EECS 6720 HW2

JINGYING ZHOU

JZ2673

Question 1 Problem Set up: $x_n \sim N(Wz_n, \sigma^2 I)$

$z_n \sim N(0, I)$

At each iteration $P(W) = \left(\frac{\lambda}{2\pi}\right)^{\frac{dk}{2}} \exp\left\{\frac{1}{2}\text{trace}(W^T W)\right\}$

E step

$q(z_n) \propto \text{const} \cdot \exp\left\{\left[-\frac{1}{2}(x_n^T - W_{t-1} z_n^T)^T (\sigma^2 I)^{-1} (x_n^T - W_{t-1} z_n)^T\right]\right\} \cdot \text{const} \cdot \exp\left\{-\frac{1}{2}(z_n z_n^T)\right\}$

Calculate posteriors for each z_n:

$$\propto \exp\left\{-\frac{1}{2}\left[\frac{1}{\sigma^2}\left(x_n x_n^T + z_n W_{t-1}^T W_{t-1} z_n^T - 2 z_n W_{t-1}^T \cdot x_n\right)\right] - \frac{1}{2}(z_n z_n^T)\right\}$$

$$\propto \exp\left\{-\frac{1}{2}\left[\left(\frac{W_{t-1}^T W_{t-1}}{\sigma^2} + I\right) z_n z_n^T - 2 z_n W_{t-1}^T x_n\right]\right\}, \text{ the rest are constant w.r.t. } z_n$$

If we add constant terms to the exponent and complete squares, then it follows a normal form with:

Expectation: $\mu = \dfrac{W_{t-1}^T x_n}{\frac{W_{t-1}^T W_{t-1}}{\sigma^2} + I}$

Variance: $\text{Var} = \dfrac{1}{\frac{W_{t-1}^T W_{t-1}}{\sigma^2} + I}$. These two would be used in M step.

M step a) Take expectations of the log joint likelihood

$$\mathcal{L}(W) = E_q[\ln(P(X, Z|W) P(W))] = \ln P(W) + \sum_{i=1}^{n} E_q[\ln(z_i)] + \sum_{i=1}^{n} E_q[\ln(P(x|z, W))] + \text{const}$$

$$= -\frac{\lambda}{2}\text{trace}(W^T W) + \text{const} + \text{const} + \sum_{i=1}^{n} \frac{1}{2\sigma^2} E_q\left[-\text{const} - z_i^T W^T W z_i + 2 z_i^T W^T x_i\right]$$

$$= -\frac{\lambda}{2}\text{trace}(W^T W) - \frac{1}{2\sigma^2}\sum_{i=1}^{n} E_q\left[z_i^T W^T W z_i - 2 z_i^T W^T x_i\right].$$

$$= -\frac{\lambda}{2}\text{trace}(W^T W) - \frac{1}{2\sigma^2}\sum_{i=1}^{n} \text{trace}\left(W^T W (\text{Var} + \mu_i^2) - 2 W^T x_i \mu_i\right]$$

Maximize with respect to W:

b) $$\frac{\partial \mathcal{L}(W)}{\partial W} = -\lambda W^T - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(2W^T(\text{Var} + \mu_i^2) - 2 x_i \mu_i^T\right) = 0$$

$$\Rightarrow W^T = \left\{\lambda I + \frac{1}{\sigma^2}\left[\sum_{i=1}^{n}(\text{Var} + \mu_i^2)\right]\right\}^{-1} \cdot \left(\frac{1}{\sigma^2}\sum_{i=1}^{n} x_i \mu_i^T\right)$$

$$W^T = \left\{\lambda I + \frac{1}{\sigma^2}\left(\sum_{i=1}^{n}(\text{Var} + \mu_i \mu_i^T)\right)\right\}^{-1} \cdot \left(\frac{1}{\sigma^2}\sum_{i=1}^{n} x_i \mu_i^T\right)$$

$$W^T = \left\{\lambda I \sigma^2 + \sum_{i=1}^{n}(\text{Var} + \mu_i \mu_i^T)\right\}^{-1} \cdot \left(\sum_{i=1}^{n} x_i \mu_i^T\right)$$

We then have updated our weight vector and can proceed to next iteration.

Pseudo-Code:

1. Initialize weight vector at step 0 (iteration 0)
2. At every iteration i ($i = 1, 2, \cdots M$ where M is maximum iteration you se
   E step: Calculate poster~~s~~ for each z_n using W_{t-1} that comes from previous step (Mstep). i.e. calculate $q(z_n)$ given W_{t-1}
   M step: 1) Write out $\mathcal{L}(w)$, take expectations of log joint likelihood;
   2) Maximize $\mathcal{L}(w)$ w.r.t. W, and update W.

Problem 2. (35 points)

a) Implement the EM algorithm for probit regression described in the class notes. Use the parameter setting σ = 1:5 and λ = 1. Run your algorithm on the data set provided for T = 100 iterations.

In [1]:

```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
from scipy.stats import norm
from numpy.linalg import inv
```

In [2]:

```
xtrain = np.genfromtxt ('hw1_data_csv/Xtrain.csv', delimiter=",")
ytrain = np.genfromtxt('hw1_data_csv/ytrain.csv', delimiter=",")
xtest = np.genfromtxt('hw1_data_csv/Xtest.csv', delimiter=",")
ytest = np.genfromtxt('hw1_data_csv/ytest.csv', delimiter=",")
```

In [3]:

```
log = math.log
pi = math.pi
d = xtrain.shape[1]

"""
Input: `it`: number of iterations
Output: `w` weight vector after `it` iterations
        `wonder`: A list of all ln p(~y_wtj|X). Need it in question b)
"""
def em(it):
    w = np.zeros(d)
    wonder = []
    for i in range(it):
        # E step
        ## If Y_i = 0, term3 = 1 ; if Y_i = 1, term2 = 1. Used these two terms to ge
        term2 = np.power(norm.pdf(-np.dot(xtrain, np.transpose(w)/1.5))/(1-norm.cdf(
        term3 = np.power(-norm.pdf(-np.dot(xtrain, np.transpose(w)/1.5))/(norm.cdf(-
        exp = np.dot(xtrain,np.transpose(w)) + 1.5 * term2 * term3
        # M Step
        wt = np.dot(inv((np.ones(d*d).reshape(d,d)) + np.dot(np.transpose(xtrain),xt
        wonder.append(d/2*log(1/2/pi) - 1/2/(np.dot(np.transpose(wt),wt)) + np.dot(y
        w = wt
    return(w,wonder)
```

In [4]:

```
wt, wonder = em(100)
```

b) Plot ln p(~y;wtjX) as a function of t.

In [5]:

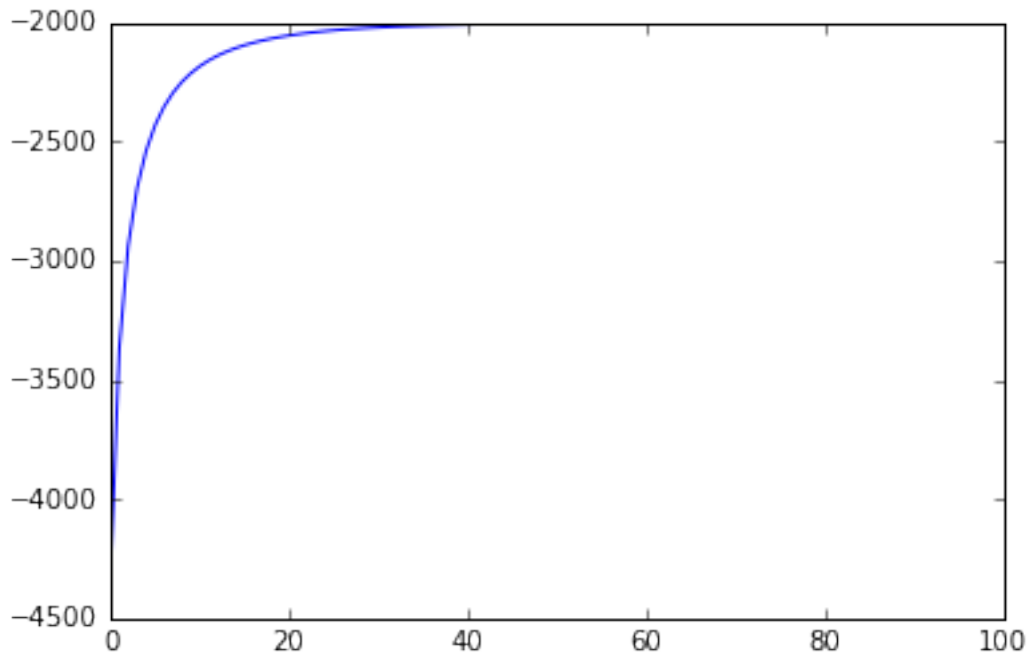
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
```

In [6]:

```
plt.plot(range(100), wonder)
```

Out[6]:

```
[<matplotlib.lines.Line2D at 0x1177cf3c8>]
```



c) Make predictions for all data in the testing set by assigning the most probable label to each feature vector. In a 22 table, list the total number of 4's classied as 4's, 9's classied as 9's, 4's classied as 9's, and 9's classied as 4's (i.e., a confusion matrix). Use the provided ground truth for this evaluation.

In [7]:

```
ypred = np.zeros(xtest.shape[0])
ypred[norm.cdf(np.dot(xtest,wt)/1.5)>=0.5] =1
```

In [8]:

```
from sklearn.metrics import confusion_matrix
confusion_matrix(ytest,ypred,labels=[0, 1])
```

Out[8]:

```
array([[931,  51],
       [ 77, 932]])
```

Then the confusion matrix can be written as:

	4	9
4	931	51
9	77	932

d) Pick three misclassied digits and reconstruct the images as described in the readme le.Show these three images and their predictive probabilities.

In [9]:

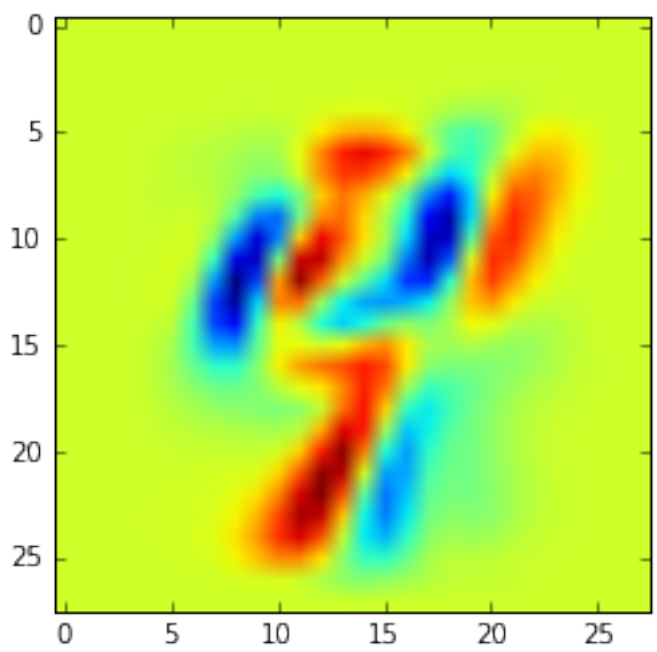
```
Q = np.genfromtxt ('hw1_data_csv/Q.csv', delimiter=",")
# Mapped: the first three mis-classified digits
mapped = np.dot(Q,np.transpose(xtest[ypred!=ytest][0:3]))
```

In [10]:

```
plt.imshow(np.transpose(mapped)[0].reshape(28,28))
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x1176a4a58>
```



In [11]:

```
ytest[ypred!=ytest][0]
```

Out[11]:

```
0.0
```

In [12]:

```
norm.cdf(np.dot(xtest[ypred!=ytest][0],wt)/1.5)
```

Out[12]:

```
0.67796816835763052
```

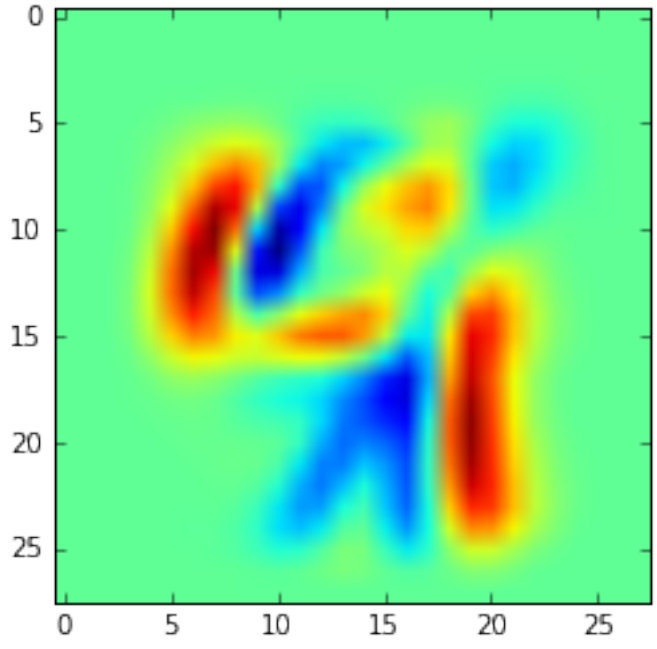
This number is 4 and we predicted it as 9. The predicted probability is 0.68.

In [13]:

```
plt.imshow(np.transpose(mapped)[1].reshape(28,28))
```

Out[13]:

```
<matplotlib.image.AxesImage at 0x117e40710>
```



In [14]:

```
[ytest[ypred!=ytest][1],norm.cdf(np.dot(xtest[ypred!=ytest][1],wt)/1.5)]
```

Out[14]:

```
[0.0, 0.695060229769051 13]
```

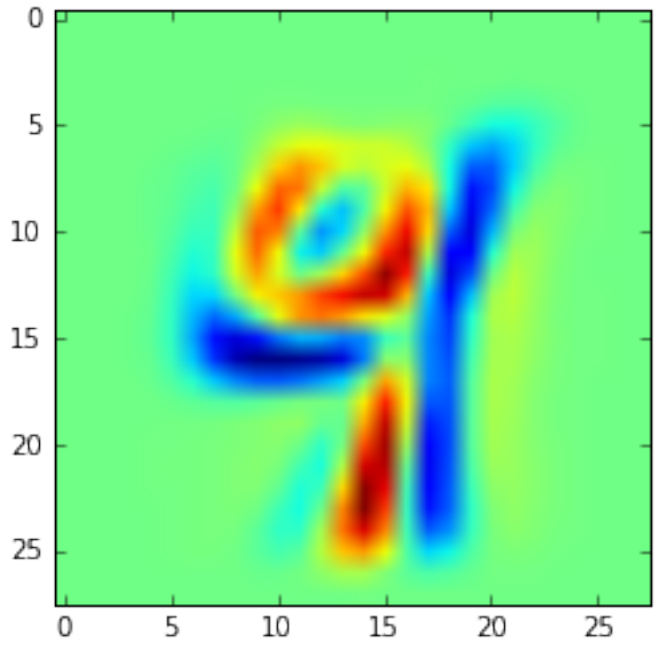
This number is 4 and we predicted it as 9. The predicted probability of being 9 is 0.70.

In [15]:

```
plt.imshow(np.transpose(mapped)[2].reshape(28,28))
```

Out[15]:

```
<matplotlib.image.AxesImage at 0x117c316a0>
```



In [16]:

```
[ytest[ypred!=ytest][2],norm.cdf(np.dot(xtest[ypred!=ytest][2],wt)/1.5)]
```

Out[16]:

```
[0.0, 0.9039906855277708]
```

This number is 4 and we predicted it as 9. The predicted probability of being 9 is 0.90. It really does look like 9!

e) Pick the three most ambiguous predictions, i.e., the digits whose predictive probabilities are the closest to 0:5. Reconstruct the three images as described in the readme le and show them and their predictive probabilities.

In [17]:

```
# Compute the probability and rank them
array = abs(norm.cdf(np.dot(xtest,wt)/1.5)-0.5)
order = array.argsort()
ranks = order.argsort()
```

In [18]:

```
# Reconstruct the digits
digits = np.dot(xtest,np.transpose(Q))
```

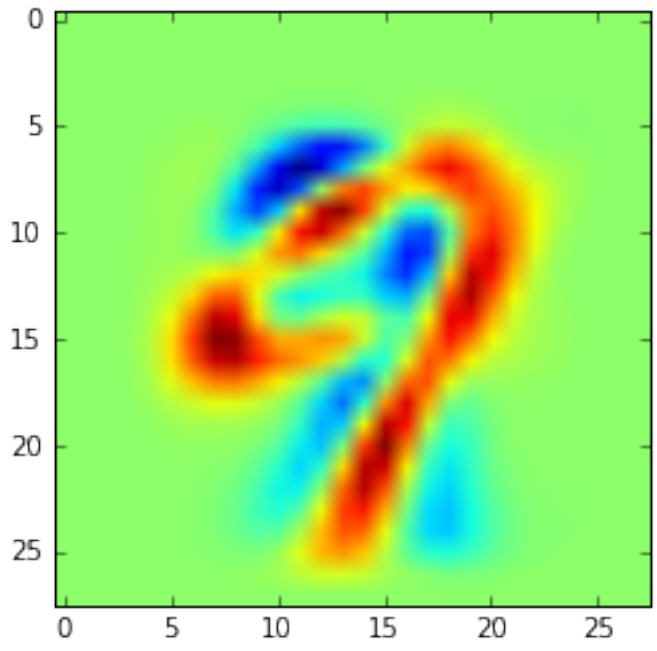
1) The most ambiguous one

In [19]:

```
plt.imshow(digits[ranks==0].reshape(28,28))
```

Out[19]:

```
<matplotlib.image.AxesImage at 0x117913f60>
```



In [20]:

```
print ('predicted probabilities:',(norm.cdf(np.dot(xtest,wt)/1.5))[ranks==0][0])
```

```
predicted probabilities: 0.501255614196
```

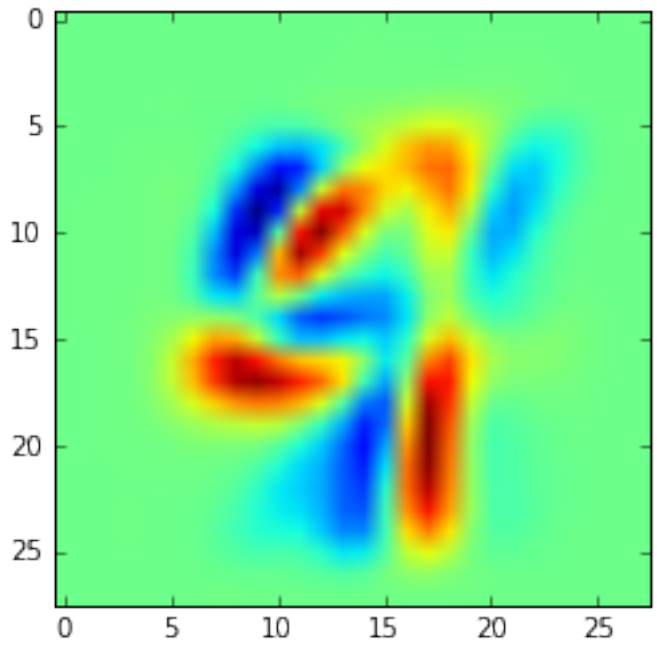
Second most ambiguous

In [21]:

```
plt.imshow(digits[ranks==1].reshape(28,28))
```

Out[21]:

```
<matplotlib.image.AxesImage at 0x11727de80>
```



In [22]:

```
print ('predicted probabilities:',(norm.cdf(np.dot(xtest,wt)/1.5))[ranks==1][0])
```

```
predicted probabilities: 0.502074068535
```

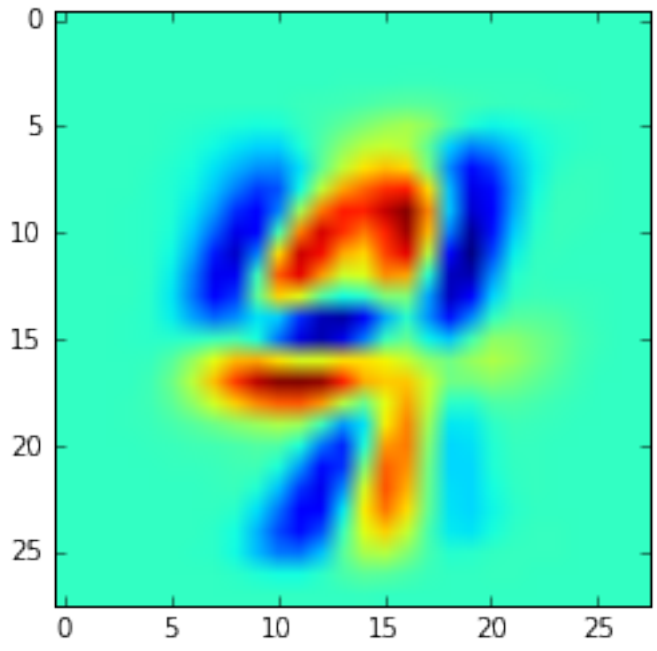
Third Most Ambiguous

In [23]:

```
plt.imshow(digits[ranks==2].reshape(28,28))
```

Out[23]:

```
<matplotlib.image.AxesImage at 0x1175f24e0>
```



In [24]:

```
print ('predicted probabilities:',(norm.cdf(np.dot(xtest,wt)/1.5))[ranks==2][0])
```

```
predicted probabilities: 0.496686428475
```

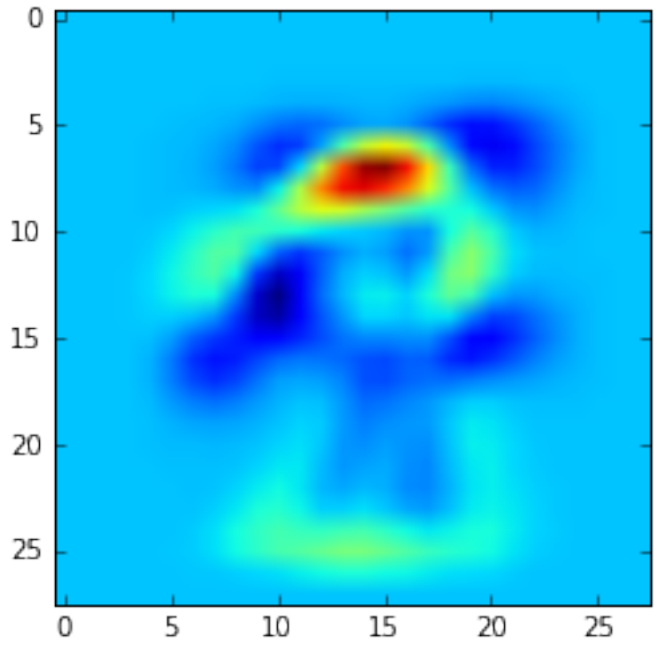
f) Treat the vector wt as if it were a digit and reconstruct it as an image for t = 1; 5; 10; 25; 50; 100. Show these images and comment on what you observe.

In [25]:

```
plt.imshow(np.dot(Q,np.transpose(em(1)[0])).reshape(28,28))
```

Out[25]:

```
<matplotlib.image.AxesImage at 0x11c137470>
```

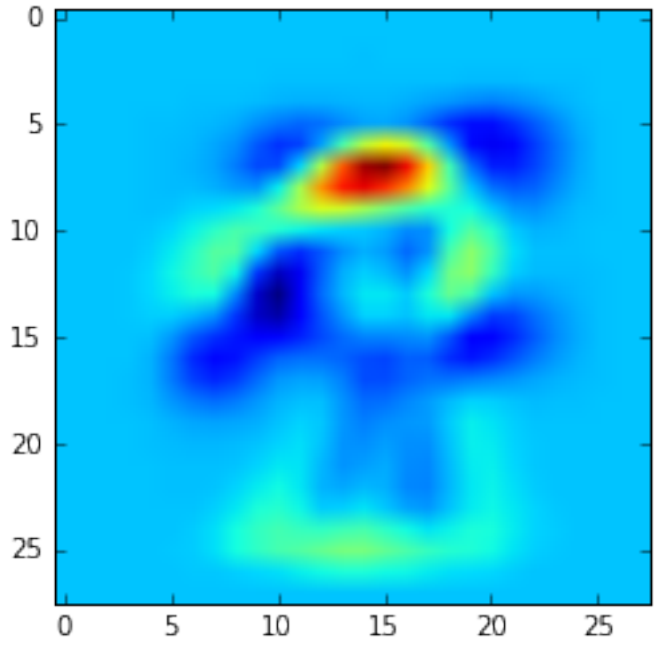


In [26]:

```
plt.imshow(np.dot(Q,np.transpose(em(5)[0])).reshape(28,28))
```

Out[26]:

```
<matplotlib.image.AxesImage at 0x11c210b70>
```

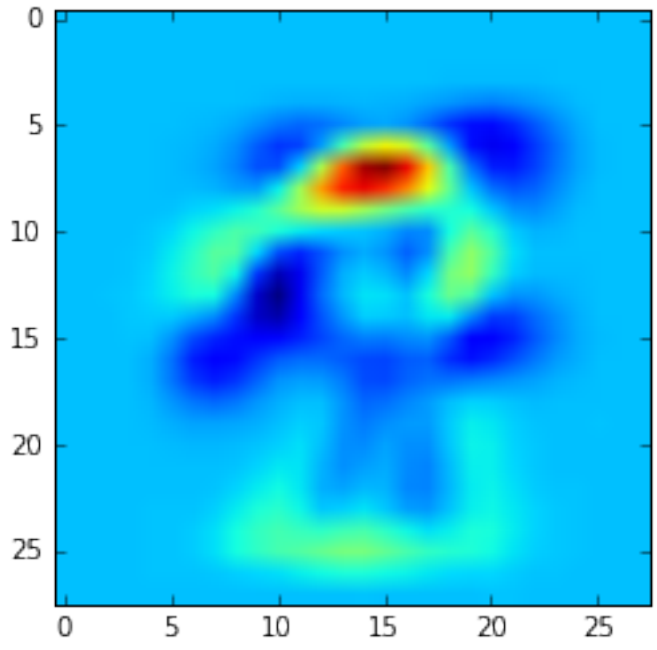


In [27]:

```
plt.imshow(np.dot(Q,np.transpose(em(10)[0])).reshape(28,28))
```

Out[27]:

```
<matplotlib.image.AxesImage at 0x11c326a58>
```

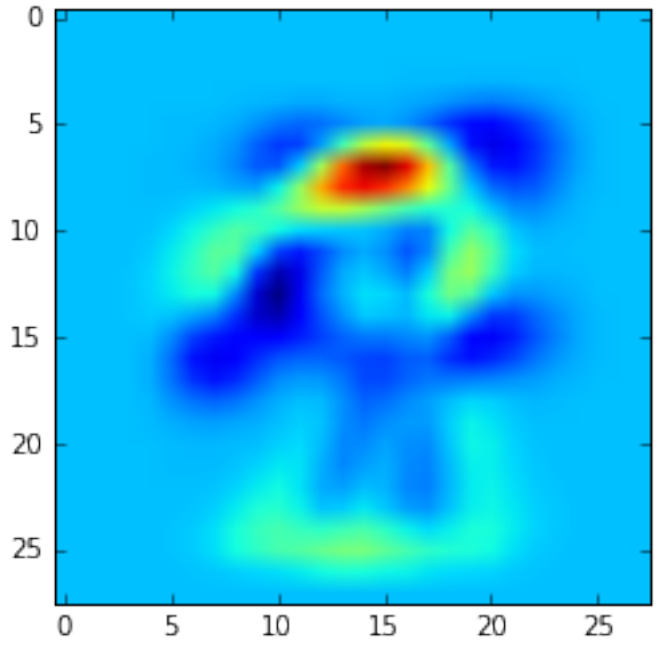


In [28]:

```
plt.imshow(np.dot(Q,np.transpose(em(25)[0])).reshape(28,28))
```

Out[28]:

```
<matplotlib.image.AxesImage at 0x11c4f94a8>
```

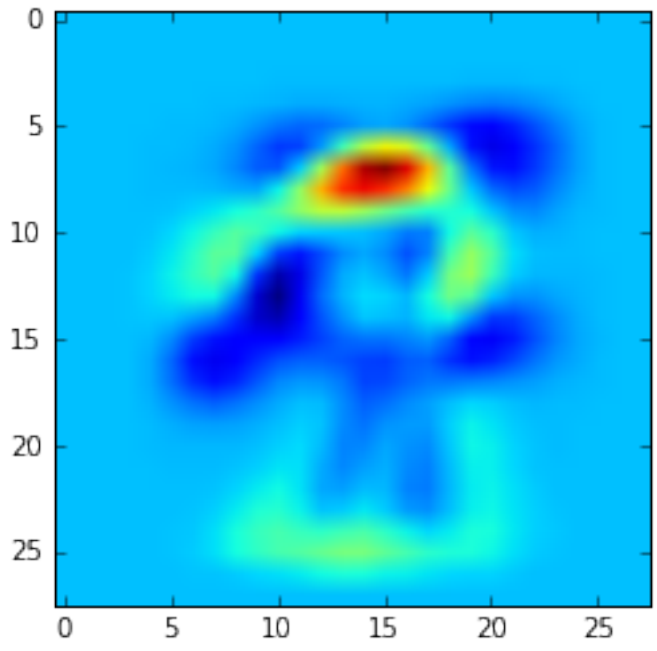


In [29]:

```
plt.imshow(np.dot(Q,np.transpose(em(50)[0])).reshape(28,28))
```

Out[29]:

```
<matplotlib.image.AxesImage at 0x11c43aa20>
```

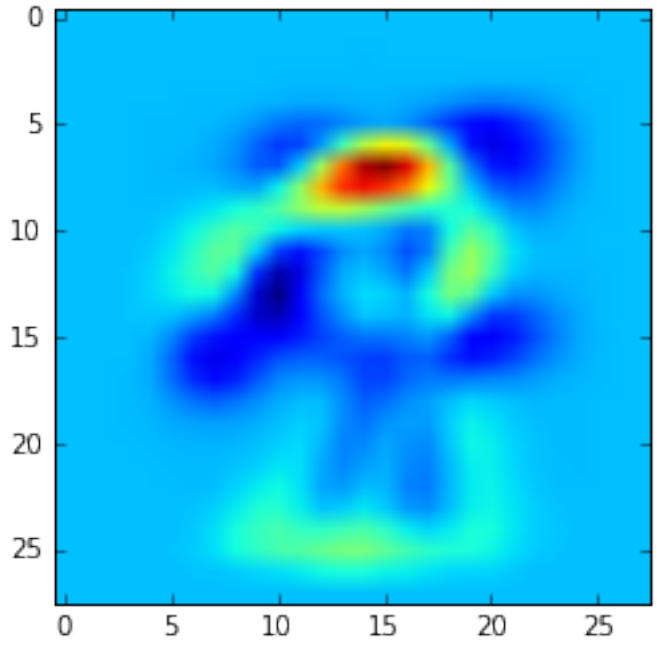


In [30]:

```
plt.imshow(np.dot(Q,np.transpose(em(100)[0])).reshape(28,28))
```

Out[30]:

```
<matplotlib.image.AxesImage at 0x11c8ac240>
```



1) General patterns we see:
We can slightly tell a frame of 4 and 9, for the dark blues captures the frames of 4 that is not present in 9. and the green and red show the "arch" of 9 that's not in 9.

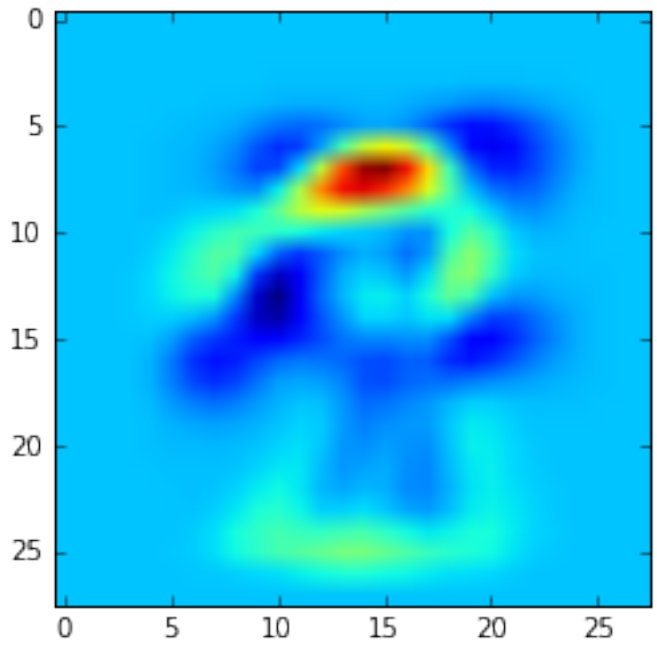
2) Gradual change across different interations The change in weight is small due to scale. But the intensity of color changes with the increase of number of iterations we do. If we go one step further to plot 500 iteration and compare it with only one iteration:

In [31]:

```
plt.imshow(np.dot(Q,np.transpose(em(1)[0])).reshape(28,28))
```

Out[31]:

```
<matplotlib.image.AxesImage at 0x11c909c88>
```

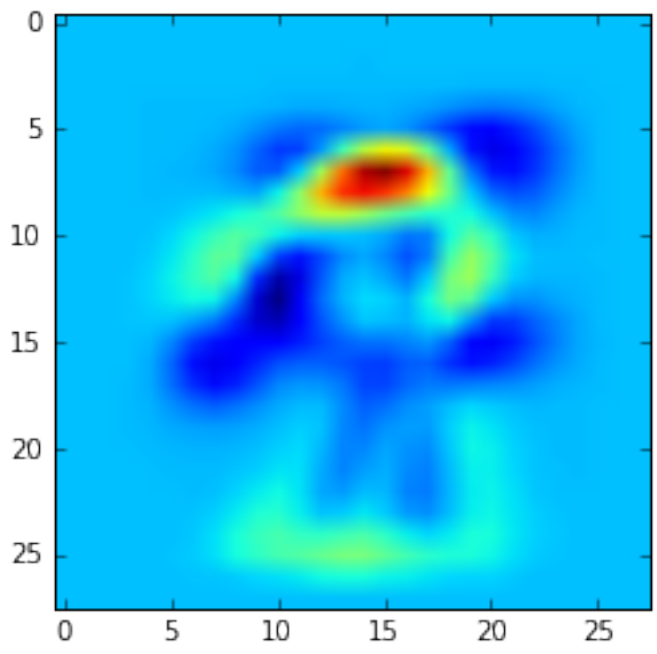


In [32]:

```
plt.imshow(np.dot(Q,np.transpose(em(500)[0])).reshape(28,28))
```

Out[32]:

```
<matplotlib.image.AxesImage at 0x11cbad710>
```



The colors get darker in some areas. I guess this means the first iteration captures a lot of the differece in 4 and 9 and changed the weight vector a lot. After the first iteration we are just moving step after step to get closer to

a better estimate.