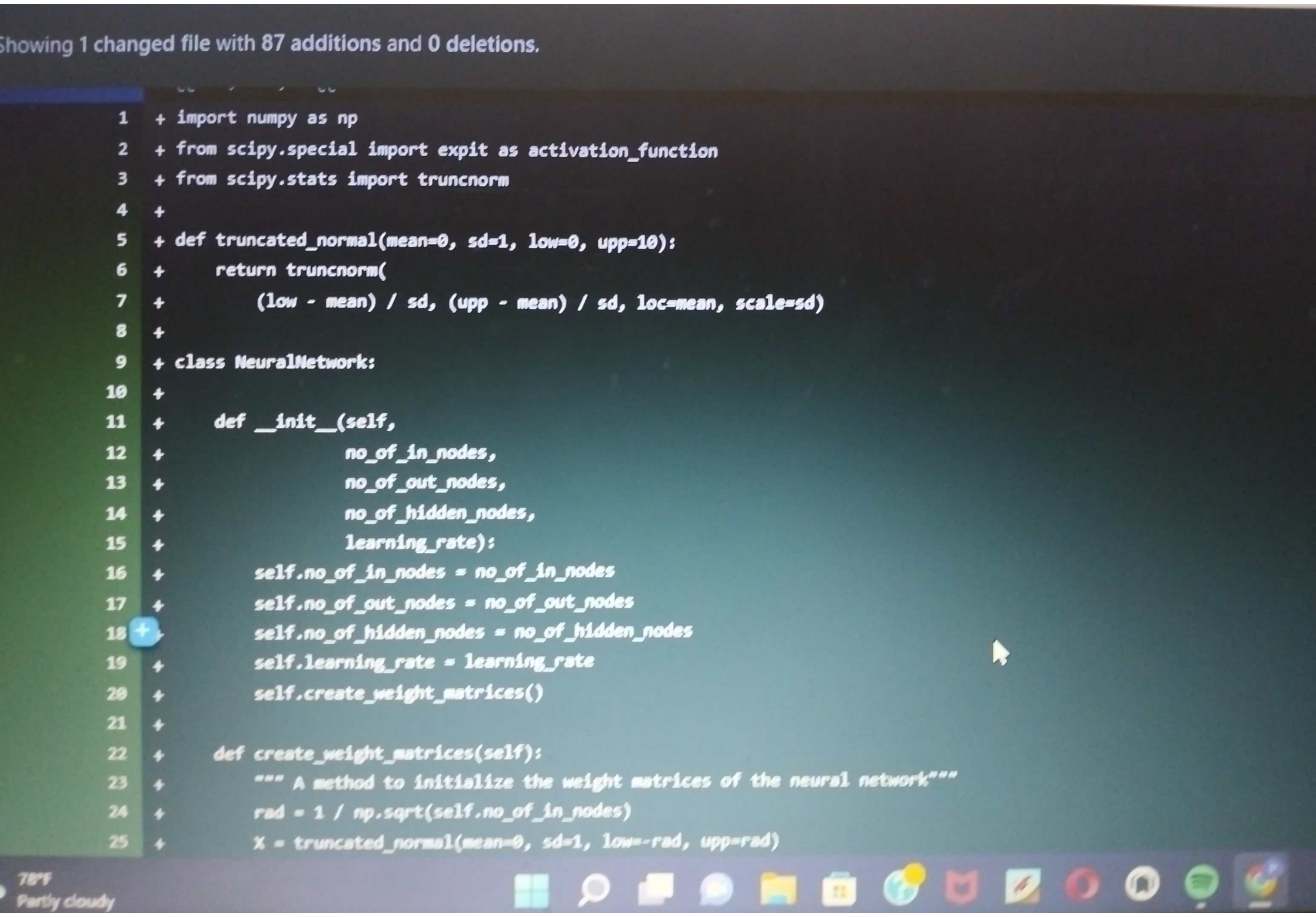
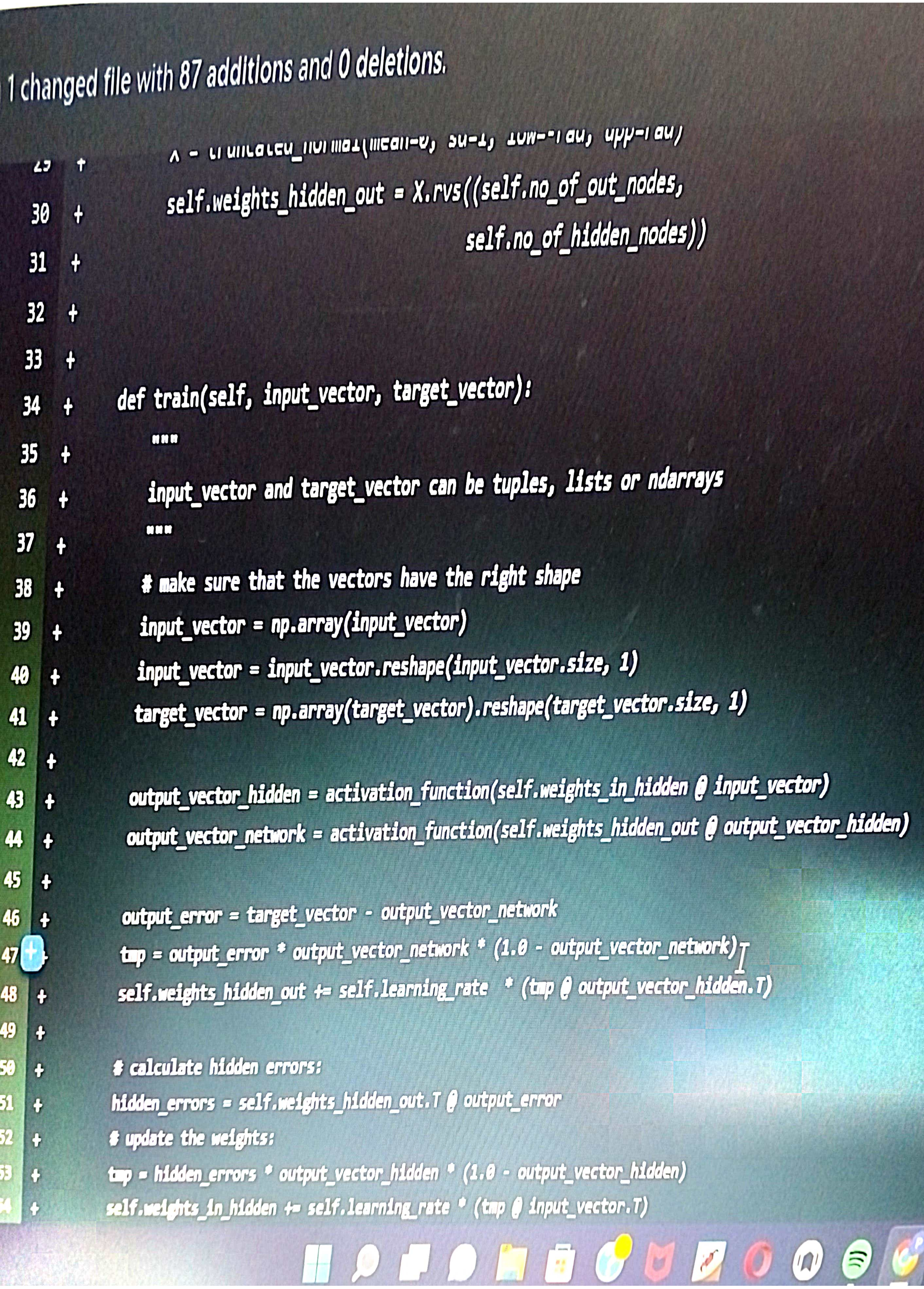


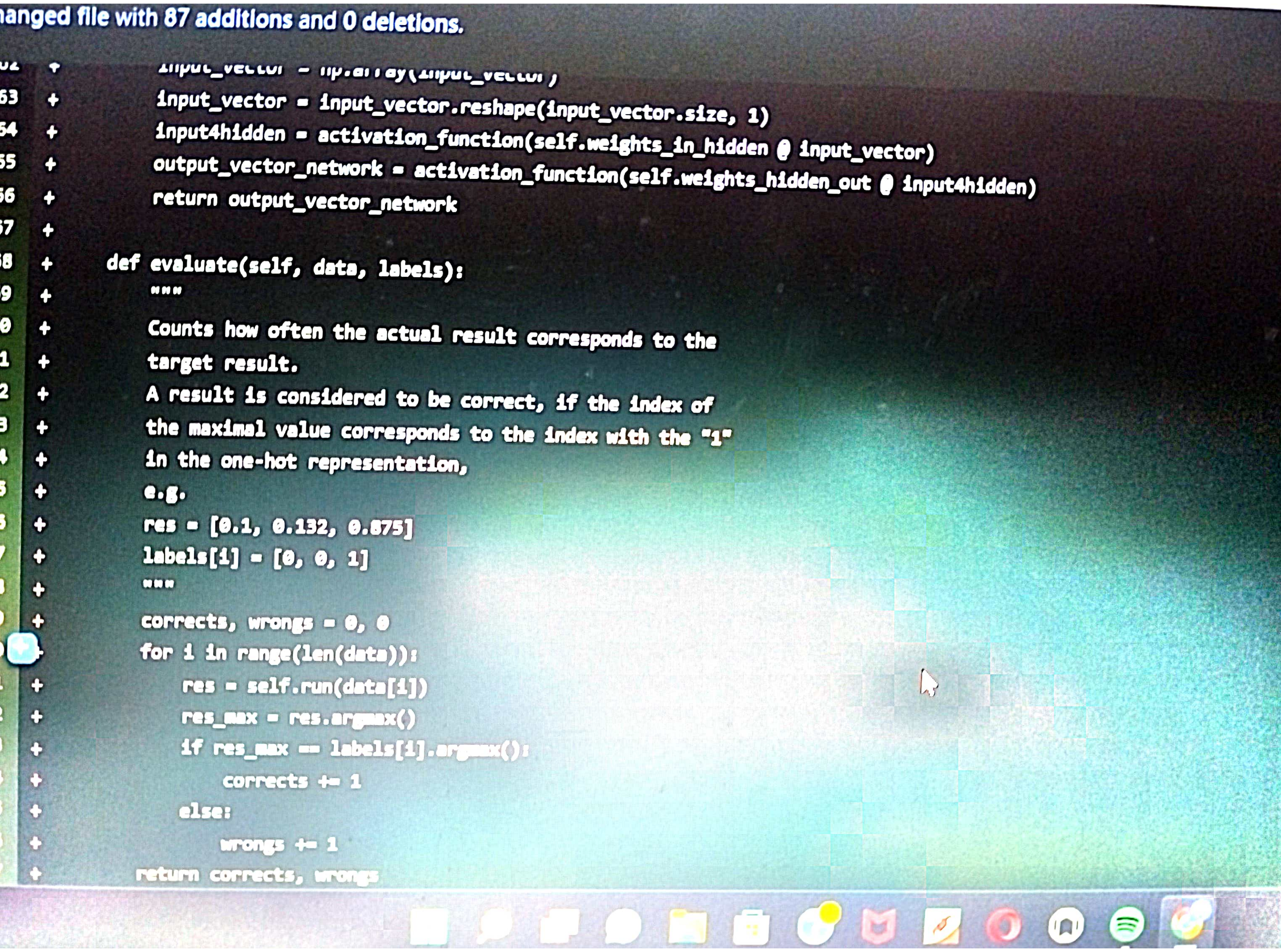
Showing 1 changed file with 87 additions and 0 deletions.

```
+ import numpy as np
+ from scipy.special import expit as activation_function
+ from scipy.stats import truncnorm
+
+ def truncated_normal(mean=0, sd=1, low=0, upp=10):
+     return truncnorm(
+         (low - mean) / sd, (upp - mean) / sd, loc=mean, scale=sd)
+
+ class NeuralNetwork:
+
+     def __init__(self,
+                  no_of_in_nodes,
+                  no_of_out_nodes,
+                  no_of_hidden_nodes,
+                  learning_rate):
+         self.no_of_in_nodes = no_of_in_nodes
+         self.no_of_out_nodes = no_of_out_nodes
+         self.no_of_hidden_nodes = no_of_hidden_nodes
+         self.learning_rate = learning_rate
+         self.create_weight_matrices()
+
+     def create_weight_matrices(self):
+         """ A method to initialize the weight matrices of the neural network"""
+         rad = 1 / np.sqrt(self.no_of_in_nodes)
+         X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
```



1 changed file with 87 additions and 0 deletions.

```
+        X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
+        self.weights_hidden_out = X.rvs((self.no_of_out_nodes,
+                                         self.no_of_hidden_nodes))
+
+
+
+    def train(self, input_vector, target_vector):
+        """
+        input_vector and target_vector can be tuples, lists or ndarrays
+        """
+        # make sure that the vectors have the right shape
+        input_vector = np.array(input_vector)
+        input_vector = input_vector.reshape(input_vector.size, 1)
+        target_vector = np.array(target_vector).reshape(target_vector.size, 1)
+
+        output_vector_hidden = activation_function(self.weights_in_hidden @ input_vector)
+        output_vector_network = activation_function(self.weights_hidden_out @ output_vector_hidden)
+
+        output_error = target_vector - output_vector_network
+        tmp = output_error * output_vector_network * (1.0 - output_vector_network)
+        self.weights_hidden_out += self.learning_rate  * (tmp @ output_vector_hidden.T)
+
+        # calculate hidden errors:
+        hidden_errors = self.weights_hidden_out.T @ output_error
+        # update the weights:
+        tmp = hidden_errors * output_vector_hidden * (1.0 - output_vector_hidden)
+        self.weights_in_hidden += self.learning_rate * (tmp @ input_vector.T)
```



hanged file with 87 additions and 0 deletions.

```
+        input_vector = np.array(input_vector)
+        input_vector = input_vector.reshape(input_vector.size, 1)
+        input4hidden = activation_function(self.weights_in_hidden @ input_vector)
+        output_vector_network = activation_function(self.weights_hidden_out @ input4hidden)
+        return output_vector_network
+
+    def evaluate(self, data, labels):
+        """
+        Counts how often the actual result corresponds to the
+        target result.
+        A result is considered to be correct, if the index of
+        the maximal value corresponds to the index with the "1"
+        in the one-hot representation,
+        e.g.
+        res = [0.1, 0.132, 0.875]
+        labels[i] = [0, 0, 1]
+        """
+        corrects, wrongs = 0, 0
+        for i in range(len(data)):
+            res = self.run(data[i])
+            res_max = res.argmax()
+            if res_max == labels[i].argmax():
+                corrects += 1
+            else:
+                wrongs += 1
+        return corrects, wrongs
```