DS505: INTRODUCTION TO DEEP LEARNING

P02: Artificial Neural Network

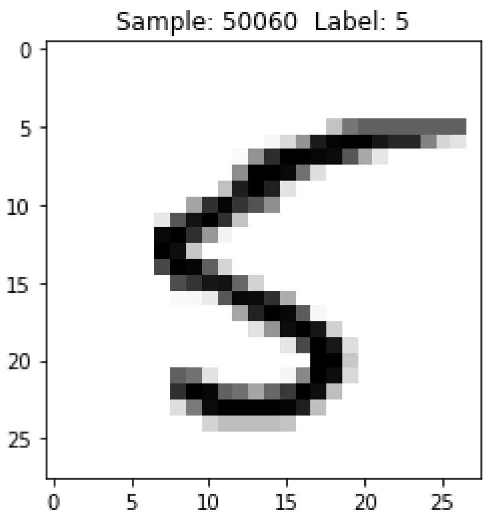
In [1]:
```
import tensorflow as tf

a = tf.Variable(1, name="a")
b = tf.Variable(2, name="b")
f = a + b

tf.print("The sum of a and b is", f)
```

```
The sum of a and b is 3
```

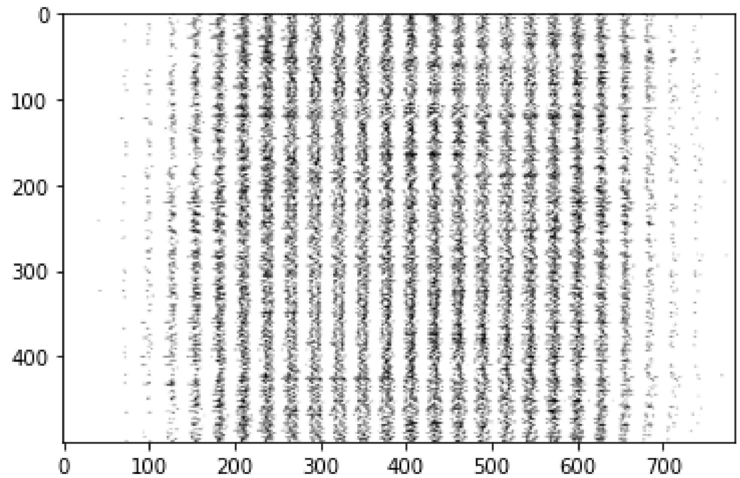
Handwriting Recognisation

In [2]:
```
# Prepare MNIST data.
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist

# MNIST dataset parameters
num_classes = 10 # total classes (0-9 digits)
num_features = 784 # data features (img shape: 28*28)

(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Convert to float32
x_train, x_test = np.array(x_train, np.float32), np.array(x_test, np.float32)

# Flatten images to 1-D vector of 784 features (28*28)
x_train, x_test = x_train.reshape([-1, num_features]), x_test.reshape([-1, num_fe

# Normalize images value from [0, 255] to [0, 1]
x_train, x_test = x_train / 255., x_test / 255.
```

In [3]:
```
%matplotlib inline

import matplotlib.pyplot as plt

def display_sample(num):
    #Print this sample's label
    label = y_train[num]

    #Reshape the 784 values to a 28x28 image
    image = x_train[num].reshape([28,28])
    plt.title('Sample: %d  Label: %d' % (num, label))
    plt.imshow(image, cmap=plt.get_cmap('gray_r'))
    plt.show()

display_sample(50060)
```



In [4]:
```
images = x_train[0].reshape([1,784])
for i in range(1, 500):
    images = np.concatenate((images, x_train[i].reshape([1,784])))
plt.imshow(images, cmap=plt.get_cmap('gray_r'))
plt.show()
```



In [5]:
```
# Training parameters.
learning_rate = 0.001
training_steps = 3000
batch_size = 250
display_step = 100

# Network parameters.
n_hidden = 512 # Number of neurons.
```

In [6]:
```
# Use tf.data API to shuffle and batch data.
train_data = tf.data.Dataset.from_tensor_slices((x_train, y_train))
train_data = train_data.repeat().shuffle(60000).batch(batch_size).prefetch(1)
```

In [7]:
```
# Store layers weight & bias

# A random value generator to initialize weights initially
random_normal = tf.initializers.RandomNormal()

weights = {
    'h': tf.Variable(random_normal([num_features, n_hidden])),
    'out': tf.Variable(random_normal([n_hidden, num_classes]))
}
biases = {
    'b': tf.Variable(tf.zeros([n_hidden])),
    'out': tf.Variable(tf.zeros([num_classes]))
}
```

```
C:\Users\Alaissa\anaconda3\lib\site-packages\keras\initializers\initializers_v
2.py:120: UserWarning: The initializer RandomNormal is unseeded and being calle
d multiple times, which will return identical values  each time (even if the in
itializer is unseeded). Please update your code to provide a seed to the initia
lizer, or avoid using the same initalizer instance more than once.
  warnings.warn(
```

In [8]:
```
# Create model.
def neural_net(inputData):
    # Hidden fully connected layer with 512 neurons.
    hidden_layer = tf.add(tf.matmul(inputData, weights['h']), biases['b'])
    # Apply sigmoid to hidden_layer output for non-linearity.
    hidden_layer = tf.nn.sigmoid(hidden_layer)

    # Output fully connected layer with a neuron for each class.
    out_layer = tf.matmul(hidden_layer, weights['out']) + biases['out']
    # Apply softmax to normalize the logits to a probability distribution.
    return tf.nn.softmax(out_layer)
```

In [9]:
```
def cross_entropy(y_pred, y_true):
    # Encode label to a one hot vector.
    y_true = tf.one_hot(y_true, depth=num_classes)
    # Clip prediction values to avoid log(0) error.
    y_pred = tf.clip_by_value(y_pred, 1e-9, 1.)
    # Compute cross-entropy.
    return tf.reduce_mean(-tf.reduce_sum(y_true * tf.math.log(y_pred)))
```

In [10]:
```
optimizer = tf.keras.optimizers.SGD(learning_rate)

def run_optimization(x, y):
    # Wrap computation inside a GradientTape for automatic differentiation.
    with tf.GradientTape() as g:
        pred = neural_net(x)
        loss = cross_entropy(pred, y)

    # Variables to update, i.e. trainable variables.
    trainable_variables = list(weights.values()) + list(biases.values())

    # Compute gradients.
    gradients = g.gradient(loss, trainable_variables)

    # Update W and b following gradients.
    optimizer.apply_gradients(zip(gradients, trainable_variables))
```

In [11]:
```
# Accuracy metric.
def accuracy(y_pred, y_true):
    # Predicted class is the index of highest score in prediction vector (i.e. ar
    correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.cast(y_true, tf.int64)
    return tf.reduce_mean(tf.cast(correct_prediction, tf.float32), axis=-1)
```

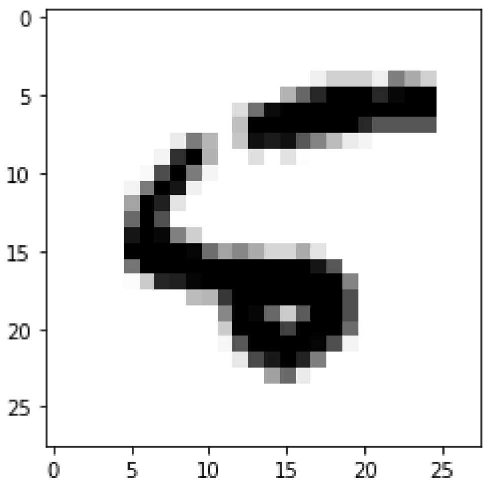
In [12]:
```
# Run training for the given number of steps.
for step, (batch_x, batch_y) in enumerate(train_data.take(training_steps), 1):
    # Run the optimization to update W and b values.
    run_optimization(batch_x, batch_y)

    if step % display_step == 0:
        pred = neural_net(batch_x)
        loss = cross_entropy(pred, batch_y)
        acc = accuracy(pred, batch_y)
        print("Training epoch: %i, Loss: %f, Accuracy: %f" % (step, loss, acc))
```

```
Training epoch: 100, Loss: 168.700500, Accuracy: 0.808000
Training epoch: 200, Loss: 130.224243, Accuracy: 0.868000
Training epoch: 300, Loss: 77.815361, Accuracy: 0.916000
Training epoch: 400, Loss: 84.074432, Accuracy: 0.912000
Training epoch: 500, Loss: 83.447197, Accuracy: 0.912000
Training epoch: 600, Loss: 72.608917, Accuracy: 0.916000
Training epoch: 700, Loss: 90.761040, Accuracy: 0.904000
Training epoch: 800, Loss: 79.992157, Accuracy: 0.912000
Training epoch: 900, Loss: 73.183571, Accuracy: 0.920000
Training epoch: 1000, Loss: 95.224503, Accuracy: 0.892000
Training epoch: 1100, Loss: 74.952393, Accuracy: 0.904000
Training epoch: 1200, Loss: 64.661255, Accuracy: 0.924000
Training epoch: 1300, Loss: 70.087814, Accuracy: 0.916000
Training epoch: 1400, Loss: 45.076416, Accuracy: 0.956000
Training epoch: 1500, Loss: 92.095306, Accuracy: 0.892000
Training epoch: 1600, Loss: 55.825661, Accuracy: 0.944000
Training epoch: 1700, Loss: 58.137417, Accuracy: 0.904000
Training epoch: 1800, Loss: 62.384689, Accuracy: 0.932000
Training epoch: 1900, Loss: 51.687489, Accuracy: 0.932000
Training epoch: 2000, Loss: 69.218071, Accuracy: 0.928000
Training epoch: 2100, Loss: 72.990067, Accuracy: 0.932000
Training epoch: 2200, Loss: 68.935272, Accuracy: 0.920000
Training epoch: 2300, Loss: 48.394333, Accuracy: 0.932000
Training epoch: 2400, Loss: 60.100368, Accuracy: 0.924000
Training epoch: 2500, Loss: 56.140739, Accuracy: 0.944000
Training epoch: 2600, Loss: 44.652145, Accuracy: 0.948000
Training epoch: 2700, Loss: 45.617447, Accuracy: 0.948000
Training epoch: 2800, Loss: 52.635300, Accuracy: 0.924000
Training epoch: 2900, Loss: 66.669724, Accuracy: 0.944000
Training epoch: 3000, Loss: 73.622437, Accuracy: 0.908000
```

In [13]:
```
# Test model on validation set.
pred = neural_net(x_test)
print("Test Accuracy: %f" % accuracy(pred, y_test))
```

```
Test Accuracy: 0.932400
```

In [14]:
```
n_images = 200
test_images = x_test[:n_images]
test_labels = y_test[:n_images]
predictions = neural_net(test_images)

for i in range(n_images):
    model_prediction = np.argmax(predictions.numpy()[i])
    if (model_prediction != test_labels[i]):
        plt.imshow(np.reshape(test_images[i], [28, 28]), cmap='gray_r')
        plt.show()
        print("Original Labels: %i" % test_labels[i])
        print("Model prediction: %i" % model_prediction)
```




```
Original Labels: 5
Model prediction: 6
```