Introduction: Neural networks, primarily known for their applications in classification tasks, can also be employed for regression problems. In this exercise, we aim to demonstrate the usage of a simple neural network in predicting a linear relationship between input features and the output.

Objective:

The primary objective is to predict the output value y based on two input features x1 and x2 using a basic neural network model with the equation y=3x1+4x2. The neural network will be trained using a feed-forward and back-propagation mechanism to minimize the error between the predicted and actual values.

Methodology:

Activation Functions:

Define the sigmoid function, a popular activation function used in neural networks.

Define the tanh function, which is another common activation function that returns values between -1 and 1.

Define the relu function, which returns positive values as they are and zeroes out negative values

```
# import Library Function
import numpy as np
import matplotlib.pyplot as plt
```

```
# Sigmoid Activation Function
def sigmoid(x):
  return 1/(1+np.exp(-x))

# tanh Activation function
def tanh(x):
  return np.tanh(x)

# Relu Activation function
def relu(x):
  return np.maximum(0,x)
```

2.Sample Data Use the Following sample data for training:

x1=3, x2=2, yactual=17

```
x1=3
x2=2
y_actual=17
```

Feed Forward: .Initialize the random weights w1 and w2 between 1 and 10. .Predict the output ypred using the equation ypred= x1 x w1 + x2 x w2

```
# Initial random weights
w1, w2 = np.random.uniform(1, 10, 2)

# Input values
x1, x2 = 3, 2

# Prediction
y_pred = x1 * w1 + x2 * w2
print(y_pred)
```

```
14.580755838855051
```

4. Error Calculation: ● Compute the squared error as error = (yactual-ypred)*(yactual-ypred)

```
error = (y_actual-y_pred)**2
print(error)
```

```
5.852742311233928
```

Back Propagation: Compute the gradients of the error with respect to the weights. Update the weights using the computed gradients and a learning rate

```
learning_rate = 0.01
grad_w1 = -2 * (y_actual - y_pred) * x1  # Gradient with respect to w1
grad_w2 = -2 * (y_actual - y_pred) * x2  # Gradient with respect to w2

# Update the weights
w1 = w1 - learning_rate * grad_w1
w2 = w2 - learning_rate * grad_w2
```

Training: The above steps (Feed Forward to Back Propagation) are performed iteratively (20 times in this example) to refine the weights and minimize the error.

```
errors = []
predictions = []
for epoch in range(20):
    # Feed forward
    y_pred = x1 * w1 + x2 * w2

    # Error calculation
    error = (y_actual - y_pred) ** 2

    # Back-propagation
    grad_w1 = -2 * (y_actual - y_pred) * x1
    grad_w2 = -2 * (y_actual - y_pred) * x2

    # Update weights
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2

    errors.append(error)  # Store error for this epoch
    predictions.append(y_pred) # Store prediction for this epoch

    # Print error and prediction for each epoch
    print(f'Epoch {epoch+1}, Error: {error}, Prediction: {y_pred}')
```

```
Epoch 1, Error: 3.2049616896316975, Prediction: 15.209759320752738
Epoch 2, Error: 1.7550370212423168, Prediction: 15.675221897357027
Epoch 3, Error: 0.9610582728322974, Prediction: 16.019664204044197
Epoch 4, Error: 0.5262755102029626, Prediction: 16.27455151099271
Epoch 5, Error: 0.2881884693871426, Prediction: 16.463168118134604
Epoch 6, Error: 0.1578120058363993, Prediction: 16.602744407419607
Epoch 7, Error: 0.08641785439601317, Prediction: 16.706030861490508
Epoch 8, Error: 0.047322417067256595, Prediction: 16.782462837502976
Epoch 9, Error: 0.025913755586029163, Prediction: 16.839022499752204
Epoch 10, Error: 0.014190372558910314, Prediction: 16.880876649816628
Epoch 11, Error: 0.007770648013259037, Prediction: 16.911848720864306
Epoch 12, Error: 0.004255206852061038, Prediction: 16.934768053439583
Epoch 13, Error: 0.00233015127218857, Prediction: 16.951728359545292
Epoch 14, Error: 0.0012759908366502678, Prediction: 16.96427898606352
Epoch 15, Error: 0.0006987325821499722, Prediction: 16.973566449687
Epoch 16, Error: 0.0003826259619852302 3, Prediction: 16.98043917276838
Epoch 17, Error: 0.00020952597678309767, Prediction: 16.985524987848603
Epoch 18, Error: 0.00011473642488644408, Prediction: 16.989288491007965
Epoch 19, Error: 6.282966626786972e-05, Prediction: 16.99207348334589
Epoch 20, Error: 3.4405525248248785e-05, Prediction: 16.994134377675962
```

7. Visualization: Plot the progression of the error across epochs. Plot the progression of y_pred across epochs

```
#epochs
epochs = range(1, 21)

# Plot error progression
plt.plot(epochs, errors)
plt.xlabel('Epochs')
plt.ylabel('Error')
plt.title('Error across Epochs')
plt.show()

# Plot y_pred progression
plt.plot(epochs,predictions )
plt.xlabel('Epochs')
plt.ylabel('Predicted Value')
```

```
plt.title('Prediction across Epochs')
plt.show()
```