Advanced Machine Learning
Lecture 12

Tricks of the Trade II

12.12.2016

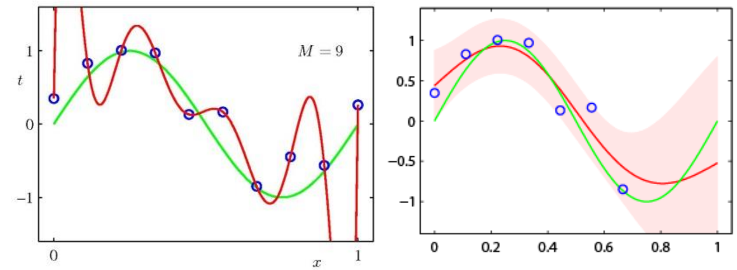
Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de/

leibe@vision.rwth-aachen.de

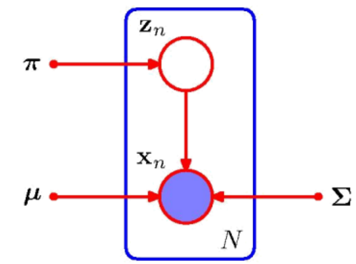
This Lecture: *Advanced Machine Learning*

- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes

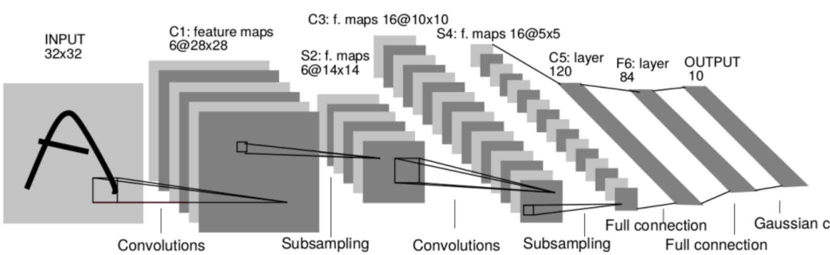
$$f : \mathcal{X} \to \mathbb{R}$$



- Approximate Inference
  - Sampling Approaches
  - MCMC



- **Deep Learning**
  - Linear Discriminants
  - Neural Networks
  - Backpropagation & **Optimization**
  - CNNs, RNNs, ResNets, etc.

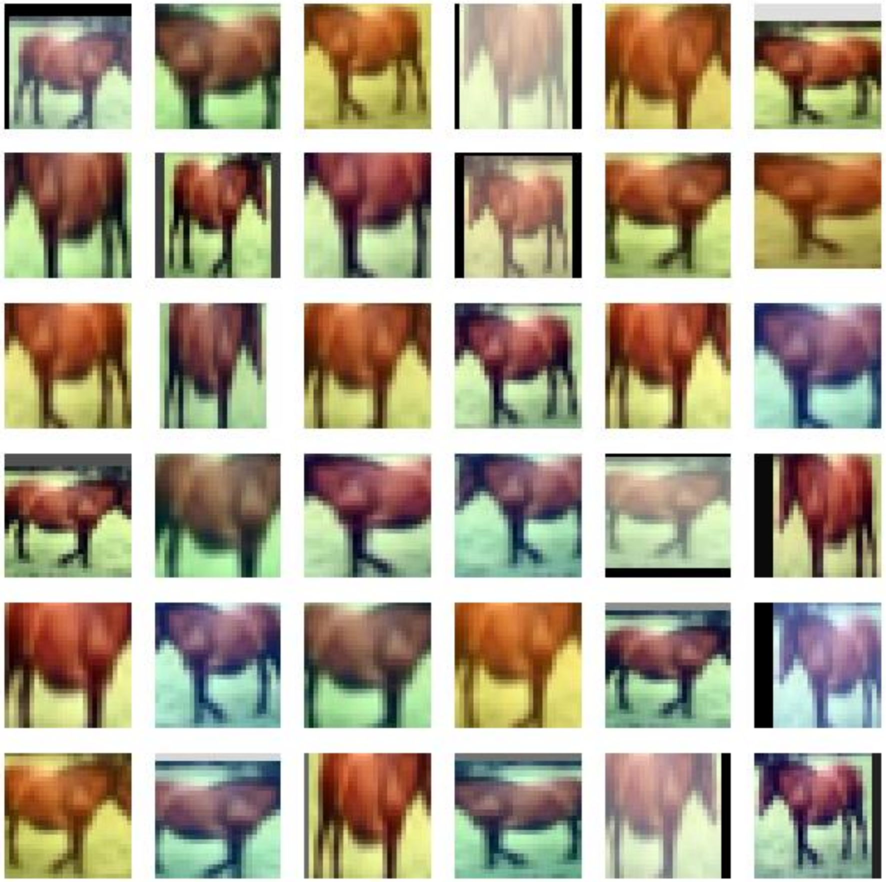


Topics of This Lecture

- Recap: Data (Pre-)processing
  - Stochastic Gradient Descent & Minibatches
  - Data Augmentation
  - Normalization
  - Initialization
- Convergence of Gradient Descent
  - Choosing Learning Rates
  - Momentum & Nesterov Momentum
  - RMS Prop
  - Other Optimizers
- Other Tricks
  - Batch Normalization
  - Dropout

Recap: 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.

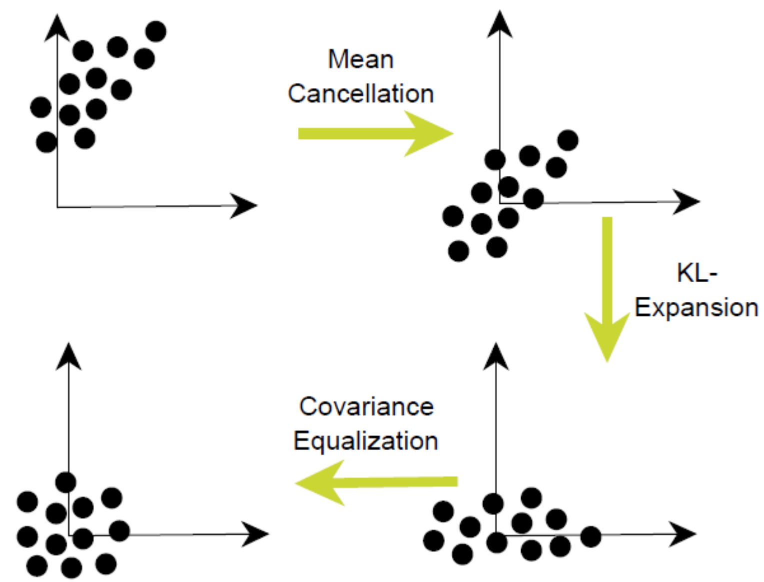


Augmented training data (from one original image)

Image source: Lucas Beyer

Recap: 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)
  - 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).

Image source: Yann LeCun et al., Efficient BackProp (1998)

Recap: Glorot Initialization

[Glorot & Bengio, '10]

- 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.
  - We *want the variance of the input and output of a unit to be the same*, therefore $n \operatorname{Var}(W_i)$ should be 1. This means

$$\operatorname{Var}(W_i) = \frac{1}{n} = \frac{1}{n_{\text{in}}}$$

  - Or for the backpropagated gradient

$$\operatorname{Var}(W_i) = \frac{1}{n_{\text{out}}}$$

  - As a compromise, Glorot & Bengio propose to use

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

$\Rightarrow$ Randomly sample the weights with this variance. That's it.

Recap: He Initialization

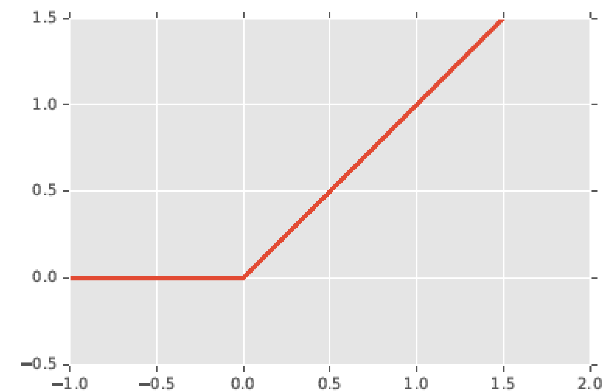
[He et al., '15]

- Extension of Glorot Initialization to ReLU units
  - Use Rectified Linear Units (ReLU)

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

  - Effect: gradient is propagated with a constant factor

$$\frac{\partial g(a)}{\partial a} = \begin{cases} 1, & a > 0 \\ 0, & \text{else} \end{cases}$$



- Same basic idea: Output should have the input variance
  - However, the Glorot derivation was based on tanh units, linearity assumption around zero does not hold for ReLU.
  - He et al. made the derivations, proposed to use instead

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

Topics of This Lecture

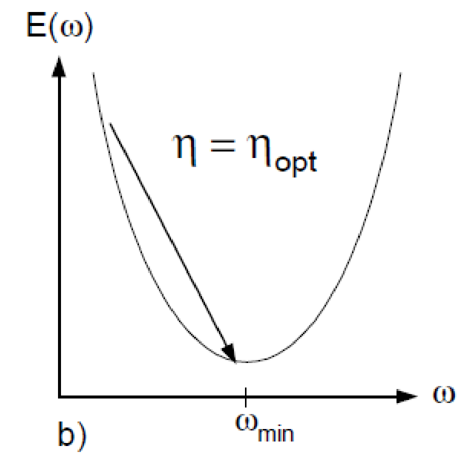
- Recap: Data (Pre-)processing
  - Stochastic Gradient Descent & Minibatches
  - Data Augmentation
  - Normalization
  - Initialization
- Convergence of Gradient Descent
  - Choosing Learning Rates
  - Momentum & Nesterov Momentum
  - RMS Prop
  - Other Optimizers
- Other Tricks
  - Batch Normalization
  - Dropout

Choosing the Right Learning Rate

- **Analyzing the convergence of Gradient Descent**
  - Consider a simple 1D example first

$$W^{(\tau-1)} = W^{(\tau)} - \eta \frac{\mathrm{d}E(W)}{\mathrm{d}W}$$

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



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

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

  - *What happens if we exceed this learning rate?*

Image source: Yann LeCun et al., Efficient BackProp (1998)

Choosing the Right Learning Rate

- **Behavior for different learning rates**

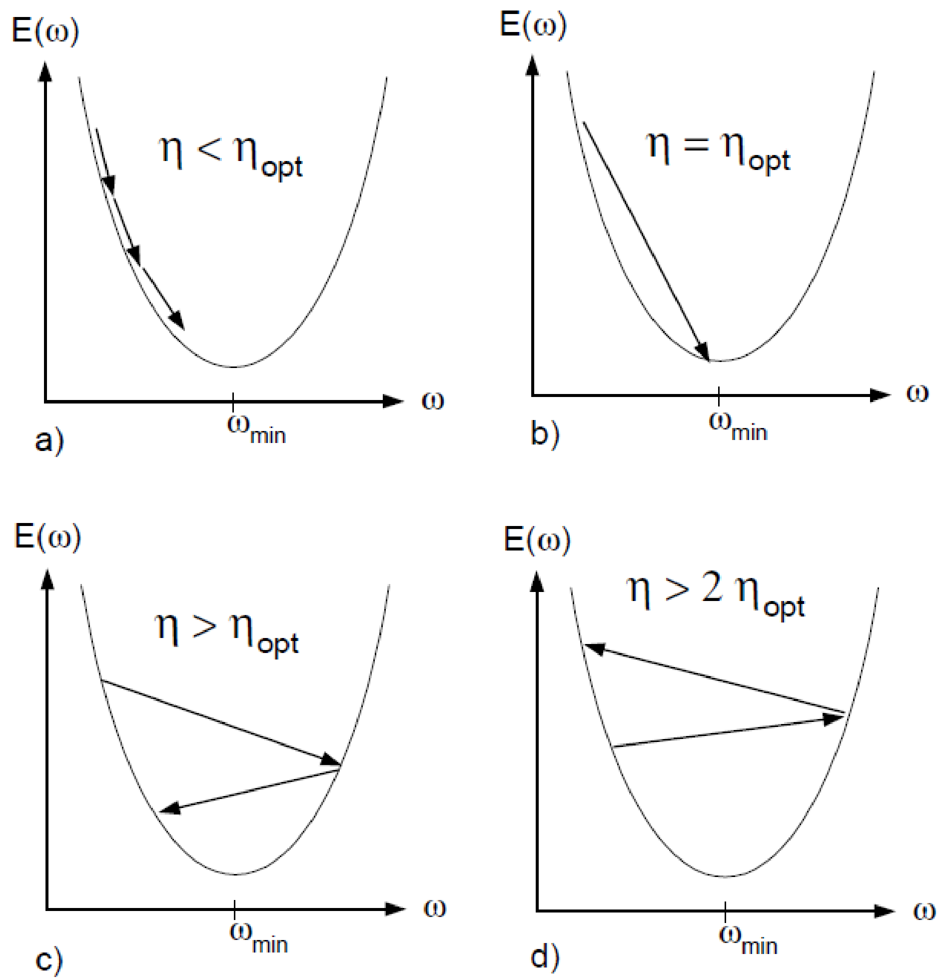


Image source: Yann LeCun et al., Efficient BackProp (1998)

Learning Rate vs. Training Error

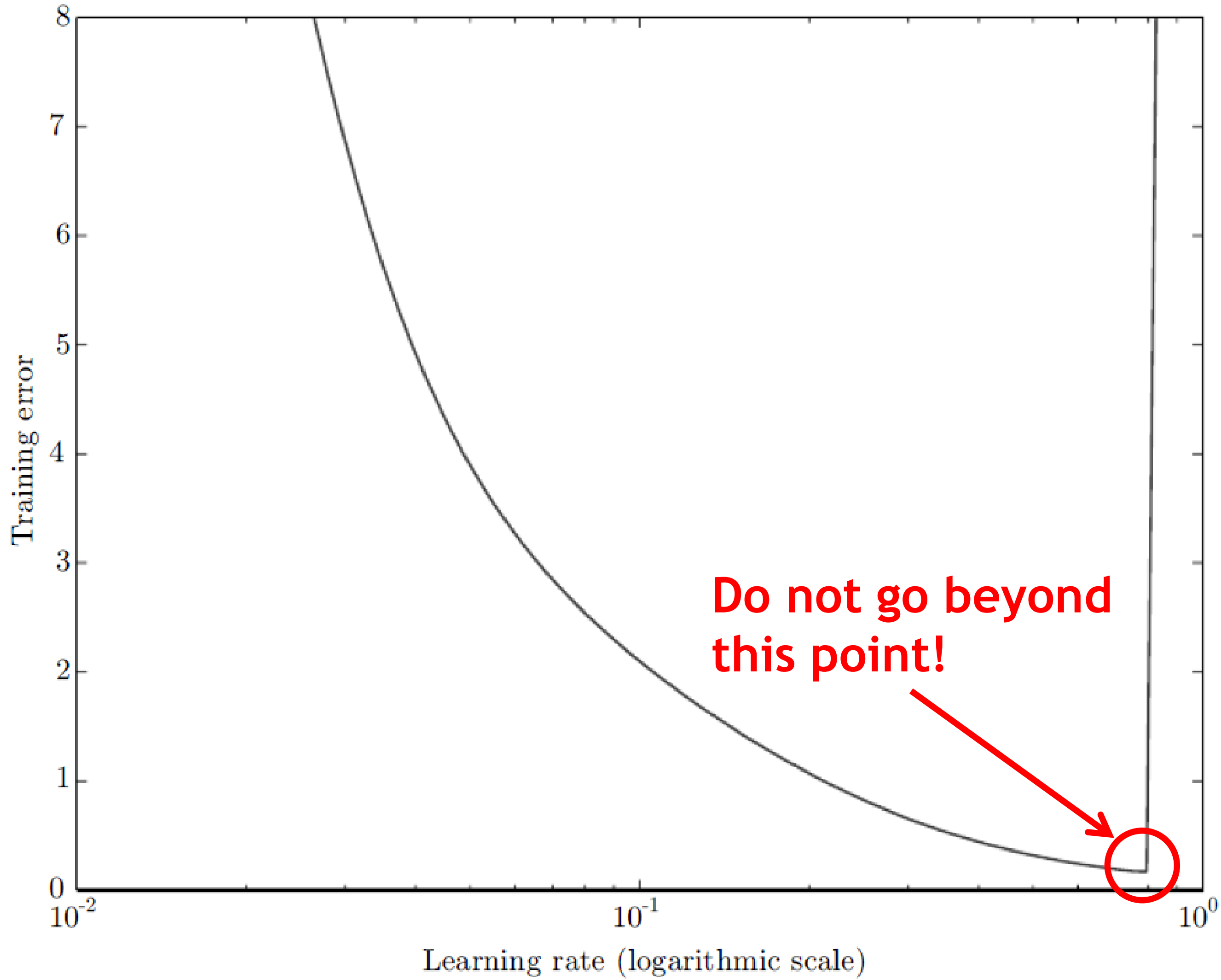
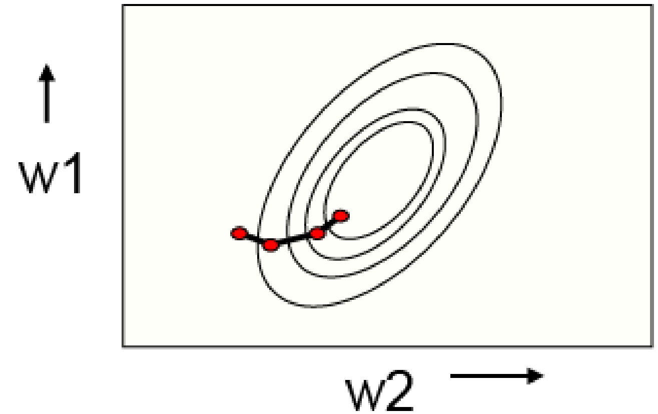


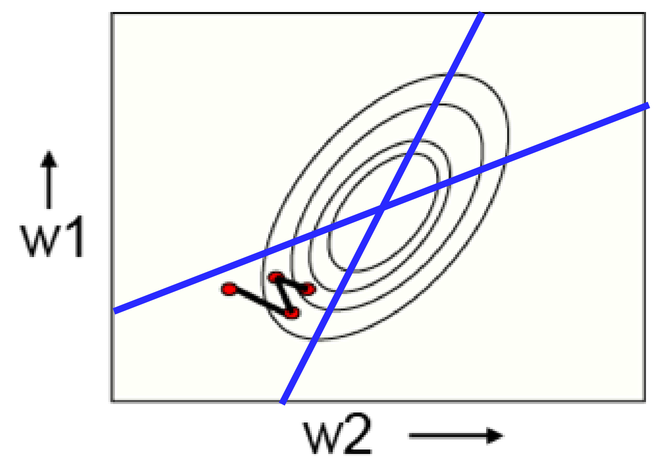
Image source: Goodfellow & Bengio book

Batch vs. Stochastic Learning

- **Batch Learning**
  - Simplest case: steepest decent on the error surface.
  - ⇒ Updates perpendicular to contour lines

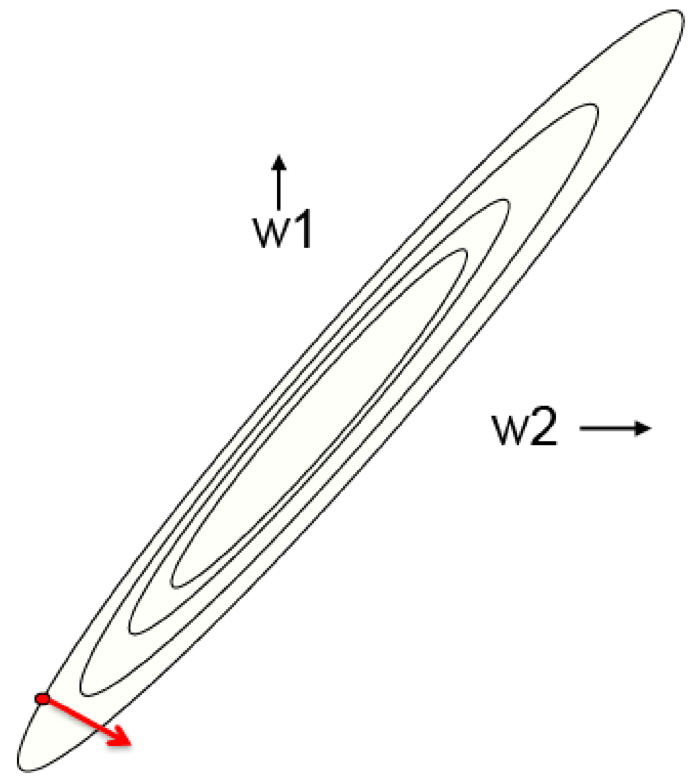


- **Stochastic Learning**
  - Simplest case: zig-zag around the direction of steepest descent.
  - ⇒ Updates perpendicular to constraints from training examples.



Why Learning Can Be Slow

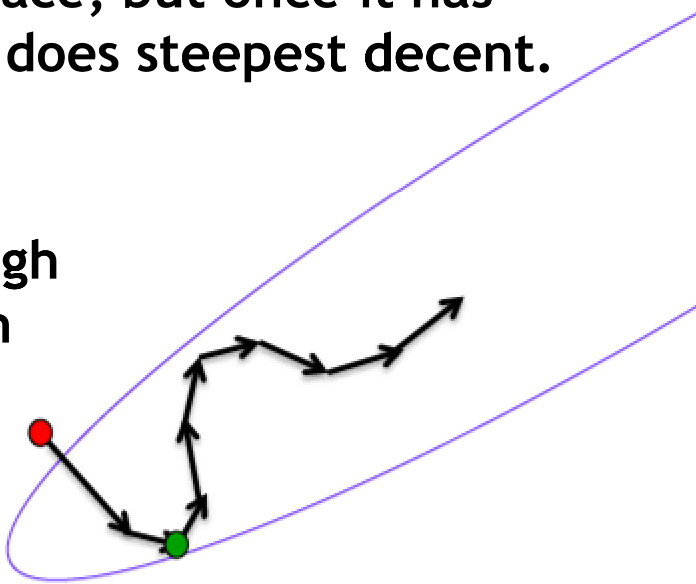
- If the inputs are correlated
  - The ellipse will be very elongated.
  - The direction of steepest descent is almost perpendicular to the direction towards the minimum!



This is just the opposite of what we want!

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.



The Momentum Method: Implementation

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

$$\mathbf{v}(t) = \alpha \mathbf{v}(t-1) - \varepsilon \frac{\partial E}{\partial \mathbf{w}}(t)$$

  - Set the weight change to the current velocity

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

Slide credit: Geoff Hinton

The Momentum Method: Behavior

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

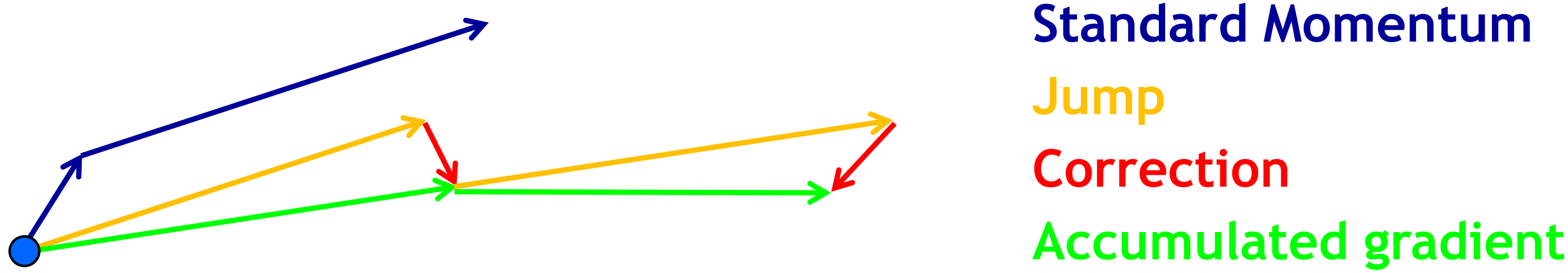
$$\mathbf{v}(\infty) = \frac{1}{1-\alpha}\left(-\varepsilon \frac{\partial E}{\partial \mathbf{w}}\right)$$

    - 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.

Improvement: Nesterov-Momentum

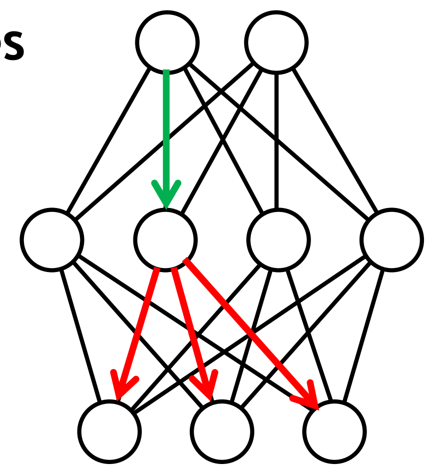


- **Standard Momentum method**
  - **First compute the gradient at the current location**
  - **Then jump in the direction of the updated accumulated gradient**
- **Improvement [Sutskever 2012]**
  - **(Inspiration: Nesterov method for optimizing convex functions.)**
  - **First jump in the direction of the previous accumulated gradient**
  - **Then measure the gradient where you end up and make a correction.**

$\Rightarrow$ ***Intuition: It's better to correct a mistake after you've made it.***

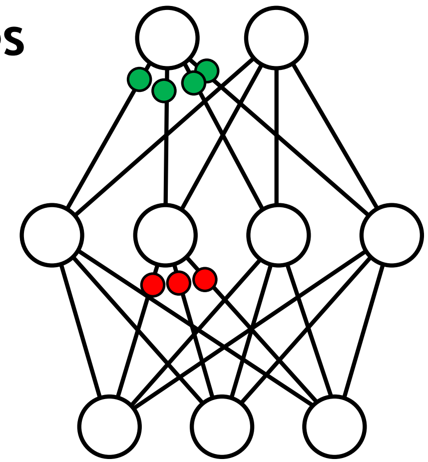
Separate, Adaptive Learning Rates

- 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.



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)

Adaptive Learning Rates

- One possible strategy
  - Start with a local gain of 1 for every weight
  - Increase the local gain if the gradient for the weight does not change the sign.
  - Use small additive increases and multiplicative decreases (for mini-batch)

$$\Delta w_{ij} = -\varepsilon g_{ij} \frac{\partial E}{\partial w_{ij}}$$

$$\text{if} \quad \left(\frac{\partial E}{\partial w_{ij}}(t) \frac{\partial E}{\partial w_{ij}}(t-1) \right) > 0$$

$$\text{then} \;\; g_{ij}(t) = g_{ij}(t-1) + 0.05$$

$$\text{else} \;\; g_{ij}(t) = g_{ij}(t-1) * 0.95$$

$\Rightarrow$ Big gains will decay rapidly once oscillation starts.

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 $\mathrm{sqrt}(MeanSq(w_{ij},t))$.

Other Optimizers (Lucas)

- 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

Behavior in a Long Valley

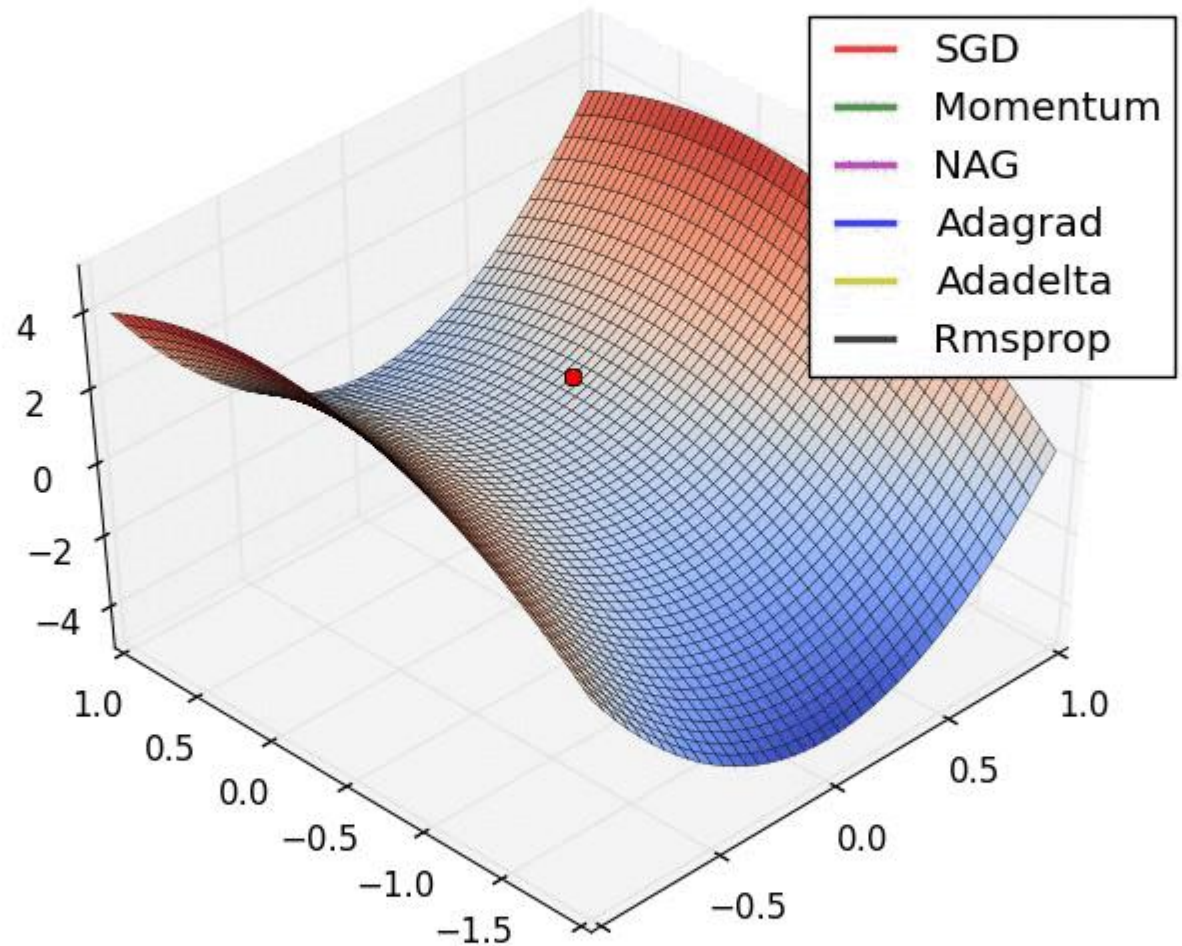


Image source: Aelc Radford, http://imgur.com/a/Hqolp

RWTH AACHEN UNIVERSITY

Behavior around a Saddle Point

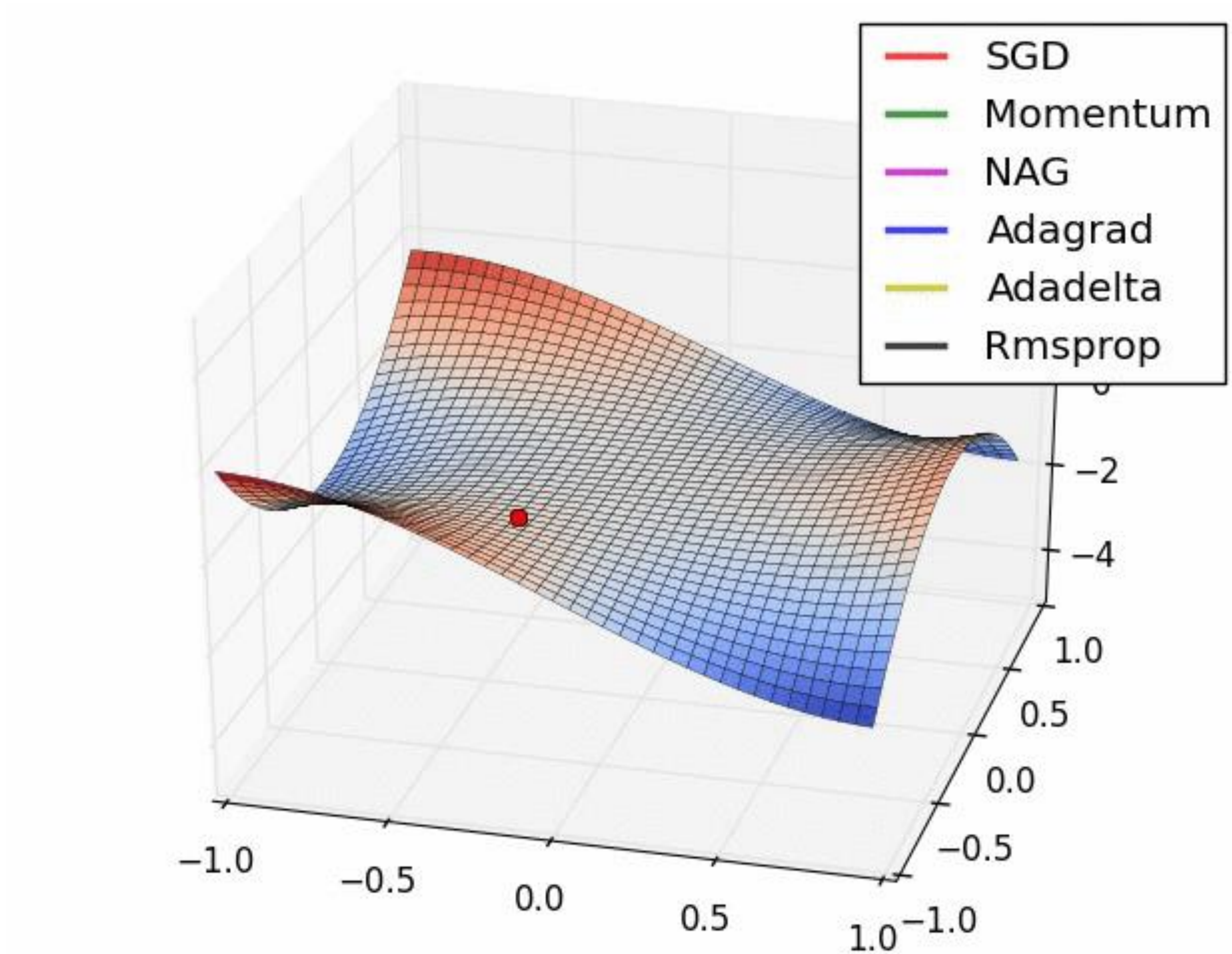


Image source: Aelc Radford, http://imgur.com/a/Hqolp

Visualization of Convergence Behavior

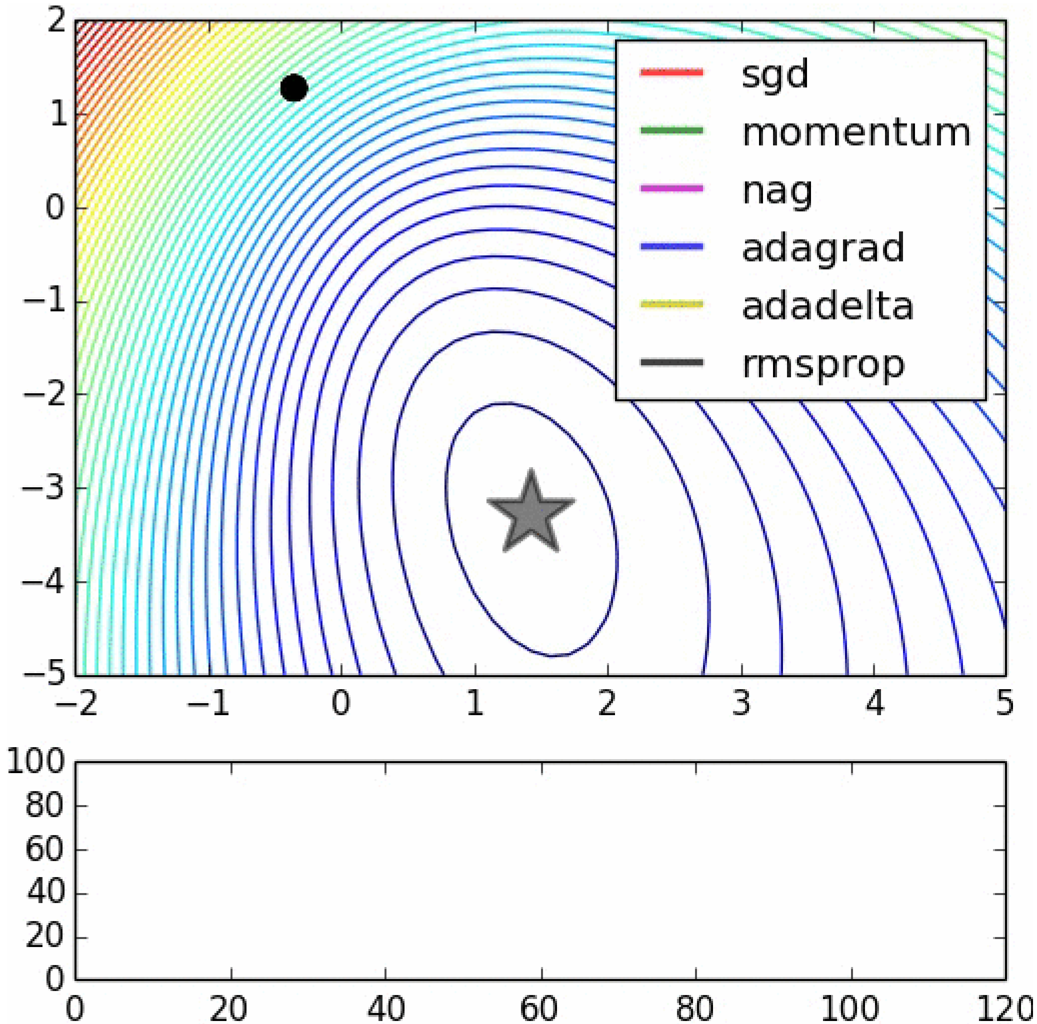
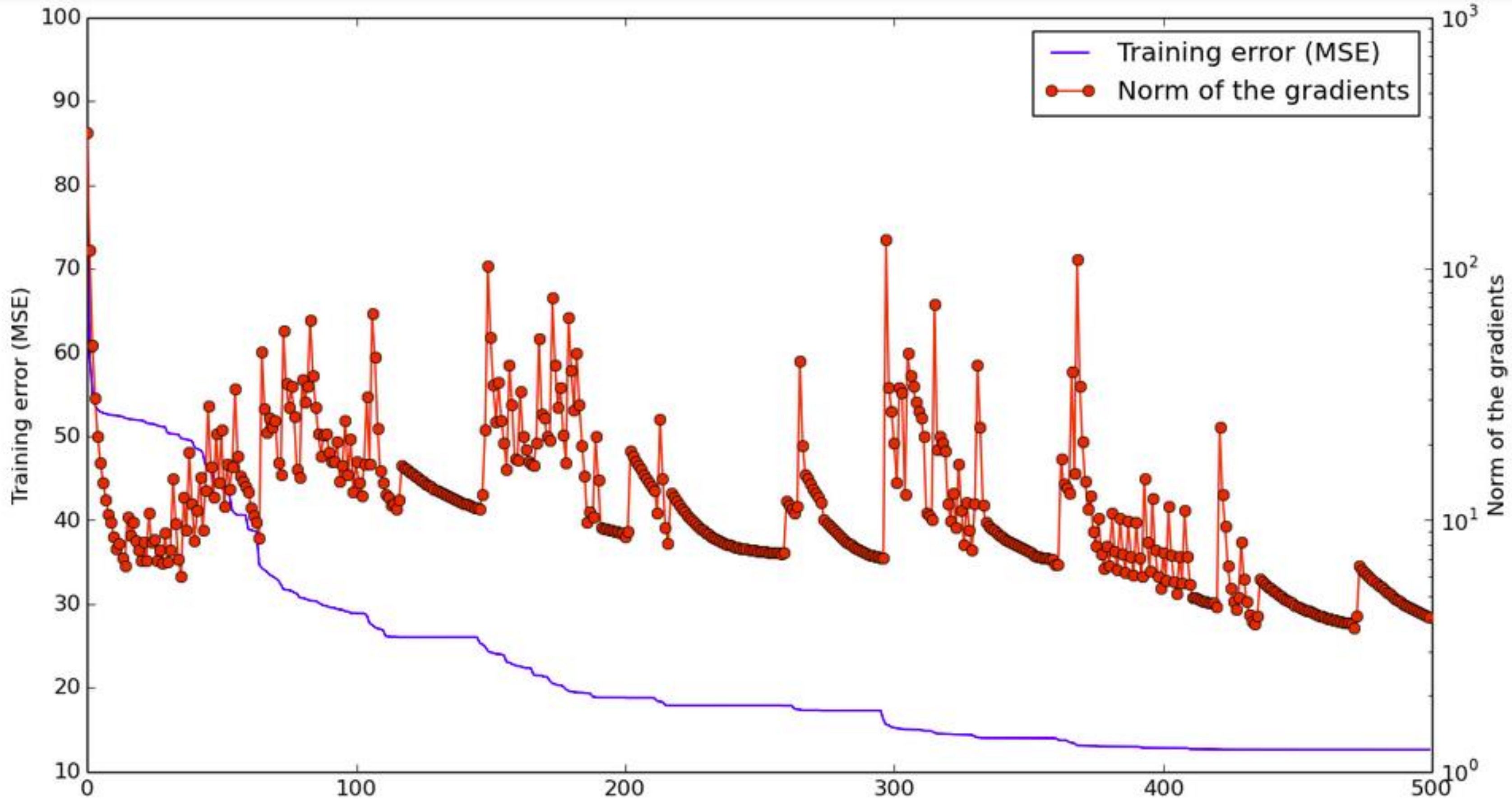


Image source: Aelc Radford, http://imgur.com/SmDARzn

Trick: Patience

- Saddle points dominate in high-dimensional spaces!

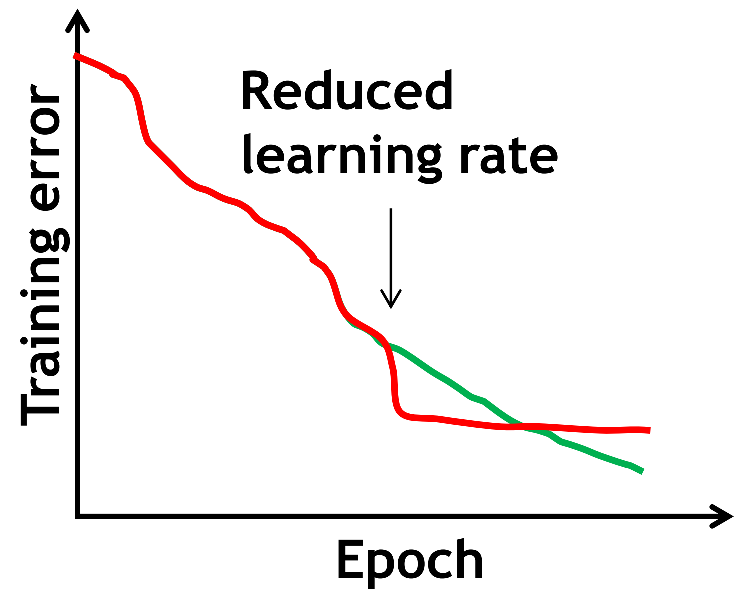


$\Rightarrow$ Learning often doesn't get stuck, you just may have to wait...

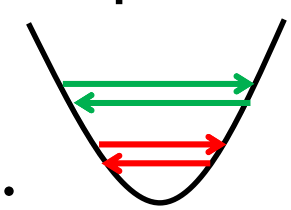
Image source: Yoshua Bengio

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 after that.

Topics of This Lecture

- Recap: Data (Pre-)processing
  - Stochastic Gradient Descent & Minibatches
  - Data Augmentation
  - Normalization
  - Initialization
- Convergence of Gradient Descent
  - Choosing Learning Rates
  - Momentum & Nesterov Momentum
  - RMS Prop
  - Other Optimizers
- **Other Tricks**
  - **Batch Normalization**
  - **Dropout**

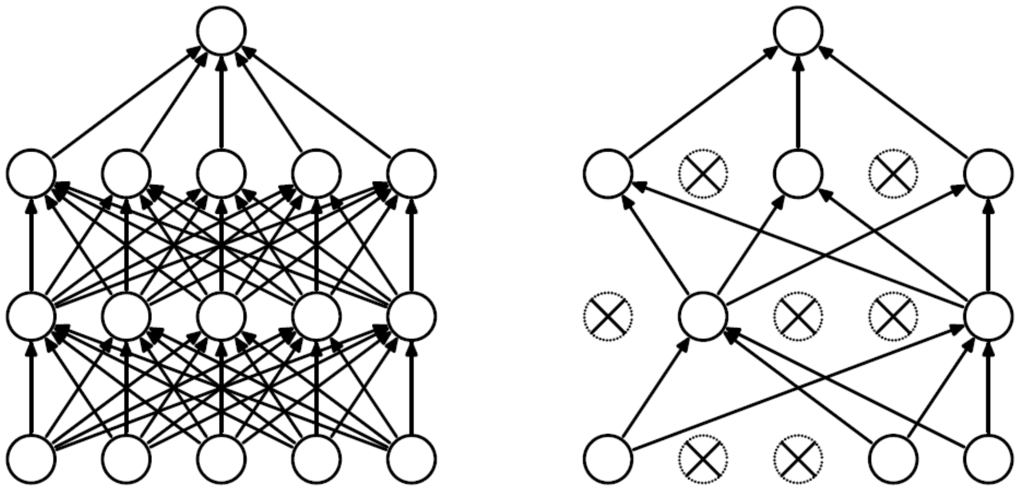
Batch Normalization [Ioffe & Szegedy '14]

- 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

- Effect
  - Much improved convergence

Dropout [Srivastava, Hinton '12]



(a) Standard Neural Net (b) After applying dropout.

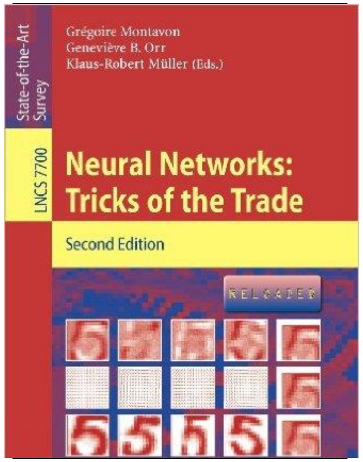
- **Idea**
  - Randomly switch off units during training.
  - Change network architecture for each data point, effectively training many different variants of the network.
  - When applying the trained network, multiply activations with the probability that the unit was set to zero.

  ⇒ Greatly improved performance

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.

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.