Home Work 2.
CSE 253

APOORVE DAVE
A53103070

Q1. Sum of squared Errors :

Let the distribution be $t = y(x) + \epsilon$

Now Since $y(x)$ is deterministic we consider
$\epsilon = \mathcal{N}(\mu = 0, \sigma^2)$ [Gaussian Noise]

To model this, consider we estimate t by expression

$$y(x, \theta) = \sum_{k=1}^{k} \theta_k x_k$$

Assumption: Consider x, y to be drawn from i.i.d distribution
i.e. $t_1, t_2 \ldots t_n$ are i.i.d

Then
Maximum likelihood of data can be obtained by

$$\max_\theta P(x, t) = \max_\theta P(t|x) \cdot P(x)$$

Since x is indep of $\theta \Rightarrow$

$$\max_\theta P(t|x)$$

To find best θ which maximises likelihood.

$$\Rightarrow \boxed{\arg\max_\theta P(t|x).}$$

Assuming iid

$$\underset{\theta}{\operatorname{argmax}}\ P(t|x) = \underset{\theta}{\operatorname{argmax}} \prod_i (P(t_i|x_i))$$

Now from $t_i = \underbrace{y_i(x)}_{\text{deterministic}} + \underline{e_i} \longrightarrow \mathcal{N}(0, \sigma^2)$

$$\Rightarrow \quad P(t_i|x_i) \sim \mathcal{N}(y_i(x), \sigma^2)$$

this Distro. can be written as:

$$\boxed{p(t_i|x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{-(y_i - t_i)^2}{2\sigma^2}\right]} \quad \text{— (1)}$$

$$\text{So} \quad y(x,\theta) = \underset{y(x,\theta)}{\operatorname{argmax}} \prod_i P(t_i|x_i)$$

Taking log & Maximizing [$\because$ log is increasing f^n].

$$y(x,\theta) = \underset{y(x,\theta)}{\operatorname{argmax}} \sum_i \log(P(t_i|x_i))$$

$$= \operatorname{argmax} \sum_i \left[-(y_i - t_i)^2 + \log C\right]$$

[from Eqn (1). where C is independ of θ]

$$\text{or.} \quad \theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} (t_i - y_i)^2$$

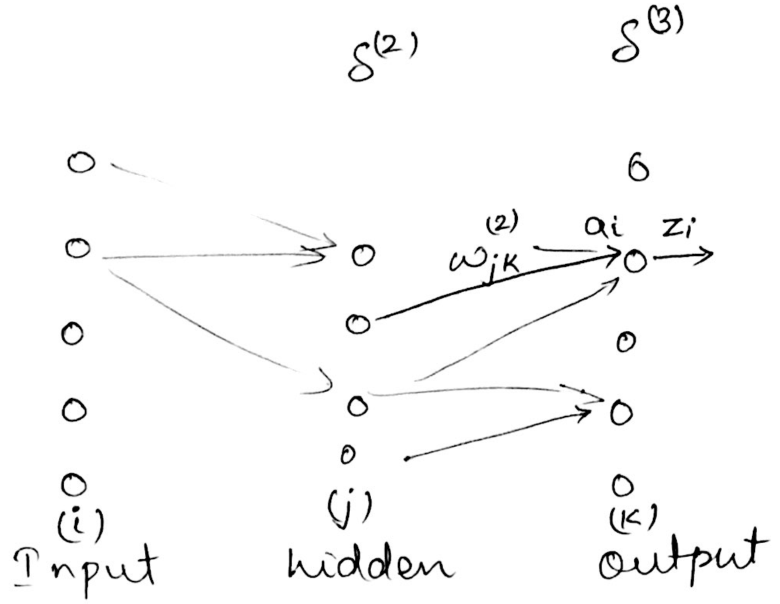
[removing -ve singn & change to argmin.

~~θ^* = argmax~~

$$\boxed{\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{N} (t_i - \theta.x_i)^2}$$

QED

Q2.



Notations : $\delta^{(3)}$ => δ of output layer.

$\delta^{(2)}$ = δ of hidden.

$w_{ij}^{\cancel{(2)}}$ = weigth from input i to hidden j

w_{jk} = weight from hidden j to output k

① $\boxed{\delta_i = -\dfrac{\partial E}{\partial a_i}}$

Output Layer

Considering Softmax, $E = -\left[\sum_i t_i \log y_i + (1-t_i)\log(1-y_i)\right]$

From our notations $E = -\left[\sum_K t_k \log z_k + (1-t_k)\log(1-z_k\right.$

$\therefore \dfrac{\partial E}{\partial z_K} = -\left[\dfrac{t_K}{z_K} - \dfrac{(1-t_K)}{(1-z_K)}\right]$

$$\boxed{\frac{\partial E}{\partial z_K} = -\frac{(t_K - z_K)}{z_K(1-z_K)}} \quad \text{— ①}$$

- was part of prev. assign

Since we use softmax for output layer, whose input is denoted as a_k, output as z_k.

$$\Rightarrow \quad z_k = \frac{\exp(a_k)}{\sum_{k'} \exp(a_{k'})}$$

$$\Rightarrow \quad \frac{\partial z_k}{\partial a_k} = \frac{\left(\sum \exp(a_{k'})\right)\exp(a_k) - \left(\exp(a_k)\right)^2}{\left[\sum \exp(a_k)\right]^2} \qquad — ②$$

$$\boxed{\frac{\partial z_k}{\partial a_k} = z_k(1 - z_k)}$$

Merging Eqn ① & ② For output layer:

$$\boxed{\cancel{\delta_k = -\frac{\partial E}{\partial a_k} \cdot \partial z_k}}$$

$$\delta_k = -\frac{\partial E}{\partial z_k} \cdot \frac{\partial z_k}{\partial a_k} = +(t_k - z_k)$$

$$\boxed{\delta_k = -(z_k - t_k)} \qquad — ③.$$

or in vectorised form: $\boxed{\delta^L = z - t}$

For Hidden Layer.

Let activation function be $f(x)$. Let its derivative be $f'(x)$. We have to find an Expression for (as before)

$$\boxed{\delta_j = -\frac{\partial E}{\partial a_j}}$$

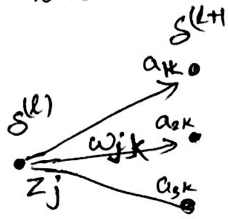
$$\delta_j = \frac{-\partial E}{\partial a_j}$$

$$= \frac{-\partial E}{\partial z_j} \cdot \frac{\partial z_j}{\partial a_j}$$

$$= -\frac{\partial E}{\partial z_j} \cdot f'(a_j)$$

[$\because$ activation of a_j $= f(a_j) = z_j$] — (4)

Since back prop is flowing backward from "future" $l+1$ layer (if current layer is l).



$$= -\left(\sum_k w_{jk}(-\delta_k)\right) \cdot f'(a_j)$$

[$\because \delta_k = \frac{\partial E}{\partial a_k}$] [replaced here]

$$= + f'(a_j) \cdot \left(\sum_k w_{jk} \cdot \delta_k\right)$$

$$\boxed{\delta_j = f'(a_j) \cdot \sum_k w_{jk}\, \delta_k}$$ — (5)

or vectorised: $\boxed{\delta^J = f'(a^J) \odot w \cdot \delta}$

(6) Update Rule

$$\boxed{w_i = w_i - \alpha \frac{\partial E}{\partial w_i}}$$

from Chain rule, $\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial a_j} \cdot \frac{\partial a_j}{\partial w_i}$

Since $\boxed{a_j = \sum_i w_i z_i}$ $= \delta_j \cdot z_i$

$\therefore$ Update rule :

$$\boxed{w_i \leftarrow w_i + \delta_j z_i \cdot \alpha}$$

or vectorised :

$$\boxed{w \leftarrow w + \alpha \left(Z^{(i)} \cdot \delta^{(j)T} \right)}$$

where $Z^{(i)}$ is z vector in ith layer.

$\delta^{(j)}$ is δ vector in jth layer.

Apoorve Dave
A53103070

Q2.

Gradient Descent checking with numerical gradient:
Differences of means of weights and biases are of the following order:
Mean difference of w_{ij} = -3.38982204936e-13
Mean difference of w_{jk} = -3.58549017475e-13

Various Experiments and their results are shown as below(Graph Plots along with the code a t the end):

Basic

Layers:	3
Hidden Nodes:	30
Epochs:	10
LearningRate:	0.1
Mini Batch Size:	10
Activation Fn:	sigmoid
Gamma(Momentum):	0.0
Lambda(Regularize):	0.0

Epoch 1 train,test accuracy: 0.93865 0.9315
Epoch 2 train,test accuracy: 0.950083333333 0.9376
Epoch 3 train,test accuracy: 0.954716666667 0.9401
Epoch 4 train,test accuracy: 0.958566666667 0.9406
Epoch 5 train,test accuracy: 0.961983333333 0.943
Epoch 6 train,test accuracy: 0.965116666667 0.9435
Epoch 7 train,test accuracy: 0.967016666667 0.9419
Epoch 8 train,test accuracy: 0.969416666667 0.9442
Epoch 9 train,test accuracy: 0.9712 0.9431
Epoch 10 train,test accuracy: 0.972233333333 0.9428

Lambda = 0.001: The training and test accuracy has both reduced a bit. This was expected for training but since test accuracy has decreased, it means we are regularizing more than necessary.

Layers:	3
Hidden Nodes:	30
Epochs:	10
LearningRate:	0.1
Mini Batch Size:	10
Activation Fn:	sigmoid
Gamma(Momentum):	0.0
Lambda(Regularize):	0.001

Epoch 1 train,test accuracy: 0.935316666667 0.9297
Epoch 2 train,test accuracy: 0.942766666667 0.9328
Epoch 3 train,test accuracy: 0.944966666667 0.9365
Epoch 4 train,test accuracy: 0.945416666667 0.9359
Epoch 5 train,test accuracy: 0.946783333333 0.9361
Epoch 6 train,test accuracy: 0.951083333333 0.9406
Epoch 7 train,test accuracy: 0.949366666667 0.9359

Epoch 8 train,test accuracy: 0.95205 0.9398
Epoch 9 train,test accuracy: 0.951083333333 0.9423
Epoch 10 train,test accuracy: 0.953833333333 0.9409

Lambda = 0.0001: Test accuracy increased with this change from 2(d). This is good value for regularization

Layers:	3
Hidden Nodes:	30
Epochs:	10
LearningRate:	0.1
Mini Batch Size:	10
Activation Fn:	sigmoid
Gamma(Momentum):	0.0
Lambda(Regularize):	0.0001

Epoch 1 train,test accuracy: 0.938233333333 0.9318
Epoch 2 train,test accuracy: 0.950416666667 0.9382
Epoch 3 train,test accuracy: 0.954183333333 0.9395
Epoch 4 train,test accuracy: 0.95785 0.9405
Epoch 5 train,test accuracy: 0.9613 0.9437
Epoch 6 train,test accuracy: 0.9647 0.9449
Epoch 7 train,test accuracy: 0.96575 0.9434
Epoch 8 train,test accuracy: 0.967 0.9446
Epoch 9 train,test accuracy: 0.9675 0.9464
Epoch 10 train,test accuracy: 0.9691 0.9466

Gamma = 0.9: With addition of gamma, the overall accuracy has decreased keeping the other parameters constant. This will probably work after a few epochs only when the weight values are changing very slowly. Probably a slow learning rate will help.

Layers:	3
Hidden Nodes:	30
Epochs:	10
LearningRate:	0.1
Mini Batch Size:	10
Activation Fn:	sigmoid
Gamma(Momentum):	0.9
Lambda(Regularize):	0.0

Epoch 1 train,test accuracy: 0.917583333333 0.9157
Epoch 2 train,test accuracy: 0.9278 0.9233
Epoch 3 train,test accuracy: 0.933266666667 0.9253
Epoch 4 train,test accuracy: 0.933666666667 0.9239
Epoch 5 train,test accuracy: 0.935383333333 0.9251
Epoch 6 train,test accuracy: 0.941983333333 0.9357
Epoch 7 train,test accuracy: 0.942666666667 0.9336
Epoch 8 train,test accuracy: 0.940066666667 0.9316
Epoch 9 train,test accuracy: 0.945816666667 0.9356
Epoch 10 train,test accuracy: 0.946516666667 0.9347

Activation: tanh, Learning rate: 0.1: Gave slightly worse performance compared to sigmoid, keeping the remaining parameters constant.

Layers:	3
Hidden Nodes:	30
Epochs:	10
LearningRate:	0.1
Mini Batch Size:	10
Activation Fn:	tanh
Gamma(Momentum):	0.0
Lambda(Regularize):	0.0
Epoch 1 train,test accuracy:	0.929516666667 0.9219
Epoch 2 train,test accuracy:	0.9389 0.9274
Epoch 3 train,test accuracy:	0.944583333333 0.9344
Epoch 4 train,test accuracy:	0.946416666667 0.9338
Epoch 5 train,test accuracy:	0.951266666667 0.9341
Epoch 6 train,test accuracy:	0.953616666667 0.9374
Epoch 7 train,test accuracy:	0.954083333333 0.9348
Epoch 8 train,test accuracy:	0.956233333333 0.9364
Epoch 9 train,test accuracy:	0.958216666667 0.9373
Epoch 10 train,test accuracy:	0.958 0.9368

Activation: relu, Learning rate: 0.001: Also a good option. Compared to tanh and sigmoid, it gave a very bad performance for learning rate of 0.1. Only after learning rate was changed to 0.001 did it show any promising results. Rest of the params constant.

Layers:	3
Hidden Nodes:	30
Epochs:	10
LearningRate:	0.001
Mini Batch Size:	10
Activation Fn:	relu
Gamma(Momentum):	0.0
Lambda(Regularize):	0.0
Epoch 1 train,test accuracy:	0.87645 0.8834
Epoch 2 train,test accuracy:	0.902033333333 0.9049
Epoch 3 train,test accuracy:	0.91405 0.9141
Epoch 4 train,test accuracy:	0.92145 0.9208
Epoch 5 train,test accuracy:	0.92645 0.9266
Epoch 6 train,test accuracy:	0.930783333333 0.9305
Epoch 7 train,test accuracy:	0.93475 0.934
Epoch 8 train,test accuracy:	0.937933333333 0.9366
Epoch 9 train,test accuracy:	0.9407 0.9387
Epoch 10 train,test accuracy:	0.942833333333 0.9392

Hidden nodes = 15: Halving the number of hidden units reduced the performance of the model, as expected. That said, I still achieved 92.5% accuracy on test set in just 10 epochs. The training was considerably faster compared to 30 hidden units.

Layers:	3
Hidden Nodes:	15

Epochs: 10
LearningRate: 0.1
Mini Batch Size: 10
Activation Fn: sigmoid
Gamma(Momentum): 0.0
Lambda(Regularize): 0.0
Epoch 1 train,test accuracy: 0.920333333333 0.9153
Epoch 2 train,test accuracy: 0.9241 0.9164
Epoch 3 train,test accuracy: 0.930916666667 0.9182
Epoch 4 train,test accuracy: 0.935383333333 0.9229
Epoch 5 train,test accuracy: 0.9391 0.9251
Epoch 6 train,test accuracy: 0.94045 0.9261
Epoch 7 train,test accuracy: 0.94305 0.9264
Epoch 8 train,test accuracy: 0.9442 0.9261
Epoch 9 train,test accuracy: 0.944283333333 0.9261
Epoch 10 train,test accuracy: 0.945983333333 0.9258

Hidden nodes = 60: Training was slower, but much higher accuracy attained in less number of epochs. It bested the previous (30 node) result in just 2 epochs for test set.
Layers: 3
Hidden Nodes: 60
Epochs: 10
LearningRate: 0.1
Mini Batch Size: 10
Activation Fn: sigmoid
Gamma(Momentum): 0.0
Lambda(Regularize): 0.0
Epoch 1 train,test accuracy: 0.950233333333 0.9438
Epoch 2 train,test accuracy: 0.964933333333 0.9534
Epoch 3 train,test accuracy: 0.9716 0.9577
Epoch 4 train,test accuracy: 0.975083333333 0.9568
Epoch 5 train,test accuracy: 0.979233333333 0.9579
Epoch 6 train,test accuracy: 0.9818 0.9575
Epoch 7 train,test accuracy: 0.98425 0.9599
Epoch 8 train,test accuracy: 0.986583333333 0.959
Epoch 9 train,test accuracy: 0.987966666667 0.9598
Epoch 10 train,test accuracy: 0.989 0.9602

Hidden Nodes = 100: Slowest, and most accurate of my test with 100 hidden units.(expected)
Layers: 3
Hidden Nodes: 100
Epochs: 10
LearningRate: 0.1
Mini Batch Size: 10
Activation Fn: sigmoid
Gamma(Momentum): 0.0

Apoorve Dave
A53103070

Lambda(Regularize): 0.0
Epoch 1 train,test accuracy: 0.9534 0.9465
Epoch 2 train,test accuracy: 0.9685 0.9543
Epoch 3 train,test accuracy: 0.976583333333 0.9605
Epoch 4 train,test accuracy: 0.98225 0.9648
Epoch 5 train,test accuracy: 0.985833333333 0.9647
Epoch 6 train,test accuracy: 0.989383333333 0.9665
Epoch 7 train,test accuracy: 0.992 0.9671
Epoch 8 train,test accuracy: 0.993533333333 0.9668
Epoch 9 train,test accuracy: 0.995116666667 0.9671
Epoch 10 train,test accuracy: 0.995816666667 0.9672

Num Layers = 4: Very Slightly better(on test) and slower to train compared to 3 layer network. This requires a lot of parameters to train as we can end up in vanishing gradient problem. With other settings left as they were, this network was slow to train as well as did not give much improvement.
Layers: 4
Hidden Nodes: 30
Epochs: 10
LearningRate: 0.1
Mini Batch Size: 10
Activation Fn: sigmoid
Gamma(Momentum): 0.0
Lambda(Regularize): 0.0
Epoch 1 train,test accuracy: 0. 92326666666666 0. 9176999999999
Epoch 2 train,test accuracy: 0. 9454666666666666 0. 9335
Epoch 3 train,test accuracy: 0. 9567666666666666 0. 9383
Epoch 4 train,test accuracy: 0. 9545166666666667 0. 9413
Epoch 5 train,test accuracy: 0. 9574833333333334 0. 9427
Epoch 6 train,test accuracy: 0.961266666666667 0.9462
Epoch 7 train,test accuracy: 0.9664166666666667 0. 9472
Epoch 8 train,test accuracy: 0. 9645366666666668 0.9426
Epoch 9 train,test accuracy: 0.9693 0.94375
Epoch 10 train,test accuracy: 0.9711333333333333 0.9493

HW2Q2 - Second Attempt-lambda1

January 26, 2016

```
In [19]: import numpy as np
         from numpy import shape, matrix, log, exp, zeros,random,dot,multiply
         from mnist import readWithoutBias
         from math import sqrt
```

```
In [2]: data_train, label_train = readWithoutBias(dataset="training")
        data_test, label_test = readWithoutBias(dataset="testing")
        label_train = matrix(label_train)
        label_test  = matrix(label_test)
        data_train  = matrix(data_train).T
        data_test   = matrix(data_test).T
```

```
In [20]: train_mean  = data_train.mean(axis=1)
         train_std   = data_train.std(axis=1)
         data_train  = np.nan_to_num((data_train - train_mean)/train_std)
         data_test   = np.nan_to_num((data_test - train_mean)/train_std)
```

```
/Users/apoorve/anaconda/lib/python2.7/site-packages/IPython/kernel/__main__.py:3: RuntimeWarning: invalid
  app.launch_new_instance()
/Users/apoorve/anaconda/lib/python2.7/site-packages/IPython/kernel/__main__.py:4: RuntimeWarning: divide
/Users/apoorve/anaconda/lib/python2.7/site-packages/IPython/kernel/__main__.py:4: RuntimeWarning: invalid
```

```
In [21]: X_test = data_test
         y_test = label_test

         train = zip(data_train.T,label_train)
         mini_batch_size = 10

         n_input = shape(data_train)[0] # excluding bias term
         n_hidden = 100                # excluding bias term
         n_output = 10
         epochs = 10
         alpha = 0.001
         mini_batch_size = 10
```

0.1 Initializations

```
In [22]: random.seed(0)
         theta1 = matrix(random.randn(n_hidden,n_input))/sqrt(n_input)
         bias1  = matrix(random.randn(n_hidden,1))
         theta2 = matrix(random.randn(n_output,n_hidden))/sqrt(n_hidden)
         bias2  = matrix(random.randn(n_output,1))

         afunc, afuncGradient = act_funcs["leaky_relu"]
```

```
#Regularization term
lam = 0.01

#Momentum Term
gamma = 0.0
v1 = np.zeros_like(theta1)
vb1= np.zeros_like(bias1)
v2 = np.zeros_like(theta2)
vb2= np.zeros_like(bias2)
```

0.1.1 Training Code

```
In [23]: print "Layers:\t\t\t", 3
         print "Hidden Nodes:\t\t",n_hidden
         print "Epochs:\t\t\t", epochs
         print "LearningRate:\t\t",alpha
         print "Mini Batch Size:\t",mini_batch_size
         print "Activation Fn:\t\tsigmoid"
         print "Gamma(Momentum):\t",gamma
         print "Lambda(Regularize):\t",lam

         for i in range(epochs):
             random.shuffle(train)
             mini_batches = [train[k:k+mini_batch_size] for k in xrange(0,len(train),mini_batch_size)]
             for mini_batch in mini_batches:
                 d1  = np.zeros_like(theta1)
                 d2  = np.zeros_like(theta2)
                 db1 = np.zeros_like(bias1)
                 db2 = np.zeros_like(bias2)

                 for X,y in mini_batch:
                     gradTheta1, gradBias1, gradTheta2, gradBias2 = \
                         backPropGradient(X.T, y, theta1, theta2, bias1, bias2) #just one example passe
                     d1  += gradTheta1
                     db1 += gradBias1
                     d2  += gradTheta2
                     db2 += gradBias2

                 d1   = d1/mini_batch_size + lam*theta1
                 db1  = db1/mini_batch_size
                 d2   = d2/mini_batch_size + lam*theta2
                 db2  = db2/mini_batch_size

                 v1 = alpha*d1  + v1*gamma
                 vb1= alpha*db1 + vb1*gamma
                 v2 = alpha*d2  + v2*gamma
                 vb2= alpha*db2 + vb2*gamma

                 theta1 = theta1 - v1
                 bias1  = bias1  - vb1
                 theta2 = theta2 - v2
                 bias2  = bias2  - vb2
             print "Epoch",i+1,"train,test accuracy:\t",accuracy(data_train, label_train, theta1, theta
```

```
                    accuracy(data_test, label_test , theta1, theta2, bias1, bias2)

Layers:                         3
Hidden Nodes:                 100
Epochs:                         10
LearningRate:                 0.001
Mini Batch Size:         10
Activation Fn:                 sigmoid
Gamma(Momentum):         0.0
Lambda(Regularize):        0.01
Epoch 1 train,test accuracy:        0.888633333333 0.8923
Epoch 2 train,test accuracy:        0.90835 0.9108
Epoch 3 train,test accuracy:        0.9181 0.9193
Epoch 4 train,test accuracy:        0.924666666667 0.9218
Epoch 5 train,test accuracy:        0.928966666667 0.9268
Epoch 6 train,test accuracy:        0.93245 0.9291
Epoch 7 train,test accuracy:        0.935116666667 0.9329
Epoch 8 train,test accuracy:        0.937533333333 0.9346
Epoch 9 train,test accuracy:        0.939366666667 0.936
Epoch 10 train,test accuracy:        0.941183333333 0.9376

/Users/apoorve/anaconda/lib/python2.7/site-packages/IPython/kernel/__main__.py:1: RuntimeWarning: overflo
  if __name__ == '__main__':
```

0.1.2 Checking backpropgradient vs numerical gradient

```
In [ ]: X, y = train[0]
        X = X.T
        gradTheta1, gradBias1, gradTheta2, gradBias2 = backPropGradient(X,y,theta1,theta2, bias1, bias2
        numGrad1,numGradBias1,numGrad2,numGradBias2  = numericalGradient(X,y,theta1,theta2, bias1, bias
        print np.sum(np.subtract(numGradBias2,gradBias2))
        print np.sum(np.subtract(numGradBias1,gradBias1))
        print np.sum(np.subtract(numGrad2,gradTheta2))
        print np.sum(np.subtract(numGrad1,gradTheta1))
```

0.1.3 Helper Functions

```
In [5]: def accuracy(X, y, theta1, theta2, bias1, bias2):
            a1 = X
            z2 = dot(theta1,a1) + bias1
            a2 = afunc(z2)
            z3 = dot(theta2,a2) + bias2
            a3 = sigmoid(z3)

            pred = np.argmax(a3,axis=0)
            return np.sum(np.equal(pred,y.T))/float(len(y))
```

```
In [6]: def backPropGradient(X, y, theta1, theta2, bias1, bias2):
            a1 = X
            z2 = dot(theta1,a1) + bias1
            a2 = afunc(z2)
            z3 = dot(theta2,a2) + bias2
            a3 = sigmoid(z3)

            t = np.zeros_like(a3)
```

```
        for i in range(len(y)):
            t[y[i],i] = 1

        delta3 = a3-t #shape (10,1)
        delta2 = dot(theta2.T,delta3)#shape (100,10)X(10,1) = (100,1)
        delta2 = multiply(delta2,afuncGradient(z2))

        gradTheta2 = dot(delta3,a2.T)
        gradBias2  = delta3
        gradTheta1 = dot(delta2,a1.T)
        gradBias1  = delta2
        return gradTheta1, gradBias1, gradTheta2, gradBias2
```

```
In [7]: def errorFn(X, y, theta1, theta2, bias1, bias2):
            n_examples = shape(X)[1]
            a1 = X
            z2 = dot(theta1,a1) + bias1
            a2 = afunc(z2)
            z3 = dot(theta2,a2) + bias2
            a3 = sigmoid(z3)

            t = np.zeros_like(a3)
            for i in range(len(y)):
                t[y[i],i] = 1

            error = np.sum(multiply(t,log(a3)) + multiply((1-t),log(1-a3)))
            error = -error/n_examples
            return error
```

```
In [8]: def numericalGradient(X, y, theta1, theta2, bias1, bias2):
            epsilon = 10**-5
            numGrad1     = np.zeros_like(theta1)
            numGradBias1 = np.zeros_like(bias1)
            numGrad2     = np.zeros_like(theta2)
            numGradBias2 = np.zeros_like(bias2)

            for i in range(shape(theta1)[0]):
                for j in range(shape(theta1)[1]):
                    theta1_pos = np.copy(theta1)
                    theta1_neg = np.copy(theta1)
                    theta1_pos[i,j] += epsilon
                    theta1_neg[i,j] -= epsilon
                    numGrad1[i,j]    = (errorFn(X, y, theta1_pos, theta2, bias1, bias2) - \
                                        errorFn(X, y, theta1_neg, theta2, bias1, bias2))/2/epsilon
            for i in range(shape(theta2)[0]):
                for j in range(shape(theta2)[1]):
                    theta2_pos = np.copy(theta2)
                    theta2_neg = np.copy(theta2)
                    theta2_pos[i,j] += epsilon
                    theta2_neg[i,j] -= epsilon
                    numGrad2[i,j]    = (errorFn(X, y, theta1, theta2_pos, bias1, bias2) - \
                                        errorFn(X, y, theta1, theta2_neg, bias1, bias2))/2/epsilon
            for i in range(shape(bias1)[0]):
                for j in range(shape(bias1)[1]):
                    bias1_pos         = np.copy(bias1)
```

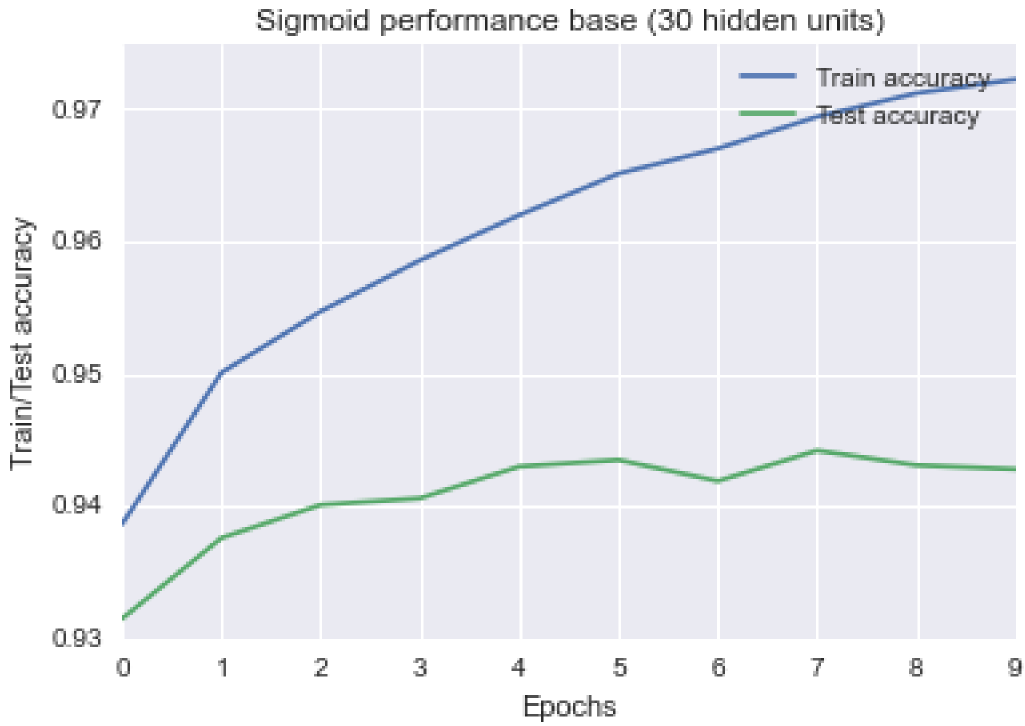
```
                bias1_neg          = np.copy(bias1)
                bias1_pos[i,j]    += epsilon
                bias1_neg[i,j]    -= epsilon
                numGradBias1[i,j] = (errorFn(X, y, theta1, theta2, bias1_pos, bias2) - \
                                     errorFn(X, y, theta1, theta2, bias1_neg, bias2))/2/epsilon
        for i in range(shape(bias2)[0]):
            for j in range(shape(bias2)[1]):
                bias2_pos          = np.copy(bias2)
                bias2_neg          = np.copy(bias2)
                bias2_pos[i,j]    += epsilon
                bias2_neg[i,j]    -= epsilon
                numGradBias2[i,j] = (errorFn(X, y, theta1, theta2, bias1, bias2_pos) - \
                                     errorFn(X, y, theta1, theta2, bias1, bias2_neg))/2/epsilon
        return numGrad1,numGradBias1,numGrad2,numGradBias2
```

```
In [9]: sigmoid = lambda z: 1.0/(1.0+np.exp(-z))
        sigmoid_prime = lambda z: multiply(sigmoid(z),(1-sigmoid(z)))
        ftanh = lambda z: np.tanh(z)
        ftanh_prime = lambda z: 1 - multiply(ftanh(z),ftanh(z))
        funny_tanh = lambda z: 1.7159 * np.tanh(2.0/3.0 * z) + .001*z
        funny_tanh_prime = lambda z: 1.7159 * 2.0 / 3.0 * (1.0 / multiply(np.cosh(2.0/3.0 * z),np.cosh(
        relu = lambda z: multiply(z,(z > 0))
        relu_prime = lambda z: z >= 0
        leaky_relu = lambda z: np.maximum(.1*z, z)
        leaky_relu_prime = lambda z: 1*(z>=0) + .1*(z<0)

        act_funcs = {'sigmoid': (sigmoid, sigmoid_prime),
                     'ftanh': (ftanh, ftanh_prime),
                     'funny_tanh': (funny_tanh, funny_tanh_prime),
                     'relu': (relu,relu_prime),
                     'leaky_relu': (leaky_relu,leaky_relu_prime)}
```

```
In [36]: import numpy as np
         import matplotlib.pyplot as plt
```

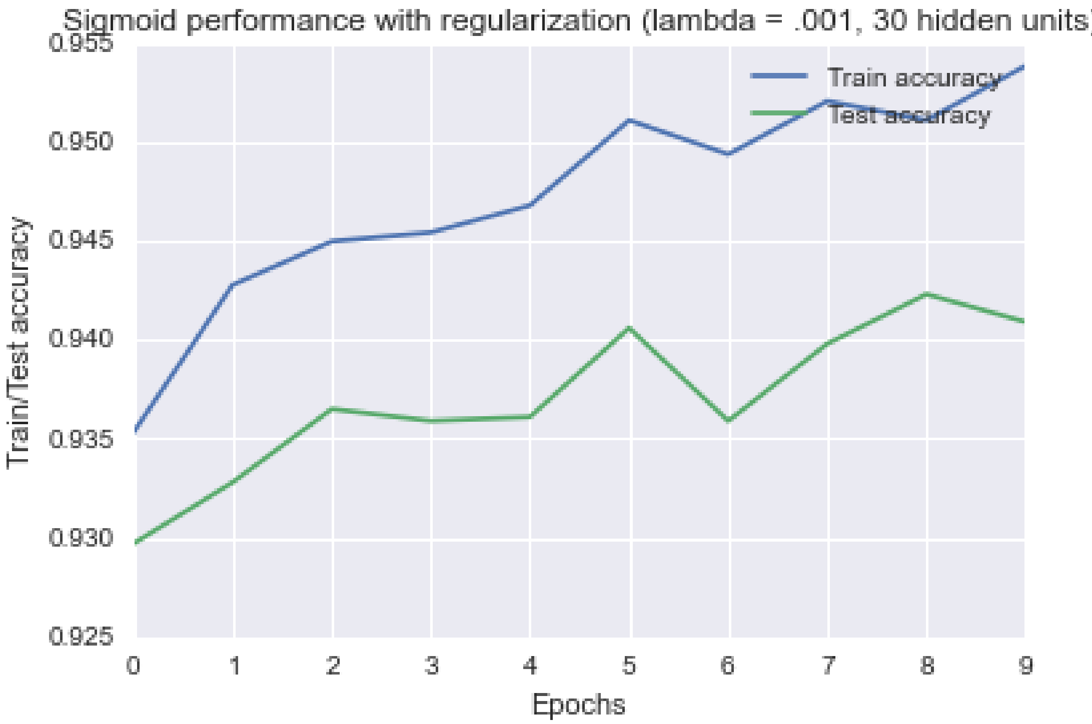
```
In [57]: x = [i for i in xrange(10)]

         sigmoid_train = ['0.93865', '0.950083333333', '0.954716666667', '0.958566666667', '0.96198333
         sigmoid_test = ['0.9315', '0.9376', '0.9401', '0.9406', '0.943', '0.9435', '0.9419', '0.9442',

         sns.plt.xlabel("Epochs")
         sns.plt.ylabel("Train/Test accuracy")
         sns.plt.plot(x, sigmoid_train,  label = "Train accuracy" )
         sns.plt.plot(x, sigmoid_test,  label = "Test accuracy" )
         sns.plt.title("Sigmoid performance base (30 hidden units)")
         sns.plt.legend()
```

```
Out[57]: <matplotlib.legend.Legend at 0x125ff4450>
```

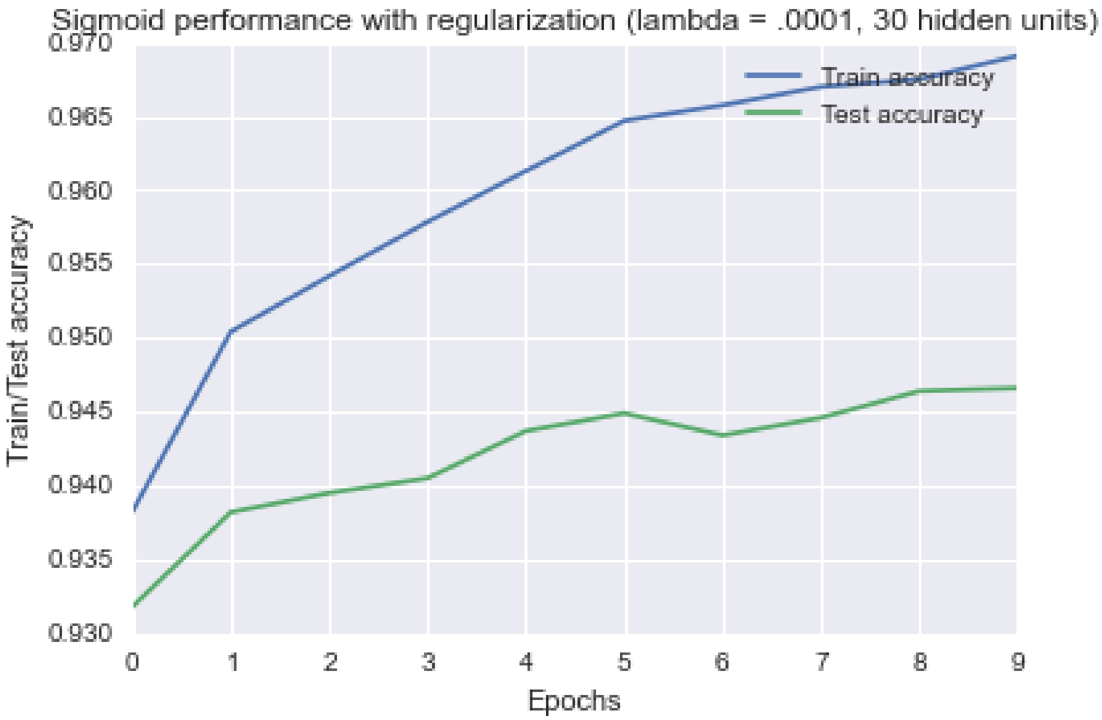


In [63]:
```
x = [i for i in xrange(10)]

sigmoid_train = ['0.935316666667', '0.942766666667', '0.944966666667', '0.945416666667', '0.94
sigmoid_test = ['0.9297', '0.9328', '0.9365', '0.9359', '0.9361', '0.9406', '0.9359', '0.9398'

sns.plt.xlabel("Epochs")
sns.plt.ylabel("Train/Test accuracy")
sns.plt.plot(x, sigmoid_train,  label = "Train accuracy" )
sns.plt.plot(x, sigmoid_test,  label = "Test accuracy" )
sns.plt.title("Sigmoid performance with regularization (lambda = .001, 30 hidden units)")
sns.plt.legend()
```

Out[63]: <matplotlib.legend.Legend at 0x12780c650>

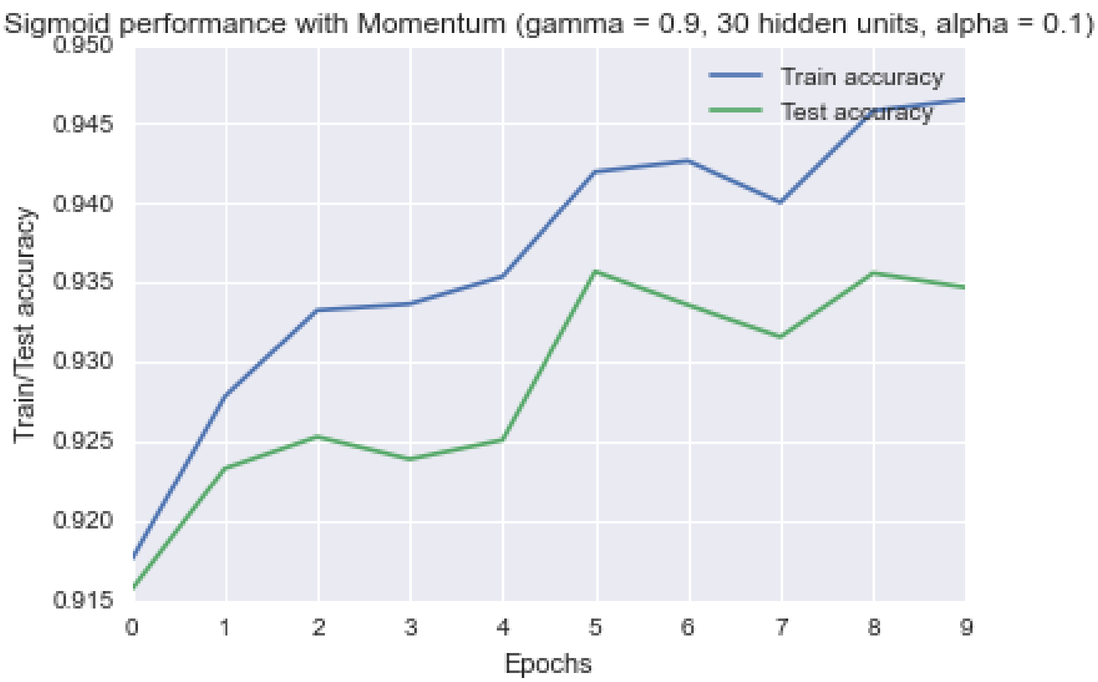



```
In [65]: x = [i for i in xrange(10)]

         sigmoid_train = ['0.938233333333', '0.950416666667', '0.954183333333', '0.95785', '0.9613', '0
         sigmoid_test = ['0.9318', '0.9382', '0.9395', '0.9405', '0.9437', '0.9449', '0.9434', '0.9446'

         sns.plt.xlabel("Epochs")
         sns.plt.ylabel("Train/Test accuracy")
         sns.plt.plot(x, sigmoid_train,  label = "Train accuracy" )
         sns.plt.plot(x, sigmoid_test,  label = "Test accuracy" )
         sns.plt.title("Sigmoid performance with regularization (lambda = .0001, 30 hidden units)")
         sns.plt.legend()

Out[65]: <matplotlib.legend.Legend at 0x127086ed0>
```



In [69]:
```
x = [i for i in xrange(10)]

sigmoid_train = ['0.917583333333', '0.9278', '0.933266666667', '0.933666666667', '0.93538333333
sigmoid_test = ['0.9157', '0.9233', '0.9253', '0.9239', '0.9251', '0.9357', '0.9336', '0.9316'

sns.plt.xlabel("Epochs")
sns.plt.ylabel("Train/Test accuracy")
sns.plt.plot(x, sigmoid_train,  label = "Train accuracy" )
sns.plt.plot(x, sigmoid_test,  label = "Test accuracy" )
sns.plt.title("Sigmoid performance with Momentum (gamma = 0.9, 30 hidden units, alpha = 0.1)")
sns.plt.legend()
```

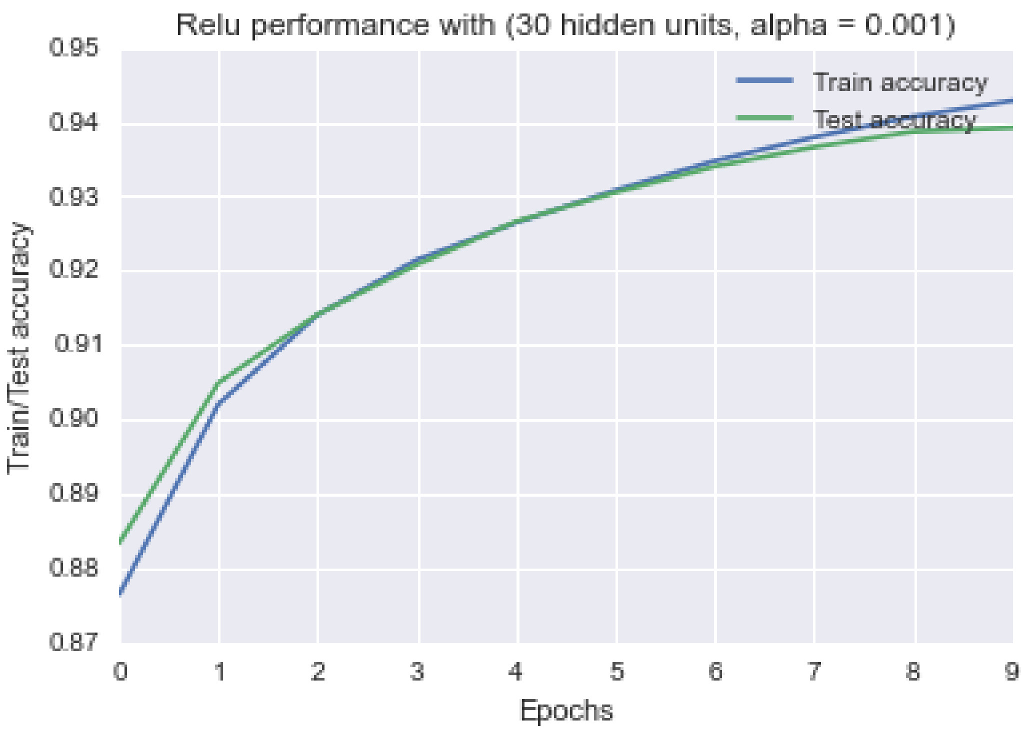
Out[69]:
```
<matplotlib.legend.Legend at 0x126477550>
```




```
In [81]: x = [i for i in xrange(10)]

         sigmoid_train = ['0.929516666667', '0.9389', '0.944583333333', '0.946416666667', '0.95126666666
         sigmoid_test = ['0.9219', '0.9274', '0.9344', '0.9338', '0.9341', '0.9374', '0.9348', '0.9364'

         sns.plt.xlabel("Epochs")
         sns.plt.ylabel("Train/Test accuracy")
         sns.plt.plot(x, sigmoid_train,  label = "Train accuracy" )
         sns.plt.plot(x, sigmoid_test,  label = "Test accuracy" )
         sns.plt.title("Tanh performance with 30 hidden units, alpha = 0.1")
         sns.plt.legend()

Out[81]: <matplotlib.legend.Legend at 0x125f99ed0>
```

In [93]:
```
x = [i for i in xrange(10)]

sigmoid_train = ['0.87645', '0.902033333333', '0.91405', '0.92145', '0.92645', '0.930783333333
sigmoid_test = ['0.8834', '0.9049', '0.9141', '0.9208', '0.9266', '0.9305', '0.934', '0.9366',

sns.plt.xlabel("Epochs")
sns.plt.ylabel("Train/Test accuracy")
sns.plt.plot(x, sigmoid_train,  label = "Train accuracy" )
sns.plt.plot(x, sigmoid_test,  label = "Test accuracy" )
sns.plt.title("Relu performance with (30 hidden units, alpha = 0.001)")
sns.plt.legend()
```

Out[93]: <matplotlib.legend.Legend at 0x127d03910>

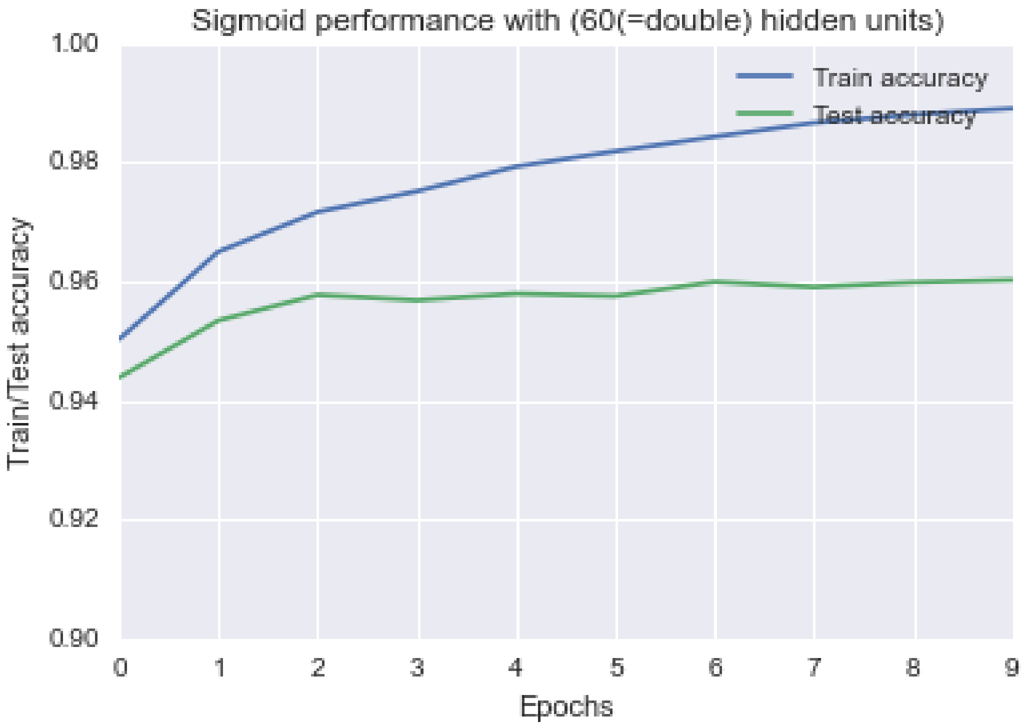


In [94]:
```
x = [i for i in xrange(10)]

sigmoid_train = ['0.920333333333', '0.9241', '0.930916666667', '0.935383333333', '0.9391', '0.9
sigmoid_test = ['0.9153', '0.9164', '0.9182', '0.9229', '0.9251', '0.9261', '0.9264', '0.9261'

sns.plt.xlabel("Epochs")
sns.plt.ylabel("Train/Test accuracy")
sns.plt.plot(x, sigmoid_train,  label = "Train accuracy" )
sns.plt.plot(x, sigmoid_test,  label = "Test accuracy" )
sns.plt.title("Sigmoid performance with (30(=half) hidden units)")
sns.plt.legend()
```

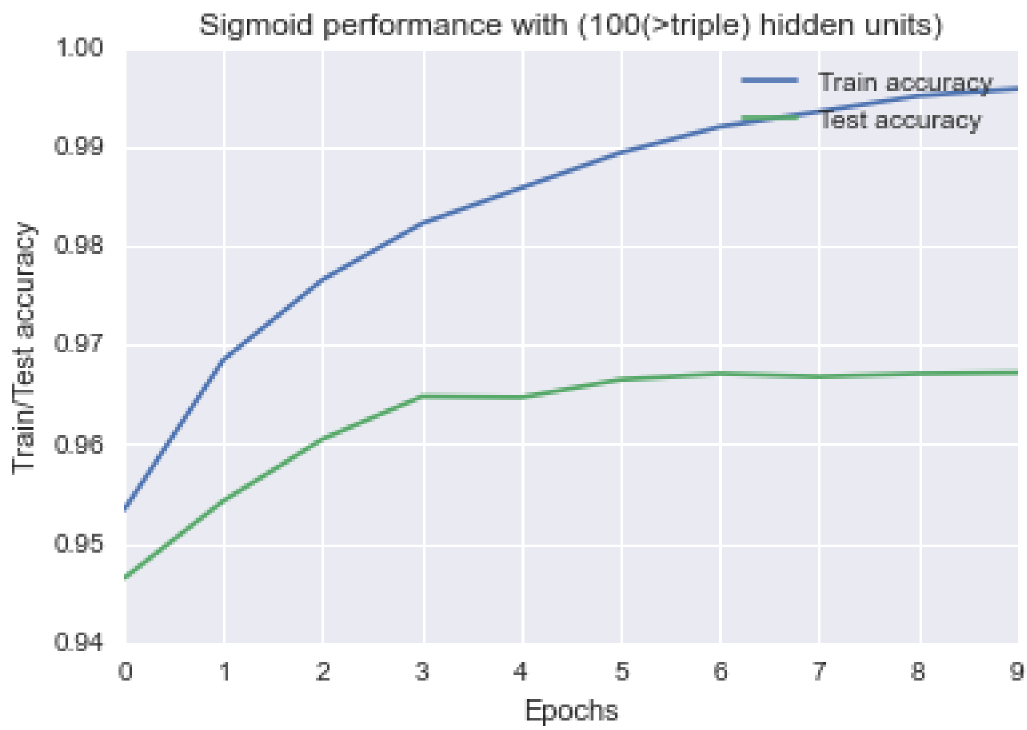
Out[94]: <matplotlib.legend.Legend at 0x127b8f650>

In [91]:
```
x = [i for i in xrange(10)]

sigmoid_train = ['0.950233333333', '0.964933333333', '0.9716', '0.975083333333', '0.97923333333
sigmoid_test = ['0.9438', '0.9534', '0.9577', '0.9568', '0.9579', '0.9575', '0.9599', '0.959',

sns.plt.xlabel("Epochs")
sns.plt.ylabel("Train/Test accuracy")
sns.plt.plot(x, sigmoid_train,  label = "Train accuracy" )
sns.plt.plot(x, sigmoid_test,  label = "Test accuracy" )
sns.plt.title("Sigmoid performance with (60(=double) hidden units)")
sns.plt.legend()
```

Out[91]: <matplotlib.legend.Legend at 0x10a644f10>




```
In [92]: x = [i for i in xrange(10)]

         sigmoid_train = ['0.9534', '0.9685', '0.976583333333', '0.98225', '0.985833333333', '0.9893833
         sigmoid_test = ['0.9465', '0.9543', '0.9605', '0.9648', '0.9647', '0.9665', '0.9671', '0.9668'

         sns.plt.xlabel("Epochs")
         sns.plt.ylabel("Train/Test accuracy")
         sns.plt.plot(x, sigmoid_train,  label = "Train accuracy" )
         sns.plt.plot(x, sigmoid_test,  label = "Test accuracy" )
         sns.plt.title("Sigmoid performance with (100(>triple) hidden units)")
         sns.plt.legend()

Out[92]: <matplotlib.legend.Legend at 0x12710df10>
```


Sigmoid performance with (100(>triple) hidden units)
Train accuracy
Test accuracy
Train/Test accuracy
Epochs
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0
1
2
3
4
5
6
7
8
9