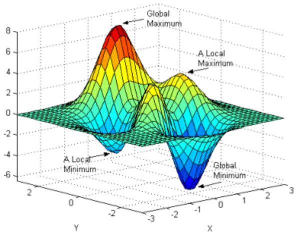
Deep Learning

Basic Training



Learning goals

- Empirical risk minimization
- Gradient descent
- Stochastic gradient descent

TRAINING NEURAL NETWORKS

- In ML we use **empirical risk minimization** (ERM) to minimize prediction losses over the training data

$$\mathcal{R}_{\text{emp}} = \frac{1}{n} \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)$$

- In DL, $\boldsymbol{\theta}$ represents the weights (and biases) of the NN.
- Often, L_2-loss in regression:

$$L\left(y, f(\mathbf{x})\right) = \frac{1}{2}(y - f(\mathbf{x}))^2$$

- or cross-entropy for binary classification

$$L\left(y, f(\mathbf{x})\right) = -(y \log f(\mathbf{x}) + (1 - y) \log(1 - f(\mathbf{x})))$$

- ERM can be implemented by **gradient descent** (GD).

GRADIENT DESCENT

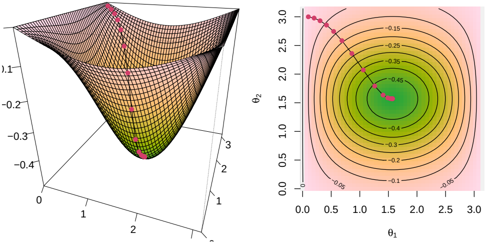
- Neg. risk gradient points in the direction of the **steepest descent**

$$-\mathbf{g} = -\nabla \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = -\left(\frac{\partial \mathcal{R}_{\text{emp}}}{\partial \theta_1}, \ldots, \frac{\partial \mathcal{R}_{\text{emp}}}{\partial \theta_d}\right)^\top$$

- "Standing" at a point $\boldsymbol{\theta}^{[t]}$, we locally improve by:

$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \alpha \mathbf{g},$$

- α is called **step size** or **learning rate**.

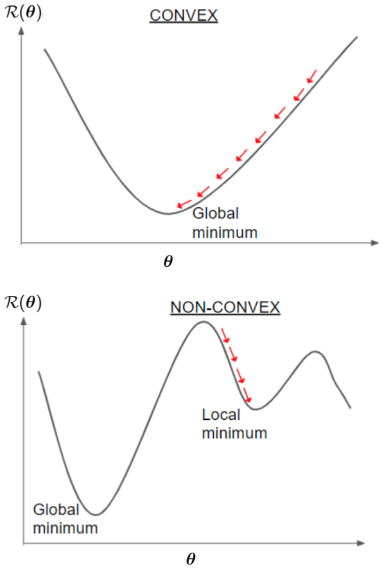


GRADIENT DESCENT AND OPTIMALITY

- GD is a greedy algorithm: In every iteration, it makes locally optimal moves.

- If $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$ is **convex** and **differentiable**, and its gradient is Lipschitz continuous, GD is guaranteed to converge to the global minimum (for small enough step-size).

- However, if $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$ has multiple local optima and/or saddle points, GD might only converge to a stationary point (other than the global optimum), depending on the starting point.



GRADIENT DESCENT AND OPTIMALITY

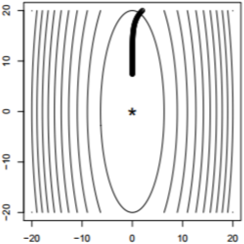
Note: It might not be that bad if we do not find the global optimum:

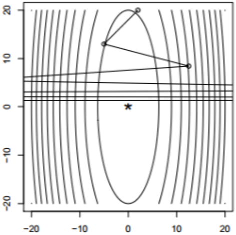
- We don’t optimize the (theoretical) risk, but only an approximate version, i.e. the empirical risk.
- For very flexible models, aggressive optimization might overfitting.
- Early-stopping might even increase generalization performance.

LEARNING RATE (LR)

The step-size α plays a key role in the convergence of the algorithm.

If the step size is too small, the training process may converge **very** slowly (see left image). If the step size is too large, the process may not converge, because it **jumps** around the optimal point (see right image).



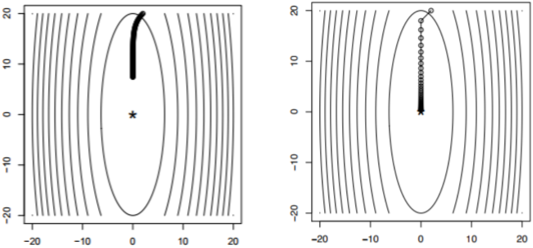


LEARNING RATE

So far we have assumed a fixed value of α in every iteration:

$$\alpha^{[t]} = \alpha \quad \forall t = \{1, \ldots, T\}$$

However, it makes sense to adapt α in every iteration:



Steps of gradient descent for $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = 10\,\theta_1^2 + 0.5\,\theta_2^2$. Left: 100 steps for with a fixed learning rate. Right: 40 steps with an adaptive learning rate.

WEIGHT INITIALIZATION

- Weights (and biases) of an NN must be initialized in GD.
- We somehow must "break symmetry" – which would happen in full-0-initialization. If two neurons (with the same activation) are connected to the same inputs and have the same initial weights, then both neurons will have the same gradient update and learn the same features.
- Weights are typically drawn from a uniform a Gaussian distribution (both centered at 0 with a small variance).
- Two common initialization strategies are ’Glorot initialization’ and ’He initialization’ which tune the variance of these distributions based on the topology of the network.

STOCHASTIC GRADIENT DESCENT (SGD)

GD for ERM was:

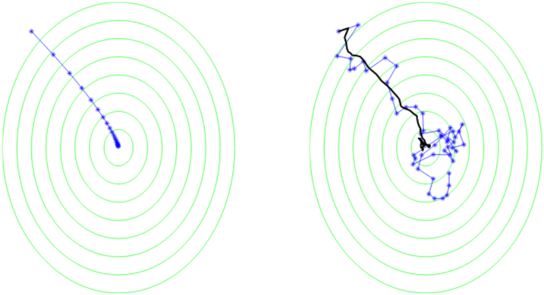
$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \alpha \cdot \frac{1}{n} \cdot \sum_{i=1}^{n} \nabla_\theta L\left(y^{(i)}, f(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}^{[t]})\right)$$

- Using the entire training set in GD to is called **batch** or **deterministic** or **offline training**. This can be computationally costly or impossible, if data does not fit into memory.
- **Idea:** Instead of letting the sum run over the whole dataset, use small stochastic subsets (**minibatches**), or only a single $\mathbf{x}^{(i)}$.
- If batches are uniformly sampled from $\mathcal{D}_{\text{train}}$, our stochastic gradient is in expectation the batch gradient $\nabla_\theta \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$.
- The gradient w.r.t. a single $\mathbf{x}$ is fast to compute but not reliable. But its often used in a theoretical analysis of SGD.

→ We have a **stochastic**, noisy version of the batch GD.

STOCHASTIC GRADIENT DESCENT (SGD)

SGD on function $1.25(x_1 + 6)^2 + (x_2 - 8)^2$.



Source : Shalev-Shwartz and Ben-David. Understanding machine learning: From theory to algorithms. Cambridge University Press, 2014.

Figure: Left = GD, right = SGD. Black line is an average of different SGD runs.

STOCHASTIC GRADIENT DESCENT

Algorithm Basic SGD pseudo code

```
Initialize parameter vector $\boldsymbol{\theta}^{[0]}$
$t \leftarrow 0$
**while** stopping criterion not met **do**
    Randomly shuffle data and partition into minibatches $J_1, ..., J_K$ of size $m$
    **for** $k \in \{1, ..., K\}$ **do**
        $t \leftarrow t + 1$
        Compute gradient estimate with $J_k$: $\hat{g}^{[t]} \leftarrow \frac{1}{m} \sum_{i \in J_k} \nabla_\theta L(y^{(i)}, f(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}^{[t-1]}))$
        Apply update: $\boldsymbol{\theta}^{[t]} \leftarrow \boldsymbol{\theta}^{[t-1]} - \alpha \hat{g}^{[t]}$
    **end for**
**end while**
```

- A full SGD pass over data is an **epoch**.
- Minibatch sizes are typically between 50 and 1000.

STOCHASTIC GRADIENT DESCENT

- SGD is the most used optimizer in ML and especially in DL.
- We usually have to add a considerable amount of tricks to SGD to make it really efficient (e.g. momentum). More on this later.
- SGD with (small) batches has a high variance, although is unbiased. Hence, the LR α is smaller than in the batch mode.
- When LR is slowly decreased, SGD converges to local minimum.
- Recent results indicate that SGD often leads to better generalization than GD, and may result in indirect regularization.