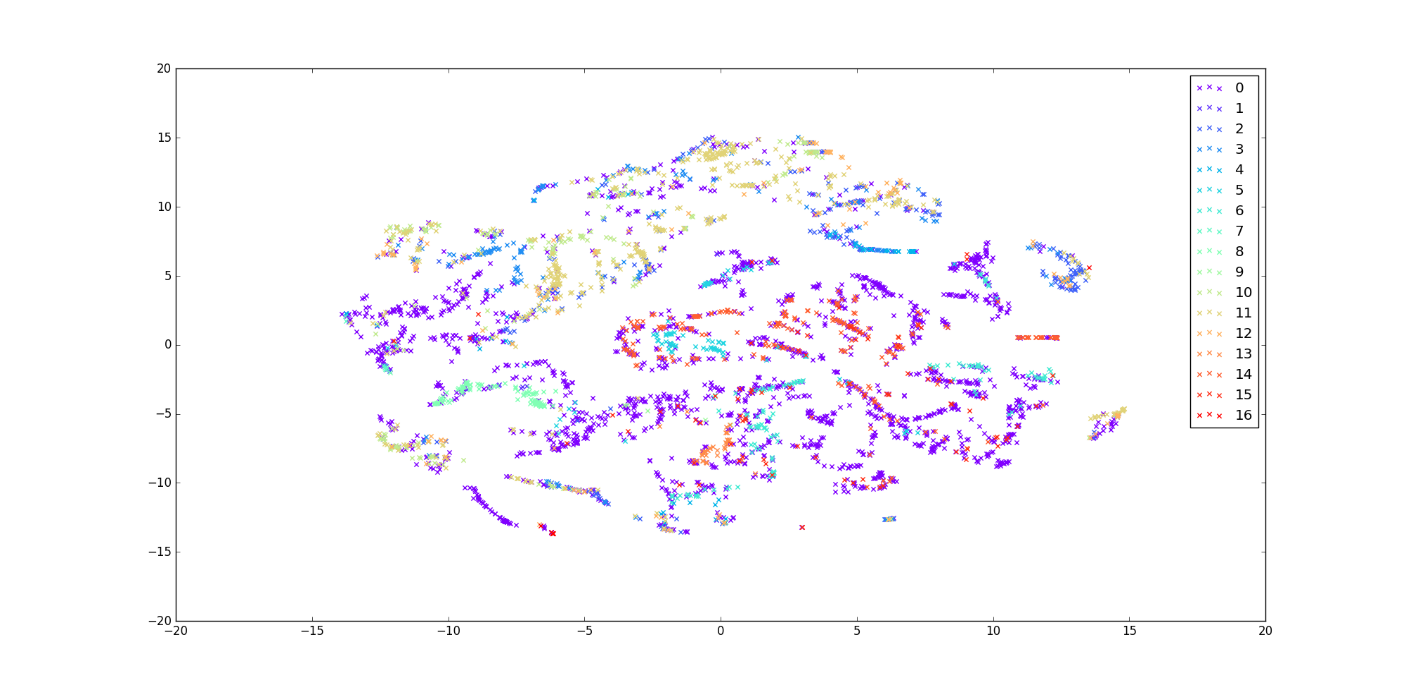
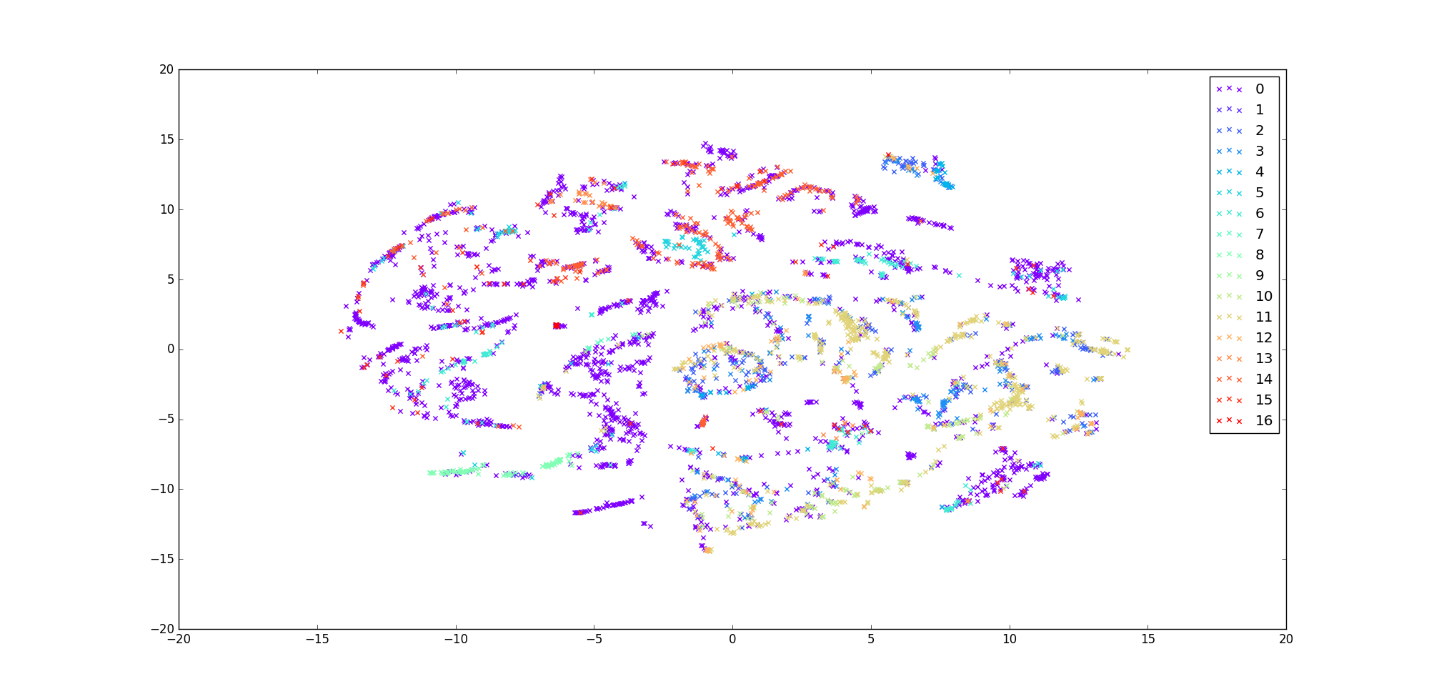
1. ***Visualisation :***

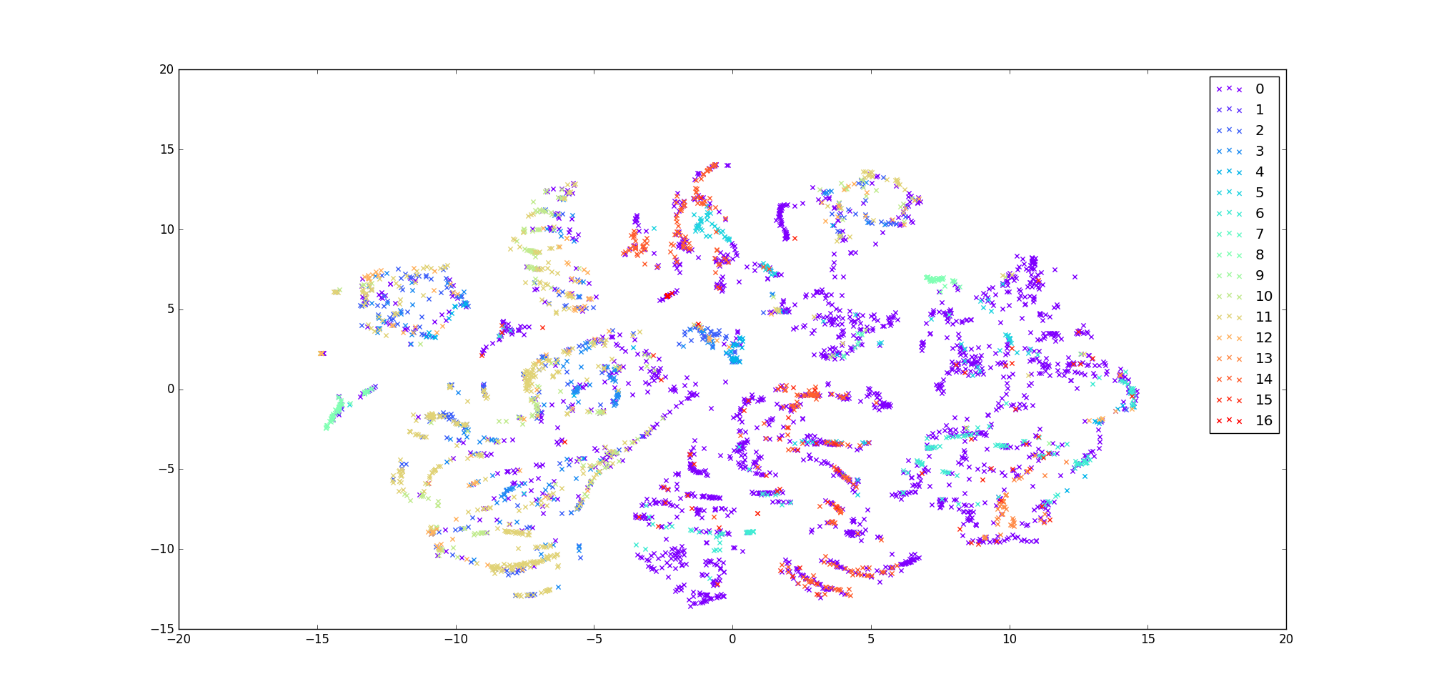
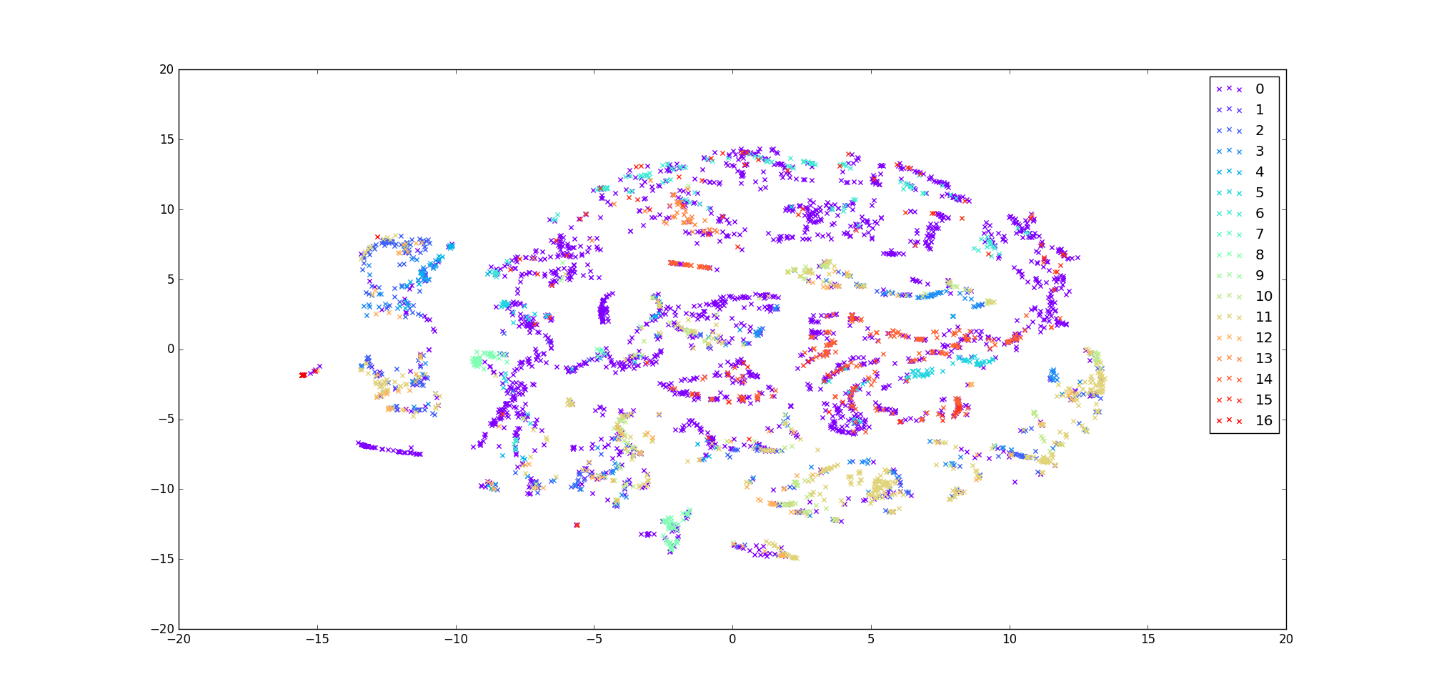
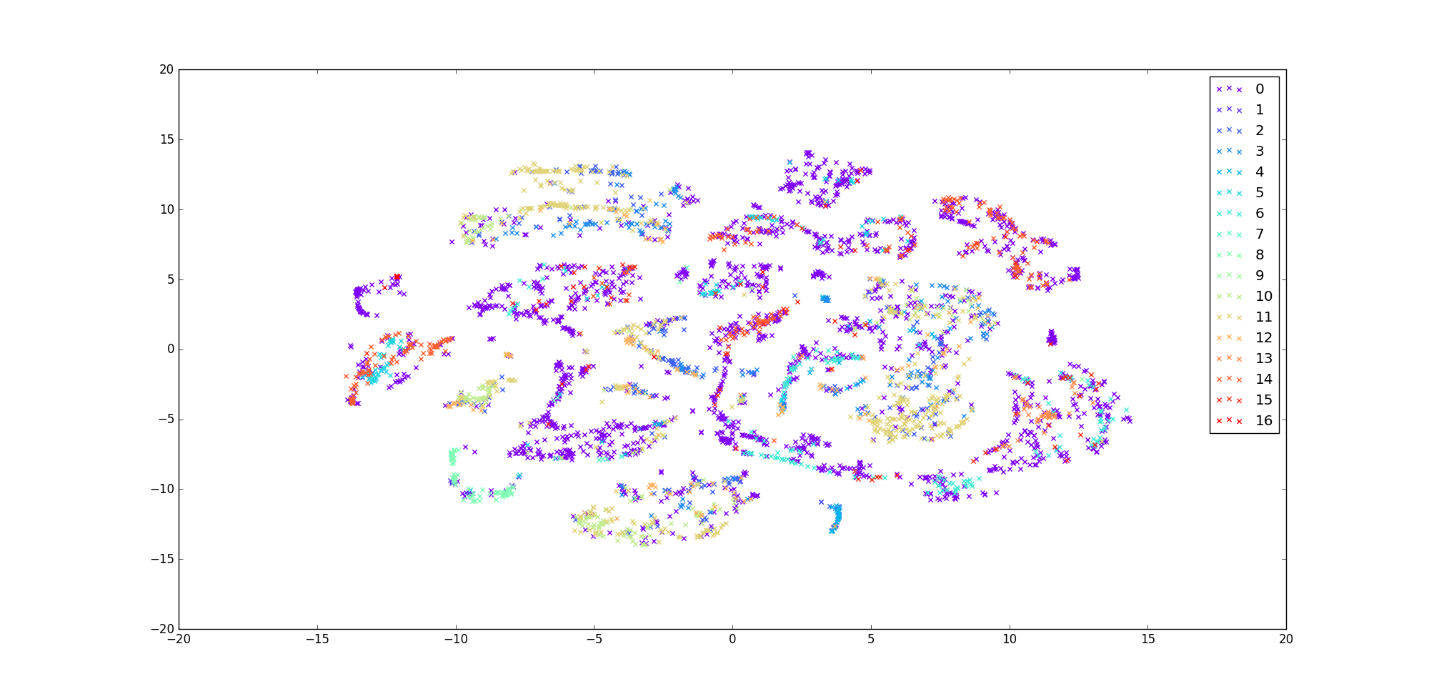
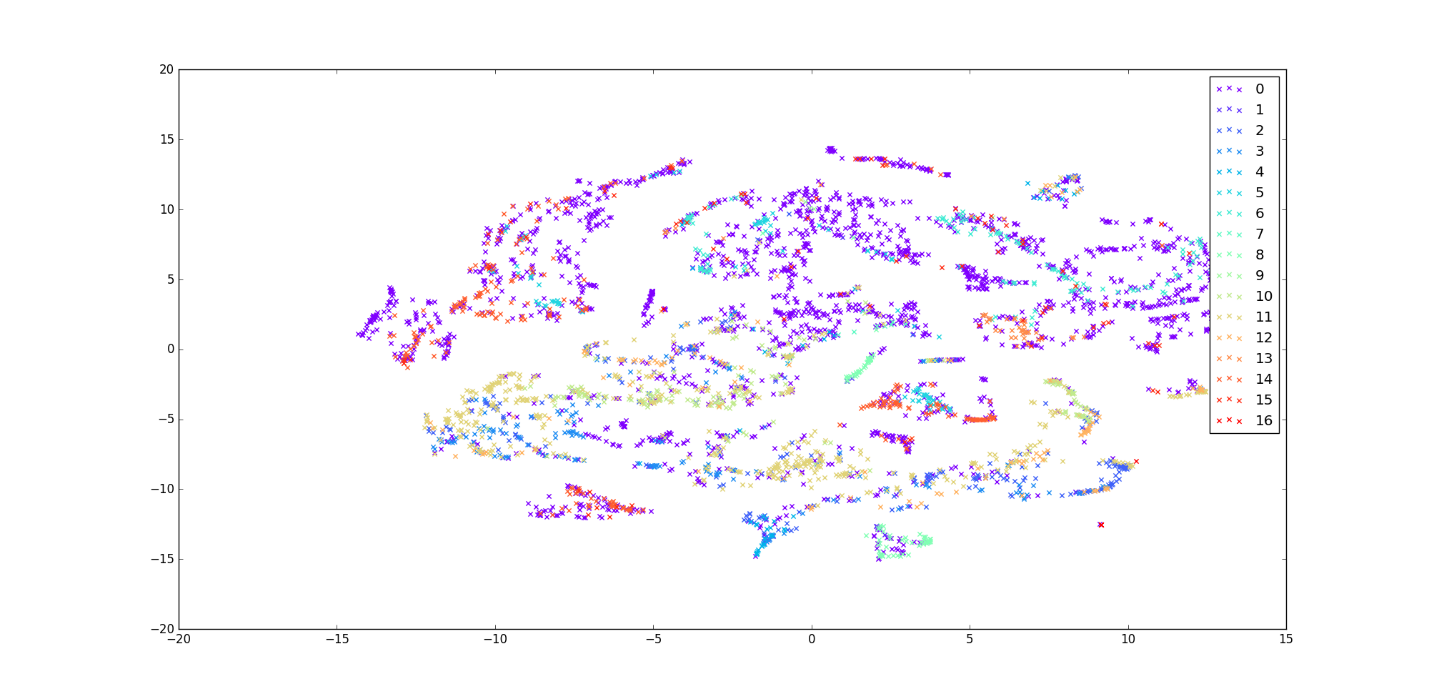
We used TSNE from from sklearn.manifold import :

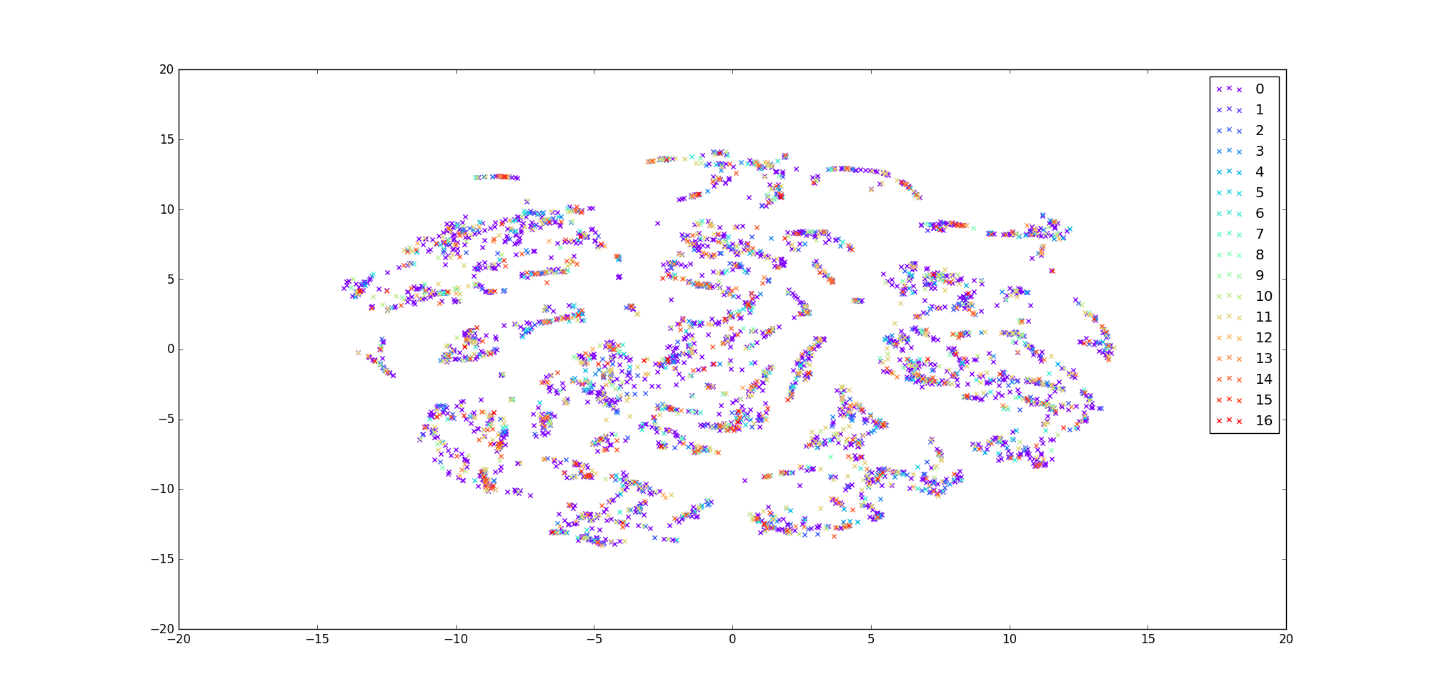
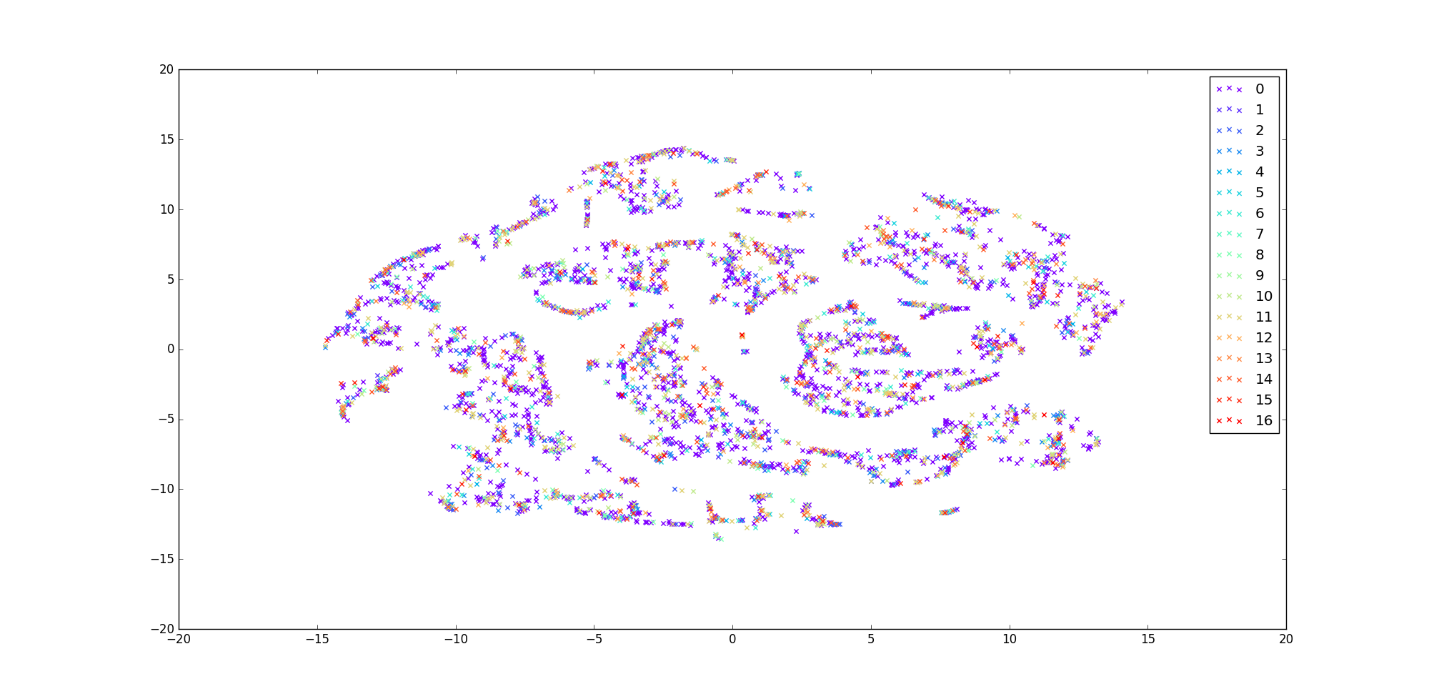
*X\_embedded = TSNE(n\_components=2, perplexity=40, verbose=2).fit\_transform(data)*

Parameter *data* was :

* 1. Raw data - We run two times with raw data and got those two results :



* 1. Data from PCA with 100 components
  2. Data from AE with 100 components
  3. Data from Stacked AE – 100 neurons – 50 neurons

******

We do not know how to interpret those images? Shapes are “similar” but we do not know if that similarities mean that methods work well.

I would say that the stacked AE is definitely not working. The points are more or less randomly intermixed. The others seem relatively equal but class 0 is certainly not helping as it overlaps the other classes. Can you repeat those without class 0? Also, did you try different perplexity parameters, I think that these can change the results quite a lot. It appears that this is quite a difficult problem!

1. ***PCA and LR***

We used PCA to reduce dimensionality to 100. After that, we used LR to classify pixels.

Train accuracy : 51%

Validation accuracy : 43%

Here is confusion matrix for train and validation set :

Train :

Predicted 0 11 All

Acctual

0 6449 35 6484

1 31 0 31

2 852 0 852

3 519 0 519

4 141 0 141

5 289 0 289

6 439 0 439

7 15 0 15

8 292 0 292

9 11 0 11

10 567 6 573

11 1448 0 1448

12 368 0 368

13 123 0 123

14 741 0 741

15 236 0 236

16 5 48 53

All 12526 89 12615

Validation :

Predicted 0 11 All

Acctual

0 1834 289 2123

1 7 0 7

2 284 0 284

3 165 0 165

4 50 0 50

5 42 60 102

6 137 0 137

7 7 0 7

8 111 0 111

9 6 0 6

10 199 0 199

11 519 0 519

12 104 0 104

13 35 0 35

14 80 178 258

15 72 5 77

16 21 0 21

All 3673 532 4205

1. ***SVM***

SVM is used from from sklearn, algorithm SVC

We tried SVM as classificator when input data are row data, data from autoencoder and data from stacked autoencoder . We noticed that varying parameters of SVM can significantly influence on results. The best results we got when use

* Kernel = default, in documentation it is ‘rbf’
* Gamma = 10
* C = 10
* class\_weight="balanced"

**Raw data :**

Predicted 0.0 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0 11.0 12.0 \

Acctual

0.0 632 18 41 81 88 125 102 20 15 39 54 43 68

1.0 0 11 0 0 0 0 0 0 0 0 0 0 0

2.0 4 0 159 13 27 0 0 0 0 0 26 47 5

3.0 0 0 11 110 7 0 0 0 0 0 0 35 4

4.0 2 0 0 1 44 0 0 0 0 0 0 0 1

5.0 8 0 0 0 1 83 3 0 0 0 0 0 1

6.0 3 0 0 0 0 0 153 0 0 0 0 0 0

7.0 0 1 0 0 0 0 0 4 0 0 0 0 0

8.0 0 4 0 0 0 0 0 1 92 0 0 0 0

9.0 1 0 0 0 0 0 0 0 0 2 0 0 0

10.0 1 0 1 9 8 0 0 0 0 0 135 24 5

11.0 3 0 21 49 10 0 0 0 0 1 48 355 5

12.0 1 0 0 4 5 0 0 0 0 0 5 4 79

13.0 1 0 0 0 0 0 0 0 0 0 0 0 0

14.0 2 0 0 0 0 2 1 0 0 0 0 0 0

15.0 17 0 0 1 0 2 5 0 0 5 0 0 0

16.0 0 0 0 0 0 0 0 0 0 0 0 0 0

All 675 34 233 268 190 212 264 25 107 47 268 508 168

Predicted 13.0 14.0 15.0 16.0 All

Acctual

0.0 13 371 419 12 2141

1.0 0 0 0 0 11

2.0 1 0 1 0 283

3.0 0 0 0 0 167

4.0 0 0 0 0 48

5.0 0 0 1 0 97

6.0 0 1 2 0 159

7.0 0 0 0 0 5

8.0 0 0 0 0 97

9.0 0 0 1 0 4

10.0 0 0 0 0 183

11.0 0 0 0 0 492

12.0 0 0 0 0 98

13.0 50 0 0 0 51

14.0 0 251 9 0 265

15.0 0 9 46 0 85

16.0 0 0 0 19 19

All 64 632 479 31 4205

So SVM gets around 53% classification overall (but more meaningful classifications)…

**Data from AE – compressed**

Predicted 0 1 2 3 4 5 6 7 8 9 10 11 12 13 \

Acctual

0 590 18 41 88 69 110 110 20 20 27 49 30 61 22

1 0 11 0 0 0 0 0 0 0 0 0 0 0 0

2 0 0 154 18 40 1 0 0 0 0 25 36 15 0

3 0 0 8 104 6 0 0 0 0 0 3 27 6 0

4 0 0 0 2 43 0 1 0 0 0 0 0 2 0

5 10 0 0 0 1 83 0 0 0 0 0 0 1 0

6 1 0 0 0 0 1 148 0 0 0 0 0 0 0

7 1 0 0 0 0 0 0 4 0 0 0 0 0 0

8 0 4 0 0 0 0 0 1 85 0 0 0 0 0

9 0 0 0 0 0 0 0 0 0 2 0 0 0 0

10 0 0 2 11 18 1 0 1 0 0 143 12 7 0

11 2 0 37 62 26 3 0 1 0 0 55 303 11 0

12 0 0 8 3 11 0 0 0 0 0 6 2 81 0

13 1 0 0 0 0 0 0 0 0 0 0 0 0 44

14 2 0 0 0 0 4 0 0 0 0 0 0 0 0

15 9 0 0 0 0 0 5 0 0 2 0 0 0 1

16 0 0 0 0 0 0 0 0 0 0 0 0 0 0

All 616 33 250 288 214 203 264 27 105 31 281 410 184 67

Predicted 14 15 16 All

Acctual

0 385 497 13 2150

1 0 0 0 11

2 0 0 0 289

3 0 0 0 154

4 0 0 0 48

5 1 2 0 98

6 0 5 0 155

7 0 0 0 5

8 0 0 0 90

9 0 0 0 2

10 0 0 0 195

11 0 1 1 502

12 0 0 0 111

13 0 0 0 45

14 237 13 0 256

15 14 42 0 73

16 0 0 21 21

All 637 560 35 4205

And 49.8% when using autoencoder output…

Without class\_weight="balanced" we got prediction for only one class ( class number can vary depends on other SVM parameters)

SVM seems to be doing quite well. Again class 0 messes things up though. But this could be because it is using the balanced classes setting, from the documentation it gives weights: n\_samples / (n\_classes \* n\_class\_samples) which is more likely to give a value between 0 and 1, except when a class is very rare (python code: n\_samples / (n\_classes \* np.bincount(y)) [http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html]. With the autoencoder it does slightly worse but not greatly. Did you choose the best SVM parameters for each method or use the same for both? If so, perhaps different parameters are better when using the AE output.

1. ***Removing class 0***

Data: Indian Pines – 145x145 = 21025 pixels, 200 bands, 17 classes

After removing class 0 (unclassified data): 10249 pixels, 16 classes

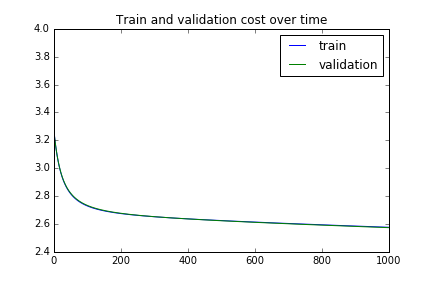
Autoencoder: one hidden layer, units per layer: [200, 100, 200]

Classifier: logistic regression

Without class balancing

**Results**

Cost function:

****

I am surprised that the validation set cost follows the training set… have you tried for 2000 iterations to see if it increases at some point? Perhaps this indicates a problem. We can discuss on Monday.

Predicts 5/16 classes. For other classes, there is not even one prediction. Confusion matrix for training set looks like this:

Predicted 1 5 7 10 13 All

Acctual

0 0 27 0 1 0 28

1 18 5 0 823 0 846

2 3 1 0 492 0 496

3 17 17 0 109 0 143

4 5 75 0 20 213 313

5 0 361 0 4 61 426

6 0 0 0 20 0 20

7 73 132 14 57 0 276

8 0 11 0 0 0 11

9 0 1 0 583 0 584

10 0 12 0 1477 0 1489

11 2 0 0 363 0 365

12 0 115 0 0 0 115

13 0 6 0 0 743 749

14 1 107 0 12 114 234

15 0 0 0 54 0 54

All 119 870 14 4015 1131 6149

1. ***Class balance – cost function***

Same as previous.

Added class balancing to the cost function. To every class was given a weight coefficient, so classes with fewer pixels can have larger weight and larger impact on cost value.

Coefficients are calculated as follows:

,

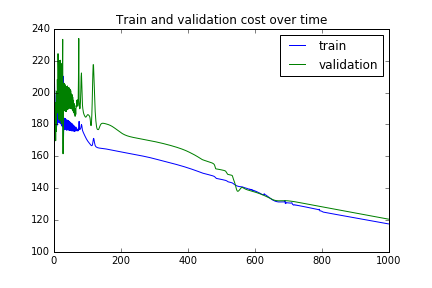
- number of pixels in class ,  
 - total number of pixels  
 – percentage share of -th class  
 – weight coefficient of -th class for cost function, which is inverse proportional to percentage and gives larger weights to smaller classes.

Is that how they do it in some of the papers? I am wondering whether the weight should be scaled to between 0 and 1 in the cost function? (Also, see above comment re cost function)

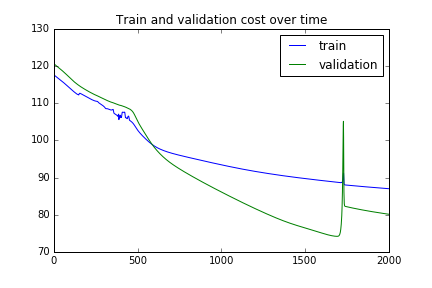
**Results:**

Cost function becomes pretty unstable, because of huge differences in weights of different training examples. It has different behavior in every run.

For 1000 epochs:



For 2000 epochs:



We don’t know if something is wrong with, or it’s just unstable because of weight coefficients.

Confusion matrix for training set, 2000 epochs:

Predicted 1 4 7 10 13 14 15 All

Acctual

0 0 2 28 0 0 0 0 30

1 421 7 7 408 0 15 0 858

2 170 4 1 311 0 7 0 493

3 82 8 1 26 0 37 0 154

4 3 236 7 0 31 13 0 290

5 0 108 3 0 0 318 0 429

6 0 3 10 0 0 0 0 13

7 0 3 284 0 0 0 0 287

8 0 5 0 0 0 7 0 12

9 136 14 4 442 0 6 0 602

10 139 25 11 1294 0 19 0 1488

11 122 5 5 188 0 19 0 339

12 0 0 0 0 0 118 0 118

13 0 121 0 0 629 7 0 757

14 2 47 0 0 58 113 0 220

15 7 0 0 0 0 0 52 59

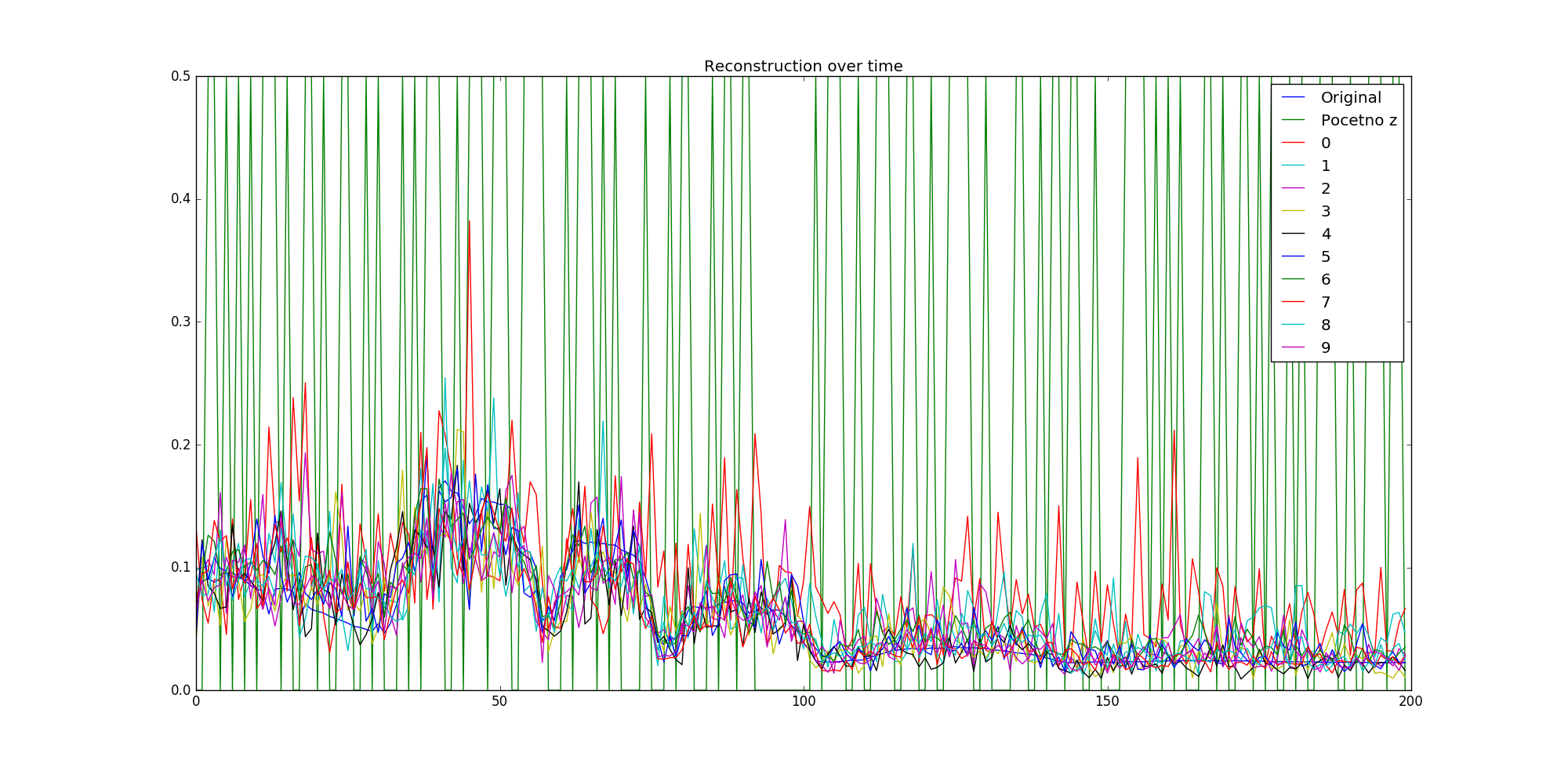
All 1082 588 361 2669 718 679 52 6149

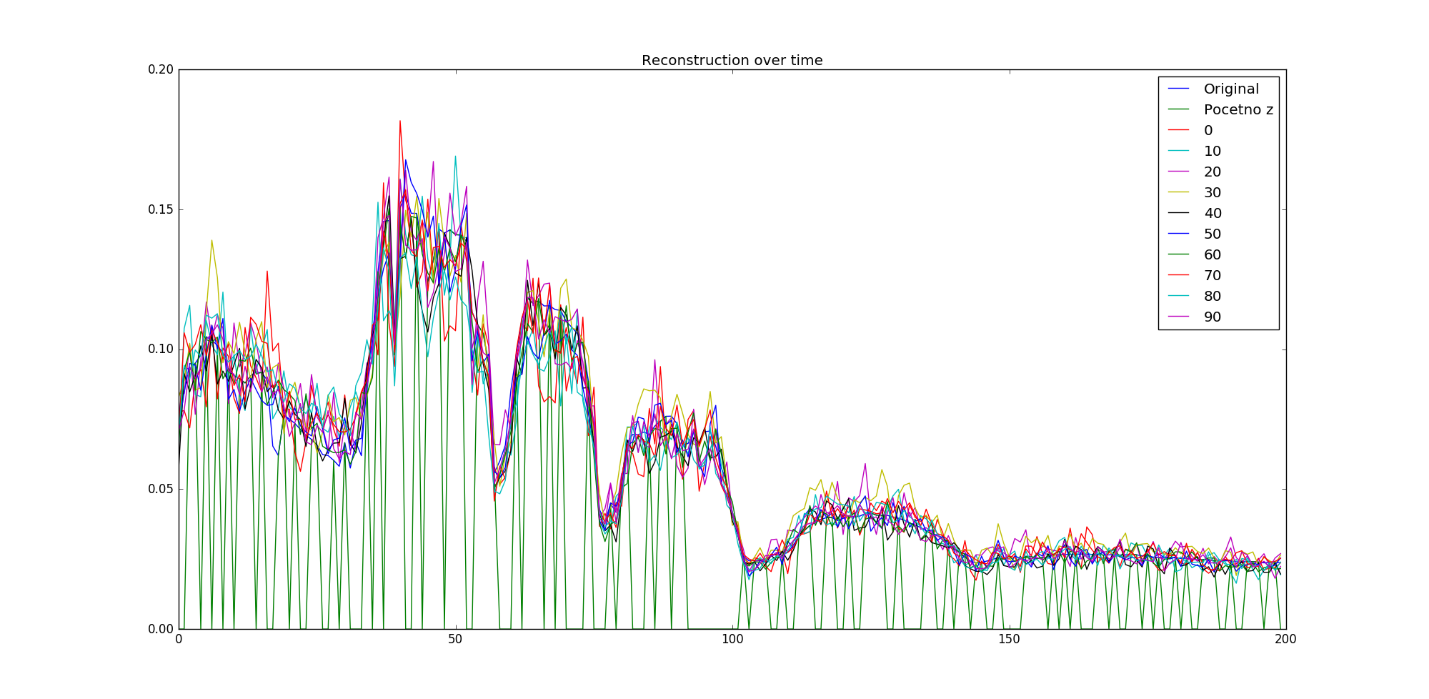
Predicted 1 4 7 10 12 13 14 15 All

Now it predicts 7/16 classes. It seems there are too few examples for other classes to be properly trained, even with weighted cost.

1. ***Adding noise to AE***

We added noise to data by randomly pick bands and set their value to 0. Interesting thing is that AE very fast converge ( almost with same speed when use non-corrupted data). We tried different corruption level and results are below:

* Fixed corruption : We used corruption of 100% where every band is replaced by 0 or 0.5
* Corruption of 50% - randomly picked bands are set to 0



We did not expect that AE will converge so soon. After first epoch, signal shape is almost caught. Because of it, we tried to use fixed corruption, but results are same. Maybe reason is in fact that all band’s values are near 0 ( 0. 07431977,0. 10454439,…) so removing some bands do not affect results.

Do you normalise the spectrum before adding the noise or afterwards? Also, 100% and 50% corruption seems like quite a lot, is this what they’ve done in the papers?

For normalization we used Euclidian distance. Are we correct? Maybe this kind of noise is not appropriate for this data… or maybe we missed something else :)

Euclidean distance sounds fine to me...

Also, cost function is changed very slow (generally, in every case of AE ) – in first couple of iteration change is huge, but after, say 5 epoch, changes are on third decimal. Is that usually for this kind of problem? Maybe it indicated that something is wrong….

Is there a learning rate parameter in the implementation? Does this change with time (training iteration)?

One more question: If we get similar cost value for different methods (for example adding noise and without noise), does that mean that classification will be almost the same, generally?

If it is the same data then generally yes. It doesn't mean that you will get the same classification results because it could do worse on one example and better on another and that would have the same effect on the cost value but different classification results.