Introduction to Deep Learning

Chapter 2: Basic Training

Bernd Bischl

Department of Statistics – LMU Munich

WS 2021/2022

TRAINING NEURAL NETWORKS

Training of neural nets is composed of two iterative steps:

1. **Forward pass:** The information of the inputs flows through the model to produce a prediction. Based on this prediction, the empirical loss is computed.

2. **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 $L(y, f(\mathbf{x}, \boldsymbol{\theta}))$, where y and $f(\mathbf{x}, \boldsymbol{\theta})$ 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:

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

TRAINING NEURAL NETWORKS

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

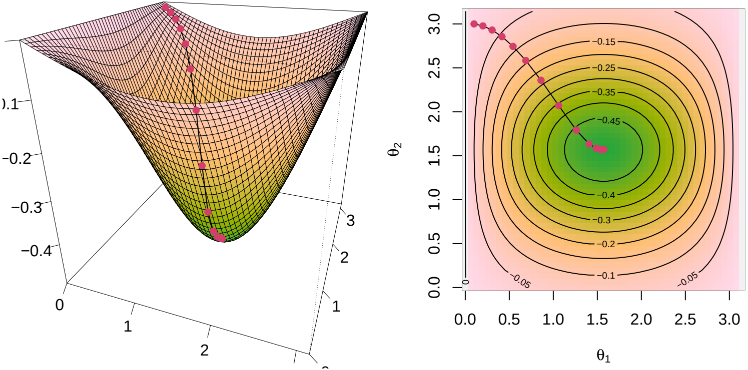
- First, we calculate the gradient $\nabla\mathcal{R}$ at a point $\theta^{[t]}$.
- "Standing" at $\boldsymbol{\theta}^{[t]}$, we then improve the minimization by performing the following update:

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

- α 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 $w \in \boldsymbol{\theta}$ 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:

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

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.

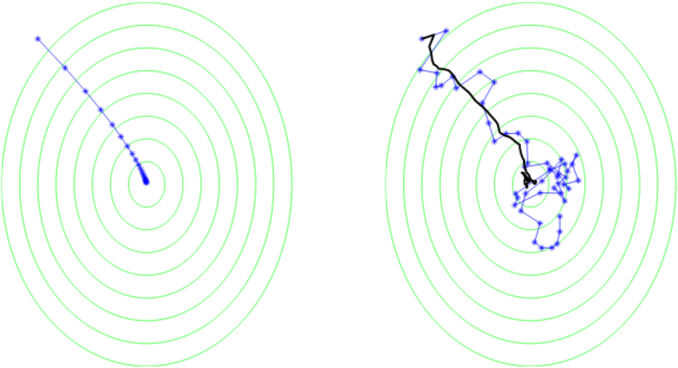
- Instead of running the sum over the whole dataset (**batch mode**), one can run over small subsets (**minibatches**) of size m.

- 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 $\boldsymbol{\theta}$.



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

Initialize parameter vector $\boldsymbol{\theta}^{[0]}$

$t \leftarrow 0$

while stopping criterion not met **do**

  Randomly shuffle data and partition into minibatches $J_1, \ldots, J_K$ of size m

  for $k \in \{1, \ldots, K\}$ **do**

    $t \leftarrow t + 1$

    Compute gradient estimate with J_k:

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

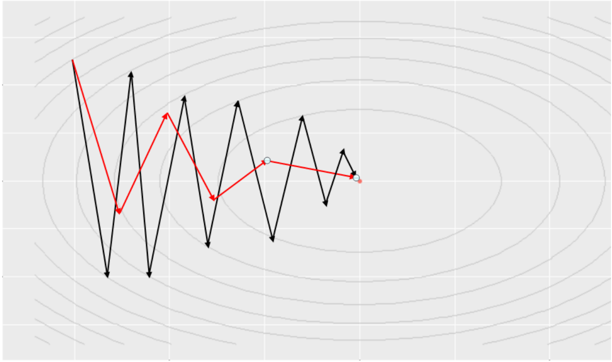
    Apply update: $\boldsymbol{\theta}^{[t]} \leftarrow \boldsymbol{\theta}^{[t-1]} - \alpha \hat{g}^{[t]}$

  end for

end while

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