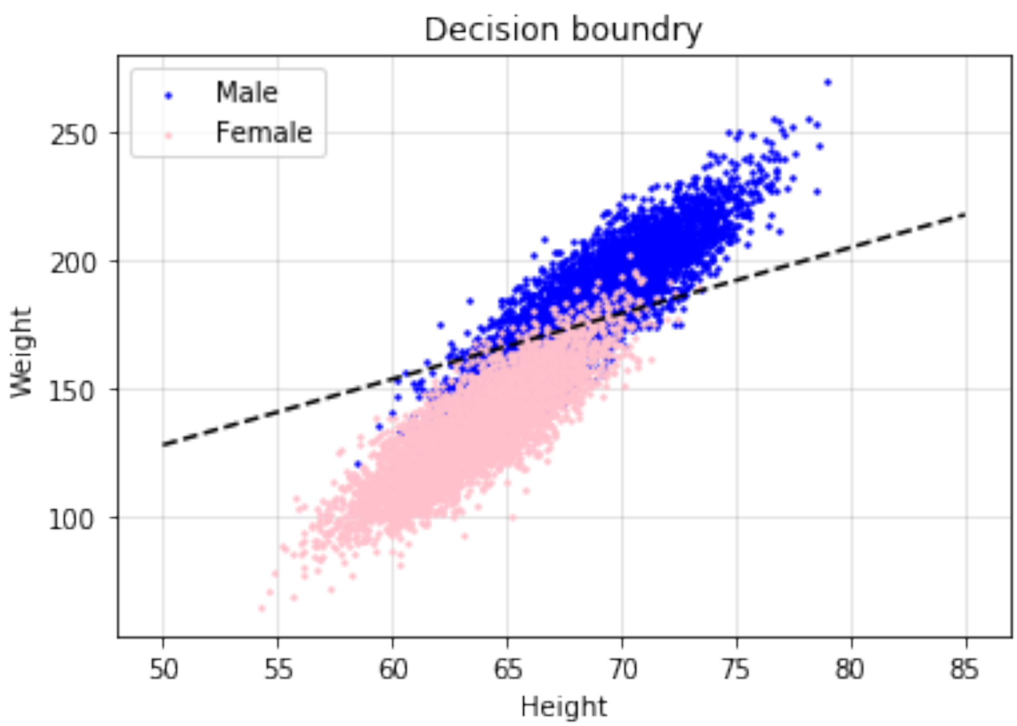
```
draw_plot(input, w)

percentage = 0
for person in data:
    out = activation(person[1:3], w, threshold, bias)
    if out == person[0]:
        percentage = percentage + 1

print("accuracy", percentage / 10000)
```

```
[ 0.86939397 -0.72755562]
```




```
accuracy 0.8677
```

3 Design a MLP neural network using keras to learn MNIST dataset

In this question we had to design a MLP neural network using keras to learn MNIST dataset

Below are the packages used in this question.

[12]:
```
import keras
from keras.datasets import mnist
```

```python
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

Here the dataset is loaded into the program, reshaped and normalized.

[13]:
```python
(x_train, y_train), (x_test, y_test) = mnist.load_data("/Users/alireza/Desktop/
 ↪CI/mnist.npz")
x_train = np.array(x_train).reshape(60000,784)
x_test = np.array(x_test).reshape(10000,784)
x_train = keras.utils.normalize(x_train)
x_test = keras.utils.normalize(x_test)
```

Our model in this question is a *sequential* model and has 2 dense(fully connected) hidden layers each with *ReLU* activation function and 256 and 128 neurons respectively.

Output layer is also a dense layer with 10 neurons and softmax activation function in order to use one hot encoding.

Stochastic Gradient Descent algorithm is used as optimizer with learning rate = 0.1.

decay is the rate of lowering the learning rate in each iteration.

momentum is the parameter that accelerates SGD in the relevant direction and dampens oscillations.

nesterov is a boolean. Whether to apply Nesterov momentum.

The loss funtion is *sparse_categorical_crossentropy* and *accuracy* is used for measuring the network.

The model is trained in 10 epochs and then tested against the test data.

[14]:
```python
model = Sequential()
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
sgd = keras.optimizers.sgd(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(optimizer=sgd, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(x_train, y_train, validation_data=(x_test,y_test), epochs=10)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 3s 55us/step - loss: 0.3032 -
accuracy: 0.9055 - val_loss: 0.1550 - val_accuracy: 0.9505
Epoch 2/10
60000/60000 [==============================] - 3s 46us/step - loss: 0.1315 -
accuracy: 0.9594 - val_loss: 0.1096 - val_accuracy: 0.9642
Epoch 3/10
```

```
60000/60000 [==============================] - 3s 44us/step - loss: 0.0992 -
accuracy: 0.9686 - val_loss: 0.0960 - val_accuracy: 0.9692
Epoch 4/10
60000/60000 [==============================] - 3s 54us/step - loss: 0.0801 -
accuracy: 0.9738 - val_loss: 0.0804 - val_accuracy: 0.9749
Epoch 5/10
60000/60000 [==============================] - 3s 53us/step - loss: 0.0678 -
accuracy: 0.9786 - val_loss: 0.0793 - val_accuracy: 0.9738
Epoch 6/10
60000/60000 [==============================] - 3s 47us/step - loss: 0.0566 -
accuracy: 0.9821 - val_loss: 0.0913 - val_accuracy: 0.9732
Epoch 7/10
60000/60000 [==============================] - 3s 48us/step - loss: 0.0519 -
accuracy: 0.9831 - val_loss: 0.0763 - val_accuracy: 0.9770
Epoch 8/10
60000/60000 [==============================] - 3s 51us/step - loss: 0.0447 -
accuracy: 0.9852 - val_loss: 0.0829 - val_accuracy: 0.9761
Epoch 9/10
60000/60000 [==============================] - 3s 48us/step - loss: 0.0399 -
accuracy: 0.9873 - val_loss: 0.0866 - val_accuracy: 0.9771
Epoch 10/10
60000/60000 [==============================] - 3s 48us/step - loss: 0.0337 -
accuracy: 0.9887 - val_loss: 0.0635 - val_accuracy: 0.9816
```

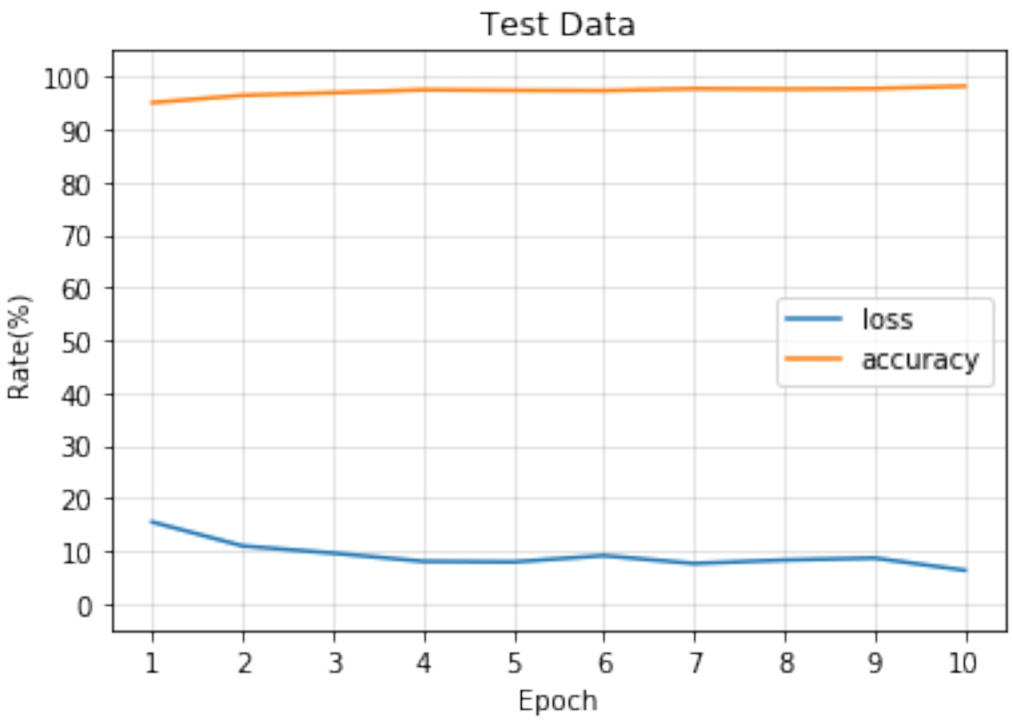
```
[14]: <keras.callbacks.callbacks.History at 0x13f576c88>
```

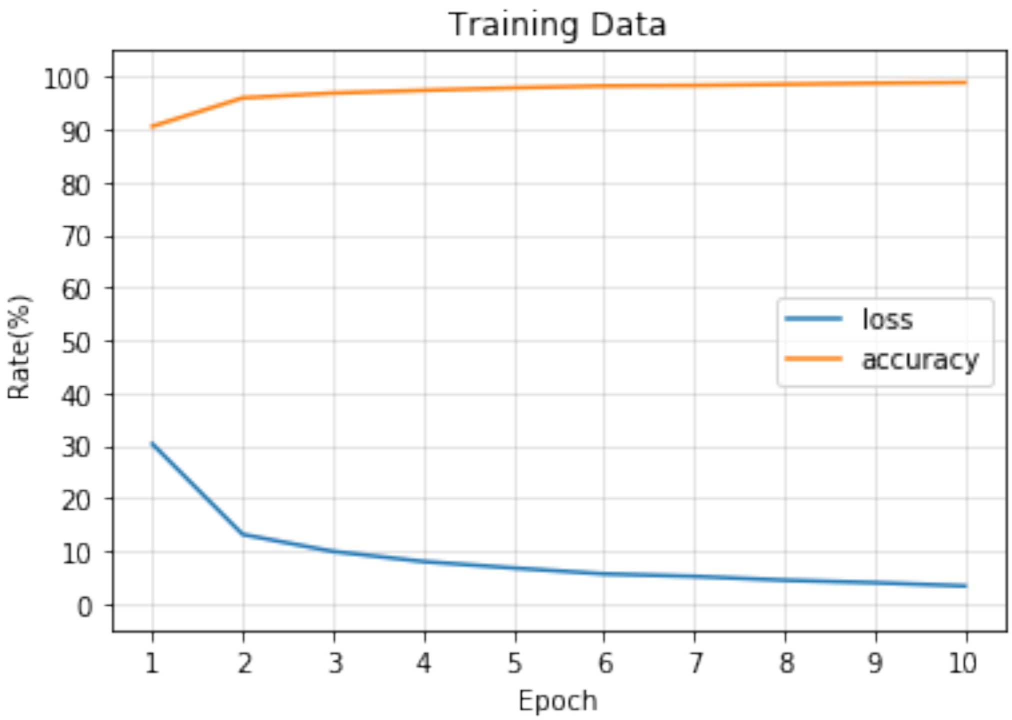
Below code is used to plot training and test data loss and accuracy.

```
[15]: fig, ax = plt.subplots()
x = range(1,len(model.history.history['loss'])+1)
y = range(0,101,10)
ax.grid(color='gray', alpha=0.25)
ax.set_axisbelow(True)
plt.title("Test Data")
plt.xlabel("Epoch")
plt.ylabel("Rate(%)")
plt.xticks(x)
plt.yticks(y)
plt.ylim(-5,105)
ax.plot(x,100*np.array(model.history.history['val_loss']), label= 'loss')
ax.plot(x,100*np.array(model.history.history['val_accuracy']), label='accuracy')
ax.legend()
plt.savefig("Q4-test.png")
plt.show()

fig, ax = plt.subplots()
ax.grid(color='gray', alpha=0.25)
ax.set_axisbelow(True)
```

```
plt.title("Training Data")
plt.xlabel("Epoch")
plt.ylabel("Rate(%)")
plt.xticks(x)
plt.yticks(y)
plt.ylim(-5,105)
ax.plot(x,100*np.array(model.history.history['loss']), label= 'loss')
ax.plot(x,100*np.array(model.history.history['accuracy']), label='accuracy')
ax.legend()
plt.savefig("Q4-train.png")
plt.show()
```





Finally the model is evaluated with the test data and the result is reported below.

[16]:
```
test_loss, test_acc = model.evaluate(x_test, y_test)
test_acc
```

```
10000/10000 [==============================] - 0s 18us/step
```

[16]:
```
0.9815999865531921
```

4 Add drop-out to the previous question

The code and network design is the same as previous question except there are 2 *drop-outs* each with 35% chance.

Drop-out prevents the network from *over-fitting*.

[17]:
```
import keras
from keras.datasets import mnist
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Dropout
```

[18]:

```
model = Sequential()
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.35))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.35))
model.add(Dense(10, activation='softmax'))
sgd = keras.optimizers.sgd(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(optimizer=sgd, loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.fit(x_train, y_train, validation_data=(x_test,y_test), epochs=10)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 3s 57us/step - loss: 0.4540 -
accuracy: 0.8577 - val_loss: 0.1789 - val_accuracy: 0.9443
Epoch 2/10
60000/60000 [==============================] - 3s 54us/step - loss: 0.2699 -
accuracy: 0.9190 - val_loss: 0.1553 - val_accuracy: 0.9498
Epoch 3/10
60000/60000 [==============================] - 3s 56us/step - loss: 0.2258 -
accuracy: 0.9326 - val_loss: 0.1306 - val_accuracy: 0.9614
Epoch 4/10
60000/60000 [==============================] - 3s 56us/step - loss: 0.2021 -
accuracy: 0.9401 - val_loss: 0.1202 - val_accuracy: 0.9614
Epoch 5/10
60000/60000 [==============================] - 3s 56us/step - loss: 0.1852 -
accuracy: 0.9457 - val_loss: 0.1061 - val_accuracy: 0.9674
Epoch 6/10
60000/60000 [==============================] - 4s 66us/step - loss: 0.1759 -
accuracy: 0.9468 - val_loss: 0.1063 - val_accuracy: 0.9687
Epoch 7/10
60000/60000 [==============================] - 4s 72us/step - loss: 0.1636 -
accuracy: 0.9508 - val_loss: 0.1075 - val_accuracy: 0.9684
Epoch 8/10
60000/60000 [==============================] - 4s 63us/step - loss: 0.1582 -
accuracy: 0.9528 - val_loss: 0.0966 - val_accuracy: 0.9703
Epoch 9/10
60000/60000 [==============================] - 3s 55us/step - loss: 0.1476 -
accuracy: 0.9554 - val_loss: 0.0988 - val_accuracy: 0.9698
Epoch 10/10
60000/60000 [==============================] - 3s 54us/step - loss: 0.1415 -
accuracy: 0.9589 - val_loss: 0.0952 - val_accuracy: 0.9708
```

[18]:

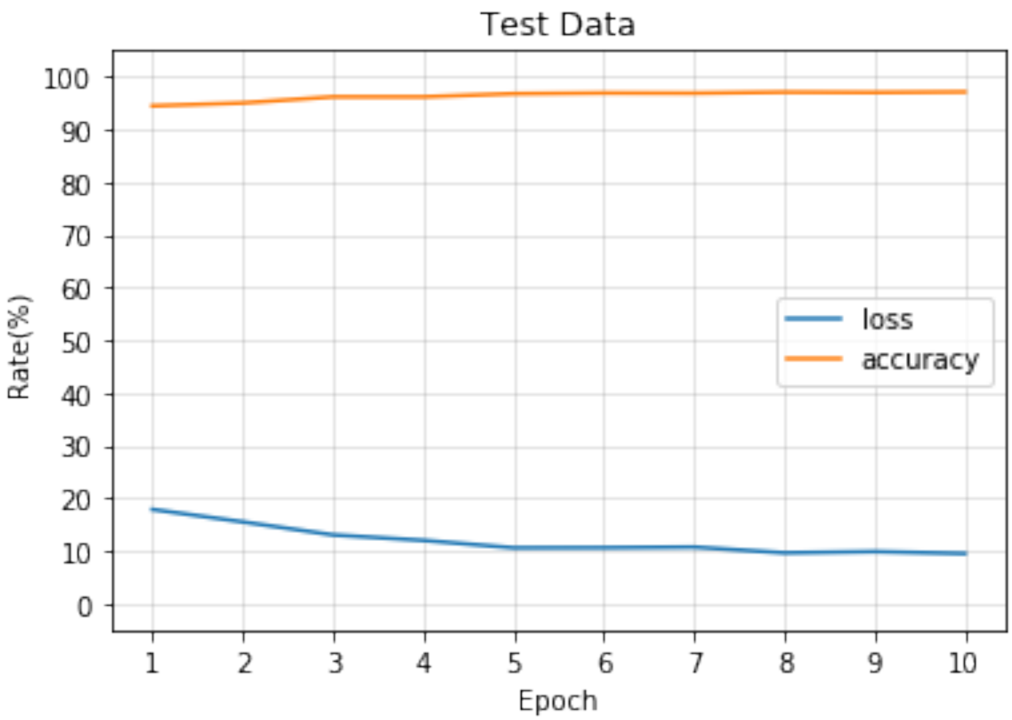
```
<keras.callbacks.callbacks.History at 0x13cd9bf60>
```

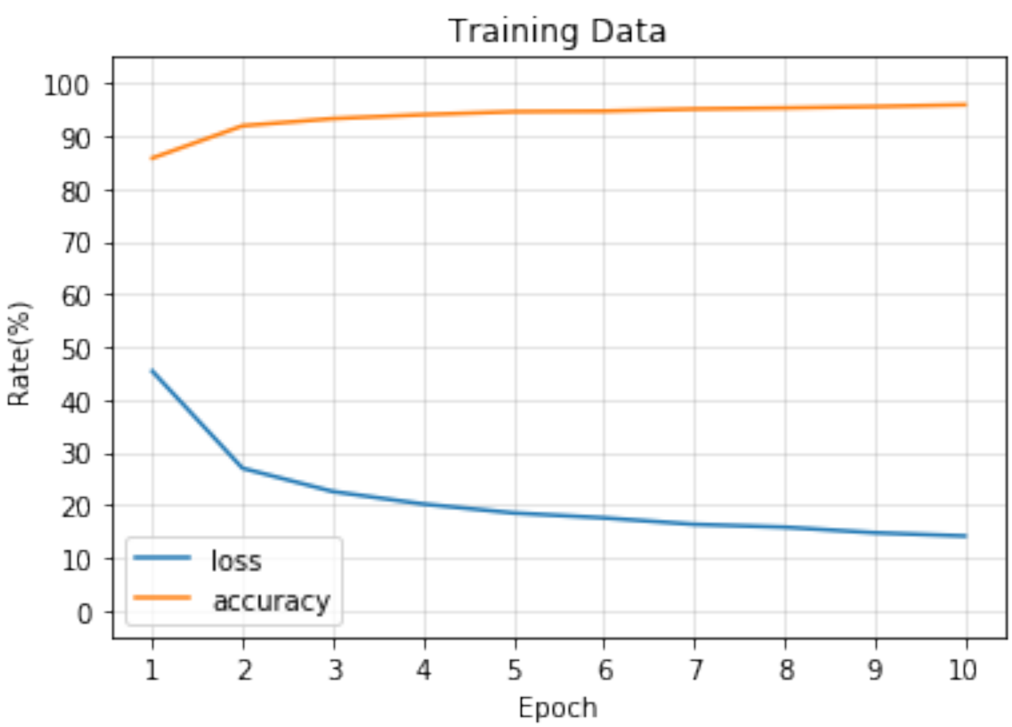
The rest of the code is the same as previous question.

As you can see because this network does not over-fit the final results are more or less the same as previous question.

[19]:
```
fig, ax = plt.subplots()
x = range(1,len(model.history.history['loss'])+1)
y = range(0,101,10)
ax.grid(color='gray', alpha=0.25)
ax.set_axisbelow(True)
plt.title("Test Data")
plt.xlabel("Epoch")
plt.ylabel("Rate(%)")
plt.xticks(x)
plt.yticks(y)
plt.ylim(-5,105)
ax.plot(x,100*np.array(model.history.history['val_loss']), label= 'loss')
ax.plot(x,100*np.array(model.history.history['val_accuracy']), label='accuracy')
ax.legend()
plt.savefig("Q4-test-dropout.png")
plt.show()

fig, ax = plt.subplots()
ax.grid(color='gray', alpha=0.25)
ax.set_axisbelow(True)
plt.title("Training Data")
plt.xlabel("Epoch")
plt.ylabel("Rate(%)")
plt.xticks(x)
plt.yticks(y)
plt.ylim(-5,105)
ax.plot(x,100*np.array(model.history.history['loss']), label= 'loss')
ax.plot(x,100*np.array(model.history.history['accuracy']), label='accuracy')
ax.legend()
plt.savefig("Q4-train-dropout.png")
plt.show()
test_loss, test_acc = model.evaluate(x_test, y_test)
test_acc
```


Test Data
Rate(%)
Epoch
loss
accuracy
0
10
20
30
40
50
60
70
80
90
100
1
2
3
4
5
6
7
8
9
10


Training Data
Rate(%)
Epoch
loss
accuracy
0
10
20
30
40
50
60
70
80
90
100
1
2
3
4
5
6
7
8
9
10