Tips for DNN

**Summery first if you just want to know what you can do but not why we can do like this:**

1. **When performance on training data is bad.**
2. **Choose new activation function**

**Try sigmoid, tanh, relu an so on**

1. **Adaptive learning rate**

**Used to use adagrad, but another method to get adaptive learning rate can be used named “RMSProp”**

1. **When the performance on training data is good, but not satisfactory on test data:**
2. **Early stopping**

**e.g. your epoch is 1000, but you find after 600, the accuracy on test data is getting worse, then stop at 600, and set the epoch = 600 [Attention: here test data is not real test data, but more like validation data that you split it from the original training dataset]**

1. **Regularization**
2. **Dropout(!!!)**

**Check dropout function in**

**TensorFlow: keep\_prob = tf.placeholder(tf.float32)**

**h\_fc1\_drop = tf.nn.dropout(h\_fc1, keep\_prob))**

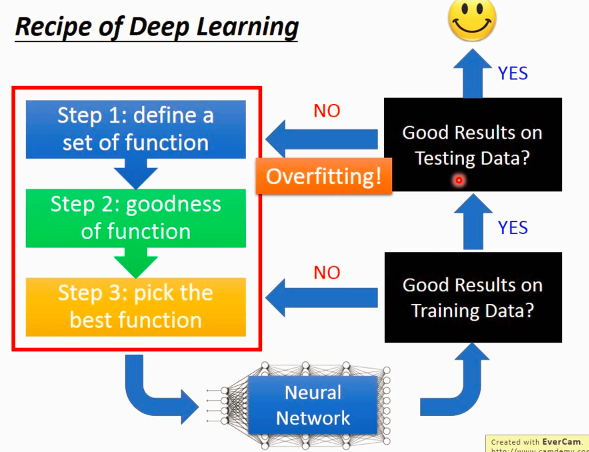
**Keras: model.add(Dropout(0.25))**

**(you can dropout both the input data and hidden layer neurals)**

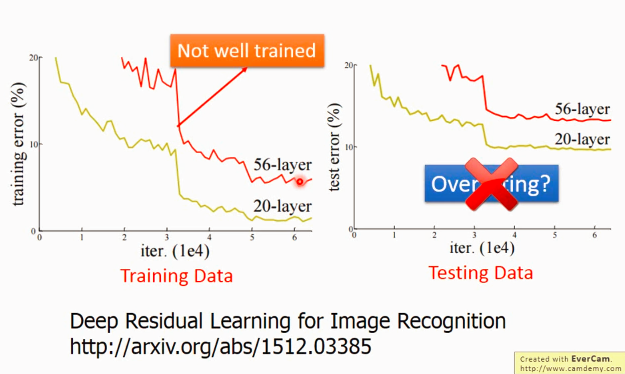
When we evaluate the goodness of the network, we first check the accuracy on training data, if it’s even not good on training data, you can check from step1-3 to see if there’s a problem. It’s definitely not the problem of “over-fitting”. Unlike k-nearest neighbors or decision tree which the accurate on training data will be very good, the result of deep learning method may not be good even on training data.

Then, when the result on training data is good, test the network on test data. If the result is bad, it’s may be overfitting then.

And again, after you modify the “step”, you need to test on training set first. Only when the result is better on both sets, you can output the model.



But the more layers the worse result does not necessarily mean overfitting. It could be: it stuck at local minima.



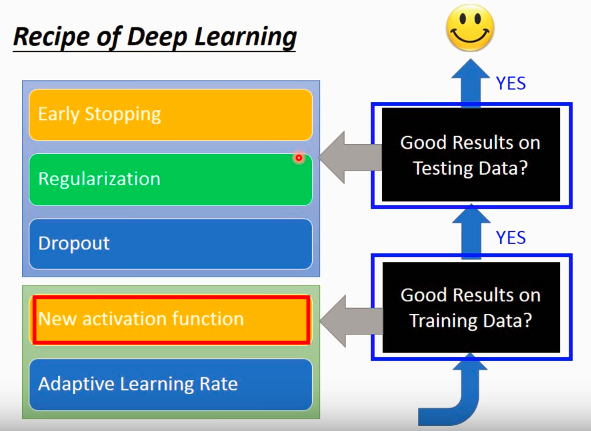
Theoretically, 56 layers network can definitely do what 20 layers network can do. But

When to use dropout?

When result on test data is not good, but on training data is good. Attention that if the result on training data is not good, forget about dropout!

In the following, dropout will be introduced with more detail.

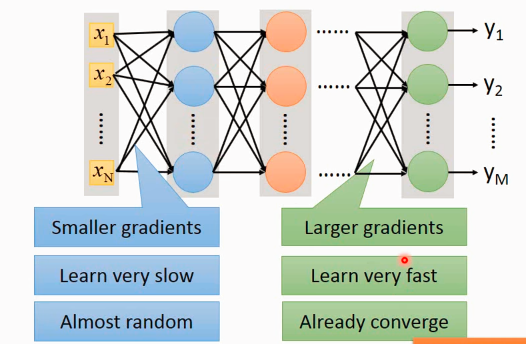
**How can we make the result better?**



Result not good on training data

1. Change activation function (sigmoid, tanh, relu …)
2. Adaptive learning rate.

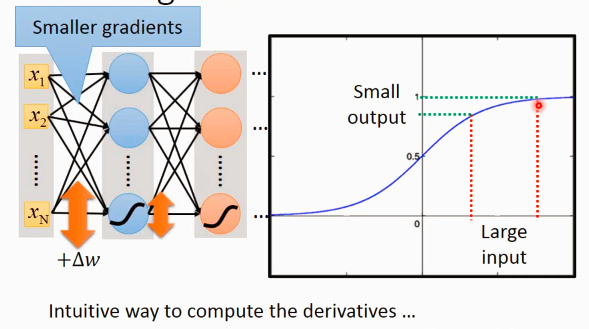
Vanishing Gradient Problem



The differentiation of loss function with respect to parameters is small near the input end, while the gradient is small near the output end.

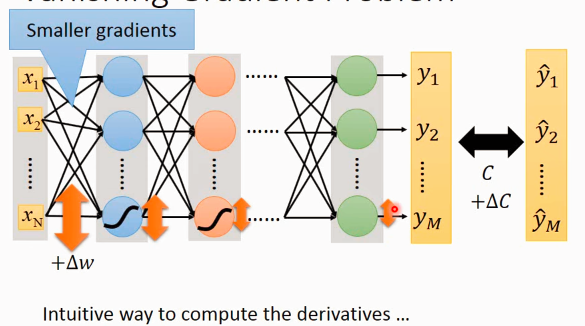
So if we set the same learning rate, it learn slow at input end, while fast at output end.

So while input is almost random, the output is almost converged. So the result is not good based on random.



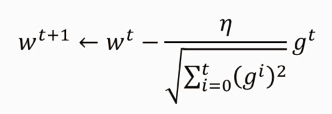
When input is big, it’s squeezed to small output using sigmoid function.

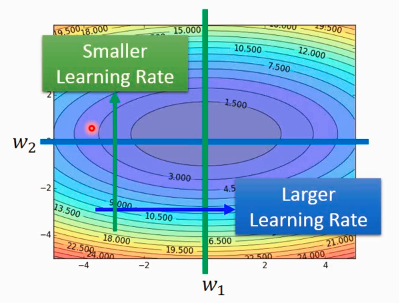
The deeper the layers, so smaller the effect of w in input end to loss function(?)



Another thing we can do to improve the result (if it is not good on training data) is to have adaptive learning rate.

We have already learnt about Adagrad.



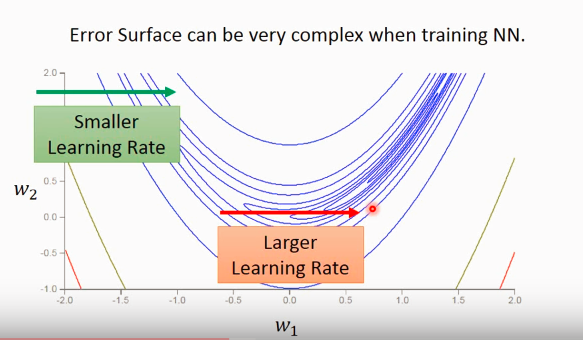
e.g. 

if the gradient of w1 on horizontal direction is small, then the learning rate is big.

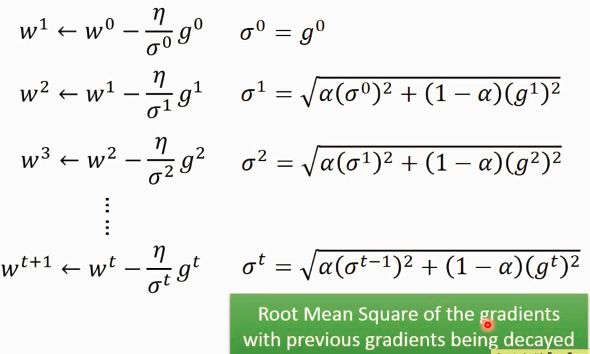
On vertical direction, it is steep, so smaller learning rate.

But this is under the condition when the shape (of loss function) is convex, but in practice, it could be any shape. (in one direction, it’s always steep or always smooth, but it does not consider the condition when it may steep and smooth along the same direction at different point)

for example:



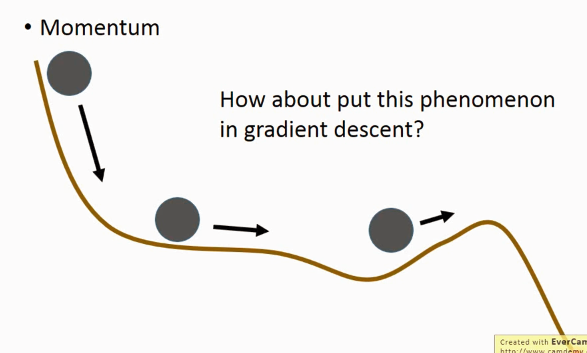
So here, we have this method **RMSProp**



Similar to Adagrad, (signa)^2 also include information of all previous information. Besides, the alpha value can be modified, when it close to 0, it means I trust more about last/new gradient than old ones.

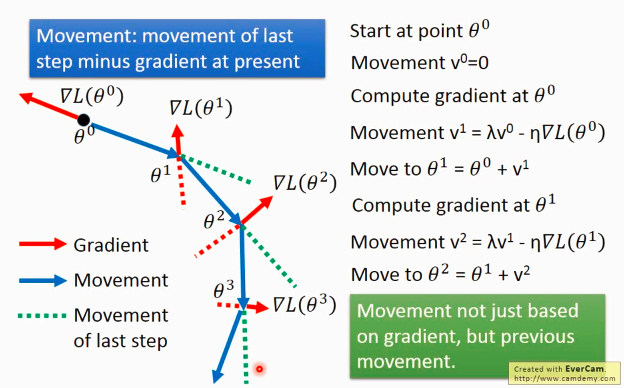
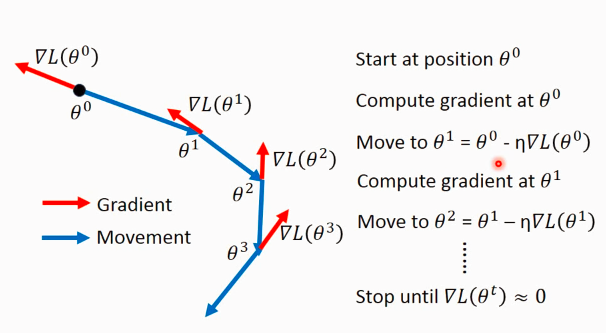
Another problem is that the optimization process may stuck at local minima or saddle point. But someone said, you don't need to worry much about the local minima problem, because, local minima means every gradient is minima at this point, if we have 1000 parameter, it is very unlikely to have a point to satisfy this condition. (emm.. I think it make sense..)

To solve the stuck problem in saddle area or local minima, let us see an example in real world:

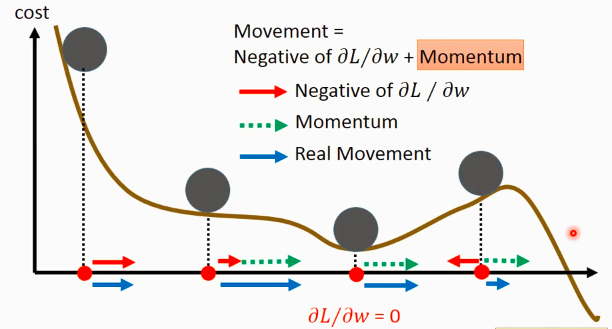


Due momentum, the ball can still move in saddle area or even though the local minima.

Lets use this idea in our case:



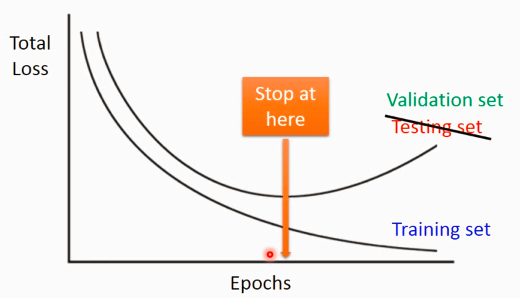
It does not gurantee to reach the local minima, but give some hope( hope is important in life!!!)



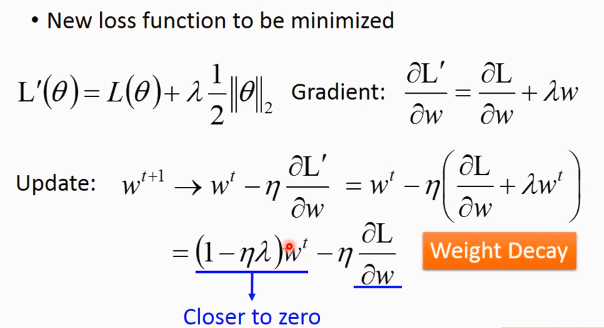
Till now, something you can do to improve the result on training data is given.

Then, if the result is good on training data, but not good on test data, let’s see what we can do then.

1. Early stopping

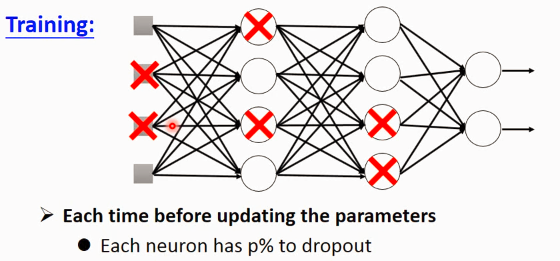


1. Regularization

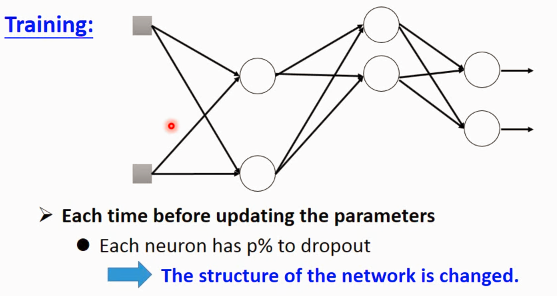


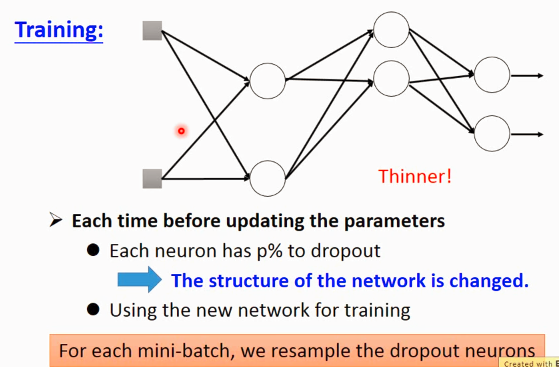
We don't want w to be too far from 0.(decrease the epoch may have the same function as regularization that we don't want to it be too far away from 0)

1. Dropout (important)

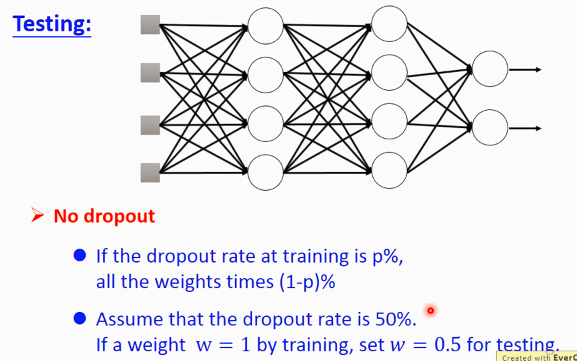


After drop those neural, the corresponding weight are dropped too.

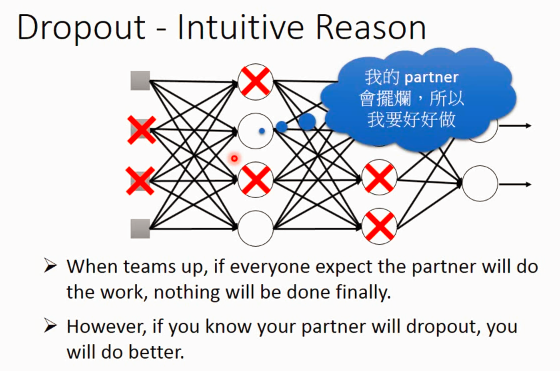




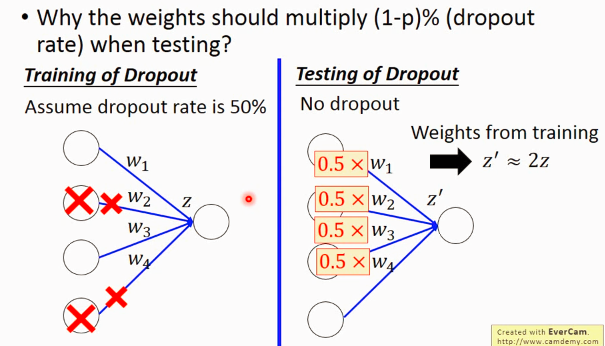
It means: we divide the whole training data into small batches, if we use dropout, for different batches, different neural are trained.



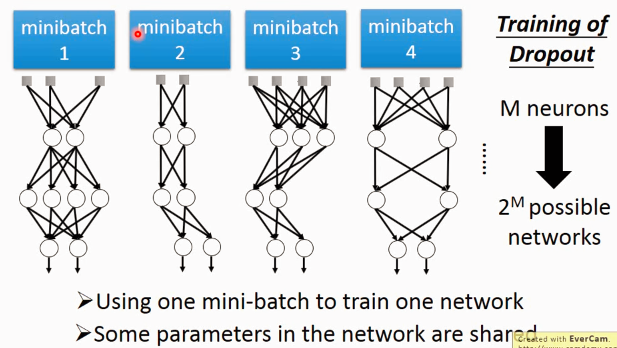
Why dropout help to improve the result on test data? Prof. Lee give a interesting and understandable example.



Why change the learnt wright on test?

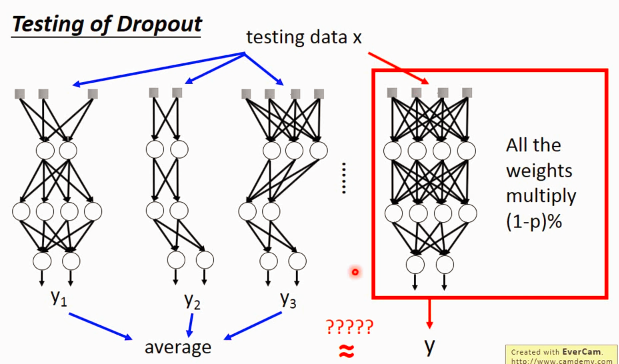


Dropout is a kind of ensemble.

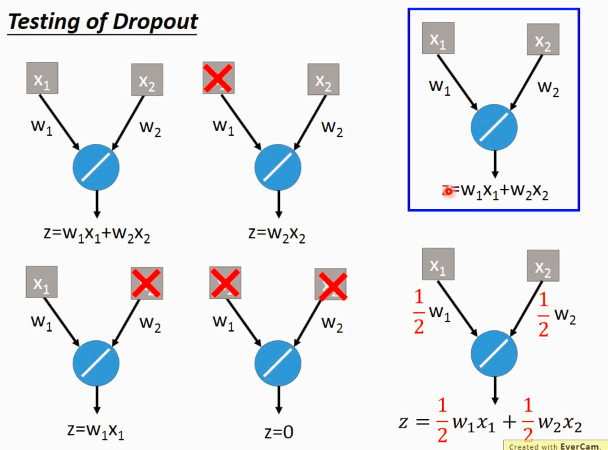


Although each network is trained by only one batch, but one weight/neural is trained by many batches ( because it may appear in different network)

Theotically, to get the result on test data, we need to put it in different networks, and average the result to get the final result. But if we have thousands of networks, it is crazy. So what we do is to use the weight of the thinned weight, and get the result on the network without dropout, but multiply 0.5 on weight.



Here’s a little example to show it works:



But yes, it’s not always totally equivalent like this, but it works.

If the network close to linear, the performance of dropout would be good.

e.g. if use relu, maxout, the performance is better!