[1]

In this lecture we will describe the gradient descent algorithm which is an efficient and automatic way of finding the global minimum for a linear system of N variables (or features) and N weights which form a hyperplane prediction function in an N-dimensional feature space.

[2]

Recall that in a system of one descriptive feature and one target we have two weights w[0] representing the y-axis intercept and w[1] representing the slope. When we graph these weights in a 3D plot with the error function on the Z-axis, we generate an error surface representing all of the residual error points on a continuous surface for all of the variations in the weight parameters. The global minimum for this function is the point on the surface for which chosen values of w[0] and w[1] produce the lowest error value. Gradient decent is the way we will automatically find this minimum

[3]

To get an intuition for how gradient descent works, consider a hill walker who must navigate from the summit to the base of a mountain. Suppose further that there is poor visibility and that the walker can only see one to two meters in front. The best strategy would be to look around and take a small step in the downward direction with the best incline. The walker would repeat this process step by step until she reaches the bottom. A analogous process takes place in the gradient descent algorithm. The algorithm starts with some randomly initialised points on the hyperplane and the proceeds to find the global minimum with respect to all the dimensions until it converges at the lowest point.

[4]

The key idea is the slope of the error surface with respect to each of the parameters w[0] to w[n]. When that slope is equal to zero, the local minimum with respect to that parameter has been reached. Because we have to consider n weights we consider each in turn, then sum these and then find the minimum of that sum. The partial derivate with respect to w[j] is the derivative of the loss function we have already defined for linear regression systems

[5]

As the our loss function is of the form function of a function (because of the square exponentiation), we use the chain rule to expand the terms. The reason for including the one half constants multiplier in our loss function definition now becomes clear as this cancels out when we take the derivative of the squared residual part. When we expand the weighted model expression it’s easy to see that only the term related to j will remain in this partial derivative. So this gives us a nice result for one of the N partial derivates we must obtain

[6]

The algorithm proceeds as follows. Initialize the weights vector to random values. The choice of these random values will be discussed later. Then for each weight in the vector update the new weight with some delta value times a constant factor learning rate alpha. The determination of alpha and delta are considered further on. It’s important that all these weight updates are done simultaneously and before the next iteration of the loop. Proceed until convergence, that is until the global minimum cannot be made any smaller.

[7]

The delta value, the amount by which to descend, is computed as a sub function and uses the partial derivates presented earlier. For simplicity without loss of generality we can drop the one over m divisor when summing the partial derivates to give us our first result. The slope returned by this partial derivate function is in the direction towards the highest points on the error surface but we want our delta function to take steps downwards towards the lowest points so we define our delta function as the negative of the derivative function. Multiplying though we arrive at the update expression as shown. Intuitively, the new weight is a function of the existing model prediction and the associated feature. It should now be clear why the weight updates must be simultaneous and atomic with respect to the model predictor function. That is, all the new weights are updated using the old weights before the new weights are used in the next function iteration.

[8]

The learning rate constant alpha is a way to control the rate of descent of the gradient descent algorithm. This should be a positive real number greater than zero. The choice of alpha is important. Too high a value for alpha risks overshooting the minimum and causing the algorithm to oscillate around but never converge on the minimum. Less serious is an alpha value which is too small, and which causes the convergence to take too long. The idea is to take measured steps towards the minimum as quickly as possible but in such as way as you are guaranteed to converge.

[9]

There are no hard and fast rules as to the choices of the random weight initialisations or the choice of the alpha constant learning rate. At this point in our knowledge of the linear regression model, it is recommended to experiment with different initialisations and train and test your models over randomly sampled training and test datasets to see which values make sense. Each problem domain will be different from another, so it is difficult to be definitive about the best choice.

[10]

In summary, the gradient descent algorithm is an efficient and automatic way of finding the global minimum in a linear system. The global minimum is the point on the error hyperplane surface as which the model weights are at their optimal values for minimising the residuals between the observed training data points and the model predictions. Gradient descent is widely used in machine earning, not just in linear regression systems. The mathematical expressions in the algorithm lend themselves nicely optimisations on modern CPU and GPU architectures for which there is widespread support in