Capstone-Finnegan Documentation

Release 0.1

Cole Howard

January 06, 2016

Contents

1 Finnegan - An Exploration in Neural Nets **3**
- 1.1 Overview . . . 3
- 1.2 Requirements . . . 3
- 1.3 Usage . . . 4
- 1.4 Acknowledgements . . . 4

2 Network Class for Neural Network **5**

3 Layer Class for Neural Network **9**

4 Mini Net Helper Module **11**

5 Django View for Finnegan Web App **13**

6 Helper Method for Parsing Web App Input **15**

7 Indices and tables **17**

Python Module Index **19**

Contents:

CHAPTER 1

Finnegan - An Exploration in Neural Nets

1.1 Overview

Finnegan is a basic multi-class neural net classifier. It is designed to be trained with any labled dataset, and then make predictions (classifications) on unseen, similar examples.

A front end user experience, in the form of a web app is provided around the task of classifying handwritten digits (0-9). To try: navigate here in a browser.

The webapp uses django and boostrap to serve a javascript page to collect the handwriting sample. On the backend the server parses the drawing to a greyscale image and then uses the pixel values as the input vector to the neural net.

The neural net in the webapp is pre-trained (as real time training would be impractical from the perspective of a user's time).

The pre-training was done on the MNIST dataset, a set of 60,000 handwritten digit examples. Each a 28px by 28px greyscale image. The architecture of the net was chosen by running the helper function "ext_mini_net.py" to iterate over many possible combinations of layer sizes (number of neurons) and number of layers. The most successful of those runs against a validation set were chosen and the state of that trained network was "pickled" using Python Pickle.

The webapp reinstantiates that trained network and runs the user's sample resized (see this post on my blog for deatails on that). It then makes a guess as to which digit it is. It also provides a confidence and requests feedback from the user on the validity of the guess. The data surrounding each submission is stored in a PostgreSQL database: image, resized image, guess, confidence, correct/incorrect, actual digit (if guess was incorrect).

Lastly, a statistic page is provided for the overall performance of the classifier.

1.2 Requirements

Basic Neural Network Needs

- Scipy
- Numpy
- Scikit-learn (for randomize dataset, or using their handwritten, smaller digits)

Webapp Needs (in addition to above)

- Django
- Bootstrap
- PosgreSQL

1.3 Usage

To run the webapp locally, clone the repo.

You will need to setup a PostgreSQL database and point the app to it in: capstone/net/settings.py

Then from the capstone directory:

```
python manage.py makemigrations
python manage.py migrate
python manage.py runserver
```

This will serve the page on localhost:8000. Navigate your browser there and enjoy.

To use the Finnegan network separately, from capstone/finnegan:

```
from network import Network
```

Create an instance from the class:

```
net = Network(layers, neuron_count, vector)
```

Where: - layers is number of layers including the output layer - neuron_count is a list of integers representing the number of neurons per layer, with the output layer being the last in the list (and therefor should be the total number of possible classifications). - vector is a sample input vector (for determining size and shape)

Then:

```
net.train(dataset, answers, epcohs)
guesses = net.run_unseen(test_dataset)
net.report_results(guesses, test_answers)
```

Will train the net, and provide a report on the success of guessing against the test set.

1.4 Acknowledgements

sketch.js is a javascript/jquery tool for capturing user input on an html canvas. Borrowed and modified from here.

The blog I Am Trask provided me with the key insight to vectorize each layer as a single unit and thereby get rid of the for loops as that took FOREVER to run.

CHAPTER 2

Network Class for Neural Network

Author: Cole Howard

An extinsible neural net designed to explore Neural Network Architecture via extensive visualizations.

class finnegan.network.**Network**(*layers*, *neuron_count*, *vector*)

A multi layer neural net with backpropogation.

Parameters

- **layers** (*int*) – Number of layers to use in the network.
- **neuron_count** (*list*) – A list of integers that represent the number of neurons present in each hidden layer. (Size of input/output layers are dictated by dataset)
- **vector** (*list*) – Example vector to get size of initial input

possible

list

A list of possible output values

_backprop(*guess_vector*, *target_vector*)

Takes the output of the net and initiates the backpropogation

In output layer: generate error matrix [(out * (1-out) * (Target-out)) for each neuron] update weights matrix [[+= l_rate * error_entry * input TO that amount] for each neuron]

In hidden layer generate error matrix [out * (1-out) * dotproduct(entry in n+1 error matrix, n+1 weight of that entry)] update weights matrix [[+= l_rate for each weight] for each neuron]

Parameters

- **guess_vector** (*numpy array*) – The output from the last layer during a training pass
- **target_vector** (*list*) – List of expected values

Returns As evidence of execution

Return type True

_pass_through_net(*vector*, *dropout=True*)

Sends a vector into the net

Parameters

- **vector** (*numpy array*) – A numpy array representing a training input (without the target)

- **dropout** (*bool*) – Whether or not you should perform random dropout in the pass through the net. (Set False for the tesing set vectors)

Returns Output of the last layer in the chain

Return type numpy array

_softmax(*w*, *t=1.0*)

Author: Jeremy M. Stober, edits by Martin Thoma Program: softmax.py Date: Wednesday, February 29 2012 and July 31 2014 Description: Simple softmax function. Calculate the softmax of a list of numbers w.

Parameters

- **w** (*list of numbers*) –
- **t** (*float*) –

Returns

Return type a list of the same length as w of non-negative numbers

Examples

```
>>> softmax([0.1, 0.2])
array([ 0.47502081,  0.52497919])
>>> softmax([-0.1, 0.2])
array([ 0.42555748,  0.57444252])
>>> softmax([0.9, -10])
array([  9.99981542e-01,   1.84578933e-05])
>>> softmax([0, 10])
array([  4.53978687e-05,   9.99954602e-01])
```

report_results(*guess_list*, *answers*)

Reports results of guesses on unseen set

Parameters

- **guess_list** (*list*) –
- **answers** (*list*) –

run_unseen(*test_set*)

Makes guesses on the unseen data, and switches over the test answers to validation set if the bool is True

For each vector in the collection, each neuron in turn will either fire or not. If a vector fires, it is collected as a possible correct guess. Not firing is collected as well, in case there an no good guesses at all. The method will choose the vector with the highest dot product, from either the fired list or the dud list.

Parameters **test_set** (*list*) – List of numpy arrays representing the unseen vectors

Returns a list of ints (the guesses for each vector)

Return type list

train(*dataset*, *answers*, *epochs*)

Runs the training dataset through the network a given number of times.

Parameters

- **dataset** (*Numpy nested array*) – The collection of training data (vectors and the associated target value)

- **answers** (*numpy array*) – The array of correct answers to associate with each training vector
- **epochs** (*int*) – Number of times to run the training set through the net

CHAPTER 3

Layer Class for Neural Network

Author: Cole Howard

class finnegan.layer.**Layer**(*num_neurons*, *vector_size*)

A matrix representation of the neurons in a layer Inspired by: I Am Trask http://iamtrask.github.io/2015/07/12/basic-python-network/

Parameters

- **num_neurons** (*int*) – The number of instances of the class Neuron in each layer.
- **vector_size** (*int*) – The number of inputs from the previous layer/original input. Also, equivalent to the length of incoming input vector.

weights

numpy array

A matrix reprsentation of the weight space. Each column represents a neurnon in the layer. Each entry in those columns is the value of a weight in that neuron.

mr_output

numpy array

Output of the layer

mr_input

numpy array

Input vector from the layer below (or original input)

deltas

numpy array

Calculated change in the weightspace for the backprop

l_rate

float

The learning rate for the update weight method

reg_rate

float

The factor by which the weights are adjusted for regularization to prevent overfitting.

_act_derivative(*vector*)

Calculate the derivative of the activation function

Parameters **vector** (*numpy array*) – A vector representing the most recent output of a given layer

Returns

Return type numpy array

_layer_level_backprop(*output*, *layer_ahead*, *target_vector*, *hidden=True*)
Calculates the error at this level

Parameters

- **layer_ahead** (*object*) – The instance of Layer that this layer's output is connected to
- **target_vector** (*numpy array*) – A representation of the expected output of the net for the original vector input on this particular pass
- **hidden** (*bool*) – Whether or not the current layer is hidden (default: True)

Returns For acknoledgment of execution

Return type True

_update_weights()
Update the weights of each neuron based on the backprop calculation

_vector_pass(*vector*, *do_dropout=True*)
Takes the vector through the neurons of the layer

Parameters

- **vector** (*numpy array*) – The input array to the layer
- **do_dropout** (*bool*) – Whether or not weight dropout should happen as the vector passes through the layer

Returns The ouput of the layer

Return type numpy array

CHAPTER 4

Mini Net Helper Module

Author: Cole Howard

Helper module to create or reinstantiate a neural network. Commented out code is for training the network via Scikit dataset or original MNIST dataset. The resulting network is then pickled and saved to a file. The current uncommented code is for reading that file and reinstantiating it for testing new input against.

mini_net.**run_mnist**(*vector*, *epochs=0*, *layers=0*, *neuron_count=0*)

Builds network (or reinstantiates it) based on the MNIST Digits dataset.

Parameters

- **epochs** (*int*) – Number of iterations of the the traininng loop for the whole dataset
- **layers** (*int*) – Number of layers (not counting the input layer, but does count output layer)
- **neuron_count** (*list*) – The number of neurons in each of the layers (in order), does not count the bias term

mini_net.**target_values**

list

The possible values for each training vector

mini_net.**run_scikit_digits**(*vector*, *epochs=0*, *layers=0*, *neuron_count=[]*)

Builds network (or reinstantiates it) based on the Scikit Digits dataset.

Parameters

- **epochs** (*int*) – Number of iterations of the the traininng loop for the whole dataset
- **layers** (*int*) – Number of layers (not counting the input layer, but does count output layer)
- **neuron_count** (*list*) – The number of neurons in each of the layers (in order), does not count the bias term

mini_net.**target_values**

list

The possible values for each training vector

CHAPTER 5

Django View for Finnegan Web App

guess.views.**parse_data**(*request*, **args*, ***kwargs*)

As a request comes from the home page via POST, parse the canvas image, downsize it to match the dims of the training data, and then pass it to the pre-trained neural network for it make a prediction.

The results of the prediction (value and confidence), as well as the array representations of the images themselves are stored in the model, and hence the PostgresQL database.

guess.views.**show_data**(*request*)

Sends a render of image as drawn and the network's guess and confidence via the report template.

guess.views.**drawing_obj**

Drawing model instance

The last line from the database (specifically the most recent entry).

guess.views.**stats_work**(*request*)

Work out statistics for results

guess.views.**valid_info**(*request*, **args*, ***kwargs*)

Stores the users validation (or dis-validation) of the network's guess in the database along with what the user intended it to be, in the case of an incorrect guess.

CHAPTER 6

Helper Method for Parsing Web App Input

Accept an array representing a test sample, parse it and pass through a pre-trained neural net then store results in a pre-existing database

guess.predict.**parse_to_test_sample**(*info*)

Takes the info retrieved from the script.js implementation on main page and parses it. Data is run against existing neural net and then stores net's output and original data in PostgreSQL database.

guess.predict.**orig_size**

int

The length of a side of the square html input canvas.

guess.predict.**train_data_size**

int

The length of a side of the square 2d array representing the training data.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

f

finnegan.layer, 9
finnegan.network, 5

g

guess.predict, 15
guess.views, 13

m

mini_net, 11

Symbols

_act_derivative() (finnegan.layer.Layer method), 9
_backprop() (finnegan.network.Network method), 5
_layer_level_backprop() (finnegan.layer.Layer method), 10
_pass_through_net() (finnegan.network.Network method), 5
_softmax() (finnegan.network.Network method), 6
_update_weights() (finnegan.layer.Layer method), 10
_vector_pass() (finnegan.layer.Layer method), 10

D

deltas (finnegan.layer.Layer attribute), 9
drawing_obj (in module guess.views), 13

F

finnegan.layer (module), 9
finnegan.network (module), 5

G

guess.predict (module), 15
guess.views (module), 13

L

l_rate (finnegan.layer.Layer attribute), 9
Layer (class in finnegan.layer), 9

M

mini_net (module), 11
mr_input (finnegan.layer.Layer attribute), 9
mr_output (finnegan.layer.Layer attribute), 9

N

Network (class in finnegan.network), 5

O

orig_size (in module guess.predict), 15

P

parse_data() (in module guess.views), 13
parse_to_test_sample() (in module guess.predict), 15
possible (finnegan.network.Network attribute), 5

R

reg_rate (finnegan.layer.Layer attribute), 9
report_results() (finnegan.network.Network method), 6
run_mnist() (in module mini_net), 11
run_scikit_digits() (in module mini_net), 11
run_unseen() (finnegan.network.Network method), 6

S

show_data() (in module guess.views), 13
stats_work() (in module guess.views), 13

T

target_values (in module mini_net), 11
train() (finnegan.network.Network method), 6
train_data_size (in module guess.predict), 15

V

valid_info() (in module guess.views), 13

W

weights (finnegan.layer.Layer attribute), 9