SUPERVISED LEARNING WITH A GRADIENT SEARCH

Prof FRANCESCO A.No PACHIELI
UNIVERSITA' DELLA CAMPANIA
LUIGI VANVITELLI

CORSO DI SIGNAL PROCESSING & DATA FUSION Oct 2023

In the previous chapter, both for regression and for classification, we have the problem of finding a parametric vector function $\underline{f}(\underline{x};\underline{\theta})$ that can match a vector $\underline{d}$ on the training set according to a specified loss function. More specifically, given a training set of pairs

$$\mathcal{Z} = \{(\underline{x}[n], \underline{d}[n]), n = 1, \dots, N_z\},$$

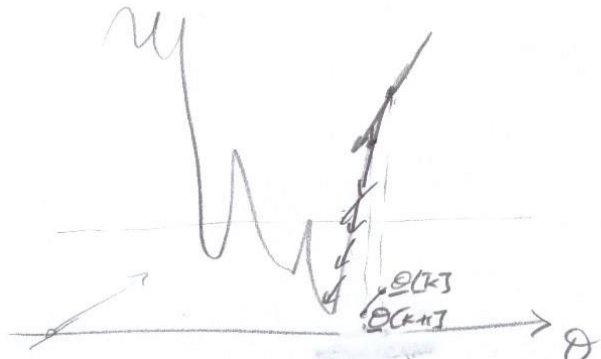
the problem is to find

$$\underline{\theta}^* = \arg\min_{\underline{\theta}} \frac{1}{N_z} \sum_{n=1}^{N_z} L(\underline{f}(\underline{x}[n], \underline{\theta}), \underline{d}[n])$$

the cost function $\mathcal{E}(\underline{\theta}) = \frac{1}{N_z} \sum_{n=1}^{N_z} L(\underline{f}(\underline{x}[n], \underline{\theta}), \underline{d}[n])$ may be very complex and defined on a large-dimensional space $\underline{\theta}$. It is expected that $\mathcal{E}(\underline{\theta})$ is not convex and it presents many local minima and flat regions.

Solving the problem in closed-form is essentially impossible, except for some very limited special cases (linear function)

Therefore we have to resort to an optimization algorithm to find the solution. The most common approach is to use an iterative gradient search.



A gradient search consists in computing the current gradient of the cost function

$$g[k] = \nabla_\theta \mathcal{E}(\theta)\Big|_{\theta=\theta[k]}$$

and take a small step in the direction opposite to it

$$\underline{\theta}[k] = \underline{\theta}[k-1] - \mu[k]\, \underline{g}[k-1] \qquad (\text{see Note } (*))$$

where μ is a speed parameter (stepsize parameter) that can be fixed ($\mu[k]=\mu$) or time-varying.
Starting from an initial value $\theta[0]$, it is expected that the algorithm would head towards the valleys of $\mathcal{E}(\theta)$ and stop when a minimum ($g(k)=0$) has been reached. However a number of issues must be considered:

1) The cost function is in general non convex with possibly many local minima. We usually know almost nothing about $\mathcal{E}(\theta)$ as it depends both on the function $f(x,\theta)$ and the dataset. In the very rare cases of convex cost functions (linear systems, MMSE) the minimum is unique and the gradient algorithm heads towards the bottom of a bowl-shaped $\mathcal{E}(\theta)$.

NOTE 4)

Note that the gradient $g[k-1]$ is computed at the parameters $\underline{\theta}(k-1)$

In some cases it has been proposed to use an "a-posteriori" strategy, i.e. the two-step algorithm:

a) Let $\underline{\theta}'(k) = \underline{\theta}(k-1) - \mu[k]\,\underline{g}(k-1)$ → computed with $\underline{\theta}(k-1)$

b) Recompute the gradient $\underline{g}'(k)$ with $\underline{\theta}'(k)$

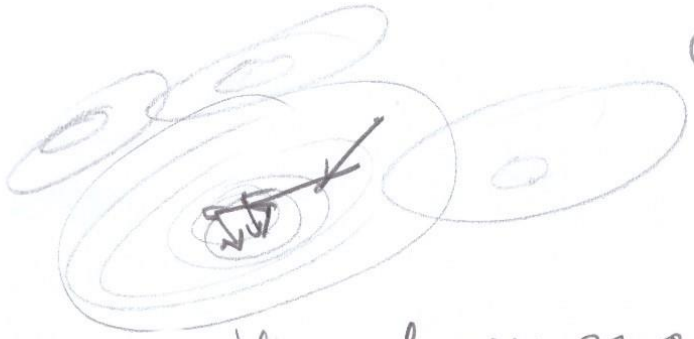
c) Get a new $\underline{\theta}(k)$ as

$$\underline{\theta}(k) = \underline{\theta}(k-1) - \mu\,\underline{g}'[k]$$

This strategy is used sometimes, but its the effectiveness is not always noticeable

In practice, however, vast experience on these kinds of problems, has revealed that $\mathcal{E}(\underline{\theta})$ almost always exhibits many local minima. These minima may correspond to unsatisfactory or satisfactory solutions. Usually the algorithm is re-run various times using different initial conditions, to choose at the end the best one. Ideally, if we had enough time and resources, the algorithm could be run an infinite number of times to reach the global minimum (global convergence). Practically after running the search a limited number of times, we accept the best solution we got. We may never know if a better solution (global?) even exists!

2) The nature of the gradient search is such that the trajectory $\{\underline{\theta}[k]\}$ is attracted towards the bottom of valleys of our cost function. However, if the steps taken are too large, the trajectory may wander around the minimum point(s) and never land on it (there) which would be the condition for null gradient. This phenomenon, known as misadjustment, that happens also for convex cost functions, can be mitigated with an appropriate strategy to control the stepsize parameter $\mu[k]$.



Stepsize vectors are then chosen as a compromise between being small enough to avoid misadjustment and being big enough to avoid slow convergence.
Even if the standard straightforward gradient search is based on a constant stepsize ($\mu[k]=\mu$), a number of strategies have been proposed in the literature to adopt a time-varying $\mu[k]$. Clearly, towards convergence the stepsize should become small to avoid misadjustment, while at the beginning it should be large to quickly head towards relevant valleys, and jump over possible barriers. ($\mu[k] \underset{k\to\infty}{\to} 0$)
Most algorithms proposed in the literature, and specifically for neural networks, are based on experience and good heuristics.
Convergence proofs for various $\mu[k]$ profiles are also a bit debatable because, as we mentioned above, we know very little about the nature of $E(\underline{\theta})$ that depends on $f(\underline{x},\underline{\theta})$ and the dataset.

However we do know from experience that $E(\theta)$ may have large plateaus in some regions of the search space and be very rapidly varying in others.

The best ideas about controlling $\mu[k]$ during search are based on time-varying gradient statistics. In other words, the best algorithms try to control dynamically the speed or convergence of the evolution of the current trajectory history.

The ADAptive Movement estimation algorithm (ADAM)

In machine learning practice, it has become very popular the ADAM algorithm that encompasses most of heuristics mentioned above, and that has shown great success in a large number of applications.

The idea is to maintain a moving average of first (mean) and second (mean square) moments for each element $g_i[k]$ of the vector $\underline{g}[k]$

$$\begin{cases} m_i[k] = \beta_1 m_i[k-1] + (1-\beta_1)\, g_i[k-1] \\ v_i[k] = \beta_2 v_i[k-1] + (1-\beta_2)\, g_i^2[k-1] \end{cases} \qquad 0 < \beta_1, \beta_2 < 1$$

At every time step the moments are bias-corrected

$$\hat{m}_i[k] = \frac{m_i[k]}{1-\beta_1[k]} \qquad \beta_1[k] = \beta_1^k$$

$$\hat{v}_i[k] = \frac{v_i[k]}{1-\beta_2[k]} \qquad \beta_2[k] = \beta_2^k$$

and the parameter update for each parameter θ_i of $\underline{\theta}$ is

$$\theta_i[K] = \theta_i[K-1] - \alpha \frac{\hat{m}_i[K]}{\sqrt{\hat{v}_i[K]} + \varepsilon} \quad \forall i$$

The rule is quite effective because each parameter may show a different convergence behavior and the different stepsize evolution aims at capturing it.

Few comments on ADAM:

Think of parameters in different layers in a large neural network. They may see gradients with very different scales

1) The parameter ε, typically very small, $\varepsilon \simeq 10^{-8}$, is included to avoid possible division by zero during the iterations. This parameter is determined by trial-and-error.

2) The algorithm updates depend on a "smooth" version of the gradient $m[K]$ which is bias-corrected essentially only at the beginning because $\beta_1[K] = \beta_1^K \xrightarrow[K]{} 0$ very quickly as $\beta_1 \simeq 0.9$. (0.9^{20} is already $= 0.12$)

3) α is a fixed stepsize, typical value is 0.001, and the update is also controlled by $\sqrt{\hat{v}_i[K]}$ at the denominator: large $\hat{v}_i[K]$ slows down the search. The bias correction also acts essentially only at the beginning, just as for $\hat{m}_i[K]$, because $\beta_2[K] = \beta_2^K \xrightarrow[K]{} 0$ very quickly.

Parameters $\beta_1, \beta_2, \alpha, \varepsilon$ are usually chosen by the user in the range of the suggested typical values by trial-and-error.

Other search strategies, such as RANDOM SEARCHES (GENETIC ALGORITHMS) can be adopted to learn $f(x, \theta)$.

The interested student can consult the extensive literature on the topic.

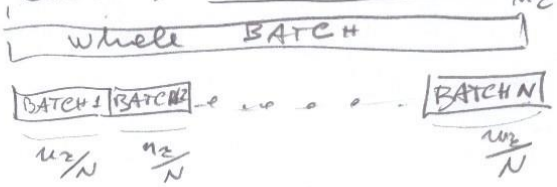
USE OF THE TRAINING SET

The typical flow for a gradient search is a BATCH SEARCH

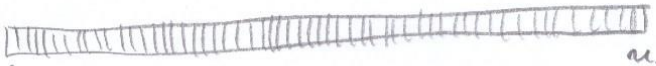
1) Set initial condition for the parameters $\underline{\theta}[0]$
2) Compute the gradient $\nabla_{\theta} E[k]$ on the whole training set $\mathcal{Z}$ (the whole batch)
3) Take a step $\underline{\theta}[k] = \underline{\theta}[k-1] - \mu[k] \nabla_{\theta} E[k]$ (or modified as in ADAM)
4) Check for convergence on coefficients
   $\|\underline{\theta}[k] - \underline{\theta}[k-1]\| < \varepsilon$
   and/or cost function
   $|E(\underline{\theta}[k]) - E(\underline{\theta}[k-1])| < \beta$
   Continue to (2) if not satisfied
   else exit.

As we will see in the following for specific architectures, the computation of the gradient $\nabla_{\theta} E[k]$ on the whole training set may be computationally heavy (Think of a neural network that has to work on images with a large training set)
This may become particularly critical if the search has to be restarted many times to find better local minima, perhaps starting from different (random) initial conditions.

To mitigate this issue often a partioned strategy is adopted dividing the training set in smaller batches MINI-BATCH SEARCH



A search is conducted on a mini batch and the result is used as initial condition on another mini batch. The process can be repeated many times, perhaps changing the scheduling order of the mini batches.
This MINI BATCH search has to advantage of providing intermediate valuable solutions and mitigate the local minimum problem. It is expected that a "noisy" or "imperfect" gradient (computed on subsets of the examples) may help to shake up the search trajectories to jump over local barriers in the cost function.
The limit to the partitioning of the training set, is to consider one-sample-at-a-time



in computing the gradient and performing the update. This is named ON-LINE SEARCH or STOCHASTIC SEARCH. (This is well known in adaptive filtering theory where linear filters are adapted with the LMS algorithm [3])
In all cases the training set is presented multiple times. Each time the set is presented is named EPOQUE.