Machine Learning - Lecture 14

Deep Learning II

20.06.2016

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

- Fundamentals (2 weeks)
  - Bayes Decision Theory
  - Probability Density Estimation

- Discriminative Approaches (5 weeks)
  - Linear Discriminant Functions
  - Statistical Learning Theory & SVMs
  - Ensemble Methods & Boosting
  - Randomized Trees, Forests & Ferns
  - Deep Learning

- Generative Models (4 weeks)
  - Bayesian Networks
  - Markov Random Fields
Topics of This Lecture

• Recap: Learning Multi-layer Networks
  - Backpropagation
  - Computational graphs
  - Automatic differentiation

• Gradient Descent
  - Stochastic Gradient Descent & Minibatches
  - Data Augmentation
  - Nonlinearities
  - Choosing Learning Rates
  - Momentum
  - RMS Prop
  - Other Optimizers
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
    \[ E(W) = \sum_{n} L(t_n, y(x_n; W)) + \lambda \Omega(W) \]
    
    with a loss \(L(\cdot)\) and a regularizer \(\Omega(\cdot)\).
  - E.g., \(L(t, y(x; W)) = \sum_{n} (y(x_n; W) - t_n)^2\) \(L_2\) loss
    
    \[ \Omega(W) = \|W\|_F^2 \]
    \(L_2\) regularizer ("weight decay")

\[ \Rightarrow \text{Update each weight } W_{ij}^{(k)} \text{ in the direction of the gradient } \frac{\partial E(W)}{\partial W_{ij}^{(k)}} \]
Gradient Descent

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

last lecture today
Recap: Backpropagation Algorithm

• Core steps
  1. Convert the discrepancy between each output and its target value into an error derivative.
  2. Compute error derivatives in each hidden layer from error derivatives in the layer above.
  3. Use error derivatives w.r.t. activities to get error derivatives w.r.t. the incoming weights.

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

\[
\frac{\partial E}{\partial y_j} = -(t_j - y_j)
\]
Recap: Backpropagation Algorithm

\[ \frac{\partial E}{\partial z_j} = \frac{\partial y_j}{\partial y_j} \frac{\partial E}{\partial y} = y_j (1 - y_j) \frac{\partial E}{\partial y_j} \]

\[ \frac{\partial E}{\partial y_i} = \sum_j \frac{\partial y_j}{\partial y_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j} \]

\[ \frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j} \]

- Efficient propagation scheme
  - \( y_i \) is already known from forward pass! (Dynamic Programming)
  - Propagate back the gradient from layer \( j \) and multiply with \( y_i \).

Slide adapted from Geoff Hinton
Recap: MLP Backpropagation Algorithm

• **Forward Pass**

\[
y^{(0)} = x
\]

\[
\text{for } k = 1, \ldots, l \text{ do}
\]

\[
z^{(k)} = W^{(k)} y^{(k-1)}
\]

\[
y^{(k)} = g_k(z^{(k)})
\]

\text{endfor}

\[
y = y^{(l)}
\]

\[
E = L(t, y) + \lambda \Omega(W)
\]

• **Backward Pass**

\[
h \leftarrow \frac{\partial E}{\partial y} = \frac{\partial}{\partial y} L(t, y) + \lambda \frac{\partial}{\partial y} \Omega
\]

\[
\text{for } k = l, l-1, \ldots, 1 \text{ do}
\]

\[
h \leftarrow \frac{\partial E}{\partial z^{(k)}} = h \odot g'(y^{(k)})
\]

\[
\frac{\partial E}{\partial W^{(k)}} = hy^{(k-1)\top} + \lambda \frac{\partial \Omega}{\partial W^{(k)}}
\]

\[
h \leftarrow \frac{\partial E}{\partial y^{(k-1)}} = W^{(k)\top} h
\]

\text{endfor}

• **Notes**

- For efficiency, an entire batch of data \( X \) is processed at once.
- \( \odot \) denotes the element-wise product
Topics of This Lecture

• Recap: Learning Multi-layer Networks
  - Backpropagation
  - Computational graphs
  - Automatic differentiation

• Gradient Descent
  - Stochastic Gradient Descent & Minibatches
  - Data Augmentation
  - Nonlinearities
  - Choosing Learning Rates
  - Momentum
  - RMS Prop
  - Other Optimizers
Computational Graphs

• We can think of mathematical expressions as graphs
  - E.g., consider the expression
    \[ e = (a + b) \times (b + 1) \]
  - We can decompose this into the operations
    \[ c = a + b \]
    \[ d = b + 1 \]
    \[ e = c \times d \]
  and visualize this as a computational graph.

• Evaluating partial derivatives \( \frac{\partial Y}{\partial X} \) in such a graph
  - General rule: sum over all possible paths from \( Y \) to \( X \)
    and multiply the derivatives on each edge of the path together.

Slide inspired by Christopher Olah
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 $3 \times 3$ 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
      $$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma) \ast (\delta + \epsilon + \zeta)$$
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.

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

\[
\frac{\partial X}{\partial X} = 1 \\
\frac{\partial Y}{\partial X} = \alpha + \beta + \gamma \\
\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta)
\]

Reverse-Mode Differentiation \( \left( \frac{\partial}{\partial \bar{z}} \right) \)

\[
\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma)(\delta + \epsilon + \zeta) \\
\frac{\partial Z}{\partial Y} = \delta + \epsilon + \zeta \\
\frac{\partial Z}{\partial \bar{z}} = 1
\]

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

Apply operator \( \frac{\partial}{\partial \bar{z}} \) to every node.
Why Do We Care?

- Let’s consider the example again
  - Using forward-mode differentiation from \( b \) up...
  - Runtime: \( \mathcal{O}(\#\text{edges}) \)
  - Result: derivative of every node with respect to \( b \).
Why Do We Care?

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

\[ \frac{\partial e}{\partial b} = 5 \]
\[ \frac{\partial c}{\partial b} = 1 \]
\[ \frac{\partial a}{\partial b} = 0 \]
\[ \frac{\partial b}{\partial b} = 1 \]

\[ \frac{\partial e}{\partial d} = 3 \]
\[ \frac{\partial c}{\partial d} = 1 \]
\[ \frac{\partial d}{\partial d} = 1 \]

\[ \frac{\partial d}{\partial b} = 1 \]

\[ \frac{\partial e}{\partial c} = 2 \]

\[ \frac{\partial c}{\partial b} = 1 \]

\[ \frac{\partial d}{\partial b} = 1 \]

\[ \frac{\partial e}{\partial a} \]

⇒ This is what we want to compute in Backpropagation!

- Forward differentiation needs one pass per node. With backward differentiation can compute all derivatives in one single pass.

⇒ Speed-up in \( \mathcal{O}(\#\text{inputs}) \) compared to forward differentiation!
Topics of This Lecture

- Recap: Learning Multi-layer Networks
  - Backpropagation
  - Computational graphs
  - Automatic differentiation

- Gradient Descent
  - Stochastic Gradient Descent & Minibatches
  - Data Augmentation
  - Nonlinearities
  - Choosing Learning Rates
  - Momentum
  - RMS Prop
  - Other Optimizers
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
Modular Implementation (e.g., Torch)

- 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 $y$ of the module given its inputs $x$
     \[ y = \text{module.fprop}(x) \]
     where $x$, $y$, and intermediate results are stored in the module.
  2. Computing the gradient $\frac{\partial E}{\partial x}$ of a scalar cost w.r.t. the inputs $x$ given the gradient $\frac{\partial E}{\partial y}$ w.r.t. the outputs $y$
     \[ \frac{\partial E}{\partial x} = \text{module.bprop}(\frac{\partial E}{\partial y}) \]
Topics of This Lecture

• Recap: Learning Multi-layer Networks
  - Backpropagation
  - Computational graphs
  - Automatic differentiation

• Gradient Descent
  - Stochastic Gradient Descent & Minibatches
  - Data Augmentation
  - Nonlinearities
  - Choosing Learning Rates
  - Momentum
  - RMS Prop
  - Other Optimizers
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**
  \[ w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \frac{\partial E(w)}{\partial w_{kj}} \bigg|_{\mathbf{w}^{(\tau)}} \]

- **Main questions**
  - On what data do we want to apply this?
  - How should we choose the step size \( \eta \) (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.

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

**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.
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 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

\[
E(W) = \frac{1}{N} \sum_n L(t_n, y(x_n; W)) + \frac{\lambda}{N} \Omega(W)
\]
Data Augmentation

• Idea
  ➢ Augment original data with synthetic variations to reduce overfitting

• Example augmentations for images
  ➢ Cropping
  ➢ Zooming
  ➢ Flipping
  ➢ Color PCA

Image source: Lucas Beyer
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
General Guideline

APPLY ALL

THE AUGMENTATIONS
Topics of This Lecture

• Recap: Learning Multi-layer Networks
  - Backpropagation
  - Computational graphs
  - Automatic differentiation

• Gradient Descent
  - Stochastic Gradient Descent & Minibatches
  - Data Augmentation
  - Nonlinearities
  - Choosing Learning Rates
  - Momentum
  - RMS Prop
  - Other Optimizers
Commonly Used Nonlinearities

- **Sigmoid**
  \[ g(a) = \sigma(a) = \frac{1}{1 + \exp\{-a\}} \]

- **Hyperbolic tangent**
  \[ g(a) = \tanh(a) = 2\sigma(2a) - 1 \]

- **Softmax**
  \[ g(a) = \frac{\exp\{-a_i\}}{\sum_j \exp\{-a_j\}} \]
Commonly Used Nonlinearities (2)

- **Hard tanh**

\[ g(a) = \max \{-1, \min \{1, a\}\} \]

- **Rectified linear unit (ReLU)**

\[ g(a) = \max \{0, a\} \]

- **Maxout**

\[ g(a) = \max_i \left\{ w_i^\top a + b_i \right\} \]
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.
Topics of This Lecture

• Recap: Learning Multi-layer Networks
  - Backpropagation
  - Computational graphs
  - Automatic differentiation

• Gradient Descent
  - Stochastic Gradient Descent & Minibatches
  - Data Augmentation
  - Nonlinearities
  - Choosing Learning Rates
  - Momentum
  - RMS Prop
  - Other Optimizers
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 \( \eta_{\text{opt}} \)?
  - If \( E \) is quadratic, the optimal learning rate is given by the inverse of the Hessian
    \[ \eta_{\text{opt}} = \left( \frac{d^2E(W^{(\tau)})}{dW^2} \right)^{-1} \]
  - What happens if we exceed this learning rate?
Choosing the Right Learning Rate

- Behavior for different learning rates

\begin{align*}
\text{a) } & \quad \eta < \eta_{\text{opt}} \\
\text{b) } & \quad \eta = \eta_{\text{opt}} \\
\text{c) } & \quad \eta > \eta_{\text{opt}} \\
\text{d) } & \quad \eta > 2 \eta_{\text{opt}}
\end{align*}

Learning Rate vs. Training Error

Do not go beyond this point!

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.

Slide credit: Geoff Hinton
Image source: Geoff Hinton
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
    \[
    \Delta \mathbf{w} = \mathbf{v}(t) = \alpha \mathbf{v}(t - 1) - \varepsilon \frac{\partial E}{\partial \mathbf{w}}(t) = \alpha \Delta \mathbf{w}(t - 1) - \varepsilon \frac{\partial E}{\partial \mathbf{w}}(t)
    \]
The Momentum Method: Behavior

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

\[
MeanSq(w_{ij}, t) = 0.9MeanSq(w_{ij}, t - 1) + 0.1 \left( \frac{\partial E}{\partial w_{ij}}(t) \right)^2
\]

- Divide the gradient by \( \text{sqrt}(MeanSq(w_{ij}, t)) \).
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

Behavior around a Saddle Point
Visualization of Convergence Behavior

Trick: Patience

- Saddle points dominate in high-dimensional spaces!

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

B. Leibe

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.

- Slide adapted from Geoff Hinton
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 this lecture, we could only skim the surface. If you are interested in using Deep Learning yourself, please check out the Advanced ML lecture from last winter!
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