



# Natural Language Processing

## Lecture 04 Artificial Neural Networks; CNN for Text Classification

Qun Liu, Valentin Malykh  
Huawei Noah's Ark Lab



Spring 2020  
A course delivered at MIPT, Moscow



# Content

- 1 Artificial neural networks (ANNs)
- 2 Multilayer perceptron (MLP)
- 3 Convolutional neural networks (CNNs)
- 4 Convolutional Networks for Text Classification



# Content

- 1 Artificial neural networks (ANNs)
- 2 Multilayer perceptron (MLP)
- 3 Convolutional neural networks (CNNs)
- 4 Convolutional Networks for Text Classification

# Limitation of linear classifiers

- Logistic regression is a linear classifier, which means its decision boundary is a hyperplane:

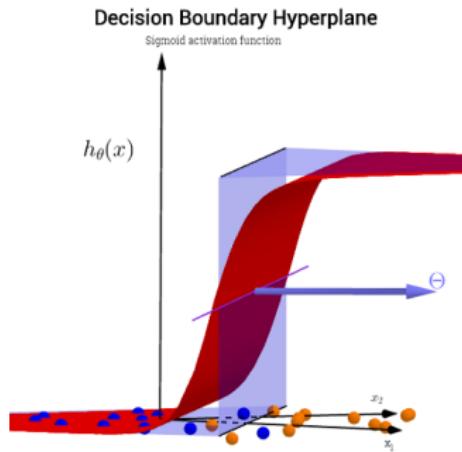


Figure source: <https://stats.stackexchange.com/questions/291492/>



# Limitation of linear classifiers

- However, some of the classification problems are not linear separable:

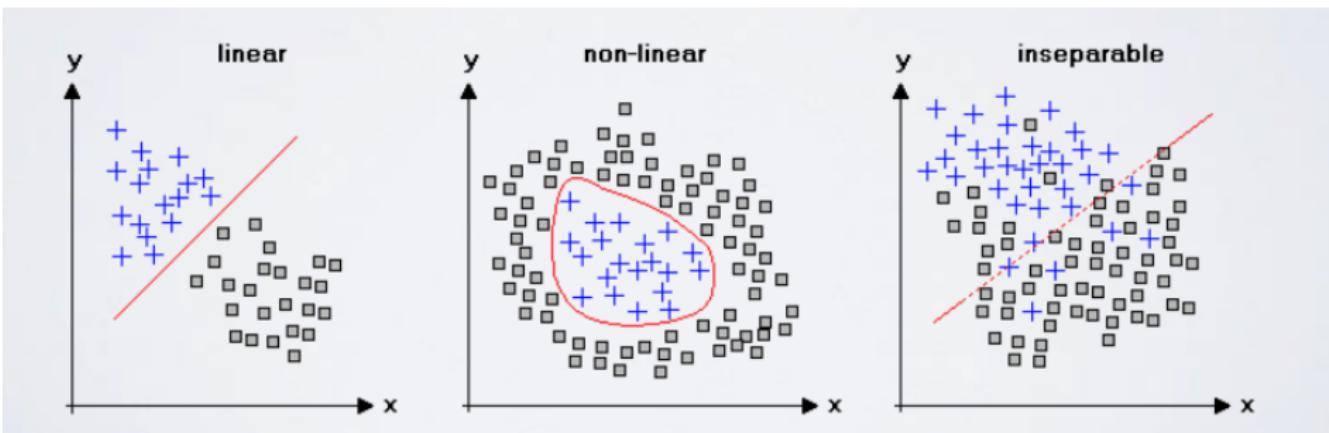


Figure source: <http://dingyancs.blogspot.com/2017/10/mle-linear-classification.html>



# Exclusive-or function

- A typical linear-inseparable problem is the exclusive-or function:

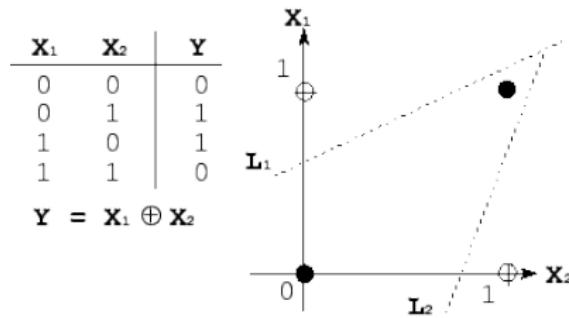


Figure source: <http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node19.html>

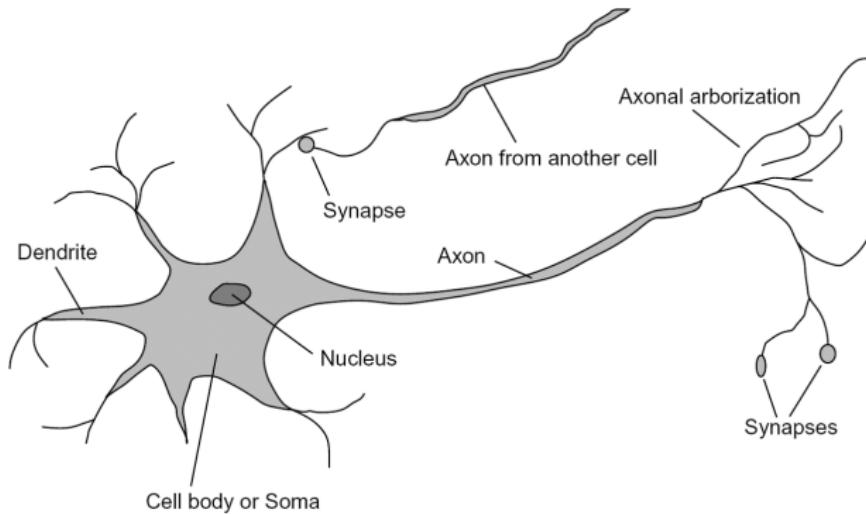


# Artificial Neural networks (ANNs)

- Neural networks (NNs), or artificial neural networks (ANNs), provide an efficient way to solve non-linear-separable problem.
- Artificial neural networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. – Wikipedia
  - An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain.
  - Each connection, like the synapses in a biological brain, can transmit a signal to other neurons.
  - An artificial neuron that receives a signal then processes it and can signal neurons connected to it.

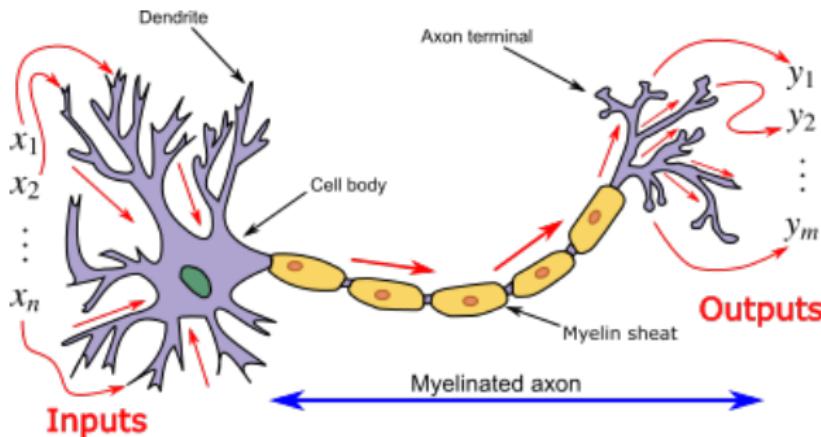


# Neuron



Structure of a neuron

# Neuron, myelinated axon, and the signal flow



Neuron and myelinated axon, with signal flow from inputs at dendrites to outputs at axon terminals

By Egm4313.s12 (Prof. Loc Vu-Quoc) - Own work, CC BY-SA 4.0, <https://commons.wikimedia.org/w/index.php?curid=72816083>



# Content

- 1 Artificial neural networks (ANNs)
- 2 Multilayer perceptron (MLP)
- 3 Convolutional neural networks (CNNs)
- 4 Convolutional Networks for Text Classification



# Content

2

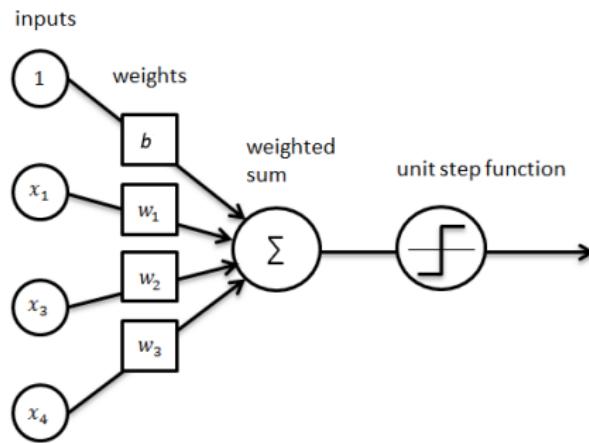
## Multilayer perceptron (MLP)

- Multilayer perceptron
- Backpropagation
- Notation in vectorized form: Jacobian Matrix
- Capacity of multilayer perceptron



# Perceptron – simplest artificial neuron

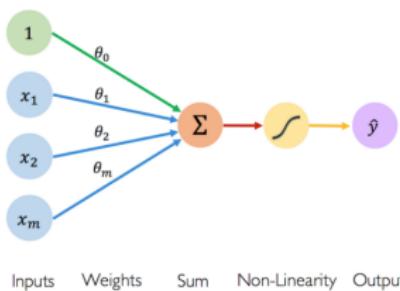
$$f(x) = \begin{cases} 0, & \text{if } \sum_i w_i x_i + b \leq 0 \\ 1, & \text{otherwise} \end{cases}$$





# Improved perceptron – logistic regression

- Perceptron is also an effective linear classifier but we cannot apply gradient descent algorithm to train it.
- Logistic regression can be regarded as improved version of perceptron to which gradient descent training can be applied.

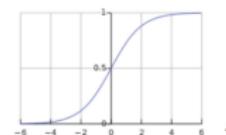


Activation Functions

$$\hat{y} = g(\theta_0 + \mathbf{x}^T \boldsymbol{\theta})$$

- Example: sigmoid function

$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



MIT: Alexander Amini, 2018 introtodeeplearning.com



# From linear classifier to non-linear classifier

- Perceptrons (including Logistic regression classifiers) are linear classifiers, which do not work for non-linear-separable problems.
- Multilayer perceptrons are proposed to solve non-linear-separable problems.

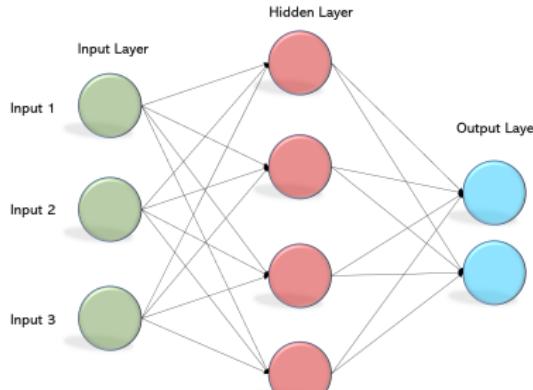


Figure source: <https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f>



# Multilayer perceptrons

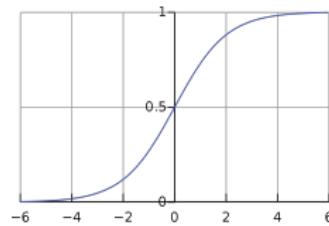
- An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer.
- Except for the input nodes, each node is a neuron that uses a nonlinear activation function.
- MLP utilizes a supervised learning technique called backpropagation for training.
- Its multiple layers and non-linear activation distinguish MLP from a linear perceptron.
- It can distinguish data that is not linearly separable.



# Activation functions

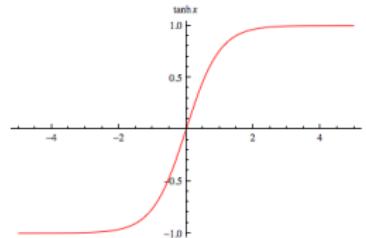
Logistic:

$$f(x) = \frac{1}{1 + e^{-x}}$$



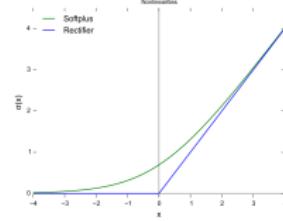
Hypbolic tangent:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



Rectifier:

$$f(x) = \max(0, x)$$





# Fully connected layers

- The MLP consists of three or more layers (an input and an output layer with one or more hidden layers) of nonlinearly-activating nodes.
- Since MLPs are fully connected, each node in one layer connects with a certain weight  $w_{ij}$  to every node in the following layer.

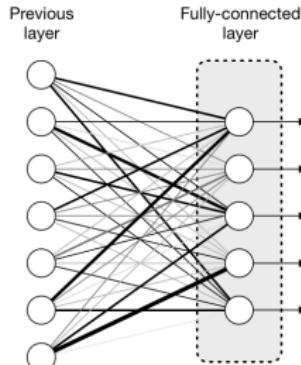


Figure source: <https://mc.ai/fully-connected-layer-with-dynamic-input-shape/>



# Output layer

- Output layer may use active functions differ from hidden layers, depending on the task nature.
- For classification problems, typically, a softmax function is used in the output layer.

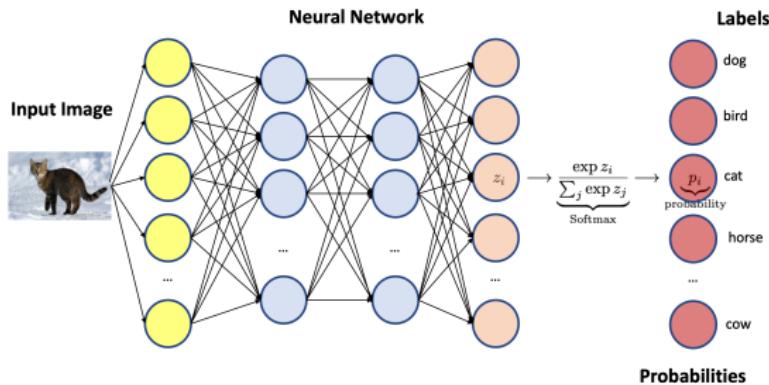


Figure source: <https://gab41.lab41.org/entropic-ghosts-35670292bc87>



# Content

2

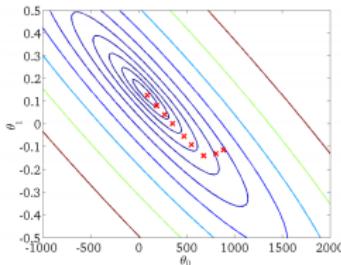
## Multilayer perceptron (MLP)

- Multilayer perceptron
- **Backpropagation**
- Notation in vectorized form: Jacobian Matrix
- Capacity of multilayer perceptron



## Recap: Gradient Descent

- **Recall:** gradient descent moves opposite the gradient (the direction of steepest descent)



- Weight space for a multilayer neural net: one coordinate for each weight or bias of the network, in *all* the layers
- Conceptually, not any different from what we've seen so far — just higher dimensional and harder to visualize!
- We want to compute the cost gradient  $d\mathcal{E}/d\mathbf{w}$ , which is the vector of partial derivatives.
  - This is the average of  $d\mathcal{L}/d\mathbf{w}$  over all the training examples, so in this lecture we focus on computing  $d\mathcal{L}/d\mathbf{w}$ .

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Univariate Chain Rule

- We've already been using the univariate Chain Rule.
- Recall: if  $f(x)$  and  $x(t)$  are univariate functions, then

$$\frac{d}{dt} f(x(t)) = \frac{df}{dx} \frac{dx}{dt}.$$

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Univariate Chain Rule

**Recall: Univariate logistic least squares model**

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Let's compute the loss derivatives.

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Univariate Chain Rule

How you would have done it in calculus class

$$\begin{aligned}\mathcal{L} &= \frac{1}{2}(\sigma(wx + b) - t)^2 \\ \frac{\partial \mathcal{L}}{\partial w} &= \frac{\partial}{\partial w} \left[ \frac{1}{2}(\sigma(wx + b) - t)^2 \right] \\ &= \frac{1}{2} \frac{\partial}{\partial w} (\sigma(wx + b) - t)^2 \\ &= (\sigma(wx + b) - t) \frac{\partial}{\partial w} (\sigma(wx + b) - t) \\ &= (\sigma(wx + b) - t) \sigma'(wx + b) \frac{\partial}{\partial w} (wx + b) \\ &= (\sigma(wx + b) - t) \sigma'(wx + b) x\end{aligned}\quad \begin{aligned}\frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial}{\partial b} \left[ \frac{1}{2}(\sigma(wx + b) - t)^2 \right] \\ &= \frac{1}{2} \frac{\partial}{\partial b} (\sigma(wx + b) - t)^2 \\ &= (\sigma(wx + b) - t) \frac{\partial}{\partial b} (\sigma(wx + b) - t) \\ &= (\sigma(wx + b) - t) \sigma'(wx + b) \frac{\partial}{\partial b} (wx + b) \\ &= (\sigma(wx + b) - t) \sigma'(wx + b)\end{aligned}$$

What are the disadvantages of this approach?

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Univariate Chain Rule

A more structured way to do it

Computing the derivatives:

Computing the loss:

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

$$\frac{d\mathcal{L}}{dy} = y - t$$

$$\frac{d\mathcal{L}}{dz} = \frac{d\mathcal{L}}{dy} \sigma'(z)$$

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{d\mathcal{L}}{dz} \times$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{d\mathcal{L}}{dz}$$

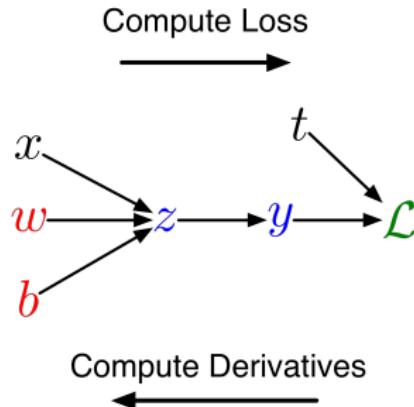
Remember, the goal isn't to obtain closed-form solutions, but to be able to write a program that efficiently computes the derivatives.

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



## Univariate Chain Rule

- We can diagram out the computations using a computation graph.
- The nodes represent all the inputs and computed quantities, and the edges represent which nodes are computed directly as a function of which other nodes.



A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Univariate Chain Rule

## A slightly more convenient notation:

- Use  $\bar{y}$  to denote the derivative  $d\mathcal{L}/dy$ , sometimes called the **error signal**.
- This emphasizes that the error signals are just values our program is computing (rather than a mathematical operation).
- This is not a standard notation, but I couldn't find another one that I liked.

## Computing the loss:

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

## Computing the derivatives:

$$\bar{y} = y - t$$

$$\bar{z} = \bar{y} \sigma'(z)$$

$$\bar{w} = \bar{z} x$$

$$\bar{b} = \bar{z}$$

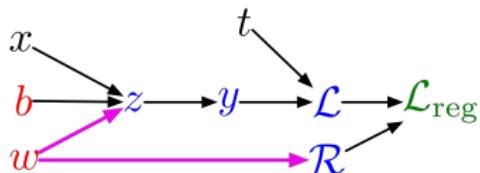
A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Multivariate Chain Rule

**Problem:** what if the computation graph has **fan-out** > 1?  
 This requires the **multivariate Chain Rule!**

## $L_2$ -Regularized regression



$$z = wx + b$$

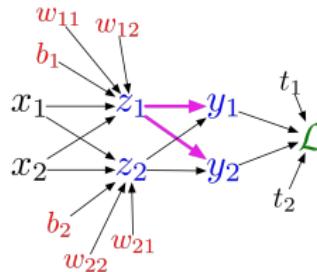
$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

$$\mathcal{R} = \frac{1}{2}w^2$$

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda \mathcal{R}$$

## Multiclass logistic regression



$$\mathcal{L}_\ell = \sum_j w_{\ell j} x_j + b_\ell$$

$$y_k = \frac{e^{z_k}}{\sum_\ell e^{z_\ell}}$$

$$\mathcal{L} = - \sum_k t_k \log \sigma(\mathcal{L}_k)$$

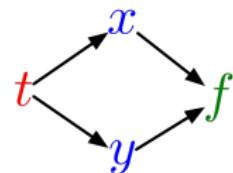
A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



## Multivariate Chain Rule

- Suppose we have a function  $f(x, y)$  and functions  $x(t)$  and  $y(t)$ . (All the variables here are scalar-valued.) Then

$$\frac{d}{dt} f(x(t), y(t)) = \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt}$$



- Example:

$$f(x, y) = y + e^{xy}$$

$$x(t) = \cos t$$

$$y(t) = t^2$$

- Plug in to Chain Rule:

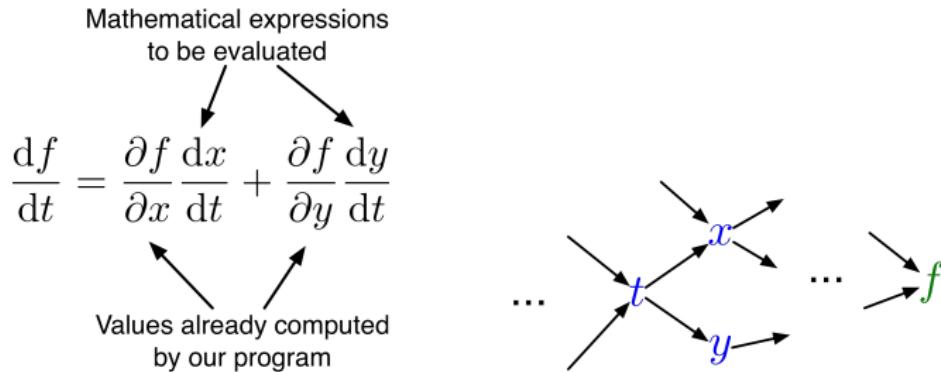
$$\begin{aligned}\frac{df}{dt} &= \frac{\partial f}{\partial x} \frac{dx}{dt} + \frac{\partial f}{\partial y} \frac{dy}{dt} \\ &= (ye^{xy}) \cdot (-\sin t) + (1 + xe^{xy}) \cdot 2t\end{aligned}$$

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



## Multivariable Chain Rule

- In the context of backpropagation:



- In our notation:

$$\bar{t} = \bar{x} \frac{dx}{dt} + \bar{y} \frac{dy}{dt}$$

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation

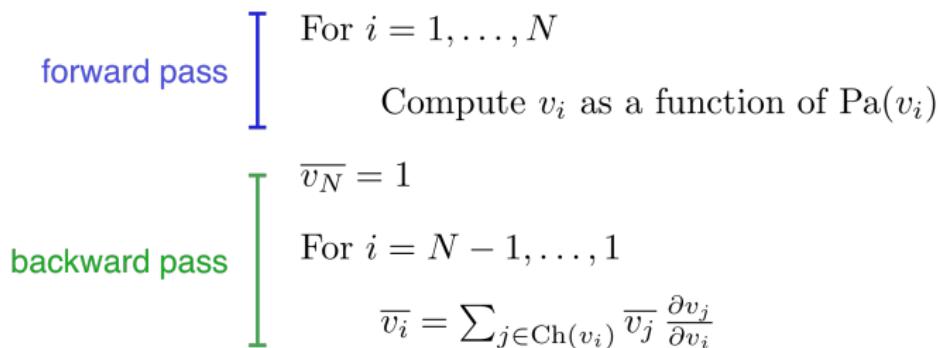


# Backpropagation

## Full backpropagation algorithm:

Let  $v_1, \dots, v_N$  be a **topological ordering** of the computation graph  
(i.e. parents come before children.)

$v_N$  denotes the variable we're trying to compute derivatives of (e.g. loss).

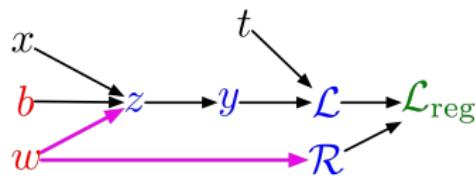


A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Backpropagation

**Example:** univariate logistic least squares regression



**Forward pass:**

$$z = wx + b$$

$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

$$\mathcal{R} = \frac{1}{2}w^2$$

$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

**Backward pass:**

$$\overline{\mathcal{L}}_{\text{reg}} = 1$$

$$\begin{aligned}\overline{\mathcal{R}} &= \overline{\mathcal{L}}_{\text{reg}} \frac{d\mathcal{L}_{\text{reg}}}{d\mathcal{R}} \\ &= \overline{\mathcal{L}}_{\text{reg}} \lambda\end{aligned}$$

$$\begin{aligned}\overline{\mathcal{L}} &= \overline{\mathcal{L}}_{\text{reg}} \frac{d\mathcal{L}_{\text{reg}}}{d\mathcal{L}} \\ &= \overline{\mathcal{L}}_{\text{reg}}\end{aligned}$$

$$\begin{aligned}\overline{y} &= \overline{\mathcal{L}} \frac{d\mathcal{L}}{dy} \\ &= \overline{\mathcal{L}}(y - t)\end{aligned}$$

$$\begin{aligned}\overline{z} &= \overline{y} \frac{dy}{dz} \\ &= \overline{y} \sigma'(z)\end{aligned}$$

$$\begin{aligned}\overline{w} &= \overline{z} \frac{\partial z}{\partial w} + \overline{\mathcal{R}} \frac{d\mathcal{R}}{dw} \\ &= \overline{z}x + \overline{\mathcal{R}}w\end{aligned}$$

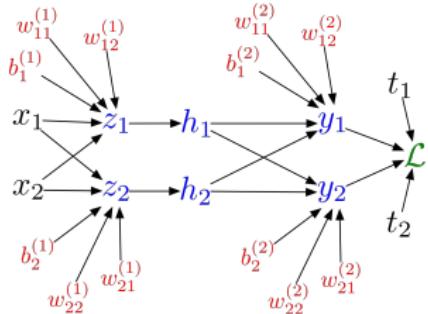
$$\begin{aligned}\overline{b} &= \overline{z} \frac{\partial z}{\partial b} \\ &= \overline{z}\end{aligned}$$

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Backpropagation

**Multilayer Perceptron** (multiple outputs):



**Forward pass:**

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma(z_i)$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)}$$

$$\mathcal{L} = \frac{1}{2} \sum_k (y_k - t_k)^2$$

**Backward pass:**

$$\overline{\mathcal{L}} = 1$$

$$\overline{y_k} = \overline{\mathcal{L}} (y_k - t_k)$$

$$\overline{w_{ki}^{(2)}} = \overline{y_k} h_i$$

$$\overline{b_k^{(2)}} = \overline{y_k}$$

$$\overline{h_i} = \sum_k \overline{y_k} w_{ki}^{(2)}$$

$$\overline{z_i} = \overline{h_i} \sigma'(z_i)$$

$$\overline{w_{ij}^{(1)}} = \overline{z_i} x_j$$

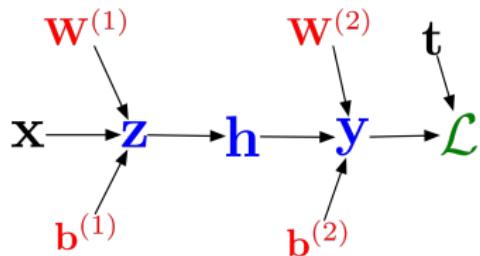
$$\overline{b_i^{(1)}} = \overline{z_i}$$

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Backpropagation

In vectorized form:



Backward pass:

$$\bar{\mathcal{L}} = 1$$

$$\bar{\mathbf{y}} = \bar{\mathcal{L}}(\mathbf{y} - \mathbf{t})$$

$$\bar{\mathbf{W}}^{(2)} = \bar{\mathbf{y}}\mathbf{h}^\top$$

$$\bar{\mathbf{b}}^{(2)} = \bar{\mathbf{y}}$$

$$\bar{\mathbf{h}} = \mathbf{W}^{(2)\top}\bar{\mathbf{y}}$$

$$\bar{\mathbf{z}} = \bar{\mathbf{h}} \circ \sigma'(\mathbf{z})$$

$$\bar{\mathbf{W}}^{(1)} = \bar{\mathbf{z}}\mathbf{x}^\top$$

$$\bar{\mathbf{b}}^{(1)} = \bar{\mathbf{z}}$$

Forward pass:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$

$$\mathbf{h} = \sigma(\mathbf{z})$$

$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$

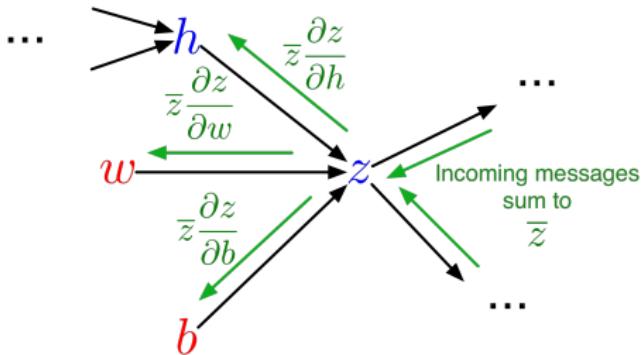
$$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2$$

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Backpropagation

**Backprop as message passing:**



- Each node receives a bunch of messages from its children, which it aggregates to get its error signal. It then passes messages to its parents.
- This provides modularity, since each node only has to know how to compute derivatives with respect to its arguments, and doesn't have to know anything about the rest of the graph.



A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



## Computational Cost

- Computational cost of forward pass: one **add-multiply operation** per weight

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

- Computational cost of backward pass: two add-multiply operations per weight

$$\overline{w_{ki}^{(2)}} = \overline{y_k} h_i$$

$$\overline{h_i} = \sum_k \overline{y_k} w_{ki}^{(2)}$$

- Rule of thumb: the backward pass is about as expensive as two forward passes.
- For a multilayer perceptron, this means the cost is linear in the number of layers, quadratic in the number of units per layer.

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Backpropagation

- Backprop is used to train the overwhelming majority of neural nets today.
  - Even optimization algorithms much fancier than gradient descent (e.g. second-order methods) use backprop to compute the gradients.
- Despite its practical success, backprop is believed to be neurally implausible.
  - No evidence for biological signals analogous to error derivatives.
  - All the biologically plausible alternatives we know about learn much more slowly (on computers).
  - So how on earth does the brain learn?

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation



# Content

2

## Multilayer perceptron (MLP)

- Multilayer perceptron
- Backpropagation
- Notation in vectorized form: Jacobian Matrix
- Capacity of multilayer perceptron



## Vector-Jacobian Products

- Previously, I suggested deriving backprop equations in terms of sums and indices, and then vectorizing them. But we'd like to implement our primitive operations in vectorized form.
- The **Jacobian** is the matrix of partial derivatives:

$$\mathbf{J} = \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \dots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \dots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}$$

- The backprop equation (single child node) can be written as a **vector-Jacobian product (VJP)**:

$$\bar{x}_j = \sum_i \bar{y}_i \frac{\partial y_i}{\partial x_j} \quad \bar{\mathbf{x}} = \bar{\mathbf{y}}^\top \mathbf{J}$$

- That gives a row vector. We can treat it as a column vector by taking

$$\bar{\mathbf{x}} = \mathbf{J}^\top \bar{\mathbf{y}}$$

A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 10: Automatic Differentiation



# Vector-Jacobian Products

## Examples

- Matrix-vector product

$$\mathbf{z} = \mathbf{W}\mathbf{x} \quad \mathbf{J} = \mathbf{W} \quad \bar{\mathbf{x}} = \mathbf{W}^\top \bar{\mathbf{z}}$$

- Elementwise operations

$$\mathbf{y} = \exp(\mathbf{z}) \quad \mathbf{J} = \begin{pmatrix} \exp(z_1) & & 0 \\ & \ddots & \\ 0 & & \exp(z_D) \end{pmatrix} \quad \bar{\mathbf{z}} = \exp(\mathbf{z}) \circ \bar{\mathbf{y}}$$

- Note: we never explicitly construct the Jacobian. It's usually simpler and more efficient to compute the VJP directly.



# Content

2

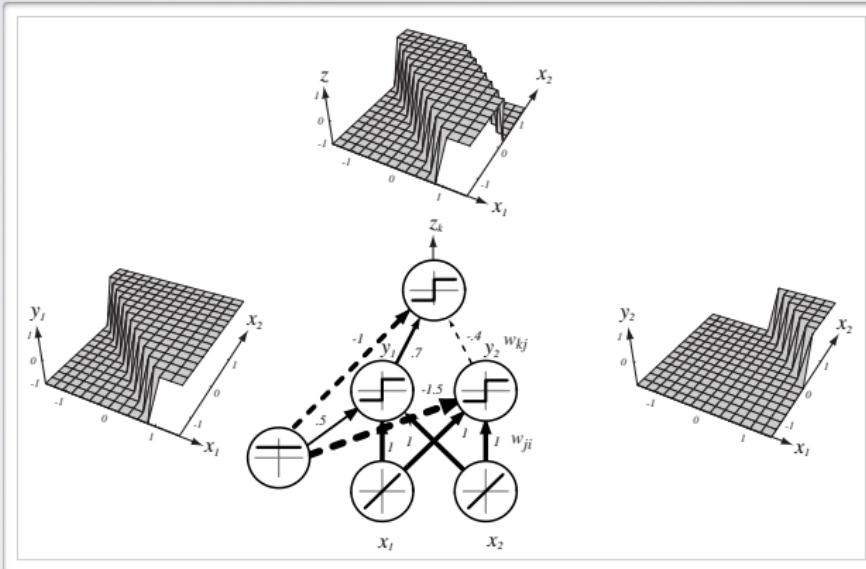
## Multilayer perceptron (MLP)

- Multilayer perceptron
- Backpropagation
- Notation in vectorized form: Jacobian Matrix
- Capacity of multilayer perceptron



# Capacity of multilayer perceptron

**Topics:** single hidden layer neural network



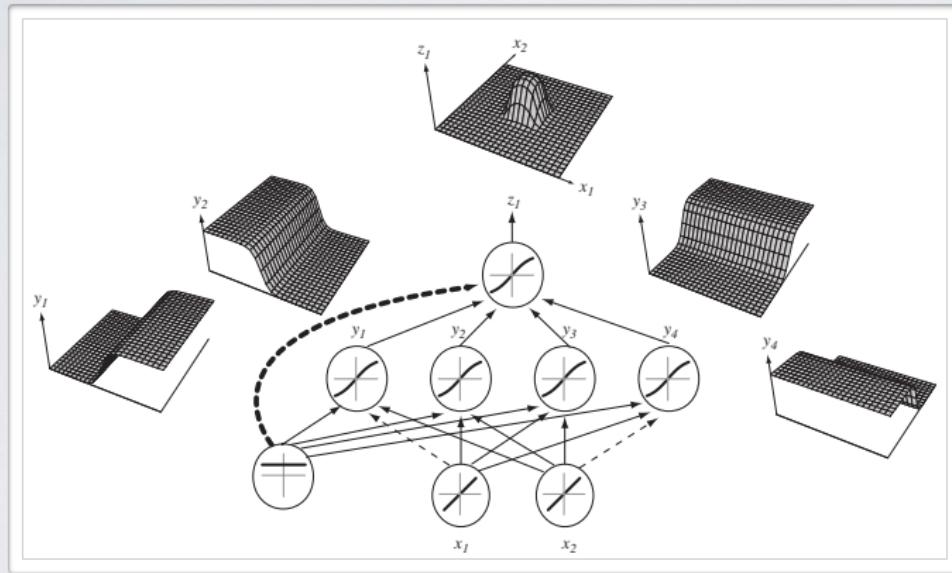
(from Pascal Vincent's slides)

A slide from: Hugo Larochelle, Neural Networks, at Deep Learning Summer School 2016



# Capacity of multilayer perceptron

**Topics:** single hidden layer neural network



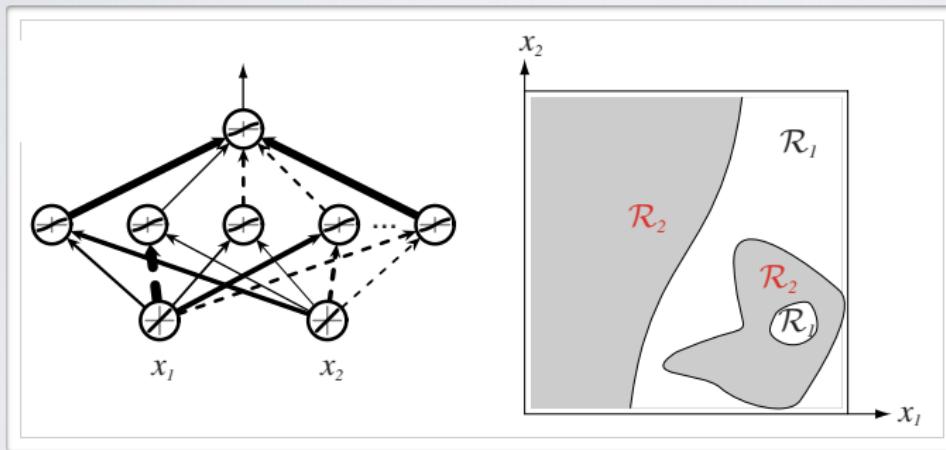
(from Pascal Vincent's slides)

A slide from: Hugo Larochelle, Neural Networks, at Deep Learning Summer School 2016



# Capacity of multilayer perceptron

**Topics:** single hidden layer neural network



(from Pascal Vincent's slides)

A slide from: Hugo Larochelle, Neural Networks, at Deep Learning Summer School 2016



# Capacity of multilayer perceptron

**Topics:** universal approximation

- Universal approximation theorem (Hornik, 1991):
  - ▶ “a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units”
- The result applies for sigmoid, tanh and many other hidden layer activation functions
- This is a good result, but it doesn’t mean there is a learning algorithm that can find the necessary parameter values!

A slide from: Hugo Larochelle, Neural Networks, at Deep Learning Summer School 2016



# Content

- 1 Artificial neural networks (ANNs)
- 2 Multilayer perceptron (MLP)
- 3 Convolutional neural networks (CNNs)
- 4 Convolutional Networks for Text Classification



# Content

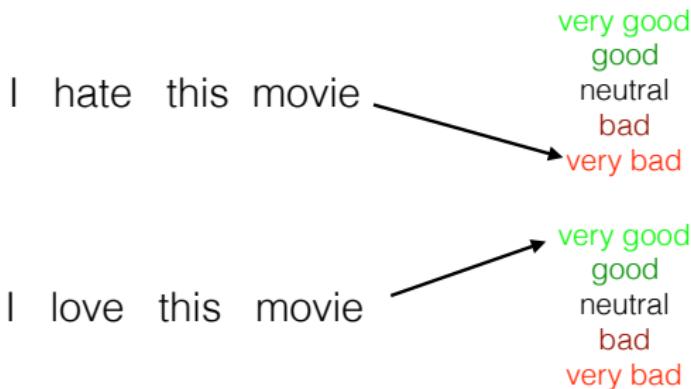
3

## Convolutional neural networks (CNNs)

- Bag of words
- Handling Combinations
- Convolutional Neural Networks
- Stacked Convolution and Dilated Convolution
- Non-linear Active Functions
- An example

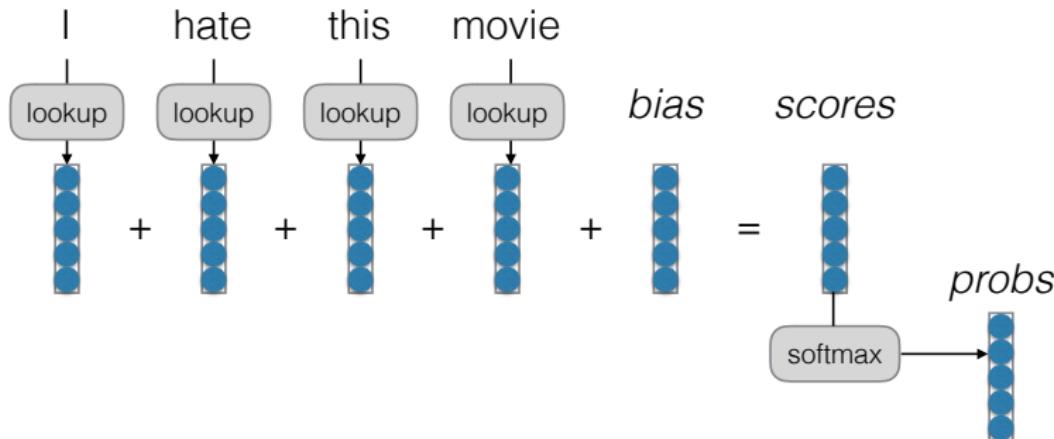


# An Example Prediction Problem: Sentence Classification





# A First Try: Bag of Words (BOW)



A slide from Graham Neubig's CS11-747 Neural Networks for NLP in CMU2019



# Build It, Break It

I don't love this movie

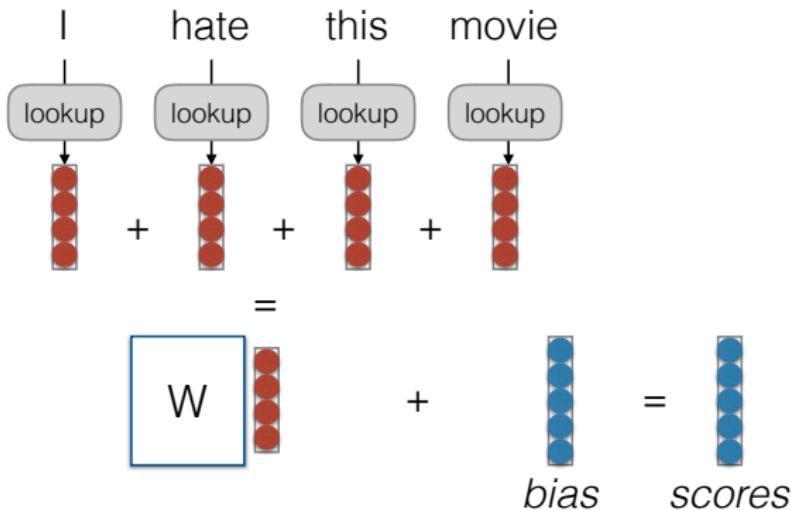
very good  
good  
neutral  
bad  
very bad

There's nothing I don't  
love about this movie

very good  
good  
neutral  
bad  
very bad



# Continuous Bag of Words (CBOW)



A slide from Graham Neubig's CS11-747 Neural Networks for NLP in CMU2019