Lab No: 6

Aim:
Classification of Handwritten Digits using Artificial Neural Networks

Description:
Perform the following task with using inbuilt Python Libraries:
1. Search relevant datasets to perform classification (MNIST dataset)
2. Classify handwritten digits using a simple neural network that has only input and output layers.
3. Add a hidden layer and see how the performance of the model improves.
4. Apply various activation functions to hidden and output layers to assess the model performance.
5. Apply various cost functions to measure the error of the model.

Activation Functions:
1. Softmax: Output layer for multi-class classification probabilities.
2. Sigmoid: Binary classification activation for output layer probabilities.
3. Tanh: Symmetric activation for hidden layers, mapping values to [-1, 1].
4. ReLU: Rectified Linear Unit, popular for hidden layer activations.

Cost/Loss Functions:
1. Mean Squared Error: Measures squared difference between predicted and actual values.
2. Categorical Crossentropy: Ideal for multi-class classification tasks, penalizing class probability deviations.
3. Binary Crossentropy: Suited for binary classification problems, optimizing log-likelihood of true labels.
4. Poisson: Used for count data; models distribution of rare events.

Source Code:

Classification of Handwritten Digits using Artificial Neural Networks

Perform the following steps for above mentioned problem statement:

1. Search relevant datasets to perform classification

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
print(test_images)
```

```
print(test_labels)
```

2. Classify handwritten digits using a simple neural network that has only input and output layers.

```
from keras.models import Sequential
from keras.layers import Flatten, Dense
from keras.utils import to_categorical

train_images = train_images.reshape((60000, 28 * 28)).astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28)).astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model = Sequential()
model.add(Dense(10, activation='softmax', input_shape=(28 * 28,)))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=25, validation_split=0.2)
evaluation = model.evaluate(test_images, test_labels)
print(f"Test Accuracy: {evaluation[1]*100:.2f}%")
print(f"Test Loss: {evaluation[0]}")
```

3. Add a hidden layer and see how the performance of the model improves.

```
model = Sequential()
model.add(Dense(100, activation='relu', input_shape=(28 * 28,)))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(train_images, train_labels, epochs=5, batch_size=25, validation_split=0.2)
evaluation = model.evaluate(test_images, test_labels)
print(f"Test Accuracy: {evaluation[1]*100:.2f}%")
print(f"Test Loss: {evaluation[0]}")
```

4. Apply various activation functions to hidden and output layers to assess the model performance.

```
activation_functions = ['sigmoid', 'tanh', 'relu']
for activation_function in activation_functions:
    model = Sequential()
    model.add(Dense(100, activation='relu', input_shape=(28 * 28,)))
    model.add(Dense(10, activation= activation_function))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    model.fit(train_images, train_labels, epochs=5, batch_size=25, validation_split=0.2)
    evaluation = model.evaluate(test_images, test_labels)
    print(f"Test Accuracy with {loss_function}: {evaluation[1]*100:.2f}%")
    print(f"Test Loss with {loss_function}: {evaluation[0]}")
```

5. Apply various cost functions to measure the error of the model.

```
loss_functions = ['mean_squared_error', 'categorical_crossentropy', 'binary_crossentropy', 'poisson']
for loss_function in loss_functions:
    print(f"\nUsing {loss_function} as loss function:")
    model.compile(optimizer='adam', loss=loss_function, metrics=['accuracy'])
    model.fit(train_images, train_labels, epochs=5, batch_size=25, validation_split=0.2)
    evaluation = model.evaluate(test_images, test_labels)
    print(f"Test Accuracy: {evaluation[1]*100:.2f}%")
    print(f"Test Loss: {evaluation[0]}")
```

Output:

Task 1:

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

```
[7 2 1 ... 4 5 6]
```

Figure 5.1 Test Images and Test Label

Task 2:

```
Test Accuracy: 92.47%
Test Loss: 0.27226367592811584
```

Figure 5.2 Without Hidden Layer

Task 3:

```
Test Accuracy: 97.57%
Test Loss: 0.08260300010442734
```

Figure 5.3 With Hidden Layer

Task 4:

```
Test Accuracy with sigmoid: 97.66%
Test Loss with sigmoid: 0.07542455196380615
```

```
Test Accuracy with tanh: 9.80%
Test Loss with tanh: nan
```

```
Test Accuracy with relu: 9.80%
Test Loss with relu: nan
```

Figure 5.4 Activation Functions

Task 5:

```
Test Accuracy with mean_squared_error: 97.20%
Test Loss with mean_squared_error: 0.004385668784379959
```

```
Test Accuracy with categorical_crossentropy: 97.77%
Test Loss with categorical_crossentropy: 0.07678196579217911
```

```
Test Accuracy with binary_crossentropy: 98.04%
Test Loss with binary_crossentropy: 0.017176324501633644
```

```
Test Accuracy with poisson: 97.85%
Test Loss with poisson: 0.11014503240585327
```

Figure 5.5 Cost/Loss Function

Conclusion:

- The initial model achieves a baseline performance (92.47%) for handwritten digit classification.
- The inclusion of a hidden layer improves the model's ability having accuracy of 97.57%.
- The model's performance with “Sigmoid” activation function is the highest (97.66%).
- “Categorical cross entropy” is commonly used for classification tasks, but the experimentation with other loss functions provides a better loss function (“Binary cross entropy”) with accuracy of 98.04%.