Neural network demo

In [1]:
```
import numpy as np
import mnist
import simple_nn as nn
```

In [2]:
```
# The network hasn't been trained yet. There is no reason for it having a high accuracy.
nn.accuracy()
```

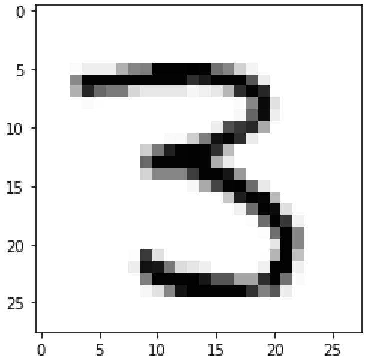
Out[2]: 0.0958

In [3]:
```
# We train the neural network. This time the accuracy should be closer to 1.
nn.train()
nn.accuracy()
```

Out[3]: 0.8752

Lets take a random image from the test dataset and see if the network recognizes correctly the handwritten digit. The matplotlib.pyplot module is used here only to visualize the image

In [4]:
```
import matplotlib.pyplot as plt
idx = np.random.randint(10000) # A random integer between 0 and 9999
image = mnist.test_images()[idx]
label = mnist.test_labels()[idx]
plt.imshow(image, 'Greys')
plt.show()
print('Label =', label)
```




```
Label = 3
```

Now lets see what the output for that image is.

In [5]:
```
x, h, ha, y, ya = nn.forward_propagation(image)
print('Output =', ya.round(3))
```

```
Output = [0.002 0.    0.002 0.973 0.    0.02  0.    0.    0.002 0.001]
```

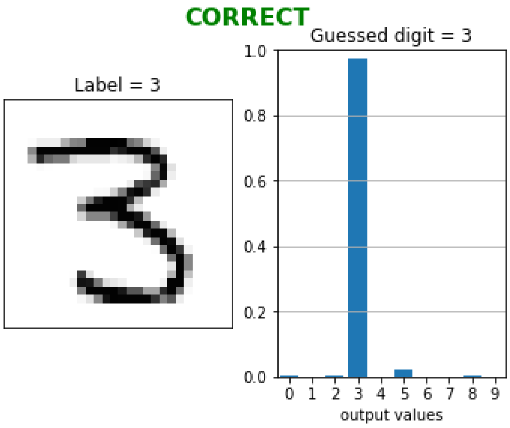
We get 10 values (each between 0 and 1) for each digit in ascending order from 0 to 9. The index of the highest value is the **guessed digit**: the digit that the network considers as the most likley to be on the image.

In [6]:
```
print('Guessed digit =', ya.argmax())
```

```
Guessed digit = 3
```

Lets build a bar chart to visulalise the output data.

In [7]:
```
plt.figure(idx)
if label == ya.argmax():
    plt.suptitle('CORRECT', color='g', fontsize=16, fontweight='bold')
else:
    plt.suptitle('INCORRECT', color='r', fontsize=16, fontweight='bold')
plt.subplot(121)
plt.imshow(image, 'Greys')
plt.xticks([])
plt.yticks([])
plt.title('Label = ' + str(label))
plt.subplot(122)
plt.bar(range(10), ya)
plt.xticks(range(10), range(10))
plt.ylim(0, 1)
plt.xlim(-.5, 9.5)
plt.grid(axis='y')
plt.xlabel('output values')
plt.title('Guessed digit = ' + str(ya.argmax()))
plt.show()
```



Training the network

Forward propagation

For a given input $\mathbf{x}$ (flattened and normalized image of 784 unicolor pixels) the output $\mathbf{y_a}$ (10 estimations of probability for each digit) is obtained as follows:

$$\mathbf{h} = \mathbf{x} \cdot \mathbf{W_1} + \mathbf{b_1} \qquad (1) \qquad (1)$$

$$\mathbf{h_a} = S(\mathbf{h}) = \frac{1}{1 + e^{-\mathbf{h}}} \qquad (2) \qquad (2)$$

$$\mathbf{y} = \mathbf{h_a} \cdot \mathbf{W_2} + \mathbf{b_2} \qquad (3) \qquad (3)$$

$$\mathbf{y_a} = \sigma(\mathbf{y}) = \frac{e^{\mathbf{y}}}{\sum_{i=0}^{9} e^{(\mathbf{y})_i}} \qquad (4) \qquad (4)$$

Where

- $\mathbf{x} \in [0,1]^{784}$ is the value of the input layer (composed of 784 neurons).
- $\mathbf{h} \in \mathbb{R}^{16}$ and $\mathbf{h_a} \in]0,1[^{16}$ are the values the hidden layer (16 neurons) before and after activation by the sigmoid function S.
- $\mathbf{y} \in \mathbb{R}^{10}$ and $\mathbf{y_a} \in]0,1[^{10}$ are the values of the output layer (10 neurons) before and after activation by the sofmax function σ.
- $\mathbf{W_1} \in \mathcal{M}_{784,16}(\mathbb{R})$ and $\mathbf{W_2} \in \mathcal{M}_{16,10}(\mathbb{R})$ are the weight matrices.
- $\mathbf{b_1} \in \mathbb{R}^{16}$ and $\mathbf{b_2} \in \mathbb{R}^{10}$ are the bias terms.

Loss function

The cross entropy loss function L is used to update the weights and biases of the network. It is calculated using the output $\mathbf{y_a}$ and the target $\mathbf{t}$ (a one hot vector obtained from the $label \in \{0,1,\dots,9\}$ of the input image).

$$\begin{aligned} L(\mathbf{t}, \mathbf{y_a}) &= -\sum_{i=0}^{9} (\mathbf{t})_i \cdot \log(\mathbf{y_a})_i && (5) \\ &= -\log(\mathbf{y_a})_{label} \qquad (5) && (6) \end{aligned}$$

Backpropagation

In order to update each of the learnable parameters (weights and biases) we need to calculate the gradient ∇L of the loss function:

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial \mathbf{b_1}} \\ \frac{\partial L}{\partial \mathbf{W_1}} \\ \frac{\partial L}{\partial \mathbf{b_2}} \\ \frac{\partial L}{\partial \mathbf{W_2}} \end{bmatrix} \qquad (6) \qquad (7)$$

Before we start, here is a quick reminder of the derivative of the sofmax function σ:

$$\frac{\partial \sigma}{\partial (\mathbf{y})_i}(\mathbf{y})_{label} = \begin{cases} (\mathbf{y_a})_{label} \cdot (1 - (\mathbf{y_a})_{label}) & \text{if } i = label \\ (\mathbf{y_a})_{label} \cdot (\mathbf{y_a})_i & \text{else} \end{cases} \qquad (7) \qquad (8)$$

Let's calculate the derivatives with respect to each learning parameters of the network: $\frac{\partial L}{\partial \mathbf{b_2}}$, $\frac{\partial L}{\partial \mathbf{W_2}}$, $\frac{\partial L}{\partial \mathbf{b_1}}$ and $\frac{\partial L}{\partial \mathbf{W_1}}$.

$$
\begin{aligned}
\frac{\partial L}{\partial \mathbf{b_2}} &= \frac{\partial L}{\partial \mathbf{y_a}} \cdot \frac{\partial \mathbf{y_a}}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{b_2}} && \tag{9} \\
&= \frac{-1}{(\mathbf{y_a})_{label}} \cdot \frac{\partial \sigma}{\partial \mathbf{y}}(\mathbf{y})_{label} && \tag{10} \\
&= \mathbf{y_a} - \mathbf{t} & (8) & \tag{11} \\
& && \tag{12} \\
\frac{\partial L}{\partial \mathbf{W_2}} &= \frac{\partial L}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial \mathbf{W_2}} && \tag{13} \\
&= \mathbf{h_a}^T \frac{\partial L}{\partial \mathbf{b_2}} & (9) & \tag{14} \\
& && \tag{15} \\
\frac{\partial L}{\partial \mathbf{b_1}} &= \frac{\partial L}{\partial \mathbf{h_a}} \cdot \frac{\partial \mathbf{h_a}}{\partial \mathbf{b_1}} && \tag{16} \\
&= \mathbf{W_2}\left(\frac{\partial L}{\partial \mathbf{b_2}}\right)^T \cdot \frac{\partial S}{\partial \mathbf{h}}(\mathbf{h}) && \tag{17} \\
&= \mathbf{W_2}\left(\frac{\partial L}{\partial \mathbf{b_2}}\right)^T \cdot \mathbf{h_a} \cdot (1 - \mathbf{h_a}) & (10) & \tag{18} \\
& && \tag{19} \\
\frac{\partial L}{\partial \mathbf{W_1}} &= \frac{\partial L}{\partial \mathbf{h}} \cdot \frac{\partial \mathbf{h}}{\partial \mathbf{W_1}} && \tag{20} \\
&= \mathbf{x}^T \frac{\partial L}{\partial \mathbf{b_1}} & (11) & \tag{21}
\end{aligned}
$$

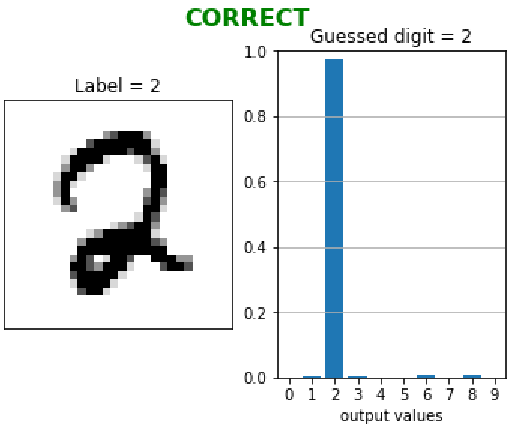
Where ^{T} is the transpose operator. In our case for a any vector $\mathbf{v}^T$ is a column vector.

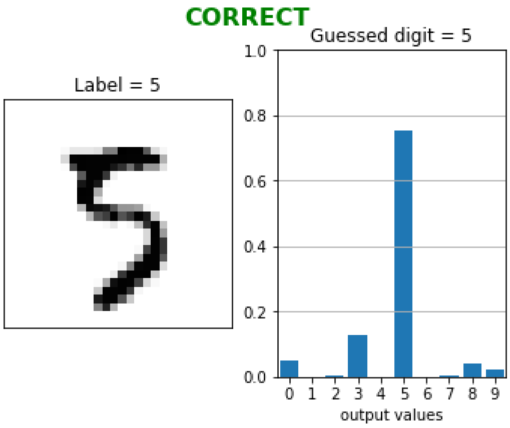
Note that $\frac{\partial \mathbf{y}}{\partial \mathbf{b_2}} = \mathbf{1}_{\mathbb{R}^{10}}$ and $\frac{\partial \mathbf{h}}{\partial \mathbf{b_1}} = \mathbf{1}_{\mathbb{R}^{16}}$, so $\frac{\partial \mathbf{y_a}}{\partial \mathbf{b_2}} = \frac{\partial \mathbf{y_a}}{\partial \mathbf{y}}$ and $\frac{\partial \mathbf{h_a}}{\partial \mathbf{b_1}} = \frac{\partial \mathbf{h_a}}{\partial \mathbf{h}}$.

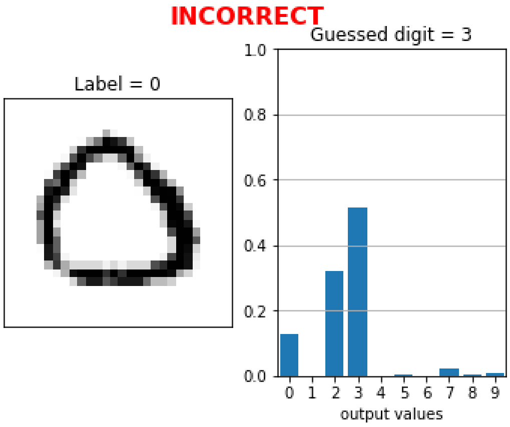
More testing

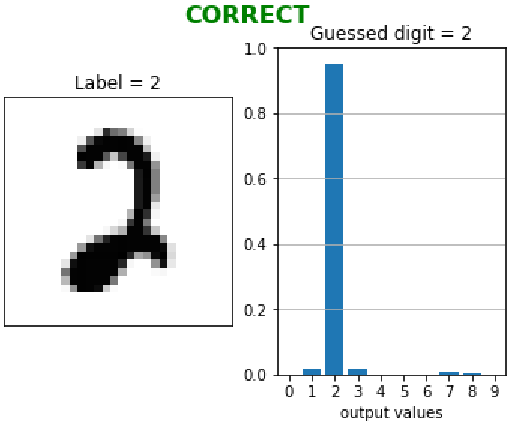
In [12]:

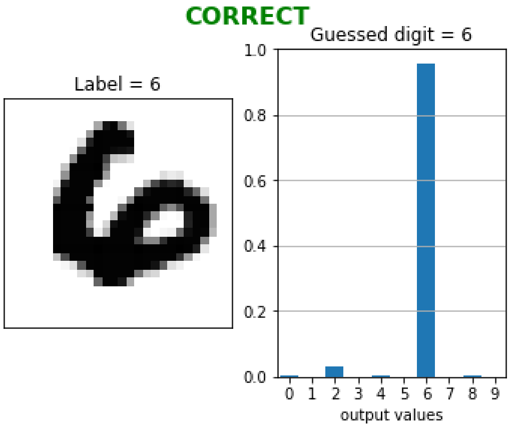
```
import matplotlib.pyplot as plt

n_examples = 10
image_ids = np.random.permutation(np.arange(10000))[:n_examples] # Random numbers between 0 and 9999

for id in image_ids:
    image = nn.test_images[id]
    label = nn.test_labels[id]
    x, h, ha, y, ya = nn.forward_propagation(image)
    plt.figure(id)
    if label == ya.argmax():
        plt.suptitle('CORRECT', color='g', fontsize=16, fontweight='bold')
    else:
        plt.suptitle('INCORRECT', color='r', fontsize=16, fontweight='bold')
    plt.subplot(121)
    plt.imshow(image, 'Greys')
    plt.xticks([])
    plt.yticks([])
    plt.title('Label = ' + str(label))
    plt.subplot(122)
    plt.bar(range(10), ya)
    plt.xticks(range(10), range(10))
    plt.ylim(0, 1)
    plt.xlim(-.5, 9.5)
    plt.grid(axis='y')
    plt.xlabel('output values')
    plt.title('Guessed digit = ' + str(ya.argmax()))
    plt.show()
```

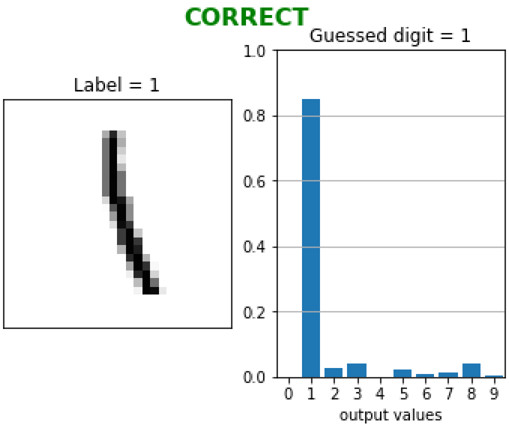


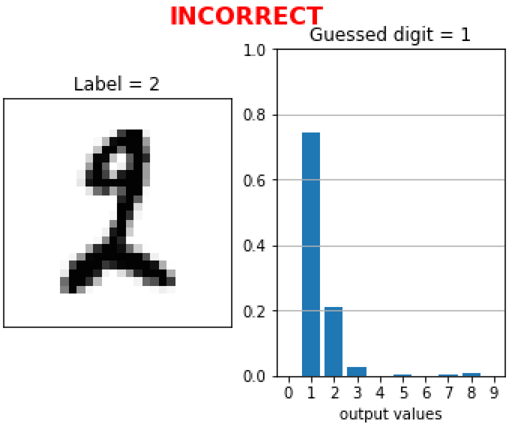

CORRECT
Guessed digit = 5
Label = 5
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
output values

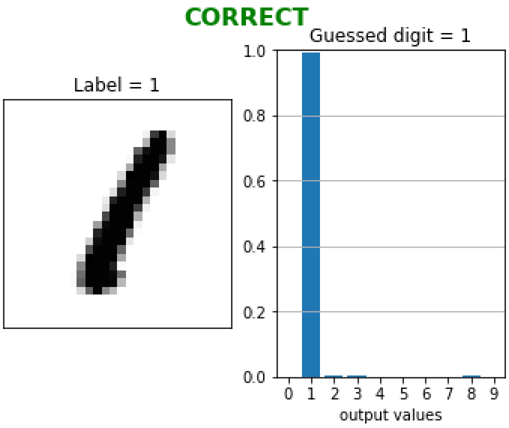

INCORRECT
Guessed digit = 3
Label = 0
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
output values

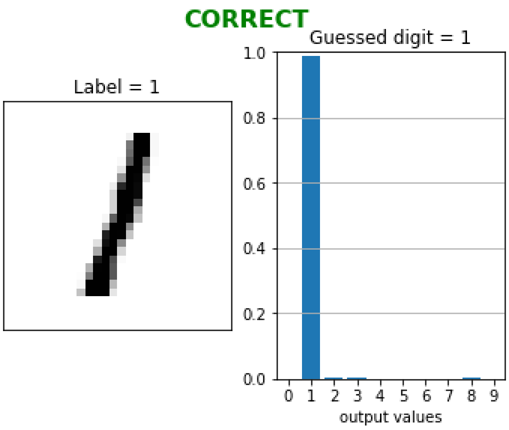

CORRECT
Guessed digit = 2
Label = 2
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
output values

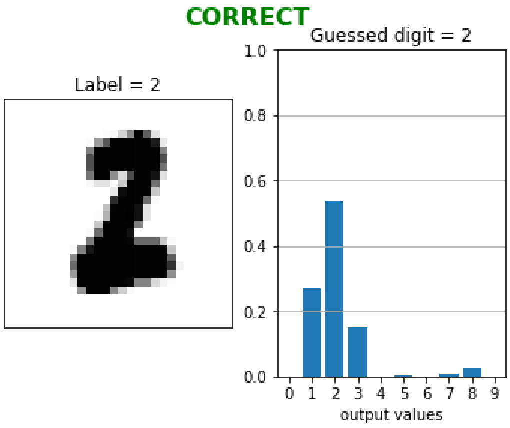

CORRECT
Guessed digit = 6
Label = 6
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
output values


CORRECT
Label = 1
Guessed digit = 1
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
output values


INCORRECT
Label = 2
Guessed digit = 1
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
output values


CORRECT
Label = 1
Guessed digit = 1
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
output values


CORRECT
Label = 1
Guessed digit = 1
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
output values


CORRECT
Label = 2
Guessed digit = 2
1.0
0.8
0.6
0.4
0.2
0.0
0 1 2 3 4 5 6 7 8 9
output values

In [ ]: