DL Lab 3

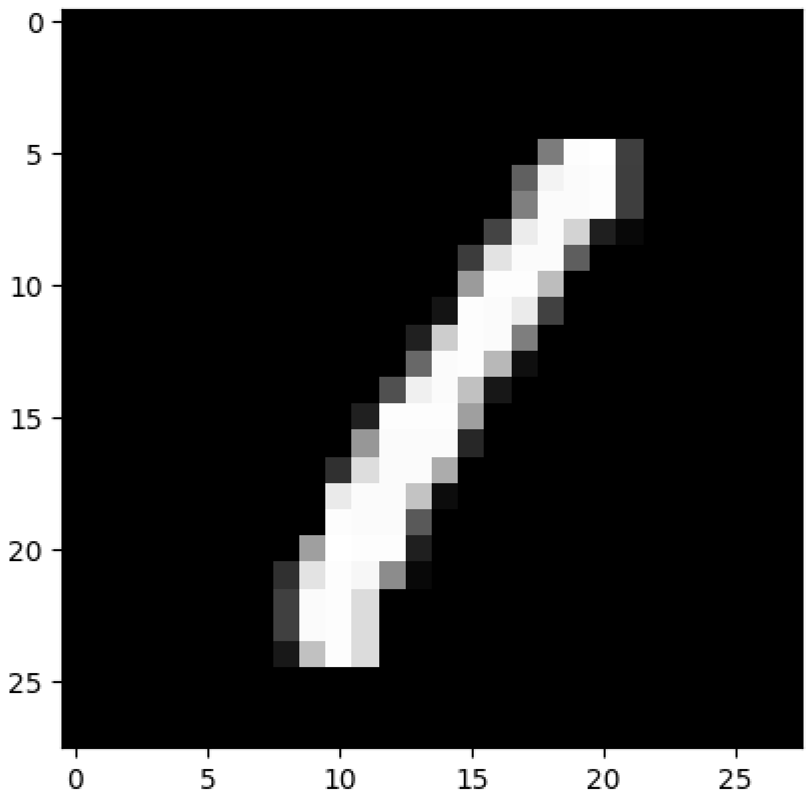
Name: Soumyadeep Ganguly

Reg No: 24MDT0082

In [24]:
```
import numpy as np
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import SGD
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
```

Question 1

In [47]:
```
batch_size = 128
num_classes = 10
epochs = 10
(X_train, y_train), (X_test, y_test) = mnist.load_data()
plt.imshow(X_train[3], cmap="grey")
plt.show()
```



Reshape and 1-hot encoding

In [48]:
```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
y_train_cat = keras.utils.to_categorical(y_train, num_classes=10)
y_test_cat = keras.utils.to_categorical(y_test, num_classes=10)
```

In [27]:
```
model = Sequential()
model.add(Dense(784, activation='relu', input_shape=(784,)))
model.add(Dense(512, activation='sigmoid', input_shape=(784,)))
```

```
model.add(Dense(512, activation='sigmoid', input_shape=(512,)))
model.add(Dense(10, activation = 'softmax'))
model.summary()
```

```
e:\VIT Study Materials\SEM 3\Deep Learning\LAB\.venv\Lib\site-packages\keras\src\layers\core\dense.py:93: UserWarning: Do not p
ass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as th
e first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 784)	615,440
dense_13 (Dense)	(None, 512)	401,920
dense_14 (Dense)	(None, 512)	262,656
dense_15 (Dense)	(None, 10)	5,130

Total params: 1,285,146 (4.90 MB)

Trainable params: 1,285,146 (4.90 MB)

Non-trainable params: 0 (0.00 B)

In [28]:
```
sgd1 = SGD(learning_rate = 0.01)
model.compile(loss='CategoricalCrossentropy', optimizer=sgd1, metrics=['accuracy'])
model.fit(X_train, y_train_cat, batch_size=50, epochs=10,verbose=1, validation_data=(X_test, y_test_cat))
```

```
Epoch 1/10
1200/1200 ━━━━━━━━━━━━━━━━━━━━ 29s 24ms/step - accuracy: 0.1612 - loss: 2.2781 - val_accuracy: 0.3554 - val_loss: 2.1177
Epoch 2/10
1200/1200 ━━━━━━━━━━━━━━━━━━━━ 29s 24ms/step - accuracy: 0.4873 - loss: 1.9752 - val_accuracy: 0.6685 - val_loss: 1.3126
Epoch 3/10
1200/1200 ━━━━━━━━━━━━━━━━━━━━ 28s 23ms/step - accuracy: 0.7168 - loss: 1.1465 - val_accuracy: 0.8160 - val_loss: 0.7699
Epoch 4/10
1200/1200 ━━━━━━━━━━━━━━━━━━━━ 27s 23ms/step - accuracy: 0.8124 - loss: 0.7218 - val_accuracy: 0.8517 - val_loss: 0.5619
Epoch 5/10
1200/1200 ━━━━━━━━━━━━━━━━━━━━ 27s 22ms/step - accuracy: 0.8530 - loss: 0.5483 - val_accuracy: 0.8738 - val_loss: 0.4611
Epoch 6/10
1200/1200 ━━━━━━━━━━━━━━━━━━━━ 27s 23ms/step - accuracy: 0.8738 - loss: 0.4618 - val_accuracy: 0.8864 - val_loss: 0.4052
Epoch 7/10
1200/1200 ━━━━━━━━━━━━━━━━━━━━ 27s 22ms/step - accuracy: 0.8833 - loss: 0.4091 - val_accuracy: 0.8940 - val_loss: 0.3713
Epoch 8/10
1200/1200 ━━━━━━━━━━━━━━━━━━━━ 27s 23ms/step - accuracy: 0.8942 - loss: 0.3762 - val_accuracy: 0.9001 - val_loss: 0.3473
Epoch 9/10
1200/1200 ━━━━━━━━━━━━━━━━━━━━ 27s 23ms/step - accuracy: 0.8993 - loss: 0.3522 - val_accuracy: 0.9056 - val_loss: 0.3279
Epoch 10/10
1200/1200 ━━━━━━━━━━━━━━━━━━━━ 27s 22ms/step - accuracy: 0.9053 - loss: 0.3326 - val_accuracy: 0.9101 - val_loss: 0.3141
```

Out[28]:
```
<keras.src.callbacks.history.History at 0x2837d0176b0>
```

Scores

In [36]:
```python
score = model.evaluate(X_test, y_test_cat, verbose=0)
print(f"Loss: {score[0]}")
print(f"Accuracy: {score[1]}")
```

```
Loss: 0.31405189633369446
Accuracy: 0.910099983215332
```

In [30]:
```python
predictions = model.predict(X_test)
for i in range(5):
    predicted_class = np.argmax(predictions[i])
    true_class = np.argmax(y_test_cat[i])
    print(f"Image {i+1}: Predicted = {predicted_class}, True = {true_class}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step
Image 1: Predicted = 7, True = 7
Image 2: Predicted = 2, True = 2
Image 3: Predicted = 1, True = 1
Image 4: Predicted = 0, True = 0
Image 5: Predicted = 4, True = 4
```

Question 2: Regularization Techniques

In [37]:
```python
from keras.layers import Dropout
from keras.callbacks import EarlyStopping
```

In [49]:
```python
model2 = Sequential()
model2.add(Dense(784, activation="relu", input_shape=(784,)))
model2.add(Dense(512, activation="sigmoid"))
model2.add(Dropout(0.2))
model2.add(Dense(512, activation="sigmoid"))
model2.add(Dropout(0.2))
model2.add(Dense(10, activation="softmax"))
model2.summary()
```

```
e:\VIT Study Materials\SEM 3\Deep Learning\LAB\.venv\Lib\site-packages\keras\src\layers\core\dense.py:93: UserWarning: Do not p
ass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as th
e first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
dense_24 (Dense)	(None, 784)	615,440
dense_25 (Dense)	(None, 512)	401,920
dropout_6 (Dropout)	(None, 512)	0
dense_26 (Dense)	(None, 512)	262,656
dropout_7 (Dropout)	(None, 512)	0
dense_27 (Dense)	(None, 10)	5,130

Total params: 1,285,146 (4.90 MB)

Trainable params: 1,285,146 (4.90 MB)

Non-trainable params: 0 (0.00 B)

In [50]:
```
X_subtrain, X_valid, y_subtrain, y_valid = train_test_split(X_train, y_train, test_size=0.1, random_state=1)
y_subtrain_cat = keras.utils.to_categorical(y_subtrain, num_classes=10)
y_valid_cat = keras.utils.to_categorical(y_valid, num_classes=10)
```

In [51]:
```
estop = EarlyStopping(monitor = 'val_loss', min_delta = 1e-5, mode= 'min', patience=4, verbose = 1, restore_best_weights=True)
```

In [52]:
```
sgd2 = SGD(learning_rate = 0.01)
```

In [54]:
```
model2.compile(loss='CategoricalCrossentropy', optimizer=sgd2, metrics=['accuracy'])
model2.fit(X_subtrain, y_subtrain_cat, batch_size = 32, epochs=50, verbose = 1, validation_data = (X_valid, y_valid_cat), call
```

```
Epoch 1/50
   3/1688 ━━━━━━━━━━━━━━━━━━━━ 46s 28ms/step - accuracy: 0.3003 - loss: 2.0419
e:\VIT Study Materials\SEM 3\Deep Learning\LAB\.venv\Lib\site-packages\tensorflow\python\data\ops\structured_function.py:258: U
serWarning: Even though the `tf.config.experimental_run_functions_eagerly` option is set, this option does not apply to tf.data
functions. To force eager execution of tf.data functions, please use `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(
```

```
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 46s 27ms/step - accuracy: 0.4407 - loss: 1.7252 - val_accuracy: 0.7493 - val_loss: 0.9479
Epoch 2/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 45s 26ms/step - accuracy: 0.7118 - loss: 0.9110 - val_accuracy: 0.8278 - val_loss: 0.6015
Epoch 3/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 44s 26ms/step - accuracy: 0.8104 - loss: 0.6229 - val_accuracy: 0.8687 - val_loss: 0.4666
Epoch 4/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 44s 26ms/step - accuracy: 0.8487 - loss: 0.5063 - val_accuracy: 0.8827 - val_loss: 0.4099
Epoch 5/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.8725 - loss: 0.4352 - val_accuracy: 0.8927 - val_loss: 0.3759
Epoch 6/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 49s 29ms/step - accuracy: 0.8757 - loss: 0.4172 - val_accuracy: 0.8983 - val_loss: 0.3505
Epoch 7/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 52s 31ms/step - accuracy: 0.8874 - loss: 0.3812 - val_accuracy: 0.9003 - val_loss: 0.3364
Epoch 8/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.8952 - loss: 0.3575 - val_accuracy: 0.9038 - val_loss: 0.3183
Epoch 9/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 28ms/step - accuracy: 0.8982 - loss: 0.3419 - val_accuracy: 0.9093 - val_loss: 0.3073
Epoch 10/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9049 - loss: 0.3271 - val_accuracy: 0.9113 - val_loss: 0.2976
Epoch 11/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9071 - loss: 0.3159 - val_accuracy: 0.9140 - val_loss: 0.2853
Epoch 12/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 28ms/step - accuracy: 0.9105 - loss: 0.3035 - val_accuracy: 0.9192 - val_loss: 0.2716
Epoch 13/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 49s 29ms/step - accuracy: 0.9151 - loss: 0.2885 - val_accuracy: 0.9228 - val_loss: 0.2643
Epoch 14/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 50s 30ms/step - accuracy: 0.9205 - loss: 0.2742 - val_accuracy: 0.9267 - val_loss: 0.2526
Epoch 15/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9224 - loss: 0.2637 - val_accuracy: 0.9282 - val_loss: 0.2419
Epoch 16/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9266 - loss: 0.2526 - val_accuracy: 0.9298 - val_loss: 0.2336
Epoch 17/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9295 - loss: 0.2410 - val_accuracy: 0.9325 - val_loss: 0.2263
Epoch 18/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9313 - loss: 0.2317 - val_accuracy: 0.9358 - val_loss: 0.2185
Epoch 19/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9337 - loss: 0.2250 - val_accuracy: 0.9370 - val_loss: 0.2109
Epoch 20/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9365 - loss: 0.2154 - val_accuracy: 0.9398 - val_loss: 0.2041
Epoch 21/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 28ms/step - accuracy: 0.9384 - loss: 0.2123 - val_accuracy: 0.9413 - val_loss: 0.1972
```

```
Epoch 22/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 51s 30ms/step - accuracy: 0.9394 - loss: 0.2057 - val_accuracy: 0.9407 - val_loss: 0.1955
Epoch 23/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 55s 33ms/step - accuracy: 0.9414 - loss: 0.1964 - val_accuracy: 0.9443 - val_loss: 0.1860
Epoch 24/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 492s 292ms/step - accuracy: 0.9428 - loss: 0.1899 - val_accuracy: 0.9467 - val_loss: 0.1819
Epoch 25/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 28ms/step - accuracy: 0.9451 - loss: 0.1808 - val_accuracy: 0.9477 - val_loss: 0.1759
Epoch 26/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 50s 30ms/step - accuracy: 0.9476 - loss: 0.1766 - val_accuracy: 0.9495 - val_loss: 0.1710
Epoch 27/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9495 - loss: 0.1751 - val_accuracy: 0.9508 - val_loss: 0.1671
Epoch 28/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 45s 27ms/step - accuracy: 0.9509 - loss: 0.1663 - val_accuracy: 0.9513 - val_loss: 0.1648
Epoch 29/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 45s 27ms/step - accuracy: 0.9530 - loss: 0.1593 - val_accuracy: 0.9530 - val_loss: 0.1607
Epoch 30/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 46s 27ms/step - accuracy: 0.9528 - loss: 0.1584 - val_accuracy: 0.9527 - val_loss: 0.1588
Epoch 31/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9548 - loss: 0.1507 - val_accuracy: 0.9542 - val_loss: 0.1547
Epoch 32/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 28ms/step - accuracy: 0.9589 - loss: 0.1412 - val_accuracy: 0.9563 - val_loss: 0.1513
Epoch 33/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 49s 29ms/step - accuracy: 0.9593 - loss: 0.1378 - val_accuracy: 0.9568 - val_loss: 0.1476
Epoch 34/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 28ms/step - accuracy: 0.9597 - loss: 0.1353 - val_accuracy: 0.9558 - val_loss: 0.1471
Epoch 35/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 29ms/step - accuracy: 0.9618 - loss: 0.1291 - val_accuracy: 0.9582 - val_loss: 0.1431
Epoch 36/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 28ms/step - accuracy: 0.9606 - loss: 0.1339 - val_accuracy: 0.9600 - val_loss: 0.1402
Epoch 37/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 28ms/step - accuracy: 0.9628 - loss: 0.1251 - val_accuracy: 0.9592 - val_loss: 0.1381
Epoch 38/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 29ms/step - accuracy: 0.9622 - loss: 0.1262 - val_accuracy: 0.9593 - val_loss: 0.1366
Epoch 39/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 28ms/step - accuracy: 0.9649 - loss: 0.1203 - val_accuracy: 0.9595 - val_loss: 0.1333
Epoch 40/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 49s 29ms/step - accuracy: 0.9642 - loss: 0.1189 - val_accuracy: 0.9613 - val_loss: 0.1327
Epoch 41/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 54s 32ms/step - accuracy: 0.9649 - loss: 0.1162 - val_accuracy: 0.9610 - val_loss: 0.1298
Epoch 42/50
```

```
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 49s 29ms/step - accuracy: 0.9658 - loss: 0.1139 - val_accuracy: 0.9630 - val_loss: 0.1277
Epoch 43/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9669 - loss: 0.1090 - val_accuracy: 0.9620 - val_loss: 0.1257
Epoch 44/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9688 - loss: 0.1050 - val_accuracy: 0.9638 - val_loss: 0.1224
Epoch 45/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 47s 28ms/step - accuracy: 0.9681 - loss: 0.1062 - val_accuracy: 0.9635 - val_loss: 0.1217
Epoch 46/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 29ms/step - accuracy: 0.9682 - loss: 0.1051 - val_accuracy: 0.9640 - val_loss: 0.1212
Epoch 47/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 51s 30ms/step - accuracy: 0.9706 - loss: 0.0981 - val_accuracy: 0.9652 - val_loss: 0.1197
Epoch 48/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 29ms/step - accuracy: 0.9709 - loss: 0.0964 - val_accuracy: 0.9633 - val_loss: 0.1190
Epoch 49/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 28ms/step - accuracy: 0.9717 - loss: 0.0944 - val_accuracy: 0.9652 - val_loss: 0.1155
Epoch 50/50
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 48s 28ms/step - accuracy: 0.9722 - loss: 0.0966 - val_accuracy: 0.9667 - val_loss: 0.1151
Restoring model weights from the end of the best epoch: 50.
```

Out[54]: `<keras.src.callbacks.history.History at 0x2837e003d10>`

In [55]:

```
score = model2.evaluate(X_test, y_test_cat, verbose=0)
print(f"Loss: {score[0]}")
print(f"Accuracy: {score[1]}")
```

```
Loss: 0.09932716935873032
Accuracy: 0.9693999886512756
```

In [56]:

```
predictions = model2.predict(X_test)
for i in range(5):
    predicted_class = np.argmax(predictions[i])
    true_class = np.argmax(y_test_cat[i])
    print(f"Image {i+1}: Predicted = {predicted_class}, True = {true_class}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
Image 1: Predicted = 7, True = 7
Image 2: Predicted = 2, True = 2
Image 3: Predicted = 1, True = 1
Image 4: Predicted = 0, True = 0
Image 5: Predicted = 4, True = 4
```

Challenging Question

In [57]:
```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def sigmoid_derivative(x):
    return x * (1 - x)
```

In [ ]:
```
class NeuralNetwork:
    def __init__(self, input_size, hidden_size, output_size):
        self.weights_input_hidden = np.random.randn(input_size, hidden_size)
        self.weights_hidden_output = np.random.randn(hidden_size, output_size)
        self.bias_hidden = np.zeros((1, hidden_size))
        self.bias_output = np.zeros((1, output_size))

    def forwardpass(self, X):
        self.hidden_input = np.dot(X, self.weights_input_hidden) + self.bias_hidden
        self.hidden_output = sigmoid(self.hidden_input)

        self.final_input = np.dot(self.hidden_output, self.weights_hidden_output) + self.bias_output
        self.output = sigmoid(self.final_input)

        return self.output

    def backwardpass(self, X, y, output, learning_rate):
        error = y - output
        d_output = error * sigmoid_derivative(output)

        error_hidden = d_output.dot(self.weights_hidden_output.T)
        d_hidden = error_hidden * sigmoid_derivative(self.hidden_output)

        self.weights_hidden_output += self.hidden_output.T.dot(d_output) * learning_rate
        self.bias_output += np.sum(d_output, axis=0, keepdims=True) * learning_rate

        self.weights_input_hidden += X.T.dot(d_hidden) * learning_rate
        self.bias_hidden += np.sum(d_hidden, axis=0, keepdims=True) * learning_rate

    def train(self, X, y, epochs, learning_rate):
```

```
        for epoch in range(epochs):
            output = self.forwardpass(X)
            self.backwardpass(X, y, output, learning_rate)

            if epoch % 1000 == 0:
                loss = np.mean(0.5 * (y - output) ** 2)
                print(f"Epoch {epoch}, Loss: {loss:.4f}")
```

In [59]:

```
X = np.array([[0, 0, 1],
              [0, 1, 1],
              [1, 0, 1],
              [1, 1, 1]])

y = np.array([[0],
              [1],
              [1],
              [0]])
```

In [60]:

```
nn = NeuralNetwork(input_size=3, hidden_size=4, output_size=1)
```

In [61]:

```
nn.train(X, y, epochs=10000, learning_rate=0.1)
```

```
Epoch 0, Loss: 0.1493
Epoch 1000, Loss: 0.0863
Epoch 2000, Loss: 0.0206
Epoch 3000, Loss: 0.0070
Epoch 4000, Loss: 0.0037
Epoch 5000, Loss: 0.0024
Epoch 6000, Loss: 0.0018
Epoch 7000, Loss: 0.0014
Epoch 8000, Loss: 0.0011
Epoch 9000, Loss: 0.0009
```

In [62]:

```
print("\nFinal predictions:")
print(nn.forwardpass(X))
```

```
Final predictions:
[[0.03430752]
 [0.94794497]
 [0.97425978]
 [0.04423091]]
```

In [63]:
```python
classified_output = (nn.forwardpass(X) > 0.5).astype(int)
print("\nClassified Output after training (0 or 1):")
print(classified_output)
```

```
Classified Output after training (0 or 1):
[[0]
 [1]
 [1]
 [0]]
```

In [ ]: