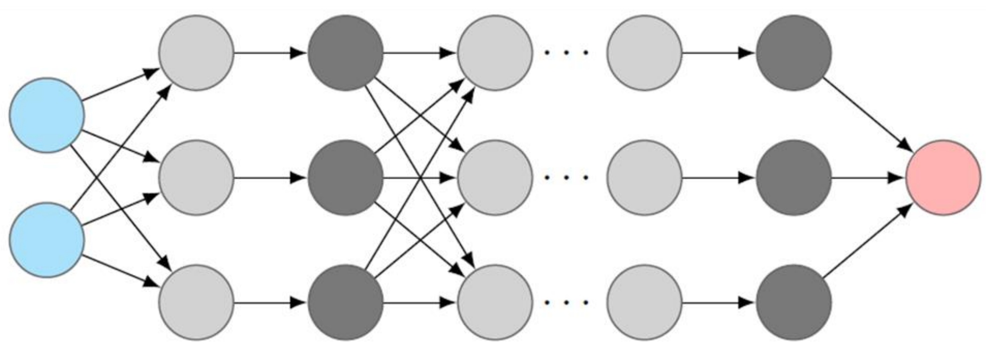
Training fully connected networks

- Stochastic Gradient Descent -

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



Gradient descent (GD)

We understood: For a continuously differentiable $E : \mathbb{R}^n \to \mathbb{R}$, the quantity

$-\nabla E(\theta)$ points into the direction of steepest descent. GD moves into this direction!

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

New parameters	Previous parameters	Direction of steepest descent

Is this cheap or expensive? That depends on E!

Common situation: *Can easily consist of 1,000,000 summands!*

$$E(\theta) = \sum_{\text{training examples } j} \mathcal{L}(\mathcal{N}(x_j;\theta), y_j)$$

$$\mathcal{N}(x;\theta) = \ell^L(\ell^{L-1}(\ldots(\ell^1(x;\theta^1)\ldots);\theta^{L-1});\theta^L)$$

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

Stochastic gradient descent (SGD)

Idea for:

$$E(\theta) = \sum_{\text{training examples } j} \mathcal{L}(\mathcal{N}(x_j;\theta), y_j)$$

Use only a few summands to compute an approximate gradient:

$$E_k(\theta) = \sum_{j\in I(k)} \mathcal{L}(\mathcal{N}(x_j,\theta), y_j)$$ for a ***very small index set*** $I(k)$

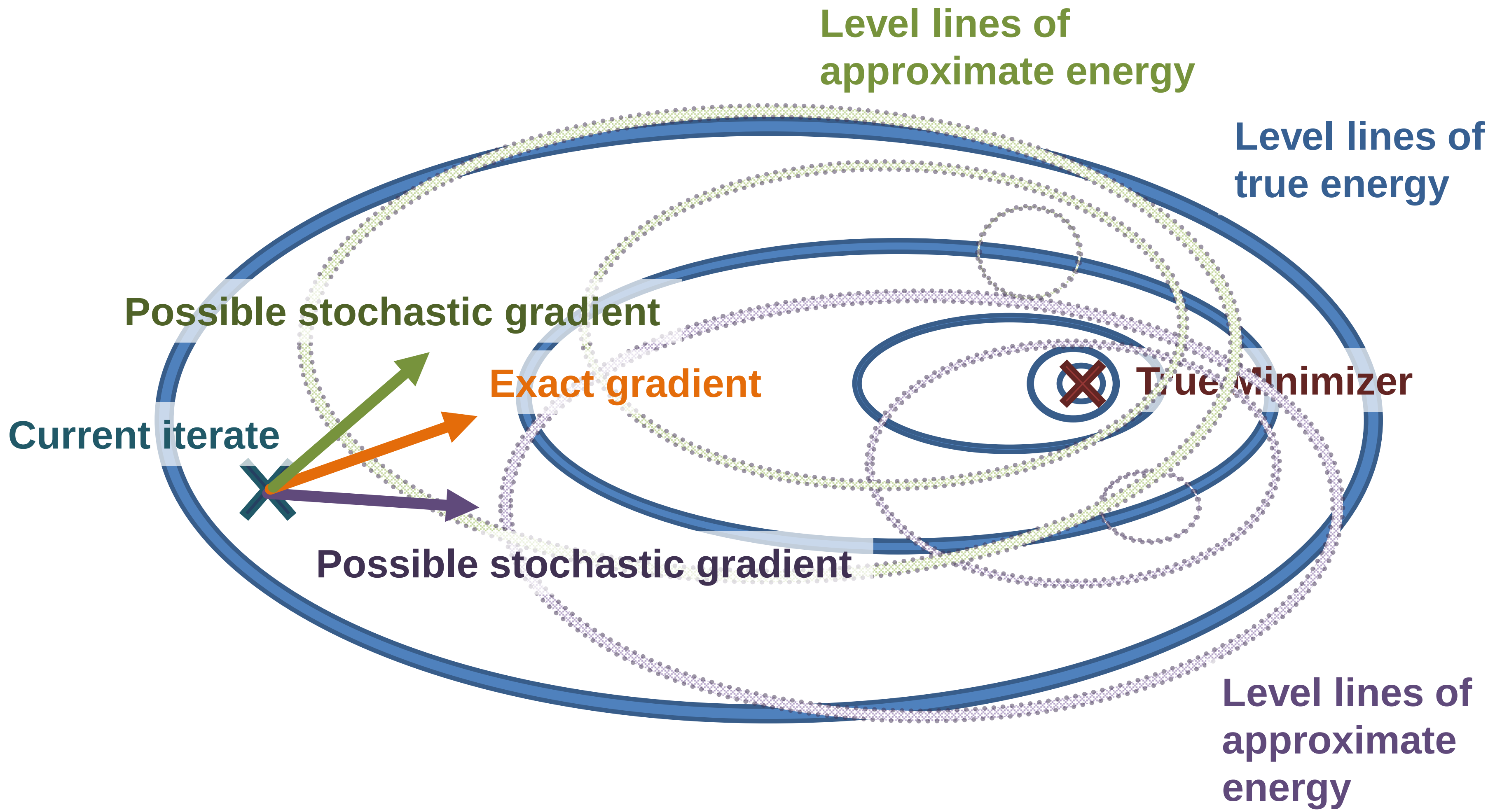
Update the parameters using this approximation

$$\theta(k+1) = \theta(k) - \tau \nabla E_k(\theta(k)) \;\approx \theta^k - \tau \nabla E(\theta^k)$$

Randomly selecting entries in the index set $I(k)$ leads to the name ***stochastic gradient descent***. The training examples (x_j, y_j) with $j \in I(k)$ are called a ***mini-batch***.

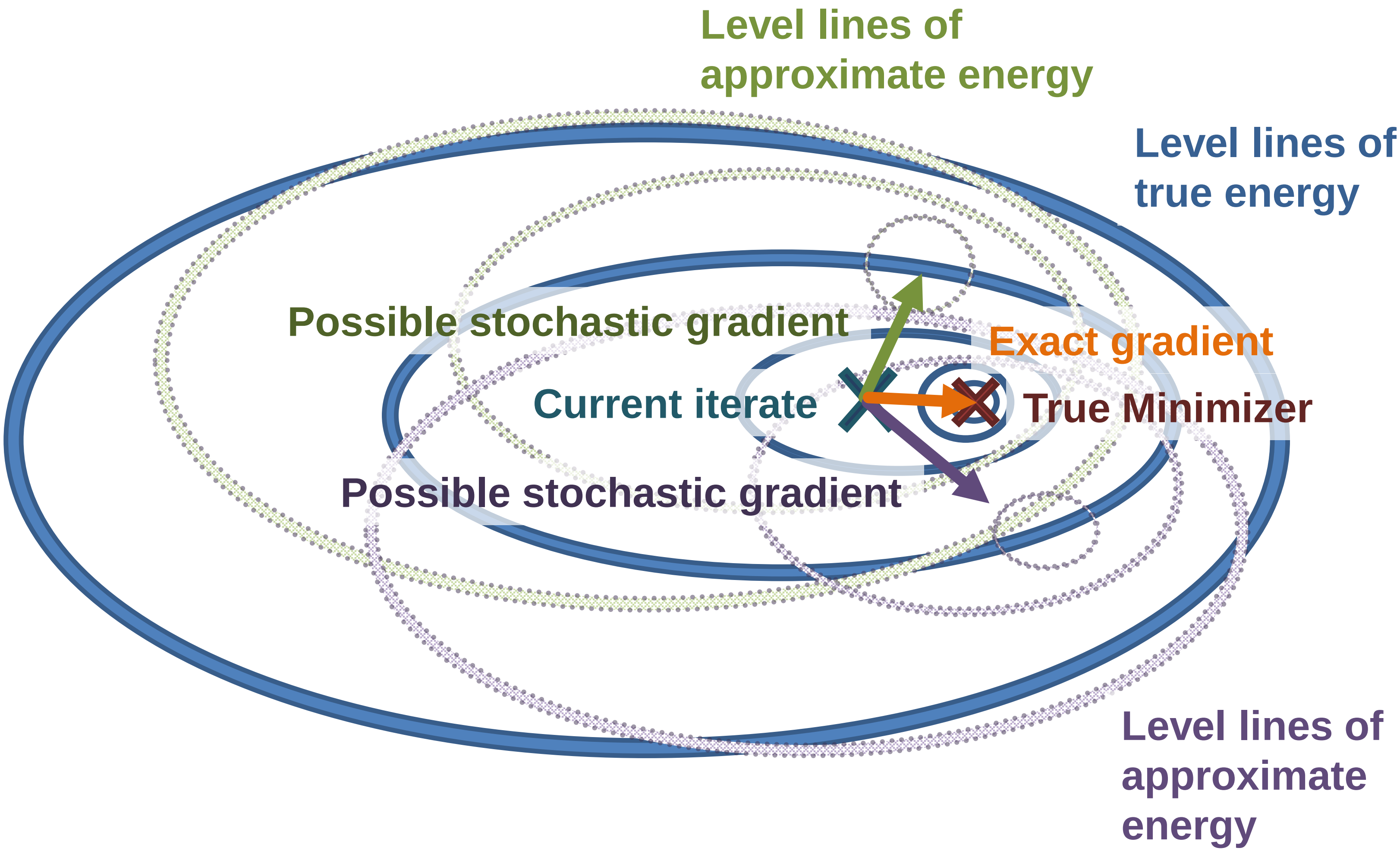
Stochastic gradient descent (SGD)

Approximating the gradients work well if one is still ``sufficiently far away'' from the true minimizer



Stochastic gradient descent (SGD)

Approximating the gradients can easily fail if we are ``sufficiently close'' to the true minimizer



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

Stochastic gradient descent (SGD)

Sanity check to be discussed in the lecture for a simple problem like

$$E(\theta) = \frac{1}{2}(\theta - 1)^2 + \frac{1}{2}(\theta + 1)^2$$

1. Once we are "close to the minimizer" (e.g. between 1 and -1 above), the approximate gradient might not point into the right direction anymore.

2. We need a stepsize that converges to zero, i.e., we need

$$\theta(k+1) = \theta(k) - \tau(k)\ \nabla E_k(\theta(k)) \qquad \text{with} \qquad \lim_{k\to\infty} \tau(k) = 0$$

   to have a chance to converge.

3. The convergence speed at which $\tau(k)$ goes to zero may not be too fast,

   e.g. $\tau(k) = \dfrac{1}{2^k}$ fails!

Stochastic gradient descent (SGD)

Consequence: We need to take the stepsize to zero (but not too fast(!)).

Theoretical results typically require

$$\sum_{k=1}^{\infty} \tau(k) = \infty, \quad \sum_{k=1}^{\infty} \tau(k)^2 < \infty$$

to show (under additional assumptions) that

$$\lim_{k \to \infty} \mathbb{E}[\|\nabla E(\theta(k))\|^2] = 0$$

What kind of convergence results are out there?

Quite well established: Convergence if E is convex

$$\mathbb{E}(E(\theta(k))) - \min_{\theta} E(\theta) = O(1/\sqrt{k})$$

- Lipschitz-continuity of ∇E does not improve the convergence rate
- Additional strong convexity still does not give a linear convergence rate

Nemirosvki et al. (2009), “Robust stochastic optimization approach to stochastic programming”
Botttou et al. (2018), “Optimization Methods for Large-Scale Machine Learning”
Nguyen et al. (2018), “SGD and Hogwild! Convergence Without the Bounded Gradients Assumption“, Theorems 3 and 4

Stochastic gradient descent (SGD)

What kind of convergence results are out there?

Much less well established: Convergence if E is not convex

Similar results of

$$\lim_{k \to \infty} \mathbb{E}[\|\nabla E(\theta(k))\|^2] = 0$$

(for similar requirements on the stepsizes) can only be shown under stronger assumptions, (including Lipschitz continuous derivative, and e.g. Lipschitz continuity of the derivative of $v \mapsto \|\nabla E(v)\|^2$)

For recent results, consider e.g.

- Botttou et al. (2018), “Optimization Methods for Large-Scale Machine Learning”, Corollary 4.12, https://arxiv.org/pdf/1606.04838.pdf
- Lei et al. (2019), „Stochastic Gradient Descent for Nonconvex Learning without Bounded Gradient Assumptions”, https://arxiv.org/abs/1902.00908

Stochastic gradient descent (SGD)

Popular choice of stepsize: ***AdaGrad***

$$c(k+1) = c(k) + \|\nabla E_k(\theta(k))\|^2, \qquad \tau(k) = (c(k+1))^{-1/2}$$

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

A thorough convergence analysis was published as a preprint e.g. in https://arxiv.org/pdf/1806.01811.pdf in June 2018.

Stochastic gradient descent (SGD)

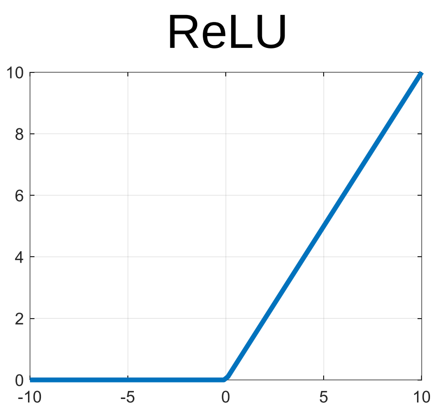
What kind of convergence results are out there?

The aforementioned results are not applicable to most practical deep networks!

Popular layers in the networks, including

- ReLUs
- Leaky or Parameterized ReLUs
- Max-Pooling

are non-differentiable!



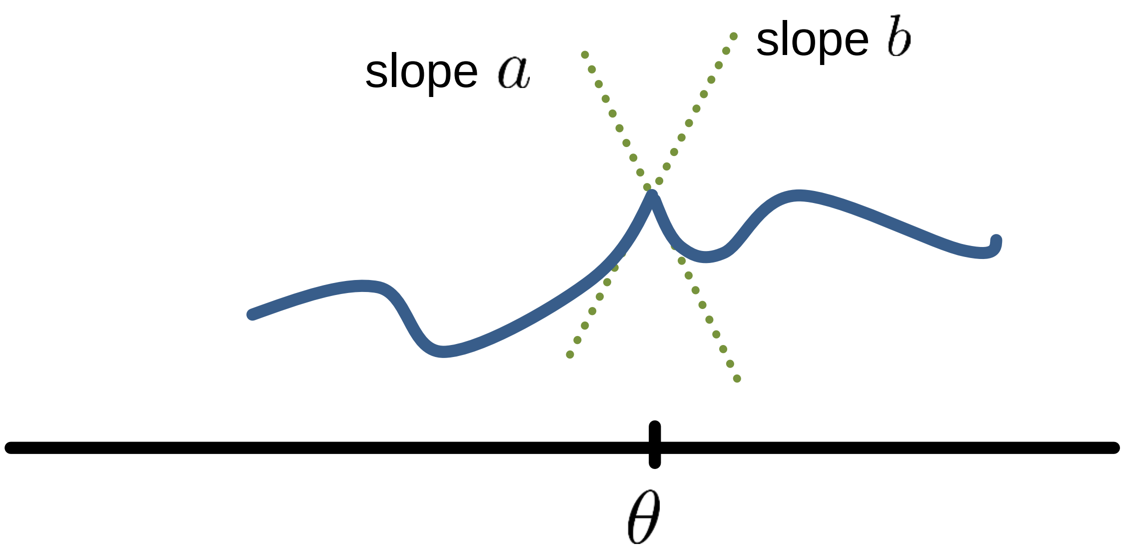
Usual approach: Turn to weaker forms of derivatives, e.g., *subgradients*

Subgradients: The non-convex setting

Clark Subdifferential for locally Lipschitz functions

$$\partial^c E(\theta) = \text{conv}(\{p \in \mathbb{R}^n \mid \exists (\phi_k)_{k\in\mathbb{N}},\ \phi_k \to \theta,\ \phi_k \in R,\ \nabla E(\phi_k) \to p\})$$

with R being the set of points where E is differentiable (almost everywhere based on Rademacher's theorem).



$$\partial^c E(\theta) = [a, b]$$

Stochastic gradient descent (SGD)

Subgradients: The non-convex setting

Deep Learning frameworks (e.g. Tensorflow and PyTorch) form a compute graph, pick a Clark subgradient for each node, and “pretend” that the sum and chain rule hold.

Shocking: This is wrong for the Clark subdifferential for common deep learning functions!

Example

$E_1(\theta) = \theta$ is equal to $E_2(\theta) = \mathrm{ReLU}(\theta) - \mathrm{ReLU}(-\theta)$

obviously

$\nabla E_1(0) = 1$ but any auto-diff framework yields $"\nabla" E_2(0) = 0$

Possible way out: Bolte and Pauwels, „Conservative set valued fields, automatic differentiation, stochastic gradient methods and deep learning“, Oct. 2019

Stochastic gradient descent (SGD)

Practical implementation of SGD for deep learning:

- **For** a desired number of *epochs* (= outer iterations)
  - Shuffle your training data!
  - **For** i **in** range (0 , total_number_of_training_examples , minibatch_size)
    - Take the chunk i:i+minibatch_size out of the (shuffled) training data
    - Do a gradient descent step with a suitable step size (e.g. AdaGrad) only using the sum of loss functions on the current chunk (called *mini-batch*)

As we will learn in the next lecture, the plain SGD step in the above update can often be improved using a technique called *momentum*, or further accelerations such as in an algorithm called *adam*.