

RWTH AACHEN UNIVERSITY

Advanced Machine Learning Lecture 13

Backpropagation

14.12.2015

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

Advanced Machine Learning Winter'15

RWTH AACHEN UNIVERSITY

This Lecture: *Advanced Machine Learning*

- Regression Approaches
 - Linear Regression
 - Regularization (Ridge, Lasso)
 - Gaussian Processes
- Learning with Latent Variables
 - Prob. Distributions & Approx. Inference
 - Mixture Models
 - EM and Generalizations
- Deep Learning
 - Linear Discriminants
 - Neural Networks
 - Backpropagation
 - CNNs, RNNs, RBMs, etc.

B. Leibe

RWTH AACHEN UNIVERSITY

Recap: Perceptrons

- One output node per class

Output layer
Weights
Input layer

- Outputs
 - Linear outputs

$$y_k(\mathbf{x}) = \sum_{i=0}^d W_{ki} x_i$$

⇒ Can be used to do multidimensional linear regression or multiclass classification.

With output nonlinearity
 $y_k(\mathbf{x}) = g\left(\sum_{i=0}^d W_{ki} x_i\right)$

3

Advanced Machine Learning Winter'15

RWTH AACHEN UNIVERSITY

Recap: Non-Linear Basis Functions

- Straightforward generalization

Output layer
Weights
Feature layer
Mapping (fixed)
Input layer

- Outputs
 - Linear outputs

$$y_k(\mathbf{x}) = \sum_{i=0}^d W_{ki} \phi(x_i)$$

with output nonlinearity
 $y_k(\mathbf{x}) = g\left(\sum_{i=0}^d W_{ki} \phi(x_i)\right)$

4

Advanced Machine Learning Winter'15

RWTH AACHEN UNIVERSITY

Recap: Non-Linear Basis Functions

- Straightforward generalization

Output layer
Weights
Feature layer
Mapping (fixed)
Input layer

- Remarks
 - Perceptrons are generalized linear discriminants!
 - Everything we know about the latter can also be applied here.
 - Note: feature functions $\phi(\mathbf{x})$ are kept fixed, not learned!

5

Advanced Machine Learning Winter'15

RWTH AACHEN UNIVERSITY

Recap: Perceptron Learning

- Process the training cases in some permutation
 - If the output unit is correct, leave the weights alone.
 - If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
 - If the output unit incorrectly outputs a one, subtract the input vector from the weight vector.
- Translation

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta (y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn}) \phi_j(\mathbf{x}_n)$$
 - This is the Delta rule a.k.a. LMS rule!
 - ⇒ Perceptron Learning corresponds to 1st-order (stochastic) Gradient Descent of a quadratic error function!

6

Advanced Machine Learning Winter'15

RWTH AACHEN UNIVERSITY

Recap: Loss Functions

- We can now also apply other loss functions
 - L_2 loss \Rightarrow Least-squares regression

$$L(t, y(\mathbf{x})) = \sum_n (y(\mathbf{x}_n) - t_n)^2$$
 - L_1 loss: \Rightarrow Median regression

$$L(t, y(\mathbf{x})) = \sum_n |y(\mathbf{x}_n) - t_n|$$
 - Cross-entropy loss \Rightarrow Logistic regression

$$L(t, y(\mathbf{x})) = -\sum_n \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$
 - Hinge loss \Rightarrow SVM classification

$$L(t, y(\mathbf{x})) = \sum_n [1 - t_n y(\mathbf{x}_n)]_+$$
 - Softmax loss \Rightarrow Multi-class probabilistic classification

$$L(t, y(\mathbf{x})) = -\sum_n \sum_k \mathbb{I}(t_n = k) \ln \frac{\exp(y_k(\mathbf{x}))}{\sum_j \exp(y_j(\mathbf{x}))}$$

7

RWTH AACHEN UNIVERSITY

Recap: Multi-Layer Perceptrons

- Adding more layers
- Output

$$y_k(\mathbf{x}) = g^{(2)} \left(\sum_{i=0}^h W_{ki}^{(2)} g^{(1)} \left(\sum_{j=0}^d W_{ij}^{(1)} x_j \right) \right)$$

8

RWTH AACHEN UNIVERSITY

Topics of This Lecture

- Learning with Hidden Units
- Obtaining the Gradients
 - Naive analytical differentiation
 - Numeric differentiation
 - Backpropagation
 - Computational graphs
 - Automatic differentiation
- Practical Issues
 - Nonlinearities
 - Sigmoid outputs and the L_2 loss
 - Implementing Softmax correctly

9

RWTH AACHEN UNIVERSITY

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., $L(t, y(\mathbf{x}; \mathbf{W})) = \sum_n (y(\mathbf{x}_n; \mathbf{W}) - t_n)^2$ L_2 loss
 $\Omega(\mathbf{W}) = \|\mathbf{W}\|_F^2$ L_2 regularizer ("weight decay")
 \Rightarrow Update each weight $W_{ij}^{(k)}$ in the direction of the gradient $\frac{\partial E(\mathbf{W})}{\partial W_{ij}^{(k)}}$

10

RWTH AACHEN UNIVERSITY

Gradient Descent

- Two main steps
 - Computing the gradients for each weight today
 - Adjusting the weights in the direction of the gradient Thursday

11

RWTH AACHEN UNIVERSITY

Topics of This Lecture

- Learning with Hidden Units
- Obtaining the Gradients
 - Naive analytical differentiation
 - Numeric differentiation
 - Backpropagation
 - Computational graphs
 - Automatic differentiation
- Practical Issues
 - Nonlinearities
 - Sigmoid outputs and the L_2 loss
 - Implementing Softmax correctly

12

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?

Advanced Machine Learning Winter'15 13

RWTH AACHEN UNIVERSITY

Excursion: Chain Rule of Differentiation

- One-dimensional case: Scalar functions

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

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

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

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

Advanced Machine Learning Winter'15 14

RWTH AACHEN UNIVERSITY

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 .

Advanced Machine Learning Winter'15 15

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?
- ⇒ With increasing depth, there will be exponentially many paths!
- ⇒ Infeasible to compute this way.

Advanced Machine Learning Winter'15 16

RWTH AACHEN UNIVERSITY

Topics of This Lecture

- Learning with Hidden Units
- Obtaining the Gradients
 - Naive analytical differentiation
 - Numerical differentiation
 - Backpropagation
 - Computational graphs
 - Automatic differentiation
- Practical Issues
 - Nonlinearities
 - Sigmoid outputs and the L_2 loss
 - Implementing Softmax correctly

Advanced Machine Learning Winter'15 17

RWTH AACHEN UNIVERSITY

Obtaining the Gradients

- Approach 2: Numerical Differentiation

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

Advanced Machine Learning Winter'15 18

RWTH AACHEN UNIVERSITY

Topics of This Lecture

- Learning with Hidden Units
- Obtaining the Gradients
 - Naive analytical differentiation
 - Numerical differentiation
 - Backpropagation
 - Computational graphs
 - Automatic differentiation
- Practical Issues
 - Nonlinearities
 - Sigmoid outputs and the L_2 loss
 - Implementing Softmax correctly

19

RWTH AACHEN UNIVERSITY

Obtaining the Gradients

- Approach 3: Incremental Analytical Differentiation

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

- 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

20

RWTH AACHEN UNIVERSITY

Backpropagation Algorithm

- Core steps

1. Convert the discrepancy between each output and its target value into an error derivative.

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

$$\frac{\partial E}{\partial y_j} = -(t_j - y_j)$$
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

$$\frac{\partial E}{\partial y_j} \rightarrow \frac{\partial E}{\partial w_{ik}}$$

21

RWTH AACHEN UNIVERSITY

Backpropagation Algorithm

E.g. with sigmoid output nonlinearity

$$\frac{\partial E}{\partial z_j} = \frac{\partial y_j}{\partial z_j} \frac{\partial E}{\partial y_j} = y_j(1 - y_j) \frac{\partial E}{\partial y_j}$$

- Notation
 - y_j Output of layer j
 - z_j Input of layer j

Connections: $z_j = \sum_i w_{ij} y_i$
 $y_j = g(z_j)$

22

RWTH AACHEN UNIVERSITY

Backpropagation Algorithm

$$\frac{\partial E}{\partial z_j} = \frac{\partial y_j}{\partial z_j} \frac{\partial E}{\partial y_j} = y_j(1 - y_j) \frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial z_j}{\partial y_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

- Notation
 - y_j Output of layer j
 - z_j Input of layer j

Connections: $z_j = \sum_i w_{ij} y_i$
 $\frac{\partial z_j}{\partial y_i} = w_{ij}$

23

RWTH AACHEN UNIVERSITY

Backpropagation Algorithm

$$\frac{\partial E}{\partial z_j} = \frac{\partial y_j}{\partial z_j} \frac{\partial E}{\partial y_j} = y_j(1 - y_j) \frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial z_j}{\partial y_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j}$$

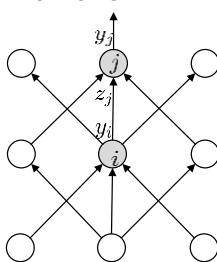
- Notation
 - y_j Output of layer j
 - z_j Input of layer j

Connections: $z_j = \sum_i w_{ij} y_i$
 $\frac{\partial z_j}{\partial w_{ij}} = y_i$

24

RWTH AACHEN UNIVERSITY

Backpropagation Algorithm



$$\frac{\partial E}{\partial z_j} = \frac{\partial y_j}{\partial z_j} \frac{\partial E}{\partial y_j} = y_j(1 - y_j) \frac{\partial E}{\partial y_j}$$

$$\frac{\partial E}{\partial y_i} = \sum_j \frac{\partial z_j}{\partial y_i} \frac{\partial E}{\partial z_j} = \sum_j w_{ij} \frac{\partial E}{\partial z_j}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial E}{\partial z_j} = y_i \frac{\partial E}{\partial z_j}$$

- 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 .

Advanced Machine Learning Winter'15 | Slide adapted from Geoff Hinton | B. Leibe | 25

RWTH AACHEN UNIVERSITY

Summary: MLP Backpropagation

- Forward Pass**

```

y(0) = x
for k = 1, ..., l do
  z(k) = W(k)y(k-1)
  y(k) = gk(z(k))
endfor
y = y(l)
E = L(t, y) + λΩ(W)

```

- Backward Pass**

```

h ← ∂E/∂y = ∂/∂y L(t, y) + λ ∂/∂y Ω
for k = l, l-1, ..., 1 do
  h ← ∂E/∂z(k) = h ⊙ g'(k)(y(k))
  ∂E/∂W(k) = h y(k-1)T + λ ∂Ω/∂W(k)
  h ← ∂E/∂y(k-1)} = W(k)T h
endfor

```

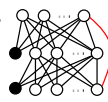
- Notes**
 - For efficiency, an entire batch of data X is processed at once.
 - \odot denotes the element-wise product

Advanced Machine Learning Winter'15 | B. Leibe | 26

RWTH AACHEN UNIVERSITY

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. ⇒ Tedious...
- Let's analyze Backprop in more detail
 - This will lead us to a more flexible algorithm formulation

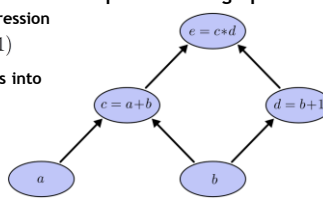


Advanced Machine Learning Winter'15 | B. Leibe | 27

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
 - $c = a + b$
 - $d = b + 1$
 - $e = c * d$
 - 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.

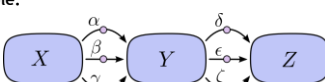


Advanced Machine Learning Winter'15 | Slide inspired by Christopher Olah | B. Leibe | Image source: Christopher Olah, colah.github.io | 28

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

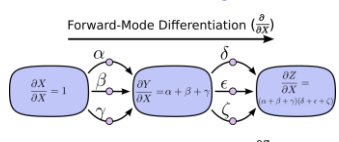
$$\frac{\partial Z}{\partial X} = (\alpha + \beta + \gamma) * (\delta + \epsilon + \zeta)$$

Advanced Machine Learning Winter'15 | Slide inspired by Christopher Olah | B. Leibe | Image source: Christopher Olah, colah.github.io | 29

RWTH AACHEN UNIVERSITY

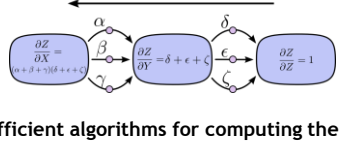
Efficient Factored Algorithms

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



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

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



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

- 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.

Advanced Machine Learning Winter'15 | Slide inspired by Christopher Olah | B. Leibe | Image source: Christopher Olah, colah.github.io | 30

Advanced Machine Learning Winter'15

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 .

Slide inspired by Christopher Olah. B. Leibe. Image source: Christopher Olah, colah.github.io

31

Advanced Machine Learning Winter'15

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.

⇒ 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!

Slide inspired by Christopher Olah. B. Leibe. Image source: Christopher Olah, colah.github.io

32

Advanced Machine Learning Winter'15

Topics of This Lecture

- Learning with Hidden Units
- Obtaining the Gradients
 - Naive analytical differentiation
 - Numerical differentiation
 - Backpropagation
 - Computational graphs
 - Automatic differentiation
- Practical Issues
 - Nonlinearities
 - Sigmoid outputs and the L_2 loss
 - Implementing Softmax correctly

B. Leibe

33

Advanced Machine Learning Winter'15

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

34

Advanced Machine Learning Winter'15

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
 - 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.
 - Computing the gradient $\partial E / \partial x$ of a scalar cost w.r.t. the inputs x given the gradient $\partial E / \partial y$ w.r.t. the outputs y

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

B. Leibe

35

Advanced Machine Learning Winter'15

Topics of This Lecture

- Learning with Hidden Units
- Obtaining the Gradients
 - Naive analytical differentiation
 - Numerical differentiation
 - Backpropagation
 - Computational graphs
 - Automatic differentiation
- Practical Issues
 - Nonlinearities
 - Sigmoid outputs and the L_2 loss
 - Implementing Softmax correctly
 - Efficient batch processing

B. Leibe

36

Advanced Machine Learning Winter'15

Commonly Used Nonlinearities

- Sigmoid

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

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

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

B. Leibe 37

Advanced Machine Learning Winter'15

Commonly Used Nonlinearities (2)

- Hard tanh

$$g(a) = \max\{-1, \min\{1, a\}\}$$
- Rectified linear unit (ReLU)

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

$$g(\mathbf{a}) = \max_i \{\mathbf{w}_i^\top \mathbf{a} + b_i\}$$

B. Leibe 38

Advanced Machine Learning Winter'15

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 40

Advanced Machine Learning Winter'15

Topics of This Lecture

- Learning with Hidden Units
- Obtaining the Gradients
 - Naive analytical differentiation
 - Numeric differentiation
 - Backpropagation
 - Computational graphs
 - Automatic differentiation
- Practical Issues
 - Nonlinearities
 - Sigmoid outputs and the L_2 loss
 - Implementing Softmax correctly

B. Leibe 41

Advanced Machine Learning Winter'15

Another Note on Error Functions

- Squared error on sigmoid/tanh output function
 - Avoids penalizing “too correct” data points.
 - But: zero gradient for confidently incorrect classifications!

⇒ Do not use L_2 loss with sigmoid outputs (instead: cross-entropy)!

Image source: Bishop, 2006

B. Leibe 42

Advanced Machine Learning Winter'15

Topics of This Lecture

- Learning with Hidden Units
- Obtaining the Gradients
 - Naive analytical differentiation
 - Numerical differentiation
 - Backpropagation
 - Computational graphs
 - Automatic differentiation
- Practical Issues
 - Nonlinearities
 - Sigmoid outputs and the L_2 loss
 - Implementing Softmax correctly

B. Leibe 43

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 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.

References and Further Reading

- More information on Backpropagation can be found in Chapter 6 of the Goodfellow & Bengio book

Ian Goodfellow, Aaron Courville, Yoshua Bengio
Deep Learning
MIT Press, in preparation



<https://goodfeli.github.io/dlbook/>