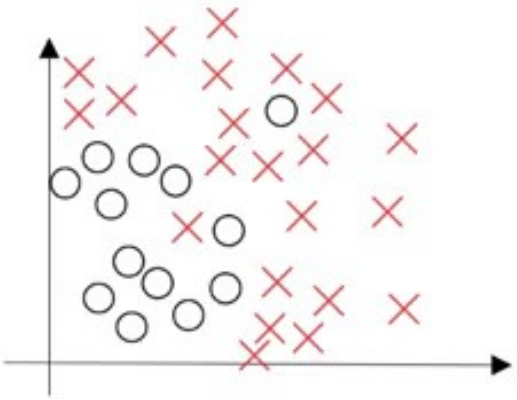
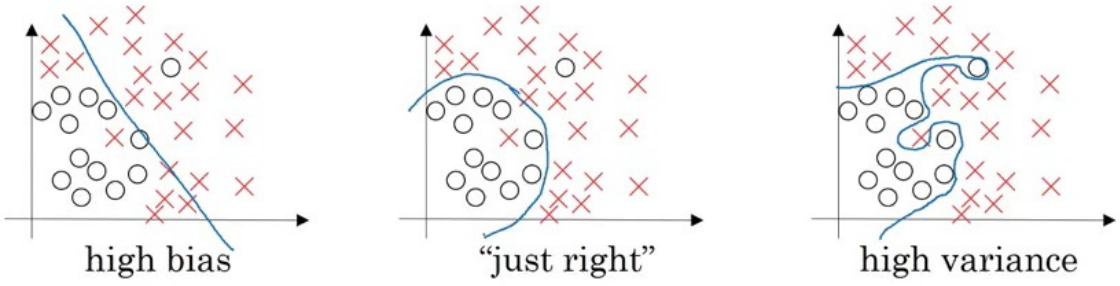
Bias and Variance

Consider a dataset which gives us the below plot:



What will happen if we fit a straight line to classify the points into different classes? The model will under-fit and have a high bias. On the other hand, if we fit the data perfectly, i.e., all the points are classified into their respective class, we will have high variance (and overfitting). The right model fit is usually found between these two extremes:



We train the model on the training data. After training the model, we check how well it performs on the validation set. When we have a final model (i.e., the model that has performed well on both training as well as validation set), we evaluate it on the test set in order to get an unbiased estimate of how well our algorithm is doing.

- If the validation set error is much more than the train set error, the model is overfitting and has a high variance.
- When both train and validation set errors are high, the model is underfitting and has a high .
- If the train set error is high and the validation set error is even worse, the model has both high bias and high variance.
- When both the train and validation set errors are small, the model fits the data reasonably and has low bias and low variance

High training error results in high bias. In such cases, we can try bigger networks, train models for a longer period of time, or try different neural network architectures. To reduce the variance, we can get more data, use regularization, or try different neural network architectures

Regularization

Variance can be reduced by increasing the amount of data. But is that really a feasible option every time? Perhaps there is no other data available, and if there is, it might be too expensive for your project to source. This is quite a common problem. And that's why the concept of regularization plays an important role in preventing overfitting.

1. L2 Regularization

Let's take the example of logistic regression. We try to minimize the loss function:

$$\hat{y} = w_1 x_1 + w_2 x_2 + \ldots + w_N x_N + b$$

$$\min_{w,b} J(w,b)$$

$$J(w,b) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Now, if we add regularization to this cost function, it will look like:

This is called L2 regularization. λ is the regularization parameter which we can tune while training the model. Now, let's see how to use regularization for a neural network. The cost function for a neural network can be written as:

$$J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

We can add a regularization term to this cost function (just like we did in our logistic regression equation):

$$J(w^{[1]}, b^{[1]}, \ldots, w^{[L]}, b^{[L]}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2m} \sum_{l=1}^{L} \|w^{[l]}\|^2$$

$$\|w^{[l]}\|^2 = \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l]}} (w_{ij}^{[l]})^2$$

Regularization in gradient descent algorithm

Update equation without regularization is given by:

$$dw^{[l]} = \text{from backpropagtion}$$

$$w^{[l]} = w^{[l]} - \alpha \, dw^{[l]}$$

Regularized form of these update equations will be:

$$dw^{[l]} = (\text{from backpropagtion}) + (\lambda/m) \, w^{[l]}$$

$$w^{[l]} = w^{[l]} - \alpha \, [dw^{[l]} + (\lambda/m) \, w^{[l]}]$$

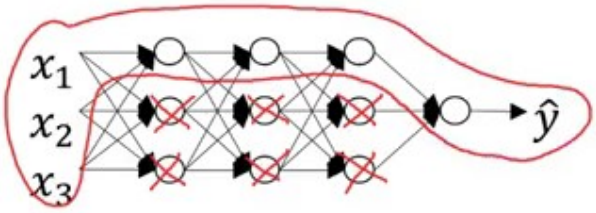
As you can surmise from the above equations, the reduction in weights will be more in case of regularization (since we are adding a higher quantity from the weights). This is the reason L2 regularization is also known as weight decay.

How regularization reduce Overfitting?

The primary reason overfitting happens is because the model learns even the tiniest details present in the data. So after learning all the possible patterns it can find, the model tends to perform extremely well on the training set but fails to produce good results on the dev and test sets. It falls apart when faced with previously unseen data.

One way to prevent overfitting is to reduce the complexity of the model. This is exactly what regularization does! If we set the regularization parameter λ to a large value, the decay in the weights during gradient descent update will be more. Hence, the weights of most of the hidden units will be close to zero.

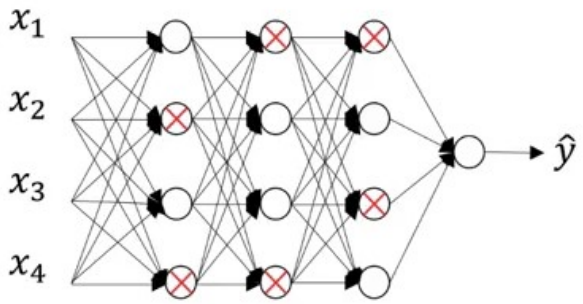
Since the weights are negligible, the model will not learn much from these units.This will end up making the network simpler and thus reduce overfitting:



2. Dropout Regularization

Researchers have noticed that Neural networks over-fit due to “co-adaption” between neurons. Co-adaption occurs when two or more neurons in the network begins to detect the same feature repeatedly, which means network isn’t utilizing its full capacity efficiently. This shows, it is wasting computational resources by computing the activation for redundant neurons that are all doing the same thing. In order to break the co-adaption, noise is introduced in the network during training. Each output from a layer is randomly set to zero with some probability p

Figure shows difference between the standard neural network and network after applying dropout.

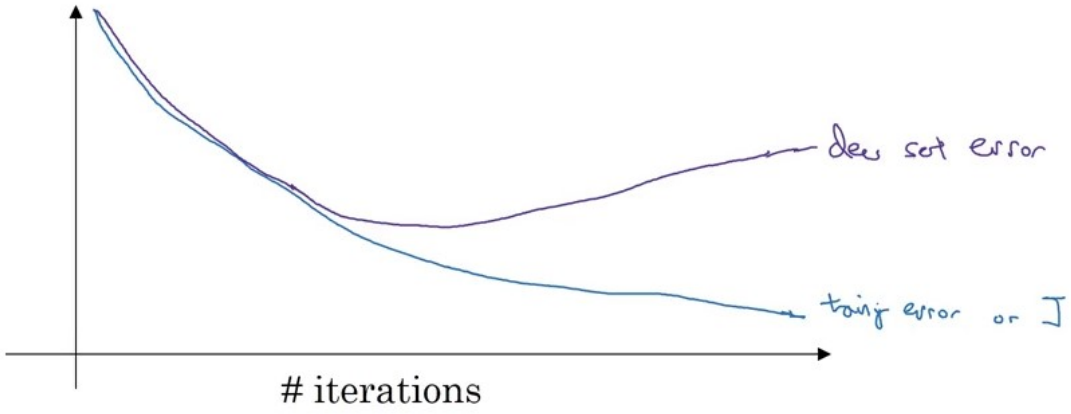


At each training stage, individual nodes are either "dropped out" of the net with probability 1 − p or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage.

3. Other Regularization methods

Data Augmentation **:** Suppose we are building an image classification model and are lacking the requisite data due to various reasons. In such cases, we can use data augmentation, i.e., applying some changes such as flipping the image, taking random crops of the image, randomly rotating images, etc. These can potentially help us get more training data and hence reduce overfitting.

Early Stopping **:** Consider the below example,



In [ ]: