perceptron-for-apple detection

November 6, 2017

In [18]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import random
import torch
from torch.autograd import Variable
```

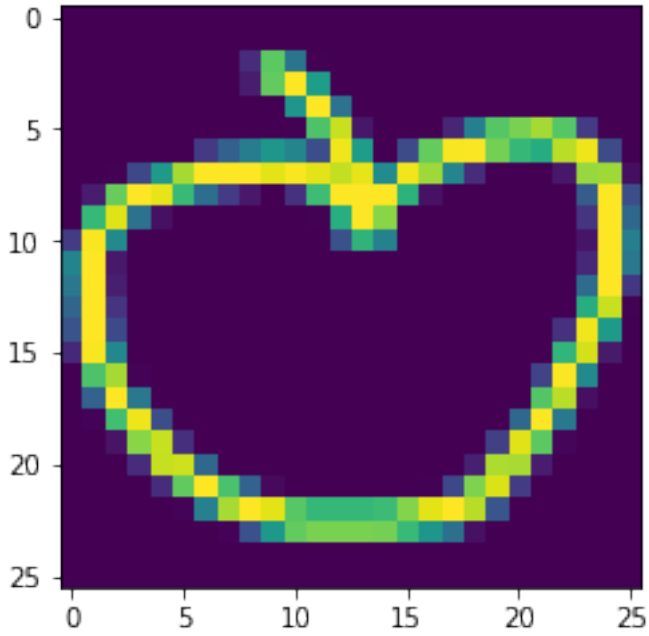
In [19]:
```
images=np.load('data/images.npy')
labels=np.load('data/labels.npy')
labels=labels.astype(int)
labels[labels!=0]=-1
labels[labels==0]=1
labels[labels==-1]=0
```

0.0.1 Number of apples in the test set = 10000

In [20]:
```
apples=images[labels==1]
print(len(apples))
plt.imshow(apples[0])
```

```
10000
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x14ed17080>
```



0.1 Flattening the image

```
In [21]: shape_images_flat=(images.shape[0],images.shape[1]*images.shape[2])
         images_flat=np.ndarray(shape=shape_images_flat)
         for index in range(len(images)):
             images_flat[index]=images[index].flat
         images=images_flat
         images=(images_flat-images_flat.mean())/images.std()

In [22]: images_len=len(images)
         shuffler_list=list(range(0, images_len))
         # print(list)
         random.shuffle(shuffler_list)

In [23]: shuffled_images=images[shuffler_list]
         shuffled_labels=labels[shuffler_list]

In [24]: train_images=shuffled_images[0:40000]
         train_labels=shuffled_labels[0:40000]

         validation_images=shuffled_images[40000:45000]
         validation_labels=shuffled_labels[40000:45000]

         test_images=shuffled_images[45000:50000]
         test_labels=shuffled_labels[45000:50000]
```

1 Random coin flipping accuracy

Random coin flipping gives us: 1. Correct prediction Probabilty=0.5 if apple 2. Correct prediction Probabilty=0.5 if not apple

Probabilty of apple=0.2
Probabilty of not apple=0.8
Overall correct clasification probabilty =0.5 x 0.2 + 0.5 x 0.8
Overall accuracy thus is=0.5

2 Majority Classifier accuracy

We always predict not apple in this case

Probabilty of not apple=0.8
Thus, majority classifer accuracy=0.8

3 Accuracy function

```
In [25]: def accuracy(y, y_hat):
             """Compute accuracy.
             Args:
             y: A 1-D int NumPy array.
             y_hat: A 1-D int NumPy array.
             Returns:
             A float, the fraction of time y[i] == y_hat[i].
             """

             a=(y==y_hat)
             return a.astype(np.float).mean()
```

4 Computes Accuracy Graph for training and validation

```
In [26]: training_accuracy_list=[]
         validation_accuracy_list=[]
         def compute_accuracy_graph(W):
             train_images_len=len(train_images)
             train_shuffler_list=list(range(0, train_images_len))
             random.shuffle(train_shuffler_list)
             shuffled_train_images=train_images[train_shuffler_list]
             shuffled_train_labels=train_labels[train_shuffler_list]

             shuffled_train_images_used=shuffled_train_images[0:1000]
             shuffled_train_labels_used=shuffled_train_labels[0:1000]

             d_train=shuffled_train_images_used.dot(W)
             d_train[d_train>0]=1
             d_train[d_train<=0]=0
```

```
ac_train=accuracy(shuffled_train_labels_used,d_train)
training_accuracy_list.append(ac_train)


validation_images_len=len(validation_images)
validation_shuffler_list=list(range(0, validation_images_len))
random.shuffle(validation_shuffler_list)
shuffled_validation_images=validation_images[validation_shuffler_list]
shuffled_validation_labels=validation_labels[validation_shuffler_list]

shuffled_validation_images_used=shuffled_validation_images[0:1000]
shuffled_validation_labels_used=shuffled_validation_labels[0:1000]

d_validation=shuffled_validation_images_used.dot(W)
d_validation[d_validation>0]=1
d_validation[d_validation<=0]=0

ac_validation=accuracy(shuffled_validation_labels_used,d_validation)
validation_accuracy_list.append(ac_validation)
```

5 Hyper Parameters

In [27]:
```
epochs=3
learning_rate=0.001
```

6 Perceptron Code

In [28]:
```
X=train_images
Y=train_labels

W_tensor=torch.torch.DoubleTensor(X.shape[1]).zero_()
W_tensor=Variable(W_tensor,requires_grad=True)
number_of_images=X.shape[0]
for epoch in range(0,epochs):
    for i in range(0,number_of_images):
        x_tensor=Variable(torch.from_numpy(X[i]),requires_grad=False)
        w_x=torch.dot(x_tensor,W_tensor)
        y_hat=torch.sign(w_x)+1
        y_hat=torch.sign(y_hat)
        J=(y_hat-float(Y[i]))*w_x
        J.backward()
        W_tensor.data -= learning_rate * W_tensor.grad.data
```

```
            if(i%100==0):
                compute_accuracy_graph(W_tensor.data.numpy())

            # Manually zero the gradients after updating weights
            W_tensor.grad.data.zero_()

    weights_now=W_tensor.data.numpy()
```

7 Compute test accuracy

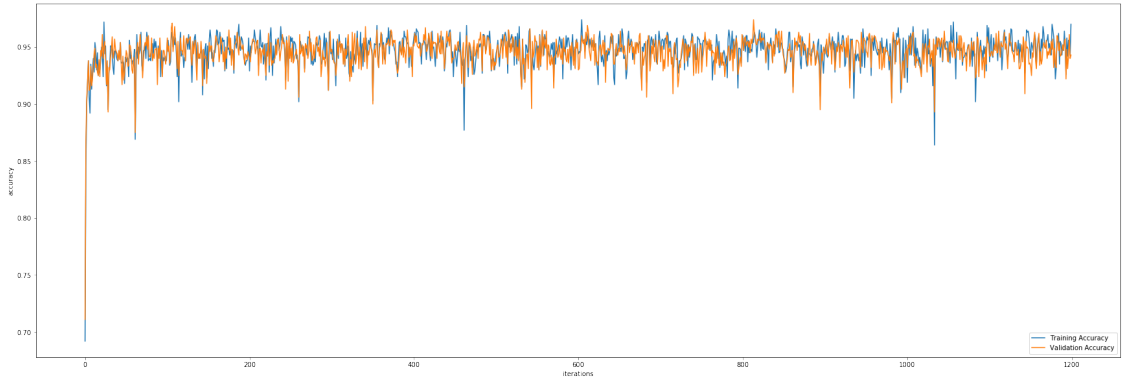
```
In [30]: X=test_images
         y=test_labels
         d=X.dot(weights_now)
         d[d>0]=1
         d[d<=0]=0
         ac=accuracy(y,d)
         print(ac)
```

```
0.9508
```

8 Plot Computation Graph for cross valiadation and training

```
In [31]: plt.figure(figsize=(30, 10))
         plt.xlabel('iterations')
         plt.ylabel('accuracy')
         training_accuracy_line,=plt.plot(training_accuracy_list,label='Training Accuracy')
         validation_accuracy_line,=plt.plot(validation_accuracy_list,label='Validation Accuracy
         plt.legend(handles=[training_accuracy_line, validation_accuracy_line])
```

```
Out[31]: <matplotlib.legend.Legend at 0x12eeaa7f0>
```



I am not overfitting here as there both the cross-validation and the traing accuracy follow each other quite smoothly, if i was overfitting, though my traing would shoot up but would lead to decrease in validation accuracy.

Also , A linear function so limited scope of overfitting on a non-linearrly separable dataset.