ANN Assignment 2

1.(a) Formulization of linear regression as maximization of likelihood function:

What we try to do in linear regression is to find some function y_hat that can relate the input feature to output. The process involves randomly choosing some parameters or weights and biases and after producing the mathematical output, feed it into our loss function which is predefined.

The next step is performing the same steps over all the data provided to us and then taking average of the sum of loss function, which is quoted as "cost function". we then update the parameters in a way which reduces the overall cost.

So taking an example if we define:

$$L(y\_hat, y) = \frac{1}{2}(y\_hat - y)^2$$

If we try to minimize the cost function means we try to minimize the loss function. And this simply means trying to make the y_hat as close to y as possible. Mathematically this basic process is nothing but trying to maximize the likelihood of y_hat being the right function that maps from input to output.

1.(b) Classification algorithms like logistic regression learns a probability distribution over classes instead of the right classes itself:

The answer is basic: the data is processed as if it lists the input and the probability of the input being in a particular class. For example, the following sample data will be seen as below:

7	seven
6	six
5	five

class	Zero	One	Two	Three	Four	Five	six	Seven	eight	Nine
7	0	0	0	0	0	0	0	1	0	0
6	0	0	0	0	0	0	1	0	0	0
5	0	0	0	0	0	1	0	0	0	0

So what the classification algorithm is trying to do is to find the function that fits this data which have as its output nothing but the probabilities of a particular no being in a particular class. That's why they end up learning the probability distribution of classes instead of providing an answer like this no belong to this particular class.

2. Stochastic Gradient Descent with Momentum:

Stochastic Gradient Descent is different from batch gradient descent algorithm in the way it does the updates to the parameters. While BGD updates the parameters after going through all the data, SGD just randomly selects one particular example and do the update based on this single

example. So the plus point in SGD is its speed and the negative is the large fluctuations in the update values which makes it harder to reach to the minima even when we are near it. Taking a physical equivalent of the problem, consider a ball going down a hill, but choosing random directions at each step.

What BGD do is to choose a direction towards the minima but the updates are not too random as that depends on whole data. Thus though the process of choosing the direction may take time it certainly do so in a consistent manner i.e. never deviating too much from the steepest direction towards the minima.

While the decision making in SGD is fast, it is too random. This means the algorithm deviates too much from the steepest path. The update for one step looks like:

$$w_{updated} = w - \alpha \cdot \frac{\partial L}{\partial w}$$

$$b_{updated} = b - \alpha \cdot \frac{\partial L}{\partial b}$$

To decrease the fluctuation of updates we make some changes in the update term:

$$v_t = \beta \cdot v_{t-1} + (1-\beta) \cdot \frac{\partial L}{\partial w}$$

$$w_{updated} = w - \alpha \cdot v_t$$

Where the value of beta is generally taken as 0.9

So by adding the first term we are somehow remembering the past updates and as the value of beta is less than 1 the effect of updates done so long ago will not make much difference. The algorithm is called Momentum because of the analogy with physical momentum.

Following analogies/justification might be helpful:

1. Adding first term is like remembering what we have done earlier and using the same to determine what to do next. In physical world, an object with momentum follows similar properties. It's state in the next moment is determined by its previous state(Newton's first law).
2. Also as we can see the momentum term keeps on increasing as we move ahead. This makes the process of learning accelerated. Analogically in real world as the body with momentum goes down its momentum increases making it to reach minima sooner than otherwise.
3. It is also found that this momentum term helps in overcoming the local minima faster than otherwise. If we see local minima as a pit on road and the update process as a ball going down the road, it is easy to observe that a heavy ball is more likely to overcome the pit faster than a slow moving light ball(momentum 0).

3.(a) Mini – Batches :

As already stated above the BGD is computationally inefficient and while SGD has great speed it cannot ignore the noise in the data which introduce a lot of randomness in the update values.

What the Mini-batch descent algorithm does is to combine the best of both algorithms. It randomly selects a particular no of training example and then do updates on parameters after going over that "mini-batch". So mini-batches comes out to be better than both SGD(mini-batch_size =1) and BGD because:

1. It is computationally efficient as it selects only few training examples in each iteration rather than going over all the data. This results in making it a fast algorithm.
2. It is a robust algo as using a no of example makes it less random and as a result it does not have trouble locating minima as well as overcoming the local minima points.

3.(b) Symmetric initialization of parameters:

The purpose of introducing different layers as well as different hidden units is to learn different relation bw the features and then optimizing the parameters so as to go as close to real results as possible. So the basic difference bw a simple or shallow network and a deep network is that while simple network consider only a single relation comprising all the features at once, the deep networks learn many possible relations and then giving different weights to different relations and so on. But if we initialize the parameters symmetrically in deep network, both the input and parameters for every hidden unit is same the hidden units end up doing the same thing and after a while doing same update. Because of the same update the parameters are still same so the symmetricity continues and by induction we can prove that they will end up learning same parameters at the end. So the basic purpose of learning different relations is not achieved in such an initialization. We need to randomly initialize the parameters so as to break the symmetry.

3.(c) Regularization:

Overfitting is the state when our model fits too well in the training sets but fails to generalize it to real world data. To overcome overfitting one of the method is regularization. In most of the cases the overfitting of data is related to large parameters values. What regularization does is to stop the model in making these parameters large. For this a term is added in the cost function:

$$J = \frac{1}{2 \cdot m}\sum_{i=1}^{m} L\big(y_{hat}(i), y(i)\big) + \frac{\lambda}{m}\sum_{i=1}^{m}\sum_{j=1}^{n} ||w_{i,j}||^2 \ \ in\ particular\ using\ \ frobenius\ norm$$

Lambda is a hyperparameter. So what the step does is to penalize for large parameters. As we have to minimize the cost if some parameter comes out to be large, it will end up making the process to update it to a lesser value so as to minimize the second term.

There are many ways to overcome overfitting. Some of them are listed below:

1.cross validation

2.training with more data

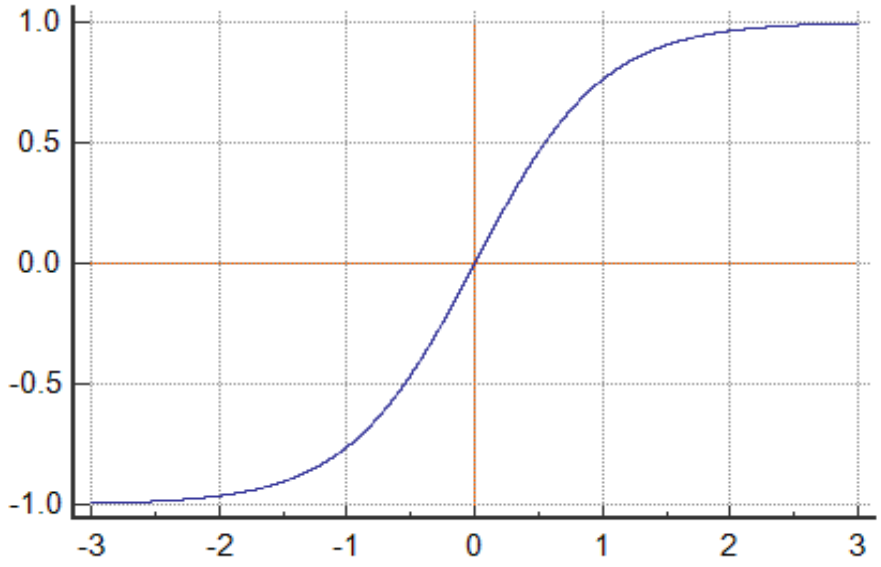
3.early stopping

4.dropout

3.(d) Batch Normalization:

Batch normalization is a process of normalizing the input data to a smaller and less spread values. It simply involves subtracting the batch mean from every input and then dividing them by the standard batch deviation. This results in making the input to the same scale and results in more stable model. This is how:

Consider the tanh function:



As clear from the figure if the input is in the range -1 to 1 the curve is close to linear and for really large values the learning becomes really slow. This is how batch normalization is working. Before normalization the learning was slow for large values of input features but now as all the data is in range -1 to 1 or close to this range the learning is consistent and the model performs better on the same data amount.

4.Convolutional networks:

In order to calculate the no of parameters in conv layer:

The filter size is [filter_height,filter_width,in_channels,out_channels]

This gives us the no of parameters as 3 x 3 x 3 x 8 = 216

And with padding = 1 and stride =2 if image size is N1 x N2 x 3 the output size is simply given by

[(N1 +2 *P -3)/S + 1 , (N2+2*P-3)/S + 1 , 8]

Putting values:

(N1+1)/2 x (N2+1)/2 x 8

Is the requires output size.