Assignment 2

May 21, 2020

1 Introduction

1.1 Linear regression as maximization of a likelihood function

We can also find the parameters of a model by maximising likelihood function. The purpose of this method is to find the parameters such that they maximize the likelihood that the output of the model is as close to the actual output as possible. In other words, we find particular values for the parameters so that the resultant model with those parameters would have generated the given data or data as similar to given data as possible. The likelihood expression is of the form :

$$L(parameters|data) = P(data|parameters)$$

We thus try to maximize the conditional probability of observing the data given a specific probability distribution and its parameters and this conditional probability is the likelihood. In this, first we decide a statistical model that we think best fits the given data. Then we try to use MLE to find values of params. Let's learn it for the linear regression model.

First we have to decide for a statistical distribution. Let's assume that the target value is in normal distribution. We have two parameters for normal distribution i.e. mean and variance. Now what we do in linear regression is that we take the model as the mean of the above stated normal distribution. Let's say we have simple linear regression model :

$$\hat{y} = m * x + c$$

Then what we need to do is find m, c s.t. joint probability distribution of all the points is maximum. We also assume that each data point is independent of others to avoid calculating conditionals. Then the resultant probability is the product as below:

$$\prod_{i=1}^{n} \frac{e^{\frac{-1}{2*\sigma^2}(y_i - h(x_i,\beta))^2}}{\sqrt{2\pi\sigma^2}}$$

So we have to find the values of and that results in giving the maximum value of the above expression. And we can do this by differentiating and finding maxima of the function.

1.2 Classification algorithms learning probability distribution over classes conditioned on the input instead of directly producing an output y

The classification algorithms like logistic regression learn a probability distribution over classes conditioned on the input instead of directly producing an output y with an appropriate loss function defined on the output y instead of on the output probabilities. The reasons are:

- If we define a loss function on the probability instead of the output y and probability distribution over probability then, in that case we wont have a value for the probability for output y. We just have the output labels and so we wont be able to train as the final output wont be labelled. So it does not make sense.
- If we try to fit parameters to our output labels directly then what will happen is that it will give output in a continuous range and it will give output in negative in between labels and outside that which is not correct for our model.
- Using maximum likelihood as well, we use binomial probability distribution in logistic regression where each example is one outcome of Bernoulli trial i.e. in that we calculate probability of the particular example as outcome. And we calculate cost on the actual outputs and that is what we want to minimize and not the probability. So at the end, we want out output labels to be accurate and not the probability to be maximum so we train according to that.

2 Stochastic gradient descent with momentum

Stochastic gradient descent is when we have mini batch size of one in batch gradient descent or simply when we are taking a training example, calculating gradient descent on it and then moving to next example in the set.
Gradient descent with momentum is a technique in which we use exponentially weighted averages of gradients (instead of the gradient itself) and then use those to update the weights of our parameters. The algorithm is:

For each iteration i:

$$vdW = \beta * vdW + (1 - \beta)dW$$

$$vdb = \beta * vdb + (1 - \beta)db$$
$$W = W - \alpha * vdW$$
$$b = b - \alpha * vdb$$

Now lets try to understand these equations. What we are doing here is instead of taking gradient at each step, we are averaging the gradient of last few iterations. Say we are moving over to minima in an elliptical function, and say its elongated on the horizontal axis. So with regular gradient descent the optimization will be very slow and will have a lot of up and down oscillations in vertical direction. But with momentum approach, we have average of previous gradients so the average in the horizontal direction won't change much but in the vertical direction it will average out the up and down oscillations to zero and therefore it will bring these down to nearly zero and thus will help in moving faster towards the minimum.

Analogy with physics momentum
Say you want to minimize a bowl shaped function. Then these derivative terms act as acceleration, β as friction and $vdW or vdb$ as the velocity. Then acceleration provides the direction and increases or decreases the velocity in particular direction. Since the beta term is less than one, so it works as friction and tries to stop it from changing blindly. And the vdW is the velocity that we are trying to control with these. And thus with this, we can say it accelerates and gains momentum in a particular direction such that it can move down to minima the fastest.

3 Explain the following

3.1 Why mini batches are used ?

In mini batch gradient descent, we take a small set of examples and call it a mini batch and perform gradient descent on that (and repeat it for all the mini batches of the train set) rather than taking the entire training set all at once. Using batch gradient descent requires the entire training data set in memory and available to the algorithm and thus it may become slow for large data sets. In turn if we use SGD, then we are using one train example at a time so we lose the speed we get due to vectorisation and also it is computationally very expensive. Also since we are updating one at a time, then such frequent updates can result in a noisy gradient signal, which may cause the model parameters and in turn the model error to have a higher variance.
Therefore, we neither use the entire set nor single train example, and we use mini batches so that we can get the speed due to vectorization. It also provides efficiency of not having all training data in memory and algorithm implementations.

3.2 Why symmetric weight initialization is a problem

If the weights are initialized to all 0, then when we calculate the derivative in say layer l w.r.t. the loss function then the values will be same for the activation units in a particular layer. So the consequent updating parameters will generate same values for W[l] in each iteration. SO the function will be symmetric.
To avoid this we initialize weights randomly in order to break this symmetry. The bias values may be initialized to zeros but the weights are random. We also have to take care of not having very large or very small initial values of these weights as this will lead to very large values for Z[l] which when passed through sigmoid activation may end up being very close to zero or one and thus the gradient descent will be very slow. So careful initialization is to be done and there are methods like **He initialization**, **Xavier Initialization** etc.

3.3 Regularization and over fitting

Over fitting is when the model performs too well on the train set and learns it too well and therefore is not able to generalize and perform well on test set or a new, unseen set of data.
With regularization, we add a penalty term in which with large value of λ, the weights will be close to zero. So the intuition is that it nullifies the effect of some activation units and reduces the model to a simpler model. This helps in reducing over fitting. In another method, we use dropout regularization in which we randomly drop some units at each iteration carrying a similar effect of simplifying the neural network.
Other ways of reducing high variance or overfitting may be by including more data (use data augmentation or collect more data etc), trying a different architecture maybe, reducing number of features (but you should not prefer this and check what you are dropping as it may lead to loss of important information)

3.4 Batch Normalization in Neural Network

In batch normalisation, we try to normalize the hidden layers just like we normalize the input layers. The purpose it serves is same as normalizing the input layers. It speeds up learning and helps the layers to learn a bit more independently of the other hidden layers. It centers the activation units around zero (by subtracting mean) and vary around a smaller range (by dividing with variance) which helps in reducing the problem of input values changing or shifting as they pass through layers. Say we have a model trained on black cats and we have a test set of colored cats, then without BN the test accuracy will be low due to co-variance shift. But with BN, the accuracy will be good on the test set as well. So it is quiet helpful technique.
The algorithm for applying Batch normalization includes:

$$mean = \frac{1}{m} * \sum_{i=1}^{m} (z[i])$$

$$variance = \frac{1}{m} * \sum_{i=1}^{m} (z[i] - mean)^2$$

$$Z_{norm}^{[i]} = \frac{(z^{[i]} - mean)}{\sqrt{variance + \epsilon}}$$

$$Z_{tilde}^{[i]} = \gamma * Z_{norm}^{[i]} + \beta$$

$$A^{[i]} = f(Z_{tilde}^{[i]})$$

So we first calculate the norm of Z and then apply the activation function on that. We can also first calculate the activation and then apply normalization. Both γ and β are learnable parameters. We can also emit the bias parameters b as it will nullify on calculating and subtracting mean. So with BN, we have to find weight inputs, γ and β for all the layers.

4 Dimensioning the Conv Layer

- Kernel size : $3x3$ or fxf
- Input channels : 3
- Output Channels or the number of filters used : 8
- Padding : $P = 1$
- Stride : $s = 2$
- Input image : $N1xN2x3$

1. The weight parameters are :
   weight = Kernel Size * Input Channels * Number of filters = 72

2. The output dimensions are:

$$output = \lfloor \frac{N1 + 2P - f}{s} + 1 \rfloor * \lfloor \frac{N2 + 2P - f}{s} + 1 \rfloor * 8$$

$$output = \lfloor \frac{N1 + 3}{2} \rfloor * \lfloor \frac{N2 + 3}{2} \rfloor * 8$$