LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Introduction to Machine Learning

Chapter 4: Deep Learning- TrainingX

Bernd Bischl

Department of Statistics – LMU Munich

Winter term 2021

TRAINING NEURAL NETWORKS

Training of neural nets is composed of two iterative steps:

1. **Forward pass:** The information of the inputs flow through the model to produce a prediction. Based on that, we compute the empirical loss.

2. **Backward pass:** Information of the error of the predictions flows backwards through the model and weights are updated in such a way the error is reduced.

The error is calculated by a loss function $L(y, f(x, \theta))$ of the true target y and the networks output $f(x, \theta)$.

TRAINING NEURAL NETWORKS

- For regression, we typically use the L2 loss:

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

- For classification we typically apply binary/categorical cross entropy:

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

- Evaluated on the data, we refer to it as the risk function:

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

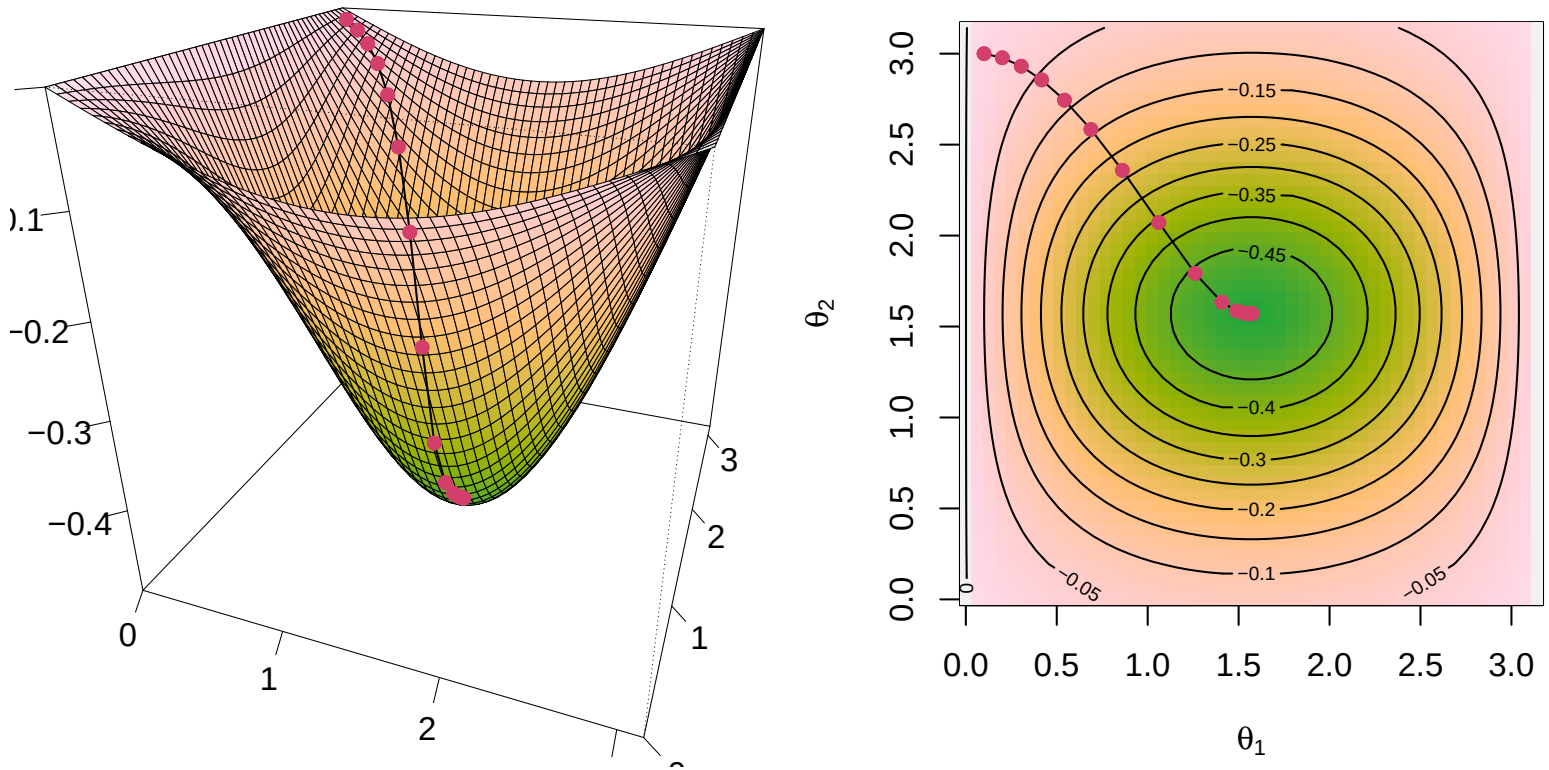
TRAINING NEURAL NETWORKS

- To minimize the risk, we need to exploit the method of gradient descent in numerical optimization.
- At a point $\theta^{[t]}$ we can calculate the gradient $\nabla\mathcal{R}$, which always points in the direction of the steepest ascent.
- Thus, $-\nabla\mathcal{R}$ points in the direction of the steepest descent!
- At a point $\theta^{[t]}$ during minimization, we can improve by doing the following step:

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

where α determines the length of the step and is called learning rate.

TRAINING NEURAL NETWORKS



Example of gradient descent with $\theta = (w_1, w_2)$: *"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 we can update the weights by gradient descent

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

WEIGHT UPDATES WITH BACKPROPAGATION

- 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 letting the sum run over the whole dataset (batch mode) one can also let it run only over small subsets of size m (minibatches).
- With minibatches of size m, a full pass over the training set (called an epoch) consists of $\frac{n}{m}$ gradient updates.

WEIGHT UPDATES WITH BACKPROPAGATION

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

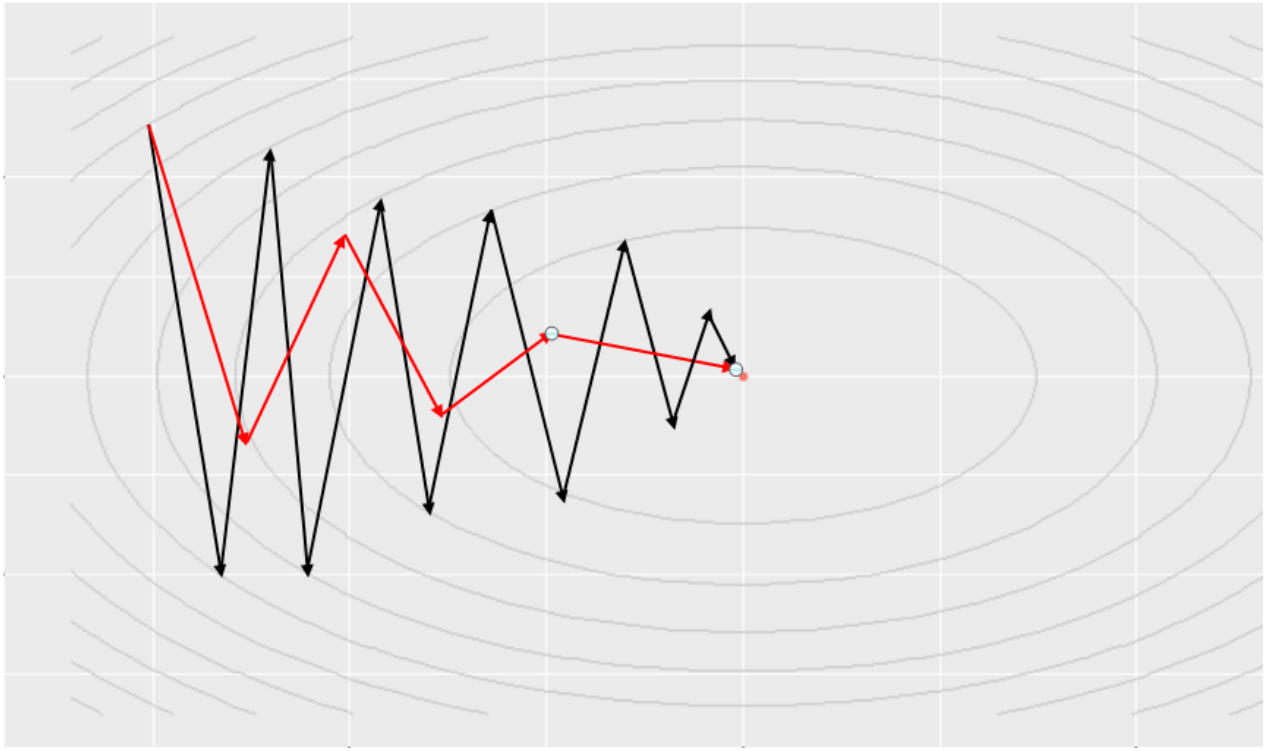
WEIGHT UPDATES WITH BACKPROPAGATION

- While SGD remains a popular optimization strategy, learning with it can sometimes be slow.
- Momentum is designed to accelerate learning, especially when facing high curvature, small but consistent or noisy gradients:

$$\begin{aligned} \nu &\leftarrow \varphi\nu - \alpha \boldsymbol{g}(\theta) \\ \theta &\leftarrow \theta + \nu \end{aligned}$$

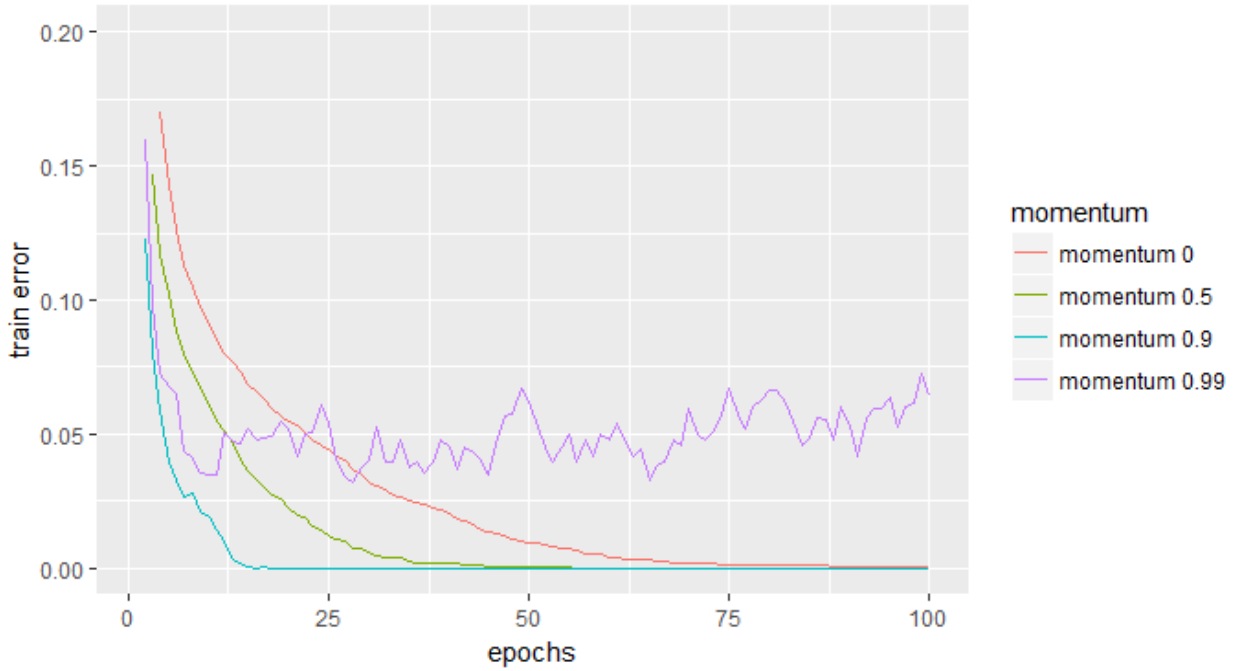
- Accumulate an exponentially decaying moving average of past gradients.

SGD WITH MOMENTUM



The contour lines show a quadratic loss function with a poorly conditioned Hessian matrix. The to curves show how standard gradient descent (black) and momentum (red) learn when dealing with ravines.

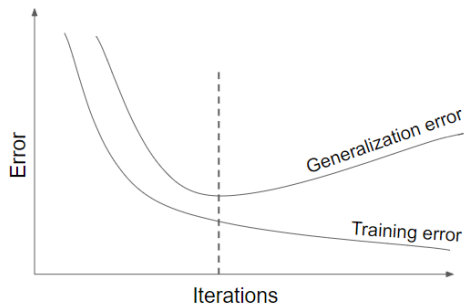
MOMENTUM IN PRACTICE



The higher momentum, the faster SGD learns the weights on the training data, but if momentum is too large, the training and test error fluctuates.

EARLY STOPPING

- When training with an iterative optimizer such as SGD, 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 early, before the generalization error increases.



After a certain number of iterations, the algorithm begins to overfit.

EARLY STOPPING

How early stopping works:

1. Split training data $(X^{(train)}, y^{(train)})$ into $(X^{(subtrain)}, y^{(subtrain)})$ and $(X^{(validation)}, y^{(validation)})$ (e.g. with a ratio of 2:1).
2. Use $(X^{(subtrain)}, y^{(subtrain)})$ and evaluate model using the $(X^{(validation)}, y^{(validation)})$.
3. Stop training when validation error stops decreasing (after a range of "patience" steps).
4. Use parameters of the previous step for the actual model.

EARLY STOPPING

Strengths:

- Effective and simple & Periodical evaluation of validation error
- Applicable to almost any model without adjustment
- Combinable with other regularization methods

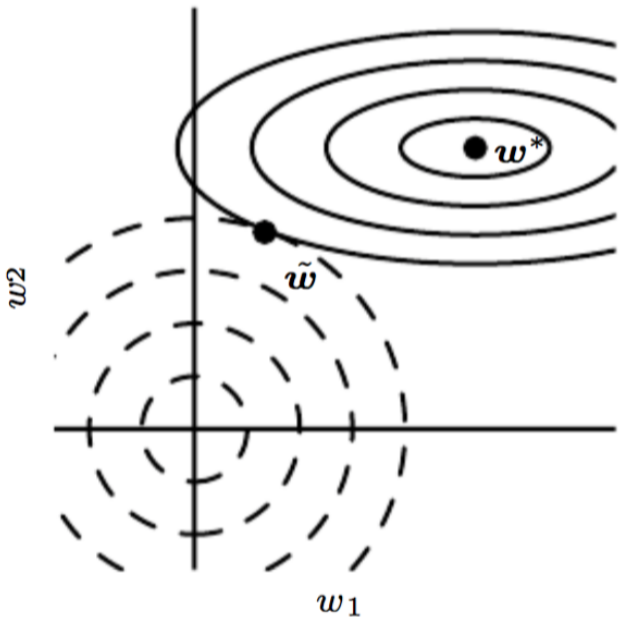
Weaknesses:

- Periodical evaluation of validation error
- Less data for training $\rightarrow$ include $(X^{(validation)}, y^{(validation)})$ afterwards

FURTHER REGULARIZATION STRATEGIES

Parameter penalties

- Same as Rigde 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.