

# Natural Language Processing

## Lecture 04: Artificial Neural Networks; CNN for Text Classification; NN Tips and Tricks

Valentin Malykh, MTS AI



Autumn 2024

The course is delivered at ITMO University, Saint-Petersburg,  
Bauman University, Moscow, IITU, Almaty

# Content

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

# Content

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

# Limitation of linear classifiers

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

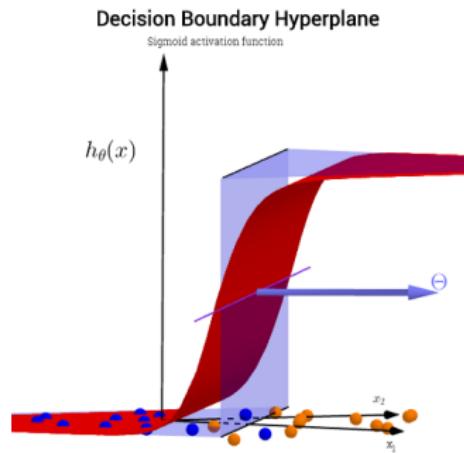


Figure source: <https://stats.stackexchange.com/questions/291492/>

# Limitation of linear classifiers

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

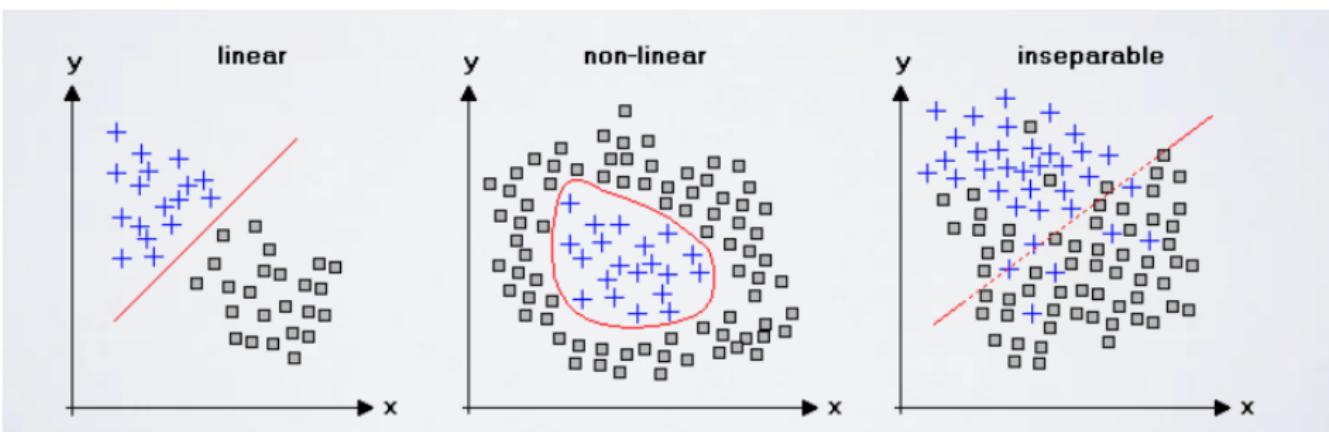


Figure source: <http://dingyancs.blogspot.com/2017/10/mle-linear-classification.html>

# Exclusive-or function

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

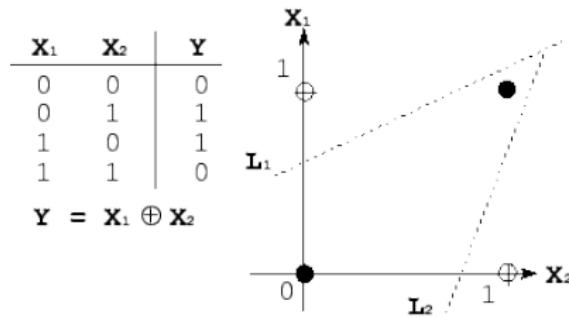
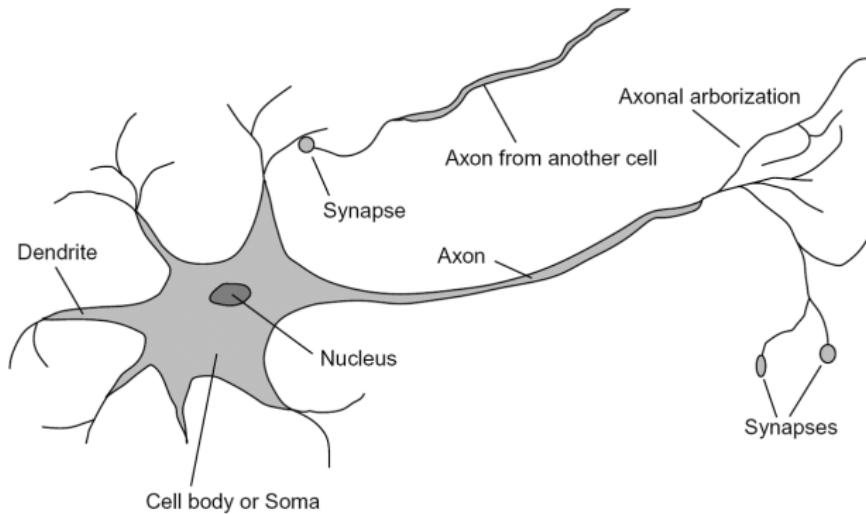


Figure source: <http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node19.html>

# Artificial Neural networks (ANNs)

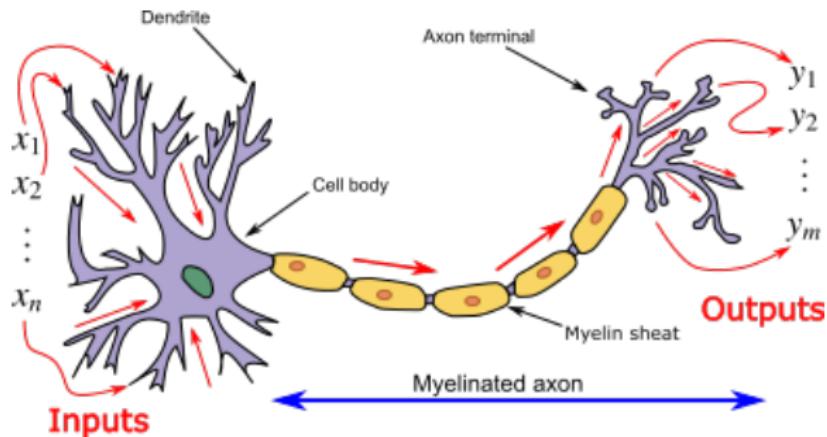
- Neural networks (NNs), or artificial neural networks (ANNs), provide an efficient way to solve non-linear-separable problem.
- Artificial neural networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. – Wikipedia
  - 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
- 5 Neural Networks Tips and Tricks

# Content

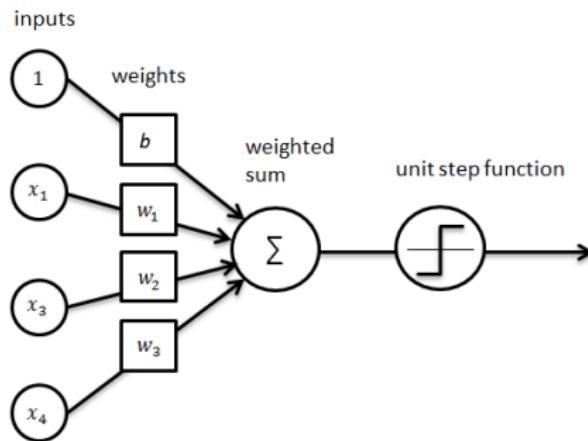
2

## Multilayer perceptron (MLP)

- Multilayer perceptron
- Backpropagation
- Capacity of multilayer perceptron

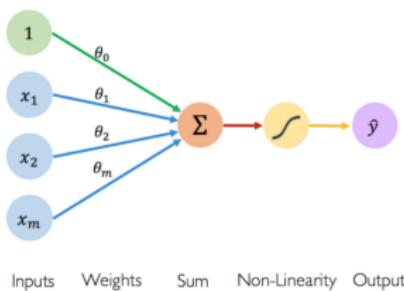
# Perceptron – simplest artificial neuron

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



# 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.

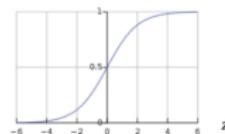


Activation Functions

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

- 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.

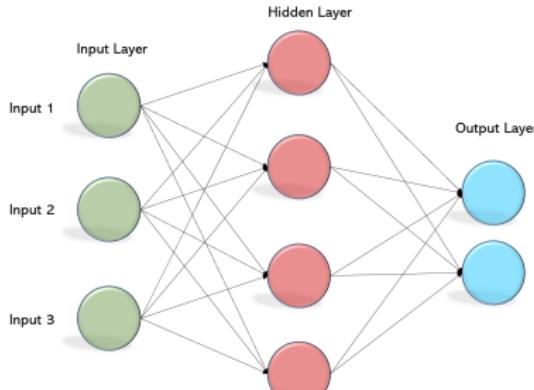


Figure source: <https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f>

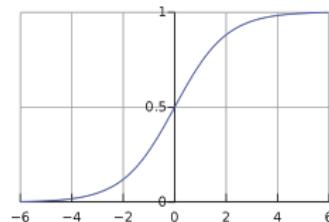
# Multilayer perceptrons

- 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.

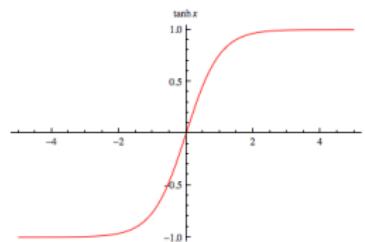
# Activation functions

Logistic:

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

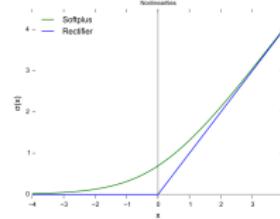


Hypbolic tangent:  $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



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.

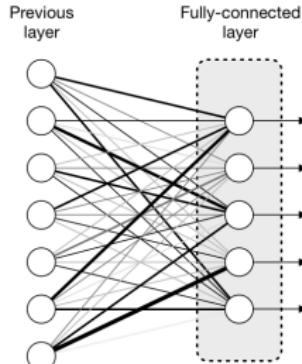


Figure source: <https://mc.ai/fully-connected-layer-with-dynamic-input-shape/>

# 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.

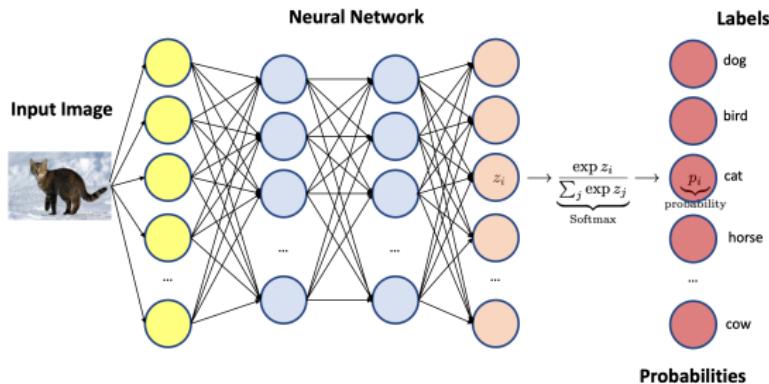


Figure source: <https://gab41.lab41.org/entropic-ghosts-35670292bc87>

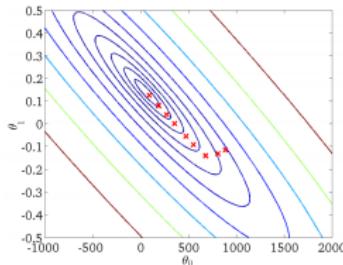
# Content

## 2 Multilayer perceptron (MLP)

- Multilayer perceptron
- **Backpropagation**
- 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:

$$z = wx + b$$

$$y = \sigma(z)$$

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

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

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

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

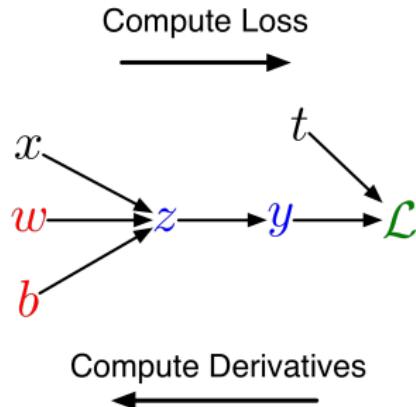
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{d\mathcal{L}}{dz}$$

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:

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

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

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

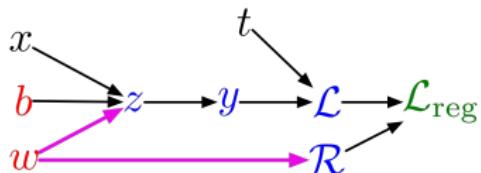
$$\bar{b} = \bar{z}$$

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



$$z = wx + b$$

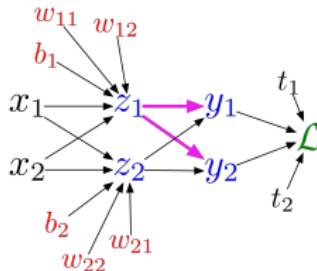
$$y = \sigma(z)$$

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

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

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

## Multiclass logistic regression



$$z_\ell = \sum_j w_{\ell j} x_j + b_\ell$$

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

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

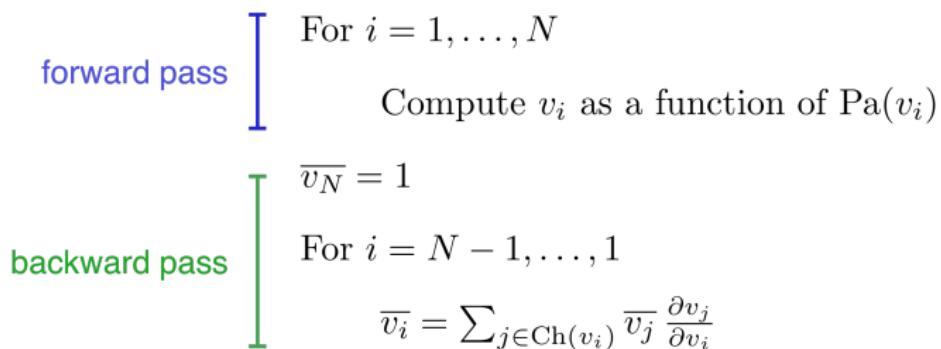
A slide from: Roger Grosse, Stanford University CSC321(2019), Lecture 6: Backpropagation

# Backpropagation

## Full backpropagation algorithm:

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

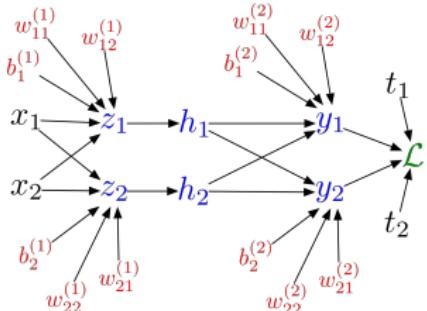
$v_N$  denotes the variable we're trying to compute derivatives of (e.g. loss).



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

# Backpropagation

**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:**

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

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

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

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

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

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

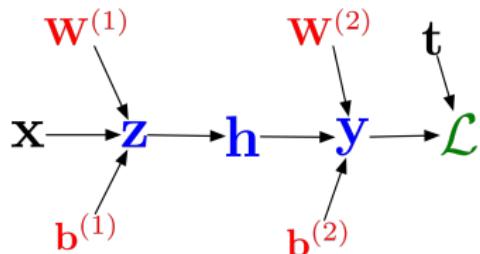
$$\overline{w_{ij}^{(1)}} = \overline{z_i} x_j$$

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

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:

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

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

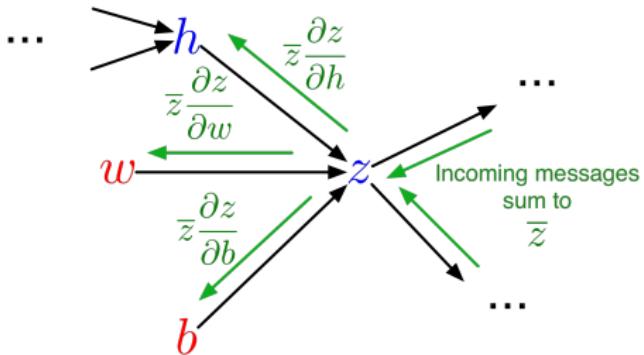
$$\mathbf{y} = \mathbf{W}^{(2)} \mathbf{h} + \mathbf{b}^{(2)}$$

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

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

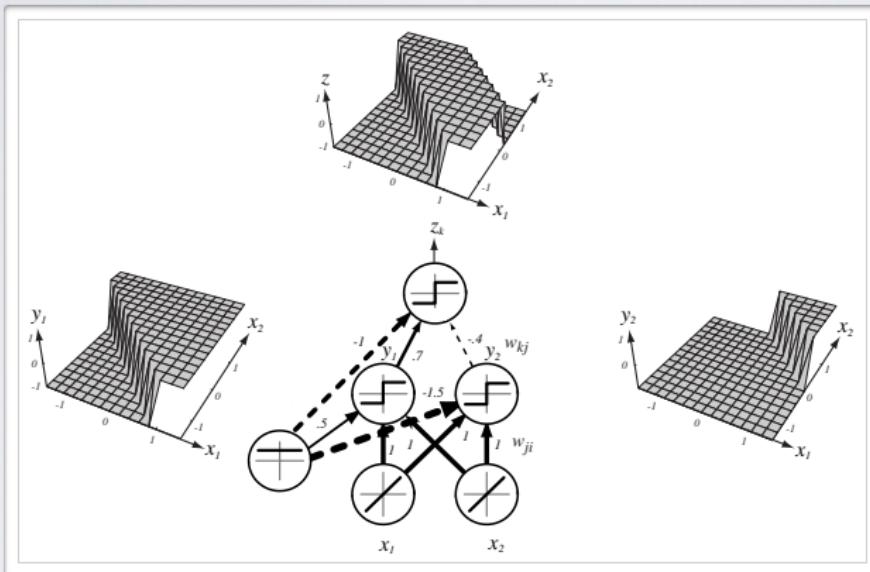
# Content

## 2 Multilayer perceptron (MLP)

- Multilayer perceptron
- Backpropagation
- Capacity of multilayer perceptron

# Capacity of multilayer perceptron

**Topics:** single hidden layer neural network

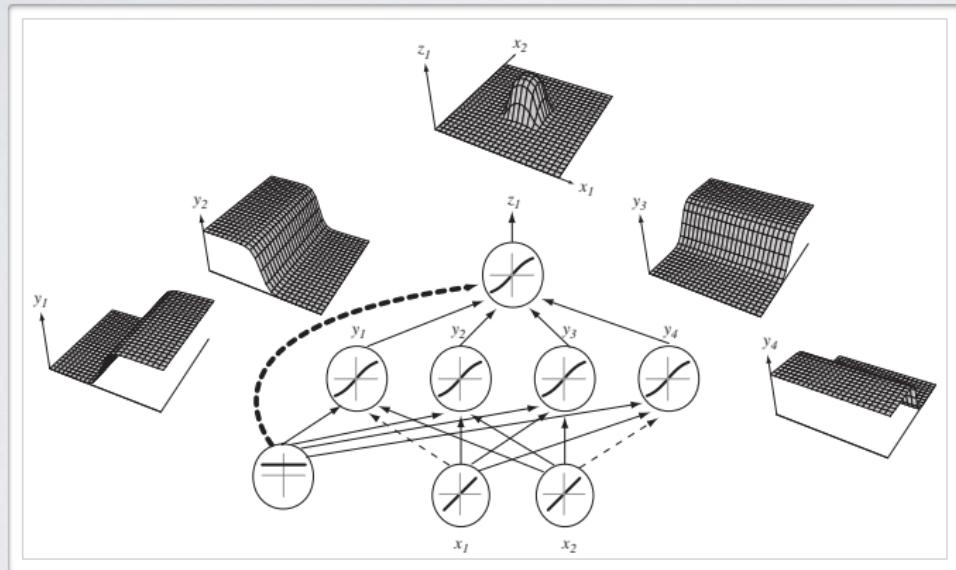


(from Pascal Vincent's slides)

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

# Capacity of multilayer perceptron

**Topics:** single hidden layer neural network

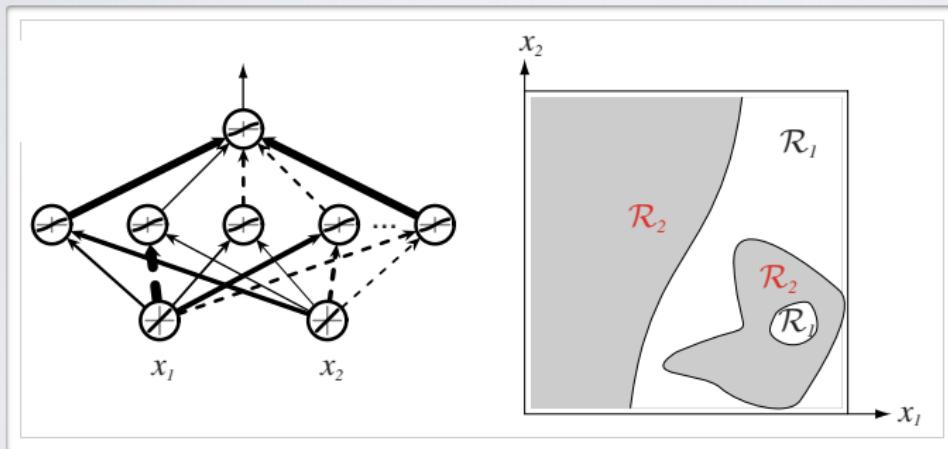


(from Pascal Vincent's slides)

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
- 5 Neural Networks Tips and Tricks

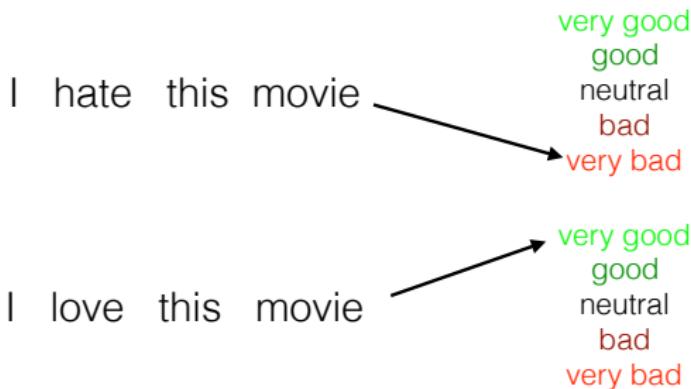
# Content

3

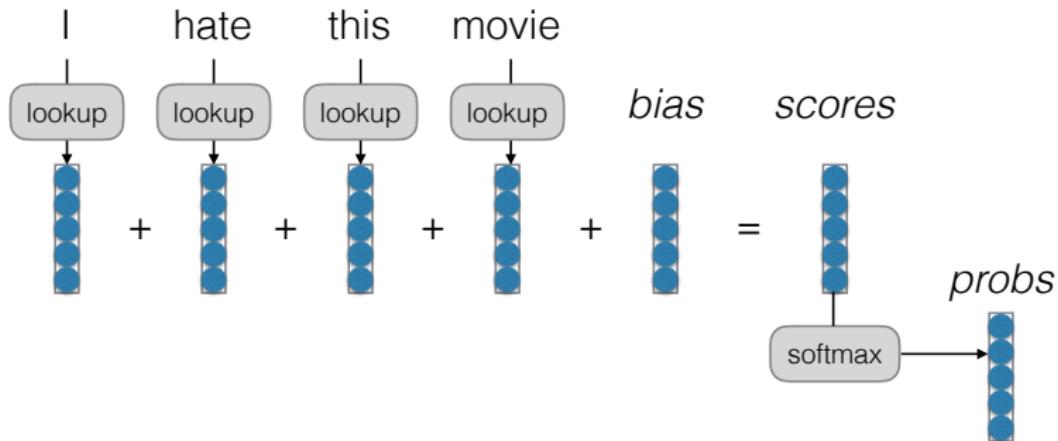
## Convolutional neural networks (CNNs)

- Bag of words
- Handling Combinations
- Convolutional Neural Networks
- 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

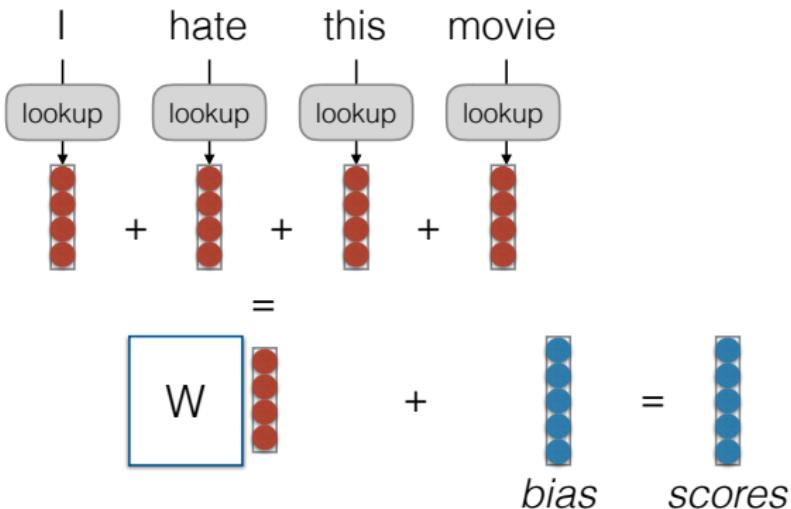
A diagram illustrating sentiment analysis. Two arrows point from movie reviews to separate vertical lists of sentiment scores. The top arrow points from "I don't love this movie" to a scale ranging from "very good" (green) at the top to "very bad" (red) at the bottom. The bottom arrow points from "There's nothing I don't love about this movie" to another identical scale.

- very good
- good
- neutral
- bad
- very bad

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

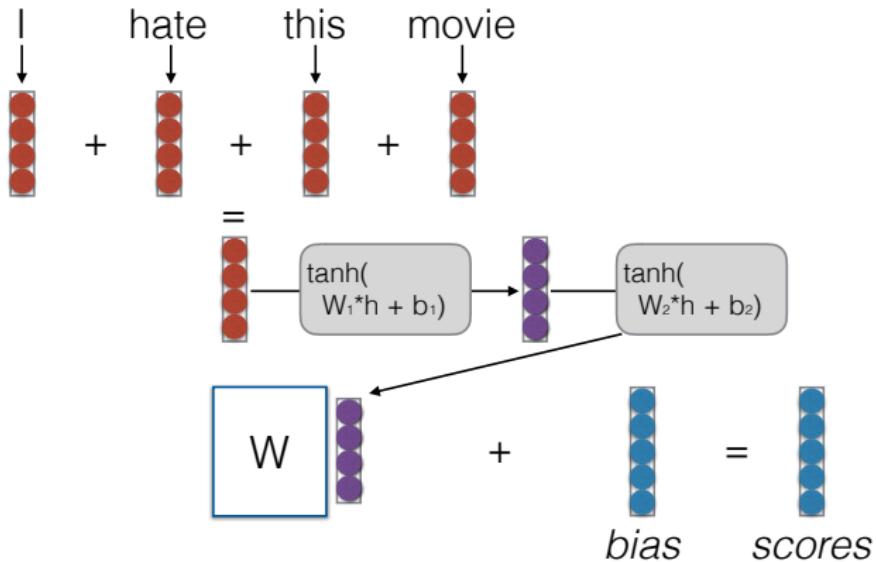
- very good
- good
- neutral
- bad
- very bad

# Continuous Bag of Words (CBOW)



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

# Deep CBOW



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

# What do Our Vectors Represent?

- We can learn feature combinations (a node in the second layer might be “feature 1 AND feature 5 are active”)
- e.g. capture things such as “not” AND “hate”
- BUT! Cannot handle “not hate”

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

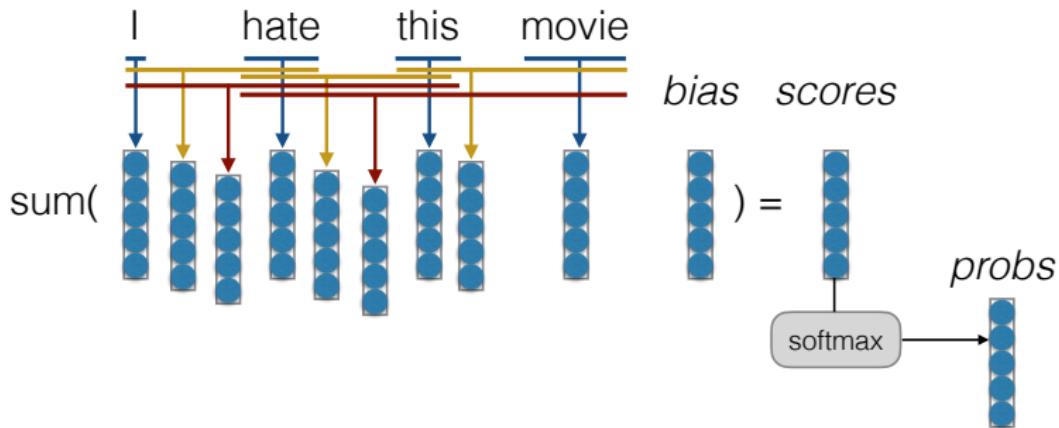
# Content

3

## Convolutional neural networks (CNNs)

- Bag of words
- **Handling Combinations**
- Convolutional Neural Networks
- Non-linear Active Functions
- An example

# Bag of n-grams



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

# Why Bag of n-grams?

- Allow us to capture combination features in a simple way “don’t love”, “not the best”
  - Works pretty well

François Chollet · @fchollet · 2 Nov 2016

We are releasing an open dataset for theorem proving, HolStep:  
[openreview.net/forum?id=rwuxY](http://openreview.net/forum?id=rwuxY) - can you beat our 83% accuracy baseline?

1

11

8

123

6

Hal Daumé III @haldaume3 : 2 Nov 2016

[@fchollet](#) sure, I'll play. 85%, took me about an hour. (totally possible I did something wrong in preprocessing though!)

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

# What Problems w/ Bag of n-grams?

- Same as before: parameter explosion
- No sharing between similar words/n-grams

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

# Content

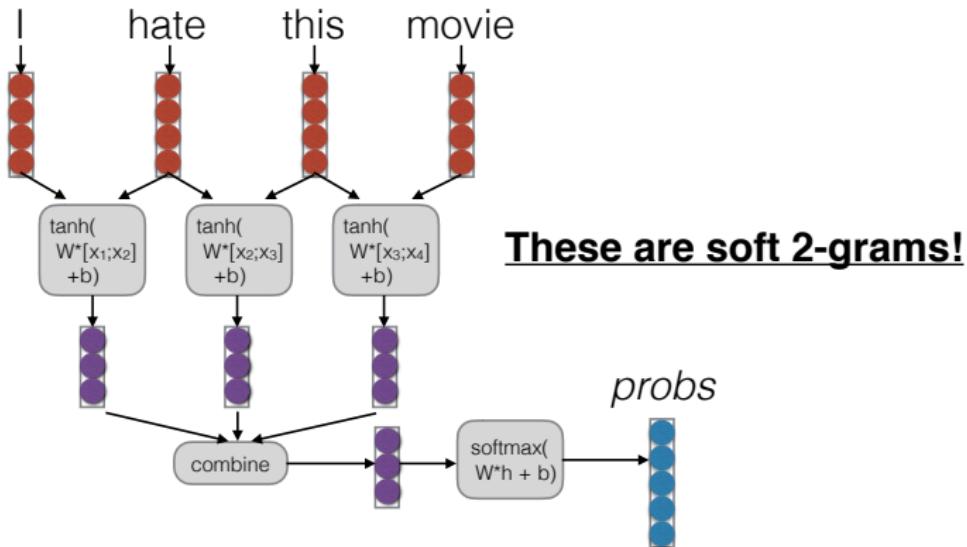
3

## Convolutional neural networks (CNNs)

- Bag of words
- Handling Combinations
- **Convolutional Neural Networks**
- Non-linear Active Functions
- An example

# 1-dimensional Convolutions / Time-delay Networks

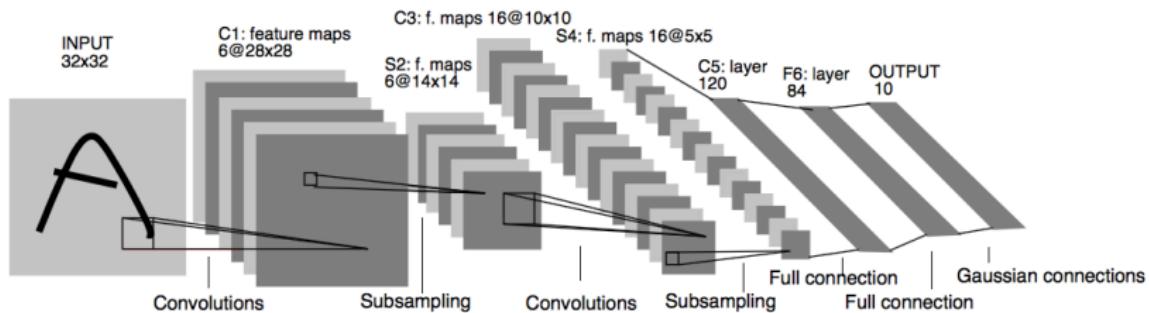
(Waibel et al. 1989)



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

# 2-dimensional Convolutional Networks

(LeCun et al. 1997)



Parameter extraction performs a 2D sweep, not 1D

# CNNs for Text

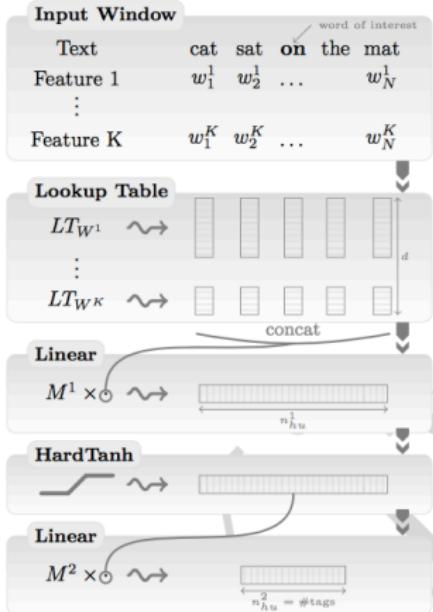
## (Collobert and Weston 2011)

- Generally based on 1D convolutions
  - But often uses terminology/functions borrowed from image processing for historical reasons
- Two main paradigms:
  - **Context window modeling:** For tagging, etc. get the surrounding context before tagging
  - **Sentence modeling:** Do convolution to extract n-grams, pooling to combine over whole sentence

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

# CNNs for Tagging

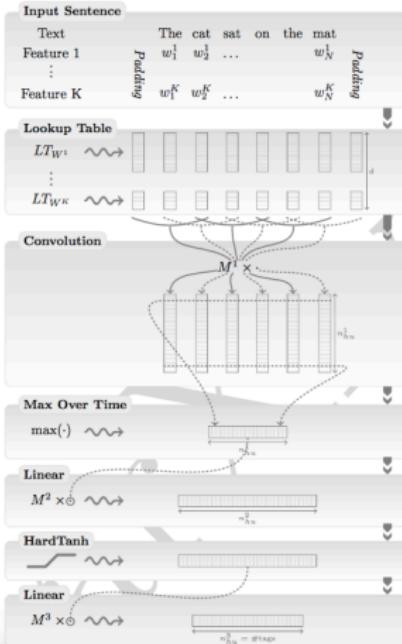
(Collobert and Weston 2011)



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

# CNNs for Sentence Modeling

(Collobert and Weston 2011)



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

# Standard conv2d Function

- 2D convolution function takes input + parameters
- **Input:** 3D tensor
  - rows (e.g. words), columns, features (“channels”)
- **Parameters/Filters:** 4D tensor
  - rows, columns, input features, output features

# Padding

- After convolution, the rows and columns of the output tensor are either
  - = to rows/columns of input tensor (“*same*” convolution)
  - = to rows/columns of input tensor minus the size of the filter plus one (“*valid*” or “*narrow*”)
  - = to rows/columns of input tensor plus filter minus one (“*wide*”)

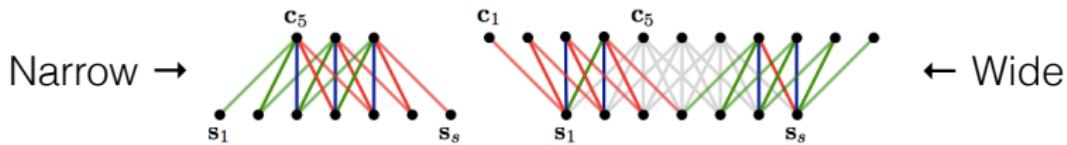
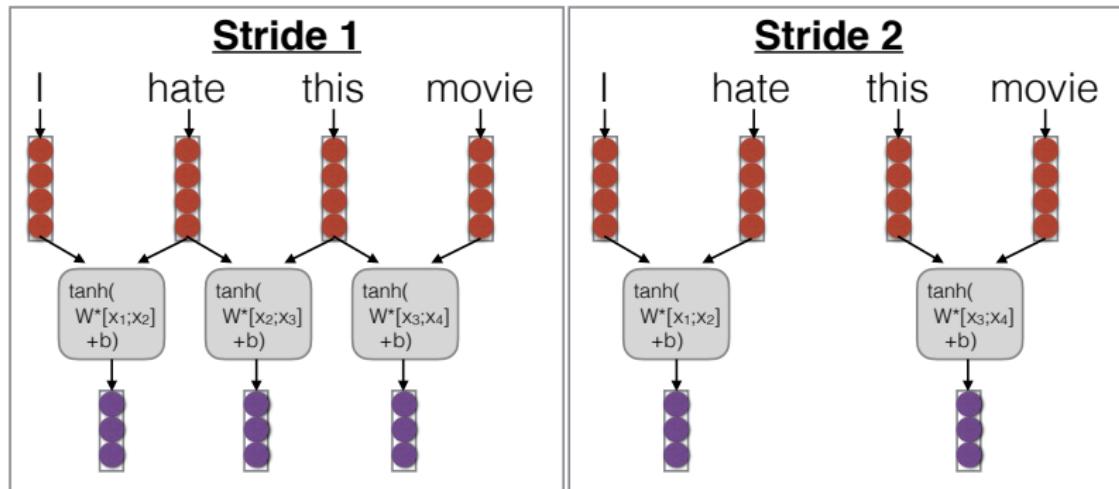


Image: Kalchbrenner et al. 2014

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

# Striding

- Skip some of the outputs to reduce length of extracted feature vector



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

# Pooling

- Pooling is like convolution, but calculates some reduction function feature-wise
- **Max pooling:** “Did you see this feature anywhere in the range?” (most common)
- **Average pooling:** “How prevalent is this feature over the entire range”
- **k-Max pooling:** “Did you see this feature up to k times?”
- **Dynamic pooling:** “Did you see this feature in the beginning? In the middle? In the end?”

# Stacked Convolution

- Feeding in convolution from previous layer results in larger area of focus for each feature

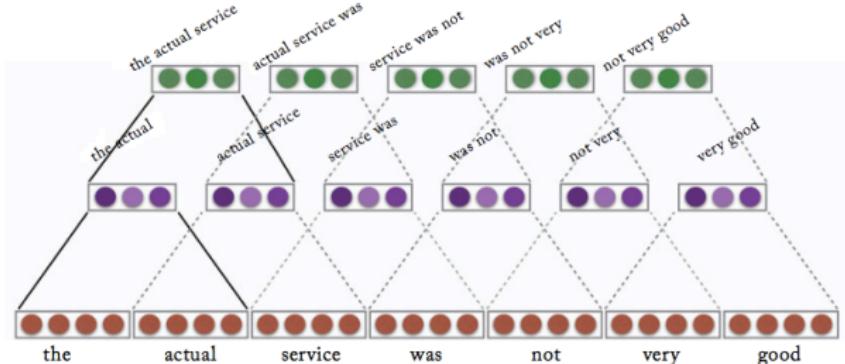


Image Credit: Goldberg Book

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

# Content

3

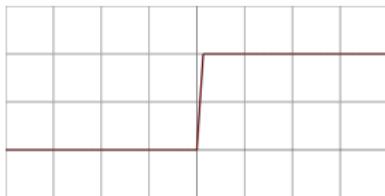
## Convolutional neural networks (CNNs)

- Bag of words
- Handling Combinations
- Convolutional Neural Networks
- Non-linear Active Functions
- An example

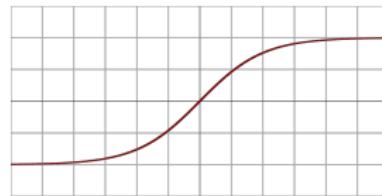
# Non-linear Functions

- Proper choice of a non-linear function is essential in stacked networks

step



tanh

rectifier  
(ReLU)

soft plus



- Functions such as ReLU or softplus allegedly better at preserving gradients

Image: Wikipedia

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

# Which Non-linearity Should I Use?

- Ultimately an empirical question
- Many new functions proposed, but search by Eger et al. (2018) over NLP tasks found that standard functions such as tanh and relu quite robust

sigmoid	$f(x) = \sigma(x) = 1/(1 + \exp(-x))$
swish	$f(x) = x \cdot \sigma(x)$
maxsig	$f(x) = \max\{x, \sigma(x)\}$
cosid	$f(x) = \cos(x) - x$
minsin	$f(x) = \min\{x, \sin(x)\}$
arctid	$f(x) = \arctan(x)^2 - x$
maxtanh	$f(x) = \max\{x, \tanh(x)\}$
lrelu-0.01	$f(x) = \max\{x, 0.01x\}$
lrelu-0.30	$f(x) = \max\{x, 0.3x\}$
penalized tanh	$f(x) = \begin{cases} \tanh(x) & x > 0, \\ 0.25 \tanh(x) & x \leq 0 \end{cases}$
best	penalized tanh (6), swish (6), elu (4), relu (4), lrelu-0.01 (4)
mean	penalized tanh (16), tanh (13), sin (10)

Table 5: Top-3 winner statistics. In brackets: number of times within top-3, keeping only functions with four or more top-3 rankings.

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

# Content

3

## Convolutional neural networks (CNNs)

- Bag of words
- Handling Combinations
- Convolutional Neural Networks
- Non-linear Active Functions
- An example



## A 1D convolution for text

tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3

t,d,r	-1.0
d,r,t	-0.5
r,t,k	-3.6
t,k,g	-0.2
k,g,o	0.3

Apply a **filter (or kernel)** of size 3

3	1	2	-3
-1	2	1	-3
1	1	-1	1

10

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n



## 1D convolution for text with padding

$\emptyset$	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
$\emptyset$	0.0	0.0	0.0	0.0

$\emptyset, t, d$	-0.6
$t, d, r$	-1.0
$d, r, t$	-0.5
$r, t, k$	-3.6
$t, k, g$	-0.2
$k, g, o$	0.3
$g, o, \emptyset$	-0.5

Apply a **filter (or kernel)** of size 3

11	3	1	2	-3
	-1	2	1	-3
	1	1	-1	1

Christopher Manning, Natural Language Processing with Deep Learning, Stanford U. CS224n



## 3 channel 1D convolution with padding = 1

$\emptyset$	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
$\emptyset$	0.0	0.0	0.0	0.0

$\emptyset, t, d$	-0.6	0.2	1.4
$t, d, r$	-1.0	1.6	-1.0
$d, r, t$	-0.5	-0.1	0.8
$r, t, k$	-3.6	0.3	0.3
$t, k, g$	-0.2	0.1	1.2
$k, g, o$	0.3	0.6	0.9
$g, o, \emptyset$	-0.5	-0.9	0.1

Apply 3 filters of size 3

3	1	2	-3	1	0	0	1	1	-1	2	-1
-1	2	1	-3	1	0	-1	-1	1	0	-1	3
1	1	-1	1	0	1	0	1	0	2	2	1
--											

Could also use (zero) padding = 2  
Also called “wide convolution”



## conv1d, padded with max pooling over time

<b>Ø</b>	<b>0.0</b>	<b>0.0</b>	<b>0.0</b>	<b>0.0</b>
<b>tentative</b>	0.2	0.1	-0.3	0.4
<b>deal</b>	0.5	0.2	-0.3	-0.1
<b>reached</b>	-0.1	-0.3	-0.2	0.4
<b>to</b>	0.3	-0.3	0.1	0.1
<b>keep</b>	0.2	-0.3	0.4	0.2
<b>government</b>	0.1	0.2	-0.1	-0.1
<b>open</b>	-0.4	-0.4	0.2	0.3
<b>Ø</b>	<b>0.0</b>	<b>0.0</b>	<b>0.0</b>	<b>0.0</b>

<b>Ø,t,d</b>	-0.6	0.2	1.4
<b>t,d,r</b>	-1.0	1.6	-1.0
<b>d,r,t</b>	-0.5	-0.1	0.8
<b>r,t,k</b>	-3.6	0.3	0.3
<b>t,k,g</b>	-0.2	0.1	1.2
<b>k,g,o</b>	0.3	0.6	0.9
<b>g,o,Ø</b>	-0.5	-0.9	0.1
<b>max p</b>	0.3	1.6	1.4

Apply 3 filters of size 3

	3	1	2	-3		1	0	0	1		1	-1	2	-1
	-1	2	1	-3		1	0	-1	-1		1	0	-1	3
13	1	1	-1	1		0	1	0	1		0	2	2	1

Christopher Manning, Natural Language Processing with Deep Learning, Stanford U. CS224n



## conv1d, padded with ave pooling over time

<b>Ø</b>	<b>0.0</b>	<b>0.0</b>	<b>0.0</b>	<b>0.0</b>
<b>tentative</b>	0.2	0.1	-0.3	0.4
<b>deal</b>	0.5	0.2	-0.3	-0.1
<b>reached</b>	-0.1	-0.3	-0.2	0.4
<b>to</b>	0.3	-0.3	0.1	0.1
<b>keep</b>	0.2	-0.3	0.4	0.2
<b>government</b>	0.1	0.2	-0.1	-0.1
<b>open</b>	-0.4	-0.4	0.2	0.3
<b>Ø</b>	<b>0.0</b>	<b>0.0</b>	<b>0.0</b>	<b>0.0</b>

<b>Ø,t,d</b>	-0.6	0.2	1.4
<b>t,d,r</b>	-1.0	1.6	-1.0
<b>d,r,t</b>	-0.5	-0.1	0.8
<b>r,t,k</b>	-3.6	0.3	0.3
<b>t,k,g</b>	-0.2	0.1	1.2
<b>k,g,o</b>	0.3	0.6	0.9
<b>g,o,Ø</b>	-0.5	-0.9	0.1
<b>ave p</b>	-0.87	0.26	0.53

Apply 3 filters of size 3

	3	1	2	-3		1	0	0	1		1	-1	2	-1
	-1	2	1	-3		1	0	-1	-1		1	0	-1	3
14	1	1	-1	1		0	1	0	1		0	2	2	1

Christopher Manning, Natural Language Processing with Deep Learning, Stanford U. CS224n



## Other less useful notions: stride = 2

$\emptyset$	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
$\emptyset$	0.0	0.0	0.0	0.0

$\emptyset, t, d$	-0.6	0.2	1.4
$d, r, t$	-0.5	-0.1	0.8
$t, k, g$	-0.2	0.1	1.2
$g, o, \emptyset$	-0.5	-0.9	0.1

Apply 3 filters of size 3

	3	1	2	-3		1	0	0	1		1	-1	2	-1
	-1	2	1	-3		1	0	-1	-1		1	0	-1	3
16	1	1	-1	1		0	1	0	1		0	2	2	1

Christopher Manning, Natural Language Processing with Deep Learning, Stanford U. CS224n



## Less useful: local max pool, stride = 2

$\emptyset$	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
$\emptyset$	0.0	0.0	0.0	0.0

$\emptyset, t, d$	-0.6	0.2	1.4
$t, d, r$	-1.0	1.6	-1.0
$d, r, t$	-0.5	-0.1	0.8
$r, t, k$	-3.6	0.3	0.3
$t, k, g$	-0.2	0.1	1.2
$k, g, o$	0.3	0.6	0.9
$g, o, \emptyset$	-0.5	-0.9	0.1
$\emptyset$	-Inf	-Inf	-Inf

Apply 3 filters of size 3

3	1	2	-3	1	0	0	1	1	-1	2	-1
-1	2	1	-3	1	0	-1	-1	1	0	-1	3
1	1	-1	1	0	1	0	1	0	2	2	1

$\emptyset, t, d, r$	-0.6	1.6	1.4
$d, r, t, k$	-0.5	0.3	0.8
$t, k, g, o$	0.3	0.6	1.2
$g, o, \emptyset, \emptyset$	-0.5	-0.9	0.1



## conv1d, k-max pooling over time, $k = 2$

$\emptyset$	0.0	0.0	0.0	0.0
tentative	0.2	0.1	-0.3	0.4
deal	0.5	0.2	-0.3	-0.1
reached	-0.1	-0.3	-0.2	0.4
to	0.3	-0.3	0.1	0.1
keep	0.2	-0.3	0.4	0.2
government	0.1	0.2	-0.1	-0.1
open	-0.4	-0.4	0.2	0.3
$\emptyset$	0.0	0.0	0.0	0.0

$\emptyset, t, d$	-0.6	0.2	1.4
$t, d, r$	-1.0	1.6	-1.0
$d, r, t$	-0.5	-0.1	0.8
$r, t, k$	-3.6	0.3	0.3
$t, k, g$	-0.2	0.1	1.2
$k, g, o$	0.3	0.6	0.9
$g, o, \emptyset$	-0.5	-0.9	0.1
<b>2-max p</b>	-0.2	1.6	1.4
	0.3	0.6	1.2

Apply 3 filters of size 3

	3	1	2	-3	1	0	0	1	1	-1	2	-1
	-1	2	1	-3	1	0	-1	-1	1	0	-1	3
18	1	1	-1	1	0	1	0	1	0	2	2	1

Christopher Manning, Natural Language Processing with Deep Learning, Stanford U. CS224n

# Content

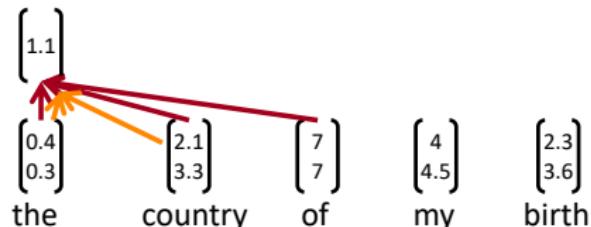
- 1 Artificial neural networks (ANNs)
- 2 Multilayer perceptron (MLP)
- 3 Convolutional neural networks (CNNs)
- 4 Convolutional Networks for Text Classification
- 5 Neural Networks Tips and Tricks

### 3. Single Layer CNN for Sentence Classification

- Yoon Kim (2014): Convolutional Neural Networks for Sentence Classification. EMNLP 2014. <https://arxiv.org/pdf/1408.5882.pdf>  
Code: <https://arxiv.org/pdf/1408.5882.pdf> [Theano!, etc.]
- A variant of convolutional NNs of Collobert, Weston et al. (2011)
- Goal: Sentence classification:
  - Mainly positive or negative sentiment of a sentence
  - Other tasks like:
    - Subjective or objective language sentence
    - Question classification: about person, location, number, ...

# Single Layer CNN for Sentence Classification

- A simple use of one convolutional layer and **pooling**
- Word vectors:  $\mathbf{x}_i \in \mathbb{R}^k$
- Sentence:  $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \dots \oplus \mathbf{x}_n$  (vectors concatenated)
- Concatenation of words in range:  $\mathbf{x}_{i:i+j}$  (symmetric more common)
- Convolutional filter:  $\mathbf{w} \in \mathbb{R}^{hk}$  (over window of  $h$  words)
- Note, filter is a vector!
- Filter could be of size 2, 3, or 4:



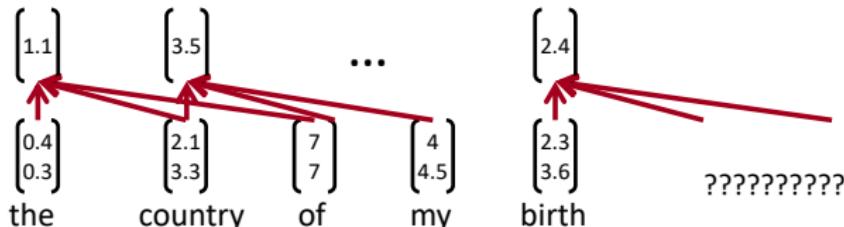
Christopher Manning, Natural Language Processing with Deep Learning, Stanford U. CS224n

## Single layer CNN

- Filter  $\mathbf{w}$  is applied to all possible windows (concatenated vectors)
- To compute feature (one *channel*) for CNN layer:

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$

- Sentence:  $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \dots \oplus \mathbf{x}_n$
- All possible windows of length  $h$ :  $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \dots, \mathbf{x}_{n-h+1:n}\}$
- Result is a feature map:  $\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



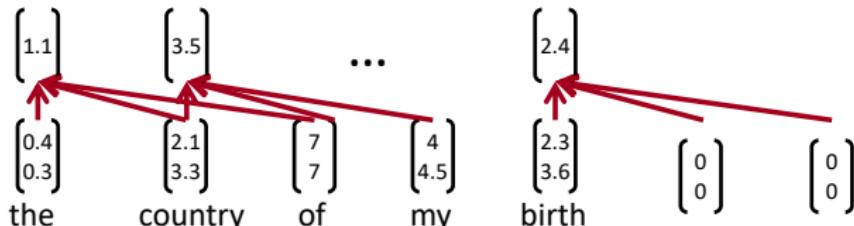
Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

## Single layer CNN

- Filter  $\mathbf{w}$  is applied to all possible windows (concatenated vectors)
- To compute feature (one *channel*) for CNN layer:

$$c_i = f(\mathbf{w}^T \mathbf{x}_{i:i+h-1} + b)$$

- Sentence:  $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus \mathbf{x}_2 \oplus \dots \oplus \mathbf{x}_n$
- All possible windows of length  $h$ :  $\{\mathbf{x}_{1:h}, \mathbf{x}_{2:h+1}, \dots, \mathbf{x}_{n-h+1:n}\}$
- Result is a feature map:  $\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$



Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

## Pooling and channels

- Pooling: max-over-time pooling layer
- Idea: capture most important activation (maximum over time)
- From feature map  $\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
- Pooled single number:  $\hat{c} = \max\{\mathbf{c}\}$
- Use multiple filter weights  $\mathbf{w}$
- Useful to have different window sizes  $h$
- Because of max pooling  $\hat{c} = \max\{\mathbf{c}\}$ , length of  $\mathbf{c}$  irrelevant  
$$\mathbf{c} = [c_1, c_2, \dots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$
- So we could have some filters that look at unigrams, bigrams, tri-grams, 4-grams, etc.

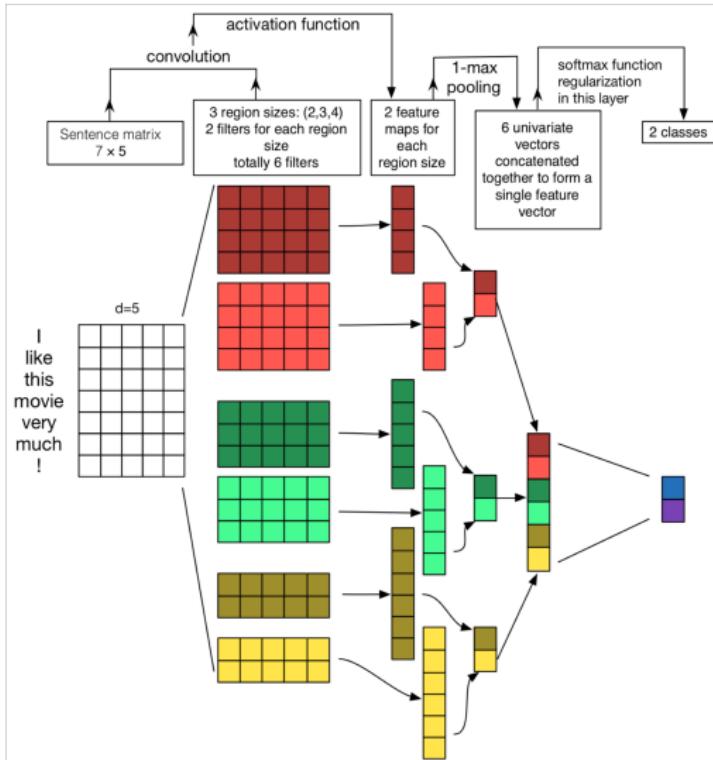
## Classification after one CNN layer

- First one convolution, followed by one max-pooling
- To obtain final feature vector:  $\mathbf{z} = [\hat{c}_1, \dots, \hat{c}_m]$  (assuming  $m$  filters  $\mathbf{w}$ )
  - Used 100 feature maps each of sizes 3, 4, 5
- Simple final softmax layer  $y = \text{softmax}\left(W^{(S)}\mathbf{z} + b\right)$

From:

Zhang and Wallace  
 (2015) A Sensitivity  
 Analysis of (and  
 Practitioners' Guide  
 to) Convolutional  
 Neural Networks for  
 Sentence  
 Classification

<https://arxiv.org/pdf/1510.03820.pdf>  
 (follow on paper, not famous, but a nice picture)



# Experiments

Model	MR	SST-1	SST-2	Subj	TREC	CR	MPQA
CNN-rand	76.1	45.0	82.7	89.6	91.2	79.8	83.4
CNN-static	81.0	45.5	86.8	93.0	92.8	84.7	<b>89.6</b>
CNN-non-static	<b>81.5</b>	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	<b>88.1</b>	93.2	92.2	<b>85.0</b>	89.4
RAE (Socher et al., 2011)	77.7	43.2	82.4	—	—	—	86.4
MV-RNN (Socher et al., 2012)	79.0	44.4	82.9	—	—	—	—
RNTN (Socher et al., 2013)	—	45.7	85.4	—	—	—	—
DCNN (Kalchbrenner et al., 2014)	—	48.5	86.8	—	93.0	—	—
Paragraph-Vec (Le and Mikolov, 2014)	—	<b>48.7</b>	87.8	—	—	—	—
CCAE (Hermann and Blunsom, 2013)	77.8	—	—	—	—	—	87.2
Sent-Parser (Dong et al., 2014)	79.5	—	—	—	—	—	86.3
NBSVM (Wang and Manning, 2012)	79.4	—	—	93.2	—	81.8	86.3
MNB (Wang and Manning, 2012)	79.0	—	—	<b>93.6</b>	—	80.0	86.3
G-Dropout (Wang and Manning, 2013)	79.0	—	—	93.4	—	82.1	86.1
F-Dropout (Wang and Manning, 2013)	79.1	—	—	<b>93.6</b>	—	81.9	86.3
Tree-CRF (Nakagawa et al., 2010)	77.3	—	—	—	—	81.4	86.1
CRF-PR (Yang and Cardie, 2014)	—	—	—	—	—	82.7	—
SVM <sub>S</sub> (Silva et al., 2011)	—	—	—	—	<b>95.0</b>	—	—

Christopher Manning, Natural Language Processing with Deep Learning, Standford U. CS224n

# Content

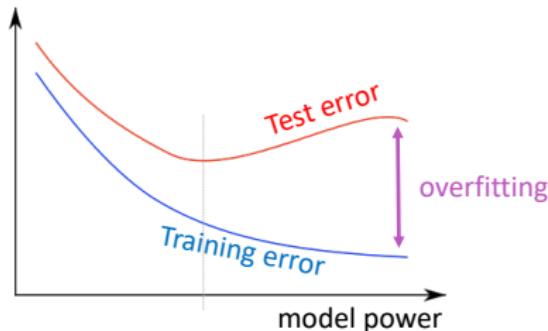
- 1 Artificial neural networks (ANNs)
- 2 Multilayer perceptron (MLP)
- 3 Convolutional neural networks (CNNs)
- 4 Convolutional Networks for Text Classification
- 5 Neural Networks Tips and Tricks

# We have models with many params! Regularization!

- Really a full loss function in practice includes **regularization** over all parameters  $\theta$ , e.g., L2 regularization:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^N -\log \left( \frac{e^{f_{y_i}}}{\sum_{c=1}^C e^{f_c}} \right) + \lambda \sum_k \theta_k^2$$

- Regularization works to prevent **overfitting** when we have a lot of features (or later a very powerful/deep model, ++)

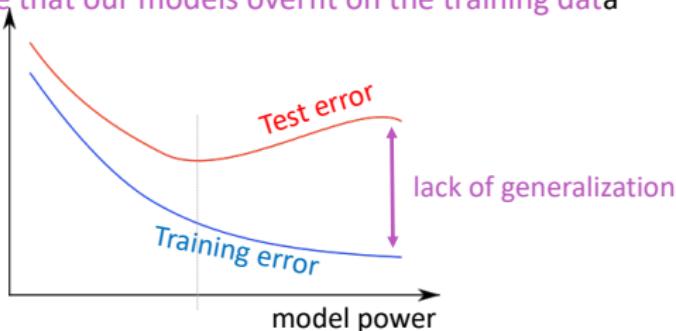


# We have models with many params! Regularization!

- Really a full loss function in practice includes **regularization** over all parameters  $\theta$ , e.g., L2 regularization:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^N -\log \left( \frac{e^{f_{y_i}}}{\sum_{c=1}^C e^{f_c}} \right) + \lambda \sum_k \theta_k^2$$

- Regularization produces models that generalize well when we have a lot of features (or later a very powerful/deep model, ++)
  - We do not care that our models overfit on the training data



# Dropout

(Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov 2012/JMLR 2014)

Preventing Feature Co-adaptation = Regularization

- Training time: at each instance of evaluation (in online SGD-training), randomly set 50% of the inputs to each neuron to 0
- Test time: halve the model weights (now twice as many)
- This prevents feature co-adaptation: A feature cannot only be useful in the presence of particular other features
- In a single layer: A kind of middle-ground between Naïve Bayes (where all feature weights are set independently) and logistic regression models (where weights are set in the context of all others)
- Can be thought of as a form of model bagging
- Nowadays usually thought of as strong, feature-dependent regularizer [Wager, Wang, & Liang 2013]

## “Vectorization”

- E.g., looping over word vectors versus concatenating them all into one large matrix and then multiplying the softmax weights with that matrix

```
from numpy import random
N = 500 # number of windows to classify
d = 300 # dimensionality of each window
C = 5 # number of classes
W = random.rand(C,d)
wordvectors_list = [random.rand(d,1) for i in range(N)]
wordvectors_one_matrix = random.rand(d,N)

%timeit [W.dot(wordvectors_list[i]) for i in range(N)]
%timeit W.dot(wordvectors_one_matrix)
```

- 1000 loops, best of 3: **639 µs** per loop  
10000 loops, best of 3: **53.8 µs** per loop

# “Vectorization”

```
from numpy import random
N = 500 # number of windows to classify
d = 300 # dimensionality of each window
C = 5 # number of classes
W = random.rand(C,d)
wordvectors_list = [random.rand(d,1) for i in range(N)]
wordvectors_one_matrix = random.rand(d,N)

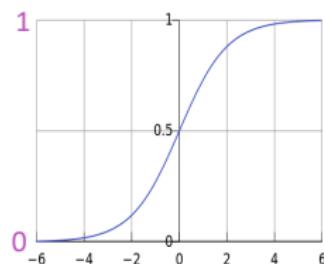
%timeit W.dot(wordvectors_list[i]) for i in range(N)
%timeit W.dot(wordvectors_one_matrix)
```

- The (10x) faster method is using a C x N matrix
- Always try to use vectors and matrices rather than for loops!
- You should speed-test your code a lot too!!
- These differences go from 1 to 2 orders of magnitude with GPUs
- tl;dr: Matrices are awesome!!!

# Non-linearities: The starting points

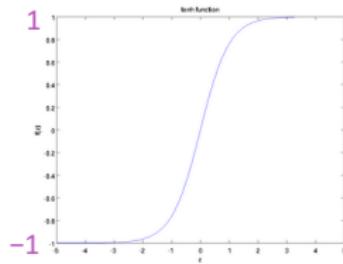
logistic (“sigmoid”)

$$f(z) = \frac{1}{1 + \exp(-z)}.$$



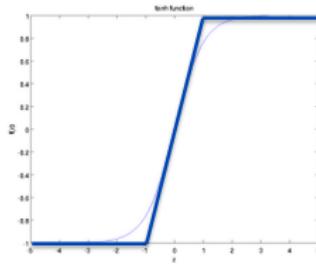
tanh

$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$



hard tanh

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases}$$



tanh is just a rescaled and shifted sigmoid ( $2 \times$  as steep,  $[-1,1]$ ):

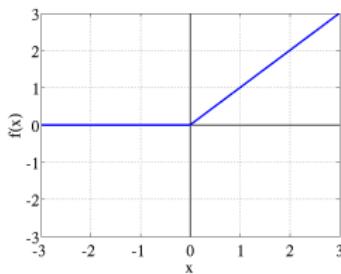
$$\tanh(z) = 2\text{logistic}(2z) - 1$$

Both logistic and tanh are still used in particular uses, but are no longer the defaults for making deep networks

# Non-linearities: The new world order

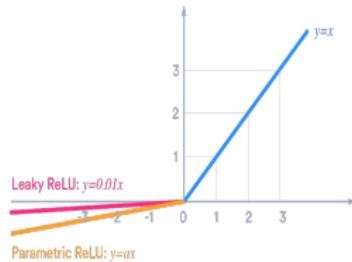
ReLU (rectified

linear unit) hard tanh  
 $\text{rect}(z) = \max(z, 0)$



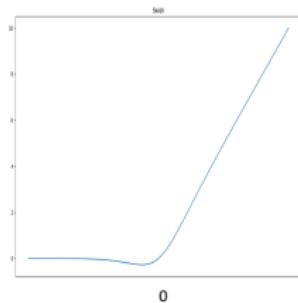
Leaky ReLU /

Parametric ReLU



Swish

[Ramachandran, Zoph & Le 2017]



- For building a deep feed-forward network, the first thing you should try is ReLU — it trains quickly and performs well due to good gradient backflow

## Parameter Initialization

- You normally must initialize weights to small random values
  - To avoid symmetries that prevent learning/specialization
- Initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g., mean target or inverse sigmoid of mean target)
- Initialize **all other weights**  $\sim \text{Uniform}(-r, r)$ , with  $r$  chosen so numbers get neither too big or too small
- Xavier initialization has variance inversely proportional to fan-in  $n_{in}$  (previous layer size) and fan-out  $n_{out}$  (next layer size):

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

# Optimizers

- Usually, plain SGD will work just fine
  - However, getting good results will often require hand-tuning the learning rate (next slide)
- For more complex nets and situations, or just to avoid worry, you often do better with one of a family of more sophisticated “adaptive” optimizers that scale the parameter adjustment by an accumulated gradient.
  - These models give differentiak per-parameter learning rates
    - Adagrad
    - RMSprop
    - Adam ← A fairly good, safe place to begin in many cases
    - SparseAdam
    - ...

## Learning Rates

- You can just use a constant learning rate. Start around  $lr = 0.001$ ?
  - It must be order of magnitude right – try powers of 10
    - Too big: model may diverge or not converge
    - Too small: your model may not have trained by the deadline
- Better results can generally be obtained by allowing learning rates to decrease as you train
  - By hand: halve the learning rate every  $k$  epochs
    - An epoch = a pass through the data (shuffled or sampled)
  - By a formula:  $lr = lr_0 e^{-kt}$ , for epoch  $t$
  - There are fancier methods like cyclic learning rates (q.v.)
- Fancier optimizers still use a learning rate but it may be an initial rate that the optimizer shrinks – so may need to start high

# Content

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