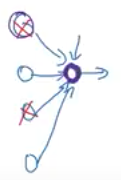
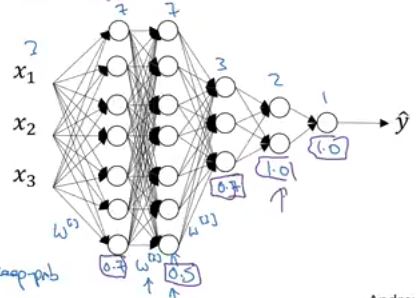
**Lec 48: Why does Droput work?**

Consider the following network with a single unit having 4 inputs:



While performing dropout (0.5) any two units of the previous layer can be dropped. Hence it is not advisable to have extremely high weight for any one particular feature. In this case, one should spread out the weights to all the units. This shrinks the squared norm of the weights. And similar to L2 regularization it prevents overfitting.

For the following network it is also possible to vary the ‘keep\_prob’ to different values for different layers:



The above is used when one is worried that some layers would overfit more than others. Consider the 2 layer having 7 hidden units, whose weights matrix is 7x7. This layer is likely to overfit, hence one can apply more dropout to this layer and apply a lower dropout to the other layers.

One problem with having different dropouts for different layers is that, there are many more hyperparameters to tune.

Dropouts are usually used in Computer vision more than other disciplines.

The downside of having dropouts is that the cost function ‘J’ is no longer a simple function which can be plotted to visualize gradient descent. This is because new terms get added for different dropouts used in each layer, which increases the hyperparameters to the function. This makes it difficult to visualize the gradient descent. One thing that can be done is to first turn off dropouts for every layer and visualize the cost function graph. If it is having a decreasing tendency, then turn on the dropout.

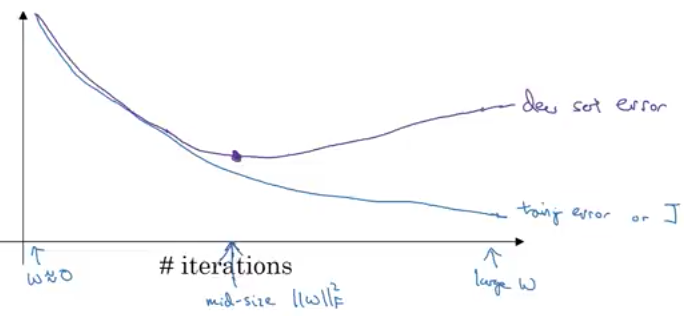
**Lec 49: Other Regularization Methods**

Apart from L2 regularization and dropouts there are many other techniques of regularization.

One technique is to get more data. But if getting data is expensive, then one can augment them. This greatly should reduce overfitting because now there are many more variations to the same data.



Early stopping is another regularization method to prevent overfitting.



Initially while training the weights are less and as you train they increase. Beyond a certain point they are so high that validation error increases. One must find the optimal point to stop iterating and save the model at that point. At that point the weights would be mid-sized hence reducing overfitting.

Early stopping has one downside:

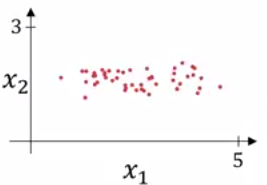
Usually while training a network the first task is to optimize the cost function (J) and then work towards avoiding overfitting using regularization methods. While using early stopping this essence is lost, because you have to perform both cost minimization and overfitting avoidance together. As a result none of the two are optimally achieved.

The other way around this is to use L2 regularization where one has to try various values to get an optimal solution. This is computationally expensive though. In early stopping one advantage is getting the optimal set of weights in one descent without having to try various values.

**Lec 50: Normalizing Input**

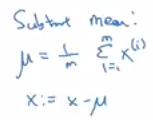
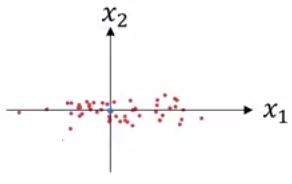
Normalizing input speeds up training.

Consider the following dataset having two features x1 and x2:



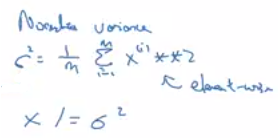
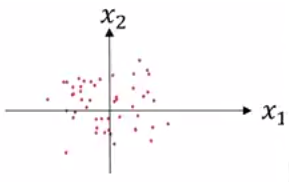
There are two steps to normalization:

***Mean subtraction:***

🡪 

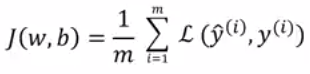
***Variance normalization***

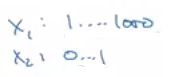
Here x1 has a lot lesser variance than x2.

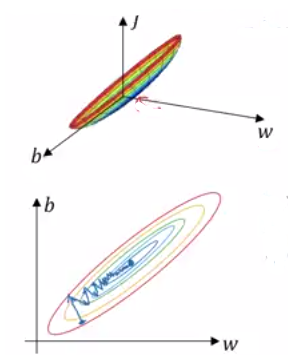
🡪 

***Note:*** use the **same** values to normalize the test set as well.

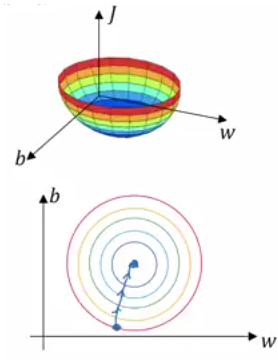
***But why normalize the input? (An intuition)***

Consider the cost function: ,

For a training set having features in the range: , they are on different scales. Hence their corresponding weights are also on different scales. The cost function ends up looking like a skewed bowl:



On the other hand, if the features are all normalized to same scale, then the weights are also on the similar scale. The cost function ends looking like this:

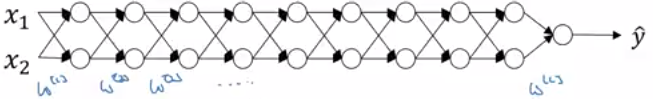


When normalized the training process can take much larger steps to converge to the minimum rather than which is not normalized, where longer time and smaller steps need to be taken.

**Lec 51: Vanishing/Exploding Gradients**

While training a network the derivatives can get very big or very small (exponential range). This makes training difficult.

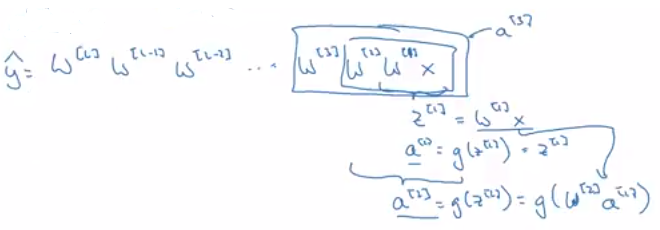
Consider we have a deep neural network, with weights w1, w2, …. Wl.



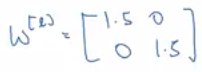
Assuming bias to be none the output is:



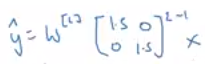
The following is how the activations and weights are for each subsequent layer:



Now let us assume that the weights for each layer is more than the identity matrix:



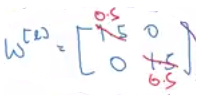
The predicted output would be:



So now the value of predicted (y) will be: 

Now if it is a deep network with many layers, the resultant would be a very large value.

Conversely if the weight matrix is less than identity:



So now the value of predicted (y) will be: . This means that the activations and the gradients would decrease exponentially for a network with many layers.

**Lec 52: Weight Initialization for deep networks**

The right initialization of weights helps avoid vanishing/exploding gradients only to a certain extent, but still helps a lot.

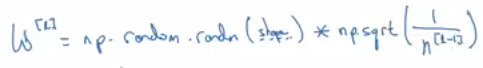
For a neural network consider:



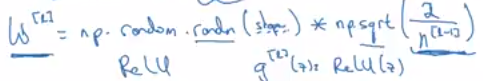
In the above notice one thing: Larger the number of layers 🡪 smaller should be the weights

This will reduce the value of ‘z’ above since it adds up all the weights. We want it to be minimum as possible.

One possible thing is to assign the variance of the weights to be: , where n is number of input features. So in python you set the weights to:



So if the input features or activations are of ‘zero’ mean and unit variance, this will cause ‘z’ to take a similar scale. This reduces gradient explosion/vanishing.

For ‘ReLU’ activation: 

For ‘tanh’ activation: 

This variance can also be another hyperparameter that can be tuned. But compared to other parameters these are not so important.

**Lec 53: Numerical Approximation of Gradients**

While implementing backpropagation there is test called ‘gradient checking’ to make sure it is correct. Before that we will see how to approximate gradients numerically.