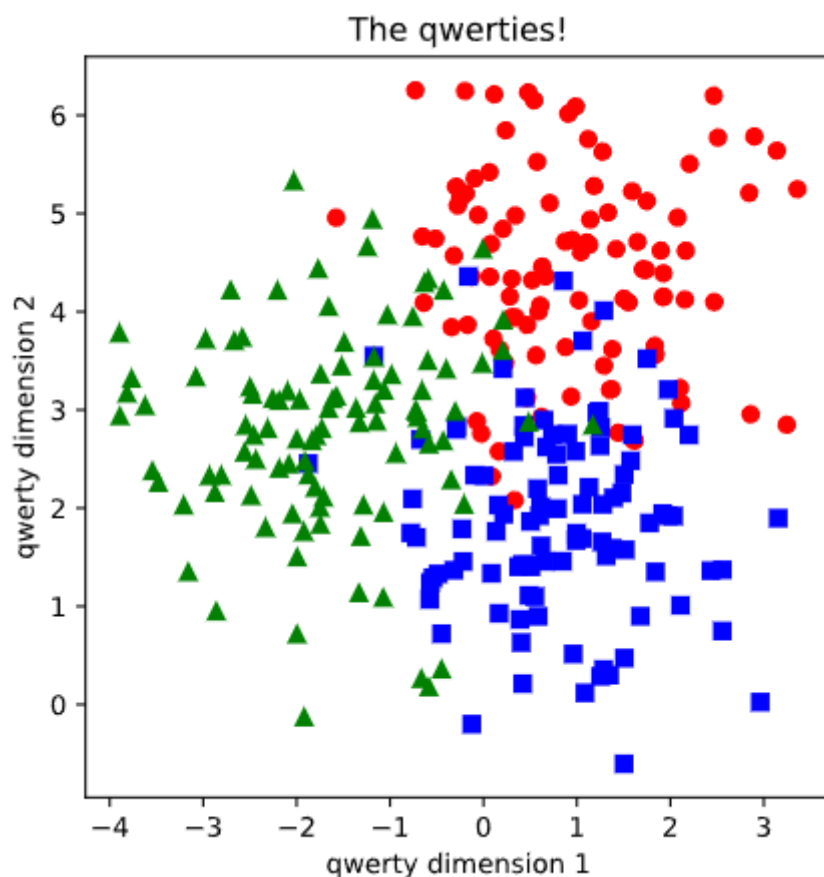


2D space with three groups of samples.

In a two-dimension space we have three groups of samples shown in the following image.



I will create a ML model that can predict to which group will belong to, a set of coordinates. In other words, can the model predict that

(-2,3) coordinate will correspond to group green. The data is called qwerties for lack of imagination.

Let's import the modules required in python. I will use PyTorch for ML computation.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
import matplotlib_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('svg')
```

```
import torch
import torch.nn as nn
```

I will give myself 300 qwerties, 100 for each group red, green and blue. I create some tensors and label the colours as 0, 1 and 2. The plot is shown at the beginning.

```
nrPoints = 300
X, y = make_blobs(nrPoints, centers=3, n_features=2, random_state=0)
x_red = X[np.where(y==0)[0],[0]]
y_red = X[np.where(y==0)[0],[1]]
x_blue = X[np.where(y==1)[0],[0]]
y_blue = X[np.where(y==1)[0],[1]]
x_green = X[np.where(y==2)[0],[0]]
y_green = X[np.where(y==2)[0],[1]]
data = torch.tensor(X).float()
labels = torch.tensor(y).long()
fig = plt.figure(figsize=(5,5))
plt.plot(x_red, y_red, 'ro')
plt.plot(x_blue, y_blue, 'bs')
plt.plot(x_green, y_green, 'g^')
plt.title('The qwerties!')
plt.xlabel('qwerty dimension 1')
plt.ylabel('qwerty dimension 2')
plt.show()
```

Here comes the model (artificial neural network).

It has 2 inputs, 3 outputs and 4 hidden layers in between.

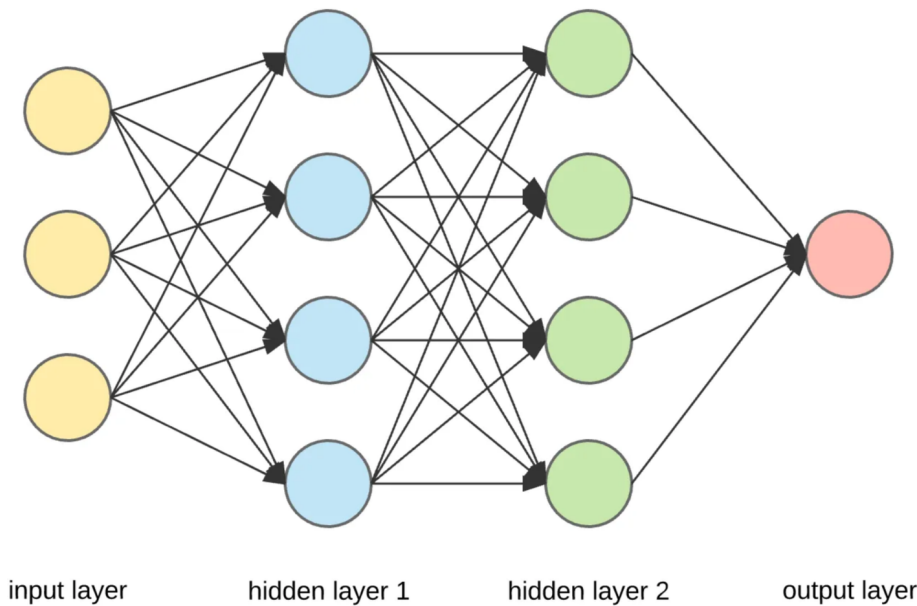
```
#model architecture
ANNq = nn.Sequential(
    nn.Linear(2,4),
    nn.ReLU(),
    nn.Linear(4,3),
    nn.Softmax(dim=1)
)

#loss function
lossfunc = nn.CrossEntropyLoss()

#optimizer
```

```
optimizer = torch.optim.Rprop(ANNq.parameters(), lr=0.1)
```

A graphical representation of ANN with 3 inputs, one output and 2 hidden layers.



The fun part begins when we need to train the model using the following code. Here I programmed the model to educate himself for 500 times, record the losses and calculate the accuracy.

```
#Train model
numepochs = 500

#Initial losses
losses = torch.zeros(numepochs)
ongoingAcc = []

for epochi in range(numepochs):

    #forward pass
    yHat = ANNq(data)

    #compute loss
    loss = lossfunc(yHat,labels)
    losses[epochi] = loss

    #backprop
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```

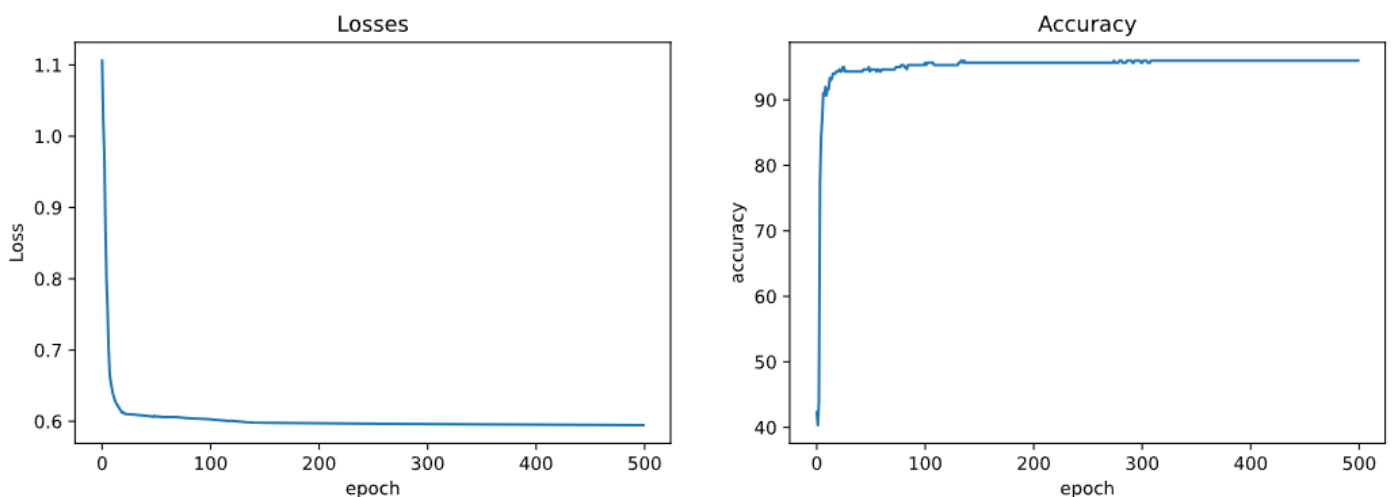
#compute accuracy
matches = torch.argmax(yHat,axis=1) == labels    #boolean false or true
matchesNumeric = matches.float()                  #convert to 0/1
accuracyPct = 100 * torch.mean(matchesNumeric)    #average and *100
ongoingAcc.append( accuracyPct )                  # add to the list of accuracy

#final forward pass
predictions = ANNq(data)

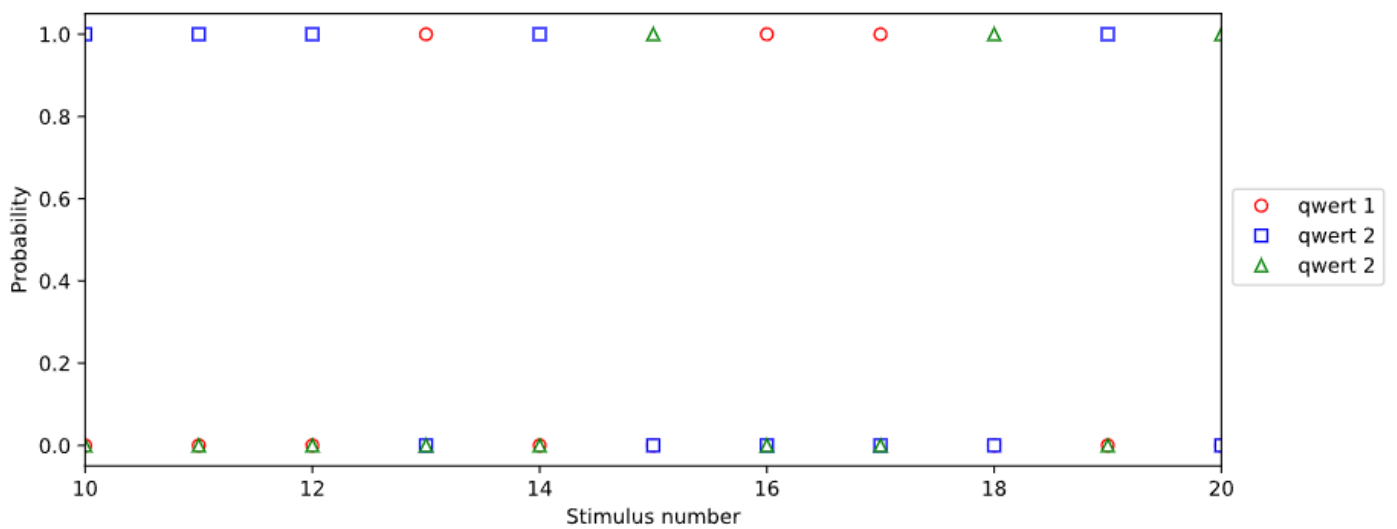
predlabels = torch.argmax(predictions,axis=1)
totalacc = 100 * torch.mean((predlabels == labels).float())

```

Best way is to visualize the losses and accuracy. It seems to be a fast learner. He could've finish his self education after only 100 trials.



Another smart way to show some outcome of the model is to plot the probability of each group. One colour will have 100 probability and the other two will have 0.



We can save the trained model and later utilize the already trained model which is much faster. Complicated models can take hours or days to train.

```
#Save model
torch.save(ANNq.state_dict(), 'traniedModel.pt')

#Another user can load the model
#Load the pre-trained model
loaded_model = ANNq
loaded_model.load_state_dict(torch.load('traniedModel.pt'))
loaded_model.eval()
```

Let's test the trained model:

```
#Testing the model
X_test = [[np.random.randint(-4,4), np.random.randint(-1,6)]]
test_data = torch.tensor(X_test).float()
results = loaded_model(test_data)
print('test numers: ', X_test)
print(results.detach())
if results[0,0].detach() == 1 :
    print('red')
elif results[0,1].detach() == 1 :
    print('blue')
else:
    print('green')
```

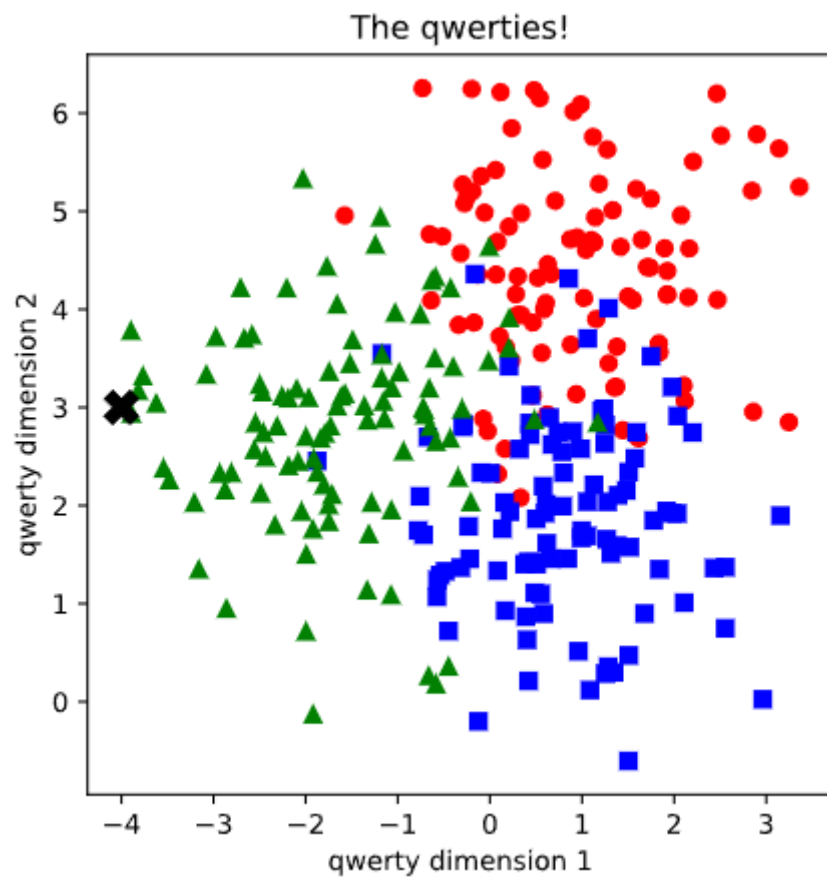
Here is the output:

```
test numbers: [[-4, 3]]
```

```
tensor([[2.7208e-40, 0.0000e+00, 1.0000e+00]])
```

```
The coordinate correspond to group: green
```

And the final result:



Revision #1

Created 20 January 2025 00:36:41 by Delta

Updated 20 January 2025 00:46:32 by Delta