

RWTH AACHEN
UNIVERSITY

Machine Learning – Lecture 11

Neural Networks II

29.11.2018

Bastian Leibe
RWTH Aachen
<http://www.vision.rwth-aachen.de>
leibe@vision.rwth-aachen.de

Machine Learning Winter '18

RWTH AACHEN
UNIVERSITY

Course Outline

- Fundamentals
 - Bayes Decision Theory
 - Probability Density Estimation
- Classification Approaches
 - Linear Discriminants
 - Support Vector Machines
 - Ensemble Methods & Boosting
 - Random Forests
- Deep Learning
 - Foundations
 - Convolutional Neural Networks
 - Recurrent Neural Networks

B. Leibe

RWTH AACHEN
UNIVERSITY

Topics of This Lecture

- Learning Multi-layer Networks
 - Backpropagation
 - Computational graphs
 - Automatic differentiation
 - Practical issues
- Gradient Descent
 - Stochastic Gradient Descent & Minibatches
 - Choosing Learning Rates
 - Momentum
 - RMS Prop
 - Other Optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization

B. Leibe

RWTH AACHEN
UNIVERSITY

Recap: Learning with Hidden Units

- How can we train multi-layer networks efficiently?
 - Need an efficient way of adapting **all** weights, not just the last layer.
- Idea: Gradient Descent
 - Set up an error function

$$E(\mathbf{W}) = \sum_n L(t_n, y(\mathbf{x}_n; \mathbf{W})) + \lambda \Omega(\mathbf{W})$$
 with a loss $L(\cdot)$ and a regularizer $\Omega(\cdot)$.
 - E.g., $L(t, y(\mathbf{x}; \mathbf{W})) = \sum_n (y(\mathbf{x}_n; \mathbf{W}) - t_n)^2$ L₂ loss
 - $\Omega(\mathbf{W}) = \|\mathbf{W}\|_F^2$ L₂ regularizer ("weight decay")
 - ⇒ Update each weight $W_{ij}^{(k)}$ in the direction of the gradient $\frac{\partial E(\mathbf{W})}{\partial W_{ij}^{(k)}}$

B. Leibe

RWTH AACHEN
UNIVERSITY

Gradient Descent

- Two main steps
 1. Computing the gradients for each weight
 2. Adjusting the weights in the direction of the gradient

B. Leibe

RWTH AACHEN
UNIVERSITY

Obtaining the Gradients

- Approach 1: Naive Analytical Differentiation

$$\frac{\partial E(\mathbf{W})}{\partial W_{10}^{(2)}} \cdots \frac{\partial E(\mathbf{W})}{\partial W_{kh}^{(2)}}$$

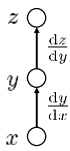
$$\frac{\partial E(\mathbf{W})}{\partial W_{10}^{(1)}} \cdots \frac{\partial E(\mathbf{W})}{\partial W_{hd}^{(1)}}$$

- Compute the gradients for each variable analytically.
- *What is the problem when doing this?*

B. Leibe

Excursion: Chain Rule of Differentiation

- One-dimensional case: Scalar functions



$$\Delta z = \frac{dz}{dy} \Delta y$$

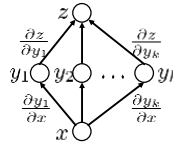
$$\Delta y = \frac{dy}{dx} \Delta x$$

$$\Delta z = \frac{dz}{dy} \frac{dy}{dx} \Delta x$$

$$\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$$

Excursion: Chain Rule of Differentiation

- Multi-dimensional case: Total derivative



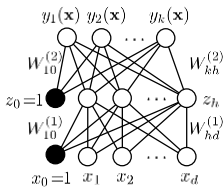
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2} \frac{\partial y_2}{\partial x} + \dots$$

$$= \sum_{i=1}^k \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

⇒ Need to sum over all paths that lead to the target variable x .

Obtaining the Gradients

- Approach 1: Naive Analytical Differentiation



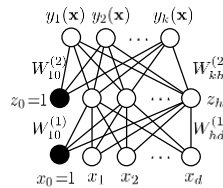
$$\frac{\partial E(\mathbf{W})}{\partial W_{10}^{(2)}} \dots \frac{\partial E(\mathbf{W})}{\partial W_{kh}^{(2)}}$$

$$\frac{\partial E(\mathbf{W})}{\partial W_{10}^{(1)}} \dots \frac{\partial E(\mathbf{W})}{\partial W_{hd}^{(1)}}$$

- Compute the gradients for each variable analytically.
- What is the problem when doing this?
 - ⇒ With increasing depth, there will be exponentially many paths!
 - ⇒ Infeasible to compute this way.

Obtaining the Gradients

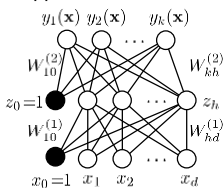
- Approach 2: Numerical Differentiation



- Given the current state $\mathbf{W}^{(r)}$, we can evaluate $E(\mathbf{W}^{(r)})$.
- Idea: Make small changes to $\mathbf{W}^{(r)}$ and accept those that improve $E(\mathbf{W}^{(r)})$.
- ⇒ Horribly inefficient! Need several forward passes for each weight. Each forward pass is one run over the entire dataset!

Obtaining the Gradients

- Approach 3: Incremental Analytical Differentiation



$$\frac{\partial E(\mathbf{W})}{\partial y_j} \rightarrow \frac{\partial E(\mathbf{W})}{\partial W_{ij}^{(2)}}$$

$$\frac{\partial E(\mathbf{W})}{\partial z_i} \rightarrow \frac{\partial E(\mathbf{W})}{\partial W_{ij}^{(1)}}$$

$$\frac{\partial E(\mathbf{W})}{\partial x_i}$$

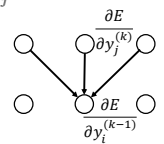
- Idea: Compute the gradients layer by layer.
- Each layer below builds upon the results of the layer above.
- ⇒ The gradient is propagated backwards through the layers.
- ⇒ **Backpropagation** algorithm

Backpropagation Algorithm

- Core steps
 - Convert the discrepancy between each output and its target value into an error derivate.

$$E = \frac{1}{2} \sum_{j \in \text{output}} (t_j - y_j)^2$$

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$
 - Compute error derivatives in each hidden layer from error derivatives in the layer above.
 - Use error derivatives w.r.t. activities to get error derivatives w.r.t. the incoming weights



$$\frac{\partial E}{\partial y_j^{(k)}} \rightarrow \frac{\partial E}{\partial w_{ji}^{(k-1)}}$$

Machine Learning Winter '18

Backpropagation Algorithm

$$\frac{\partial E}{\partial z_j^{(k)}} = \frac{\partial y_j^{(k)}}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}} = \frac{\partial g(z_j^{(k)})}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}}$$

- Notation
 - $y_j^{(k)}$ Output of layer k Connections: $z_j^{(k)} = \sum_i w_{ji}^{(k-1)} y_i^{(k-1)}$
 - $z_j^{(k)}$ Input of layer k $y_j^{(k)} = g(z_j^{(k)})$

Slide adapted from Geoff Hinton B. Leibe 16

Machine Learning Winter '18

Backpropagation Algorithm

$$\frac{\partial E}{\partial z_j^{(k)}} = \frac{\partial y_j^{(k)}}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}} = \frac{\partial g(z_j^{(k)})}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}}$$

$$\frac{\partial E}{\partial y_i^{(k-1)}} = \sum_j \frac{\partial z_j^{(k)}}{\partial y_i^{(k-1)}} \frac{\partial E}{\partial z_j^{(k)}} = \sum_j w_{ji}^{(k-1)} \frac{\partial E}{\partial z_j^{(k)}}$$

- Notation
 - $y_j^{(k)}$ Output of layer k Connections: $z_j^{(k)} = \sum_i w_{ji}^{(k-1)} y_i^{(k-1)}$
 - $z_j^{(k)}$ Input of layer k $\frac{\partial z_j^{(k)}}{\partial y_i^{(k-1)}} = w_{ji}^{(k-1)}$

Slide adapted from Geoff Hinton B. Leibe 17

Machine Learning Winter '18

Backpropagation Algorithm

$$\frac{\partial E}{\partial z_j^{(k)}} = \frac{\partial y_j^{(k)}}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}} = \frac{\partial g(z_j^{(k)})}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}}$$

$$\frac{\partial E}{\partial y_i^{(k-1)}} = \sum_j \frac{\partial z_j^{(k)}}{\partial y_i^{(k-1)}} \frac{\partial E}{\partial z_j^{(k)}} = \sum_j w_{ji}^{(k-1)} \frac{\partial E}{\partial z_j^{(k)}}$$

$$\frac{\partial E}{\partial w_{ji}^{(k-1)}} = \frac{\partial z_j^{(k)}}{\partial w_{ji}^{(k-1)}} \frac{\partial E}{\partial z_j^{(k)}} = y_i^{(k-1)} \frac{\partial E}{\partial z_j^{(k)}}$$

- Notation
 - $y_j^{(k)}$ Output of layer k Connections: $z_j^{(k)} = \sum_i w_{ji}^{(k-1)} y_i^{(k-1)}$
 - $z_j^{(k)}$ Input of layer k $\frac{\partial z_j^{(k)}}{\partial w_{ji}^{(k-1)}} = y_i^{(k-1)}$

Slide adapted from Geoff Hinton B. Leibe 18

Machine Learning Winter '18

Backpropagation Algorithm

$$\frac{\partial E}{\partial z_j^{(k)}} = \frac{\partial y_j^{(k)}}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}} = \frac{\partial g(z_j^{(k)})}{\partial z_j^{(k)}} \frac{\partial E}{\partial y_j^{(k)}}$$

$$\frac{\partial E}{\partial y_i^{(k-1)}} = \sum_j \frac{\partial z_j^{(k)}}{\partial y_i^{(k-1)}} \frac{\partial E}{\partial z_j^{(k)}} = \sum_j w_{ji}^{(k-1)} \frac{\partial E}{\partial z_j^{(k)}}$$

$$\frac{\partial E}{\partial w_{ji}^{(k-1)}} = \frac{\partial z_j^{(k)}}{\partial w_{ji}^{(k-1)}} \frac{\partial E}{\partial z_j^{(k)}} = y_i^{(k-1)} \frac{\partial E}{\partial z_j^{(k)}}$$

- Efficient propagation scheme
 - $y_i^{(k-1)}$ is already known from forward pass! (Dynamic Programming)
 - \Rightarrow Propagate back the gradient from layer k and multiply with $y_i^{(k-1)}$.

Slide adapted from Geoff Hinton B. Leibe 19

Machine Learning Winter '18

Summary: MLP Backpropagation

- Forward Pass
 - $\mathbf{y}^{(0)} = \mathbf{x}$
 - for $k = 1, \dots, l$ do
 - $\mathbf{z}^{(k)} = \mathbf{W}^{(k)} \mathbf{y}^{(k-1)}$
 - $\mathbf{y}^{(k)} = g_k(\mathbf{z}^{(k)})$
 - endfor
 - $\mathbf{y} = \mathbf{y}^{(l)}$
 - $E = L(\mathbf{t}, \mathbf{y}) + \lambda \Omega(\mathbf{W})$
- Backward Pass
 - $\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{y}} = \frac{\partial}{\partial \mathbf{y}} L(\mathbf{t}, \mathbf{y}) + \lambda \frac{\partial \Omega}{\partial \mathbf{y}}$
 - for $k = l, l-1, \dots, 1$ do
 - $\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{z}^{(k)}} = \mathbf{h} \odot g'(\mathbf{y}^{(k)})$
 - $\frac{\partial E}{\partial \mathbf{W}^{(k)}} = \mathbf{h} \mathbf{y}^{(k-1)T} + \lambda \frac{\partial \Omega}{\partial \mathbf{W}^{(k)}}$
 - $\mathbf{h} \leftarrow \frac{\partial E}{\partial \mathbf{y}^{(k-1)}} = \mathbf{W}^{(k)T} \mathbf{h}$
 - endfor

Notes

- For efficiency, an entire batch of data \mathbf{X} is processed at once.
- \odot denotes the element-wise product

B. Leibe 20

Machine Learning Winter '18

Analysis: Backpropagation

- Backpropagation is the key to make deep NNs tractable
 - However...
- The Backprop algorithm given here is specific to MLPs
 - It does not work with more complex architectures, e.g. skip connections or recurrent networks!
 - Whenever a new connection function induces a different functional form of the chain rule, you have to derive a new Backprop algorithm for it.
 - \Rightarrow Tedious...
- Let's analyze Backprop in more detail
 - This will lead us to a more flexible algorithm formulation

B. Leibe 21

RWTH AACHEN UNIVERSITY

Topics of This Lecture

- Learning Multi-layer Networks
 - Recap: Backpropagation
 - Computational graphs
 - Automatic differentiation
 - Practical issues
- Gradient Descent
 - Stochastic Gradient Descent & Minibatches
 - Choosing Learning Rates
 - Momentum
 - RMS Prop
 - Other Optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization

22

RWTH AACHEN UNIVERSITY

Computational Graphs

- We can think of mathematical expressions as graphs
 - E.g., consider the expression $e = (a + b) * (b + 1)$
 - We can decompose this into the operations $c = a + b$, $d = b + 1$, $e = c * d$

23

RWTH AACHEN UNIVERSITY

Factoring Paths

- Problem: Combinatorial explosion
 - Example:

- There are 3 paths from X to Y and 3 more from Y to Z.
- If we want to compute $\frac{\partial Z}{\partial X}$, we need to sum over 3×3 paths:

$$\frac{\partial Z}{\partial X} = \alpha\delta + \alpha\epsilon + \alpha\zeta + \beta\delta + \beta\epsilon + \beta\zeta + \gamma\delta + \gamma\epsilon + \gamma\zeta$$
- Instead of naively summing over paths, it's better to factor them

$$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma) * (\delta + \epsilon + \zeta)$$

24

RWTH AACHEN UNIVERSITY

Efficient Factored Algorithms

- Efficient algorithms for computing the sum
 - Instead of summing over all of the paths explicitly, compute the sum more efficiently by merging paths back together at every node.

25

RWTH AACHEN UNIVERSITY

Why Do We Care?

- Let's consider the example again
 - Using forward-mode differentiation from b up...
 - Runtime: $\mathcal{O}(\#edges)$
 - Result: derivative of every node with respect to b.

26

RWTH AACHEN UNIVERSITY

Why Do We Care?

- Let's consider the example again
 - Using reverse-mode differentiation from e down...
 - Runtime: $\mathcal{O}(\#edges)$
 - Result: derivative of e with respect to every node.

⇒ This is what we want to compute in Backpropagation!

- Forward differentiation needs one pass per node. With backward differentiation we can compute all derivatives in one single pass.
- ⇒ Speed-up in $\mathcal{O}(\#inputs)$ compared to forward differentiation!

27

RWTH AACHEN UNIVERSITY

Topics of This Lecture

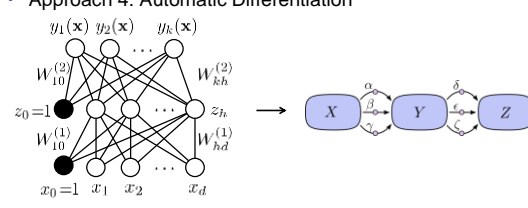
- **Learning Multi-layer Networks**
 - Recap: Backpropagation
 - Computational graphs
 - Automatic differentiation
 - Practical issues
- Gradient Descent
 - Stochastic Gradient Descent & Minibatches
 - Choosing Learning Rates
 - Momentum
 - RMS Prop
 - Other Optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization

B. Leibe 28

RWTH AACHEN UNIVERSITY

Obtaining the Gradients

- Approach 4: Automatic Differentiation



- Convert the network into a computational graph.
- Each new layer/module just needs to specify how it affects the forward and backward passes.
- Apply reverse-mode differentiation.

⇒ Very general algorithm, used in today's Deep Learning packages

B. Leibe 29

RWTH AACHEN UNIVERSITY

Modular Implementation

- Solution in many current Deep Learning libraries
 - Provide a limited form of automatic differentiation
 - Restricted to "programs" composed of "modules" with a predefined set of operations.
- Each module is defined by two main functions
 1. Computing the outputs \mathbf{y} of the module given its inputs \mathbf{x}

$$\mathbf{y} = \text{module.fprop}(\mathbf{x})$$

where \mathbf{x} , \mathbf{y} , and intermediate results are stored in the module.
 2. Computing the gradient $\partial E / \partial \mathbf{x}$ of a scalar cost w.r.t. the inputs \mathbf{x} given the gradient $\partial E / \partial \mathbf{y}$ w.r.t. the outputs \mathbf{y}

$$\frac{\partial E}{\partial \mathbf{x}} = \text{module.bprop}\left(\frac{\partial E}{\partial \mathbf{y}}\right)$$

B. Leibe 30

RWTH AACHEN UNIVERSITY

Topics of This Lecture

- **Learning Multi-layer Networks**
 - Recap: Backpropagation
 - Computational graphs
 - Automatic differentiation
 - Practical issues
- Gradient Descent
 - Stochastic Gradient Descent & Minibatches
 - Choosing Learning Rates
 - Momentum
 - RMS Prop
 - Other Optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization

B. Leibe 31

RWTH AACHEN UNIVERSITY

Implementing Softmax Correctly

- Softmax output
 - De-facto standard for multi-class outputs
$$E(\mathbf{w}) = - \sum_{n=1}^N \sum_{k=1}^K \left\{ \mathbb{I}(t_n = k) \ln \frac{\exp(\mathbf{w}_k^T \mathbf{x})}{\sum_{j=1}^K \exp(\mathbf{w}_j^T \mathbf{x})} \right\}$$
- Practical issue
 - Exponentials get very big and can have vastly different magnitudes.
 - **Trick 1:** Do not compute first softmax, then log, but instead directly evaluate log-exp in the nominator and log-sum-exp in the denominator.
 - **Trick 2:** Softmax has the property that for a fixed vector \mathbf{b}

$$\text{softmax}(\mathbf{a} + \mathbf{b}) = \text{softmax}(\mathbf{a})$$

⇒ Subtract the largest weight vector \mathbf{w}_j from the others.

B. Leibe 32

RWTH AACHEN UNIVERSITY

Topics of This Lecture

- **Learning Multi-layer Networks**
 - Recap: Backpropagation
 - Computational graphs
 - Automatic differentiation
 - Practical issues
- Gradient Descent
 - Stochastic Gradient Descent & Minibatches
 - Choosing Learning Rates
 - Momentum
 - RMS Prop
 - Other Optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization

B. Leibe 33

Gradient Descent

- Two main steps
 1. Computing the gradients for each weight
 2. Adjusting the weights in the direction of the gradient

- Recall: Basic update equation

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

- Main questions
 - On what data do we want to apply this?
 - How should we choose the step size η (the learning rate)?
 - In which direction should we update the weights?

Stochastic vs. Batch Learning

- Batch learning

- Process the full dataset at once to compute the gradient.

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

- Stochastic learning

- Choose a single example from the training set.
- Compute the gradient only based on this example
- This estimate will generally be noisy, which has some advantages.

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E_n(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

Stochastic vs. Batch Learning

- Batch learning advantages

- Conditions of convergence are well understood.
- Many acceleration techniques (e.g., conjugate gradients) only operate in batch learning.
- Theoretical analysis of the weight dynamics and convergence rates are simpler.

- Stochastic learning advantages

- Usually much faster than batch learning.
- Often results in better solutions.
- Can be used for tracking changes.

- Middle ground: Minibatches

Minibatches

- Idea

- Process only a small batch of training examples together
- Start with a small batch size & increase it as training proceeds.

- Advantages

- Gradients will be more stable than for stochastic gradient descent, but still faster to compute than with batch learning.
- Take advantage of redundancies in the training set.
- Matrix operations are more efficient than vector operations.

- Caveat

- Error function should be normalized by the minibatch size, s.t. we can keep the same learning rate between minibatches

$$E(\mathbf{W}) = \frac{1}{N} \sum_n L(t_n, y(x_n; \mathbf{W})) + \frac{\lambda}{N} \Omega(\mathbf{W})$$

Topics of This Lecture

- Learning Multi-layer Networks

- Recap: Backpropagation
- Computational graphs
- Automatic differentiation
- Practical issues

- Gradient Descent

- Stochastic Gradient Descent & Minibatches
- **Choosing Learning Rates**
- Momentum
- RMS Prop
- Other Optimizers

- Tricks of the Trade

- Shuffling
- Data Augmentation
- Normalization

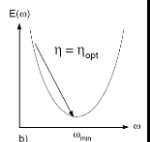
Choosing the Right Learning Rate

- Analyzing the convergence of Gradient Descent

- Consider a simple 1D example first

$$W^{(\tau+1)} = W^{(\tau)} - \eta \frac{dE(W)}{dW}$$

- What is the optimal learning rate η_{opt} ?



- If E is quadratic, the optimal learning rate is given by the inverse of the Hessian

$$\eta_{opt} = \left(\frac{d^2 E(W^{(\tau)})}{dW^2} \right)^{-1}$$

- What happens if we exceed this learning rate?

Choosing the Right Learning Rate

- Behavior for different learning rates

a) $\eta < \eta_{opt}$ b) $\eta = \eta_{opt}$

c) $\eta > \eta_{opt}$ d) $\eta > 2 \eta_{opt}$

B. Leibe image source: Yann LeCun et al., Efficient BackProp (1998)

Learning Rate vs. Training Error

Training error

Learning rate (logarithmic scale)

Do not go beyond this point!

B. Leibe image source: Goodfellow, & Bengio book

Topics of This Lecture

- Learning Multi-layer Networks
 - Recap: Backpropagation
 - Computational graphs
 - Automatic differentiation
 - Practical issues
- Gradient Descent
 - Stochastic Gradient Descent & Minibatches
 - Choosing Learning Rates
 - Momentum
 - RMS Prop
 - Other Optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization

B. Leibe

Batch vs. Stochastic Learning

- Batch Learning
 - Simplest case: steepest descent on the error surface.
 - Updates perpendicular to contour lines
- Stochastic Learning
 - Simplest case: zig-zag around the direction of steepest descent.
 - Updates perpendicular to constraints from training examples.

Slide adapted from Geoff Hinton

B. Leibe image source: Geoff Hinton

Why Learning Can Be Slow

- If the inputs are correlated
 - The ellipse will be very elongated.
 - The direction of steepest descent is almost perpendicular to the direction towards the minimum!

This is just the opposite of what we want!

Slide adapted from Geoff Hinton

B. Leibe image source: Geoff Hinton

The Momentum Method

- Idea
 - Instead of using the gradient to change the position of the weight "particle", use it to change the velocity.
- Intuition
 - Example: Ball rolling on the error surface
 - It starts off by following the error surface, but once it has accumulated momentum, it no longer does steepest descent.
- Effect
 - Dampen oscillations in directions of high curvature by combining gradients with opposite signs.
 - Build up speed in directions with a gentle but consistent gradient.

Slide credit: Geoff Hinton

B. Leibe image source: Geoff Hinton

RWTH AACHEN UNIVERSITY

The Momentum Method: Implementation

- Change in the update equations
 - Effect of the gradient: increment the previous velocity, subject to a decay by $\alpha < 1$.

$$\mathbf{v}(t) = \alpha \mathbf{v}(t-1) - \epsilon \frac{\partial E}{\partial \mathbf{w}}(t)$$
 - Set the weight change to the current velocity

$$\begin{aligned} \Delta \mathbf{w} &= \mathbf{v}(t) \\ &= \alpha \mathbf{v}(t-1) - \epsilon \frac{\partial E}{\partial \mathbf{w}}(t) \\ &= \alpha \Delta \mathbf{w}(t-1) - \epsilon \frac{\partial E}{\partial \mathbf{w}}(t) \end{aligned}$$

Machine Learning Winter '18 46
Slide credit: Geoff Hinton B. Leibe

RWTH AACHEN UNIVERSITY

The Momentum Method: Behavior

- Behavior
 - If the error surface is a tilted plane, the ball reaches a terminal velocity

$$\mathbf{v}(\infty) = \frac{1}{1-\alpha} \left(-\epsilon \frac{\partial E}{\partial \mathbf{w}} \right)$$
 - If the momentum α is close to 1, this is much faster than simple gradient descent.
 - At the beginning of learning, there may be very large gradients.
 - Use a small momentum initially (e.g., $\alpha = 0.5$).
 - Once the large gradients have disappeared and the weights are stuck in a ravine, the momentum can be smoothly raised to its final value (e.g., $\alpha = 0.90$ or even $\alpha = 0.99$).

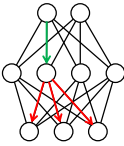
\Rightarrow This allows us to learn at a rate that would cause divergent oscillations without the momentum.

Machine Learning Winter '18 47
Slide credit: Geoff Hinton B. Leibe

RWTH AACHEN UNIVERSITY

Separate, Adaptive Learning Rates

- Problem
 - In multilayer nets, the appropriate learning rates can vary widely between weights.
 - The magnitudes of the gradients are often very different for the different layers, especially if the initial weights are small.
 - \Rightarrow Gradients can get very small in the early layers of deep nets.

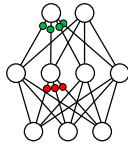


Machine Learning Winter '18 48
Slide adapted from Geoff Hinton B. Leibe

RWTH AACHEN UNIVERSITY

Separate, Adaptive Learning Rates

- Problem
 - In multilayer nets, the appropriate learning rates can vary widely between weights.
 - The magnitudes of the gradients are often very different for the different layers, especially if the initial weights are small.
 - \Rightarrow Gradients can get very small in the early layers of deep nets.
 - The fan-in of a unit determines the size of the "overshoot" effect when changing multiple weights simultaneously to correct the same error.
 - The fan-in often varies widely between layers
- Solution
 - Use a global learning rate, multiplied by a local gain per weight (determined empirically)



Machine Learning Winter '18 49
Slide adapted from Geoff Hinton B. Leibe

RWTH AACHEN UNIVERSITY

Better Adaptation: RMSProp

- Motivation
 - The magnitude of the gradient can be very different for different weights and can change during learning.
 - This makes it hard to choose a single global learning rate.
 - For batch learning, we can deal with this by only using the sign of the gradient, but we need to generalize this for minibatches.
- Idea of RMSProp
 - Divide the gradient by a running average of its recent magnitude

$$MeanSq(w_{ij}, t) = 0.9 MeanSq(w_{ij}, t-1) + 0.1 \left(\frac{\partial E}{\partial w_{ij}}(t) \right)^2$$
 - Divide the gradient by $\text{sqrt}(MeanSq(w_{ij}, t))$.

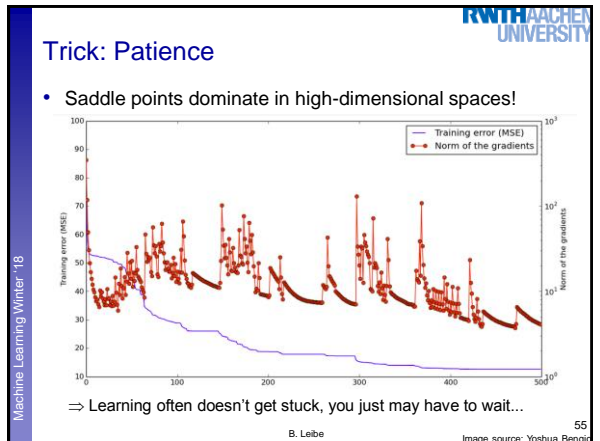
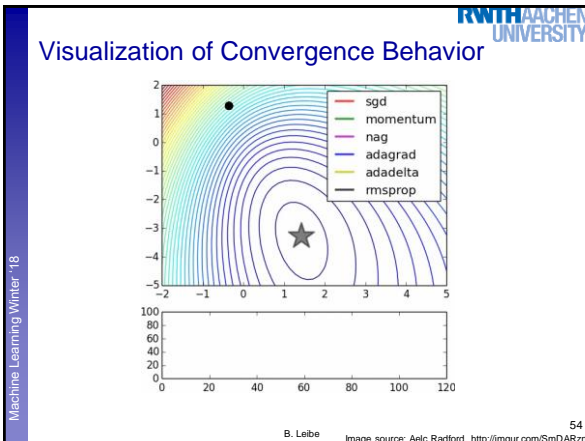
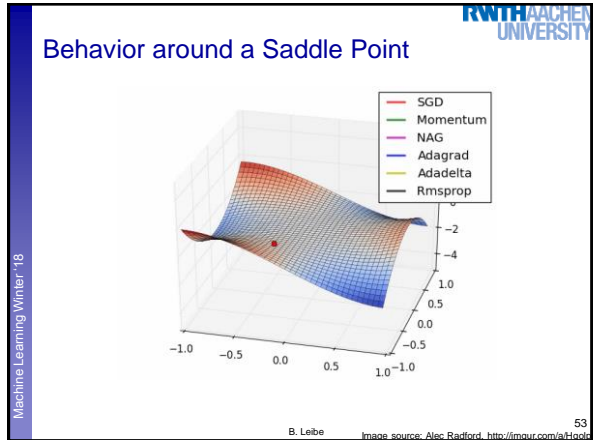
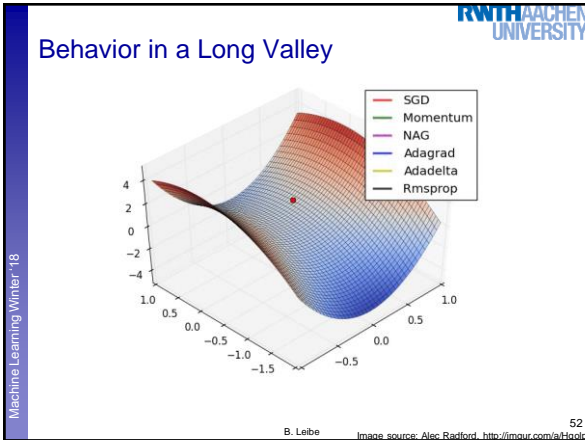
Machine Learning Winter '18 50
Slide adapted from Geoff Hinton B. Leibe

RWTH AACHEN UNIVERSITY

Other Optimizers

- AdaGrad [Duchi '10]
- AdaDelta [Zeiler '12]
- Adam [Ba & Kingma '14]
- Notes
 - All of those methods have the goal to make the optimization less sensitive to parameter settings.
 - Adam is currently becoming the quasi-standard

Machine Learning Winter '18 51
B. Leibe



Reducing the Learning Rate

- Final improvement step after convergence is reached
 - Reduce learning rate by a factor of 10.
 - Continue training for a few epochs.
 - Do this 1-3 times, then stop training.
- Effect
 - Turning down the learning rate will reduce the random fluctuations in the error due to different gradients on different minibatches.
- Be careful: Do not turn down the learning rate too soon!
 - Further progress will be much slower/impossible after that.

Machine Learning Winter '18

B. Leibe

Slide adapted from Geoff Hinton

56

Summary

- Deep multi-layer networks are very powerful.
- But training them is hard!
 - Complex, non-convex learning problem
 - Local optimization with stochastic gradient descent
- Main issue: getting good gradient updates for the lower layers of the network
 - Many seemingly small details matter!
 - Weight initialization, normalization, data augmentation, choice of nonlinearities, choice of learning rate, choice of optimizer,...
- In the following, we will take a look at the most important factors (to be continued in the next lecture...)

Machine Learning Winter '18

B. Leibe

57

Machine Learning Winter '18

Topics of This Lecture

- Learning Multi-layer Networks
 - Recap: Backpropagation
 - Computational graphs
 - Automatic differentiation
 - Practical issues
- Gradient Descent
 - Stochastic Gradient Descent & Minibatches
 - Choosing Learning Rates
 - Momentum
 - RMS Prop
 - Other Optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization

B. Leibe 58

Machine Learning Winter '18

Shuffling the Examples

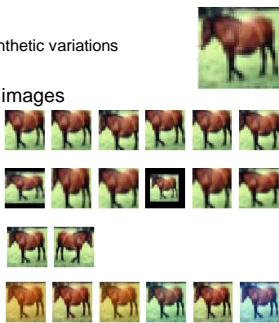
- Ideas
 - Networks learn fastest from the most unexpected sample.
 - ⇒ It is advisable to choose a sample at each iteration that is most unfamiliar to the system.
 - E.g. a sample from a *different class* than the previous one.
 - This means, do not present all samples of class A, then all of class B.
 - A large relative error indicates that an input has not been learned by the network yet, so it contains a lot of information.
 - ⇒ It can make sense to present such inputs more frequently.
 - But: be careful, this can be disastrous when the data are outliers.
- Practical advice
 - When working with stochastic gradient descent or minibatches, make use of shuffling.

B. Leibe 59

Machine Learning Winter '18

Data Augmentation

- Idea
 - Augment original data with synthetic variations to reduce overfitting
- Example augmentations for images
 - Cropping
 - Zooming
 - Flipping
 - Color PCA




B. Leibe 60

Machine Learning Winter '18

Data Augmentation

- Effect
 - Much larger training set
 - Robustness against expected variations
- During testing
 - When cropping was used during training, need to again apply crops to get same image size.
 - Beneficial to also apply flipping during test.
 - Applying several ColorPCA variations can bring another ~1% improvement, but at a significantly increased runtime.



B. Leibe 61

Machine Learning Winter '18

Practical Advice



B. Leibe 62

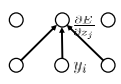
Machine Learning Winter '18

Normalization

- Motivation
 - Consider the Gradient Descent update steps

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$
 - From backpropagation, we know that

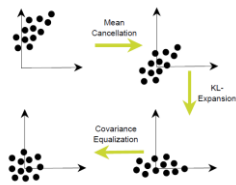
$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j}$$
 - When all of the components of the input vector y_i are positive, all of the updates of weights that feed into a node will be of the same sign.
 - ⇒ Weights can only all increase or decrease together.
 - ⇒ Slow convergence



B. Leibe 63

Normalizing the Inputs

- Convergence is fastest if
 - The mean of each input variable over the training set is zero.
 - The inputs are scaled such that all have the same covariance.
 - Input variables are uncorrelated if possible.
- Advisable normalization steps (for MLPs only, not for CNNs)
 - Normalize all inputs that an input unit sees to zero-mean, unit covariance.
 - If possible, try to decorrelate them using PCA (also known as Karhunen-Loeve expansion).



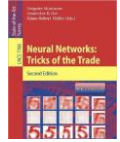
64

B. Leibe image source: Yann LeCun et al., Efficient BackProp (1998)

References and Further Reading

- More information on many practical tricks can be found in Chapter 1 of the book

G. Montavon, G. B. Orr, K-R Mueller (Eds.)
Neural Networks: Tricks of the Trade
Springer, 1998, 2012



Yann LeCun, Leon Bottou, Genevieve B. Orr, Klaus-Robert Mueller
[Efficient BackProp](#), Ch.1 of the above book., 1998.

B. Leibe

65