Ahmad Hussameldin Hamed Hassan

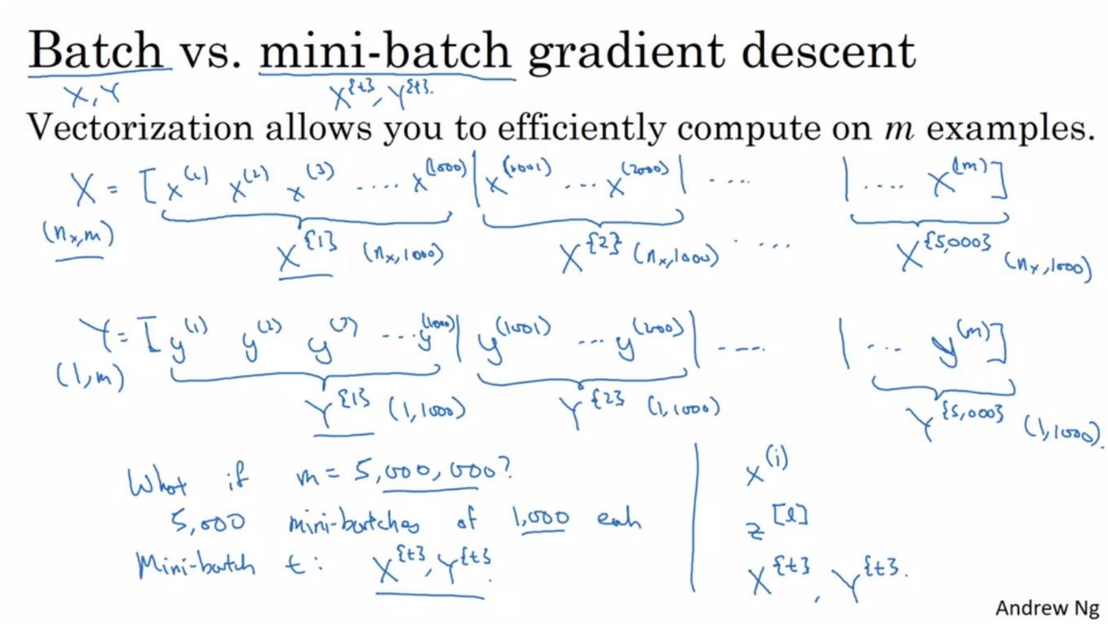
Shared Git-hub link: https://github.com/ahmadhassan1993/sharing-github

Improving Deep Neural Networks: Hyperparameter tuning, Regularization and Optimization

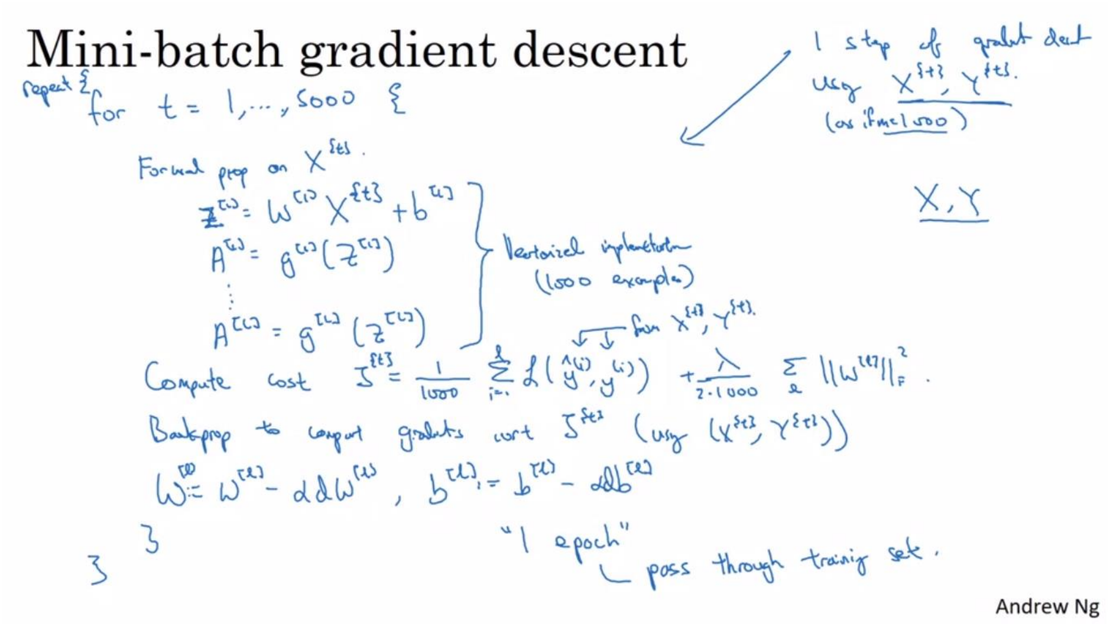
Week 2 Summary

We need optimization algorithms to fast our NN in large data set.

1- One possible algorithm is Mini-Batch gradient descent, which divide the whole m examples of data set to mini batches:

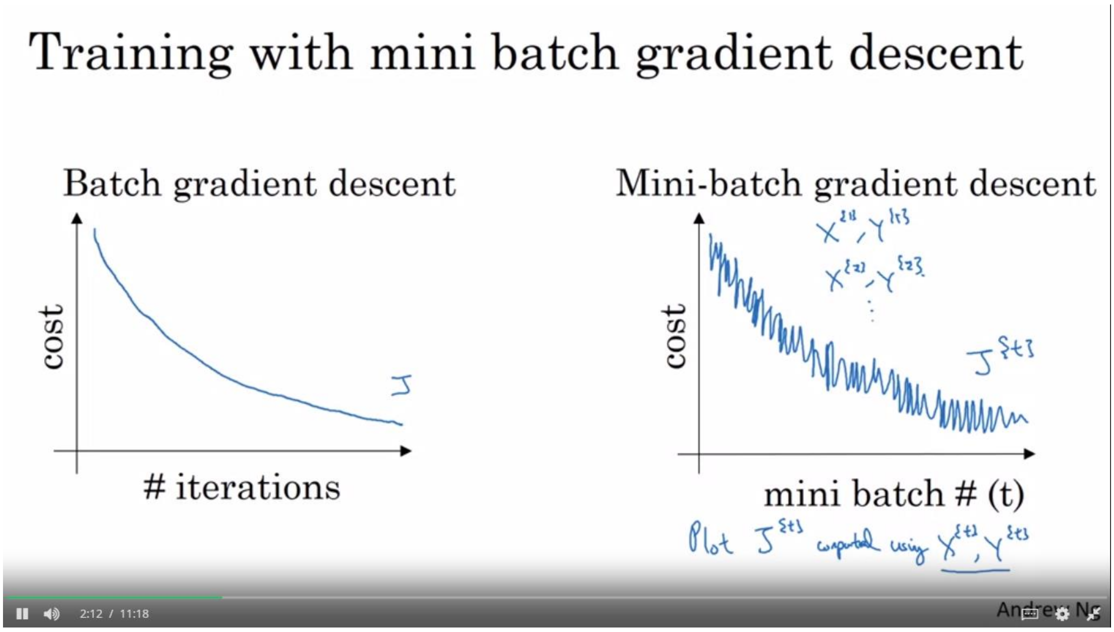


We use {t} to index the mini batch number. For example, if m=5000 then t will range from 1 to 5000.



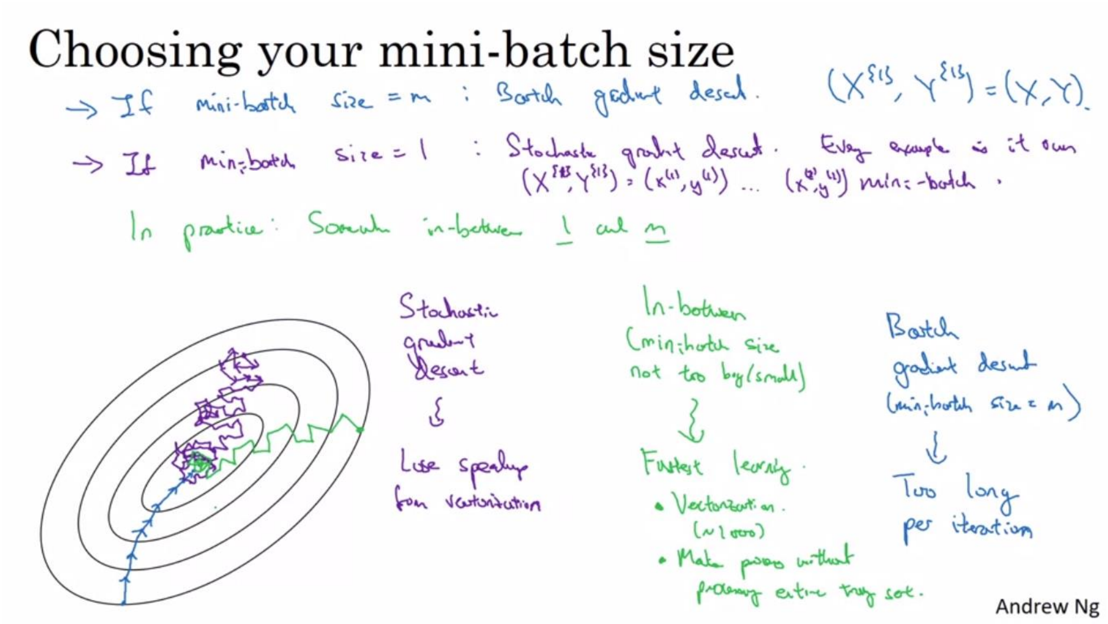
1 epoch is the single pass through training set, i.e, the output of both forward and backward propagation for the whole batch.

The cost function plot for {t} will be oscillating:



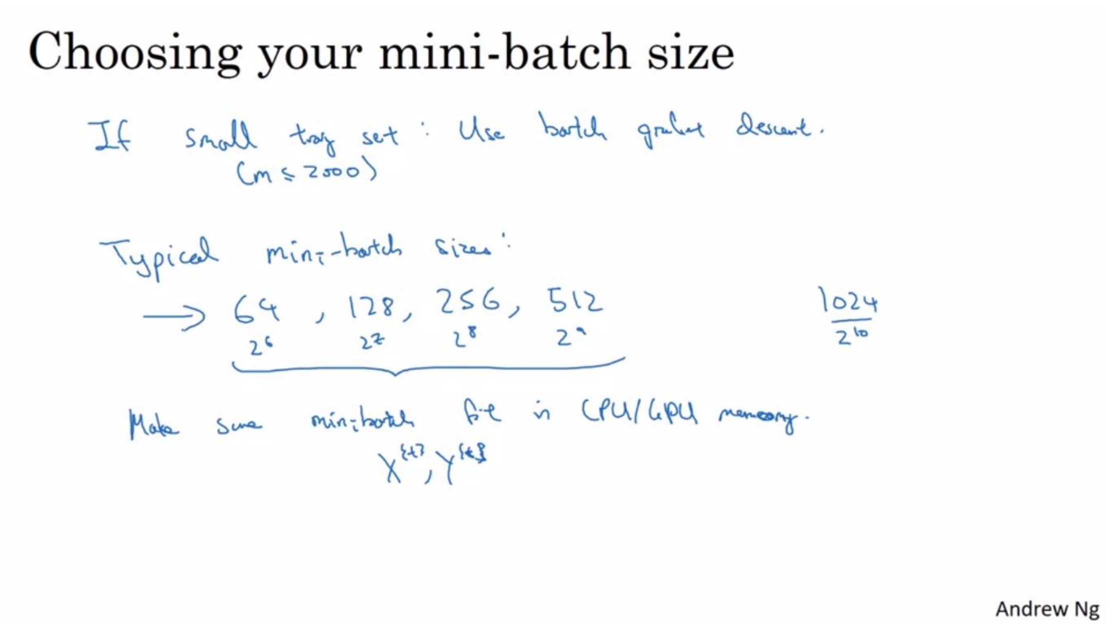
Because different epochs will have different strength on the cost function.

Choosing the size of the mini batch:



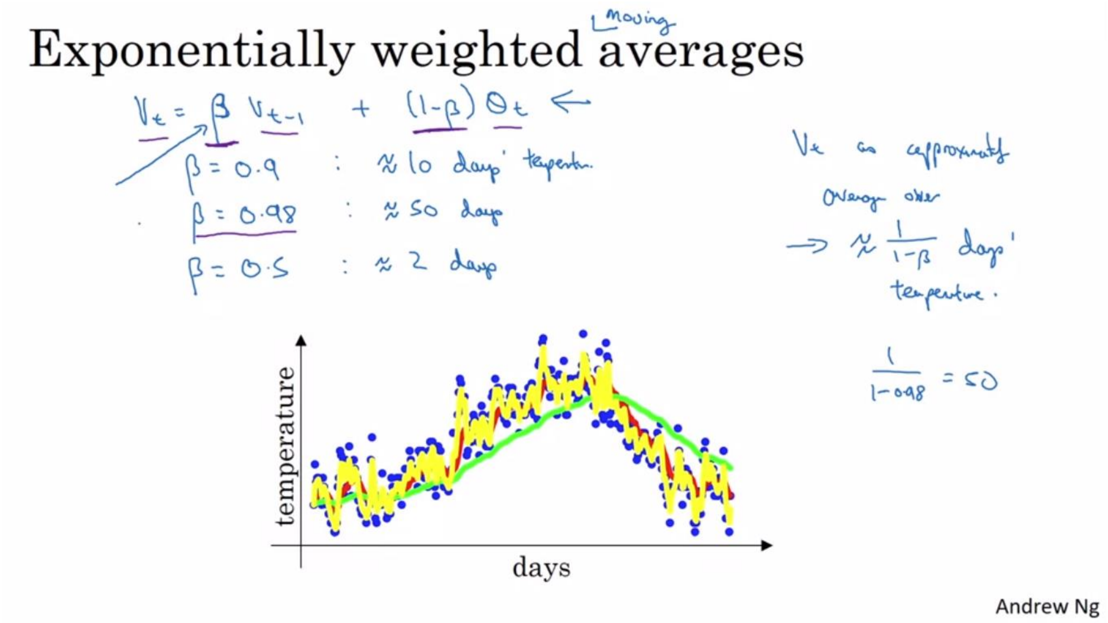
If mini batch size =m, so the full batch is processed at a time and longer time to solve. If size=1, called stochastic, we will lose both speed up from vectorization and minimum is not reached. Optimum is to take size not too big and not too small. We will see progress without even completing the full data set.

The mini batch size is a Hybrid Parameter that we should choose well. If the training set is small (<2000), then we should use the batch gradient descent. Typical size takes the power of 2:

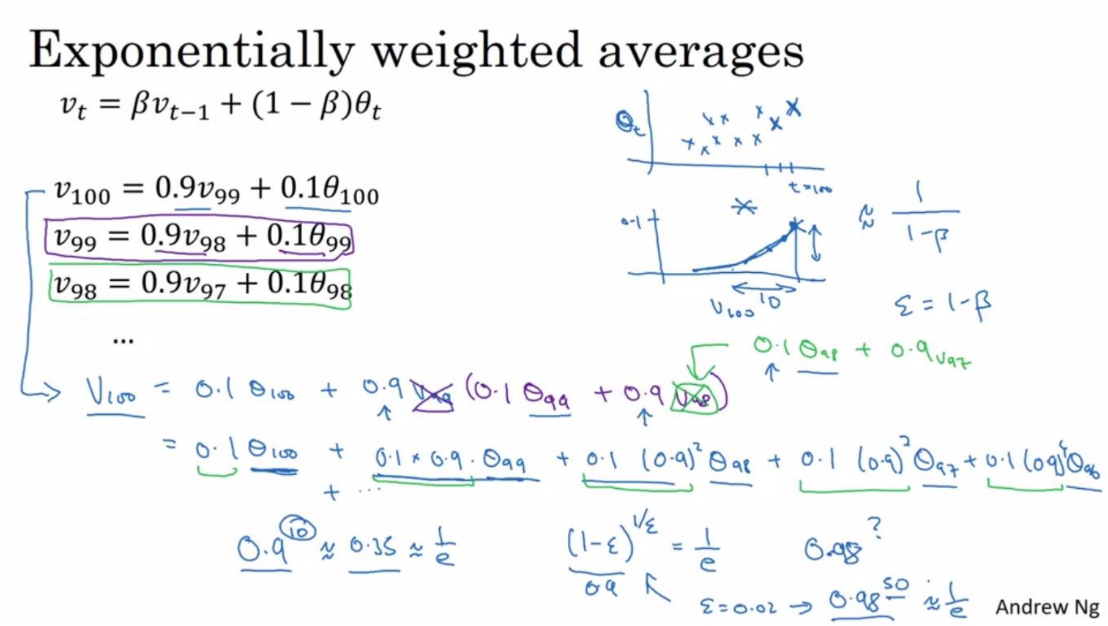


We also must be sure that the size fits the memory of our computer.

2- Exponentially Weighted Average (moving average):

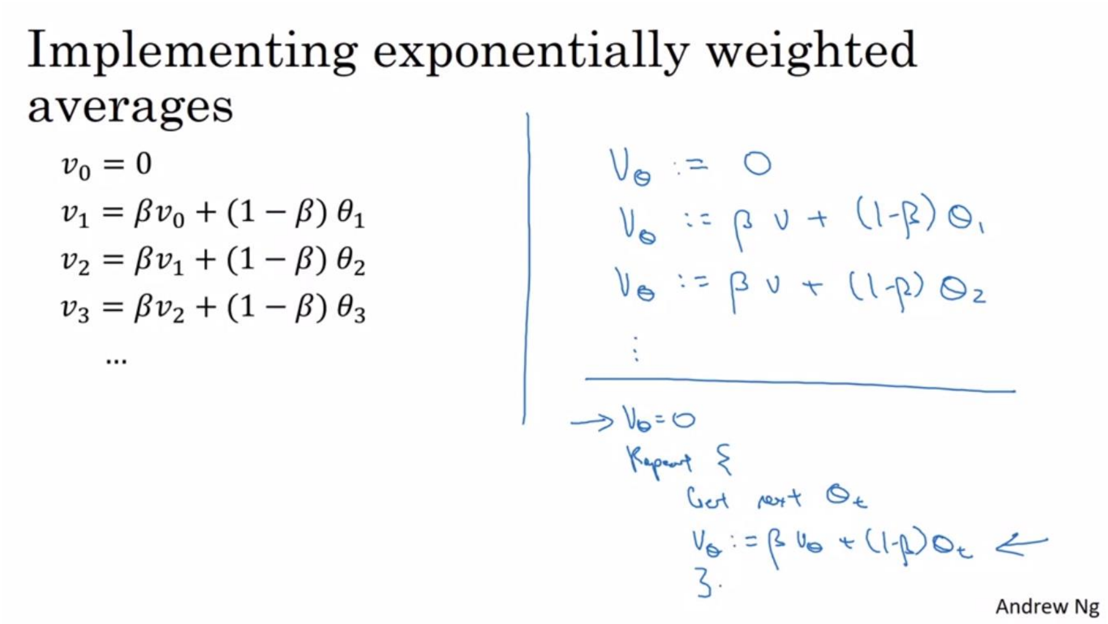


In this example we see the trend (average) of varying temperature over the whole year. A hybrid parameter β is used to decide over which number of days we take the average (1/(1-β)). We should choose value that give us a suitable number of days, not too big not too small.

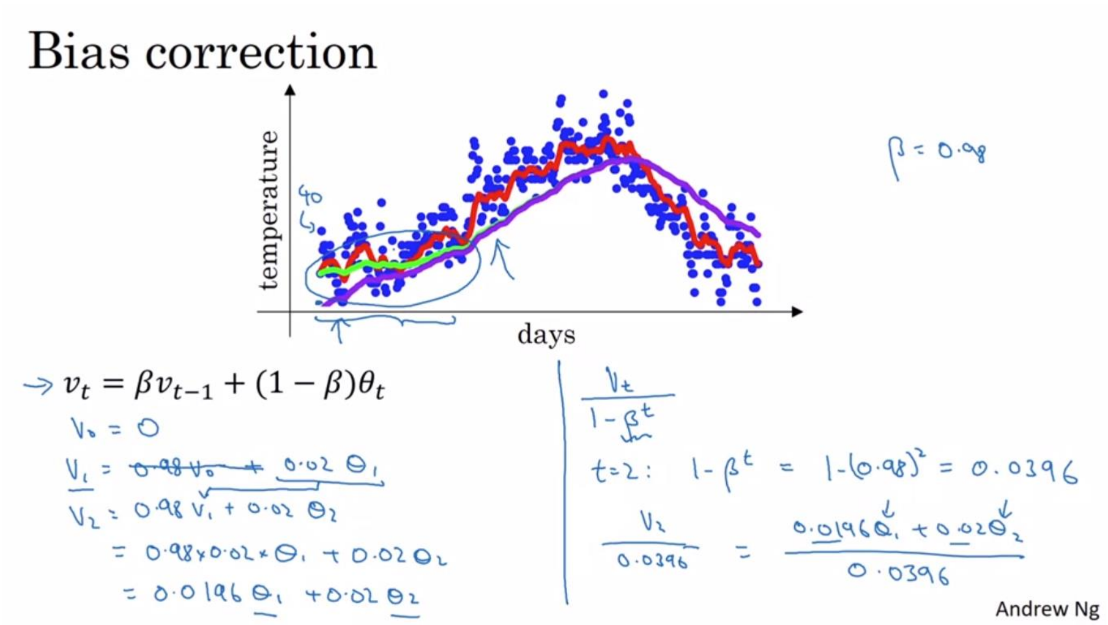


It is called exponential, because over n days the temperature degree increases exponentially from day 0 to day n. this is according to β^(1/β).

Implementation by single line of code:



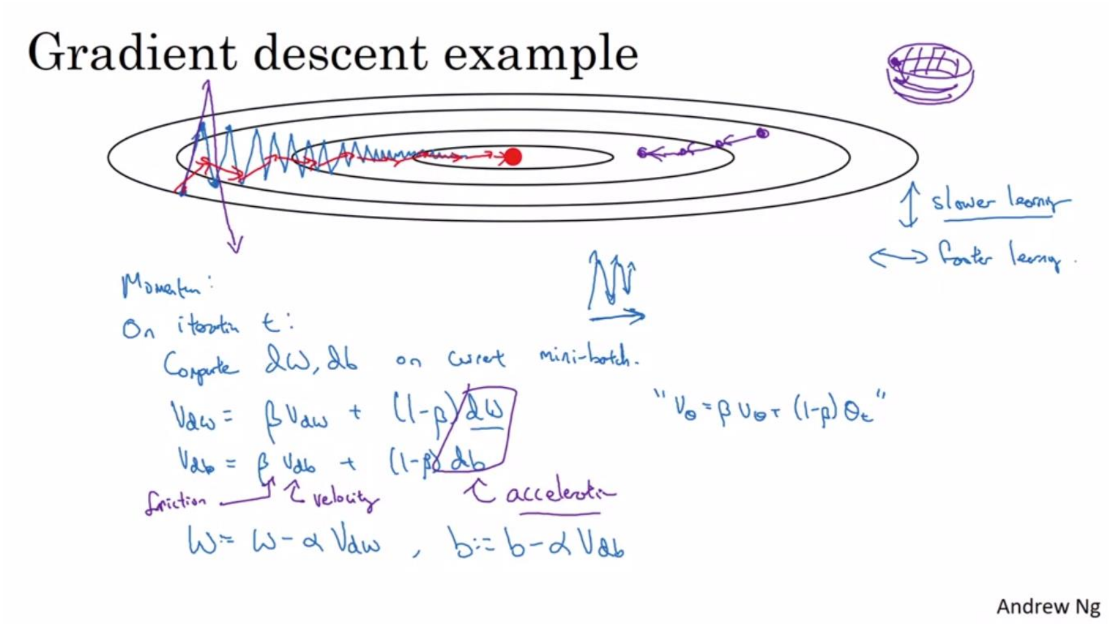
We use Bias to correct the first values:



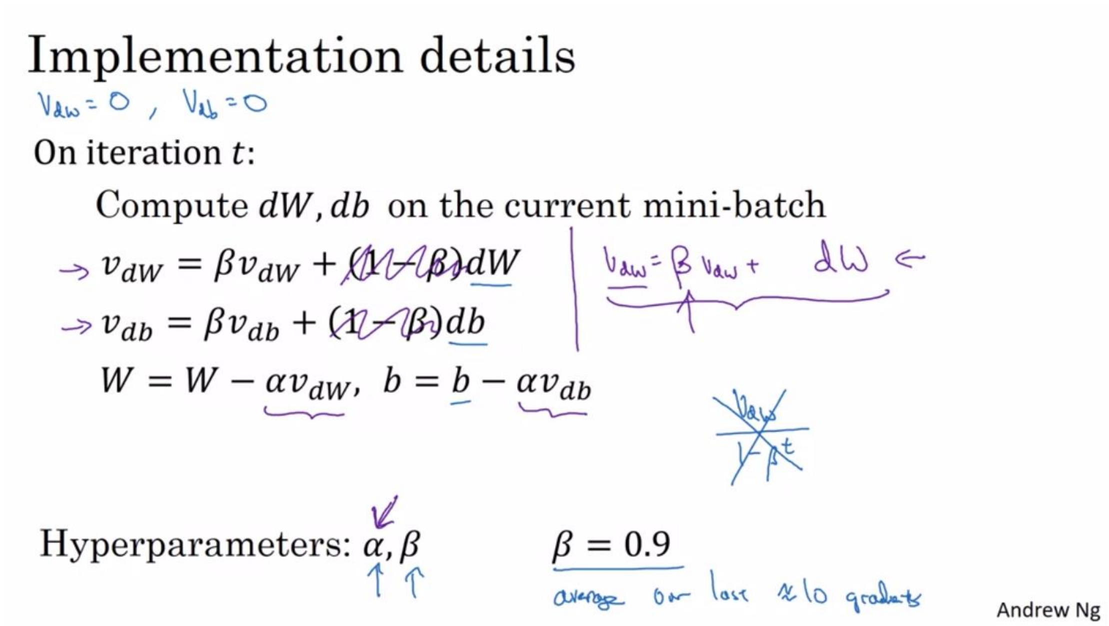
Bias=1-β^t, where t is the number of the day that we want to calculate temperature in it. We will divide the calculated average by this bias for the whole days.

3- Gradient Descent with Momentum:

We need to go faster horizontally not vertically to the solution:

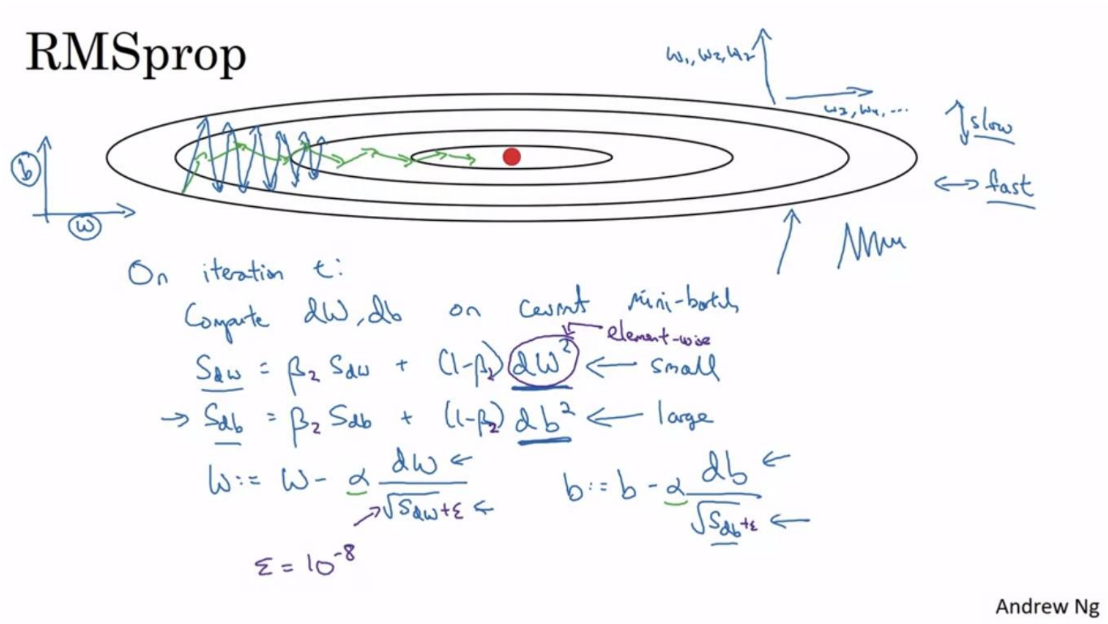


The most common value for β is 0.9. in that case, we neglect the following in equations as well as the Bias:



4- RMSprop in Gradient Descent:

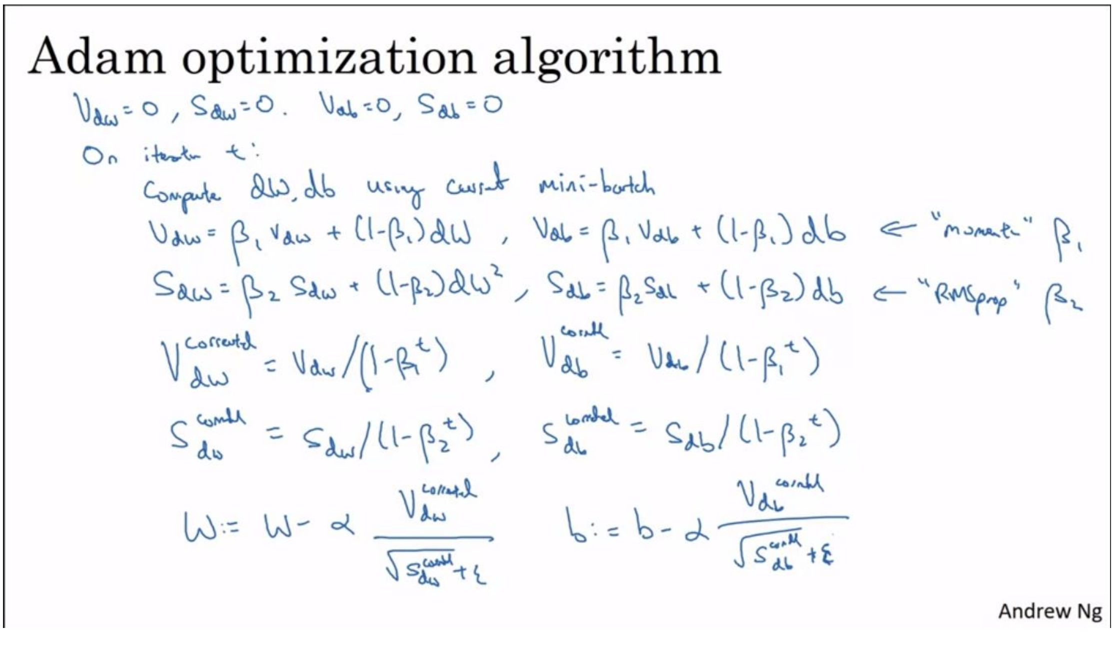
We use the following equations:

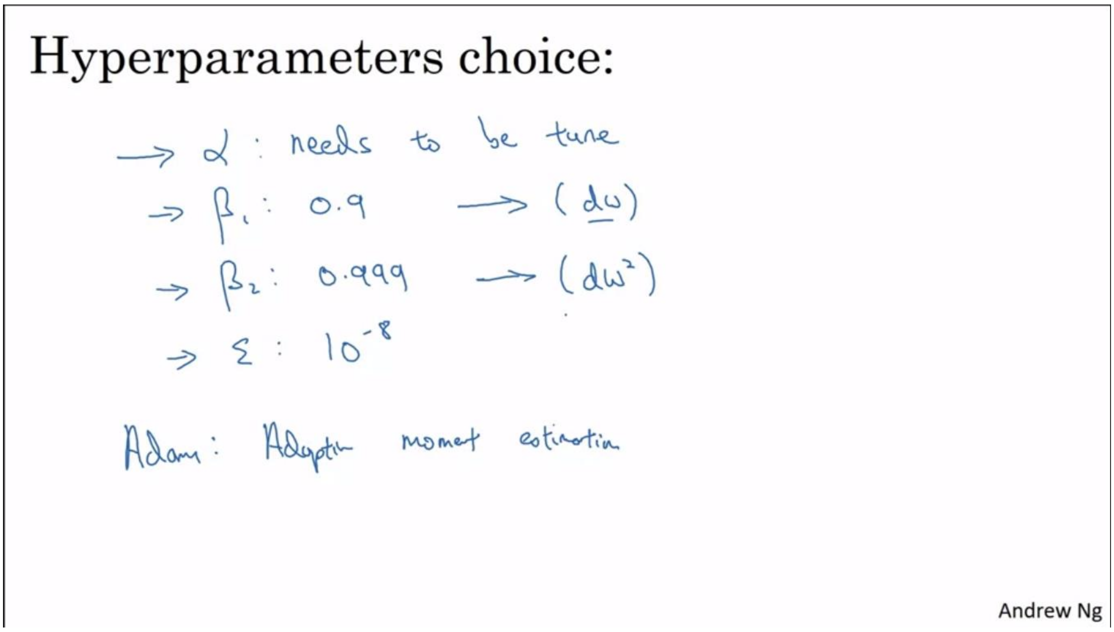


Where β2 is different than β that used in momentum. We add ε (very small) to ensure that the root is zero it will not affect the result.

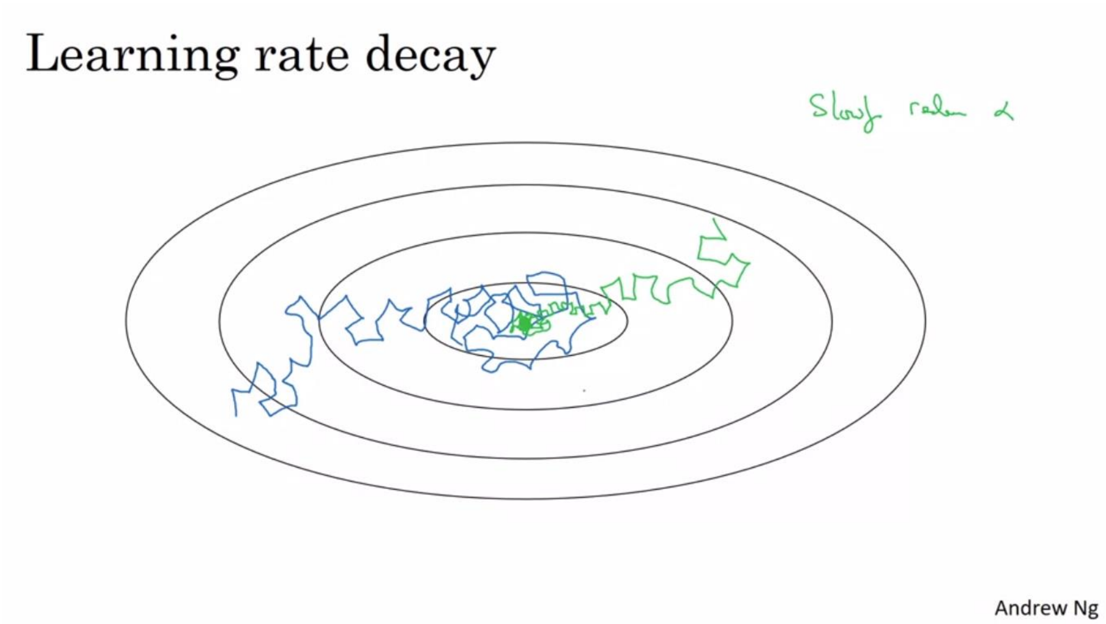
5- Adam (Adaptive Moment estimation) Optimization Algorithm:

It is a combination of two algorithms: momentum and RMSprop:



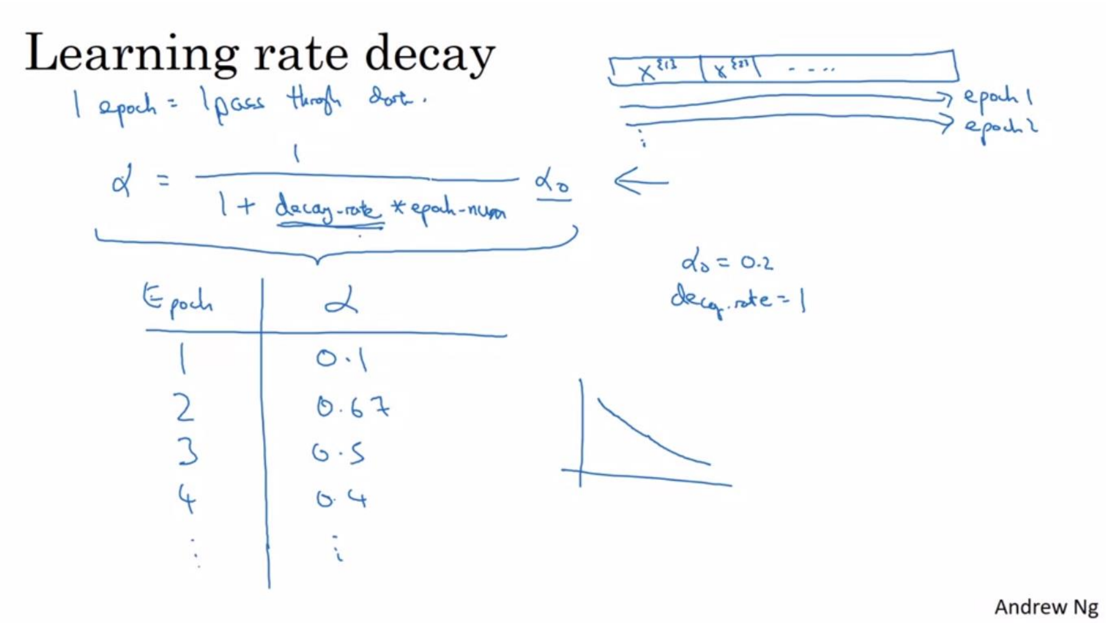


- Learning rate decay:

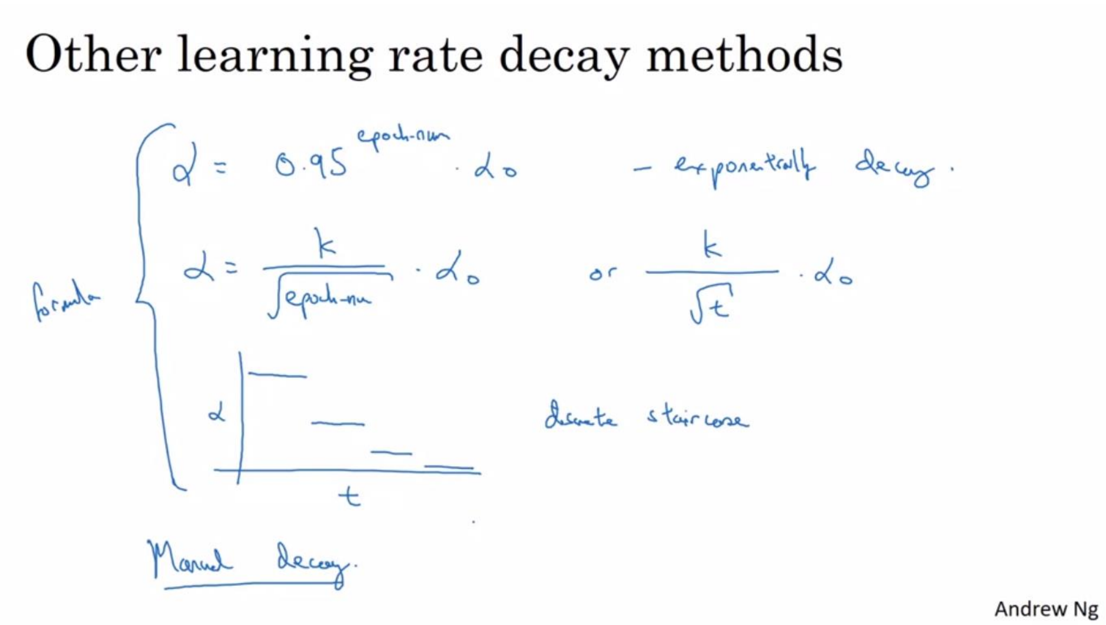


We should make the learning rate begin large then decay when approaching the solution, but slowly.

Method 1:

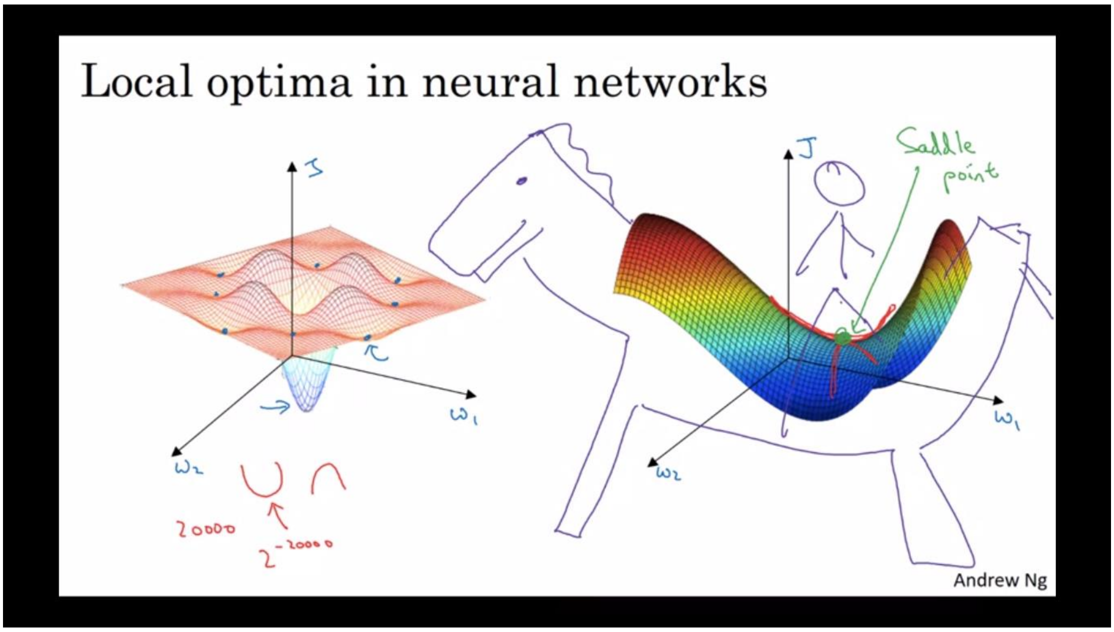


Other Methods:

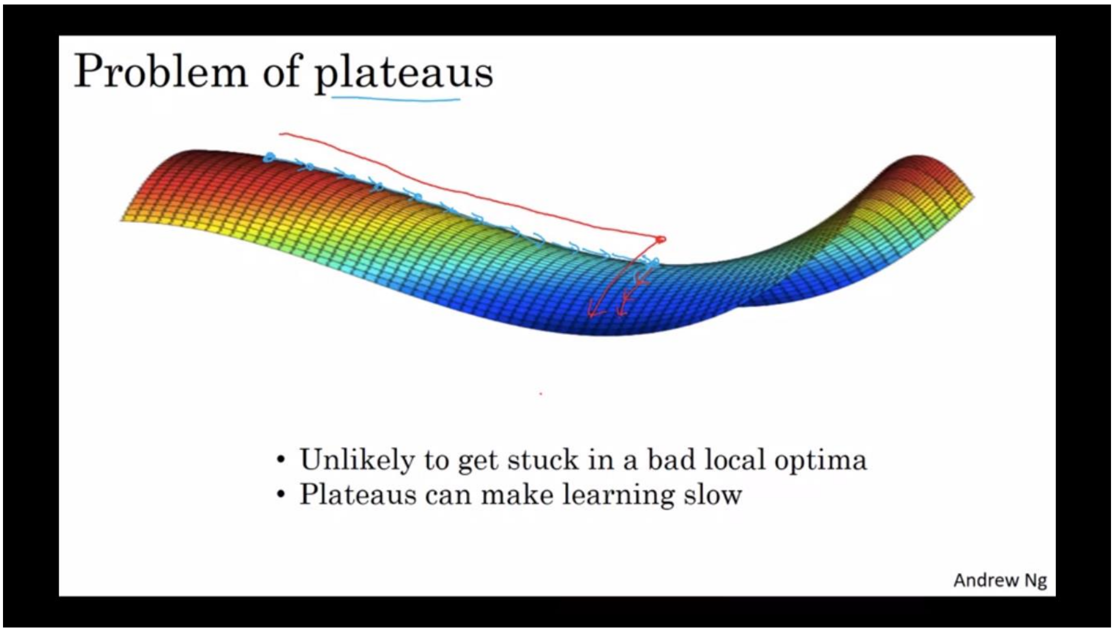


- The problem of local minimum:

We are unlikely to have a problem with local minimum in large data set. However, we have a problem in the slow of the gradient. So, the aforementioned optimization algorithms will make gradient descent faster.



Local optima in neural networks
J
w1
w2
Saddle point
20000
2^-20000
Andrew Ng

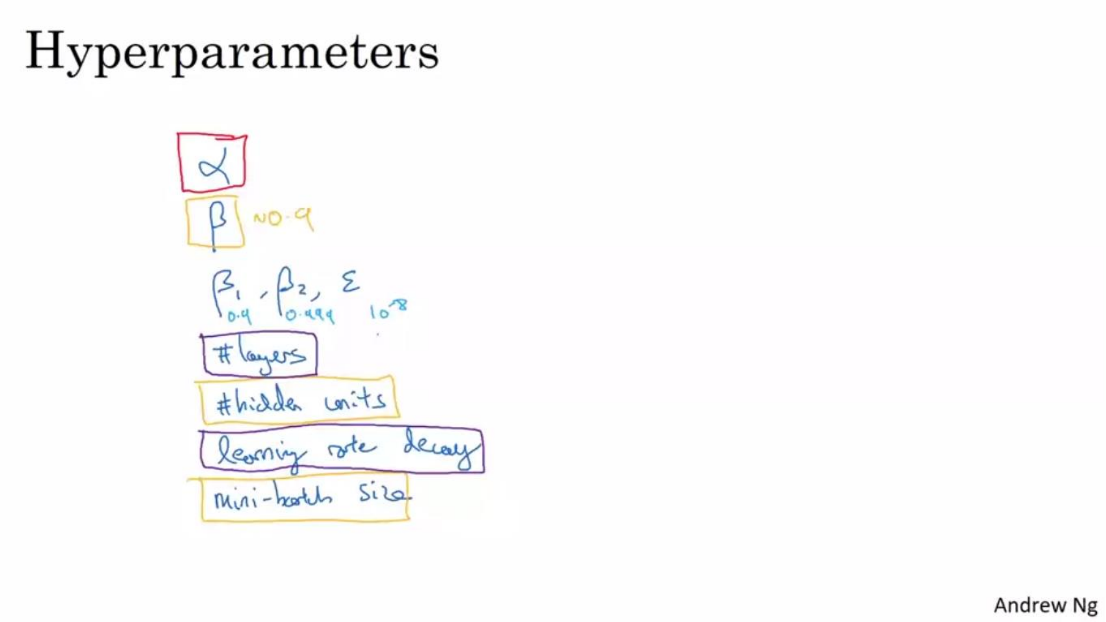


Problem of plateaus
• Unlikely to get stuck in a bad local optima
• Plateaus can make learning slow
Andrew Ng

Week 3 summary

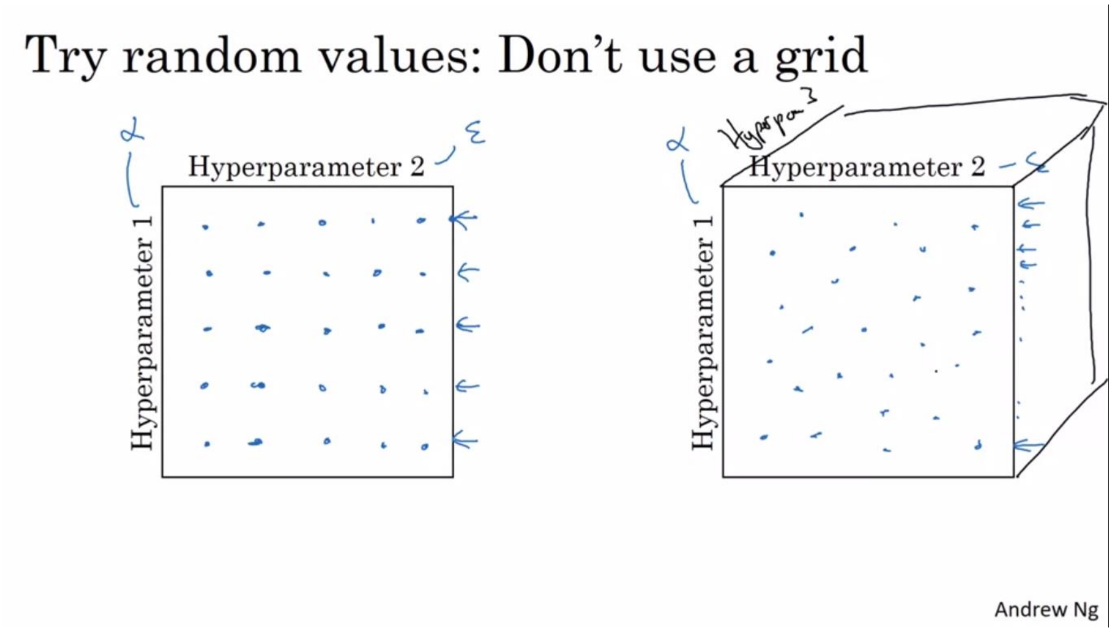
1) Hybrid parameters tuning:

The updated hybrid parameters so far are:

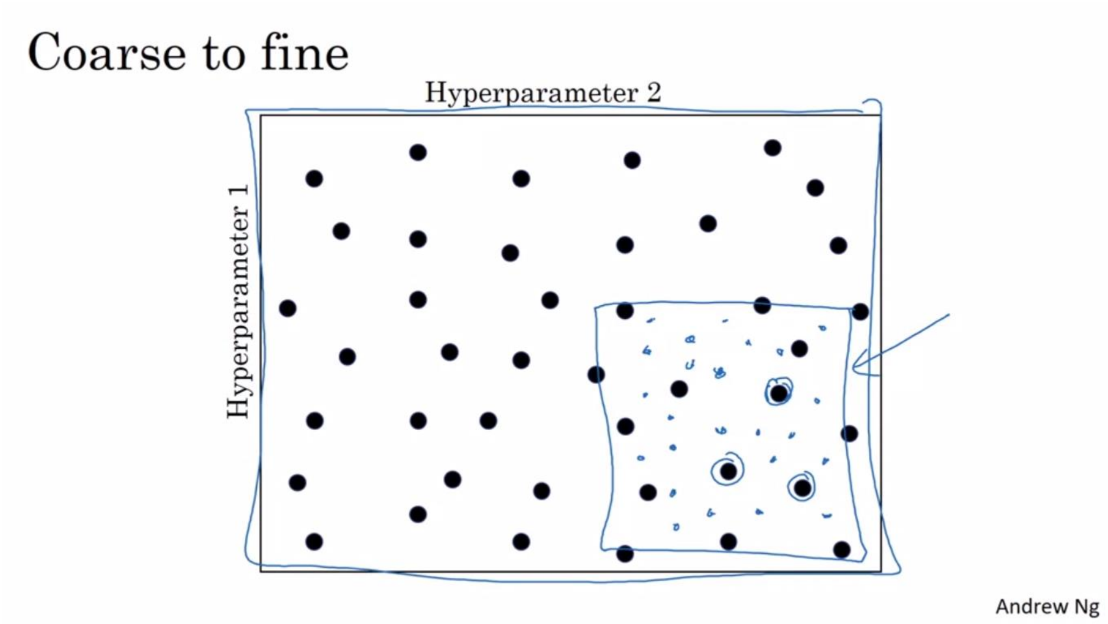


Red colored is the most important and must be tuned well. Yellow ones are less importance. Finally, burble colored are the least parameters.

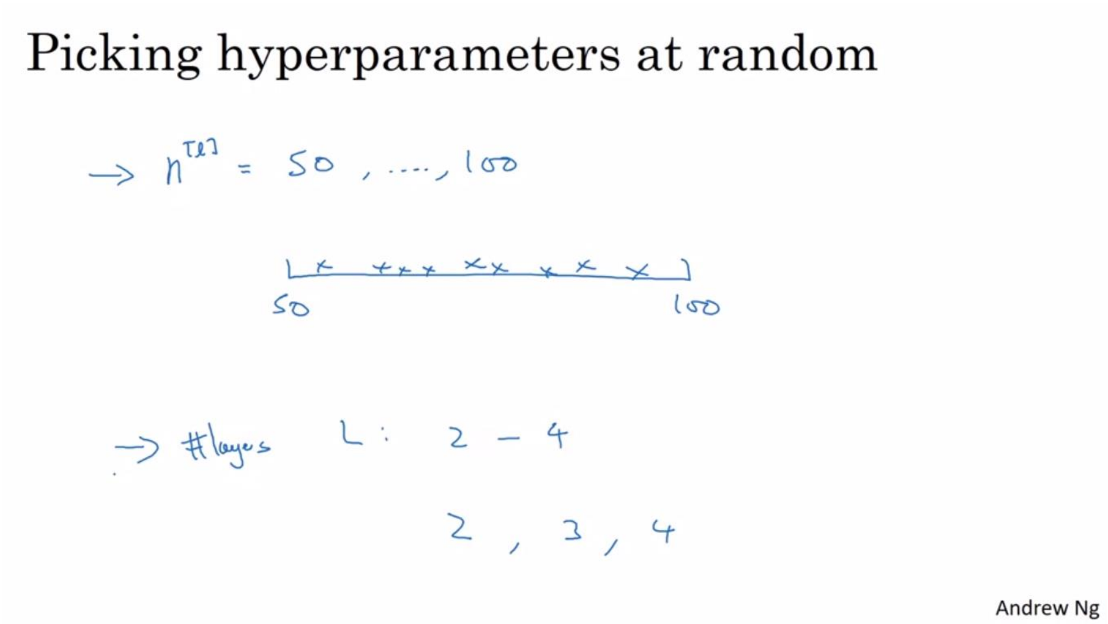
Try random values:



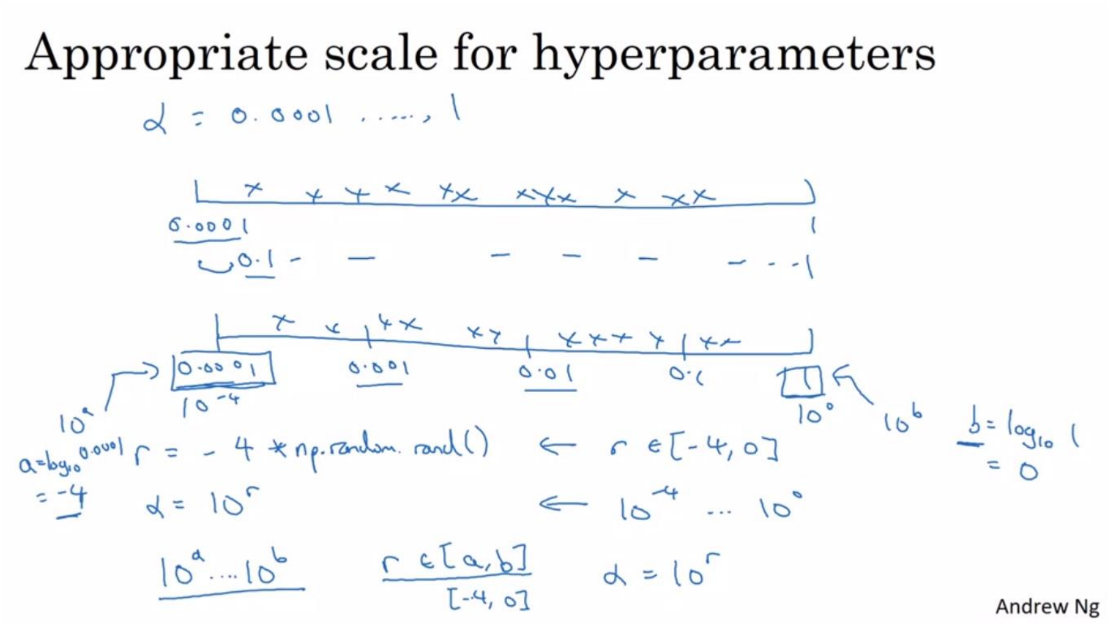
Try to fine these random results by focusing on a small box containing the most expected suitable values and choose of them:



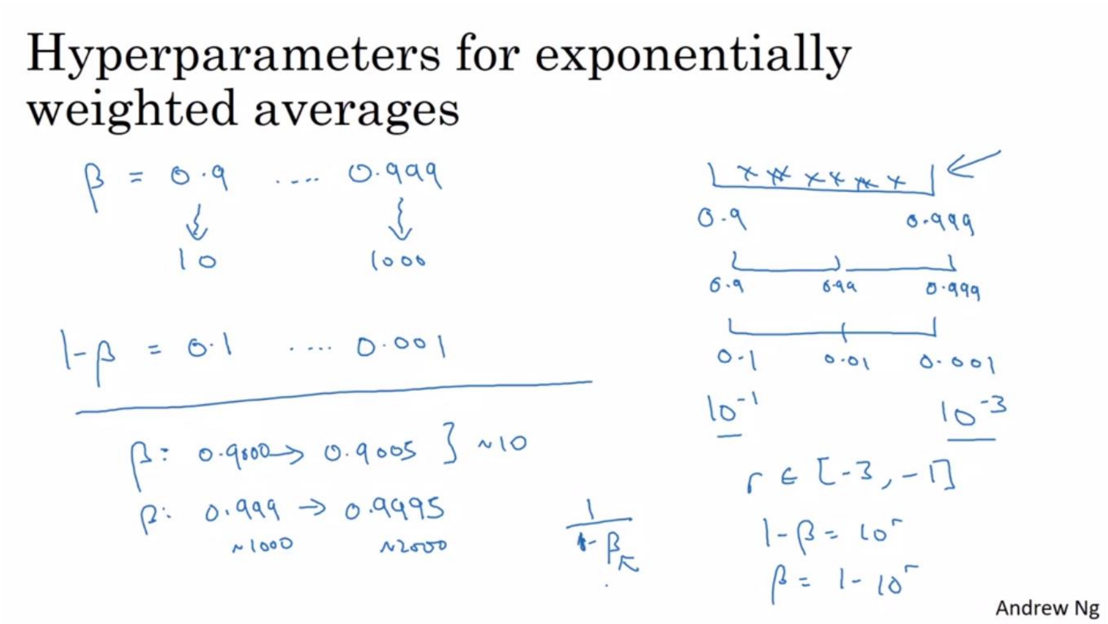
Number of layers and number of hidden units are mostly picked at uniform random:



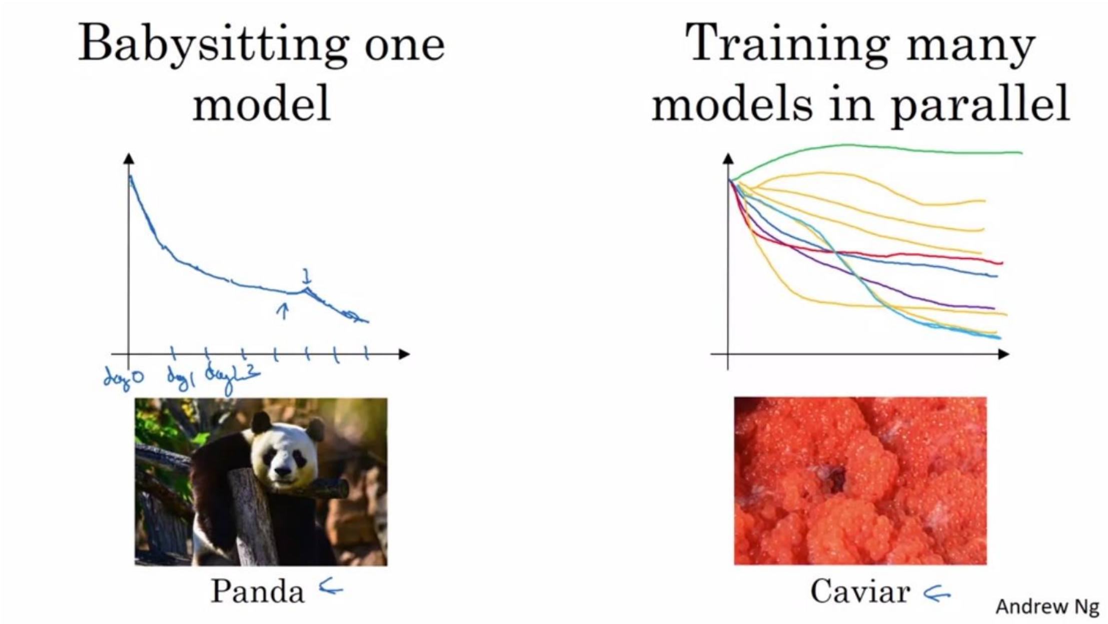
Try to use the appropriate scale so that to consider the full range. For example, log scale for learning rate α:



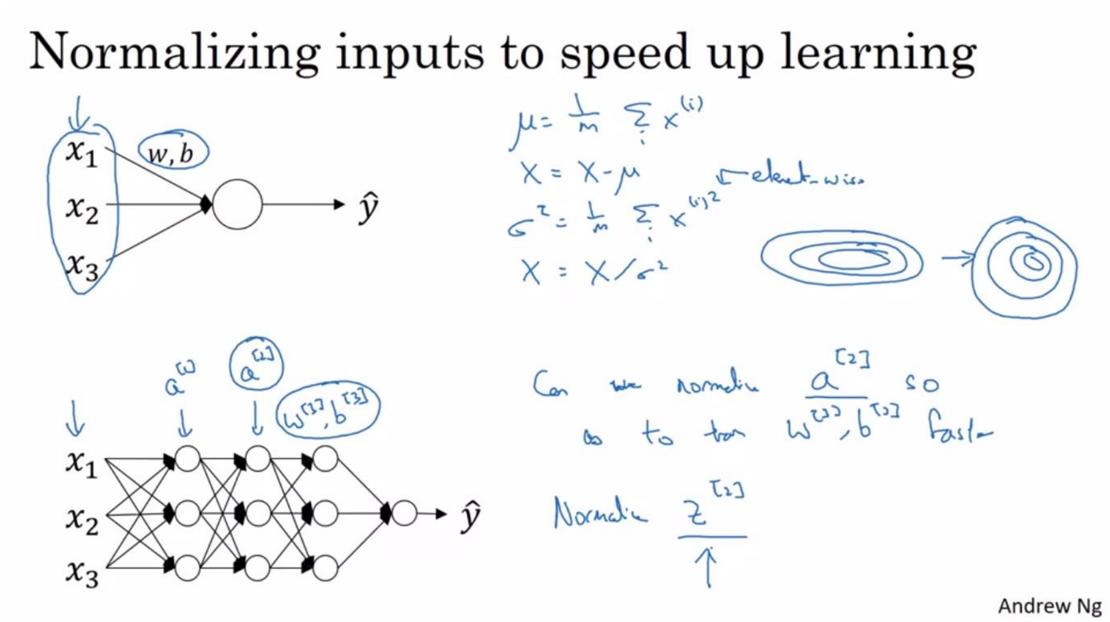
For exponentially weighted average, we also use log scale as in α. As β near 1 and 1-β is near 0:

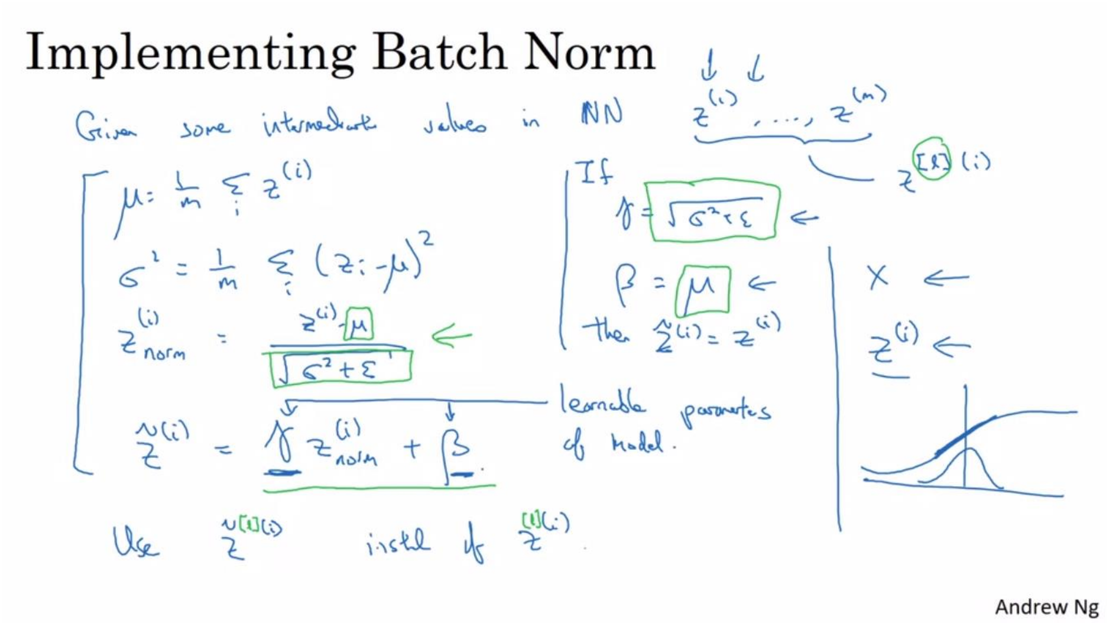


Try to use different models if you have enough resources, like computers:

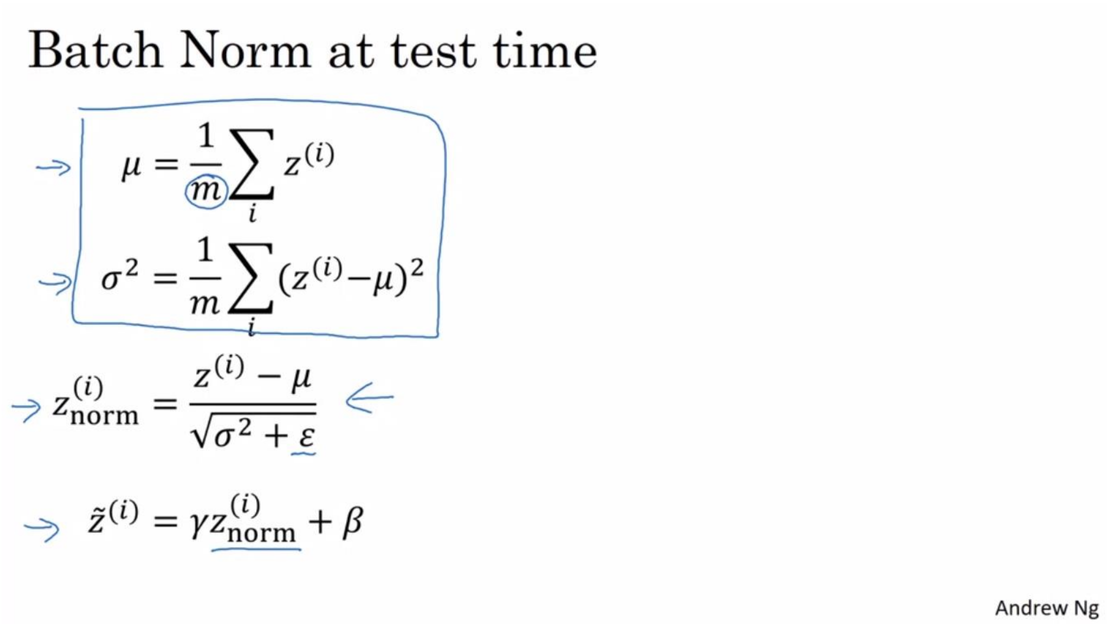


2) Batch Normalization (BN):
- Normalize activation functions:
  This is to faster the W and b computations.
  It is better to normalize Z before going to the activation function.

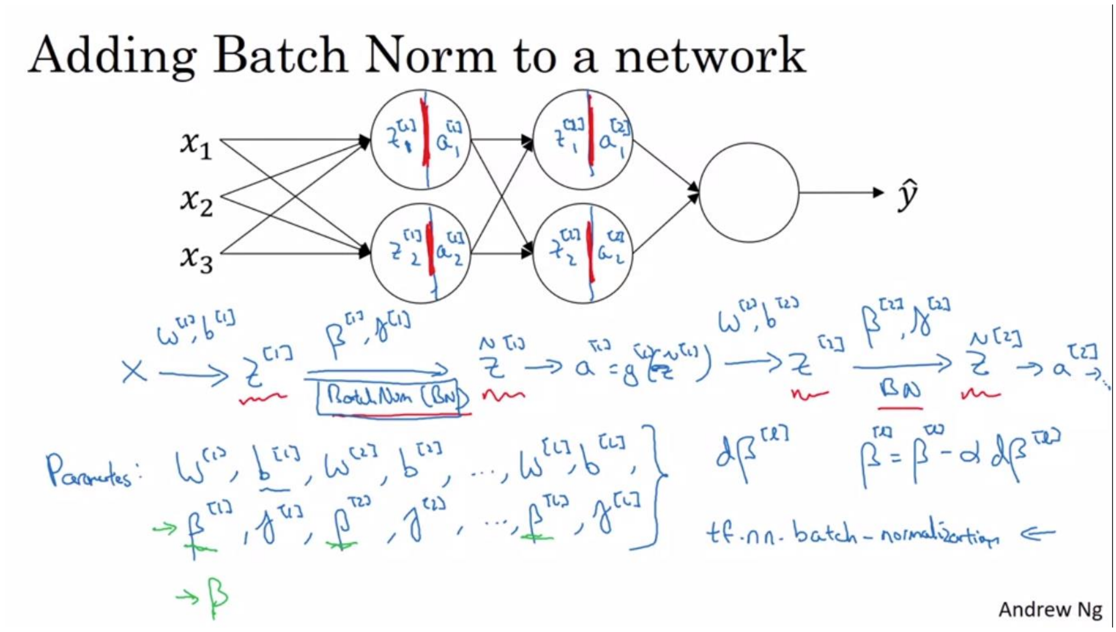




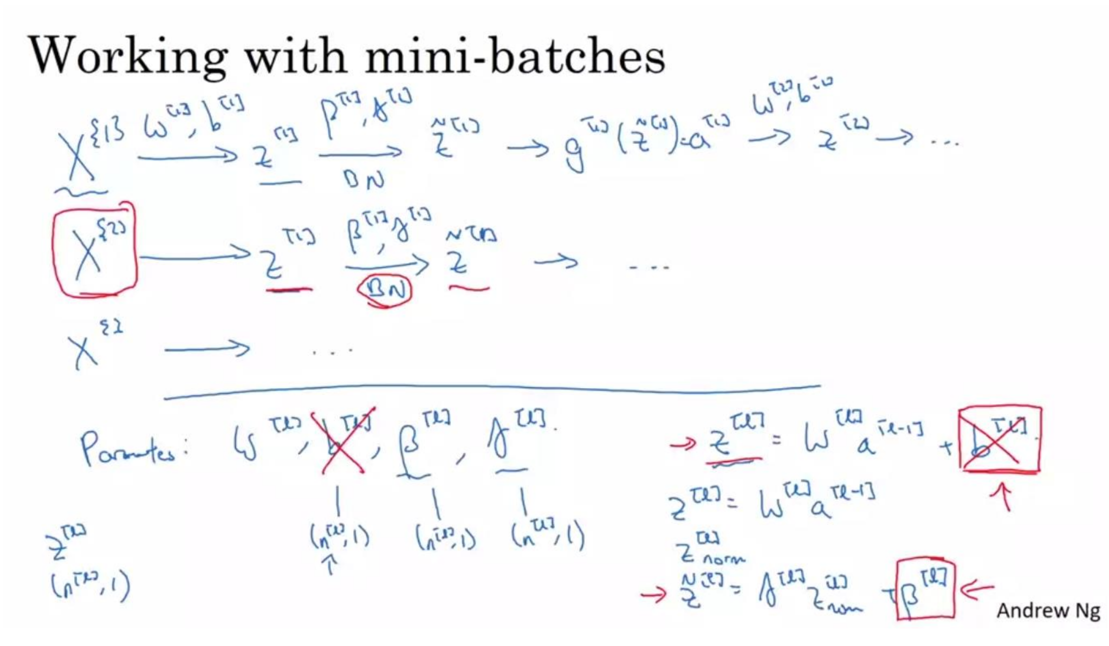
So, the mean and variance could be any values rather than 0 and 1, respectively.



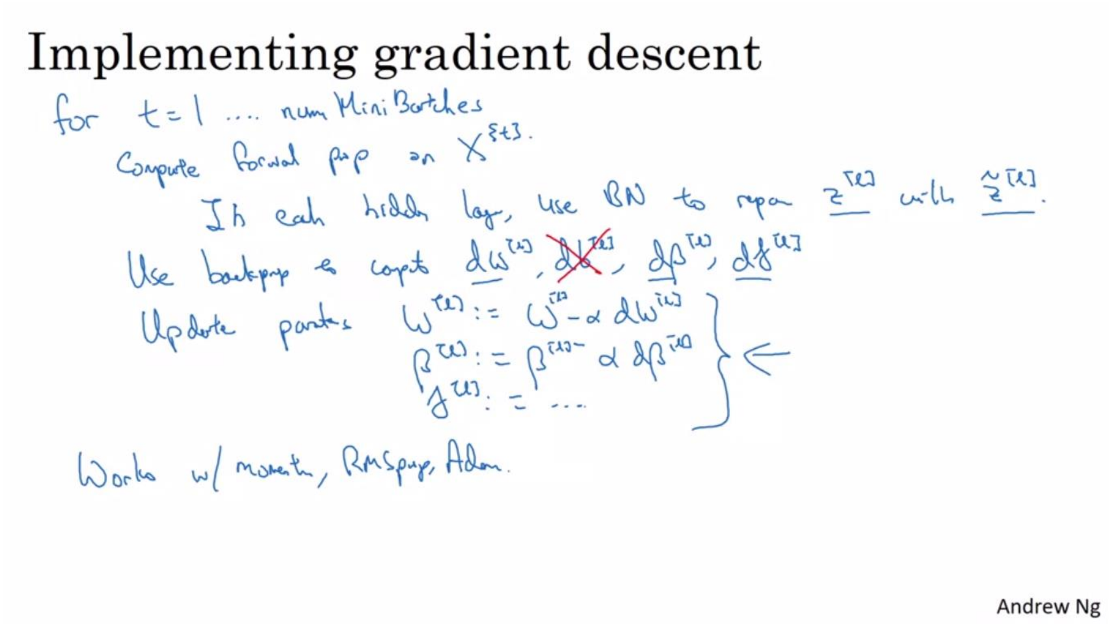
Updated parameters: W, b, β and γ where last two are for batch normalization:



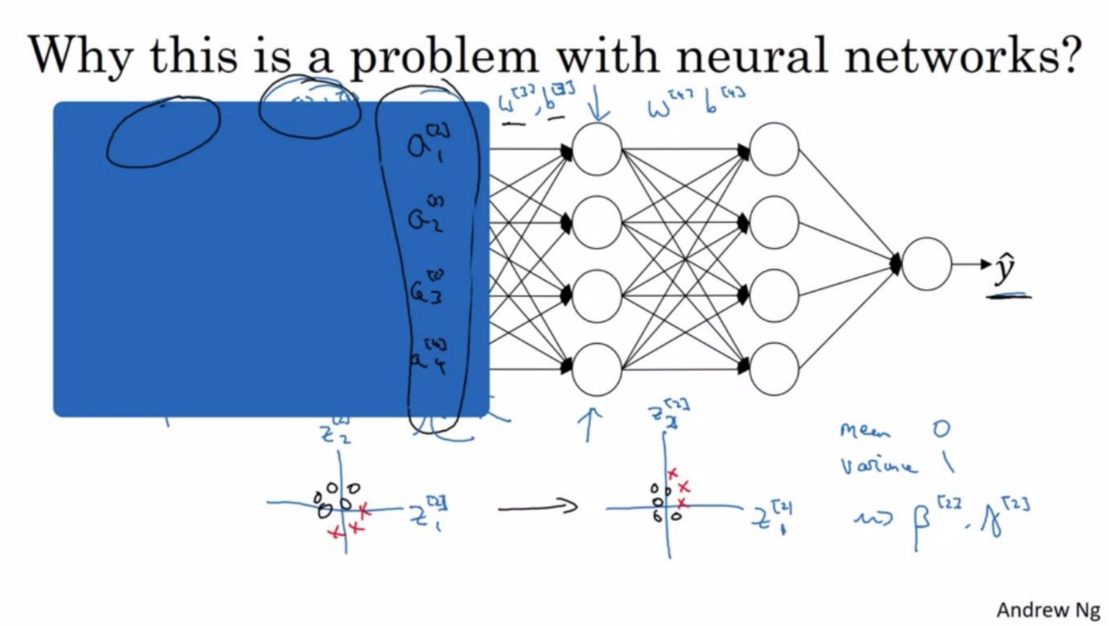
We don’t need to add parameter b, as it will be subtracted in the mean subtraction step in normalization. We instead use β and β & γ should be the same dimension as eliminated b to reserve Z dimension:



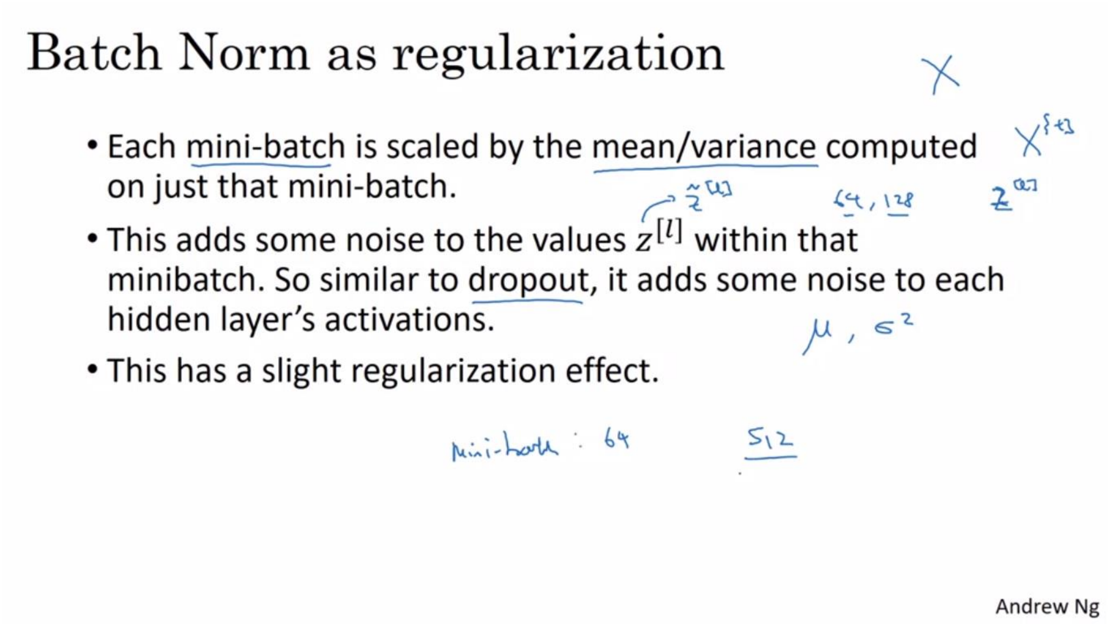
Implementation of BN on Gradient Descent summary:



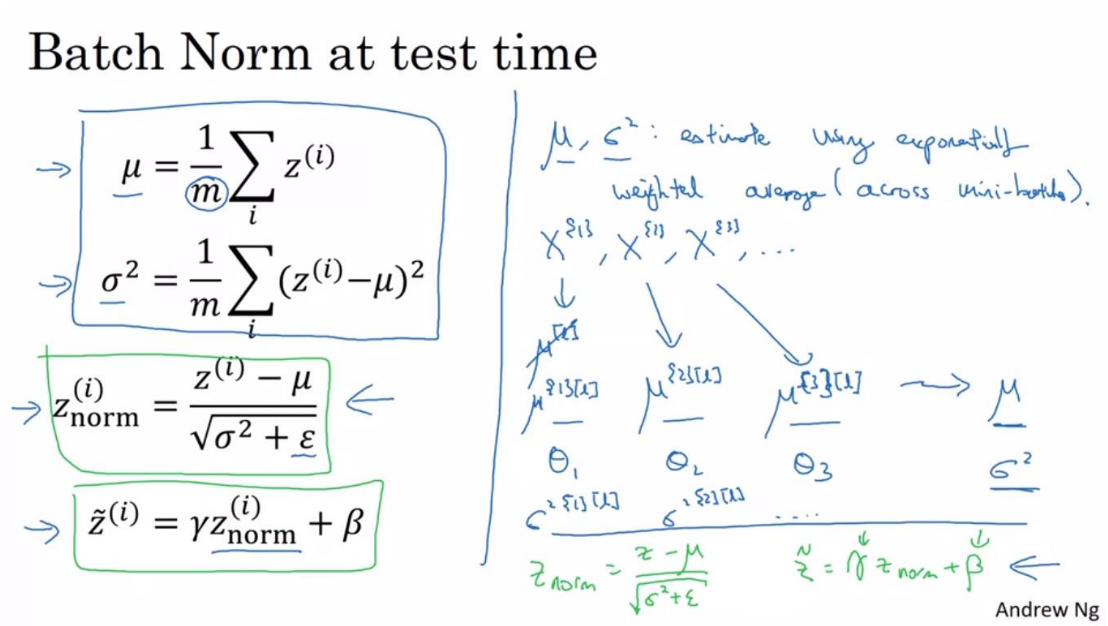
The BN reduces the effect of changing the hidden layer (i) values on layer (i+1) values by making the mean and variance almost the same for all hidden layers:



BN adds noise as the dropout regularization does. To reduce this effect, use large sized mini batches:

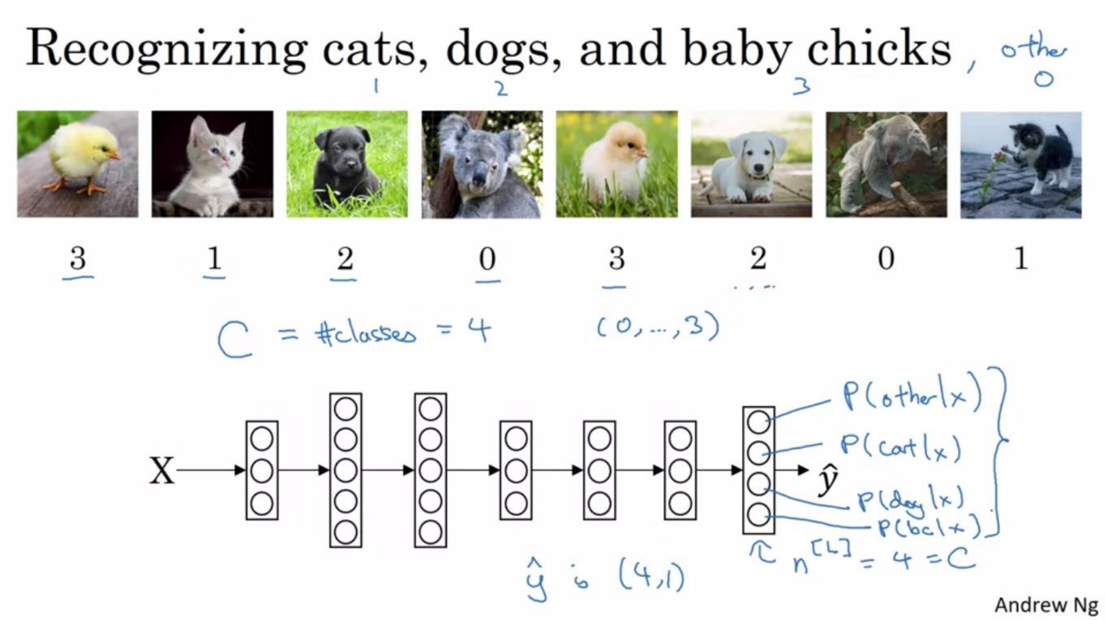


At test time, we shall use exponentially average to calculate the mean and variance:

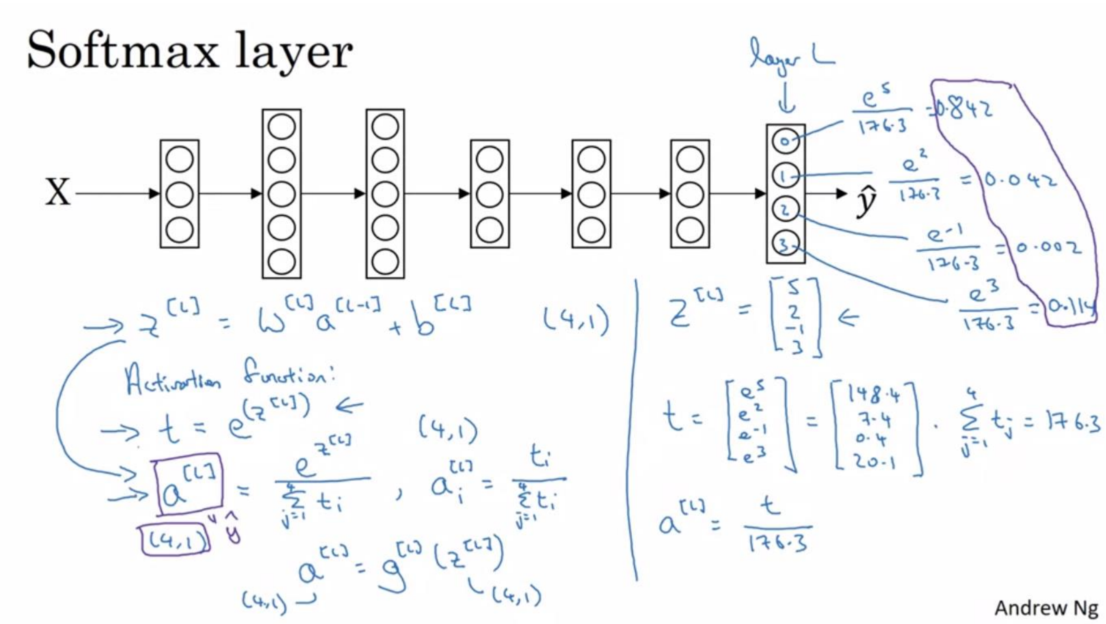


3) Multi-class Classification:

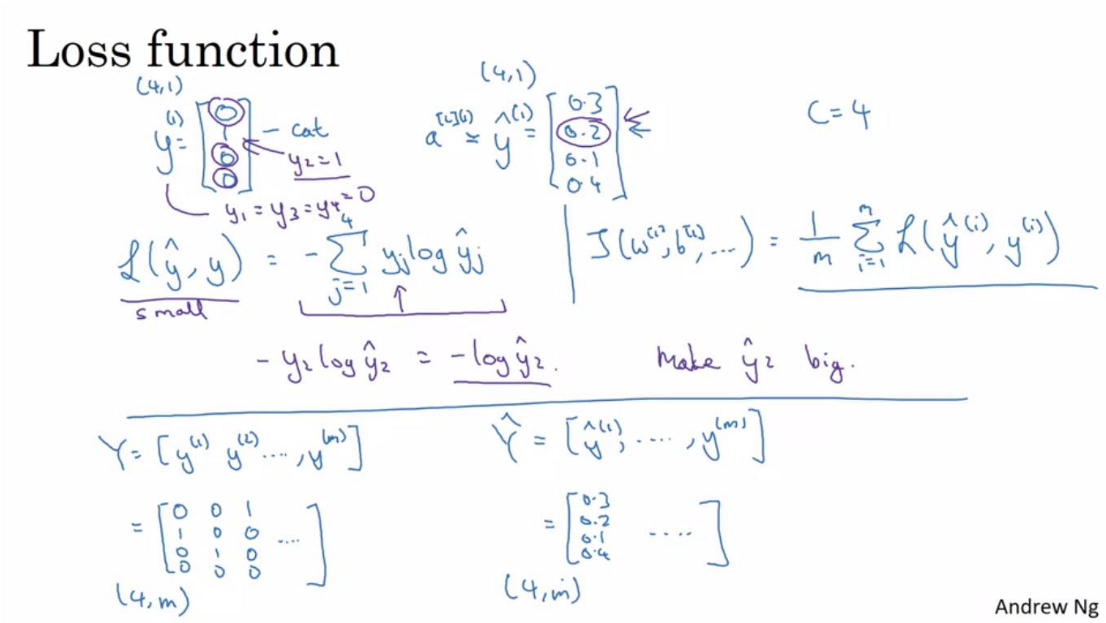
Rather than classifying 0-1 task (like is it a cat or not), we will classify multi things (like cat, dog or horse). C is the number of classes starting from class 0 to class c-1.



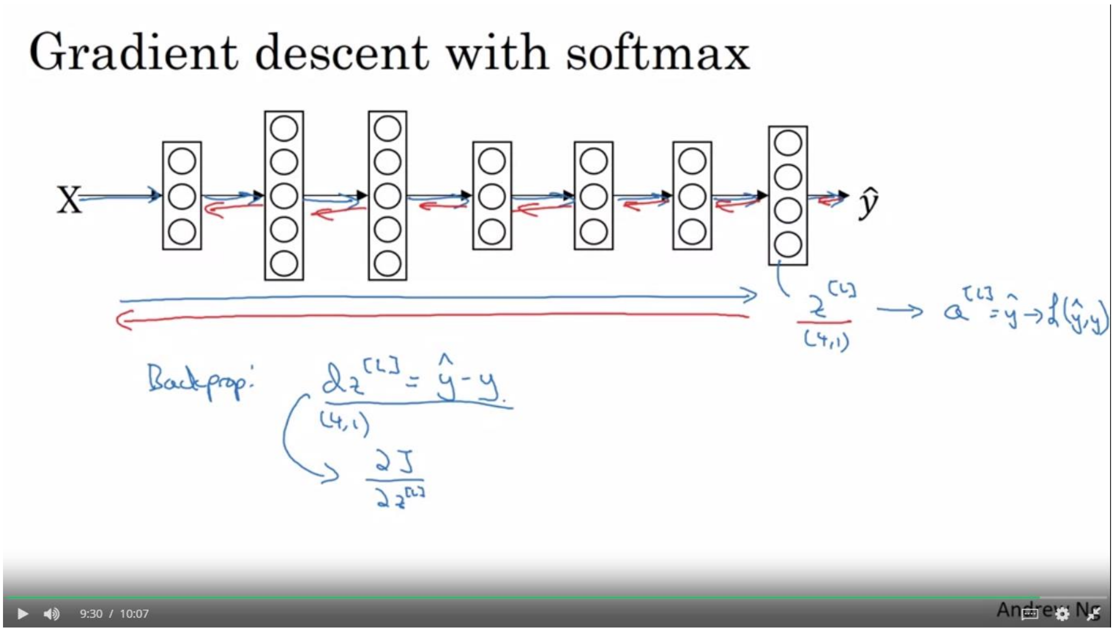
For this task, the output layer is called Softmax Layer. This layer has exponential activation function:



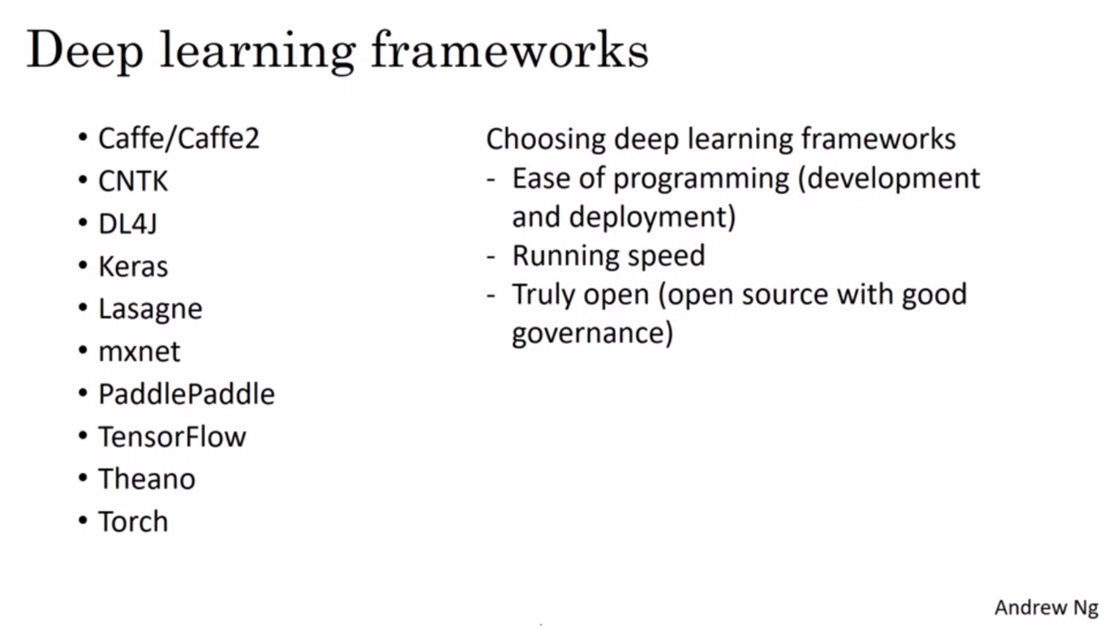
To train this NN, we need to make the specific output y^ very large in order to make the loss in it very small. Also, we need to train over m examples:



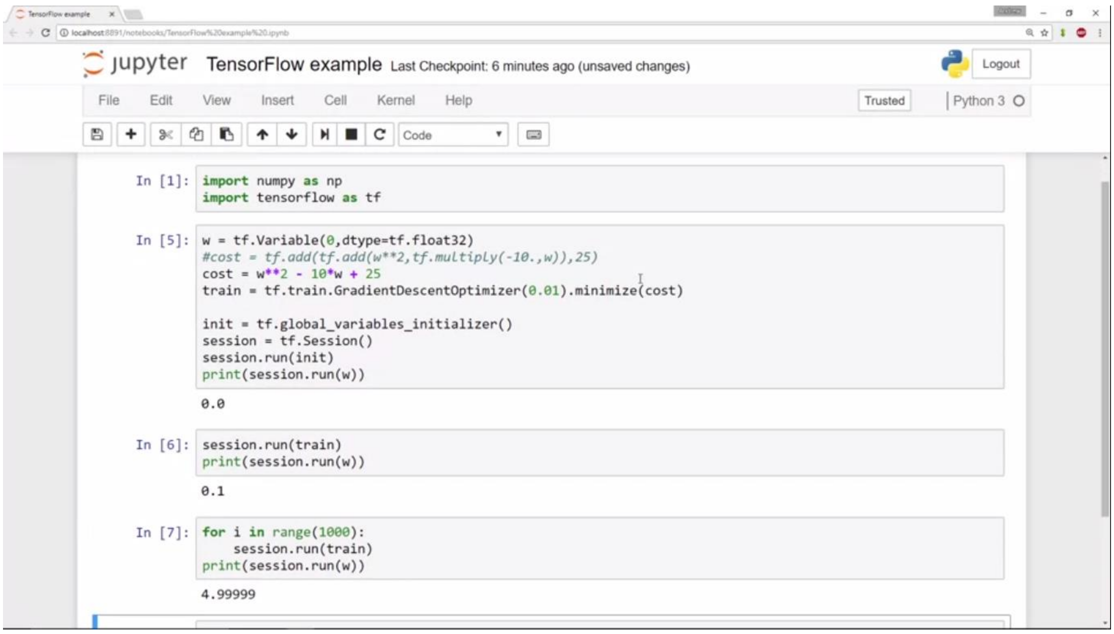
Also do the back propagation:



4) Programming frameworks:

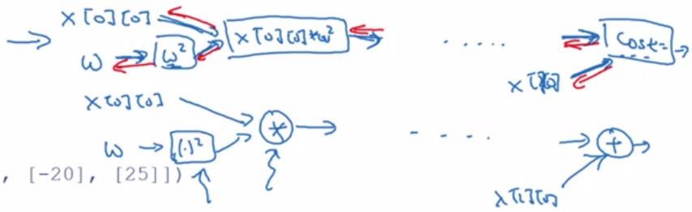


TensorFlow:



- Backpropagation is built-in in tesorflow (no need to write it)

Code example



```
import numpy as np
import tensorflow as tf

coefficients = np.array([[1], [-20], [25]])

w = tf.Variable([0],dtype=tf.float32)
x = tf.placeholder(tf.float32, [3,1])
cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]    # (w-5)**2
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
init = tf.global_variables_initializer()
session = tf.Session()
session.run(init)
print(session.run(w))

for i in range(1000):
     session.run(train, feed_dict={x:coefficients})
print(session.run(w))
```

```
with tf.Session() as session:
    session.run(init)
    print(session.run(w))
```

Andrew Ng