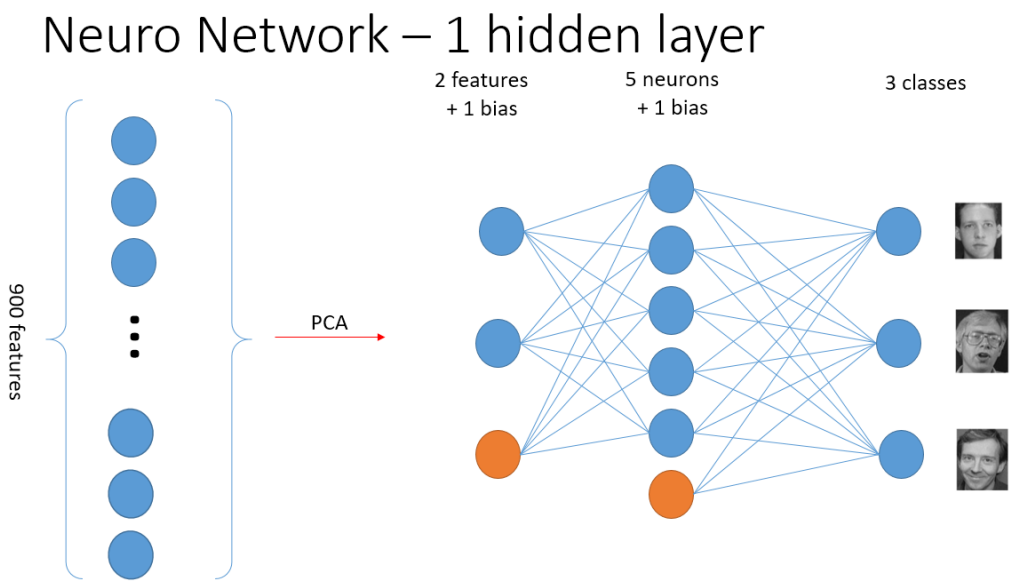
# Machine Learning – Homework 3

0550193 陳昱瑋

4/24/17

A. Model

≒ First Model




Steps:

Data: 2400 training data and 600 testing data / cross-validation method

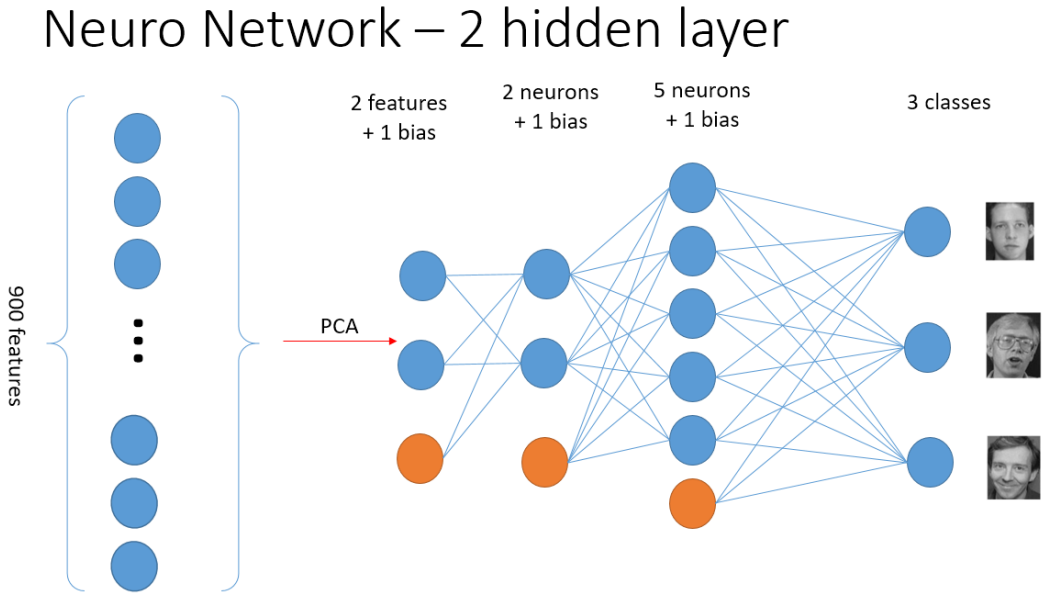
# 1 Use Principal component analysis (PCA) to map data down to 2 dimensions

# 2 Stochastic Gradient Descent in back propagation

# 3 Implement the neural network model with 1-hidden layer

# 4 Choose the sigmoid function as the activation function for tuning

≒ Second Model



Steps:
Data: 2400 training data and 600 testing data / cross-validation method

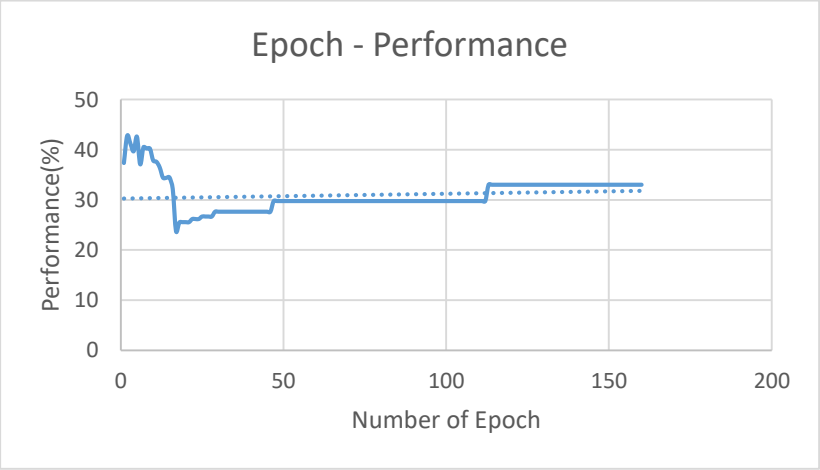
# 1 Use Principal component analysis (PCA) to map data down to 2 dimensions
# 2 Stochastic Gradient Descent in back propagation
# 3 Implement the neural network model with 2-hidden layer
# 4 Choose the rectified function as the activation function for tuning

B. Explanation

As required, the network is updated using stochastic gradient descent. However, my training result doesn't go well in SGD, I will talk more about my failure reason later. I have searched several methods on the reference part and one of the main method I would like to discuss is "momentum method" updating the weight. It provides a pulse each iteration updating the weight to help me get out of the regional optimal solution. In my following report most of the result doesn't implement momentum method, though, I will add the result combining mini-batch and momentum method to make performance a giant leap in return.

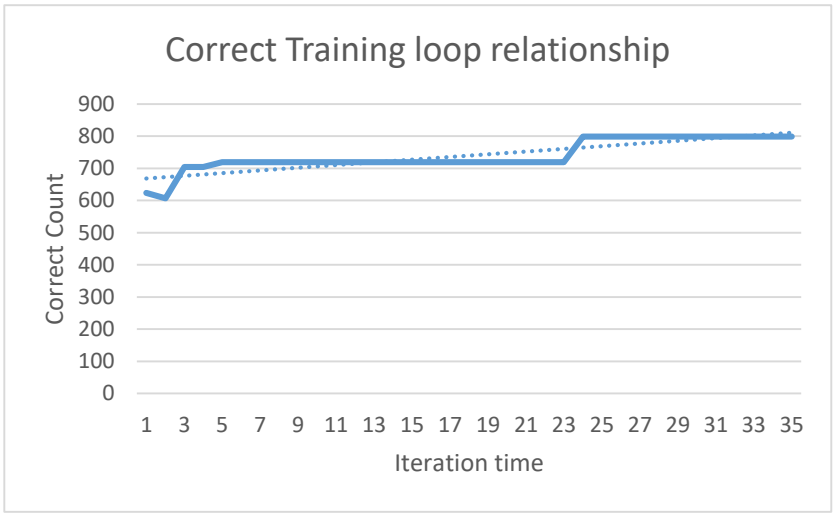
C. Results

- Epoch and Training Error relation
  SGD, mini-batch = 4, learning rate = 0.01, training data = 2400




- Learning Rate and Training Convergence

SGD, learning rate = 0.01, training data = 2400




I get a saddle point for my SGD method. :(

D. Compare with homework 2 result

From homework 2 discriminative model:

| E. Accuracy(Phi) | 2 | 3 | 4 | 5 | 10 | 50 | 100 | 200 | 500 | Average |
|---|---|---|---|---|---|---|---|---|---|---|
| $w_0$ | 20.6 | 45.33 | 46.666 | 46.83 | 34.66 | 32.66 | 32.5 | 33.83 | 32.0 | 36.12% |
| No $w_0$ | 41.5 | 42.33 | 40.5 | 41.16 | 32 | 34 | 35 | 38 | 34 | 37.61% |

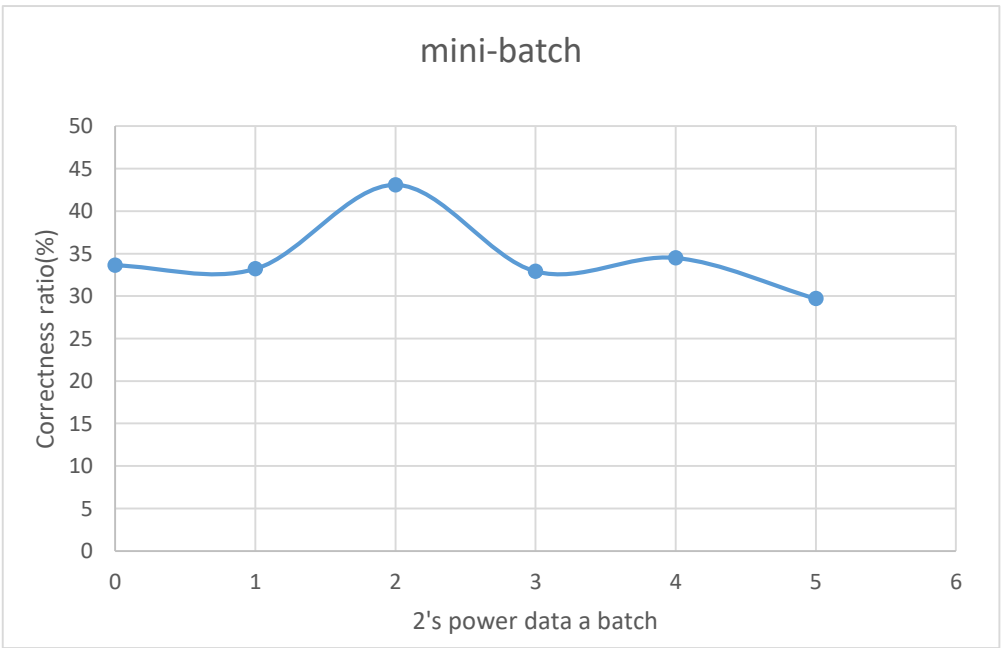
I don't have good enough model of choosing phi = 2 PCA result.

For this time, neural network, should easily beat the homework 2 works.

I get around 30 in SGD result, which is much similar performance as my discriminative model do.

F. Mini-batch bonus

here I use SGD method over 1 hidden layer neural network and found that in roughly $2^2$ data per batch will have better performance than other higher batch number.



G. Terms

## - batch gradient descent

Batch gradient descent is to consider all the data and then update the weight according to cost function as below:

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

And repeatedly update the parameters using all m examples in each iteration. Typically, the batch gradient descent approaches the global minimal value step by step.

## - mini-batch gradient descent

Compared to batch and SGD, the main difference is that mini-batch gradient descent use b examples in each iteration. Typical value of b is from 2 to 100.

Get b examples $(x^{(i)}, y^{(i)}), \dots, (x^{(i+b)}, y^{(i+b)})$

Choke the dataset every b examples, you will get m/b mini-batches.

We will choose mini-batch over stochastic because stochastic approaches the global minimal value in random process, mini-batch will show its power by getting the mean value of 10, 20 or b data mean value instead of randomly pick a direction.

However, mini-batch gradient descent requires extra time to figure out what's the good value of b.

## - stochastic gradient descent

Stochastic gradient descent compare to batch gradient descent is to update the weight each time the data it read, but we need to shuffle the dataset to get less bias data.

1. Randomly shuffle dataset
   (Recall : Bootstrap in lecture note)
2. Repeat

for $\mathrm{i} := 1, \dots, \mathrm{m}\{$

$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$

}

}

And repeatedly use 1 example in each iteration to update the parameters. Eventually, the parameters will move to some area which is really closed to global minimal value. This is good enough for us to have model trained.

## - online gradient descent.

Update the parameter for each training pattern.

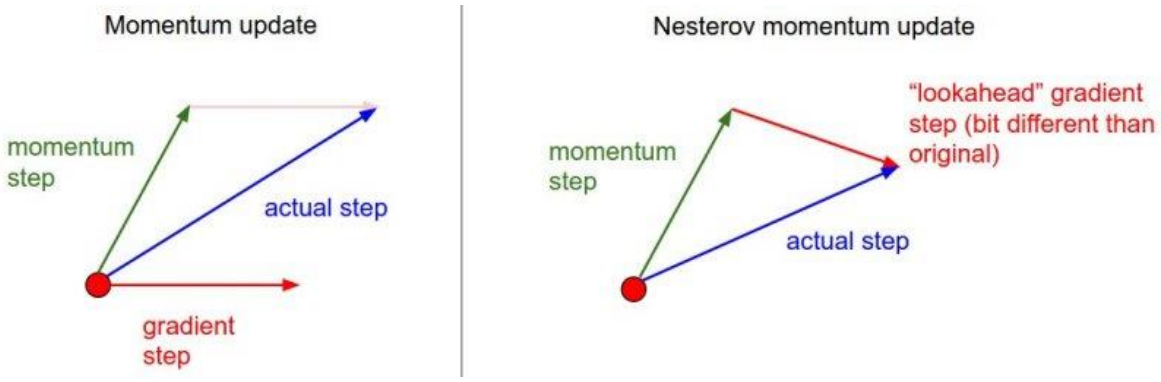
H. Disscussion

1. Little trick here is that for 2-layer neural network training, I chose Momentum

```
# Vanilla update
x += - learning_rate * dx
```

```
# Momentum update
v = mu * v - learning_rate * dx # integrate velocity
x += v # integrate position
```

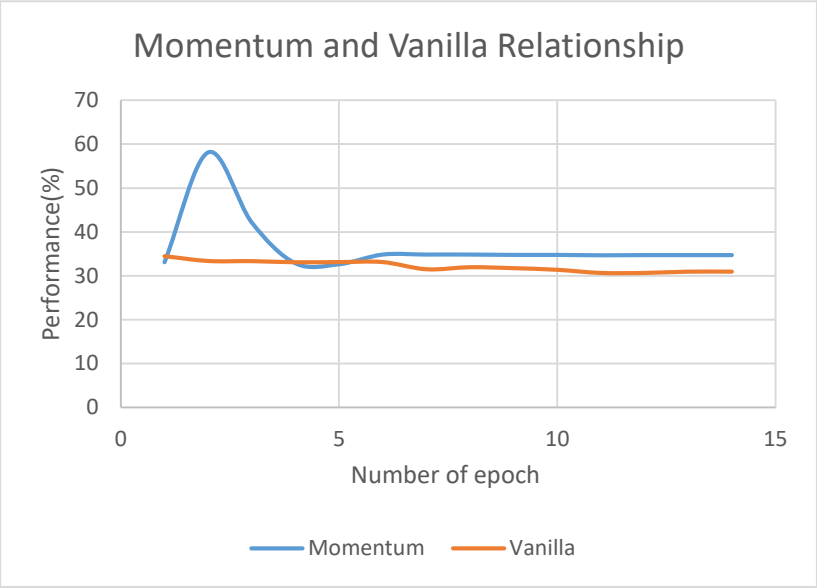
update instead of original method (Vanilla update). (reference cs231n, Stanford)




(http://cs231n.github.io/neural-networks-3/#sgd)

Quote: *With Momentum update, the parameter vector will build up velocity in any direction that has consistent gradient.*

| | Momentum | | | Vanilla | | |
|---|---|---|---|---|---|---|
| epoch | performance | correct | total | performance | correct | total |
| 1 | 33.08333 | 794 | 2400 | 34.45833 | 827 | 2400 |
| 2 | 58.125 | 1395 | 2400 | 33.375 | 801 | 2400 |
| 3 | 42.125 | 1011 | 2400 | 33.33333 | 800 | 2400 |
| 4 | 32.83333 | 788 | 2400 | 33.08333 | 794 | 2400 |
| 5 | 32.625 | 783 | 2400 | 33.125 | 795 | 2400 |
| 6 | 34.83333 | 836 | 2400 | 33.125 | 795 | 2400 |
| 7 | 34.83333 | 836 | 2400 | 31.5 | 756 | 2400 |
| 8 | 34.83333 | 836 | 2400 | 31.95833 | 767 | 2400 |
| 9 | 34.75 | 834 | 2400 | 31.75 | 762 | 2400 |
| 10 | 34.75 | 834 | 2400 | 31.375 | 753 | 2400 |
| 11 | 34.66667 | 832 | 2400 | 30.66667 | 736 | 2400 |
| 12 | 34.70833 | 833 | 2400 | 30.66667 | 736 | 2400 |
| 13 | 34.70833 | 833 | 2400 | 30.95833 | 743 | 2400 |
| 14 | 34.70833 | 833 | 2400 | 30.95833 | 743 | 2400 |




2.

During the training process, the invalid value outcome terminates the training process and I cannot find the solution instantly. However, after clearly go through the code, I found that I wrong use the softmax- backpropagation as the first process instead of using rectified-backpropagation.

but instead, I should use rectified backpropagation as first processing, as below:

```
def backpropergation(self,row):

    dE_2 = self.sigmoid_BackProp(self.y1, self.y[row])
    self.dE2 = np.asarray(dE_2).reshape(3,1)
    dEdz_1, self.dEdW_11, self.dEdb_11 = self.InnerProduct_BackProp(se
    dEda_1 = self.sigmoid_BackProp(dEdz_1.T,self.a0)
    self.dE1 = np.asarray(dEda_1).reshape(5,1)
    dEdz_0, self.dEdW_01, self.dEdb_01 = self.InnerProduct_BackProp(se
```

```
def backpropergation(self,row):

    dE_2 = self.rectified_BackProp(self.y1, self.y[row])
    self.dE2 = np.asarray(dE_2).reshape(3,1)
    dEdz_1, self.dEdW_11, self.dEdb_11 = self.InnerProduct_BackProp(self
    dEda_1 = self.sigmoid_BackProp(dEdz_1.T,self.a0)
    self.dE1 = np.asarray(dEda_1).reshape(5,1)
    dEdz_0, self.dEdW_01, self.dEdb_01 = self.InnerProduct_BackProp(self
```

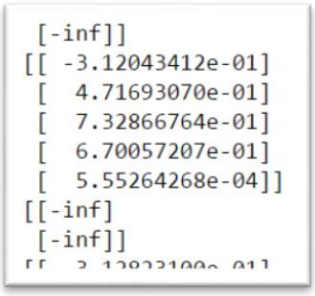
```
Networks = Network(data)
Networks.training()

C:\Python27\lib\site-packages\ipykernel\__main__.py:250: RuntimeWarning: overflow encountered in exp

Iteration: 1
Difference : inf
Correctness : 2400. Training using : 240
Iteration: 2
Difference : nan
Correctness : 2400. Training using : 480
Correct number : 2400
Wrong number : 0
Correct rate : 1
> Done training.

C:\Python27\lib\site-packages\ipykernel\__main__.py:229: RuntimeWarning: invalid value encountered in absolute
```

(the figure above demonstrate the programming error I have in my model; the yellow circled function should be counter-function of another yellow function; vise versa) Otherwise the dEdz_1 will eventually have runtime error. we can see that the weight will converge to Nan or Inf value eventually.
Runtime Error

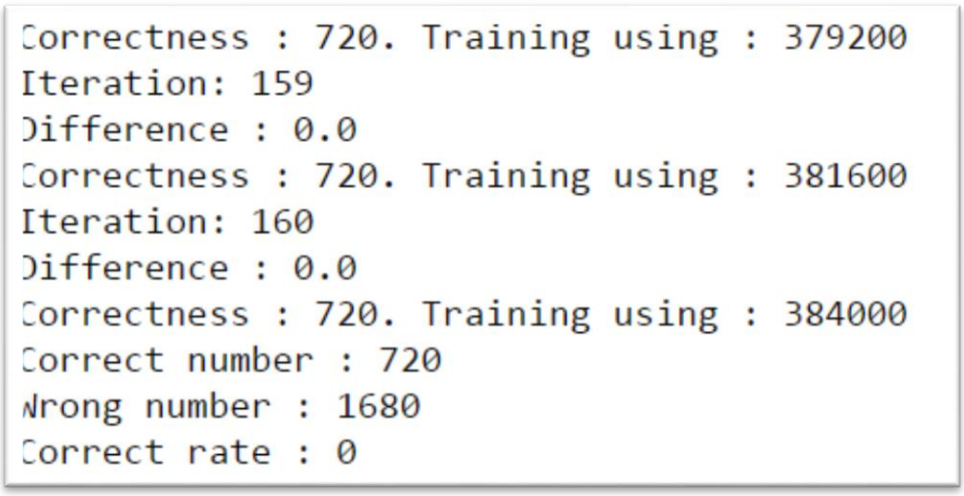


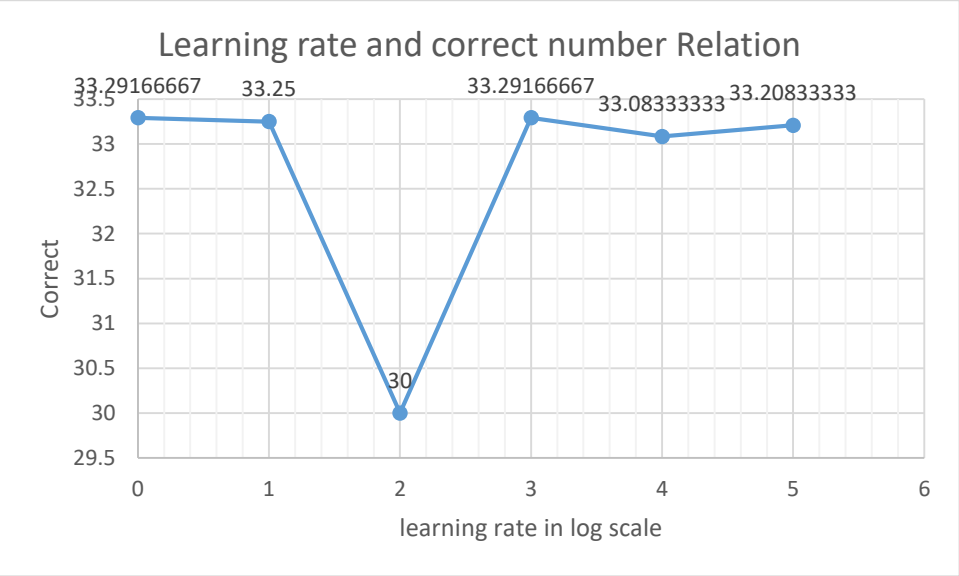

3. The result doesn't converge! (local minimal)

I have used 384,000 data to train my neural network with gradient descent, however, it's best performance doesn't meet my expectation. I will change my approach strategy:

i. Train model using mini-batch descent
ii. Modify learning rate

Here I just use 160 iterations as terminated condition for time-saving reason.






Choosing the suitable learning rate is also critical in neural network training process.

## I. Conclusion

In the end, I have discussed many relation and possibilities which might deeply infect the training result and expected computing time. However, that's the fun part about data science. There are still lots of work can be done this homework. But for now, I have tried my best to sort out the main idea behind the neural network building process.

What's more, in addition to the requirement of this time homework, I google other machine learning course on Coursera, MIT, Stanford cs231n and NTU machine learning. Thank you for the patience reading the last word of this report. For the further work should be discuss on my blog and my Github.

## J. References

1. Neuro Network and Deep Learning .com
   http://neuralnetworksanddeeplearning.com/chap1.html
2. How to Implement the Backpropagation Algorithm From Scratch In Python
   http://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/
3. Neural Network Playground
   http://playground.tensorflow.org/#activation=sigmoid&batchSize=15&dataset=xor&regDataset=reg-plane&learningRate=0.03&regularizationRate=0&noise=5&networkShape=4&seed=0.57464&showTestData=false&discretize=false&percTrainData=50&x=false&y=false&xTimesY=false&xSquared=true&ySquared=true&cosX=false&sinX=false&cosY=false&sinY=false&collectStats=false&problem=classification&initZero

=false&hideText=false&resetButton_hide=false

4. Basic Python Network
   http://iamtrask.github.io/2015/07/12/basic-python-network/
5. Deep Learning Basics: Neural Networks, Backpropagation and Stochastic Gradient Descent
   http://alexminnaar.com/deep-learning-basics-neural-networks-backpropagation-and-stochastic-gradient-descent.html