(X_test, y_test)
111
112 plt.figure(figsize=(16,8))
113 plt.scatter(X_train, y_train, label='Training', marker='.')
114 plt.scatter(X_val, y_val, label='Validation', marker='.')
115 plt.scatter(X_test, y_test, label='Testing', marker='.')
116 plt.xlabel('X Values')
117 plt.ylabel('Amplitude')
118 plt.grid()
119 # plt.title('Scatter Plot of X versus sin(X)')
120 plt.legend()
121 plt.show()
122
123 print('Train data is size: ',np.shape(y_train)[0], 'Test data is size
        : ', np.shape(y_test)[0],'Validation data is size: ',(np.shape(
        y_val)[0]))
124
125 msetrain, mseval = mlp.printer()
126 X_MSE = np.linspace(0,80,80)
127 plt.figure(figsize=(16,8))
128 plt.plot(X_MSE,msetrain, label='MSE training')
129 plt.plot(X_MSE,mseval, label='MSE validation')
```

```
130 plt.xlabel('# of epochs')
131 plt.ylabel('MSE')
132 plt.yscale('log')
133 plt.legend()
134 plt.grid()
135
136 eta = 0.005
137 scalar = np.arange(0.01,50,1)
138
139 def opti(scalar):
140     mse_optim = []
141     mse_train = []
142     mse_valid = []
143     for idx, val in enumerate(scalar):
144         print(val*eta)
145         mlp_i = MLP(eta = eta*val)
146         mlp_i.train(X_train, y_train, X_val, y_val)
147         cost = mlp_i.printer_train()
148         mse_training, mse_costing = mlp_i.printer()
149         last_cost = cost[-1]
150         last_mse_train = mse_training[-1]
151         last_mse_valid = mse_costing[-1]
152         mse_optim.append(last_cost)
153         mse_train.append(last_mse_train)
154         mse_valid.append(last_mse_valid)
155     return mse_optim, mse_train, mse_valid
156
157 mse_optim, mse_train, mse_valid = opti(scalar)
158
159 plt.figure(figsize=(16,8))
160 plt.plot(eta*scalar, mse_optim, label='Cost function')
161 plt.plot(eta*scalar, mse_train, label='MSE training')
162 plt.plot(eta*scalar, mse_valid, label='MSE validation')
163 plt.xlabel('Learning rate $\eta$')
164 plt.ylabel('Mean Square Error $\sigma$')
165 plt.legend()
166 plt.grid()
167 print('minimum at eta = ', eta*scalar[np.argmin(mse_optim)], 'here
      mse is', mse_train[np.argmin(mse_optim)])
168
169
170 X_train_val, X_test, y_train_val, y_test = train_test_split(X_values,
       sin_wave, test_size=0.25, random_state=42)
171 X_train, X_val, y_train, y_val = train_test_split(X_train_val,
      y_train_val, test_size=(1/3), random_state=42)
172
173 sca2 = np.arange(0.01, 20, 0.1)
```

```
174
175 cost, mtr, mvl = opti(sca2)
176
177 plt.figure(figsize=(16,8))
178 plt.plot(eta*sca2, cost, label='Cost function')
179 plt.plot(eta*sca2, mtr, label='MSE training')
180 plt.plot(eta*sca2, mvl, label='MSE validation')
181 plt.xlabel('Learning rate $\eta$')
182 plt.ylabel('Mean Square Error $\sigma$')
183 plt.legend()
184 plt.grid()
185 print('minimum at eta = ', eta*sca2[np.argmin(mtr)], 'here mse is',
      mtr[np.argmin(mtr)])
186 print('minimum at eta = ', eta*sca2[np.argmin(mvl)], 'here mse is',
      mvl[np.argmin(mvl)])
187
188
189
190
191 from sklearn.metrics import r2_score
192
193
194 y_pred = mlpopt.predict(X_test)
195 r2 = r2_score(y_test, y_pred)
196
197 print(f'R-squared: {r2:.4f}')
```