

RWTH AACHEN UNIVERSITY

Machine Learning – Lecture 14

Optimization / Tricks of the Trade

04.12.2019

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

Machine Learning Winter '19

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

Recap: Computational Graphs

Forward-Mode Differentiation ($\frac{\partial}{\partial X}$)

Apply operator $\frac{\partial}{\partial X}$ to every node.

Reverse-Mode Differentiation ($\frac{\partial Z}{\partial}$)

Apply operator $\frac{\partial Z}{\partial}$ to every node.

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

4

RWTH AACHEN UNIVERSITY

Recap: Automatic Differentiation

- Approach for obtaining the gradients

- 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

5

RWTH AACHEN UNIVERSITY

Recap: Choosing the Right Learning Rate

- Convergence of Gradient Descent
 - Simple 1D example
$$W^{(\tau-1)} = W^{(\tau)} - \eta \frac{dE(W)}{dW}$$
 - What is the optimal learning rate η_{opt} ?
 - If E is quadratic, the optimal learning rate is given by the inverse of the Hessian
$$\eta_{opt} = \left(\frac{d^2 E(W^{(\tau)})}{dW^2} \right)^{-1}$$
 - Advanced optimization techniques try to approximate the Hessian by a simplified form.
 - If we exceed the optimal learning rate, bad things happen!

6

RWTH AACHEN UNIVERSITY

Topics of This Lecture

- Optimization
 - Momentum
 - RMS Prop
 - Effect of optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization
- Nonlinearities
- Initialization
- Advanced techniques
 - Batch Normalization
 - Dropout

7

Machine Learning Winter '19

Batch vs. Stochastic Learning

- **Batch Learning**
 - Simplest case: steepest decent 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 9

Machine Learning Winter '19

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 10

Machine Learning Winter '19

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 decent.
- **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 11

Machine Learning Winter '19

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

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

Slide credit: Geoff Hinton B. Leibe Image source: Geoff Hinton 12

Machine Learning Winter '19

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 $\alpha = 0.99$).

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

Slide credit: Geoff Hinton B. Leibe Image source: Geoff Hinton 13

Machine Learning Winter '19

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.
 - ⇒ 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)

Slide adapted from Geoff Hinton B. Leibe Image source: Geoff Hinton 14

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

Slide adapted from Geoff Hinton

B. Leibe

15

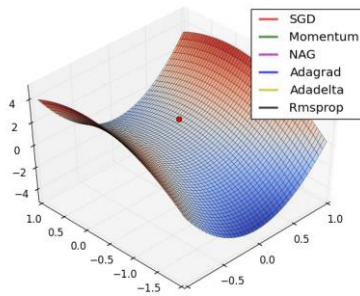
Other Optimizers

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

B. Leibe

16

Example: Behavior in a Long Valley

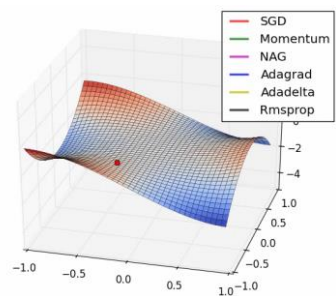


B. Leibe

Image source: Alec Radford, <http://imgur.com/a/t3ooz>

17

Example: Behavior around a Saddle Point

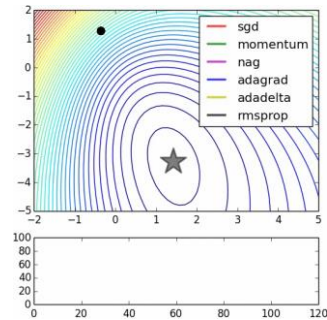


B. Leibe

Image source: Alec Radford, <http://imgur.com/a/t3ooz>

18

Visualization of Convergence Behavior



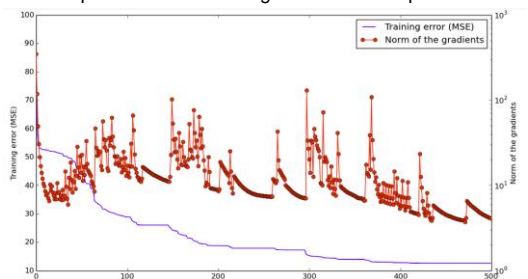
B. Leibe

Image source: Alec Radford, <http://imgur.com/SmDARz7>

19

Trick: Patience

- Saddle points dominate in high-dimensional spaces!



⇒ Learning often doesn't get stuck, you may just have to wait...

B. Leibe

Image source: Yoshua Bengio

20

Machine Learning Winter '19

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.

Slide adapted from Geoff Hinton

B. Leibe

21

Machine Learning Winter '19

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 early 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

B. Leibe

22

Machine Learning Winter '19

Topics of This Lecture

- Optimization
 - Momentum
 - RMS Prop
 - Effect of optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization
- Nonlinearities
- Initialization
- Advanced techniques
 - Batch Normalization
 - Dropout

B. Leibe

23

Machine Learning Winter '19

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

24

Machine Learning Winter '19

Data Augmentation

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

Image source: Lucas Reyer

B. Leibe

25

Machine Learning Winter '19

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.

Augmented training data (from one original image)

Image source: Lucas Reyer

B. Leibe

26

Machine Learning Winter '19

Practical Advice

RWTH AACHEN UNIVERSITY

APPLY ALL THE AUGMENTATIONS

generator.net

B. Leibe 27

Machine Learning Winter '19

Normalization

RWTH AACHEN UNIVERSITY

- 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 , 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 28

Machine Learning Winter '19

Normalizing the Inputs

RWTH AACHEN UNIVERSITY

- 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.
- Diagram illustrating:
 - Mean Cancellation
 - Covariance Equalization
 - KL-Expansion
- 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).

B. Leibe image source: Yann LeCun et al., Efficient BackProp (1998) 29

Machine Learning Winter '19

Topics of This Lecture

RWTH AACHEN UNIVERSITY

- Optimization
 - Momentum
 - RMS Prop
 - Effect of optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization
- Nonlinearities
- Initialization
- Advanced techniques
 - Batch Normalization
 - Dropout

B. Leibe 30

Machine Learning Winter '19

Commonly Used Nonlinearities

RWTH AACHEN UNIVERSITY

- Sigmoid

$$g(a) = \sigma(a) = \frac{1}{1 + \exp\{-a\}}$$
- Hyperbolic tangent

$$g(a) = \tanh(a) = 2\sigma(2a) - 1$$
- Softmax

$$g(\mathbf{a}) = \frac{\exp\{-a_i\}}{\sum_j \exp\{-a_j\}}$$

B. Leibe 31

Machine Learning Winter '19

Choosing the Right Sigmoid

RWTH AACHEN UNIVERSITY

$\tanh(a) = 2\sigma(2a) - 1$

Largest curvature at 1

- Normalization is also important for intermediate layers
 - Symmetric sigmoids, such as tanh, often converge faster than the standard logistic sigmoid.
 - Recommended sigmoid:

$$f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right)$$
 - ⇒ When used with transformed inputs, the variance of the outputs will be close to 1.

B. Leibe image source: Yann LeCun et al., Efficient BackProp (1998) 32

Machine Learning Winter '19

Usage

- Output nodes
 - Typically, a sigmoid or tanh function is used here.
 - Sigmoid for nice probabilistic interpretation (range [0,1]).
 - tanh for regression tasks
- Internal nodes
 - Historically, tanh was most often used.
 - tanh is better than sigmoid for internal nodes, since it is already centered.
 - Internally, tanh is often implemented as piecewise linear function (similar to hard tanh and maxout).
 - More recently: ReLU often used for classification tasks.

B. Leibe 33

Machine Learning Winter '19

Effect of Sigmoid Nonlinearities

- Effects of sigmoid/tanh function
 - Linear behavior around 0
 - Saturation for large inputs
- If all parameters are too small
 - Variance of activations will drop in each layer
 - Sigmoids are approximately linear close to 0
 - Good for passing gradients through, but...
 - Gradual loss of the nonlinearity
 - ⇒ No benefit of having multiple layers
- If activations become larger and larger
 - They will saturate and gradient will become zero

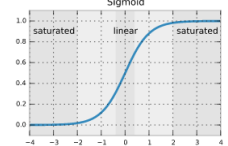
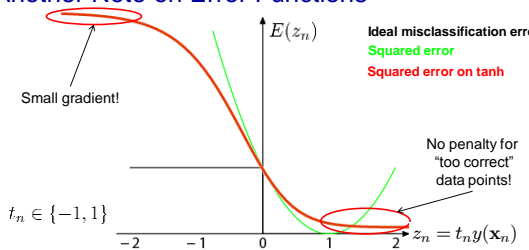


Image source: <http://deeprish.jp/2015/02/24/network-initialization>

B. Leibe 34

Machine Learning Winter '19

Another Note on Error Functions



$t_n \in \{-1, 1\}$

$z_n = t_n y(\mathbf{x}_n)$

$E(z_n)$

- Ideal misclassification error
- Squared error
- Squared error on tanh

Small gradient!

No penalty for "too correct" data points!

- Squared error on sigmoid/tanh output function
 - Avoids penalizing "too correct" data points.
 - But: almost zero gradient for confidently incorrect classifications!
 - ⇒ Do not use L_2 loss with sigmoid outputs (instead: cross-entropy)!

Image source: Bishop, 2006

B. Leibe 35

Machine Learning Winter '19

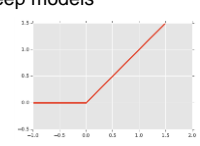
Extension: ReLU

- Another improvement for learning deep models
 - Use Rectified Linear Units (ReLU)

$$g(a) = \max\{0, a\}$$
 - Effect: gradient is propagated with a constant factor

$$\frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases}$$
- Advantages
 - Much easier to propagate gradients through deep networks.
 - We do not need to store the ReLU output separately
 - Reduction of the required memory by half compared to tanh!

⇒ ReLU has become the de-facto standard for deep networks.



B. Leibe 36

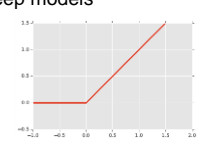
Machine Learning Winter '19

Extension: ReLU

- Another improvement for learning deep models
 - Use Rectified Linear Units (ReLU)

$$g(a) = \max\{0, a\}$$
 - Effect: gradient is propagated with a constant factor

$$\frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases}$$
- Disadvantages / Limitations
 - A certain fraction of units will remain "stuck at zero".
 - If the initial weights are chosen such that the ReLU output is 0 for the entire training set, the unit will never pass through a gradient to change those weights.
 - ReLU has an **offset bias**, since its outputs will always be positive



B. Leibe 37

Machine Learning Winter '19

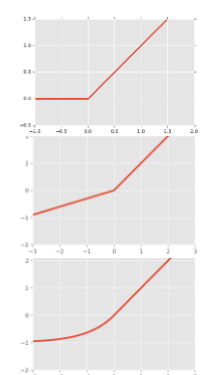
Further Extensions

- Rectified linear unit (ReLU)

$$g(a) = \max\{0, a\}$$
- Leaky ReLU

$$g(a) = \max\{\beta a, a\}$$
 - Avoids stuck-at-zero units
 - Weaker offset bias
- ELU

$$g(a) = \begin{cases} a, & x < 0 \\ e^a - 1, & x \geq 0 \end{cases}$$
 - No offset bias anymore
 - BUT: need to store activations



B. Leibe 38

Machine Learning Winter '19

Topics of This Lecture

- Optimization
 - Momentum
 - RMS Prop
 - Effect of optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization
- Nonlinearities
- Initialization
- Advanced techniques
 - Batch Normalization
 - Dropout

B. Leibe 41

Machine Learning Winter '19

Initializing the Weights

- Motivation
 - The starting values of the weights can have a significant effect on the training process.
 - Weights should be chosen randomly, but in a way that the sigmoid is primarily activated in its linear region.
- Guideline (from [LeCun et al., 1998] book chapter)
 - Assuming that
 - The training set has been normalized
 - The recommended sigmoid $f(x) = 1.7159 \tanh\left(\frac{2}{3}x\right)$ is used

the initial weights should be randomly drawn from a distribution (e.g., uniform or Normal) with mean zero and variance

$$\sigma_w^2 = \frac{1}{n_{in}}$$

where n_{in} is the fan-in (#connections into the node).

B. Leibe 42

Machine Learning Winter '19

Historical Sidenote

- Apparently, this guideline was either little known or misunderstood for a long time
 - A popular heuristic (also the standard in Torch) was to use

$$W \sim U\left[-\frac{1}{\sqrt{n_{in}}}, \frac{1}{\sqrt{n_{in}}}\right]$$
 - This looks almost like LeCun's rule. However...
- When sampling weights from a uniform distribution $[a, b]$
 - Keep in mind that the standard deviation is computed as

$$\sigma^2 = \frac{1}{12}(b-a)^2$$
 - If we do that for the above formula, we obtain

$$\sigma^2 = \frac{1}{12}\left(\frac{2}{\sqrt{n_{in}}}\right)^2 = \frac{1}{3} \frac{1}{n_{in}}$$

⇒ Activations & gradients will be attenuated with each layer! (bad)

B. Leibe 43

Machine Learning Winter '19

Glort Initialization

- Breakthrough results
 - In 2010, Xavier Glorot published an analysis of what went wrong in the initialization and derived a more general method for automatic initialization.
 - This new initialization massively improved results and made direct learning of deep networks possible overnight.
 - Let's look at his analysis in more detail...

X. Glorot, Y. Bengio, [Understanding the Difficulty of Training Deep Feedforward Neural Networks](#), AISTATS 2010.

B. Leibe 44

Machine Learning Winter '19

Analysis

- Variance of neuron activations
 - Suppose we have an input X with n components and a linear neuron with random weights W that spits out a number Y .
 - What is the variance of Y ?

$$Y = W_1X_1 + W_2X_2 + \dots + W_nX_n$$
 - If inputs and outputs have both mean 0, the variance is

$$\text{Var}(W_iX_i) = E[X_i^2]\text{Var}(W_i) + E[W_i^2]\text{Var}(X_i) + \text{Var}(W_i)\text{Var}(X_i)$$

$$= \text{Var}(W_i)\text{Var}(X_i)$$
 - If the X_i and W_i are all i.i.d, then

$$\text{Var}(Y) = \text{Var}(W_1X_1 + W_2X_2 + \dots + W_nX_n) = n\text{Var}(W_i)\text{Var}(X_i)$$

⇒ The variance of the output is the variance of the input, but scaled by $n \text{Var}(W_i)$.

B. Leibe 45

Machine Learning Winter '19

Analysis (cont'd)

- Variance of neuron activations
 - if we want the variance of the input and output of a unit to be the same, then $n \text{Var}(W_i)$ should be 1. This means

$$\text{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{in}}$$
 - If we do the same for the backpropagated gradient, we get

$$\text{Var}(W_i) = \frac{1}{n_{out}}$$
 - As a compromise, Glorot & Bengio proposed to use

$$\text{Var}(W) = \frac{2}{n_{in} + n_{out}}$$

⇒ Randomly sample the weights with this variance. That's it.

B. Leibe 46

Machine Learning Winter '19

Sidenote

- When sampling weights from a uniform distribution $[a,b]$
 - Again keep in mind that the standard deviation is computed as

$$\sigma^2 = \frac{1}{12}(b-a)^2$$
 - Glorot initialization with uniform distribution

$$W \sim U \left[-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}} \right]$$
 - Or when only taking into account the fan-in

$$W \sim U \left[-\frac{\sqrt{3}}{\sqrt{n_{in}}}, \frac{\sqrt{3}}{\sqrt{n_{in}}} \right]$$
- If this had been implemented correctly in Torch from the beginning, the Deep Learning revolution might have happened a few years earlier...*

B. Leibe 47

Machine Learning Winter '19

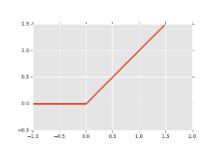
Extension to ReLU

- Important for learning deep models
 - Rectified Linear Units (ReLU)

$$g(a) = \max\{0, a\}$$
 - Effect: gradient is propagated with a constant factor

$$\frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases}$$
- We can also improve them with proper initialization
 - However, the Glorot derivation was based on tanh units, linearity assumption around zero does not hold for ReLU.
 - He et al. made the derivations, derived to use instead

$$\text{Var}(W) = \frac{2}{n_{in}}$$



B. Leibe 48

Machine Learning Winter '19

Topics of This Lecture

- Recap: Optimization
 - Effect of optimizers
- Tricks of the Trade
 - Shuffling
 - Data Augmentation
 - Normalization
- Nonlinearities
- Initialization
- Advanced techniques
 - Batch Normalization
 - Dropout

B. Leibe 49

Machine Learning Winter '19

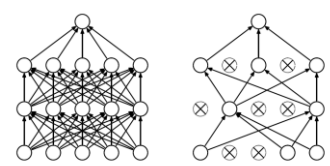
Batch Normalization [Ioffe & Szegedy '14]

- Motivation
 - Optimization works best if all inputs of a layer are normalized.
- Idea
 - Introduce intermediate layer that centers the activations of the previous layer per minibatch.
 - I.e., perform transformations on all activations and undo those transformations when backpropagating gradients
 - Complication: centering + normalization also needs to be done at test time, but minibatches are no longer available at that point.
 - Learn the normalization parameters to compensate for the expected bias of the previous layer (usually a simple moving average)
- Effect
 - Much improved convergence (but parameter values are important!)
 - Widely used in practice

B. Leibe 50

Machine Learning Winter '19

Dropout [Srivastava, Hinton '12]

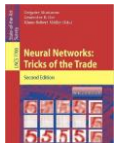


- Idea
 - Randomly switch off units during training (a form of regularization).
 - Change network architecture for each minibatch, effectively training many different variants of the network.
 - When applying the trained network, multiply activations with the probability that the unit was set to zero during training.
- ⇒ Greatly improved performance

B. Leibe 51

Machine Learning Winter '19

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. Müller (Eds.) Neural Networks: Tricks of the Trade Springer, 1998, 2012
- Yann LeCun, Leon Bottou, Genevieve B. Orr, Klaus-Robert Müller [Efficient BackProp](#), Ch. 1 of the above book., 1998.

B. Leibe 52

References

- ReLu
 - X. Glorot, A. Bordes, Y. Bengio, [Deep sparse rectifier neural networks](#), AISTATS 2011.
- Initialization
 - X. Glorot, Y. Bengio, [Understanding the difficulty of training deep feedforward neural networks](#), AISTATS 2010.
 - K. He, X.Y. Zhang, S.Q. Ren, J. Sun, [Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification](#), ArXiv 1502.01852v1, 2015.
 - A.M. Saxe, J.L. McClelland, S. Ganguli, [Exact solutions to the nonlinear dynamics of learning in deep linear neural networks](#), ArXiv 1312.6120v3, 2014.

References and Further Reading

- Batch Normalization
 - S. Ioffe, C. Szegedy, [Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#), ArXiv 1502.03167, 2015.
- Dropout
 - N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, [Dropout: A Simple Way to Prevent Neural Networks from Overfitting](#), JMLR, Vol. 15:1929-1958, 2014.