Data Book and Version Change Log

<https://github.com/somilgo/image_rec_neural_net>

2016-11-16 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Preliminary Research**

* Preliminary research conducted
  + Artificial Intelligence: A Modern Approach
  + Le Cun
  + Jiang, J.
* Potential applications of neural network
  + Stock market analysis
  + X-ray analysis
  + Optical Character Recognition
  + NP Problems

2016-11-20 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 1.0**

* Built Neural\_Network class constructor
* Built forward propagation method
  + Chose/Built sigmoid method (chose logistic)
  + Built sigmoid prime method

**Figure 1: Neural Network Class Constructor**

**import** numpy **as** np

**from** scipy **import** optimize

**import** matplotlib**.**pyplot **as** plt

**import** copy

iteration **=** 0

**class** **Neural\_Network(**object**):**

#Intialize neural network object (requires length of square image)

**def** \_\_init\_\_ **(**self**,** iLayer**=**2**,** oLayer**=**10**,** hLayer**=**10**,** Lambda**=**0**):**

self**.**inputLayerSize**=** iLayer

self**.**outputLayerSize **=** oLayer

#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

#Sigmoid Activation function

**def** sigmoid**(**self**,** z**):**

**return** 1.0 **/** **(**1.0 **+** np**.**exp**(-**z**))**

#Differentiated Sigmoid activation function

**def** sigmoidPrime**(**self**,** z**):**

**return** **(**np**.**exp**(-**z**)** **/** **((**1.0**+**np**.**exp**(-**z**))\*\***2**))**

**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

2016-12-01 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 1.1**

* Researched various cost functions
  + Variance (Squared Sums)
  + Softmax
  + Cross Entropy (Ding Ding)
* Found partial derivative of cross entropy cost in respect to and

2016-12-10 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 1.2**

* Implemented cost function
* Implemented costPrime function (derivative)

**Figure 2: Cost and Cost Gradient Functions**

**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

2016-12-11 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 1.2.1**

* Checked gradient using manual approximation of derivative
* Results matched to at least 5 decimal places for all attempted derivatives
* Approximations perturbed by 1e-4

**Figure 3: Gradient Checking Functions**

**def** getParams**(**self**):**

#Get W1 and W2 unrolled into vector:

params **=** np**.**concatenate**((**self**.**W1**.**ravel**(),** self**.**W2**.**ravel**()))**

**return** params

**def** setParams**(**self**,** params**):**

#Set W1 and W2 using single paramater vector.

W1\_start **=** 0

W1\_end **=** self**.**hiddenLayerSize **\*** self**.**inputLayerSize

self**.**W1 **=** np**.**reshape**(**params**[**W1\_start**:**W1\_end**],** **(**self**.**inputLayerSize **,** self**.**hiddenLayerSize**))**

W2\_end **=** W1\_end **+** self**.**hiddenLayerSize**\***self**.**outputLayerSize

self**.**W2 **=** np**.**reshape**(**params**[**W1\_end**:**W2\_end**],** **(**self**.**hiddenLayerSize**,** self**.**outputLayerSize**))**

**def** computeGradients**(**self**,** X**,** y**):**

dJdW1**,** dJdW2 **=** self**.**costPrime**(**X**,** y**)**

**return** np**.**concatenate**((**dJdW1**.**ravel**(),** dJdW2**.**ravel**()))**

**def** computeNumericalGradient**(**N**,** X**,** y**):**

paramsInitial **=** N**.**getParams**()**

numgrad **=** np**.**zeros**(**paramsInitial**.**shape**)**

perturb **=** np**.**zeros**(**paramsInitial**.**shape**)**

e **=** 1e-4

**for** p **in** range**(**len**(**paramsInitial**)):**

#Set perturbation vector

perturb**[**p**]** **=** e

N**.**setParams**(**paramsInitial **+** perturb**)**

loss2 **=** N**.**cost**(**X**,** y**)**

N**.**setParams**(**paramsInitial **-** perturb**)**

loss1 **=** N**.**cost**(**X**,** y**)**

#Compute Numerical Gradient

numgrad**[**p**]** **=** **(**loss2 **-** loss1**)** **/** **(**2**\***e**)**

#Return the value we changed to zero:

perturb**[**p**]** **=** 0

#Return Params to original value:

N**.**setParams**(**paramsInitial**)**

**return** numgrad

2016-12-20 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 1.3**

* Implemented Trainer class (used Conjugate Gradient algorithm from scipy)
* Tested full Neural\_Network with random test data
  + No errors

**Figure 4: Trainer Class**

**class** **Trainer(**object**):**

**def** \_\_init\_\_**(**self**,** N**):**

#Make Local reference to network:

self**.**N **=** N

**def** callbackF**(**self**,** params**):**

#Set parameters and add cost data to lists for graphing

self**.**N**.**setParams**(**params**)**

self**.**J**.**append**(**self**.**N**.**cost**(**self**.**X**,** self**.**y**,** test**=True))**

self**.**testJ**.**append**(**self**.**N**.**cost**(**self**.**testX**,** self**.**testY**,** test**=True))**

**def** costFunctionWrapper**(**self**,** params**,** X**,** y**):**

#Computes cost function and dJ/dW

self**.**N**.**setParams**(**params**)**

cost **=** self**.**N**.**cost**(**X**,** y**)**

grad **=** self**.**N**.**computeGradients**(**X**,**y**)**

**return** cost**,** grad

**def** train**(**self**,** trainX**,** trainY**,** testX**,** testY**):**

#Make an internal variable for the callback function:

self**.**X **=** trainX

self**.**y **=** trainY

self**.**testX **=** testX

self**.**testY **=** testY

#Make empty list to store training costs:

self**.**J **=** **[]**

self**.**testJ **=** **[]**

params0 **=** self**.**N**.**getParams**()**

options **=** **{**'maxiter'**:** 100**,** 'disp' **:** **True}**

#Minimize cost function using computed gradient method

\_res **=** optimize**.**minimize**(**self**.**costFunctionWrapper**,** params0**,** jac**=True,** method**=**'CG'**,**

args**=(**trainX**,** trainY**),** options**=**options**,** callback**=**self**.**callbackF**)**

self**.**N**.**setParams**(**\_res**.**x**)**

self**.**optimizationResults **=** \_res

#Plot Cost vs Iterations graphs for testing and training data

plt**.**plot**(**self**.**J**)**

plt**.**plot**(**self**.**testJ**)**

plt**.**ylabel**(**'Cost'**)**

plt**.**xlabel**(**'Iterations'**)**

plt**.**legend**([**'Training'**,** 'Test'**],** loc**=**'upper left'**)**

plt**.**show**()**

#Returns average error for testing data

**return** **(**sum**(**self**.**testJ**))/(**len**(**self**.**testJ**))**

2016-01-02 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 2.0**

* Decided to test NN with Optical Handwritten Digit Recognition (ODR)
* Began building Pygame application to collect ODR data

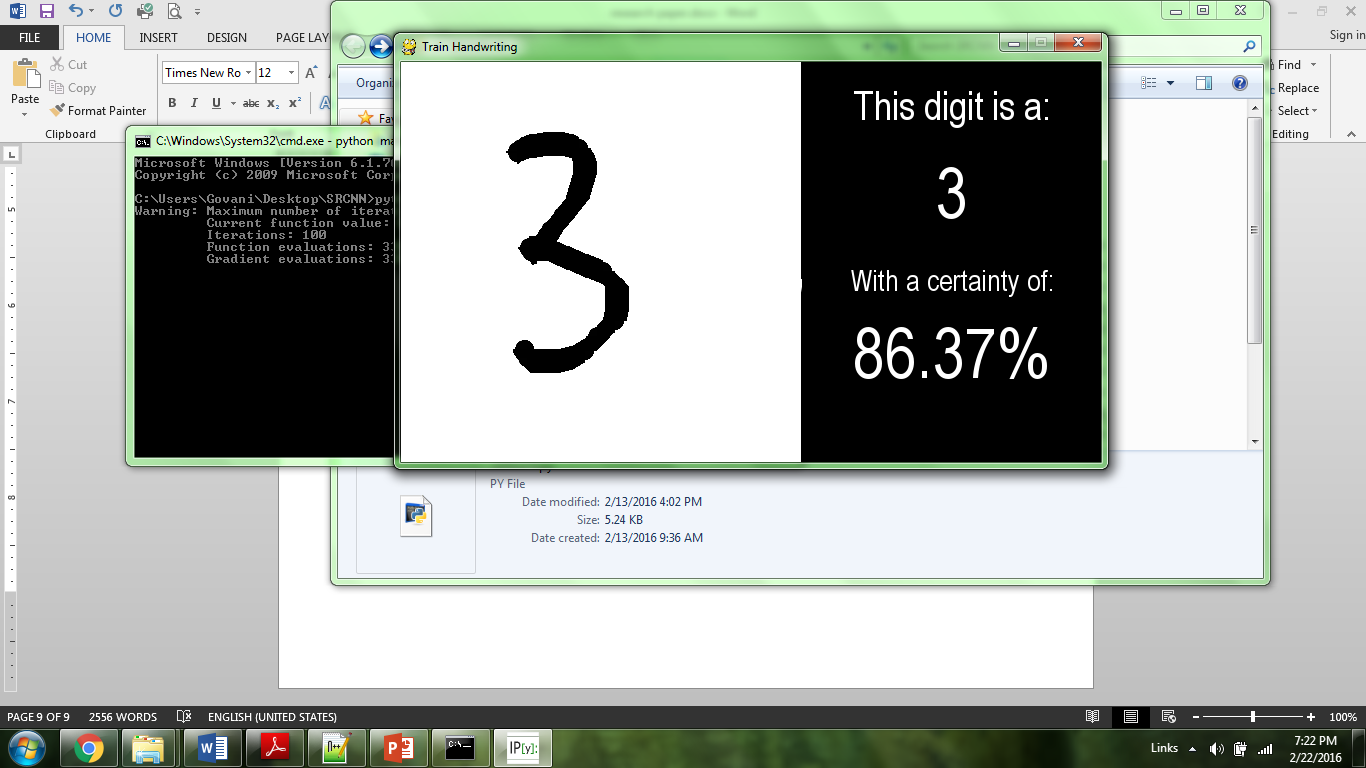
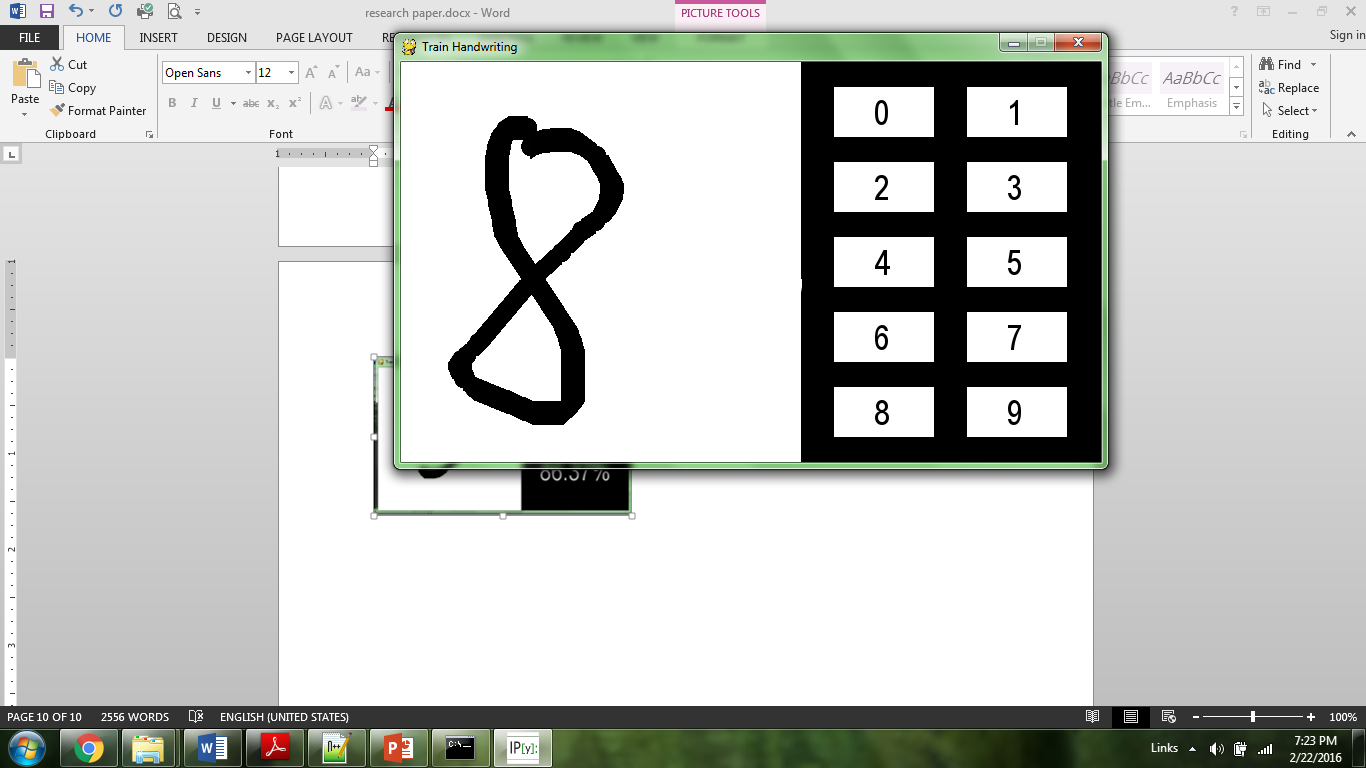
**Figure 5: PyGame Application Code**

2016-01-08 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 2.1**

* Fixed bugs in PyGame application
* Fixed method of calculating certainty
* Added filesystem to save images and pixel map matrices
* Added save button
* Integrated application with Neural Network class

**Figure 6: Pygame Application Screenshot**

****

2016-01-12 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 2.2**

* Collected supervised training data (403)
* Pixel maps stored as 16x16 (256-tuples)

2016-01-13 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 2.2.1**

* Collected supervised training data (398)
* Built data parser
* Tested neural network 🡪 showed considerable success in guessing most digits!

**Figure 7: Digit Parser**

#Collects and parses digit image data from images folder

**def** parseData**():**

x **=** **[]**

y **=** **[]**

dataDir **=** **False**

dataDirs **=** **[]**

**for** i **in** os**.**walk**(**os**.**getcwd**()+**"/images/"**):**

**if** dataDir**:**

dataDirs**.**append**(**i**[**0**])**

dataDir **=** **True**

**for** i **in** dataDirs**:**

**for** root**,** dirs**,** files **in** os**.**walk**(**i**):**

**for** f **in** files**:**

**if** f**.**endswith**(**".txt"**):**

cf **=** open**(**i**+**'/'**+**f**,** 'r'**)**

data **=** map**(**int**,** **(**cf**.**read**().**replace**(**'['**,** ''**).**replace**(**','**,** ''**).**replace**(**'-1'**,** '0'**).**replace**(**']'**,** ''**).**split**(**' '**)))**

x**.**append**(**data**)**

result **=** int**(**i**[**len**(**i**)-**1**])**

resultList **=** **[**0**]\***10

resultList**[**result**]** **=** 1

y**.**append**(**resultList**)**

**return** np**.**array**((**x**),** dtype**=**float**),** np**.**array**((**y**),** dtype**=**float**),** x**,** y

2016-01-15 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 3.0**

* Collected supervised breast cancer data from UCI Machine Learning Database
* <https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)>
* Built data parser

**Figure 8: Data Parser (Cancer Data)**

**import** os

**import** numpy **as** np

**from** random **import** shuffle

dataFile **=** open**(**os**.**getcwd**()+**"/wdbc.data"**)**

x **=** **[]**

y **=** **[]**

**for** line **in** dataFile**:**

datum **=** line**.**split**(**","**)**

datum**[**len**(**datum**)-**1**]=**datum**[**len**(**datum**)-**1**].**replace**(**"\n"**,** ""**)**

datum**.**pop**(**0**)**

**if** datum**.**pop**(**0**)** **==** 'M'**:**

y**.**append**([**1**])**

**else:**

y**.**append**([**0**])**

x**.**append**(**datum**)**

**from** network **import** **\***

xdata **=** x

ydata **=** y

picker **=** range**(**len**(**xdata**))**

#Randomizes order of data list indexes

shuffle**(**picker**)**

#Chooses 80% of the data as the training set

trainset **=** int**(**len**(**xdata**)\***.75**)**

pickersplit**=[**picker**[**x**:**x**+**trainset**]** **for** x **in** xrange**(**0**,** len**(**picker**),** trainset**)]**

trainx **=** **[]**

testx **=** **[]**

trainy **=** **[]**

testy **=** **[]**

**for** p **in** pickersplit**[**0**]:**

trainx**.**append**(**xdata**[**p**])**

trainy**.**append**(**ydata**[**p**])**

**for** t **in** pickersplit**[**1**]:**

testx**.**append**(**xdata**[**t**])**

testy**.**append**(**ydata**[**t**])**

#Creates training sets and testing sets

x **=** testx

y **=** testy

trainx **=** np**.**array**(**trainx**,** dtype**=**float**)**

trainy **=** np**.**array**(**trainy**,** dtype**=**float**)**

testx **=** np**.**array**(**testx**,** dtype**=**float**)**

testy **=** np**.**array**(**testy**,** dtype**=**float**)**

trainx **=** trainx**/**np**.**amax**(**trainx**,** axis**=**0**)**

trainy **=** trainy**/**np**.**amax**(**trainy**,** axis**=**0**)**

testx **=** testx**/**np**.**amax**(**testx**,** axis**=**0**)**

testy **=** testy**/**np**.**amax**(**testy**,** axis**=**0**)**

nn **=** Neural\_Network**(**iLayer**=** 30**,** oLayer**=**1**,** hLayer**=**15**)**

t **=** Trainer**(**nn**)**

t**.**train**(**trainx**,** trainy**,** testx**,** testy**)**

numberCorrect **=** 0

total **=** 0

**for** i **in** range**(**len**(**testx**)):**

w **=** nn**.**forward**(**testx**[**i**])**

d **=** testy**[**i**]**

**if** **(**abs**(**w**[**0**]-**d**[**0**]))** **<** .5**:**

numberCorrect**+=**1

total**+=**1

**print** numberCorrect

**print** total

**print** float**(**numberCorrect**)/**total **\*** 100

2016-01-15 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 3.1**

* Debugged the parser
  + Fixed list out of index error
  + Kept track of percent accuracy
* Added percent accuracy feature to digitParser also
* Trained Neural\_Network with cancer data 🡪 showed considerable success!

2016-01-15 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 3.2**

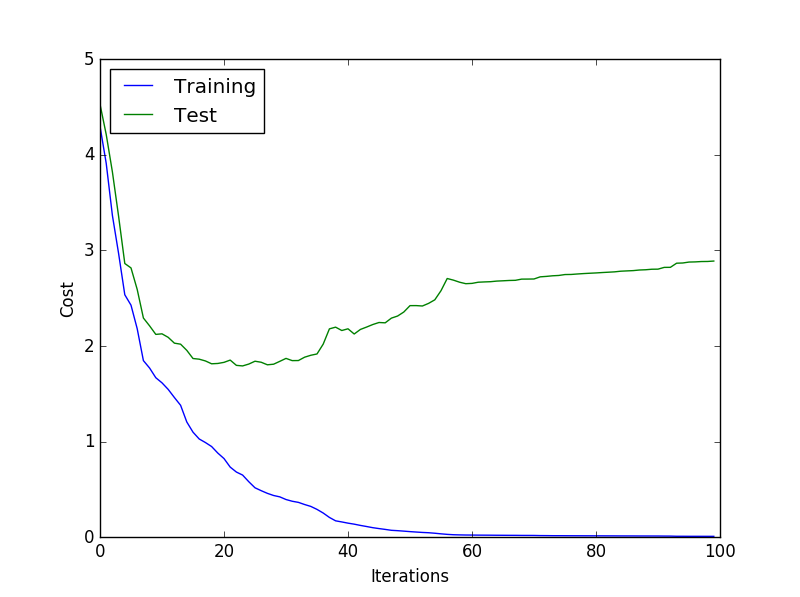
* Added matplotlib plotting to Neural\_Network, digitParser, and cancer parser in order to collect graphical data
* Debugged percent accuracy on all fronts
* Normalized all data using np.max
* Created 80/20 train/test splitter
  + Used randomization algorithm using permutations

2016-01-16 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

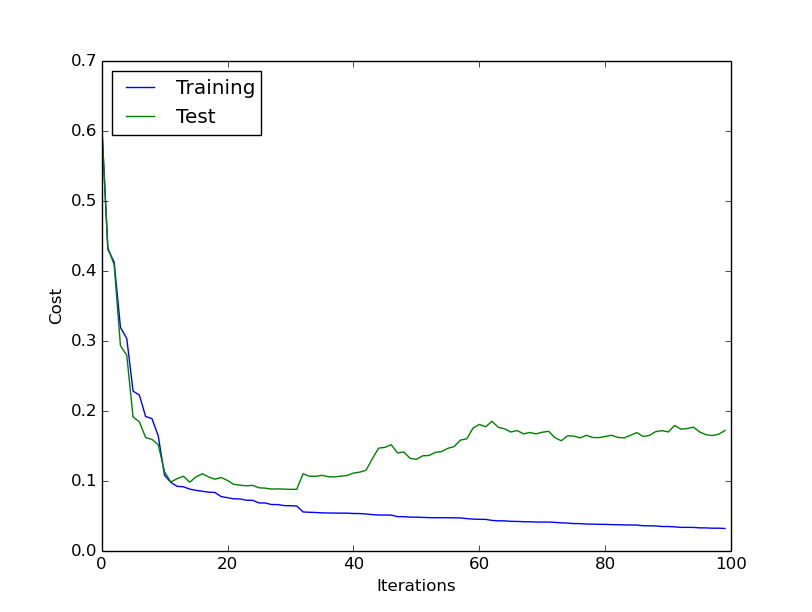
**Version 3.3**

* Began collecting data
* Collected percent accuracy for cancer and digit data
* Collected Cost vs Minimization iterations for both sets as well (100 iterations)

**Figure 9: Cost VS Minimization ODR**

****

**Figure 10: Cost VS Minimization CANCER**

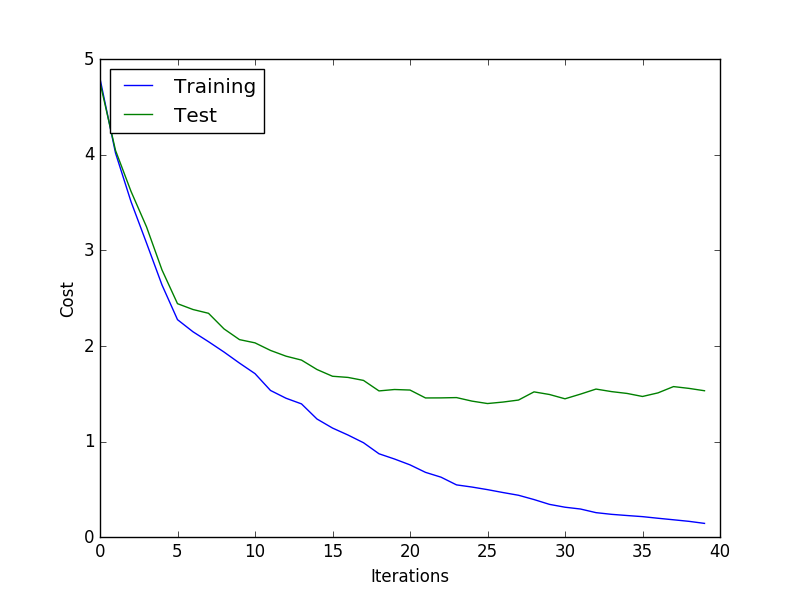
****

2016-01-20 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

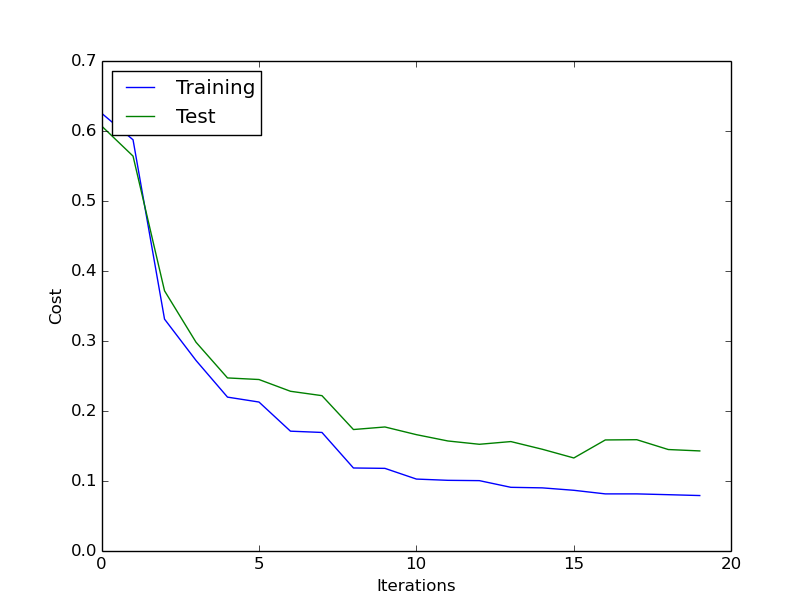
**Version 3.3.1**

* Implemented overcoming overfitting methods
  + Regularization constant
  + Early stoppage
  + Bootstrap aggregation 🡪 STILL NEED TO INTEGRATE
* Retested cost vs minimization iterations
* Retested percent accuracy
* Only ODR data still shows *slight* signs of overfitting
* CANCER data shows no signs of overfitting

**Figure 11: Cost VS Minimization ODR (overcoming overfitting)**

****

**Figure 12: Cost VS Minimization CANCER (overcoming overfitting)**

****

**Figure 13: Percent Accuracy (ODR)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Number Correct | Size of Test Sample | Percent Accurate |
| Trial 1 | 210 | 260 | 0.807692 |
| Trial 2 | 201 | 260 | 0.773077 |
| Trial 3 | 212 | 260 | 0.815385 |
| Trial 4 | 222 | 260 | 0.853846 |
| Trial 5 | 217 | 260 | 0.834615 |
| Trial 6 | 224 | 260 | 0.861538 |
| TOTAL | **1286** | **1560** | **0.824359** |

**Figure 14: Percent Accuracy (CANCER)**

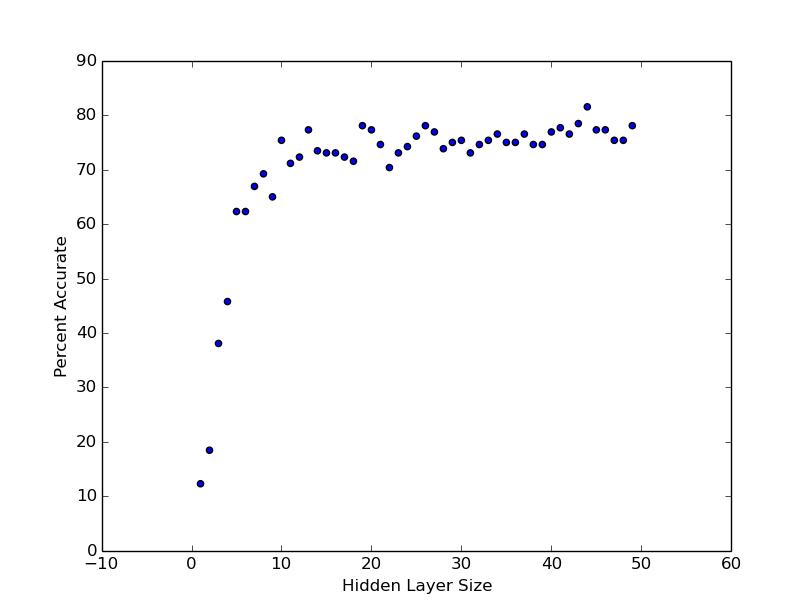
|  |  |  |  |
| --- | --- | --- | --- |
|  | Number Correct | Size of Test Sample | Percent Accurate |
| Trial 1 | 140 | 143 | 0.979020979 |
| Trial 2 | 138 | 143 | 0.965034965 |
| Trial 3 | 136 | 143 | 0.951048951 |
| Trial 4 | 138 | 143 | 0.965034965 |
| Trial 5 | 139 | 143 | 0.972027972 |
| TOTAL | **691** | **715** | **0.966433566** |

2016-02-02 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

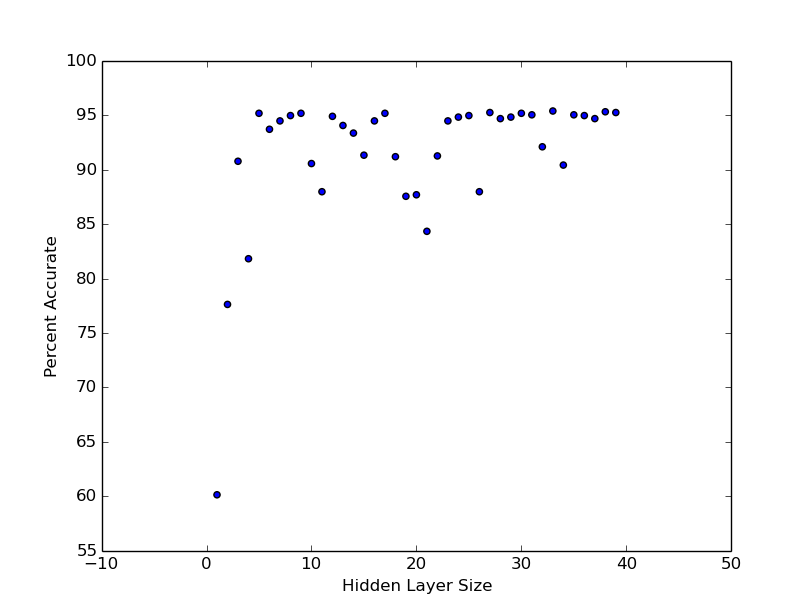
**Version 3.3.2**

* Gathered additional supporting data
  + Percent accuracy vs hidden layer size
  + Percent accuracy vs sample size (ODR only 🡪 cancer has no overfitting)

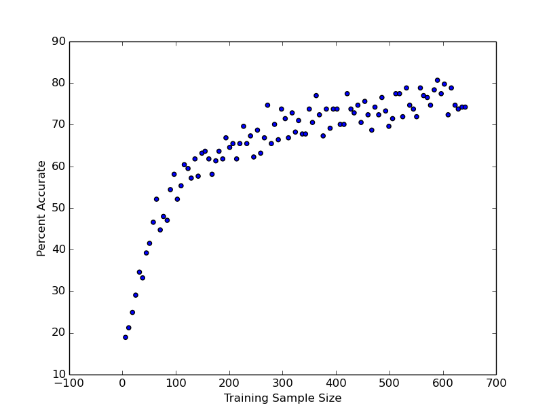
**Figure 15: Percent Accuracy vs Hidden Layer (ODR)**

****

**Figure 16: Percent Accuracy vs Hidden Layer (CANCER)**

****

**Figure 17: Percent Accuracy vs Sample Size (ODR)**

****

2016-02-05 Somil Govani [somilgovani@gmail.com](mailto:somilgovani@gmail.com)

**Version 4.0**

* Previous data shows an increase in sample size will reduce overfitting
* Built web application using Python Django in order to collect more training data
  + <http://neuralnetwork.herokuapp.com>
* Stores image data on online Postgres database

**Figure 18: Python Django Web Application**

