CME 216, ME 343 - Spring 2020

Eric Darve, ICME

In this lesson we will learn about gradient descent methods.

Many algorithms that we will present are heuristics. Finding the best method for your problem often requires trial and error.

We have seen that the most basic method to train is the method of steepest descent or gradient descent:

$$\Delta W = -\alpha \nabla_W L$$

where W are the weights and L is the loss function.

Typically we have a large training set

$$X_i,\ i = 1, \ldots, m$$

The loss function can be written as

$$L(W) = \sum_{i=1}^{m} l(X_i; W)$$

In many cases the training set is very large.

In principle, the "correct" method is to calculate

$$\nabla_W L$$

by computing each partial gradient

$$\nabla_W l(X_i; W)$$

and summing all the terms together.

However, going through the entire training data can take a while and also the DNN is not updated using we have computed all the $\nabla_W l_i$.

This has led to the idea of stochastic gradient descent.

In this method we don't wait until we have computed all the terms to apply the gradient.

There are different ways of implementing this.

Assume that we randomly select a fraction r of the training samples and compute

$$L_r(W) = \sum_{k=1}^{rm} l(X_{i_k}; W)$$

Each set $\{i_k\}$ is called a batch.

Then we update the weights using

$$\Delta W = -\alpha \nabla L_r(W)$$

and repeat this, drawing a new set of rm samples.

This method has several advantages.

First, we can update the network even after processing only rm training points.

Assume that $r = 1/4$ for example.

One "epoch" corresponds to evaluating L for all m training points.

In traditional gradient descent, we update the DNN weights only once per epoch.

That is, after processing all training samples, we apply the gradient once.

In stochastic gradient descent, we will update the DNN weights

$$\frac{1}{r} = 4$$

times during each epoch.

If we assume that each gradient calculation using samples

$$\{i_k\}, \quad i = 1, \ldots, rm$$

is reasonably accurate, the convergence can be expected to be much faster.

Basically in SGD, we are updating the DNN weights more often so we typically require fewer epochs to converge.

The second reason why SGD is more efficient is that it introduces a stochastic component during the training because

$$\nabla L_r(W) = \sum_{k=1}^{rm} \nabla l(X_{i_k}; W) \approx \nabla L(W)$$

This may help the algorithm converge and get away from points where ∇L gets small and convergence slows down.

Noise actually helps!

We will discuss convergence and the effect of noise in more details in the next video.