FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

NATURWISSENSCHAFTLICHE
FAKULTÄT

Stochastic Gradient Descent Methods

Lecture “Mathematics of Learning” 2022/23

Wigand Rathmann
Friedrich-Alexander-Universität Erlangen-Nürnberg

WIRTSCHAFTS
MATHEMATIK

Repetition for a simple training algorithm

Train a artificial neural network:

1. Set a **learning rate** η (e.g., fixed $\eta = 0.001$)
2. **Initialize** all weights and biases of θ_0 **randomly**, e.g., vector θ_0 normally distributed
3. Repeat until **no significant improvement** is achieved
   a) Loop over all pairs $(x^{(i)}, y^{(i)})$ $(i = 1, ..., N)$ in training set
      i) Compute a forward pass of the neural network $f_{\theta^k}(x^{(i)})$ and store intermediate results
      ii) Use intermediate results of forward pass and backpropagation to compute the gradient $\nabla C(\theta^k; x^{(i)})$
   b) compute average $\nabla \overline{C}(\theta^k)$ of all N gradients $\nabla C(\theta^k; x^{(i)})$
   c) update parameters using **gradient descent**:
   $\theta^{k+1} = \theta^k - \eta \nabla \overline{C}(\theta^k)$

But:

This is computationally expensive for a large training data set ($N \gg 1$)!

Repetition: Backpropagation

Consider loss function for single data point and mean squared loss function

$$C(\theta; x^{(i)}) = \frac{1}{2}||f_\theta(x^{(i)}) - y^{(i)}||_2^2$$

$$\overline{C}(\theta) = \frac{1}{2N}\sum_{i=1}^{N}||f_\theta(x^{(i)}) - y^{(i)}||_2^2$$

For every single data point (x, y):
Forward passes: Compute $a_i^{(L)}$ and $z_i^{(L)}$ in every node i in layer L (note that $z_i^{(L)} = \psi_i^{(L)}(a_i^{(L)})$)
Backpropagation: Compute partial derivatives for ∇C in a recursive way:

- Last layer: $\frac{\partial C}{\partial b_i^{(d)}} = (z_i^{(d)} - y_i) \cdot (\psi_i'^{(d)}(a_i^{(d)}))$
- Recursively all partial derivative for the biases:
  $$\frac{\partial C}{\partial b_j^{(L-1)}} = \left(\sum_{i=1}^{n_L} \frac{\partial C}{\partial b_i^{(L)}} \cdot w_{ij}^{(L)}\right) \cdot \psi_j'^{(L-1)}(a_j^{(L-1)})$$
- Recursively all partial derivative for the weights: $\frac{\partial C}{\partial w_{ij}^{(L)}} = \frac{\partial C}{\partial b_i^{(L)}} \cdot z_j^{(L-1)}$

Universal Approximation in Practice?

Theorem

Neural networks with discriminatory activation functions and sufficiently many neurons or layers can approximate any given function.

Problem: How do we compute such networks? A (partial) answer is stochastic gradient descent:

- Take a training set $T \subset X \times Y$, where X and Y are in and output spaces.
- (Choose a network architecture $f_\theta : X \to Y$ with parameters θ).
- Choose loss function $L : Y \times Y \to \mathbb{R}$ and update parameters θ via stochastic gradient descent (SGD)

$$\begin{aligned} &(1) \quad \text{randomly draw batch } B \text{ from } T \\ &(2) \quad \theta \leftarrow \theta - \eta \frac{1}{|B|} \sum_{(x,y)\in B} \nabla_\theta L(f_\theta(x), y), \quad \text{go back to (1)} \end{aligned}$$

- SGD analysis goes back to more general result by Robbins and Monro, 'A Stochastic Approximation Method,' The Annals of Mathematical Statistics, 1951, 400–407.

Risk Minimization

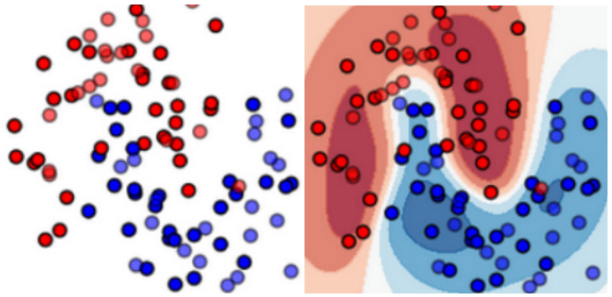
Stochastic gradient descent aims to minimize the empirical risk of f_θ on the training set:

$$R(\theta) = \frac{1}{|T|} \sum_{(x,y)\in T} L(f_\theta(x), y).$$

This is an approximation of the non-accessible population risk

$$\mathcal{R}(f_\theta) = \iint_{X\times Y} L(f_\theta(x), y)\,\mathrm{d}\rho(x, y)$$

if the training set T is sampled from a probability distribution on $X \times Y$ with density ρ.



Stochastic Gradient Descent (SGD)

One of the most popular algorithms for contemporary data analysis!
Consider fixed network and training set, consider stochastic gradient descent

$$\begin{aligned}&(1)\quad \text{randomly draw batch } B \text{ from } T\\ &(2)\quad \theta \leftarrow \theta - \eta\frac{1}{|B|}\sum_{(x,y)\in B}\nabla_\theta L(f_\theta(x),y), \quad \text{go back to (1).}\end{aligned}$$

We note: This batch can be 'everything' from $|B| = 1$ (then one gradient is randomly chosen in each step) up to all data points in training set.
More abstractly we study

$$\begin{aligned}&(1)\quad \text{sample gradient estimator } g_k\\ &(2)\quad \theta_{k+1} \leftarrow \theta_k - \eta_k g_k,\\ &(3)\quad k \leftarrow k+1, \quad \text{go back to (1),}\end{aligned}$$

Abstract Analysis of SGD

More abstractly we study

$$\begin{array}{ll} (1) & \text{sample gradient estimator } g_k \\ (2) & \theta_{k+1} \leftarrow \theta_k - \eta_k g_k, \\ (3) & k \leftarrow k+1, \quad \text{go back to (1)}, \end{array}$$

where g_k is an unbiased gradient estimator of an arbitrary loss function $\mathcal{L}$. This means:

$$\mathbb{E}\left[g_k\right] = \nabla \mathcal{L}(\theta_k),$$

In addition, we need to be able to finitely bound the variance of the estimation.

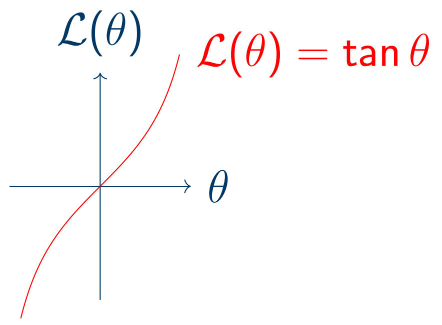
$$\mathbb{E}\left[\|g_k - \nabla\mathcal{L}(\theta_k)\|^2\right] \leq \sigma^2 \left[\iff \mathbb{E}\left[\|g_k\|^2\right] \leq \sigma^2 + \|\nabla\mathcal{L}(\theta_k)\|^2\right]$$

Bounding the variance is not so difficult: If B is whole training set, variance is zero. It is large (but finite) if $|B| = 1$.

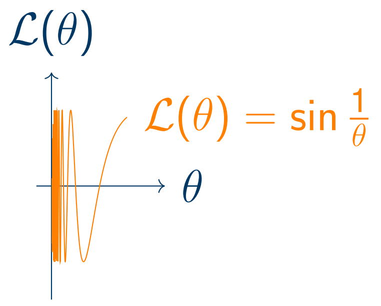
Conditions on Loss Function

We can only expect convergence for “reasonable” loss functions $\mathcal{L}$ and have to exclude situations like the following:

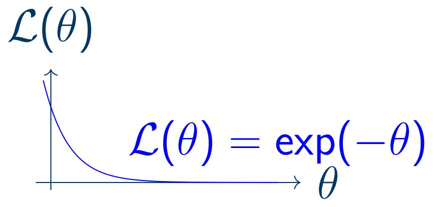
No minima, not bounded from below:



Too oscillatory:

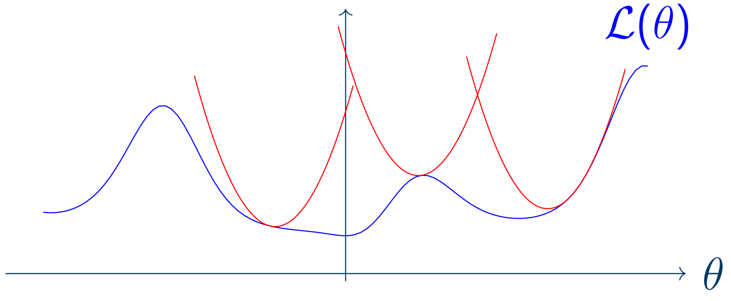


No parameter convergence:



Our Assumptions on the Loss Function

Assumption 1: The loss is bounded from below, i.e., $\mathcal{L}(\theta) \geq \mathcal{L}_*$ for all θ.
Assumption 2: The loss is L-smooth, i.e., $\|\nabla\mathcal{L}(\theta) - \nabla\mathcal{L}(\tilde{\theta})\| \leq L\|\theta - \tilde{\theta}\|$ for all $\theta, \tilde{\theta} \in \theta$.
(This forbids strong oscillations. L is Lipschitz constant, see Analysis.) Using Taylor, it follows
$\mathcal{L}(\theta) \leq \mathcal{L}(\tilde{\theta}) + \langle\nabla\mathcal{L}(\tilde{\theta}), \theta - \tilde{\theta}\rangle + \frac{L}{2}\|\theta - \tilde{\theta}\|^2$



Convergence to Critical Loss, Part 1

Using that $\mathcal{L}$ is L-smooth and the SGD update $\theta_k - \theta_{k+1} = \eta_k g_k$

$$\mathcal{L}(\theta_{k+1}) \leq \mathcal{L}(\theta_k) + \langle \nabla \mathcal{L}(\theta_k), \theta_{k+1} - \theta_k \rangle + \frac{L}{2} \|\theta_{k+1} - \theta_k\|^2$$
$$= \mathcal{L}(\theta_k) - \eta_k \langle \nabla \mathcal{L}(\theta_k), g_k \rangle + \eta_k^2 \frac{L}{2} \|g_k\|^2 .$$

Taking expectations yields ($\mathbb{E}[g_k] = \nabla \mathcal{L}(\theta_k)$)
$\mathbb{E}[\mathcal{L}(\theta_{k+1})] \leq \mathcal{L}(\theta_k) - \eta_k \|\nabla \mathcal{L}(\theta_k)\|^2 + \eta_k^2 \frac{L}{2} \left(\sigma^2 + \|\nabla \mathcal{L}(\theta_k)\|^2 \right)$ reshuffling terms in this equation leads to: $\eta_k \left(1 - \eta_k \frac{L}{2}\right) \|\nabla \mathcal{L}(\theta_k)\|^2 \leq \mathcal{L}(\theta_k) - \mathbb{E}[\mathcal{L}(\theta_{k+1})] + \eta_k^2 \frac{L}{2} \sigma^2$
We are free in how to choose learning rate η. (Goal: decrease expected value of loss!) Good choice:

Approximate loss decrease

Insert $\eta_k < \frac{2}{L}$ as learning rate. Then the loss decreases up to noise variance:

$$\mathbb{E}[\mathcal{L}(\theta_{k+1})] \leq \mathcal{L}(\theta_k) + \eta_k \sigma^2$$

I.e., approximate loss decrease depends on variance (Assumption 2).
Summing up to K using $\mathcal{L} \geq \mathcal{L}_*$ and taking expectations, this leads to

Convergence to Critical Loss, Part 2

Estimate from the last slide:

$$\sum_{k=1}^{K} \underbrace{\eta_k \left(1 - \eta_k \tfrac{L}{2}\right)}_{\geq c\eta_k \text{ if } \eta_k < \frac{2}{L}} \mathbb{E}\left[\|\nabla \mathcal{L}(\theta_k)\|^2\right] \leq \mathcal{L}(\theta_1) - \mathcal{L}_* + \frac{L}{2}\sigma^2 \sum_{k=1}^{K} \eta_k^2.$$

Hence for $\eta_k < \frac{2}{L}$ we obtain

$$\min_{k=1,\dots,K} \mathbb{E}\left[\|\nabla \mathcal{L}(\theta_k)\|^2\right] c \sum_{k=1}^{K} \eta_k \leq \mathcal{L}(\theta_1) - \mathcal{L}_* + \frac{L}{2}\sigma^2 \sum_{k=1}^{K} \eta_k^2$$

this means: for convergence to a critical point, the right-hand side necessarily has to be finite. In addition, in the left-hand side, the sum of the learning rates necessarily has to diverge as otherwise the gradients would not necessarily have to converge to a critical point.
In formulas: For convergence as $K \to \infty$ we need that

$$\sum_{k=1}^{\infty} \eta_k = \infty \qquad \text{and} \qquad \sum_{k=1}^{\infty} \eta_k^2 < \infty.$$

Choice of Learning Rate

We have the following estimate if $\eta_k < \frac{2}{L}$:

$$\min_{k=1,\dots,K} \mathbb{E}\left[\|\nabla\mathcal{L}(\theta_k)\|^2\right] c \sum_{k=1}^{K} \eta_k \le \mathcal{L}(\theta_1) - \mathcal{L}_* + \frac{L}{2}\sigma^2 \sum_{k=1}^{K} \eta_k^2$$

For diminishing learning rate $\eta_k = \frac{1}{Lk}$ we obtain (from partial sums of harmonic series)

$$\sum_{k=1}^{K} \eta_k = O(\log K), \qquad \sum_{k=1}^{K} \eta_k^2 = O(1),$$

Convergence rate for large K: $\underbrace{O\left(\frac{1}{\log K}\right)}_{\text{quite slow}}$.

If one chooses constant learning rate $\eta_k = \eta < \frac{2}{L}$ one gets

$$\sum_{k=1}^{K} \eta_k = K\eta, \quad \sum_{k=1}^{K} \eta_k^2 = K\eta^2,$$

Convergence rate for large K: $\underbrace{O\left(\frac{1}{\eta K}\right) + \sigma^2 O(\eta)}_{\substack{\text{quicker convergence “up to noise”} \\ \text{trade-off in } \eta}}$

Summary of What We Have Learned

Convergence of SGD

Assume that the learning rate meets $\eta_k < \frac{2}{L}$:

- The expected loss decays up to the noise level: $\mathbb{E}\left[\mathcal{L}(\theta_{k+1})\right] \leq \mathcal{L}(\theta_k) + \eta_k \sigma^2$.
- If the step size decays with the number of iterations such that $\sum_k \eta_k = \infty$ and $\sum_k \eta_k^2 < \infty$, then a subsequence of the iterates converges to critical loss in expectation: $\min_{k=1,\dots,K} \mathbb{E}\left[\|\nabla\mathcal{L}(\theta_k)\|^2\right] \to 0$ as $K \to \infty$.
- As a consequence, we can get convergence in, depending on the step sizes, e.g., $O(1/\log K)$ if $\eta_k \asymp 1/k$.

Q: Can we get more? For instance:

- Better convergence rates for $\mathbb{E}\left[\|\nabla\mathcal{L}(\theta_k)\|^2\right]$
- Convergence rates for $\mathcal{L}(\theta_k) - \mathcal{L}(\theta_*)$ where θ_* is a global minimum of $\mathcal{L}$
- **Most importantly:** Convergence of the iterates θ_k to θ_*

A: Yes, if we have (strong) convexity of $\mathcal{L}$. [Exercises]

Algorithmic Tricks in Practice

from new book 'Patterns, Predictions, and Actions A story about machine learning' Moritz Hardt, Benjamin Recht (https://mlstory.org)

- **Step size selection.** Instead of stepsize $\frac{1}{k}$, a rule that works for an unreasonable number of cases: Pick largest step size which does not result in divergence, then slowly reduce the step size from this initial large step size.
- **Step decay.** The step size is usually reduced after a fixed number of passes over the training data. (an epoch). A common strategy is to run with a constant step size for some fixed number of iterations, and then reduce the step size by a constant factor γ. Thus, if our initial step size is α, on the k-th epoch, the step size is $\alpha\gamma^{k-1}$. Reasonable: choose γ between 0.8 and 0.9.
- **Adagrad.** Adapts learning rate to the parameters, performing lower learning rates for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. Is well-suited for dealing with sparse data. Greatly improves robustness of SGD, e.g. for training large-scale neural nets at Google (learned to recognize cats in Youtube videos), & train GloVe word embeddings. Infrequent words require larger updates than frequent ones.

- **Minibatching.** Take average of many stochastic gradients. Suppose at each iteration we sample a batch with m data points, see earlier slides with batch $B, |B| > 1$. Minibatching reduces the variance of the stochastic gradient estimate of the true gradient, and hence tends to be a better descent direction. However, computation time increases.
- **Momentum.** Mix current gradient direction with the previously taken step. Idea: if the previous weight update was good, we may want to continue moving along this direction. The algorithm iterates: $w_{k+1} = w_k - \eta_k g_k + \beta(w_k - w_{k-1})$, where g_k denotes a stochastic gradient. In practice, very successful. β e.g., between 0.8 and 0.95. Can provide significant accelerations!

The SGD quick start guide

simple rules of thumb to get going:

1. Pick as large a minibatch size as you can given your computer's RAM.
2. Set your momentum parameter to either 0 or 0.9. Your call!
3. Find the largest constant stepsize such that SGD doesn't diverge. This takes some trial and error, but you only need to be accurate to within a factor of 10 here.
4. Run SGD with this constant stepsize until the empirical risk plateaus.
5. Reduce the stepsize by a constant factor (say, 10)
6. Repeat steps 4 and 5 until you converge.

'It's a great starting point and is good enough for probably 90% of applications we've encountered.'