Lecture 10 Notes

CS-419M

Spring 2021

1 Algorithms to Solve Supervised Learning Problems

1.1 Gradient Descent

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).
Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.

Note 1. General Form of Gradient Descent:

$$w_{t+1} = w_t - lr * \frac{\partial F(w)}{\partial w_t} \tag{1}$$

Where,

$$\begin{aligned} w_t = &\text{ the weights in the } t^{th} \text{ epoch} \\ lr = &\text{ learning rate, and} \\ F(w) &\text{ calculates cost for the given weights w} \end{aligned}$$

This weight update rule is applied until the model converges, meaning the model has reached the point of global minima, and performing any more weight changes will not lead to a lower cost. Now, how do we determine this point of convergence? How do we decide, when to stop our model from training?

1.1.1 Determining Convergence

One method many new ML practitioners do, is fix a number of epochs that the model will train. This is bad practise, as you never know whether your model has converged, and whether cost could drop even more.

Another means is to compute $|w_t - w_{t+1}|$ and check whether it is lesser than a small value ϵ. This is not as bad as the previous one, but if you see equation 1 you know that difference is $lr * \frac{\partial F(w)}{\partial w_t}$, and given a small enough lr, this value is not a good indicator whether the gradient itself is small enough.

The best way to determine convergence, is to calculate the cost on a held-out set, also known as validation set. This data is not known to the model, and hence gradient descent does not directly optimise it.

Algorithm 1: Gradient Descent with Patience on Validation Error

```
w = random()
best_cost = ∞
for i=1 to 10000 do
    w_t = weight update using gradient descent
    flag += 1
    if cost(w_t) < best_cost then
        save(w_t);
        flag = 0;
        best_cost = cost(w_t);
    end
    if flag==10 then
        break;
    end
end
```

In the above algorithm we don't end the training as soon as $cost(w_t)$ is less than $cost(w_{t+1})$, but we keep checking if that occurs for 10 additional epochs. This value 10 is called the patience parameter, and it ensures that cost can indeed not drop anymore.

1.2 Stochastic Gradient

Stochastic gradient refers to the practise where the weight update rule is performed over each data point, one at a time. When the dataset is massive, and hence not possible to load it completely into memory, SGD is one way to train the model, by loading data points one by one into memory. The main application of SGD is in online learning, where only a few data samples are available at a time, and the ML model is trained only on those.
One inherent disadvantage of SGD is that it tends to cause over-fitting, as the weight update is completely dependent on the gradient of a single data point. One way of negating this is by using a momentum parameter, which basically remembers the past weight updates and gives lesser weight to the most recent weight update. But the easier method to overcome it is discussed in the next section.

1.3 Mini Batch Gradient Descent

The natural succession to the SGD method, is to create mini-batches of data points for training. This overcomes the over-fitting issue, as we're not dependent on a single data points any more for weight updates, as well as we don't need to load the entire dataset into memory.
The size if this mini-batch is a hyper-parameter, and it is up to the programmer to decide how big or how small the batch size needs to be.

2 Practical tips for Optimisation

Always prefer to write vectorised operations rather than using loops, whenever possible. The advantage of vectorised operations is that they are massively faster, due to them being inherently parallelizable. Most operations in math libraries like scikit-learn, numpy, pandas and of course ML packages like pytorch and tensorflow have inbuilt vectorised operations for most of their functions.