Optimization in Deep Learning

Jesús Fernández-Villaverde[1] and Galo Nuño[2]

August 7, 2023

[1]University of Pennsylvania

[2]Banco de España

Descent direction iteration

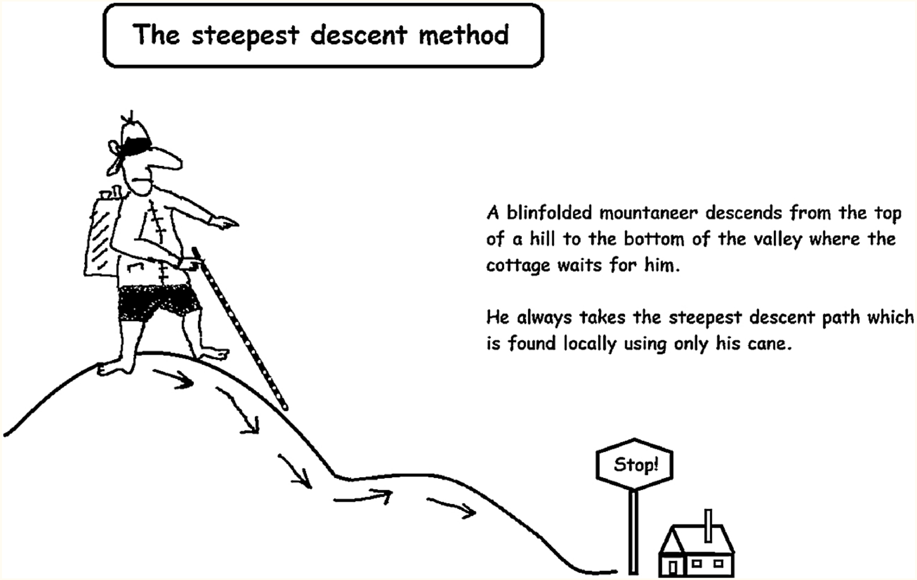
- Most training of neural networks is done with (first-order) descent direction iteration methods.
- Starting at point $\theta^{(1)}$ (determined by domain knowledge), a descent direction algorithm generates sequence of steps (called iterates) that converge to a local minimum.
- The descent direction iteration algorithm:
  1. At iteration k, check whether $\theta^{(k)}$ satisfies termination condition. If so stop; otherwise go to step 2.
  2. Determine the descent direction $\mathbf{d}^{(k)}$ using local information such as gradient or Hessian.
  3. Compute step size $\alpha^{(k)}$.
  4. Compute the next candidate point: $\theta^{(k+1)} \leftarrow \theta^{(k)} + \alpha^{(k)}\mathbf{d}^{(k)}$.
- Choice of α and $\mathbf{d}$ determines the flavor of the algorithm.

Gradient descent method, I

- A natural choice for $\mathbf{d}$ is the direction of steepest descent (first proposed by Cauchy in 1847).
- The direction of steepest descent is given by the direction opposite the gradient $\nabla \mathcal{E}(\theta)$. Thus, a.k.a. steepest descent.
- If function is smooth and the step size small, the method leads to improvement (as long as the gradient is not zero).
- The normalized direction of steepest descent is:

$$\mathbf{d}^{(k)} = -\frac{\nabla \mathcal{E}(\theta^{(k)})}{||\nabla \mathcal{E}(\theta^{(k)})||}$$

Gradient descent method, II



A blinfolded mountaneer descends from the top of a hill to the bottom of the valley where the cottage waits for him.

He always takes the steepest descent path which is found locally using only his cane.

Gradient descent method, III

- One way to set the step size is to solve a line search:

$$\alpha^k = \arg\min_{\alpha} \mathcal{E}(\theta^{(k)} + \alpha \mathbf{d}^{(k)})$$

  for example with the Brent-Dekker method.

- Under this step size choice, it can be shown $\mathbf{d}^{(k+1)}$ and $\mathbf{d}^{(k)}$ are orthogonal.

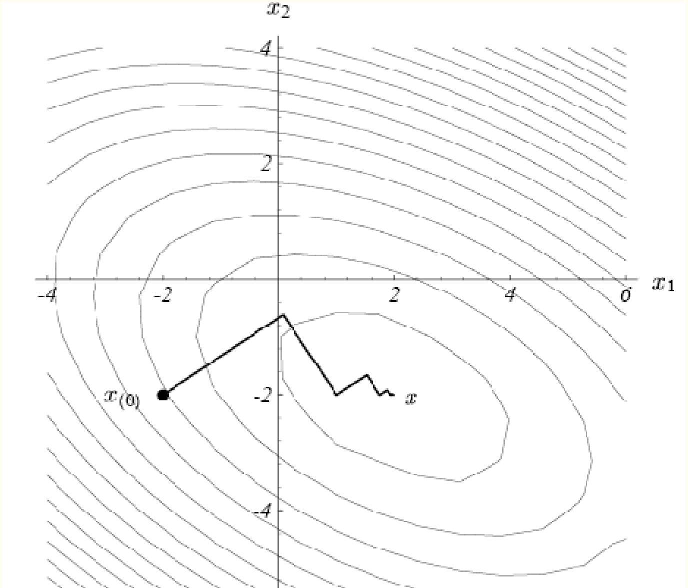
- In practice, line search can be costly and we settle for a fix α, a α^k that geometrically decays, or an approximated line search.

- Trade off between speed of convergence and robustness.

Heard in Minnesota Econ grad student lab

If you do not know where you are going, at least go slowly.

Gradient descent method, IV

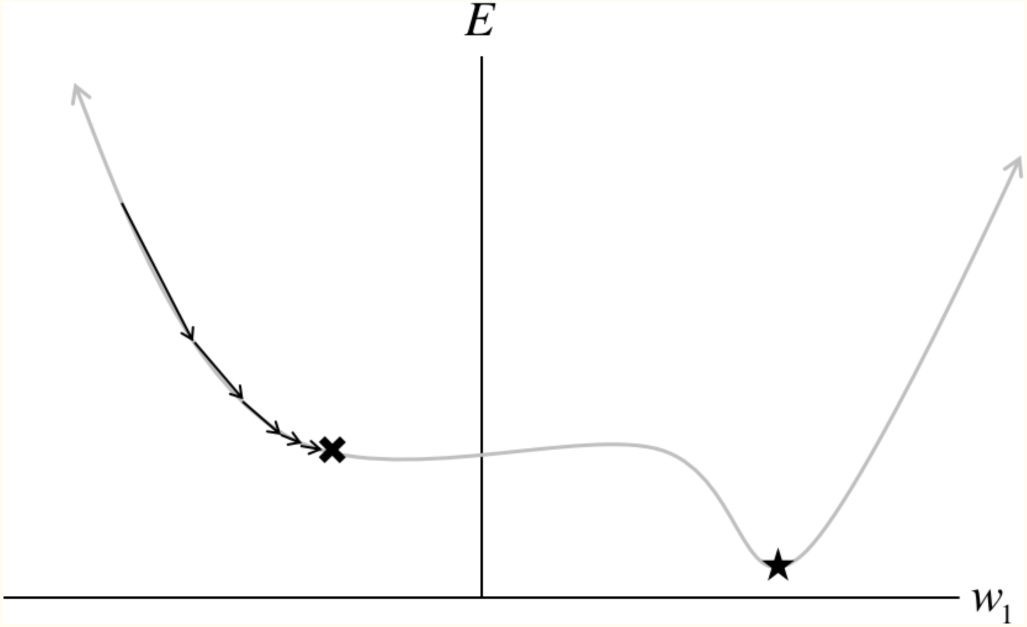


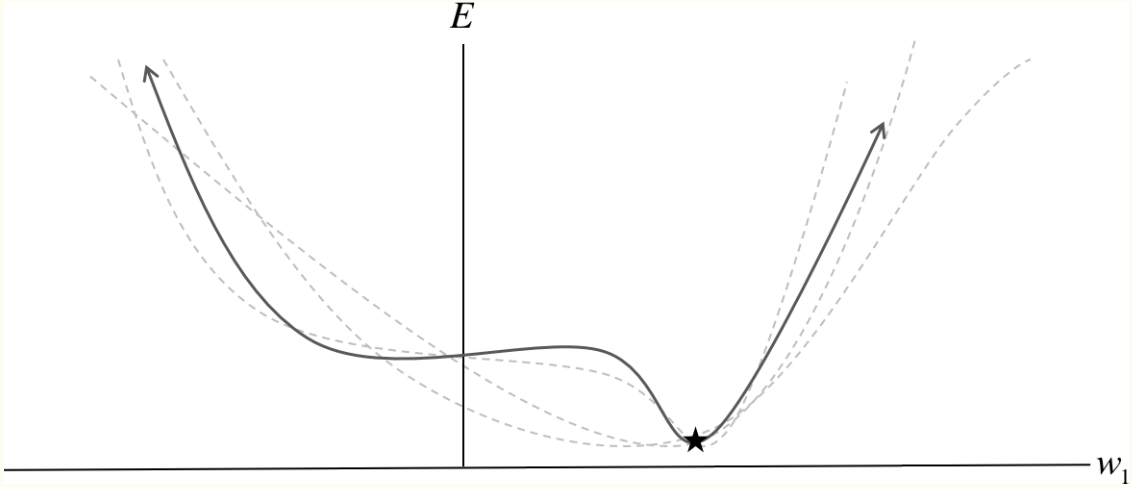
SGD, I

- Even with back propagation, evaluating the gradient for the whole training set can be costly: thousands of points to evaluate!
- Stochastic gradient descent (SGD): We use only one data point to evaluate (an approximation to) the gradient.
- We trade off slower convergence rate for faster computation and early insights in the network behavior.
- Invented by Herbert Robbins and Sutton Monro: A Stochastic Approximation Method (1951).

SGD, II

- Intuition from random algorithms: substitute sure convergence with almost sure convergence (think about Monte Carlo integration vs. quadrature).
- Also, noisy update process can allow the model to avoid local minima (implicit regularization).
- In fact, this feature can be improved using entropy SGD, sharpness aware minimization, and stochastic weight averaging (SWA).


E
w_1


E
w_1

SGD, III

- SGD converges almost surely to a global minimum when the objective function is convex (and to a local minimum otherwise).
- SGD converges exponentially fast to a neighborhood of the solution and, then, bounces around a "zone of confusion."
    - Check `https://fa.bianp.net/blog/2021/exponential-sgd/`.
- SGD can be modeled as a Markov chain with infinite states that makes monotonic progress towards its invariant distribution.
- In practice, we do not need a global min ($\neq$ likelihood). Optimization is not an end in and of itself (also, subtle issue of non-uniqueness when models are over-parametrized).
- You can flush the algorithm to a graphics processing unit (GPU) or a tensor processing unit (TPU) instead of a standard CPU.

SGD, IV

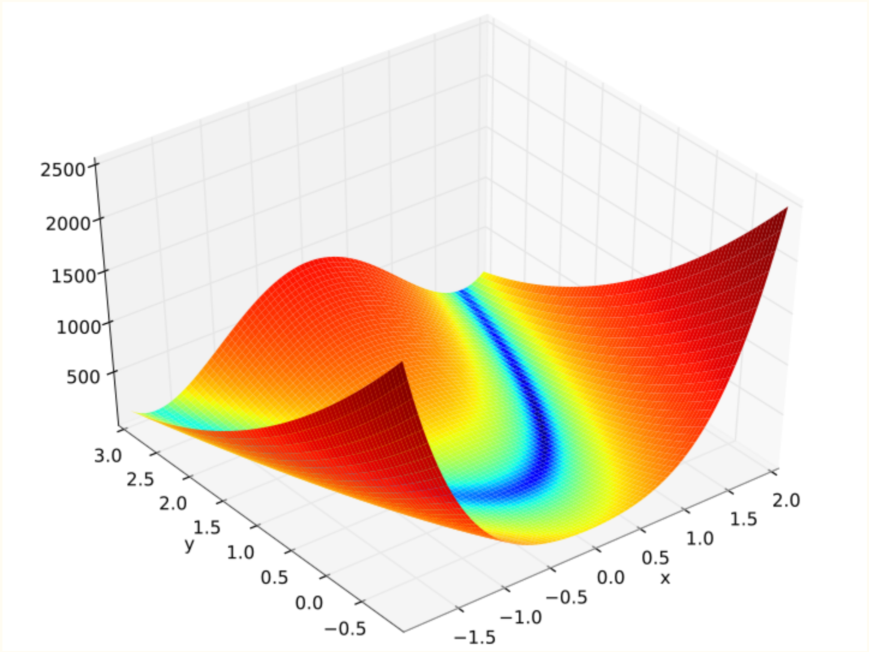
- Example: `https://colab.research.google.com/drive/1oODs4FWpo8rEfHkKn0_8wkOZ6LMejkxL?usp=sharing`.
- Check, for a lot of practical ideas, Stochastic Gradient Descent Tricks, at `https://www.microsoft.com/en-us/research/wp-content/uploads/2012/01/tricks-2012.pdf`.

Minibatch

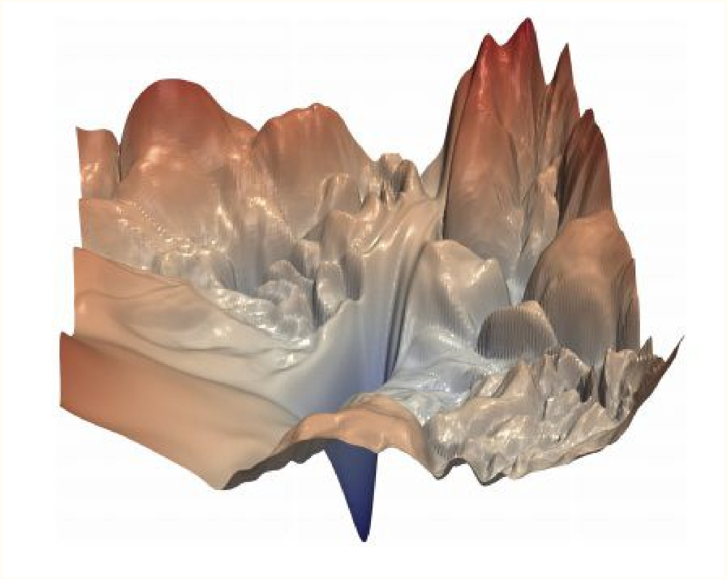
- A compromise between using the whole training set and pure stochastic gradient descent: minibatch gradient descent.
- This is the most popular algorithm to train neural networks.
- Intuition: the standard error of the mean converges slowly ($\sqrt{n}$).
- Also, usually more resilient to scaling of the update.
- Drawback: one more hyperparameter to determine.

Improving gradient descent

- Gradient descent can perform poorly in narrow valleys (it may require many steps to make progress).
- Famous example: Rosenbrock function $\to (a-x)^2 + b(y-x^2)^2$.
- Unfortunately, these are not exotica.
- We are often minimizing over hundreds of thousands of weights.


2500
2000
1500
1000
500
3.0
2.5
2.0
1.5
y
1.0
0.5
0.0
−0.5
−1.5
−1.0
−0.5
0.0
0.5
x
1.0
1.5
2.0

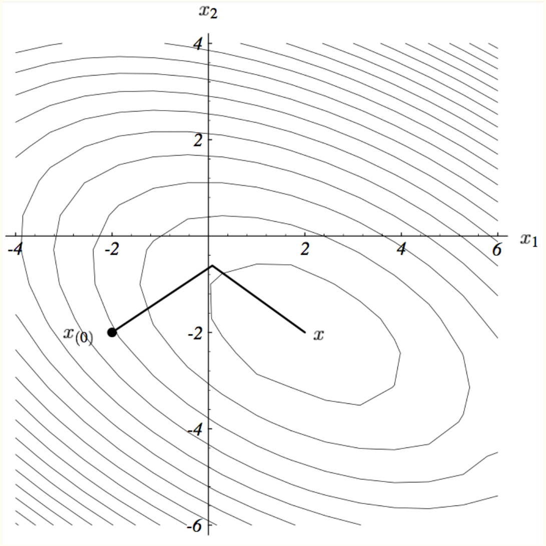
A real example



Conjugate descent method, I

- The *conjugate gradient* method overcomes this problem by constructing a direction conjugate to the old gradient, and to all previous directions traversed.
- Define $g(\theta) = \nabla\mathcal{E}(\theta)$.
- In first iteration, set: $d^{(1)} = -g(\theta^{(1)})$ and $\theta^{(2)} = \theta^{(1)} + \alpha^{(1)}\mathbf{d}^{(1)}$. Here, $\alpha^{(1)}$ is arbitrary.
- Subsequent iterations set $\mathbf{d}^{(\mathbf{k+1})} = -g^{(k+1)} + \beta^{(k)}\mathbf{d}^{(\mathbf{k})}$.

Conjugate descent method, II



Approaches in traditional optimization

- There are two approaches to set β:

  1. Fletcher-Reeves:
  $$\beta^{(k)} = \frac{g^{(k)T} g^{(k)}}{g^{(k-1)T} g^{(k-1)}}$$
  2. Olak-Ribiere:
  $$\beta^{(k)} = \frac{g^{(k)T} (g^{(k)} - g^{(k-1)})}{g^{(k-1)T} g^{(k-1)}}$$

- The Olak-Ribiere requires an automatic reset at every iteration: $\beta \leftarrow \max(\beta, 0)$.

Momentum in optimization

- If the function to minimize has flat areas, one can introduce a *momentum* update equation:

$$v^{(k+1)} = \beta v^{(k)} - \alpha g^{(k)}$$
$$\theta^{(k+1)} = \theta^{(k)} + v^{(k+1)}$$

- The modification reverts to the gradient descent version if $\beta = 0$.
- Intuitively, the momentum update is like a ball rolling down an almost horizontal surface.
- Momentum prevents the ball from getting stuck in a local valley.
- A quick intro: `https://fa.bianp.net/blog/2021/hitchhiker/`.
- A more subtle interpretation: `https://distill.pub/2017/momentum/`.

Adam

- Application to neural network training: *Adam* (Adaptive Moment Estimation), Kingma and Ba (2014).
- It uses running averages of both the gradients and the second moments of the gradients.
- Equations

$$
\begin{aligned}
m^{(k+1)} &= \gamma_1 m^{(k)} + (1-\gamma_1)\nabla\mathcal{E}(\theta^{(k)}) \\
v^{(k+1)} &= \gamma_2 v^{(k)} + (1-\gamma_2)\left(\nabla\mathcal{E}(\theta^{(k)})\right)^2 \\
\widehat{m} &= \frac{m^{(k+1)}}{1-\gamma_1} \\
\widehat{v} &= \sqrt{\frac{v^{(k+1)}}{1-\gamma_2}} \\
\theta^{(k+1)} &= \theta^{(k)} - \eta\frac{\widehat{m}}{\widehat{v}+\epsilon}
\end{aligned}
$$

A few more ideas

1. Random initializations.
2. Multi-starts.
3. Vanishing and exploding gradients.
4. Batch normalization: we normalize the data features, where the variance has some random noise. It deals efficiently with unstable data and avoid saturation of activation functions.
5. Bagging.

Alternative minimization algorithms

1. Second-order methods (e.g., Newton and Quasi-Newton) and direct methods (e.g., Cyclic Coordinate Search and Powell's method) are unlikely to be of much use in practice. Why?

2. McMc/Simulated annealing: probably too slow.

3. Genetic algorithms:

    - In fact, much of the research in deep learning incorporates some flavor of genetic selection.

    - Basic idea.