Jeremy Swartwood - 30512632

Machine Learning – Assignment 2 (Aurora images)

# Table of Contents

[Table of Contents 1](#_Toc506154589)

[1. Analysis of Perceptron learning: 2](#_Toc506154590)

[Learning rate, batch size, and iteration analysis: 2](#_Toc506154591)

[Batch Size 2](#_Toc506154592)

[Learning Rate 3](#_Toc506154593)

[Iterations: 4](#_Toc506154594)

[2. Implementation details: 5](#_Toc506154595)

[2.1 Part 1 (Separating the images) 5](#_Toc506154596)

[2.1.1 Aurora edge case image examples: 5](#_Toc506154597)

[2.2 Part 2 (Pre-process images into CSV) 5](#_Toc506154598)

[2.3 Part 3 (Perceptron) 5](#_Toc506154599)

[2.3.1 Output files: 5](#_Toc506154600)

[2.3.2 Loading from CSV: 6](#_Toc506154601)

[2.3.3 Perceptron: 6](#_Toc506154602)

[2.3.4 Weights update: 6](#_Toc506154603)

[2.3.5 Batching: 6](#_Toc506154604)

[3. How to use the scripts 7](#_Toc506154605)

[3.1 ColorHistogram.py 7](#_Toc506154606)

[Usage: 7](#_Toc506154607)

[Outputs: 7](#_Toc506154608)

[3.2 preceptron.py 7](#_Toc506154609)

[Usage: 7](#_Toc506154610)

[Outputs: 7](#_Toc506154611)

[3.3 runall.cmd 7](#_Toc506154612)

[Usage: 7](#_Toc506154613)

# 1. Analysis of Perceptron learning:

Multiple runs were captured, to see details on all learning rates, and a few different batch and iteration sizes, please see the other zip titled “RUNS.zip”. This includes the weight values as including 768 values for each image would make this file unreasonable to read and review.

## Learning rate, batch size, and iteration analysis:

Certain things played a larger role than others in the training. Analysis of those items are below.

### Batch Size

When this was lower than 100, the training and validation would take an extremely long time to learn, 5000+ iterations (batches). Additionally, each batch would have drastically high variances in the error %, depending on the random data points it chose. As batch sizes increase, the Training error % fluctuation goes down. However, the Validation error, while reduced fluctuations, still swings widely.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Learn | Iterations | Batch Size | Learn | Iterations | Batch Size |
| 0.1 | 10,000 | 20 | 0.1 | 1,000 | 5000 |
| C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\0.1_learn_10000_iter_20_batch_batch_err_pct.png  C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\0.1_learn_10000_iter_20_batch_weights_plot.png | | | C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\0.1_learn_1000_iter_5000_batch_batch_err_pct.png  C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\0.1_learn_1000_iter_5000_batch_weights_plot.png | | |
| 0.1\_learn\_10000\_iter\_20\_batch\_batch\_err\_pct.png  0.1\_learn\_10000\_iter\_20\_batch\_weights\_plot.png | | | 0.1\_learn\_1000\_iter\_5000\_batch\_batch\_err\_pct.png  0.1\_learn\_1000\_iter\_5000\_batch\_weights\_plot.png | | |

### Learning Rate

If this was 1.5 the perceptron seemed to learn fast, but had a high validation error % fluctuation.

If this was 0.001 the learning was slower, with less validation error % fluctuation, however it got stuck in a local minimum around 17% error (83% successful classification).

As the images below show, a lower learning rate drastically helps, however, some sort of “bumping” or adjustment needs to be done so that the training doesn’t get stuck in a local minimum. Perhaps starting at 1.5 and reducing it every iteration so that it learns less with each iteration. Or, when it seems to get “stuck” bumping the learn back up randomly to force it out of the local minimum.

It’s pretty obvious in the weight graphs for 0.001 learning that the weights barely moved, they stayed very close to their -1, 0, or 1 randomized starting locations. I would suspect that this value should only be used to “fine tune” near the end of learning.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Learn | Iterations | Batch Size | Learn | Iterations | Batch Size |
| 0.001 | 1,000 | 1000 | 1.5 | 1,000 | 1000 |
| C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\0.001_learn_1000_iter_1000_batch_batch_err_pct.png  C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\0.001_learn_1000_iter_1000_batch_weights_plot.png | | | C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\1.5_learn_1000_iter_1000_batch_batch_err_pct.png  C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\1.5_learn_1000_iter_1000_batch_weights_plot.png | | |
| 0.001\_learn\_1000\_iter\_1000\_batch\_batch\_err\_pct.png 0.001\_learn\_1000\_iter\_1000\_batch\_weights\_plot.png | | | 1.5\_learn\_1000\_iter\_1000\_batch\_batch\_err\_pct.png  1.5\_learn\_1000\_iter\_1000\_batch\_weights\_plot.png | | |

### Iterations:

These played a big role in the learning as well. Stop too soon and the error rate is still high, causing bad predictions. Stop too late and you’ve over trained, yet again causing bad predictions. I was looking at ways to stop the training when the Training Error % and the Validation Error % crossed, however, even with a smoothed out averaged over N batches line, it crossed so many times that it was hard to find a way to identify when to stop. Looking at the images afterwards, I can see in some where say 750 iterations would have been best for the values (learn/Batch size).

Notice in these two comparisons, running for 100 vs 1000 iterations of the same learn and batch size. The learning is just not yet complete in the 100 iteration run. There are lots of variances in the curves, leaning on to believe maybe it was done at 55 iterations, but looking at the 1000 iteration graph is it obvious that leaning really doesn’t stop until around 800-900.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Learn | Iterations | Batch Size | Learn | Iterations | Batch Size |
| 0.01 | 1,000 | 1000 | 0.01 | 100 | 1000 |
| C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\0.01_learn_1000_iter_1000_batch_batch_err_pct.png  C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\0.01_learn_1000_iter_1000_batch_weights_plot.png | | | C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\0.01_learn_100_iter_1000_batch_batch_err_pct.png  C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\runs\0.01_learn_100_iter_1000_batch_weights_plot.png | | |
| 0.01\_learn\_1000\_iter\_1000\_batch\_batch\_err\_pct.png 0.01\_learn\_1000\_iter\_1000\_batch\_weights\_plot.png | | | 0.01\_learn\_100\_iter\_1000\_batch\_batch\_err\_pct.png  0.01\_learn\_100\_iter\_1000\_batch\_weights\_plot.png | | |

# 2. Implementation details:

## 2.1 Part 1 (Separating the images)

Images were split into aurora, none, and unknown. During this process I was extremely detailed and intentionally put images with even a “hint” of aurora in them into the Aurora folder. For example, the following images are all in Aurora.

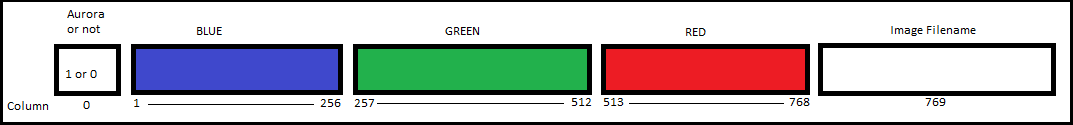
### 2.1.1 Aurora edge case image examples:

The classifier could have been a bit more accurate had the images that were “questionable” been put into none or unknown. However it was good to see how well the perceptron was able to train even with images as vastly different as these. I didn’t want to change my dataset to skew the numbers in my testing.

|  |  |  |
| --- | --- | --- |
| PF\_image2017-11-30 19\_41\_21.220397 | PF\_image2017-12-04 21\_20\_52.740852 | PF\_image2017-12-13 22\_57\_53.854595 |
| C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\images\aurora\PF_image2017-11-30 19_41_21.220397.jpg | C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\images\aurora\PF_image2017-12-04 21_20_52.740852.jpg | C:\Users\thisisme1\Documents\CSCE415 - ML\assign2\images\aurora\PF_image2017-12-13 23_41_36.685961.jpg |

## 2.2 Part 2 (Pre-process images into CSV)

During this part the python code goes through each image, gets the histogram (count of R, G, and B color portion in each of 0-255 buckets, creating 768 “bins” or features). A CSV file is created with a 1 for Aurora or 0 for none, followed by the data from the histogram, and finally the filename of the image. Each row is information about 1 image, meaning each row has 780 columns. The Unknown folder was not processed. The histogram is NOT normalized at this point, it is done after it is loaded into the perceptron.



OpenCV used BGR format, so the histogram data in the file is in that same format.

## 2.3 Part 3 (Perceptron)

The code loops over X iterations of batches size N. Batching of the data is done for Training and Validation (See [Batching section](#_Batching:) below for details). Error rate is calculated and displayed at the end, along with the production of multiple output files.

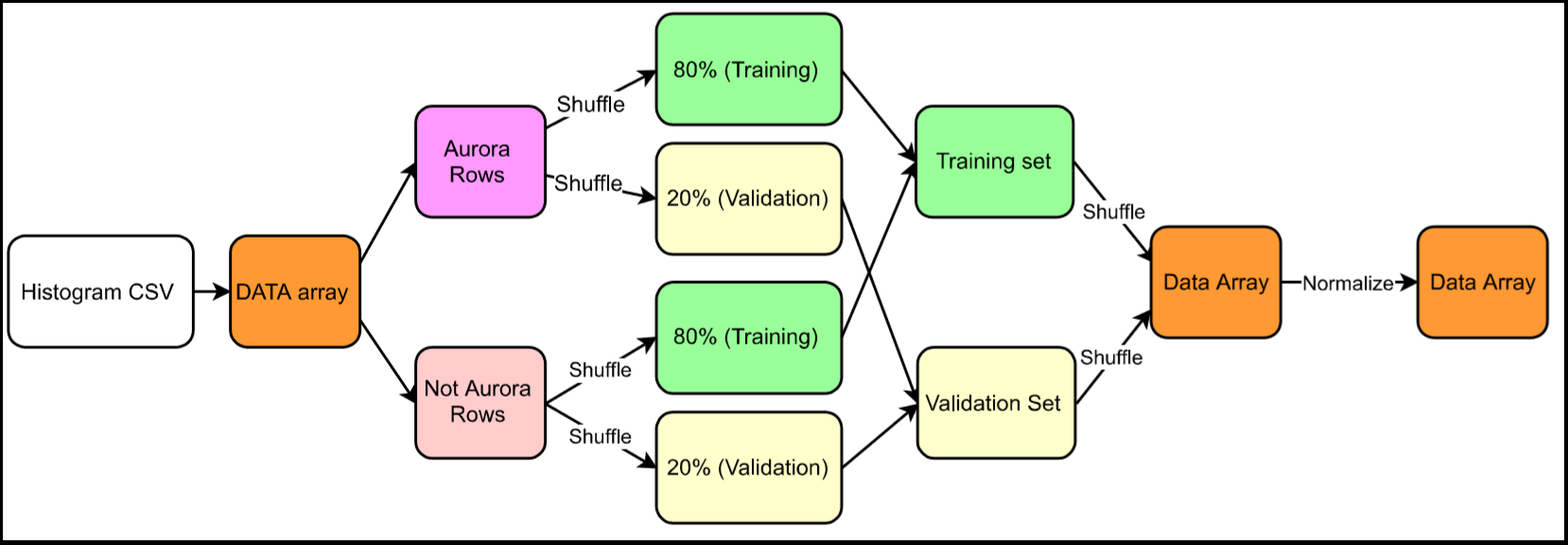
### 2.3.1 Output files:

|  |  |
| --- | --- |
| **Type** | **Content** |
| Plot | Weights vs color range |
| Plot | Training vs Validation error % |
| Plot | Training vs Validation error % (Averaged) |
| CSV | Weights (The weights for the 768 features) when training ends |
| Txt | WeightZero |
| Txt | Failed filenames from the final Validation batch |

### 2.3.2 Loading from CSV:

Data is loaded from the CSV file and goes through a series of steps to get the data into the correct format.

1. Histogram CSV file data is loaded into the [**DATA**] array.
2. [**DATA**] array is split into [**AURORA**] and [**NOT AURORA**] arrays.
3. [**AURORA**] and [**NOT AURORA**] arrays are shuffled (Randomized).
4. [**AURORA**] and [**NOT AURORA**] are split into 80% Training and 20% Validation.
5. Both 80% arrays are combined into the [**TRAINING**] set.
6. Both 20% arrays are combined into the [**VALIDATION**] set.
7. [**TRAINING**] and [**VALIDATION**] sets are shuffled
8. [**TRAINING**] and [**VALIDATION**] are “Stacked” ontop and replace the [**DATA**] array.
9. [**DATA**] array is normalized from 0-1.



### 2.3.3 Perceptron:

The perceptron was calculated via the below formula. Where a returned 1 means “Aurora” and a 0 is not.

If Weight\_zero + SUM(X[i]\*W[i]) > 0 then return 1 else return 0

In python to speed things up the numpy.dot method is used to get the dot product without looping.

### 2.3.4 Weights update:

The gradient decent algorithm was used and is as follows:

Weight[i] = Weight[i] - LearnRate (y^ – y) \* X[i]

Weight\_zero= Weight\_zero - LearnRate (y^ – y)

In python to speed things up, again the numpy.dot was used on the (LearnRate (y^ – y) \* X[i]) part. Then the resulting array was subtracted from the Weight array.

***NOTE:*** *The W0 (Weight Zero) is kept as a separate variable outside the array since it has no X to multiple against. In high sight, I could have shifted the DATA array and put 1 in the X0, then shifted the weight array and put WeightZero in for W0.*

### 2.3.5 Batching:

During a batch, rows from the TRAINING set are randomly picked until the batch size has been reached. Weights are updated after each row, not after each batch. Validation is done after every TRAINING batch. The same number of rows are randomly picked from the VALIDATION set.

**Example with a batch size of 50:**

Looping 50 times:

Pick a random row from the TRAINING set (80%)

Run perceptron code, update weights if incorrect classification, record errors for graph.

Looping 50 times:

Pick a random row from the VALIDATION set (20%)

Run perceptron code, record errors for graph.

# 3. How to use the scripts

There are three script. Two python, and 1 command file to execute multipule runs, some in parallel, against the perceptron.py file.

## 3.1 ColorHistogram.py

This program assumes there is an images directory with sub-directories aurora and none. It will go through both sub-directories, reading each file, generating a histogram, and outputting the results into aurora\_histogram.csv

### Usage:

python ColorHistogram.py

### Outputs:

Generates a CSV file with all AURORA and NOT AURORA histograms, 1 image per row.

For details on this file, see above in [section 2.2](#_2.1_Part_1).

## 3.2 preceptron.py

Yes, this is named incorrectly, should be perceptron.py. I left it as this because all my other code is assuming this name.

This program takes three parameters, and assumes an input file exists in the same directory named “aurora\_histogram.csv”.

### Usage:

python perceptron.py <learn> <iterations> <batchsize>

* learn: The learning rate (Mu)
* Iterations: Number of batches to run
* BatchSize: Number of random rows to select from the Training or Validation set during Training or Validation respectively.

### Outputs:

See [perceptron section 2.3.1](#_2.3.1_Output_files:) above for file details.

A printout to the screen of the overall (averaged) error for Training and Validation is printed to the screen.

The input parameters are printed.

The time information about runtime is printed.

## 3.3 runall.cmd

A simple windows batch file to run preceptron.py multiple times and in parallel.

### Usage:

runall.cmd