UDACITY

Logout

‹ Return to Classroom

DISCUSS ON STUDENT HUB

Predicting Bike-Sharing Patterns

REVIEW | CODE REVIEW 6 | HISTORY

▼ my_answers.py 6

```
1  import numpy as np
2
3
4  class NeuralNetwork(object):
5      def __init__(self, input_nodes, hidden_nodes, output_nodes, learning_rate):
6          # Set number of nodes in input, hidden and output layers.
7          self.input_nodes = input_nodes
8          self.hidden_nodes = hidden_nodes
9          self.output_nodes = output_nodes
10
11         # Initialize weights
12         self.weights_input_to_hidden = np.random.normal(0.0, self.input_nodes**-0.5,
                                          (self.input_nodes, self.hidden_nodes))
```

Rate this review

START

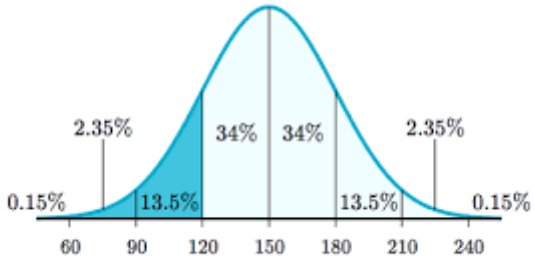
```
14
15          self.weights_hidden_to_output = np.random.normal(0.0, self.hidden_nodes**-0.5,
16                                         (self.hidden_nodes, self.output_nodes))
17
```

AWESOME

Did you notice that we are initializing weights in normal form using np.random.normal()?

This helps to initialize weights in the normal distribution. There are many other weight initialization techniques like Xavie etc.

You can learn more about these by going through this blogpost: https://towardsdatascience.com/weight-initialization-tec




```
18          self.lr = learning_rate
19
20          #### TODO: Set self.activation_function to your implemented sigmoid function ####
21          #
22          # Note: in Python, you can define a function with a lambda expression,
23          # as shown below.
24          self.activation_function = lambda x : 1/(1+np.exp(-x))  # Replace 0 with your sigmoid calc
25
```

Rate this review

ART

AWESOME

Implementation of the activation function and its derivation are all correct.

I encourage you to check the scipy library's expit() method: https://docs.scipy.org/doc/scipy/reference/generated/scipy.s

I encourage you to check the scipy library's expit() method: https://docs.scipy.org/doc/scipy/reference/generated/scipy.s

```
        ### 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 feature
            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)  # Implement the forward pa

            # Implement the backproagation function below
            delta_weights_i_h, delta_weights_h_o = self.backpropagation(final_outputs, hidden_outp
                                                                        delta_weights_i_h, delta_w
        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 ###
```

Rate this review

START

```
        # TODO: Hidden layer - Replace these values with your calculations.
        # From Lesson2, Multilayer Perceptrons
        hidden_inputs = np.dot(X,self.weights_input_to_hidden) # signals into hidden layer
        hidden_outputs = self.activation_function(hidden_inputs)  # signals from hidden layer


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

AWESOME

The implementation of forward propagation is all correct. It's returning the hidden outputs and final outputs rightly.

To multiply the arrays, you can also use the @ operator. It was introduced in Python 3 itself: https://www.python.org/dev

```
        return final_outputs, hidden_outputs

    def backpropagation(self, final_outputs, hidden_outputs, X, y, delta_weights_i_h, delta_weight
        ''' 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 ###


       #GN: from Lesson2, implementing Gradient Descent

        # TODO: Output error - Replace this value with your calculations.

        error = y - final_outputs  # Output layer error is the difference between desired target a
        # GN: error is np array

        # TODO: Backpropagated error terms - Replace these values with your calculations.
```

Rate this review

START

```
        output_error_term = error * 1

        # TODO: Calculate the hidden layer's contribution to the error
        hidden_error =  np.dot(self.weights_hidden_to_output,error)
        hidden_error_term = hidden_error * hidden_outputs * (1-hidden_outputs)

        # Weight step (hidden to output)
        delta_weights_h_o += output_error_term *  hidden_outputs[:,None]

        # Weight step (input to hidden)
        delta_weights_i_h += np.dot(X[:,None],hidden_error_term.reshape(1,self.hidden_nodes))
```

AWESOME

The errors, error terms and delta weight update steps are well coded. This helps to update the weights to improve the p

```
        return delta_weights_i_h, delta_weights_h_o

    def update_weights(self, delta_weights_i_h, delta_weights_h_o, n_records):
        ''' Update weights on gradient descent step

            Arguments
            ---------
            delta_weights_i_h: change in weights from input to hidden layers
            delta_weights_h_o: change in weights from hidden to output layers
            n_records: number of records

        '''
        self.weights_hidden_to_output += self.lr * delta_weights_h_o / n_records # update hidden-t
        self.weights_input_to_hidden += self.lr * delta_weights_i_h / n_records # update input-to-
```

AWESOME

Implementation of update_weights() method is all correct. This is helping the losses to converge and let the model perfo

is review

ART

```
    def run(self, features):

        ''' Run a forward pass through the network with input features
```

```
132
133             Arguments
134             ---------
135             features: 1D array of feature values
136         '''
137
138         #### Implement the forward pass here ####
139         # TODO: Hidden layer - replace these values with the appropriate calculations.
140         hidden_inputs = np.dot(features,self.weights_input_to_hidden) # signals into hidden layer
141         hidden_outputs = self.activation_function(hidden_inputs) # signals from hidden layer
142
143         # TODO: Output layer - Replace these values with the appropriate calculations.
144         final_inputs = np.dot(hidden_outputs, self.weights_hidden_to_output)  # signals into final
145         final_outputs = final_inputs # signals from final output layer
146
147         return final_outputs
148
149
150 ##########################################################
151 # Set your hyperparameters here
152 ##########################################################
153 iterations = 7000
154 learning_rate = 0.6
155 hidden_nodes = 25
```

AWESOME

Hyperparameter values are chosen perfectly. These are helping the model to converge and train in a reasonable amoun

I encourage you to further read about the usage of adaptive learning rate approach: https://machinelearningmastery.co networks/

```
156 output_nodes = 1
157
```

is review

START

Rate this review

START