Experiment No: 06

● **Aim: To implement a Simple Neural Network using backpropagation.**

● **Theory:**

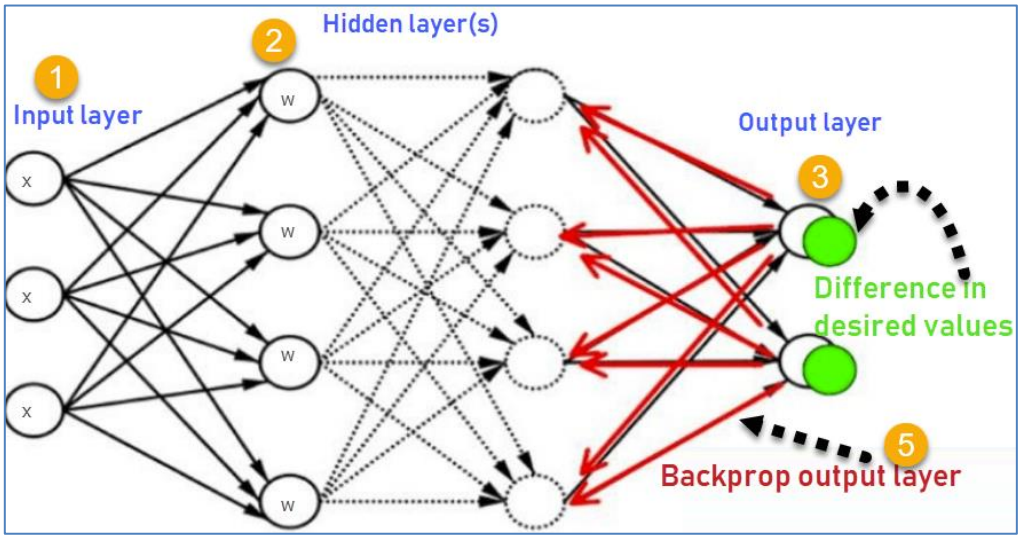
What is Backpropagation?

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization.

How Backpropagation Algorithm Works

The Back propagation algorithm in neural network computes the gradient of the loss function for a single weight by the chain rule. It efficiently computes one layer at a time, unlike a native direct computation. It computes the gradient, but it does not define how the gradient is used. It generalizes the computation in the delta rule.

Consider the following Back propagation neural network example diagram to understand:



1. Inputs X, arrive through the preconnected path
2. Input is modeled using real weights W. The weights are usually randomly selected.
3. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
4. Calculate the error in the outputs

   ErrorB= Actual Output – Desired Output

5. Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

Types of Backpropagation Networks

Two Types of Backpropagation Networks are:

- Static Back-propagation
- Recurrent Backpropagation

Static back-propagation:

It is one kind of backpropagation network which produces a mapping of a static input for static output. It is useful to solve static classification issues like optical character recognition.

Recurrent Backpropagation:

Recurrent Back propagation in data mining is fed forward until a fixed value is achieved. After that, the error is computed and propagated backward.

The main difference between both of these methods is: that the mapping is rapid in static back-propagation while it is non static in recurrent backpropagation

Best practice Backpropagation

Backpropagation in neural network can be explained with the help of "Shoe Lace" analogy

Too little tension =

- Not enough constraining and very loose

Too much tension =

- Too much constraint (overtraining)
- Taking too much time (relatively slow process)
- Higher likelihood of breaking

Pulling one lace more than other =

- Discomfort (bias)

Disadvantages of using Backpropagation

- The actual performance of backpropagation on a specific problem is dependent on the input data.
- Back propagation algorithm in data mining can be quite sensitive to noisy data
- You need to use the matrix-based approach for backpropagation instead of mini-batch.

To implement a Simple Neural Network using backpropagation:

jupyter Simple Neural Network Last Checkpoint: 09/20/2023 (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [25]:
```
# Import Libraries
import numpy as np
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
```

In [26]:
```
# Load dataset
data = load_iris()

# Get features and target
X=data.data
y=data.target
```

In [27]:
```
# Get dummy variable
y = pd.get_dummies(y).values

y[:3]
```

Out[27]:
```
array([[1, 0, 0],
       [1, 0, 0],
       [1, 0, 0]], dtype=uint8)
```

Split train and test set

In [28]:
```
#Split data into train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)
```

Initialize Hyperparameters and Weights

In [29]:
```
# Initialize variables
learning_rate = 0.1
iterations = 5000
N = y_train.size

# number of input features
input_size = 4

# number of hidden layers neurons
hidden_size = 2

# number of neurons at the output layer
output_size = 3

results = pd.DataFrame(columns=["mse", "accuracy"])
```

In [30]:
```
# Initialize weights
np.random.seed(10)

# initializing weight for the hidden layer
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))

# initializing weight for the output layer
W2 = np.random.normal(scale=0.5, size=(hidden_size , output_size))
```

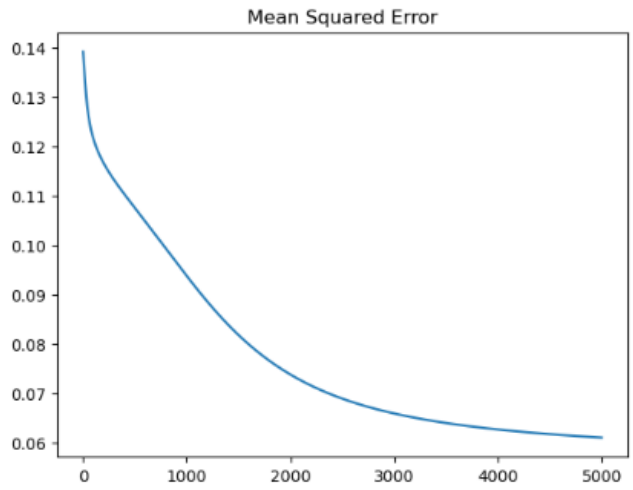
In [31]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def mean_squared_error(y_pred, y_true):
    return ((y_pred - y_true)**2).sum() / (2*y_pred.size)

def accuracy(y_pred, y_true):
    acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
    return acc.mean()
```

Backpropagation Neural Network

In [33]:
```
for itr in range(iterations):

    # feedforward propagation
    # on hidden layer
    Z1 = np.dot(X_train, W1)
    A1 = sigmoid(Z1)

    # on output layer
    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)


    # Calculating error
    mse = mean_squared_error(A2, y_train)
    acc = accuracy(A2, y_train)
    results=results.append({"mse":mse, "accuracy":acc},ignore_index=True )

    # backpropagation
    E1 = A2 - y_train
    dW1 = E1 * A2 * (1 - A2)

    E2 = np.dot(dW1, W2.T)
    dW2 = E2 * A1 * (1 - A1)


    # weight updates
    W2_update = np.dot(A1.T, dW1) / N
    W1_update = np.dot(X_train.T, dW2) / N

    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update
```

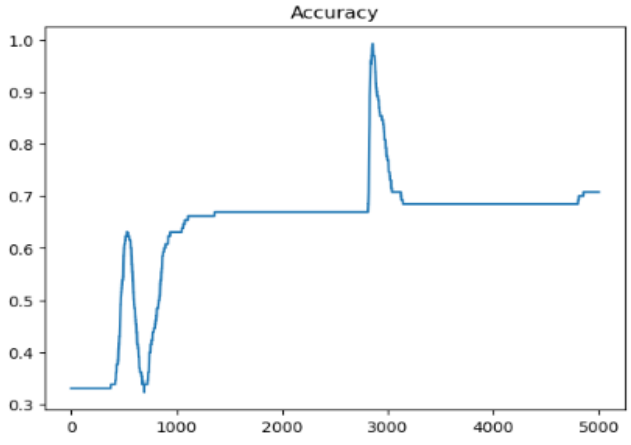
In [34]:
```
results.mse.plot(title="Mean Squared Error")
```

Out[34]:
```
<Axes: title={'center': 'Mean Squared Error'}>
```



In [35]:
```
results.accuracy.plot(title="Accuracy")
```

Out[35]:
```
<Axes: title={'center': 'Accuracy'}>
```



```
In [37]: # feedforward
         Z1 = np.dot(X_test, W1)
         A1 = sigmoid(Z1)

         Z2 = np.dot(A1, W2)
         A2 = sigmoid(Z2)

         acc = accuracy(A2, y_test)
         print("Accuracy: {}".format(acc))

         Accuracy: 0.8

In [ ]:
```

● Conclusion:

Implementing a simple neural network using backpropagation is a foundational step in the world of machine learning and artificial intelligence. It provides a solid understanding of how neural networks learn from data and how to train them to perform various tasks, from image classification to natural language processing and beyond. Further exploration and practice with more complex neural network architectures and real-world datasets will deepen your expertise in this exciting field.