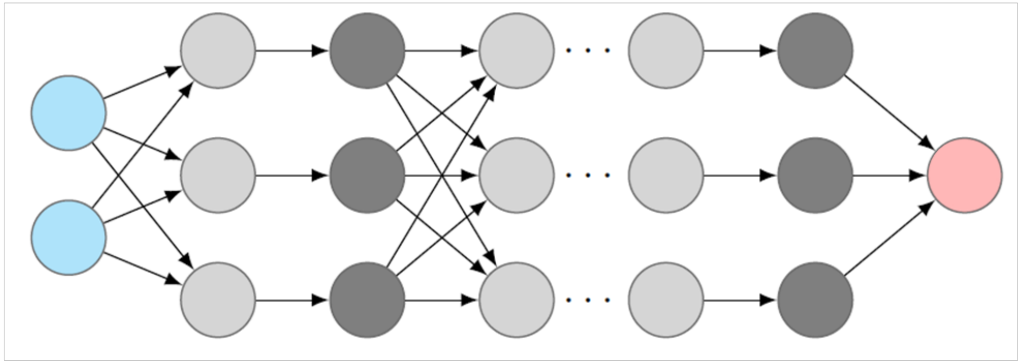
Training neural networks

- SGD + momentum, Adam -

Lecturer: Michael Möller – michael.moeller@uni-siegen.de
Exercises: Hartmut Bauermeister – hartmut.bauermeister@uni-siegen.de



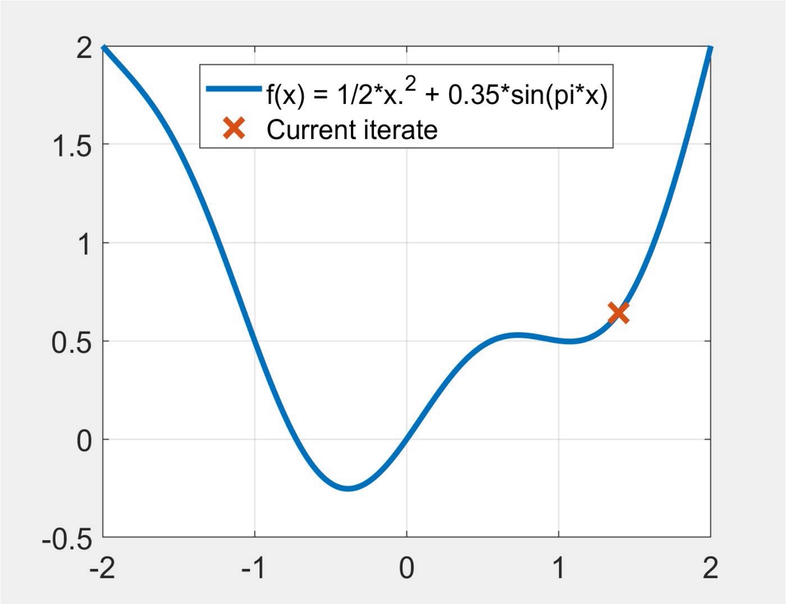
Stochastic gradient descent (SGD)

Remember ***SGD***: Use only a few summands to compute an approximate gradient!

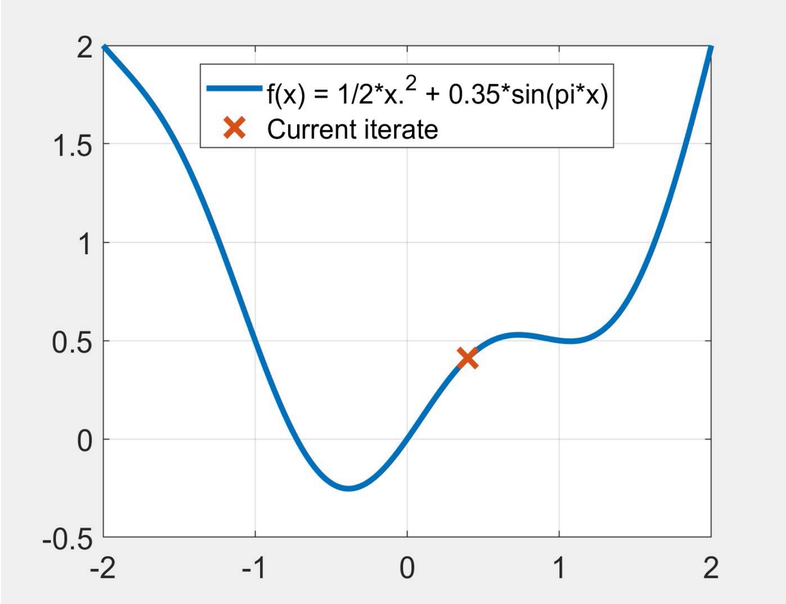
$$\theta(k+1) = \theta(k) - \tau \nabla E_k(\theta(k)) \qquad E_k(\theta) = \sum_{j \in I(k)} \mathcal{L}(\mathcal{N}(x_j, \theta), y_j)$$

-> Minibatch

Gradient descent



Gradient descent with *Momentum*



Michael Möller – michael.moeller@uni-siegen.de

GD with Nesterov Momentum

1) Where does it come from?

The version we discuss here is based on a work by *Yuri Nesterov* from 1983. He showed that for a convex (Lipschitz continuously differentiable) objective function one can obtain a convergence speed of

$$E(\theta(k)) - \min_{\theta} E(\theta) \leq \mathcal{O}(1/k^2)$$

which he proved to be optimal without further assumption on the energy!

2) How does it look in deep learning applications?

$$\theta(k+1) = \theta(k) - \tau(k)\nabla E(\theta(k)) + \alpha * v(k+1)$$

Gradient descent + additional velocity

$$v(k+1) = \alpha \cdot v(k) - \tau(k)\nabla E(\theta(k))$$

- $v(k+1)$: new velocity
- α: damping with $\alpha < 1$, could be interpreted as *friction*
- $v(k)$: old velocity
- $\tau(k)\nabla E(\theta(k))$: accumulating gradients, i.e. ``speeds + directions''

SGD with Nesterov Momentum

3) How does it look in a stochastic deep learning setting?

$$\theta(k+1) = \theta(k) - \tau(k)\nabla E_k(\theta(k)) + \alpha * v(k+1)$$

Stochastic gradient descent | + additional velocity

$$v(k+1) = \alpha \cdot v(k) - \tau(k)\nabla E_k(\theta(k))$$

new velocity | old velocity | accumulating approximate gradients

damping with $\alpha < 1$, could be interpreted as *friction*

Continuous interpretation https://arxiv.org/pdf/1503.01243.pdf (e.g. justifies ‘friction’)

Nice illustrations for why this works well https://distill.pub/2017/momentum/

Detailed theoretical convergence analysis in the deep learning setup – open problem!

Michael Möller – michael.moeller@uni-siegen.de

Adam

SGD + Nesterov Momentum is one of the two most popular methods.

Possibly slightly more popular: Adam.

It combines some techniques we have seen before.

A notion of velocity:

convex combination

$$v(k+1) = (\beta_1 \cdot v(k) - (1-\beta_1) \cdot \nabla E_k(\theta(k)))$$

new velocity

old velocity

Stochastic approximate gradient

Adam

SGD + Nesterov Momentum is one of the two most popular methods.

Possibly slightly more popular: Adam.

It combines some techniques we have seen before.

A notion of velocity:

$$v(k+1) = \frac{1}{1-(\beta_1)^k} \cdot \quad (\beta_1 \cdot v(k) - (1-\beta_1) \cdot \nabla E_k(\theta(k)))$$

Keeping track of the norm of the stochastic gradients (similar to AdaGrad):

$$c(k+1) = \qquad\qquad \cdot \quad \left(\beta_2 c(k) + (1-\beta_2)\ (\nabla E_k(\theta^k) \odot \nabla E_k(\theta^k))\right)$$

convex combination

element-wise product

Adam

SGD + Nesterov Momentum is one of the two most popular methods.

Possibly slightly more popular: Adam.

It combines some techniques we have seen before.

A notion of velocity:

$$v(k+1) = \frac{1}{1-(\beta_1)^k} \cdot \quad \left(\beta_1 \cdot v(k) - (1-\beta_1) \cdot \nabla E_k(\theta(k))\right)$$

Keeping track of the norm of the stochastic gradients (similar to AdaGrad):

$$c(k+1) = \frac{1}{1-(\beta_2)^k} \cdot \quad \left(\beta_2 c(k) + (1-\beta_2)\left(\nabla E_k(\theta^k) \odot \nabla E_k(\theta^k)\right)\right)$$

Move into the velocity direction with a vector-valued stepsize similar to AdaGrad:

$$\theta(k+1) = \theta(k) - \tau(k) \cdot d(k+1) \odot v(k+1), \qquad d(k+1) = (c(k+1)+\epsilon).^{\wedge(-1/2)}$$

element-wise product

element-wise 1/sqrt

In practice

For training a neural network:

- Your **default** choice should be **Adam** with $\beta_1 = 0.9$ and $\beta_2 = 0.999$
- Try $\tau(1) = 10^{-3}$, if this seems to make no progress increase by factors of 10, if you get oscillating, or exploding results, or quick plateaus reduce by factors of 10.
- Consider reducing the learning rate over time (more on the next slide).
- If Adam does not yield a good decay, switch to SGD + Nesterov Momentum.
- Normalize your energy w.r.t. the number of training examples:

$$E(\theta) = \frac{1}{N}\sum_{j=1}^{N} \mathcal{L}(\mathcal{N}(x_j;\theta), y_j)$$

N: number of training examples

$$E_k(\theta) = \frac{1}{|I(k)|}\sum_{j\in I(k)} \mathcal{L}(\mathcal{N}(x_j,\theta), y_j)$$

$|I(k)|$: size of the mini-batch

Learning rate selection

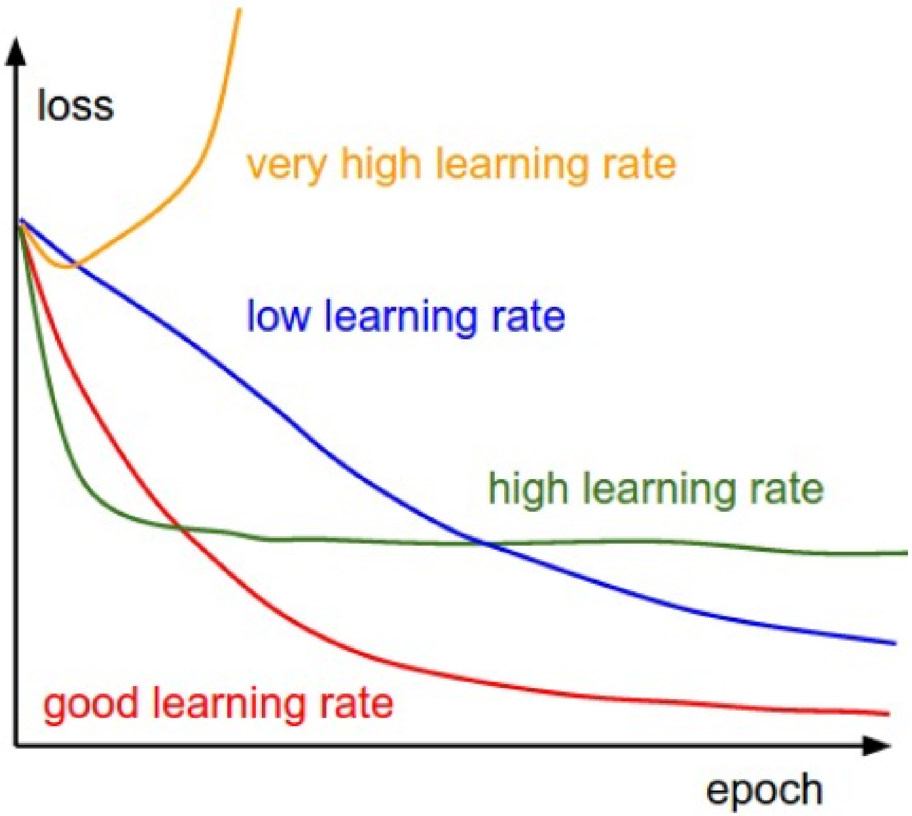


Image taken from http://cs231n.github.io/neural-networks-3/#annealing-the-learning-rate

Usually, it is good to pick a decaying learning rate, e.g. $\tau(k) \sim 1/k$

The exact schedule, particularly the initial rate, are typically trial-and-error!

There are more aggressive variants, e.g., periodic restarts https://arxiv.org/abs/1608.(

Second order methods in 1 slide

Looking at the update of Adam

$$\theta(k+1) = \theta(k) - \tau(k) \cdot d(k+1) \odot v(k+1),$$

one could consider general techniques

$$\theta(k+1) = \theta(k) - D(k+1) \cdot v(k+1),$$

for a matrix $D(k+1)$ (where Adam chose a specific diagonal matrix in this setting).

Numerous optimization methods consider the *Hessian* $D(k+1) = (\nabla^2 E(\theta(k)))^{-1}$

e.g. the famous *Newton method*, or approximations thereof.

We will not cover these techniques. Typically, they only work in small scale setting, i.e. *if the training data is small enough to be handled full batch*. Stochastic settings do not seem to benefit greatly from this *second order information*.

Up next: How to initialize the weights for training!