hw6-25912237

November 23, 2015

All relevant code is in this notebook and two files ***neuralnet.py*** *and* ***loader.py****.*

```
In [50]: %matplotlib inline
         %load_ext autoreload
         import numpy as np
         import pandas as pd
         import scipy as sp
         import scipy.io
         import datetime
         from sklearn.cross_validation import train_test_split
         from sklearn.preprocessing import normalize
         from sklearn.utils import shuffle

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

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

```
In [51]: import loader
         import neuralnet
         %aimport neuralnet
         %aimport loader
         %aimport
```

```
Modules to reload:
loader neuralnet

Modules to skip:
```

1 Problem 1

$ y_0 = 784x1, y_1 = 200x1, y_2 = 10 x 1$
$z_1 = 200x1, z_2 = 10x1$
$b_1 = 200x1, b_2 = 10x1$
$w_1 = 784x200, w_2 = 200x1$
$g = \frac{1}{1+e^{-z}}$
$t = tanh(z)$
Rather than adding a neuron, I've elected to just add an arbitrary, updating value to each z. Which is equivalent to adding another neuron.

$$\frac{\partial L}{w^l} = y^{l-1} \frac{\partial L}{\partial z^l}^T$$

$$\frac{\partial L}{w^2} = y^1 \frac{\partial L}{\partial z^2}^T$$

$$= y^1(diag(g'(z^2 + b^2)) \cdot \frac{\partial L}{\partial y^2})^T$$

$$w^2 \leftarrow w^2 - \eta * \frac{\partial L}{w^2}$$

$$\frac{\partial L}{w^1} = y^0 \frac{\partial L}{\partial z^1}^T$$

$$= y^0(diag(t'(z^1 + b^1)) \cdot \frac{\partial L}{\partial y^1})^T$$

$$= y^0(diag(t'(z^1 + b^1)) \cdot w^2 \cdot \frac{\partial L}{\partial z^2})^T$$

$$w^1 \leftarrow w^1 - \eta * \frac{\partial L}{w^l}$$

The benefit of backprop is being able to cache the results from the previous delta.

1.0.1 Squared Loss:

$$\frac{\partial L}{w^l} = y^0(diag(1 - tanh^2(z^1 + b^1)) \cdot w^2 \cdot (diag(g(z^2 + b^2)(1 - g(z^2 + b^2))) \cdot (y^2 - y)))^T$$

$$\frac{\partial L}{w^2} = y^1(diag(g(z^2 + b^2)(1 - g(z^2 + b^2)) \cdot (y^2 - y)))^T$$

1.0.2 Cross-Entropy Loss:

$$\frac{\partial L}{w^l} = y^0(diag(1 - tanh^2(z^1 + b^1)) \cdot w^2 \cdot (diag(g(z^2 + b^2)(1 - g(z^2 + b^2))) \cdot -(\frac{y}{y^2} - \frac{1 - y}{1 - y^2})))^T$$

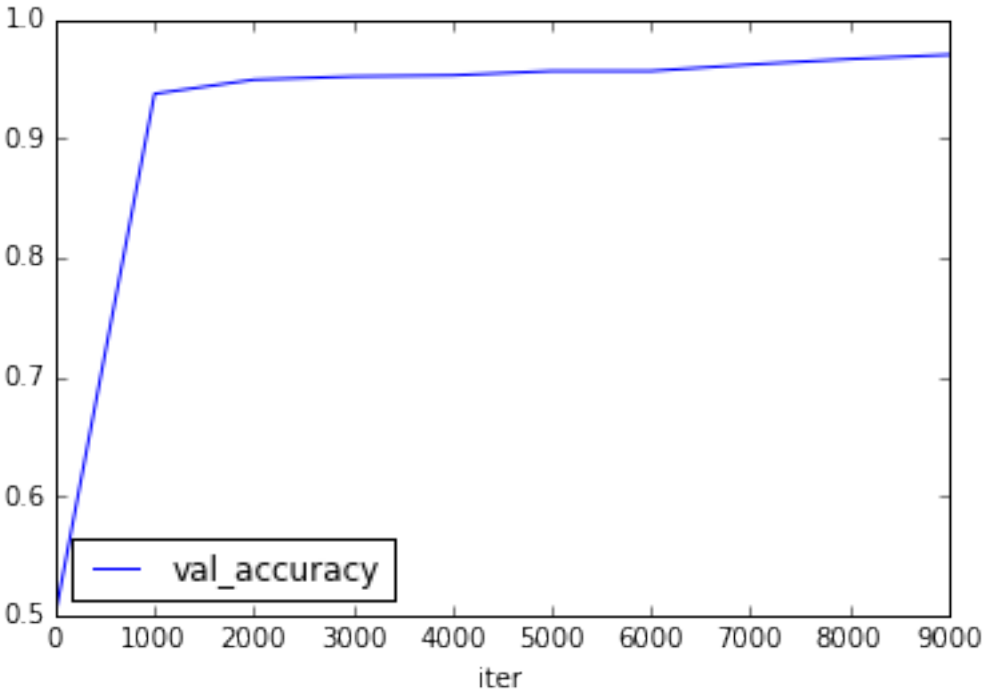
$$\frac{\partial L}{w^2} = y^1(diag(g(z^2 + b^2)(1 - g(z^2 + b^2)) \cdot -(\frac{y}{y^2} - \frac{1 - y}{1 - y^2})))^T$$

In [52]:
```
def squared_error(y, y_hat):
    return (y - y_hat).dot((y - y_hat))

def squared_error_prime(y, y_hat):
    return -1 * (y - y_hat)

def cross_ent(y, y_hat):
    part1 = y.dot(np.log(y_hat))
    part2 = (1.0 - y).dot(np.nan_to_num(np.log(1.0 - y_hat)))
    return -(part1 + part2)

def cross_ent_prime(y, y_hat):
    part1 = y / y_hat
    part2 = (1 - y)/(1 - y_hat)
    return -(part1 - part2)

def t(z):
    return np.tanh(z)

def t_prime(z):
```

```
        return 1 - np.power(t(z), 2)

    def g(z):
        return (1.0 / (1.0 + np.exp(-z)))

    def g_prime(z):
        return g(z) * (1 - g(z))
```

In [55]:
```
%autoreload
input_dim = len(X[0])
output_dim = len(y[0])
hl_size = 50
layer1 = neuralnet.Layer((input_dim, hl_size), t, t_prime)
layer2 = neuralnet.OutputLayer((hl_size, output_dim), g, g_prime)
nn = neuralnet.NeuralNet(layer1, layer2, cross_ent, cross_ent_prime)
nn.train(X, y, XVal=X, yVal=y, num_iters=10000)
nn.resulting_scores()[['val_accuracy', 'iter']].plot(x='iter')
print("Parameters used", nn.train_params)
print("Training Accuracy", nn.score(X,y)/len(X))
```

```
Total Train Time 1.48 Seconds
Parameters used {'end': datetime.datetime(2015, 11, 23, 13, 41, 26, 357580), 'start': datetime.datetime
Training Accuracy 0.969090909091
/Users/bill_chambers/AeroFS/Dev/ml289/hw6/neuralnet.py:109: FutureWarning: comparison to `None` will res
  if XVal != None:
```



2 Problem 2

1. Parameters are inline below, weights are initialized normally around $\mu = 1, \sigma = 0.01$

2. Training and Validation Accuracies are listed below.
3. Train time is listed below.
4. Plots are below. I only report every 20,000 iterations.
5. The cross entropy seemed to perform much better. Simply in terms of the equation it seems to be a more sophisticated measure of the loss. Therefore we could expect it to perform better. This aligned with the results, as it allowed me to descend faster and get to a much smaller error rate overall.

Kaggle Score for 200k iterations, 0.01 learning rate, squared loss = .929
Kaggle Score for 500k iterations, 0.01 learning rate, cross entropy loss = 0.97460
it's interesting to note that these had the exact same initialization values (for w's) Even with 200k, the cross entropy loss was far superior.

```
In [56]: train_dict = loader.load_training_data() # loader.py
         X = train_dict['X']
         y = train_dict['y']
         print("Mean of y:", y.mean())
         y = pd.get_dummies(y).values

         XVal = train_dict['XVal']
         yVal = pd.get_dummies(train_dict['yVal']).values

         print("Num Training Examples:", len(X))
```

```
Mean of y: 4.44715555556
Num Training Examples: 45000
```

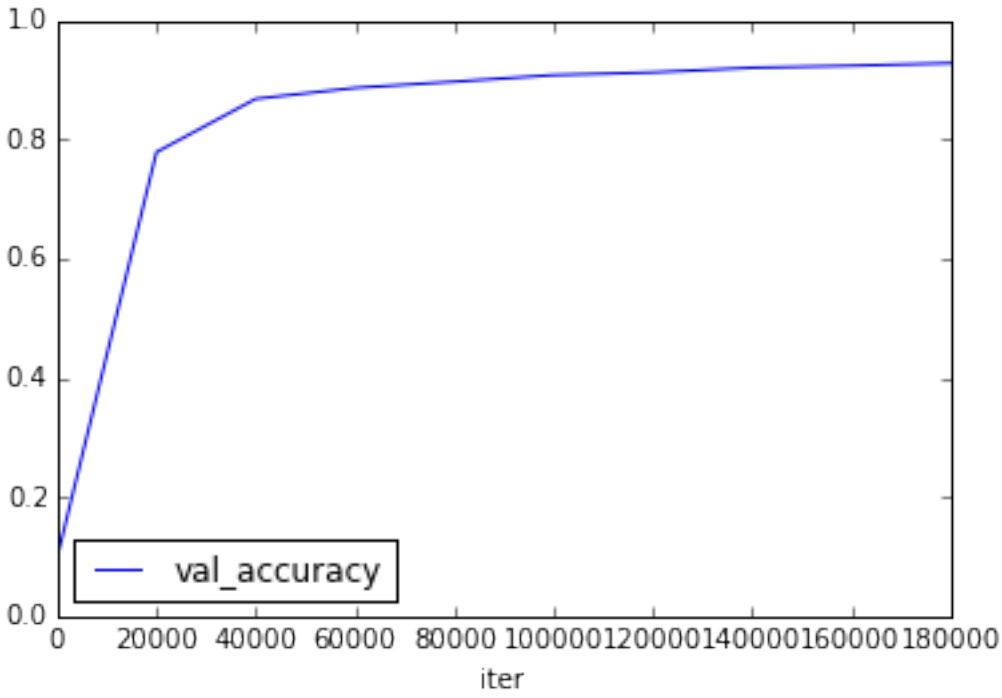
2.1 Squared Error Loss

```
In [59]: %autoreload
         input_dim = len(X[0])
         output_dim = len(y[0])
         hl_size = 200

         layer1 = neuralnet.Layer((input_dim, hl_size), t, t_prime)
         layer2 = neuralnet.OutputLayer((hl_size, output_dim), g, g_prime)

         nn = neuralnet.NeuralNet(layer1, layer2, squared_error, squared_error_prime)
         nn.train(X, y, num_iters=200000, score_every=20000, eta=0.01, XVal=XVal, yVal=yVal)
```

```
Total Train Time 352.62 Seconds

/Users/bill_chambers/AeroFS/Dev/ml289/hw6/neuralnet.py:109: FutureWarning: comparison to `None` will res
  if XVal != None:
```

```
In [60]: print("Parameters used", nn.train_params)
         nn.resulting_scores()[['val_accuracy', 'iter']].plot(x='iter')
         print("Training Accuracy", nn.score(X,y)/len(X))
         print("Validation Accuracy", nn.score(XVal,yVal)/len(XVal))
```

```
Parameters used {'score_every': 20000, 'start': datetime.datetime(2015, 11, 23, 13, 42, 11, 482967), 'et
Training Accuracy 0.9338
Validation Accuracy 0.9278
```



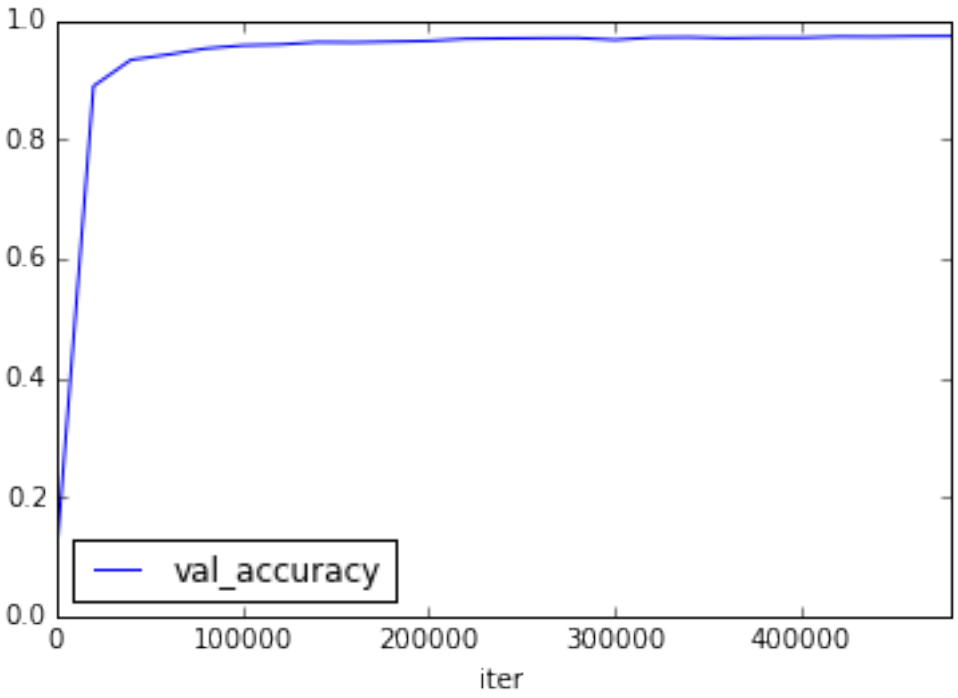
2.2 Cross Entropy Loss

```
In [64]: %autoreload
         input_dim = len(X[0])
         output_dim = len(y[0])
         hl_size = 200

         layer1 = neuralnet.Layer((input_dim, hl_size), t, t_prime)
         layer2 = neuralnet.OutputLayer((hl_size, output_dim), g, g_prime)

         nn = neuralnet.NeuralNet(layer1, layer2, cross_ent, cross_ent_prime)
         nn.train(X, y, num_iters=500000, score_every=20000, eta=0.01, XVal=XVal, yVal=yVal)
```

```
Total Train Time 656.30 Seconds

/Users/bill_chambers/AeroFS/Dev/ml289/hw6/neuralnet.py:109: FutureWarning: comparison to `None` will res
  if XVal != None:
```

```
In [65]: print("Parameters used", nn.train_params)
         nn.resulting_scores()[['val_accuracy', 'iter']].plot(x='iter')
         print("Training Accuracy", nn.score(X,y)/len(X))
         print("Validation Accuracy", nn.score(XVal,yVal)/len(XVal))
```

```
Parameters used {'score_every': 20000, 'start': datetime.datetime(2015, 11, 23, 14, 0, 31, 23813), 'eta'
Training Accuracy 0.997533333333
Validation Accuracy 0.972333333333
```



In [66]:
```
kaggleTest = loader.load_test_data()
preds = nn.predict(kaggleTest)
df = pd.DataFrame(preds).reset_index()
df.columns = ['id', 'category']
df.id = df.id + 1
df.to_csv("cross_ent 500000 iters.csv",index=None)
```