usman-akinyemi-lab12

April 24, 2024

[11]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error
import tensorflow as tf
```

[12]:
```
data = np.genfromtxt('Fertility_Diagnosis.txt', delimiter=',')
X = data[:, :-1]
y = data[:, -1]
```

[13]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[14]:
```
input_layer_size = X_train.shape[1]
output_layer_size = 1
error_tolerance = 0.05

training_errors = []
testing_errors = []
```

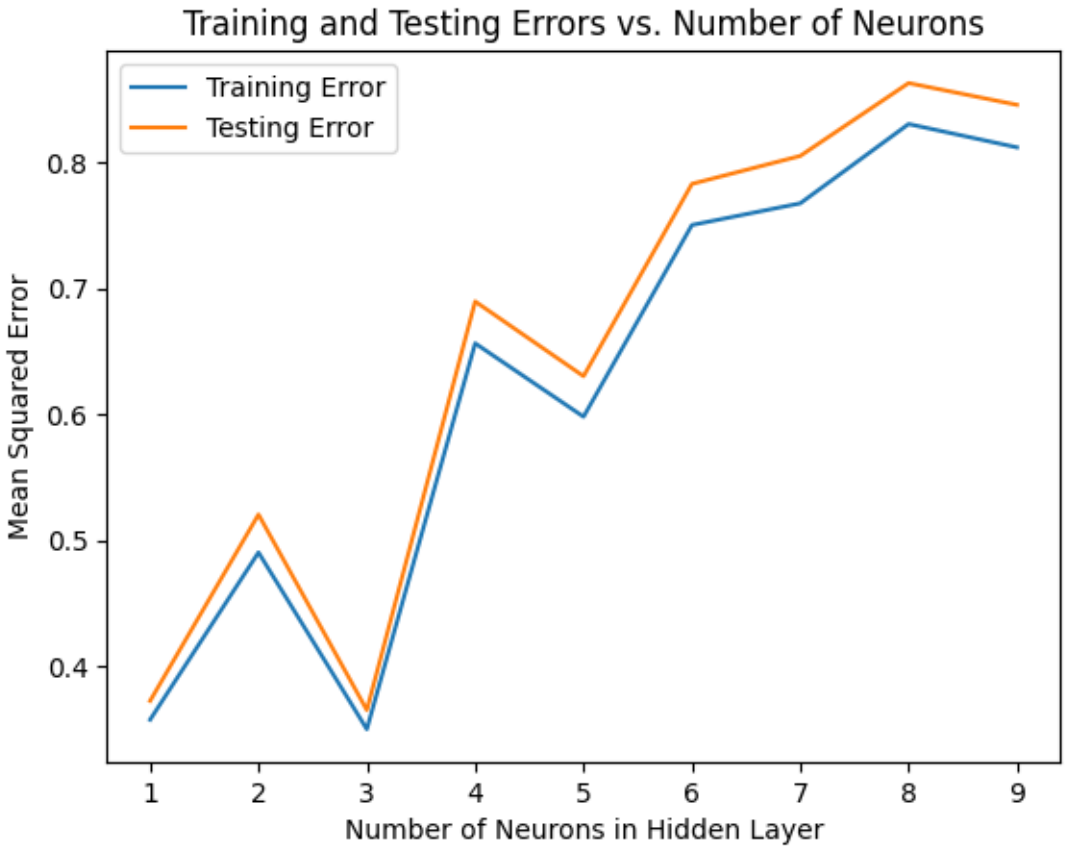
[15]:
```
for num_neurons in range(1, 10):
    # Initialize random synapse weights
    weights_input_hidden = np.random.rand(input_layer_size, num_neurons)
    weights_hidden_output = np.random.rand(num_neurons, output_layer_size)

    # Define the feedforward network for training data
    def feedforward(X):
        hidden_layer_input = np.dot(X, weights_input_hidden)
        hidden_layer_output = 1 / (1 + np.exp(-hidden_layer_input))

        output_layer_input = np.dot(hidden_layer_output, weights_hidden_output)
        output_layer_output = 1 / (1 + np.exp(-output_layer_input))

        return output_layer_output

    # Check for accuracy
    mse = mean_squared_error(y_train, feedforward(X_train))
```

```
    print(f"Training Mean Squared Error with {num_neurons} neurons: {mse}")

    for i in range(1, 1000001):
        # Implement backpropagation here

        if mse < error_tolerance:
            print(f"Stopping at {i} with {num_neurons} neurons")
            break

    mse_train = mean_squared_error(y_train, feedforward(X_train))
    print(f"Training Mean Squared Error: {mse_train}")
    training_errors.append(mse_train)

    mse_test = mean_squared_error(y_test, feedforward(X_test))
    print(f"Testing Mean Squared Error: {mse_test}")
    testing_errors.append(mse_test)
```

```
Training Mean Squared Error with 1 neurons: 0.3580079621148095
Training Mean Squared Error: 0.3580079621148095
Testing Mean Squared Error: 0.3729256809080814
Training Mean Squared Error with 2 neurons: 0.49051704443546074
Training Mean Squared Error: 0.49051704443546074
Testing Mean Squared Error: 0.5205693950717702
Training Mean Squared Error with 3 neurons: 0.35041312979655465
Training Mean Squared Error: 0.35041312979655465
Testing Mean Squared Error: 0.36561314126892663
Training Mean Squared Error with 4 neurons: 0.6562096601169989
Training Mean Squared Error: 0.6562096601169989
Testing Mean Squared Error: 0.6893887545489065
Training Mean Squared Error with 5 neurons: 0.5980641815621005
Training Mean Squared Error: 0.5980641815621005
Testing Mean Squared Error: 0.6301847567826233
Training Mean Squared Error with 6 neurons: 0.7499720757050581
Training Mean Squared Error: 0.7499720757050581
Testing Mean Squared Error: 0.7825768042489687
Training Mean Squared Error with 7 neurons: 0.7671738954036966
Training Mean Squared Error: 0.7671738954036966
Testing Mean Squared Error: 0.8047202607675512
Training Mean Squared Error with 8 neurons: 0.8300113796309034
Training Mean Squared Error: 0.8300113796309034
Testing Mean Squared Error: 0.8624710828104263
Training Mean Squared Error with 9 neurons: 0.8115138911523758
Training Mean Squared Error: 0.8115138911523758
Testing Mean Squared Error: 0.8452394193090988
```

[16]:
```
plt.figure()
plt.plot(range(1, 10), training_errors, label='Training Error')
```

```
plt.plot(range(1, 10), testing_errors, label='Testing Error')
plt.xlabel('Number of Neurons in Hidden Layer')
plt.ylabel('Mean Squared Error')
plt.title('Training and Testing Errors vs. Number of Neurons')
plt.legend()
plt.show()
```



QUESTIONS 1. What happens if all the weights are initialized to zero (zero initialization)? 2. What is the difference between epoch, batch, and iteration in neural networks? 3. What is the key difference between stochastic gradient decent and batch gradient decent? 4. What problems does a completely random initialization of a neural network lead to? 5. Name any ten hyperparameters in a multi-layer neural network.

1. If all weights are initialized to zero, the neural network will fail to learn effectively because all neurons in each layer will have the same output and thus the same gradient during back-propagation. This will result in all neurons in a layer learning the same features, which limits the representational capacity of the network.

2. In neural networks:

   - Epoch: One epoch is a single pass through the entire training dataset.

- Batch: Batch refers to the number of samples processed in one forward/backward pass of the neural network. It is a subset of the training dataset.
- Iteration: An iteration is the number of batches needed to complete one epoch.

3. - SGD: Uses only one random sample from the training set to compute the gradient at each iteration.
   - Batch Gradient Descent: Computes the gradient using the entire training set at each iteration.

4. Completely random initialization of a neural network can lead to several problems:

   - Slow convergence:
   - Vanishing or exploding gradients

5. Ten hyperparameters in a multi-layer neural network include:

   - Learning rate
   - Number of hidden layers
   - Number of neurons in each hidden layer
   - Activation function
   - Weight initialization method
   - Regularization parameter (e.g., L1 or L2 regularization)
   - Dropout rate
   - Batch size
   - Optimizer (e.g., SGD, Adam, RMSprop)
   - Number of epochs