9 Gradient learning

Date	@November 12, 2024
Topic	Theory

Gradient based learning

ERM on parameterized models

In order to get the best weights w* we need to perform this minimization

$$\min_{\mathbf{w}\in\mathbb{R}^n} f(\mathbf{w}) := (1/m)\sum_{i=1}^{m} \underbrace{L\left((\mathbf{x}^{(i)}, y^{(i)}), h^{(\mathbf{w})}\right)}_{f_i(\mathbf{w})}.$$

If each loss $f_i(w)$ is differentiable, $f(w)$ is also differentiable, such as squared error loss, logistic loss, etc...

Minimizing the empirical risk means that if you're on a mountain you want to reach the valley, which is the place with the minimum height. This means that you have 2 dimensions and you can see them as your weights w_1 and w_2. So you want to reach the place where the losses (ER) are at their minimum. As human being in this situation you need a map, which would be the explicit formulation of the hypothesis space $f(w)$ for each w, but this is not possible because it might be a very complex space since you don't have the explicit version. So what you can do is to try some w (as you know your position even if you don't know the height of your position). Another solution if you don't have a map is to go down in, but in this case you can't know if you reach the bottom of the mountain.

Gradient based method

Gradient based methods are iterative and they construct a sequence of parameter vectors (as you change your position by following a path) $w^0 \to w^1 \to w^2 ...$

This sequence converge to a minimizer of $f(w)$

$$f(\overline{\mathbf{w}}) = \bar{f} := \min_{\mathbf{w}\in\mathbb{R}^n} f(\mathbf{w}).$$

In practice it means computing the gradient:

1. approximate locally $f(w)$ around w^r

$$f(\mathbf{w}) \approx f\left(\mathbf{w}^{(r)}\right) + \left(\mathbf{w} - \mathbf{w}^{(r)}\right)^T \nabla f\left(\mathbf{w}^{(r)}\right) \text{ for } \mathbf{w} \text{ sufficiently close to } \mathbf{w}^{(r)}.$$

2. since we want to minimize f, the quantity

$$\left(\mathbf{w} - \mathbf{w}^{(r)}\right)^T \nabla f\left(\mathbf{w}^{(r)}\right)$$

should be negative

3. if the gradient is positive, we decrease **w**
4. if the gradient is negative, we enlarge **w**

Gradient descent step

This also tells you how steep it is, because if the gradient is very large, it means that it is very steep, while if it is small, it means that we're almost in a flat position.

Given the current guess w^r, the next guess is

$$\mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \alpha \nabla f(\mathbf{w}^{(r)}) \qquad \text{with a sufficiently small step size } \alpha > 0$$

where α is called learning rate.

We can apply this to different models and different losses.

For example with linear regression, where the loss is the squared error loss

$$\overline{\mathbf{w}} = \underset{\mathbf{w} \in \mathbb{R}^n}{\operatorname{argmin}} f(\mathbf{w}) \text{ with } f(\mathbf{w}) = (1/m) \sum_{i=1}^{m} \left(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\right)^2.$$

we have to compute the gradient

$$\nabla f(\mathbf{w}) = -(2/m) \sum_{i=1}^{m} \left(y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)}\right) \mathbf{x}^{(i)}.$$

and then we compute the GD steps

$$\mathbf{w}^{(r)} := \mathbf{w}^{(r-1)} + \alpha(2/m) \sum_{i=1}^{m} \left(y^{(i)} - \left(\mathbf{w}^{(r-1)}\right)^T \mathbf{x}^{(i)}\right) \mathbf{x}^{(i)}.$$

Choosing the learning rate

The α parameter tells you how big your step is.

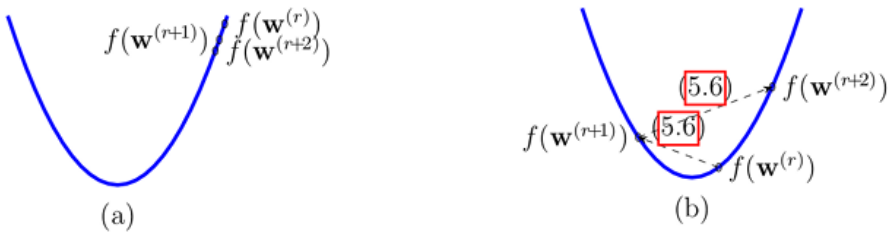

Figure 5.3: Effect of choosing bad values for the learning rate α in the GD step(5.6). (a) If the learning rate α in the GD step (5.6) is chosen too small, the iterations make very little progress towards the optimum or even fail to reach the optimum at all. (b) If the learning rate α is chosen too large, the iterates $\mathbf{w}^{(r)}$ might not converge at all (it might happen that $f(\mathbf{w}^{(r+1)}) > f(\mathbf{w}^{(r)})$!).

The choice of the learning rate in the GD step has a significant impact on the performance of the resulting ML methods. If we choose the learning rate too large, the GD steps diverge and, in turn, fail to converge towards the optimal weights w. If we choose the learning rate too small, the updates make only very little progress towards approximating the optimal parameter vector w. In applications that require real-time processing of data streams, it might be possible to repeat the GD steps only for a moderate number. If the learning rate is chosen too small, the updates will fail to arrive at a good approximation of w within an acceptable number of iterations.

There are different ways to define α and in some cases there are optimal values that can be used and it can be proved that if data are standardized the number of steps to reach the minimum is smaller.

When to stop

One solution is to use a fixed number of iterations, which are known are epochs.

Another idea is to check if the empirical risk (on training)

$$f(\mathbf{w}^{(r-1)}) - f(\mathbf{w}^{(r)})$$

is not improving any more or is not improving enough, using a threshold.

Or you can use the validation error

$$\tilde{f}(\mathbf{w}^{(r)})$$

and check if it keep decreasing or it is not improving anymore.

These three solutions can be used independently or combined together.

Stochastic gradient descent

In stochastic gradient descent we're not computing the ER on all the samples but on few samples (so it is faster).

$$\nabla f(\mathbf{w}) = (1/m) \sum_{i=1}^{m} \nabla f_i(\mathbf{w}) \text{ with } f_i(\mathbf{w}) := L\left((\mathbf{x}^{(i)}, y^{(i)}), h^{(\mathbf{w})}\right).$$

We've seen that the GD involves a sum and computing the sum in can be computationally challenging for at least two reasons:

1. is challenging for very large datasets with m in the order of billions
2. for datasets which are stored in different data centres located all over the world, the summation would require a huge amount of network resources

The idea of SGD is to replace the exact gradient f(w) by an approximation that is easier to compute than a direct evaluation:

1. iteratively replace sum with a (random) component (a single sample), so we compute the gradient just of a single sample; this is the fastest solution but this approximation could lead to errors

$$g(\mathbf{w}) := \nabla f_{\hat{i}}(\mathbf{w}). \qquad \mathbf{w}^{(r+1)} = \mathbf{w}^{(r)} - \alpha \nabla f_{\hat{i}_r}(\mathbf{w}^{(r)}),$$

2. iteratively replace sum with a (random) batch (set) of samples, and for each sample you estimate a gradient

$$\mathcal{B} = \{i_1, \ldots, i_B\} \text{ (a "batch")} \qquad g(\mathbf{w}) = (1/B) \sum_{i' \in \mathcal{B}} \nabla f_{i'}(\mathbf{w})$$

Advanced gradient based techniques

An example is to keep memory of the gradient at previous iteration (improved local approximation).

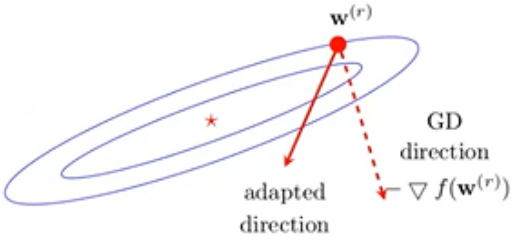

Figure 5.5: Advanced gradient-based methods use improved (non-linear) local approximations of the objective function $f(\mathbf{w})$ to "nudge" the update direction towards the optimal parameter vector $\overline{\mathbf{w}}$. The update direction of plain vanilla GD (5.6) is the negative gradient $-\nabla f\big(\mathbf{w}^{(r)}\big)$. For some objective functions the negative gradient might be only weakly correlated with the straight direction from $\mathbf{w}^{(r)}$ towards the optimal parameter vector ($\star$).