Supervised Image Classification Using an Artificial Neural Network for Optical Digit Recognition and Diagnosis of Fine Needle Aspirates of Breast Cancer

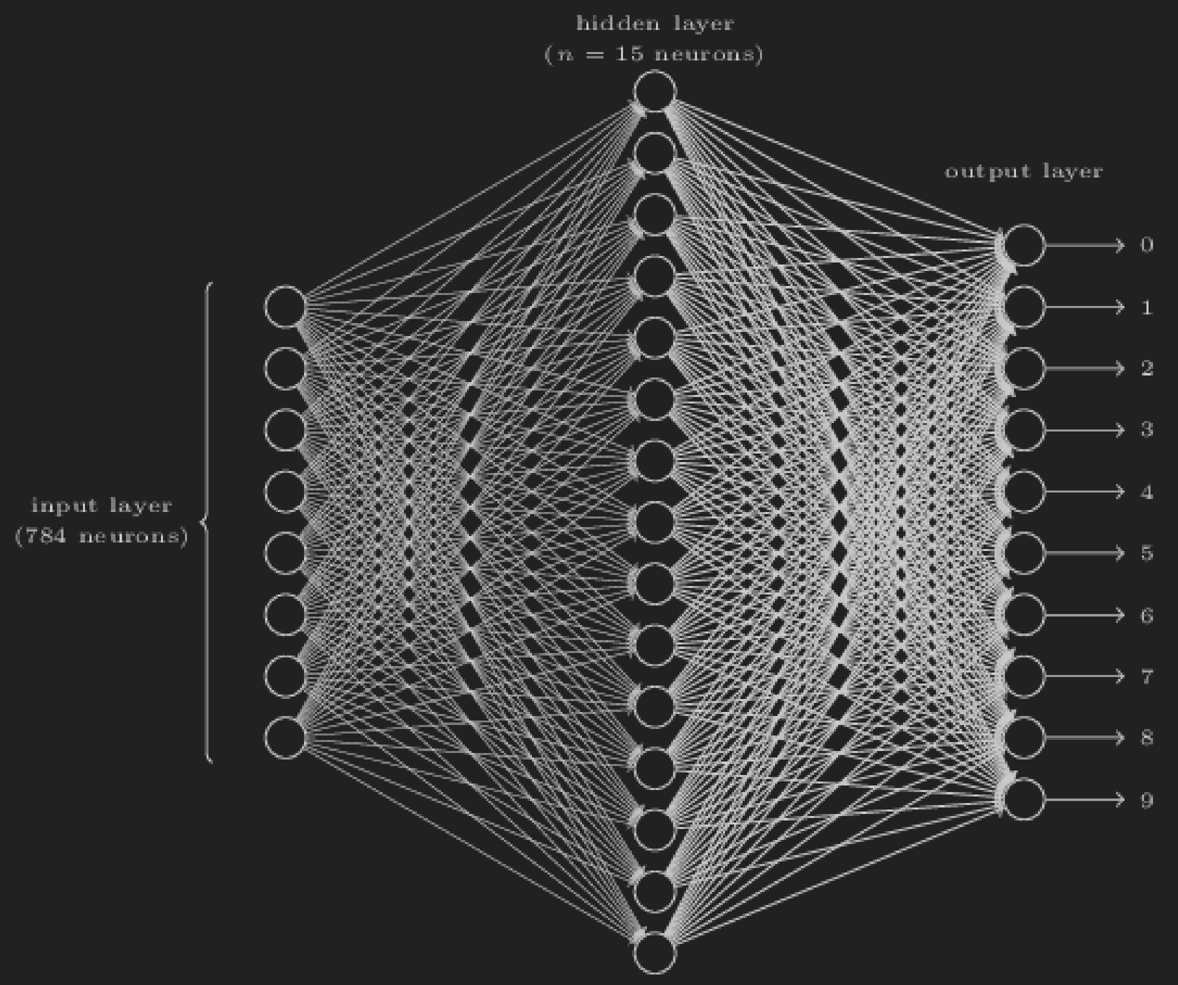
Somil Govani

The Multivariate Conundrum

- Linear or limited-variable functions are easy to evaluate
- Learning problems of classification and continuous predictions become difficult
- Humans are particularly good at this

Artificial Neural Networks (ANN)

- Machine Learning model loosely inspired by biological structure of brain
- Takes inputs, transforms and weights, gives outputs



Training & Learning Model

- Training algorithm uses supervised data to set weights
- Constructs an approximate multivariate function

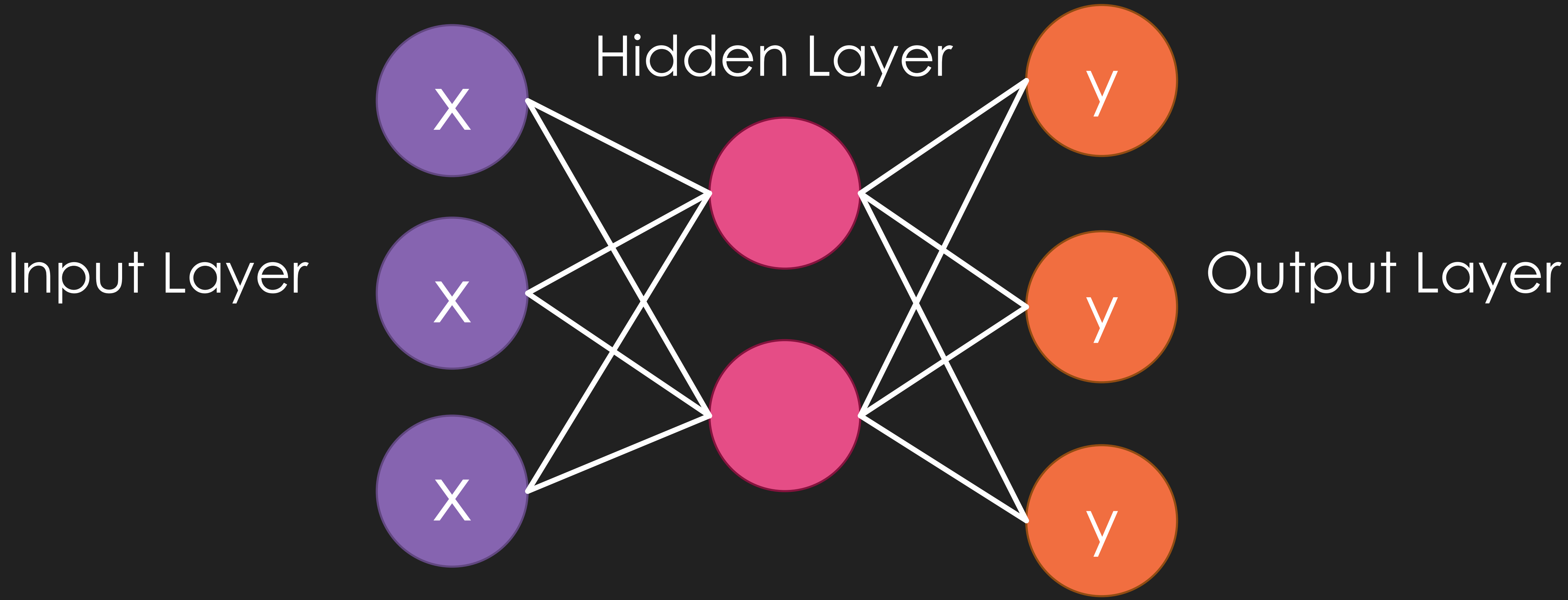
Practical Applications

- Optical Character Recognition (OCR)
- Stock Market Prediction Analysis
- Travelling Salesman Problem (TSP)
- Other multivariate fuzzy predictions
- Image Classification (Digits, Medical Images, etc.)

Goal

To develop a general-purpose, extensible artificial neural network for the supervised analysis and classification of images. To apply this ANN for Optical Digit Classification and categorical diagnosis of fine needle aspirates of breast cancer tumors.

Network Class Constructor

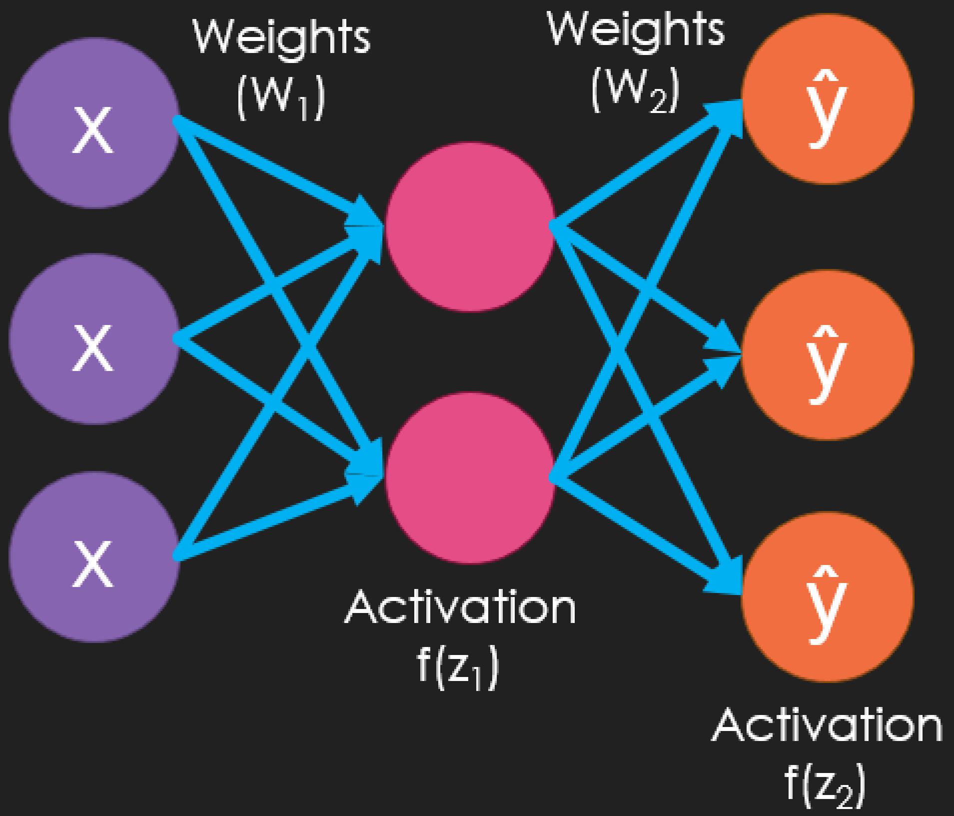


Network Class Constructor

```python
class Neural_Network(object):
    #Intialize neural network object (requires length of square image)
    def __init__ (self, imageSize, hLayer=10, Lambda=0):
        self.inputLayerSize = imageSize**2
        self.outputLayerSize = 10
        #Set number of neurons in hidden layer to mean of input layer and output layer
        self.hiddenLayerSize = hLayer

        self.W1 = np.random.randn(self.inputLayerSize, self.hiddenLayerSize)
        self.W2 = np.random.randn(self.hiddenLayerSize, self.outputLayerSize)
        self.Lambda = Lambda
```

Forward Propagation

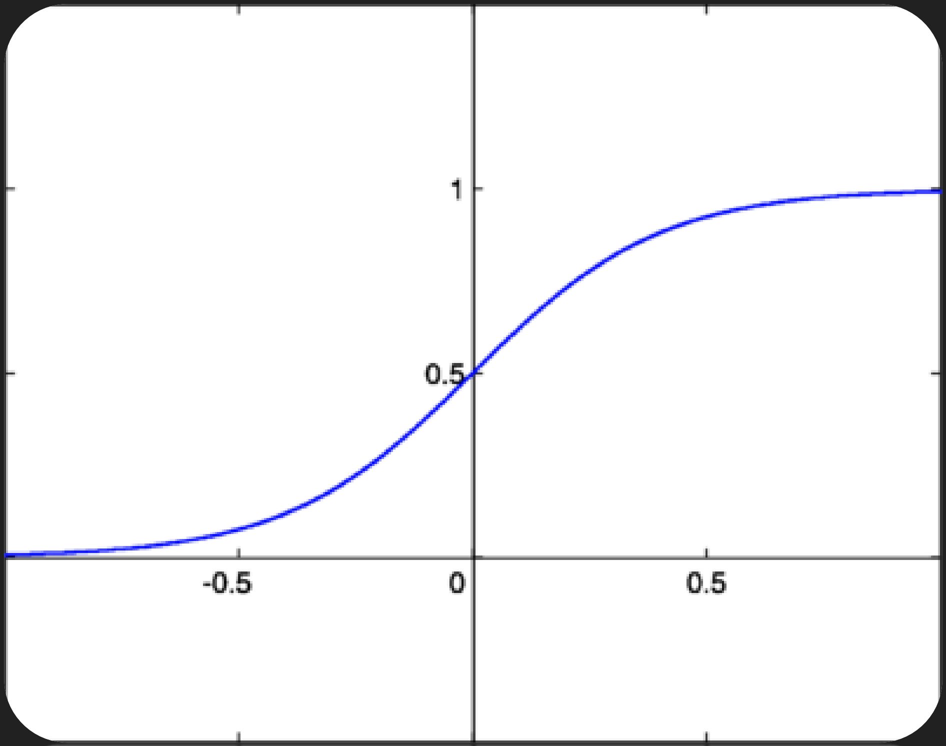
$$z_1 = XW_1$$
$$a = f(z_1)$$
$$z_2 = aW_2$$
$$\hat{y} = f(z_2)$$
$$\hat{y} = f(f(XW_1)W_2)$$



Sigmoid Activation Function f(z)

- Differentiable and introduces non-linear attributes

$$f(z) = \frac{1}{1+e^{-z}} \qquad f'(z) = \frac{-e^{-z}}{(1+e^{-z})^2}$$

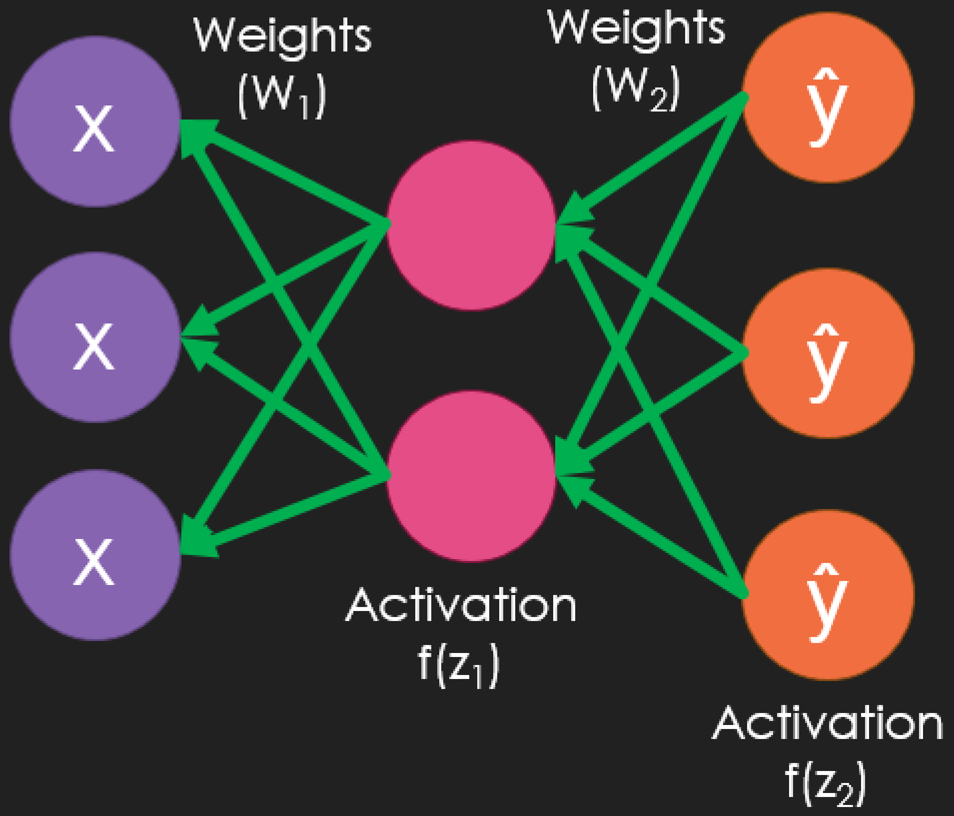


Forward Propagation Method

```
#Differentiated Sigmoid activation function
def sigmoidPrime(self, z):
    return (np.exp(-z) / ((1.0+np.exp(-z))**2))

#Sigmoid Activation function
def sigmoid(self, z):
    return 1.0 / (1.0 + np.exp(-z))

def forward(self, x):
    self.z2 = np.dot(x, self.W1)
    self.a2 = self.sigmoid(self.z2)
    self.z3 = np.dot(self.a2, self.W2)
    yHat = self.sigmoid(self.z3)
    return yHat
```

Backwards Propagation

- Goal is tune weights in order to minimize the error of predicted outputs
- Use differential analysis to compute gradients



Cross Entropy Cost Function

- Chosen for categorical outputs

$$J = -\frac{1}{N}\sum y_n \ln(\hat{y}_n) + (1 - y_n)\ln(1 - \hat{y}_n)$$

Compute Average

Compute Cost for Correct Node

Compute Cost for Incorrect Nodes

Minimizing Cost using a Computed Gradient

- Iteratively computes gradient of cost function
- Approximates Newton's Method and alters weights to approach minimum of cost function
- Use gradient to move in negative direction

$$\nabla J = \left(\frac{\partial J}{\partial W_1}, \frac{\partial J}{\partial W_2} \right)$$

Computing Partial Derivative of Cost Function in Respect to W_2

$$J = -\frac{1}{N}\sum y_n \ln(\hat{y}_n) + (1 - y_n)\ln(1 - \hat{y}_n)$$

$$\frac{\partial J}{\partial W_2} = -\frac{1}{N}\sum \frac{y}{\hat{y}} * \frac{\partial \hat{y}}{\partial W_2} + \frac{1-y}{1-\hat{y}} * \frac{-\partial \hat{y}}{\partial W_2}$$

$$\frac{\partial J}{\partial W_2} = -\frac{1}{N}\sum \left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right]\frac{\partial \hat{y}}{\partial W_2}$$

$$\frac{\partial J}{\partial W_2} = -\frac{1}{N}\sum \left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right]\frac{\partial \hat{y}}{\partial z_2}\frac{\partial z_2}{\partial W_2}$$

$$\frac{\partial J}{\partial W_2} = -\frac{1}{N}\sum \left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right] f'(z_2)a$$

$$\frac{\partial J}{\partial W_2} = -\frac{1}{N}\sum \left[\frac{y-\hat{y}}{\hat{y}(1-\hat{y})}\right] f'(z_2)\, a$$

$$\boldsymbol{\frac{\partial J}{\partial W_2} = a^T \cdot \frac{y - \hat{y}}{N}}$$

Computing Partial Derivative of Cost Function in Respect to W_1

$$J = -\frac{1}{N}\sum y_n \ln(\hat{y}_n) + (1 - y_n)\ln(1 - \hat{y}_n)$$

$$\frac{\partial J}{\partial W_1} = -\frac{1}{N}\sum \left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right]\frac{\partial \hat{y}}{\partial W_1}$$

$$\frac{\partial J}{\partial W_1} = -\frac{1}{N}\sum \left[\frac{y}{\hat{y}} - \frac{1-y}{1-\hat{y}}\right]\frac{\partial \hat{y}}{\partial z_2}\frac{\partial z_2}{\partial a}\frac{\partial a}{\partial z_1}\frac{\partial z_1}{\partial W_2}$$

$$\boldsymbol{\frac{\partial J}{\partial W_1} = X^T \cdot \left[\frac{[\hat{y} - y]}{N} \cdot {W_2}^T \cdot f'(z_2)\right]}$$

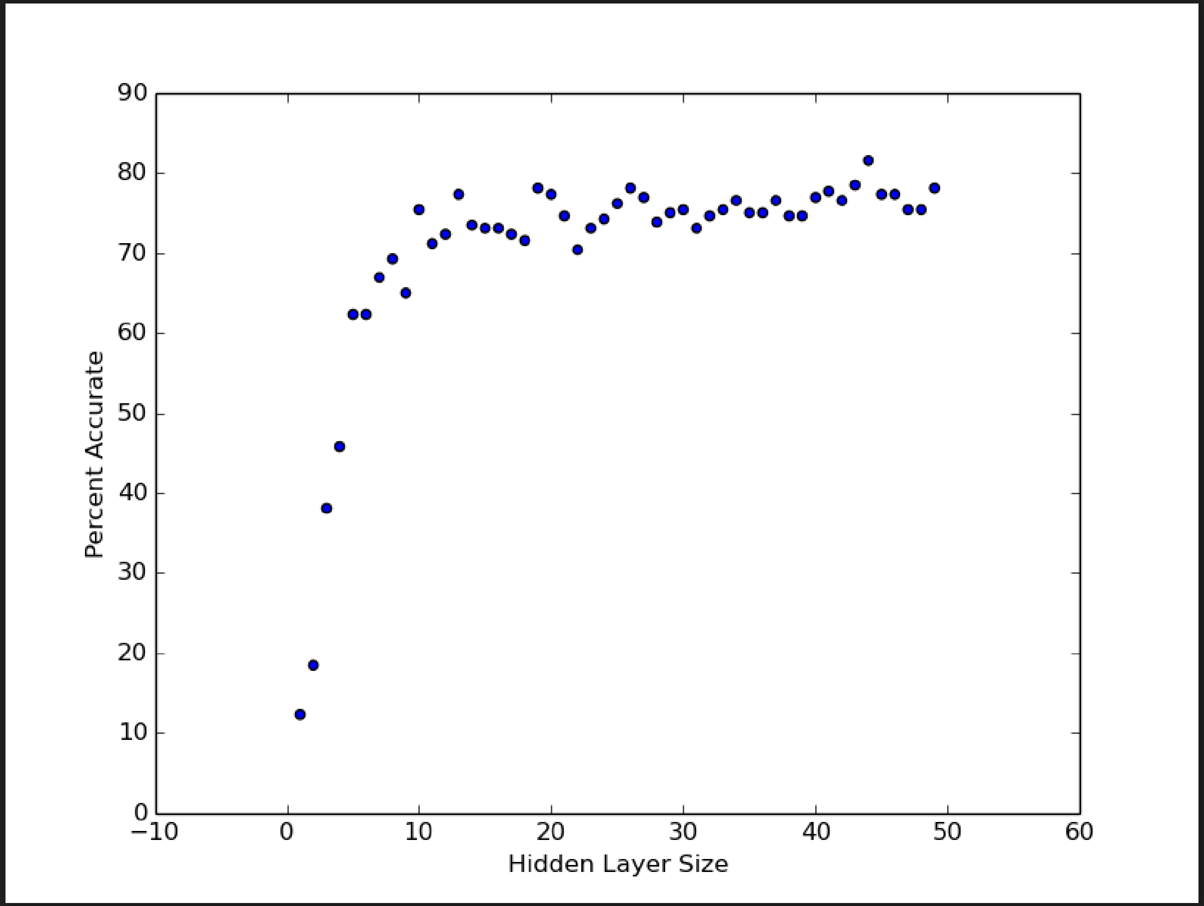
```
#Cross Entropy cost function
def cost(self, x, y, outPut=False, test=False):
    self.yHat = self.forward(x)
    if outPut:
        print self.yHat
    J = (-1.0/len(x)) * sum(sum(y * np.log(self.yHat)
        + (1-y)*np.log(1-self.yHat)))
    #Regularizes cost function to prevent overfitting
    regularize = (self.Lambda/2.0/len(x)) *
        (sum(sum(self.W1**2)) + sum(sum(self.W2**2)))
    if test:
        regularize = 0
    return J + regularize

#Compute derivative of cost function
def costPrime(self, x, y):
    self.yHat = self.forward(x)
    backError2 = (y-self.yHat)/(-float(len(x)))
    dJdW2 = np.dot(self.a2.transpose(), backError2)
        + (self.Lambda*self.W2)/(len(x))
    backError1 = np.dot(backError2, self.W2.transpose())
        * self.sigmoidPrime(self.z2)
    dJdW1 = np.dot(x.transpose(), backError1)
        + (self.Lambda*sum(sum(self.W1)))/(len(x))
    return dJdW1, dJdW2
```

Training

- In order to carryout supervised backpropagation, supervised training data is necessary
- The more training data the more accurate the approximation
- Use training data to minimize cost using BFGS/CG algorithm

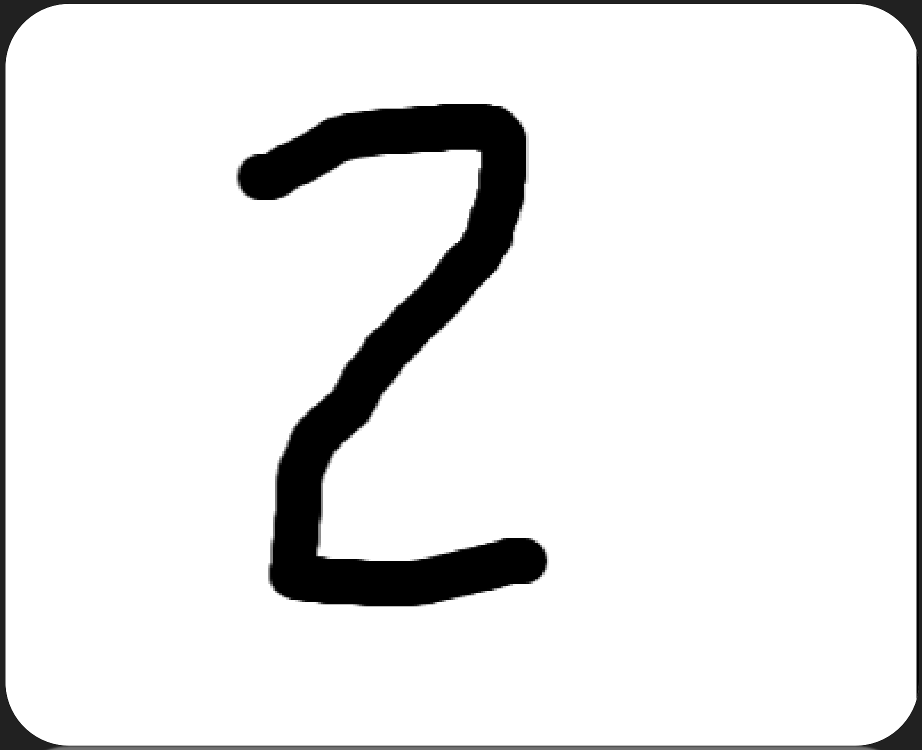
Determining Hyperparameters

- % Accuracy vs Hidden Layer Size
- Constant iterations and samples
- Change is negligible after approximately 10



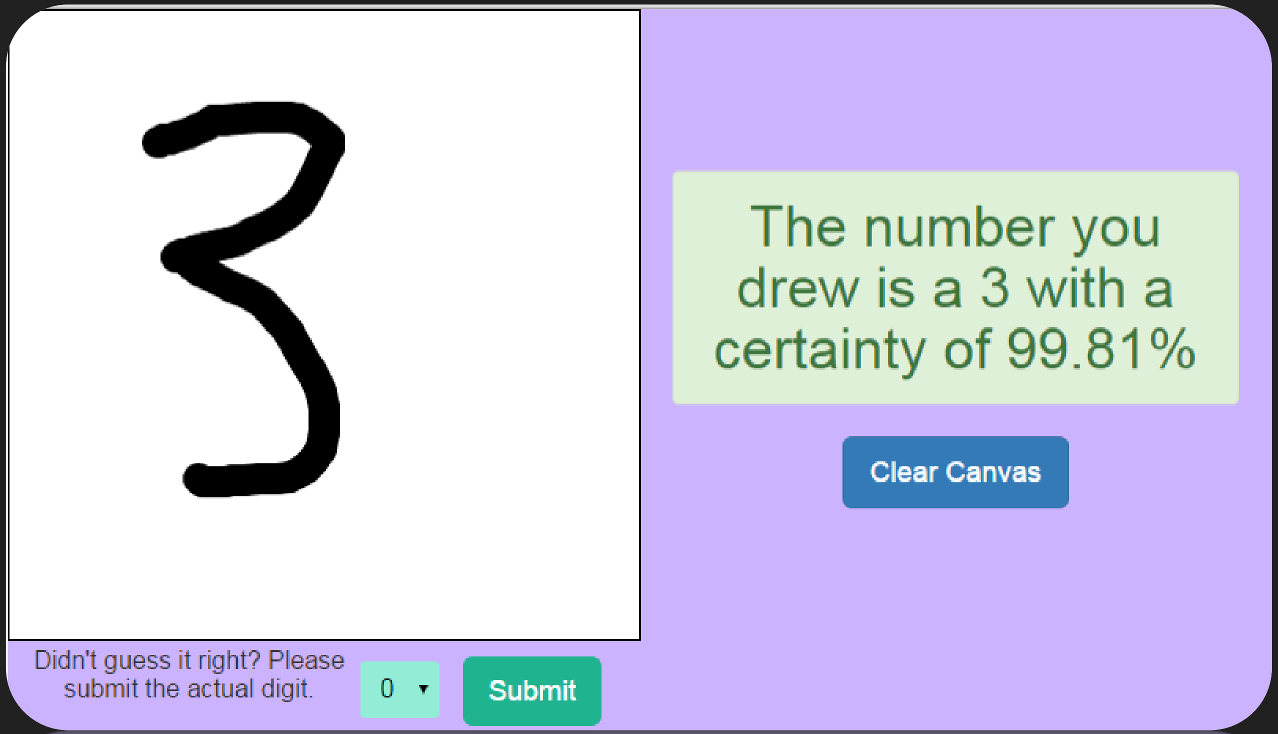
Digit Recognition

- Use handwritten digits as training data
- Crop and compress image to 16x16 pixels
  - Input uses 1-of-N encoding (-1 for White, 1 for Black)
- Output is vector of length 10 (for each digit)



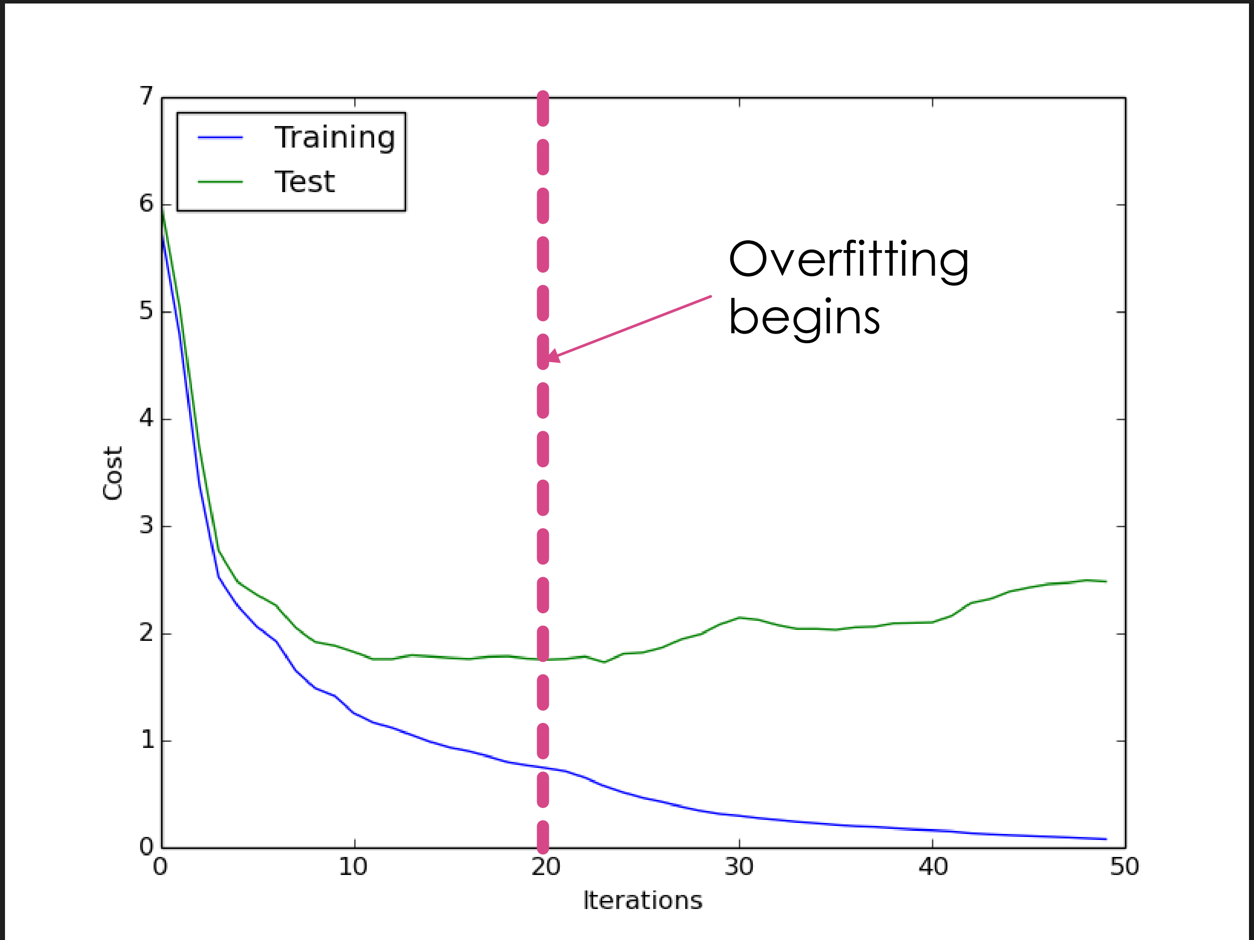
Data Collection

- Built application using PyGame
  - Mostly trained my handwriting
- Built cloud-based web application using Django
  - Crowd-sourced training
- Integrated with neural network for forward and backwards propagation



Testing

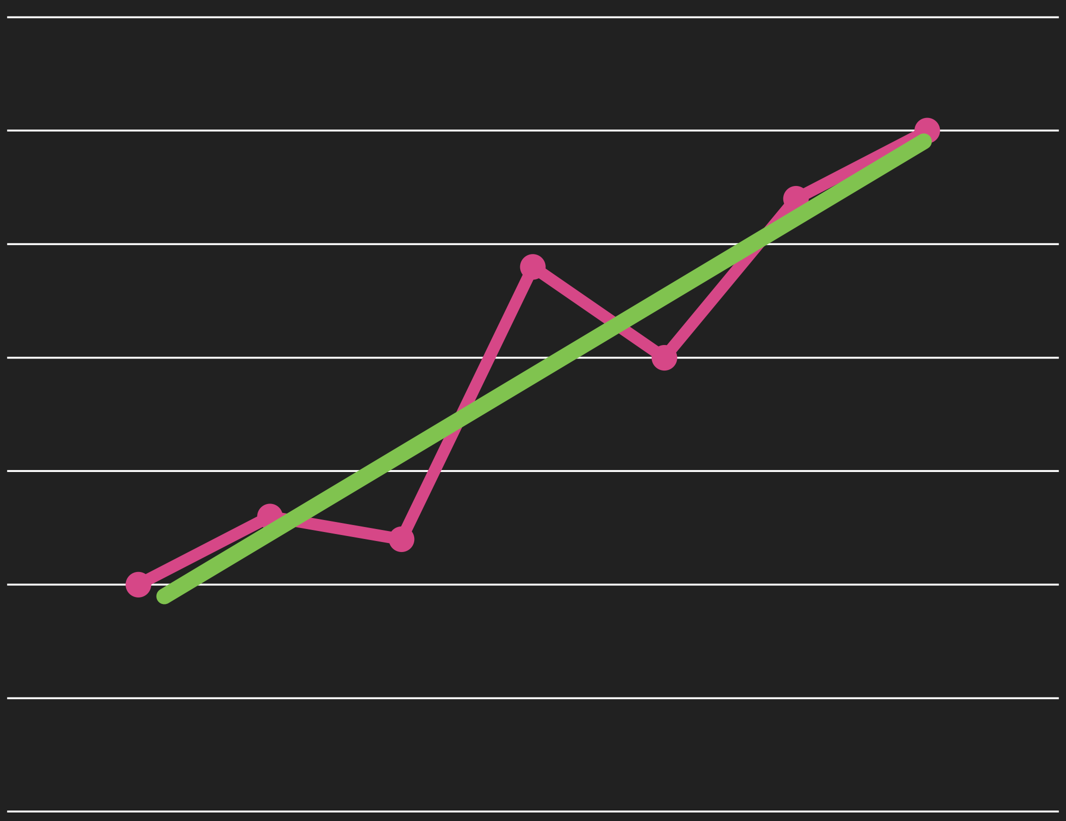
- 20% of data used for testing / 80% used for training
- Shows some signs of overfitting
- Overall accuracy of **82.4%** across 1560 trials



Accounting for Overfitting

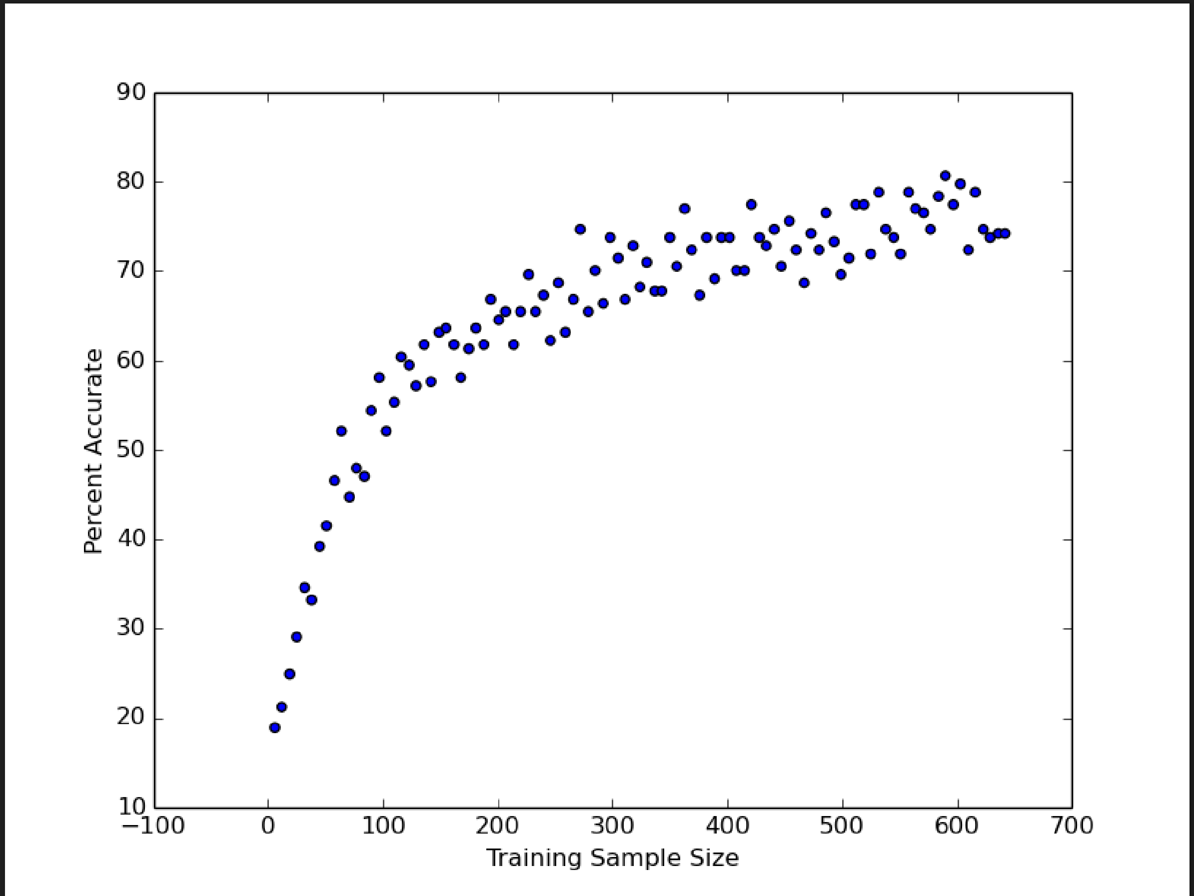
- Early stoppage
  - Stopped iterations at around 40, computed average for beginning of overfitting
- Need more data (sample size too small compared to number of inputs/outputs)
- Regularization constant

$$\frac{\lambda}{2n}\sum {w_j}^2$$



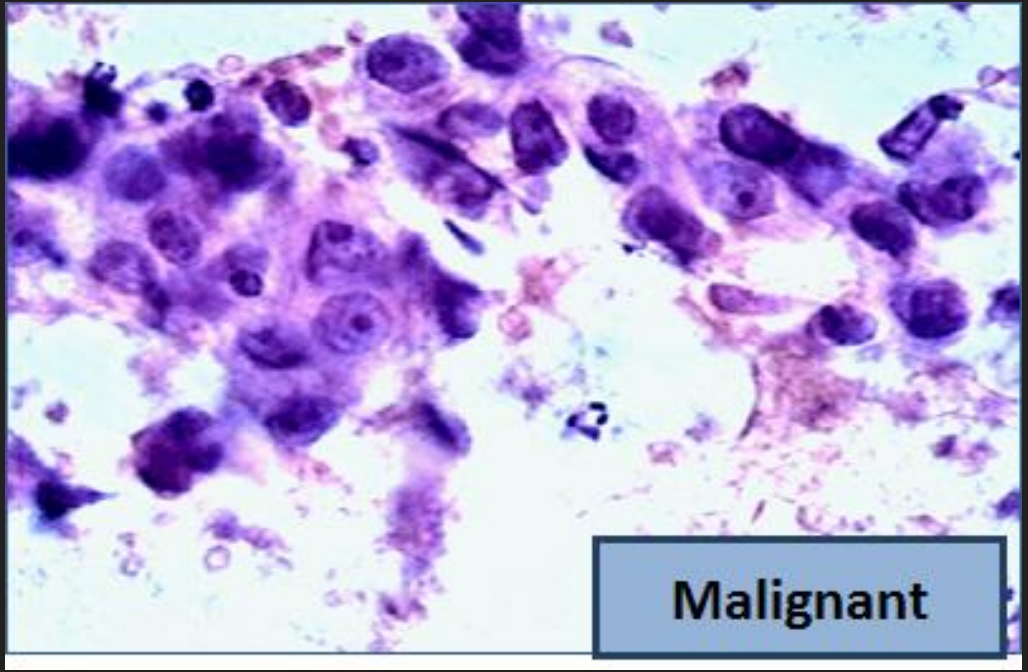
Sample Size versus Percent Accuracy

- % Accuracy vs Training Sample Size
- Shows logarithmic growth
- Shows that an increase of sample size will increase accuracy (overcome overfitting)



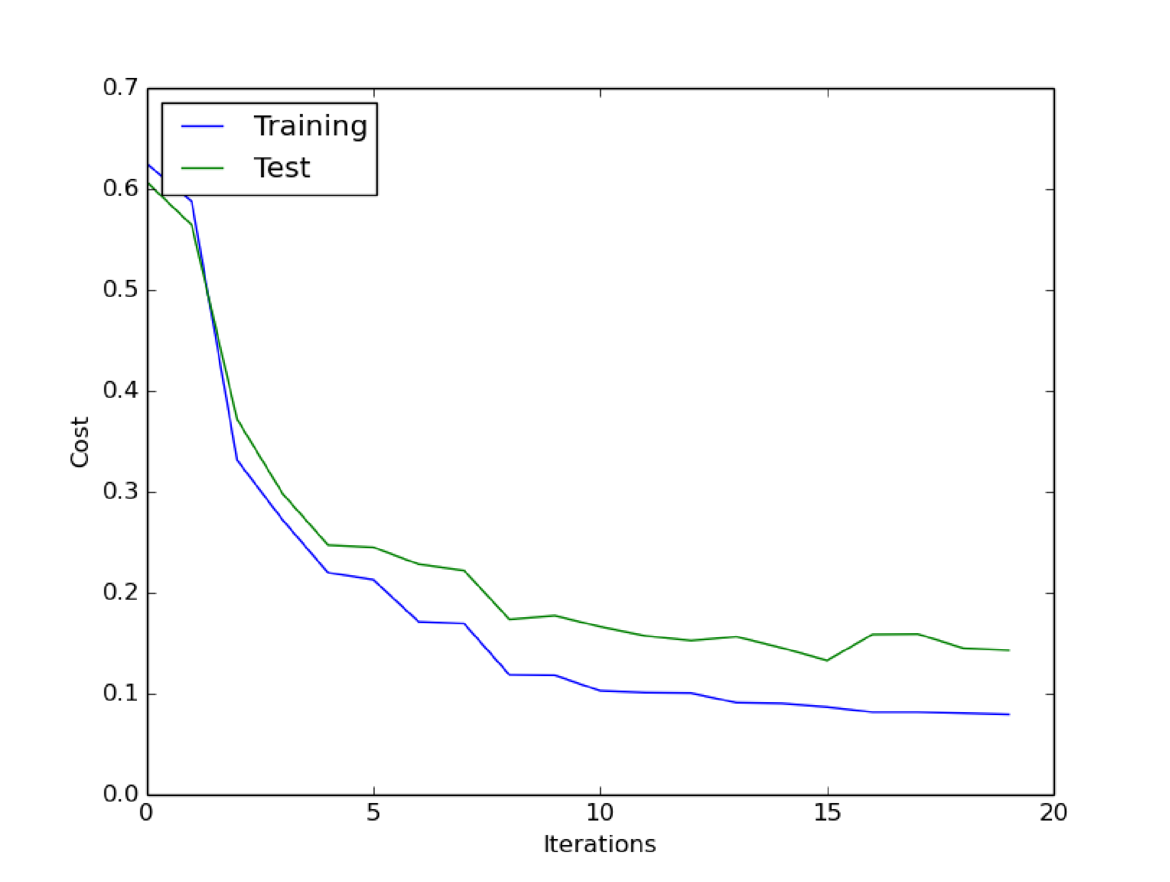
Diagnosis of Fine Needle Aspirates of Breast Tumors

- Input data from Breast biopsies from UCI Machine Learning Database
- Data consists of many attributes of biopsies (radius, texture, perimeter, smoothness, etc)
- Single Output (Benign vs Malignant)



Testing

- 30% of data used for testing / 70% used for training
- Ample amount of data (no signs of overfitting)
- Overall accuracy of **97%** across 715 trials



Conclusion

Successes

- High % accuracy for both sets
- Efficient gradient descent
- Neural network is extensible

Future Implications

- More training data (esp. for ODR)
- Bootstrap aggregation of data set (esp. for ODR)
- Investigation into more convoluted/deep neural networks

Bibliography

- Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.
- Cernazanu-Glavan, Cosmin, and Stefan Holban. *Segmentation of bone structure in X-ray images using convolutional neural network*. Timosoara: Politehnica University of Timisoara, n.d. Digital file.
- Hagan, Martin T. *Neural Network Design*. N.p.: Oklahoma State, n.d. Print.
- Jenker, Andrej. "Introduction to Artificial Neural Networks." *Artificial Neural Networks - Methodological Advances and Biomedical Applications*. Ed. Kenji Suzuki. N.p.: InTech, 2011. N. pag. Digital file.
- Rojas, Raul. *Neural Networks: A Systematic Introduction*. Berlin: Springer, 1996. Digital file.
- Zhang, Q. J. *Neural Network Structures for RF and Microwave Applications*. Orlando: IEEE AP-S Antennas and Propagation Int. Symp., 1999. Digital file.

Supervised Image Classification Using an Artificial Neural Network for Optical Digit Recognition and Diagnosis of Fine Needle Aspirates of Breast Cancer

Somil Govani