**Lec 42: Train/Dev/Test sets**

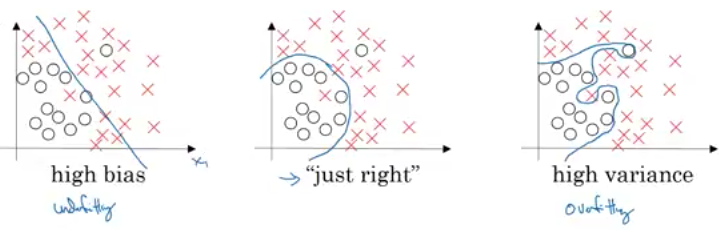
Applied ML is highly iterative is practice. This is because it is almost impossible to guess what the parameters would be in the first attempt. Hence researchers try out various parameters until they arrive at a good performance model.

In previous years, train and validation sets used to be 70-30 split. But in the modern era with a lot of training set available, it is enough to have a less number of samples for dev set and test set.

Another is problem is mismatched train/test distribution.

**Lec 43: Bias/Variance**

An example for a dataset containing two features:



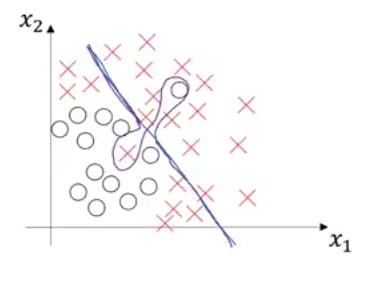
In problems involving high number of features it is difficult to visualize the decision boundary. The following will help us know if the model is an overfit or an underfit.

Consider a cat classifier:

1. ***Case1***: the train set error is 1% and the dev set error is 11%. This is an example of **overfitting** (**high variance**), because it has learned the train set well but not the dev set. The reason for this is due to poor distribution.
2. ***Case2***: the train set error is 15% and dev set error is 16%. Here the model isn’t learning well even for the train set hence it is **underfit** (**highly biased)**. On the other hand it is generalizing well in comparison to the dev set.
3. ***Case3***: the train set error is 15% and dev set error is 30%. This is an example of both high bias and high variance.
4. ***Case4***: the train set error is0.5% and dev set error is 1%. This has both low bias and low variance.

The above conclusions are made with human classifying at the rate of 0% error. If the human error rate (Bayes error) was 15%, then case 2 is a pretty good model.

Now how does a high bias and high variance model look like?



**Lec 44: Basic ‘Recipe’ of Machine Learning**

The following is the flow for the basic ‘recipe’ for a good model.

***High Bias?***

After training a model if It has high bias (underfit), i.e, not fit the training data well, one should consider the following options:

* Train a bigger network with more layers and hidden units
* Train for a longer time (more epochs/iterations)
* And maybe try a different neural network architecture

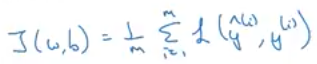
***High Variance?***

Suppose the train data fits well but the validation data doesn’t, this is overfitting (high variance). The following can help mitigate it:

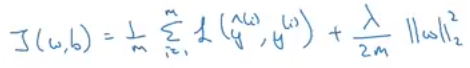
* Get more data
* Regularization
* Maybe try a different neural network architecture

**Lec 45: Regularization**

We will regularization in the light of logistic regression.

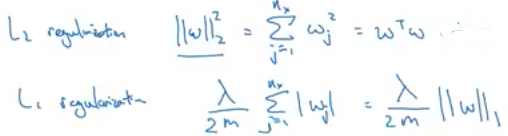
In logistic regression, we have to reduce the cost function:  having parameters ‘W’ and ‘b’.

To perform regularization the following must be added (regularization parameter):



So in the above regularization is done for the parameter ‘W’, which has a lot of parameters. It can be done for parameter ‘b’ but since it is just a single value it does not make much difference at al. Hence regularization of ‘b’ is omitted.

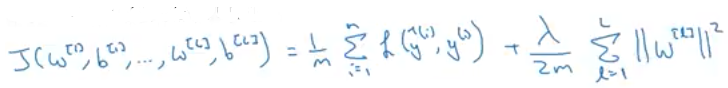
There are two kinds of regularization L2 and L1:



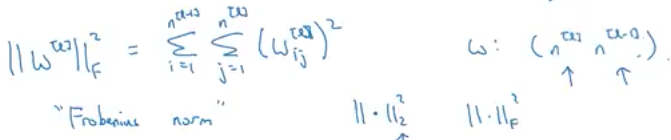
L1 results in a sparse matrix helps with lesser memory occupancy.

So there is an additional hyperparameter to tune ‘lambda’ which is the regularization parameter.

Now for a neural network the cost function changes as:

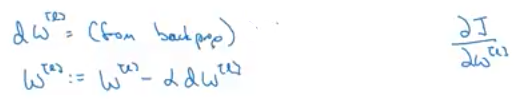


Where the ‘W’ parameters are:

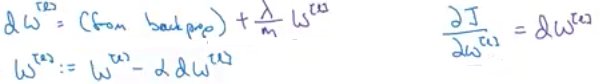


Now how do you perform backpropagation to perform gradient descent?

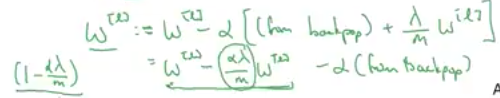
Before including the regularization term, it was carried the following way:



Now after including the regularization term:



Now there is a reason for calling ‘L2’ regularization as ‘weight decay’:



Apart from the usual weight changes another term (encircled) is also removed from the weights.

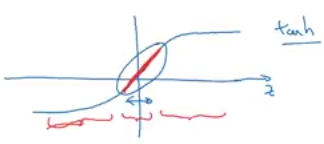
**Lec 46: Why Regularization reduces Overfitting?**

One intuition is that as the value of the regularization parameter (lambda) increases it nearly zeroes out the weight matrix parameters

The following is the next intuition:

Assuming that we have a large ‘lambda’ value, which results in a low ‘W’ parameters matrix. As a result, by this function: , the value of ‘z’ is also very small.

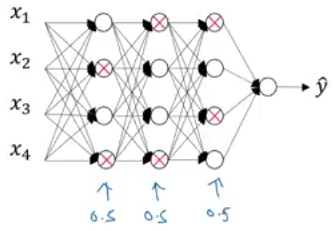
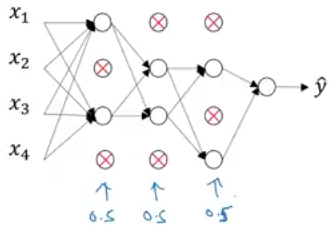
So for a ‘*tanh’* activation function, for a small value of ‘z’ the resulting value tends to fall in the linear region as shown below:



So for the entire neural network, all the layers would just be a linear function, which definitely avoids the problem of overfitting.

**Lec 47: Dropout Regularization**

After training a neural network, if you see that it is overfitting dropout regularization selectively choses some hidden units in each layer and removes them before the network can be trained again.

🡪 

***Implementing Dropout (Inverted dropout)***

Let us illustrate this at layer 3 (l=3). Let us assign a dropout vector whose probability of dropping random hidden units is 0.2.





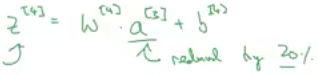
Now in the next step we perform element-wise multiplication of the 3rd activation layer with this dropout layer. This results in zeroing out certain activation units for which the probability is zero in the dropout layer:



Then we are scaling the 3rd activation layer by the following:



So for example, if you have 50 units in the layer, 10 units (20%) are shut down.



But in order to maintain the numbers properly, we include those 20 units using this scale. This will not affect the expected value of ‘z[4]’. We want to keep the same numbers as required in each layer. This is called the ‘inverted dropout’ which preserves the expected value for the given layer.

***Note:*** The dropout layer randomly zeroes different hidden units for different training examples.