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

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

Recap: Computational Graphs
Forward-Mode Differentiation
Apply operator \( \frac{\partial}{\partial x} \) to every node.

Reverse-Mode Differentiation
Apply operator \( \frac{\partial Z}{\partial y} \) to every node.

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

Recap: Automatic Differentiation
• Approach for obtaining the gradients
  \( y_1(x_1), y_2(x_2), \ldots, y_n(x_n) \)

Convex hulls:
 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

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

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
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 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 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 $\alpha < 1$.
  \[
  \mathbf{v}(t) = \alpha \mathbf{v}(t-1) - \varepsilon \nabla E(\mathbf{w})
  \]
  - Set the weight change to the current velocity
  \[
  \Delta \mathbf{w} = \mathbf{v}(t)
  \]
  \[
  = \alpha \mathbf{v}(t-1) - \varepsilon \nabla E(\mathbf{w})
  \]

The Momentum Method: Behavior

- **Behavior**
  - If the error surface is a tilted plane, the ball reaches a terminal velocity
  \[
  \mathbf{v}(\infty) = \frac{1}{1 - \alpha} \left( -\frac{\partial E}{\partial \mathbf{w}} \right)
  \]
  - If the momentum $\alpha$ 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.

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

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

Example: Behavior in a Long Valley

![Image source: Alec Radford, http://imgur.com/a/Hqolp](image)

Example: Behavior around a Saddle Point

![Image source: Alec Radford, http://imgur.com/a/Hqolp](image)

Visualization of Convergence Behavior

![Image source: Alec Radford, http://imgur.com/SmDARzn](image)

Trick: Patience

- Saddle points dominate in high-dimensional spaces!

![Image source: Yoshua Bengio](image)
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.

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

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

Data Augmentation

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

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.
**Normalization**

- **Motivation**
  - Consider the Gradient Descent update steps
    \[ w_{kj}^{t+1} = w_{kj}^t - \eta \frac{\partial E(w)}{\partial w_{kj}} \]
  - From backpropagation, we know that
    \[ \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} \]
  - 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

**Normalization of 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-Loève expansion).

**Commonly Used Nonlinearities**

- **Sigmoid**
  \[ g(a) = \sigma(a) = \frac{1}{1+\exp(-a)} \]
- **Hyperbolic tangent**
  \[ g(a) = \tanh(a) = 2\sigma(2a) - 1 \]
- **Softmax**
  \[ g(a) = \frac{\exp(-a_j)}{\sum_i \exp(-a_i)} \]

**Choosing the Right Sigmoid**

- **Normalization is also important for intermediate layers**
  - Symmetric sigmoids, such as \( \tanh \), often converge faster than the standard logistic sigmoid.
  - Recommended sigmoid:
    \[ f(x) = 1.7159 \tanh \left( \frac{x}{2} \right) \]
  - When used with transformed inputs, the variance of the outputs will be close to 1.
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.

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

Another Note on Error Functions
- **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)!

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.

Further Extensions
- **Rectified linear unit (ReLU)**
  - $g(a) = \max(0, a)$
- **Leaky ReLU**
  - $g(a) = \max(a, a)$
    - Avoids stuck-at-zero units
    - Weaker offset bias
- **ELU**
  - $g(a) = \begin{cases} a, & x < 0 \\ e^x - 1, & x \geq 0 \end{cases}$
    - No offset bias anymore
    - BUT: need to store activations

Extension: ReLU
- **Another improvement for learning deep models**
  - Use Rectified Linear Units (ReLU)
  - Leaky ReLU
    - $g(a) = \max(a, a)$
    - Avoids stuck-at-zero units
    - Weaker offset bias
  - ELU
    - $g(a) = \begin{cases} a, & x < 0 \\ e^x - 1, & x \geq 0 \end{cases}$
    - No offset bias anymore
    - BUT: need to store activations
Optimization

1. **W**
   
   If we do that for the above formula, we obtain **W**

Initialization

A popular heuristic (also the standard in Torch) was to use

\[ \text{The recommended sigmoid} \] is used

Apparently, this guideline was either little known or misunderstood for a long time

- A popular heuristic (also the standard in Torch) was to use
  \[ W \sim \mathcal{U} \left( \frac{1}{\sqrt{\text{fan-in}}} \right) \]
  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
    \[ \sigma^2 = \frac{1}{\text{fan-in}} \left( \frac{1}{12} \right) = \frac{1}{12} \frac{1}{\text{fan-in}} \]
    \[ \Rightarrow \text{Activations & gradients will be attenuated with each layer! (bad)} \]

Glorot Initialization

- **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 \( f(x) = 1.7159 \tanh \left( \frac{x}{2} \right) \)
    - Where the initial weights should be randomly drawn from a distribution (e.g., uniform or Normal) with mean zero and variance
      \[ \sigma^2_{W} = \frac{1}{\text{fan-in}} \]
      where \( \text{fan-in} \) is the fan-in (connections into the node).

Analysis

- **Variance of neuron activations**
  - Suppose we have an input \( X \) with \( n \) components and a linear neuron with random weights \( W \) that splits out a number \( Y \):
    \[ Y = W_1 X_1 + W_2 X_2 + \cdots + W_n X_n \]
    - What is the variance of \( Y \)?
      \[ Y = \text{Var}(Y) = \text{Var}(W_1) \text{Var}(X_1) + \text{Var}(W_2) \text{Var}(X_2) + \cdots + \text{Var}(W_n) \text{Var}(X_n) \]
    - If the \( X_i \) and \( W_i \) are all i.i.d, then
      \[ \text{Var}(Y) = \text{Var}(W_1 X_1 + W_2 X_2 + \cdots + W_n X_n) = n \text{Var}(W) \text{Var}(X) \]
      \[ \Rightarrow \text{The variance of the output is the variance of the input, but scaled by} \ n \ \text{Var}(W) \]

- **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 \( n \ \text{Var}(W) \) should be 1. This means
      \[ \text{Var}(W) = \frac{1}{n} \]
    - If we do the same for the backpropagated gradient, we get
      \[ \text{Var}(W) = \frac{1}{n_{\text{out}}} \]
    - As a compromise, Glorot & Bengio proposed to use
      \[ \text{Var}(W) = \frac{2}{n_{\text{in}} + n_{\text{out}}} \]
      \[ \Rightarrow \text{Randomly sample the weights with this variance. That’s it.} \]
Sidenote

- When sampling weights from a uniform distribution \([a, b]\)
  
  \[ \sigma^2 = \frac{1}{12} (b - a)^2 \]

  - Again keep in mind that the standard deviation is computed as
  
  \[ \sigma^2 = \frac{1}{12} (b - a)^2 \]

  - Glorot initialization with uniform distribution
  
  \[ W \sim U - \frac{\sqrt{6}}{\sqrt{\text{fan} \text{ in}} + \sqrt{\text{fan} \text{ out}}} \]

  - Or when only taking into account the fan-in
  
  \[ W \sim U - \frac{\sqrt{6}}{\sqrt{\text{fan} \text{ in}}} \]

  - If this had been implemented correctly in Torch from the beginning, the Deep Learning revolution might have happened a few years earlier…

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

Topics of This Lecture

- Recap: Optimization
  
  - Effect of optimizers

- Tricks of the Trade
  
  - Shuffling
  - Data Augmentation
  - Normalization

- Nonlinearities

- Initialization

- Advanced techniques
  
  - 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.

  - 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

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

  Yann LeCun, Leon Bottou, Genevieve B. Orr, Klaus-Robert Mueller
References

- **ReLu**

- **Initialization**

References and Further Reading

- **Batch Normalization**

- **Dropout**