UDACITY

‹ Return to "Deep Learning" in the classroom

DISCUSS ON STUDENT HUB

Predicting Bike-Sharing Patterns

REVIEW

CODE REVIEW 2

HISTORY

▼ my_answers.py 2

```
import numpy as np


class NeuralNetwork(object):
    def __init__(self, input_nodes, hidden_nodes, output_nodes, learning_rate):
        # Set number of nodes in input, hidden and output layers.
        self.input_nodes = input_nodes
        self.hidden_nodes = hidden_nodes
        self.output_nodes = output_nodes

        # Initialize weights
        self.weights_input_to_hidden = np.random.normal(0.0, self.input_nodes**-0.
                                       (self.input_nodes, self.hidden_nodes))

        self.weights_hidden_to_output = np.random.normal(0.0, self.hidden_nodes**-
                                       (self.hidden_nodes, self.output_nodes))
        self.lr = learning_rate

        #### TODO: Set self.activation_function to your implemented sigmoid functi
        #
        # Note: in Python, you can define a function with a lambda expression,
        # as shown below.
        self.activation_function = lambda x : 1/(1+np.exp(-x))  # Replace 0 with y

```

AWESOME

Excellent use of lambda functions.

```
        ### If the lambda code above is not something you're familiar with,
        # You can uncomment out the following three lines and put your
        # implementation there instead.
        #
        #def sigmoid(x):
            #return 0  # Replace 0 with your sigmoid calculation here
        #self.activation_function = sigmoid


    def train(self, features, targets):
        ''' Train the network on batch of features and targets.

            Arguments
            ---------

            features: 2D array, each row is one data record, each column is a feat
            targets: 1D array of target values

        '''
        n_records = features.shape[0]
        delta_weights_i_h = np.zeros(self.weights_input_to_hidden.shape)
        delta_weights_h_o = np.zeros(self.weights_hidden_to_output.shape)
        for X, y in zip(features, targets):

            final_outputs, hidden_outputs = self.forward_pass_train(X)  # Implemen
            # Implement the backproagation function below
            delta_weights_i_h, delta_weights_h_o = self.backpropagation(final_outp
                                                                        delta_weig
        self.update_weights(delta_weights_i_h, delta_weights_h_o, n_records)


    def forward_pass_train(self, X):
        ''' Implement forward pass here

            Arguments
            ---------
            X: features batch

        '''
        #### Implement the forward pass here ####
        ### Forward pass ###
        # TODO: Hidden layer - Replace these values with your calculations.
        hidden_inputs = np.dot(X, self.weights_input_to_hidden ) # signals into hi
        hidden_outputs = self.activation_function(hidden_inputs) # signals from hi

        # TODO: Output layer - Replace these values with your calculations.
        final_inputs = np.dot(hidden_outputs, self.weights_hidden_to_output ) # si
        final_outputs = final_inputs # signals from final output layer

        return final_outputs, hidden_outputs

    def backpropagation(self, final_outputs, hidden_outputs, X, y, delta_weights_i
        ''' Implement backpropagation

            Arguments
            ---------
            final_outputs: output from forward pass
            y: target (i.e. label) batch
            delta_weights_i_h: change in weights from input to hidden layers
            delta_weights_h_o: change in weights from hidden to output layers

        '''
        #### Implement the backward pass here ####
        ### Backward pass ###
```

```
89
90          # TODO: Output error - Replace this value with your calculations.
91          error = y - final_outputs # Output layer error is the difference between d
92
93          # TODO: Calculate the hidden layer's contribution to the error
94          hidden_error = np.dot(self.weights_hidden_to_output, error )
95
96          # TODO: Backpropagated error terms - Replace these values with your calcul
97          output_error_term = error
98
99          hidden_error_term = hidden_error * hidden_outputs * (1-hidden_outputs)
100
101         # Weight step (input to hidden)
102         delta_weights_i_h += hidden_error_term * X[:, None]
103         # Weight step (hidden to output)
104         delta_weights_h_o += output_error_term * hidden_outputs[:, None]
105         return delta_weights_i_h, delta_weights_h_o
106
107     def update_weights(self, delta_weights_i_h, delta_weights_h_o, n_records):
108         ''' Update weights on gradient descent step
109
110             Arguments
111             ---------
112             delta_weights_i_h: change in weights from input to hidden layers
113             delta_weights_h_o: change in weights from hidden to output layers
114             n_records: number of records
115
116         '''
117         self.weights_hidden_to_output += self.lr * delta_weights_h_o / n_records #
118         self.weights_input_to_hidden += self.lr * delta_weights_i_h / n_records #
119
120     def run(self, features):
121         ''' Run a forward pass through the network with input features
122
123             Arguments
124             ---------
125             features: 1D array of feature values
126         '''
127
128         #### Implement the forward pass here ####
129         # TODO: Hidden layer - replace these values with the appropriate calculati
130         hidden_inputs = np.dot(features, self.weights_input_to_hidden) # signals i
131         hidden_outputs = self.activation_function(hidden_inputs) # signals from hi
132
133         # TODO: Output layer - Replace these values with the appropriate calculati
134         final_inputs = np.dot(hidden_outputs, self.weights_hidden_to_output) # sig
135         final_outputs =final_inputs # signals from final output layer
136
137         return final_outputs
138
139
140 ##########################################################
141 # Set your hyperparameters here
142 ##########################################################
143 iterations = 5000
144 learning_rate = 0.5
145 hidden_nodes = 25
146 output_nodes = 1
147
148
```

AWESOME

> Awesome submission!

RETURN TO PATH