

# Introduction to Deep Learning

## Chapter 8: Basic Backpropagation

**Bernd Bischl**

Department of Statistics – LMU Munich

Winter term 2021



# TRAINING NEURAL NETWORKS

Training of neural nets is composed of two iterative steps:

- ➊ **Forward pass:** The information of the inputs flows through the model to produce a prediction. Based on this prediction, the empirical loss is computed.
- ➋ **Backward pass:** The information of the prediction error flows backward through the network to update the weights in a way that the error reduces.

**Recall:** The error is calculated via a loss function  $L(y, f(x, \theta))$ , where  $y$  and  $f(x, \theta)$  are the true target and the network outcome respectively.

# TRAINING NEURAL NETWORKS

- For regression, the L2 loss is typically used:

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

- For classification, the binary/categorical cross entropy:

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

**Note:** Evaluated the loss on the data, the **risk function** is computed:

$$\mathcal{R}_{\text{emp}} = \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)}))$$

# TRAINING NEURAL NETWORKS

To minimize the risk, the **gradient descent** (GD) method can be used.

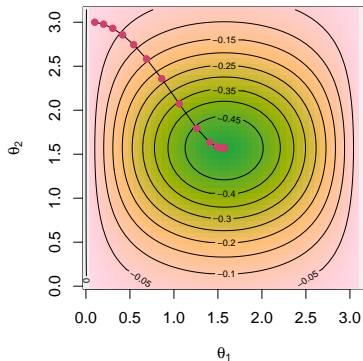
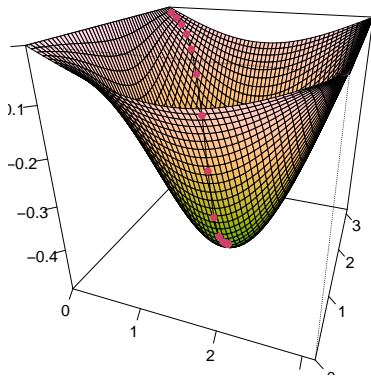
- First, we calculate the gradient  $\nabla \mathcal{R}$  at a point  $\theta^{[t]}$ .
- “Standing” at  $\theta^{[t]}$ , we then improve the minimization by performing the following update:

$$\theta^{[t+1]} = \theta^{[t]} - \alpha \nabla \mathcal{R} \left( \theta^{[t]} \right).$$

- $\alpha$  determines the length of the step and is called the **learning rate**.

**Note:** Since  $\nabla \mathcal{R}$  always points in the direction of the steepest ascent,  $-\nabla \mathcal{R}$  always points in the direction of the steepest descent!

# EXAMPLE: GRADIENT DESCENT



"Walking down the hill, towards the valley."

# WEIGHT UPDATES WITH BACKPROPAGATION

- To update each weight  $w \in \theta$  in the network, we need their gradients with regards to the risk.
- Since weights are stacked in layers inside the network, we need to repeatedly apply the “chain rule of calculus”. This process is called **backpropagation**.
- After obtaining the gradients, the weights can be updated by GD:

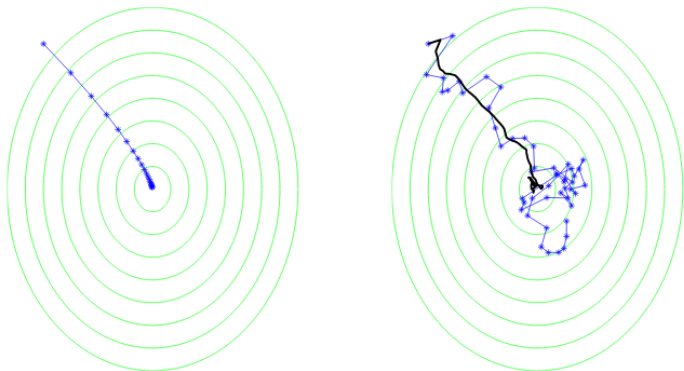
$$\theta^{[t+1]} = \theta^{[t]} - \alpha \cdot \frac{1}{n} \cdot \sum_{i=1}^n \nabla_{\theta} L \left( y^{(i)}, f(\mathbf{x}^{(i)} \mid \theta^{[t]}) \right)$$

# STOCHASTIC GRADIENT DESCENT

- Optimization algorithms that use the entire training set to compute updates in one huge step are called **batch** or **deterministic**. This is computationally very costly or often impossible.
- Instead of running the sum over the whole dataset (**batch mode**), one can run over small subsets (**minibatches**) of size  $m$ .
- With minibatches of size  $m$ , a full pass over the training set (called an **epoch**) consists of  $\frac{n}{m}$  gradient updates.
- This stochastic version of the batch gradient is known as **Stochastic Gradient Descent** (SGD).

# STOCHASTIC GRADIENT DESCENT

An illustration of the SGD algorithm: on the left is GD and on the right is SGD. The black line depicts the averaged value of  $\theta$ .



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



# STOCHASTIC GRADIENT DESCENT

---

## Algorithm Basic SGD pseudo code

---

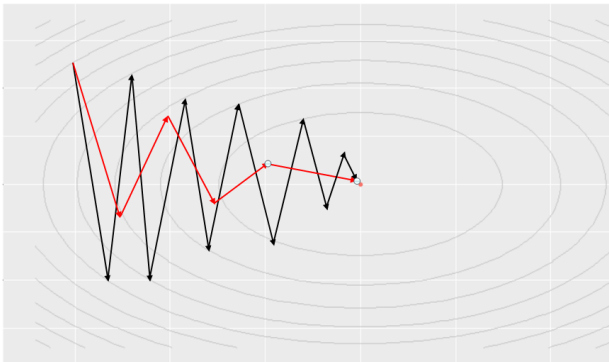
- 1: Initialize parameter vector  $\theta^{[0]}$
- 2:  $t \leftarrow 0$
- 3: **while** stopping criterion not met **do**
- 4:     Randomly shuffle data and partition into minibatches  $J_1, \dots, J_K$  of size  $m$
- 5:     **for**  $k \in \{1, \dots, K\}$  **do**
- 6:          $t \leftarrow t + 1$
- 7:         Compute gradient estimate with  $J_k$ :

$$\hat{g}^{[t]} \leftarrow \frac{1}{m} \sum_{i \in J_k} \nabla_{\theta} L(y^{(i)}, f(\mathbf{x}^{(i)} \mid \theta^{[t-1]}))$$

- 8:         Apply update:  $\theta^{[t]} \leftarrow \theta^{[t-1]} - \alpha \hat{g}^{[t]}$
  - 9:     **end for**
  - 10: **end while**
-

# SGD WITH MOMENTUM

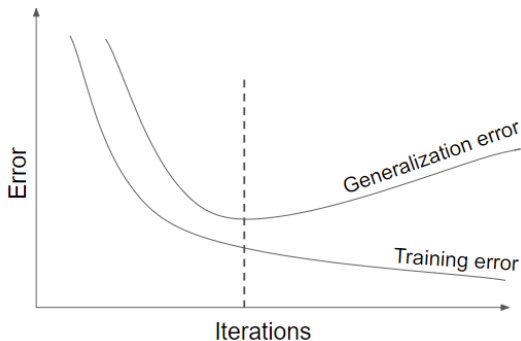
- While SGD remains a popular optimization strategy, learning with it can sometimes be slow.
- Momentum is designed to accelerate learning, by accumulating an exponentially decaying moving average of past gradients.



GD (black) versus momentum (red) when dealing with ravines

# EARLY STOPPING

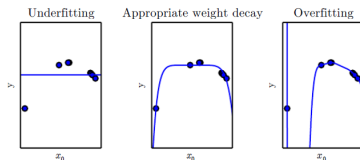
- When training with an iterative optimizer, it is commonly the case that after a certain number of iterations, generalization error begins to increase even though training error continues to decrease.
- **Early stopping** refers to stopping the algorithm, before the generalization error increases, i.e., before the algorithm begins to overfit.



# FURTHER REGULARIZATION STRATEGIES

## Parameter penalties

- Same as Ridge Regression/L2-Regularization
- Often referred to as *weight decay* since weights are pulled to zero if they are not updated by large enough values.



# FURTHER REGULARIZATION STRATEGIES

## Dropout

- Force the network to generalize by reducing its capacity to memorize data.
- Each neuron has a fixed probability to be deactivated at each training step.

