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

Content

Limitation of linear classifiers

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

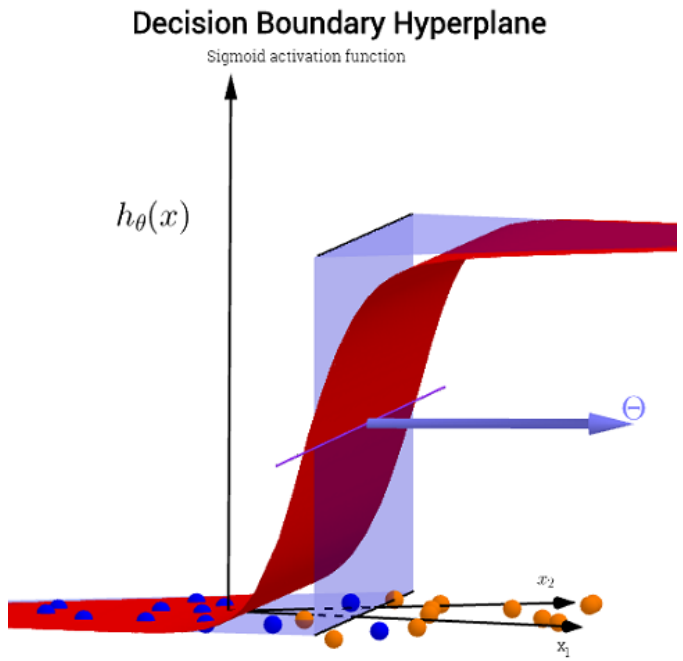


Figure source: https://stats.stackexchange.com/questions/291492/

Limitation of linear classifiers

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

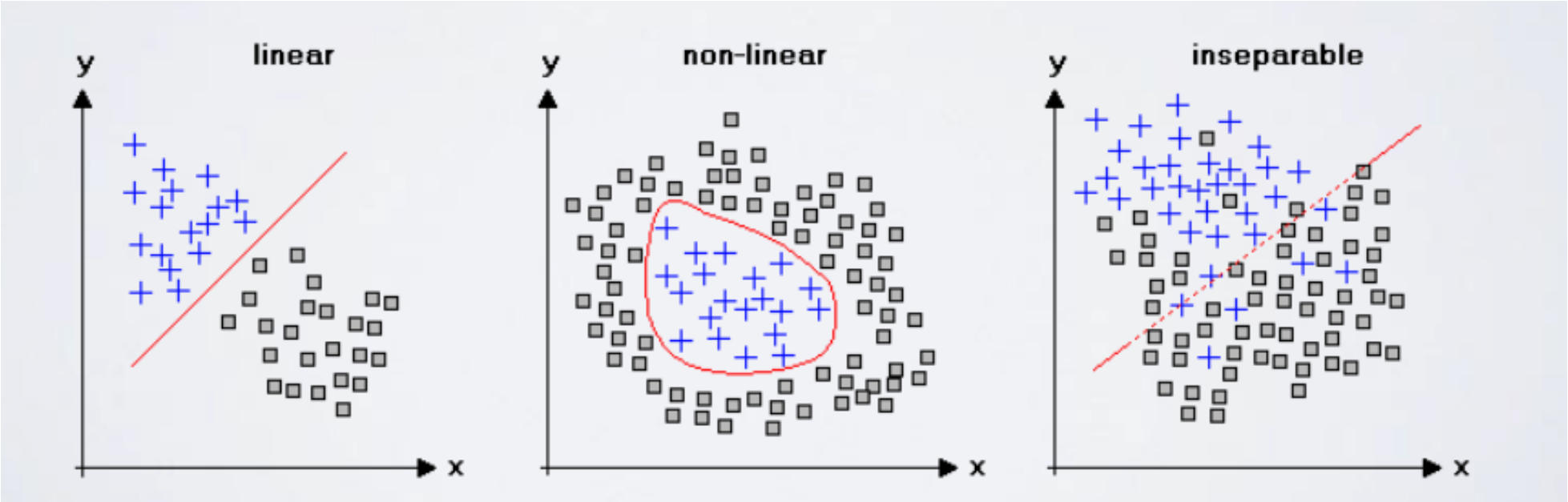


Figure source: http://dingyancs.blogspot.com/2017/10/mle-linear-classification.html

Exclusive-or function

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

X_1	X_2	Y
0	0	0
0	1	1
1	0	1
1	1	0

$Y = X_1 \oplus X_2$

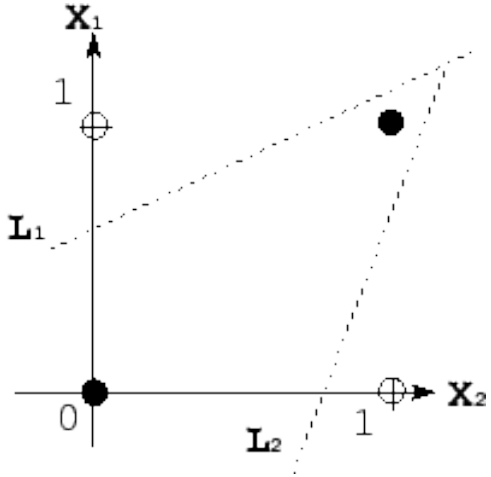
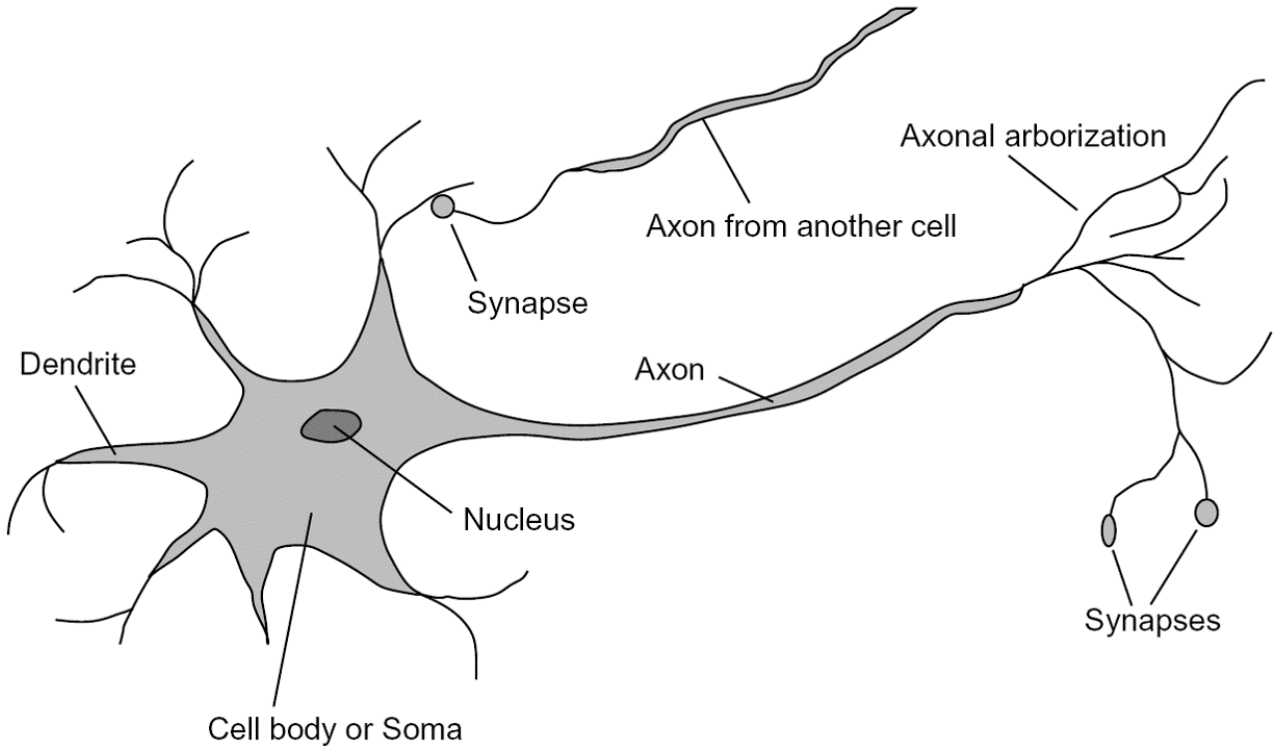


Figure source: http://www.ece.utep.edu/research/webfuzzy/docs/kk-thesis/kk-thesis-html/node19.html

Artificial Neural networks (ANNs)

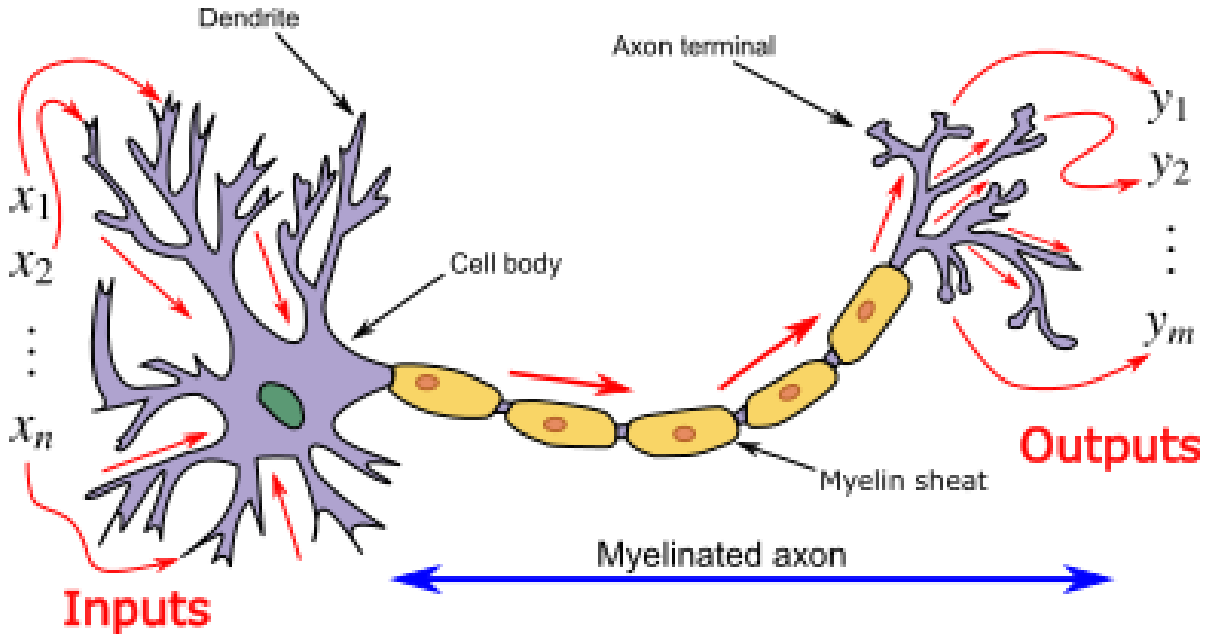
- Neural networks (NNs), or artificial neural networks (ANNs), provide an efficient way to solve non-linear-separable problem.
- Artificial neural networks (ANNs) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. – Wikipedia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

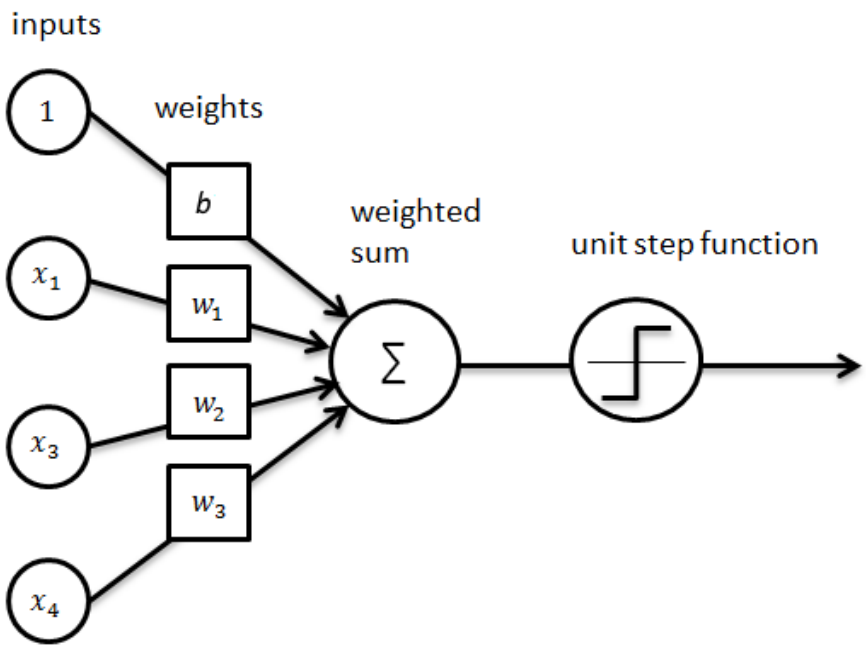
By Egm4313.s12 (Prof. Loc Vu-Quoc) - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=72816083

Content

Content

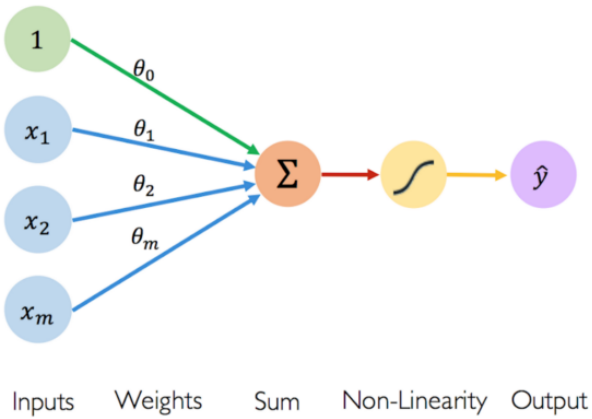
Perceptron – simplest artificial neuron

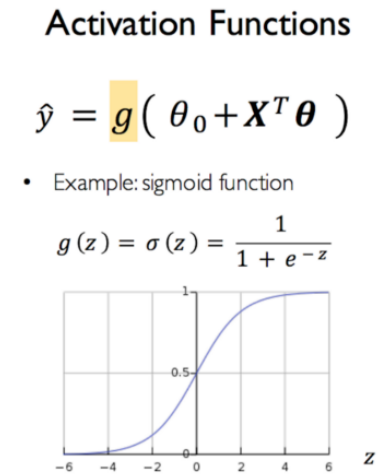
$$f(x) = \begin{cases} 0, & \text{if } \sum_i w_i x_i + b \leq 0 \\ 1, & \text{otherwise} \end{cases}$$



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.





MIT: Alexander Amini, 2018 introtodeeplearning.com

From linear classifier to non-linear classifier

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

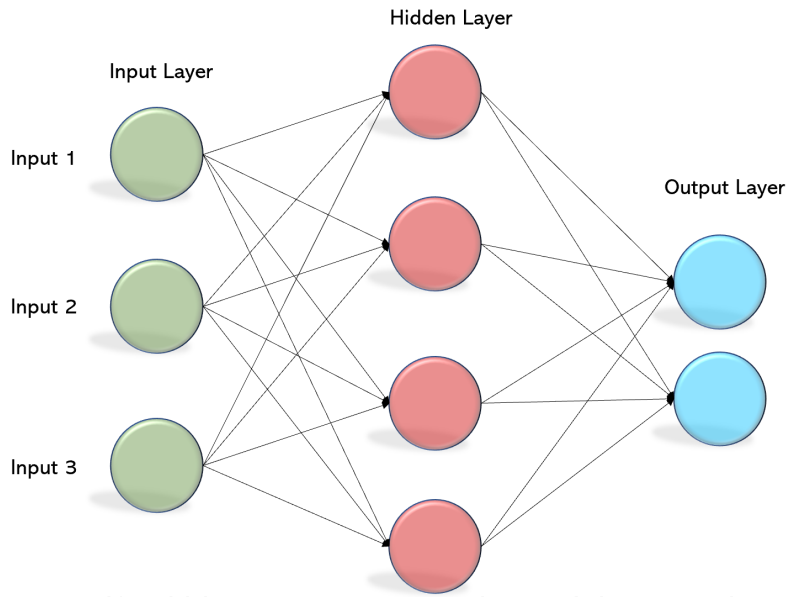


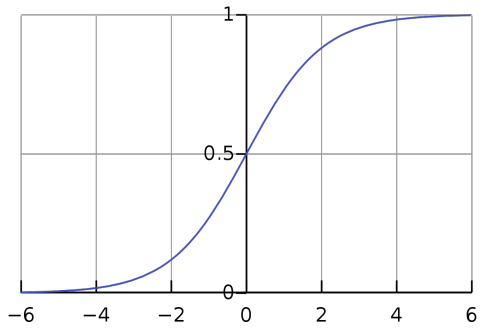
Figure source: https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f

Multilayer perceptrons

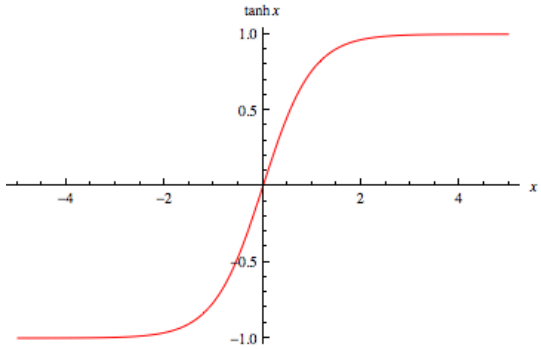
- An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer.
- Except for the input nodes, each node is a neuron that uses a nonlinear activation function.
- MLP utilizes a supervised learning technique called backpropagation for training.
- Its multiple layers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

Activation functions

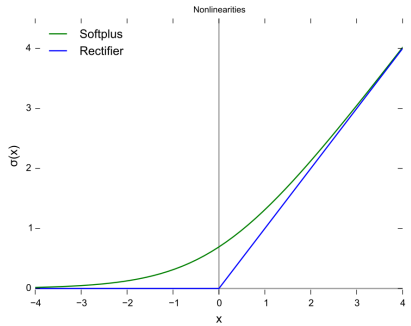
Logistic: $f(x) = \frac{1}{1 + e^{-x}}$



Hybolic 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.

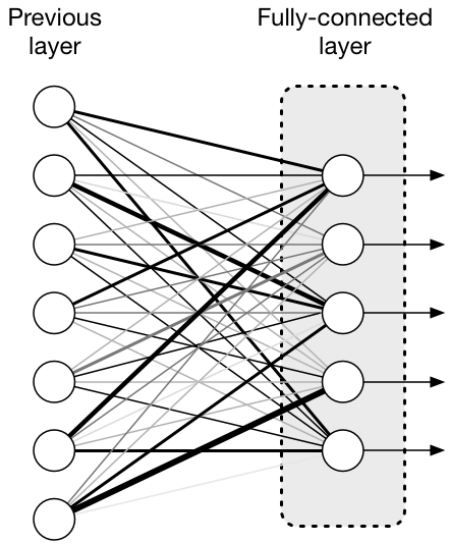


Figure source: https://mc.ai/fully-connected-layer-with-dynamic-input-shape/

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.

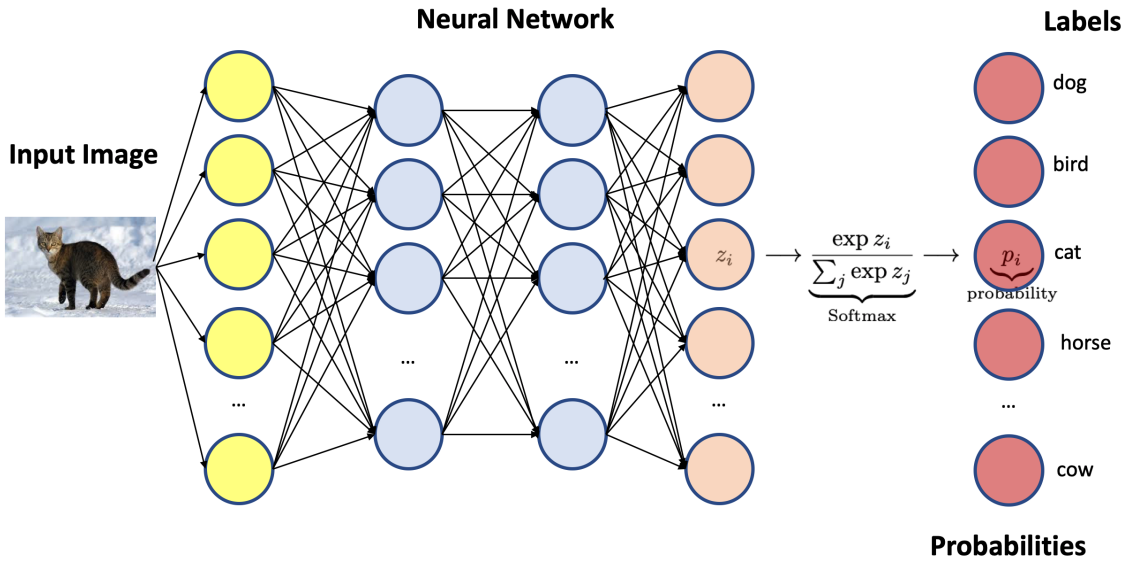
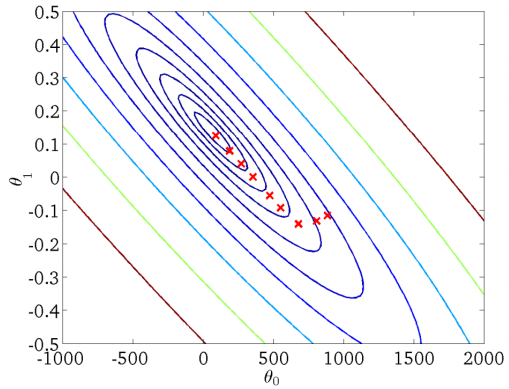


Figure source: https://gab41.lab41.org/entropic-ghosts-35670292bc87

Content

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 $\mathrm{d}\mathcal{E}/\mathrm{d}\mathbf{w}$, which is the vector of partial derivatives.
  - This is the average of $\mathrm{d}\mathcal{L}/\mathrm{d}\mathbf{w}$ over all the training examples, so in this lecture we focus on computing $\mathrm{d}\mathcal{L}/\mathrm{d}\mathbf{w}$.

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

$$\frac{\mathrm{d}}{\mathrm{d}t} f(x(t)) = \frac{\mathrm{d}f}{\mathrm{d}x} \frac{\mathrm{d}x}{\mathrm{d}t}.$$

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

$$\mathcal{L} = \frac{1}{2}(\sigma(wx+b)-t)^2$$

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

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

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 loss:

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Computing the derivatives:

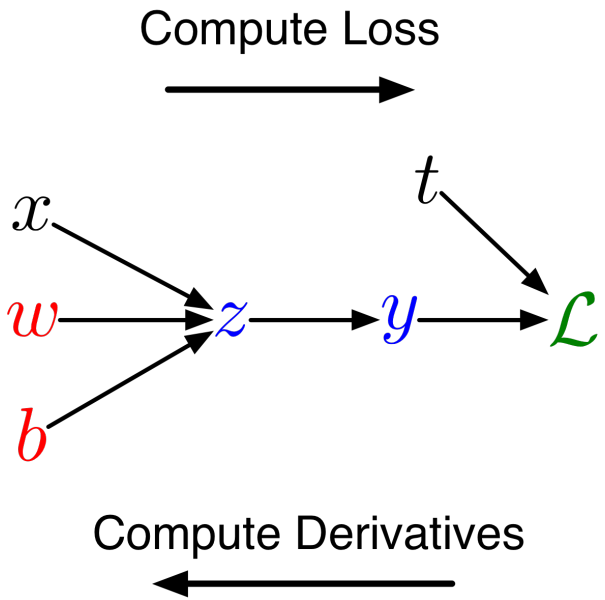
$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = y - t$$
$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\sigma'(z)$$
$$\frac{\partial\mathcal{L}}{\partial w} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}x$$
$$\frac{\partial\mathcal{L}}{\partial b} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}z}$$

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 $\overline{y}$ to denote the derivative $\mathrm{d}\mathcal{L}/\mathrm{d}y$, 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:

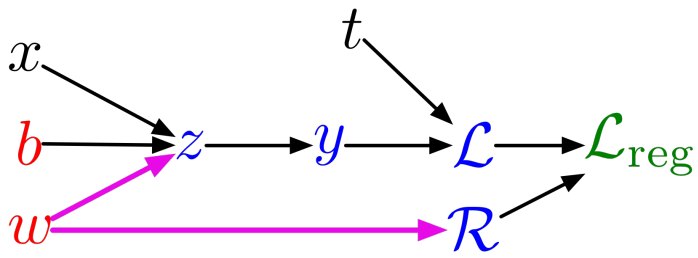
$$\overline{y} = y - t$$
$$\overline{z} = \overline{y}\,\sigma'(z)$$
$$\overline{w} = \overline{z}\,x$$
$$\overline{b} = \overline{z}$$

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

Multivariate Chain Rule

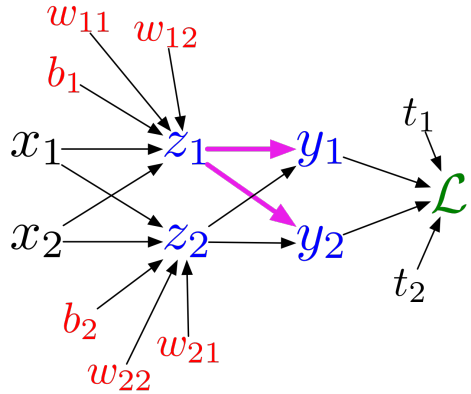
Problem: what if the computation graph has fan-out > 1?
This requires the multivariate Chain Rule!

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



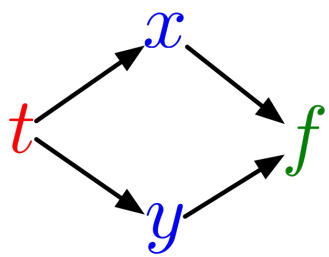
$$z_\ell = \sum_j w_{\ell j} x_j + b_\ell$$
$$y_k = \frac{e^{z_k}}{\sum_\ell e^{z_\ell}}$$
$$\mathcal{L} = -\sum_k t_k \log y_k$$

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

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



- Example:

$$\begin{aligned} f(x, y) &= y + e^{xy} \\ x(t) &= \cos t \\ y(t) &= t^2 \end{aligned}$$

- Plug in to Chain Rule:

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

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

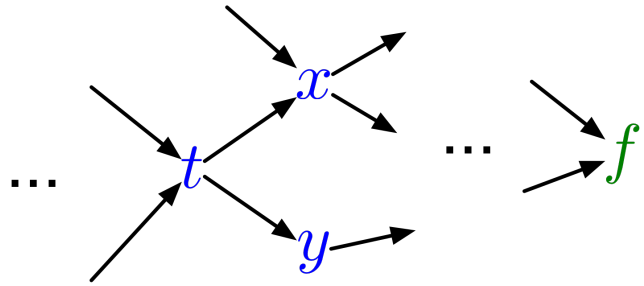
Multivariable Chain Rule

- In the context of backpropagation:

Mathematical expressions to be evaluated

$$\frac{\mathrm{d}f}{\mathrm{d}t} = \frac{\partial f}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial f}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

Values already computed by our program



- In our notation:

$$\overline{t} = \overline{x}\frac{\mathrm{d}x}{\mathrm{d}t} + \overline{y}\frac{\mathrm{d}y}{\mathrm{d}t}$$

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

Backpropagation

Full backpropagation algorithm:

Let $v_1, \ldots, v_N$ 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).

forward pass

For $i = 1, \ldots, N$

Compute v_i as a function of $\mathrm{Pa}(v_i)$

backward pass

$\overline{v_N} = 1$

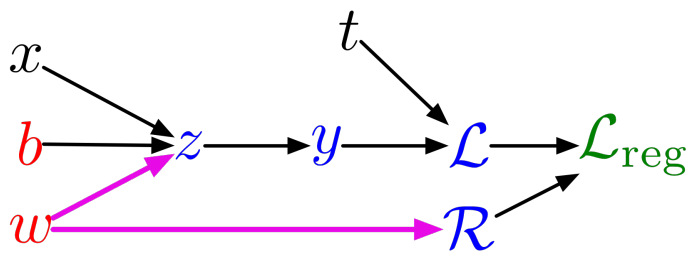
For $i = N-1, \ldots, 1$

$\overline{v_i} = \sum_{j \in \mathrm{Ch}(v_i)} \overline{v_j} \frac{\partial v_j}{\partial v_i}$

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

Backpropagation

Example: univariate logistic least squares regression



Forward pass:

$$z = wx + b$$
$$y = \sigma(z)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$
$$\mathcal{R} = \frac{1}{2}w^2$$
$$\mathcal{L}_{\text{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

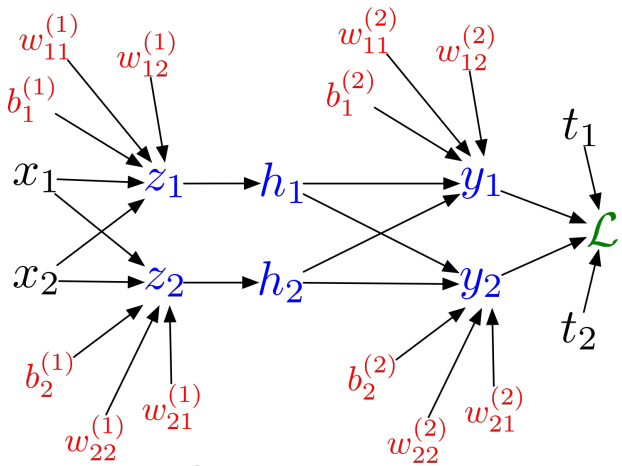
Backward pass:

$$\overline{\mathcal{L}_{\text{reg}}} = 1$$
$$\overline{\mathcal{R}} = \overline{\mathcal{L}_{\text{reg}}}\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{R}} = \overline{\mathcal{L}_{\text{reg}}}\,\lambda$$
$$\overline{\mathcal{L}} = \overline{\mathcal{L}_{\text{reg}}}\frac{\mathrm{d}\mathcal{L}_{\text{reg}}}{\mathrm{d}\mathcal{L}} = \overline{\mathcal{L}_{\text{reg}}}$$
$$\overline{y} = \overline{\mathcal{L}}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y} = \overline{\mathcal{L}}\,(y - t)$$
$$\overline{z} = \overline{y}\frac{\mathrm{d}y}{\mathrm{d}z} = \overline{y}\,\sigma'(z)$$
$$\overline{w} = \overline{z}\frac{\partial z}{\partial w} + \overline{\mathcal{R}}\frac{\mathrm{d}\mathcal{R}}{\mathrm{d}w} = \overline{z}\,x + \overline{\mathcal{R}}\,w$$
$$\overline{b} = \overline{z}\frac{\partial z}{\partial b} = \overline{z}$$

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:

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

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

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

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

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

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

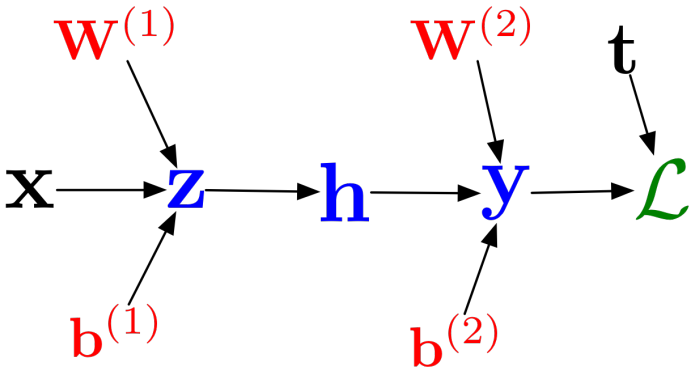
$$\overline{w_{ij}^{(1)}} = \overline{z_i} \, x_j$$

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

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

Backpropagation

In vectorized form:



Forward pass:

$$\mathbf{z} = \mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}$$
$$\mathbf{h} = \sigma(\mathbf{z})$$
$$\mathbf{y} = \mathbf{W}^{(2)}\mathbf{h} + \mathbf{b}^{(2)}$$
$$\mathcal{L} = \frac{1}{2}\|\mathbf{t} - \mathbf{y}\|^2$$

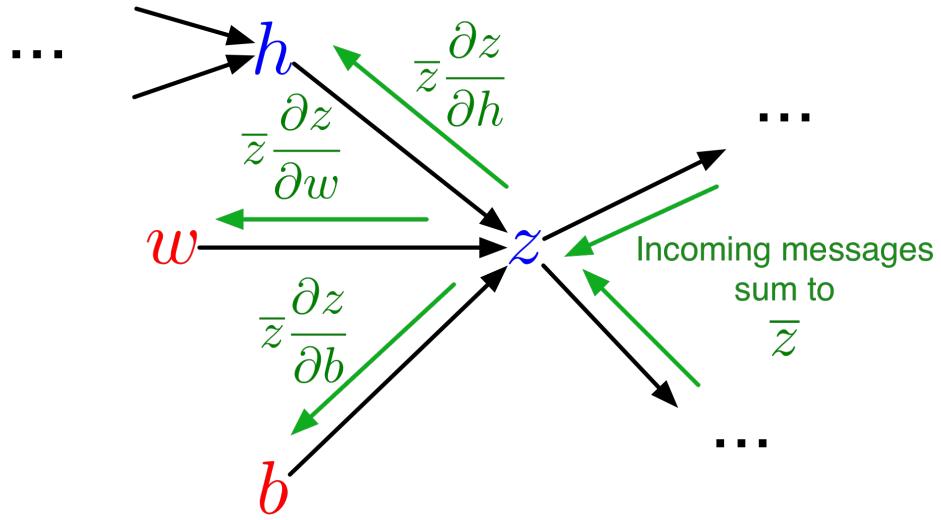
Backward pass:

$$\overline{\mathcal{L}} = 1$$
$$\overline{\mathbf{y}} = \overline{\mathcal{L}}(\mathbf{y} - \mathbf{t})$$
$$\overline{\mathbf{W}^{(2)}} = \overline{\mathbf{y}}\mathbf{h}^\top$$
$$\overline{\mathbf{b}^{(2)}} = \overline{\mathbf{y}}$$
$$\overline{\mathbf{h}} = \mathbf{W}^{(2)\top}\overline{\mathbf{y}}$$
$$\overline{\mathbf{z}} = \overline{\mathbf{h}} \circ \sigma'(\mathbf{z})$$
$$\overline{\mathbf{W}^{(1)}} = \overline{\mathbf{z}}\mathbf{x}^\top$$
$$\overline{\mathbf{b}^{(1)}} = \overline{\mathbf{z}}$$

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

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:

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

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

$$\overline{x_j} = \sum_i \overline{y_i} \frac{\partial y_i}{\partial x_j} \qquad \overline{\mathbf{x}} = \overline{\mathbf{y}}^\top \mathbf{J}$$

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

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

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

Vector-Jacobian Products

Examples

- Matrix-vector product

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

- Elementwise operations

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

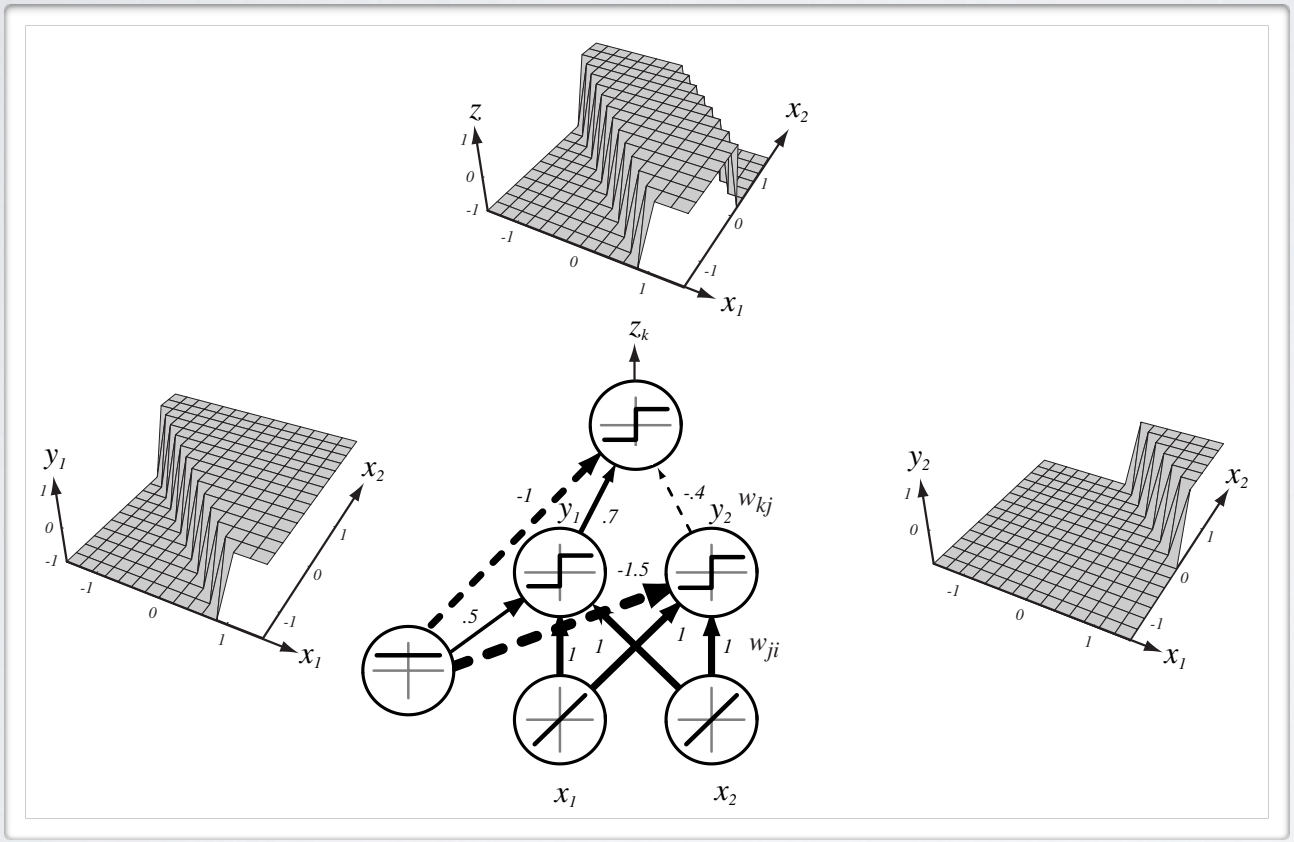
- Note: we never explicitly construct the Jacobian. It's usually simpler and more efficient to compute the VJP directly.

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

Content

Capacity of multilayer perceptron

Topics: single hidden layer neural network

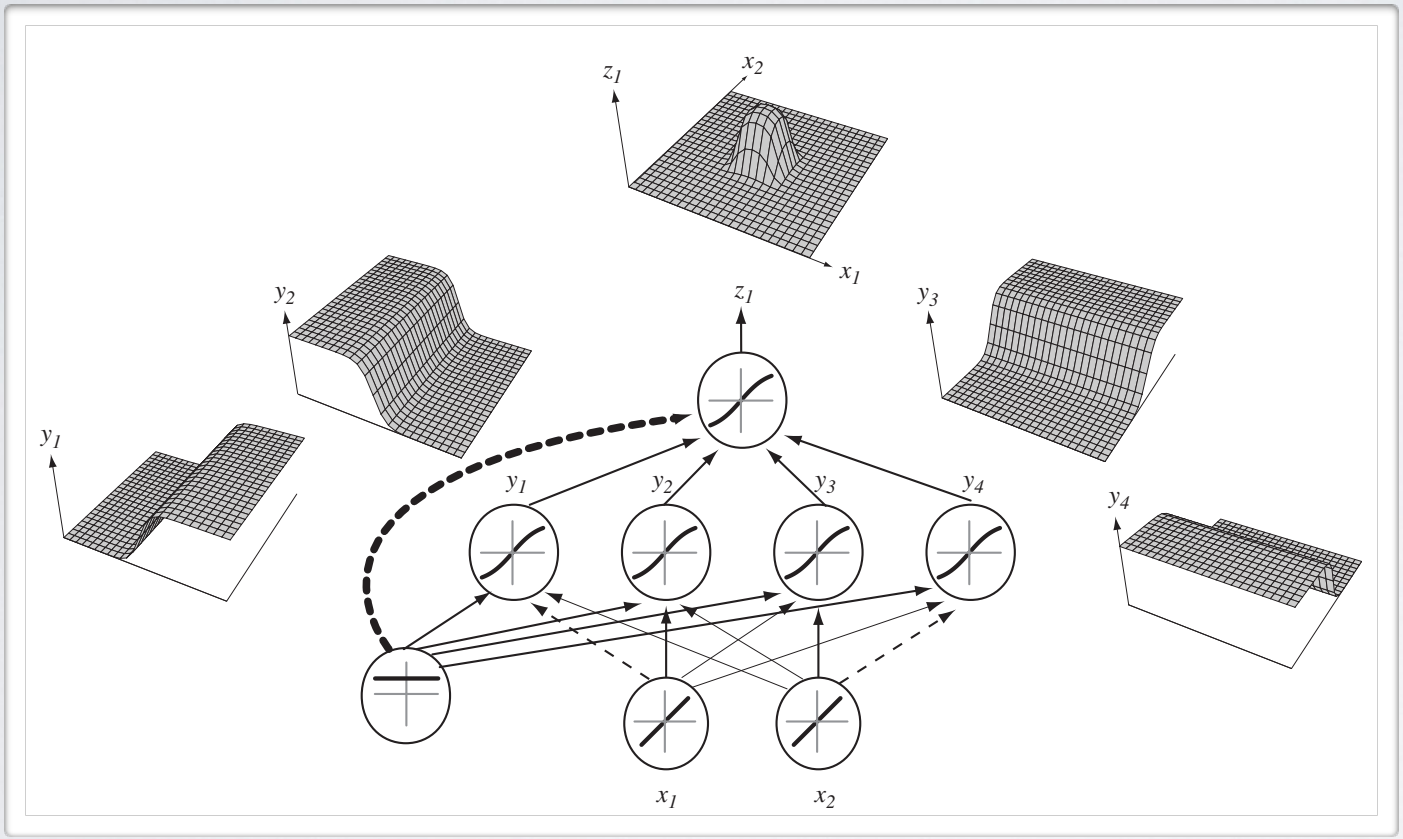


(from Pascal Vincent's slides)

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

Capacity of multilayer perceptron

Topics: single hidden layer neural network

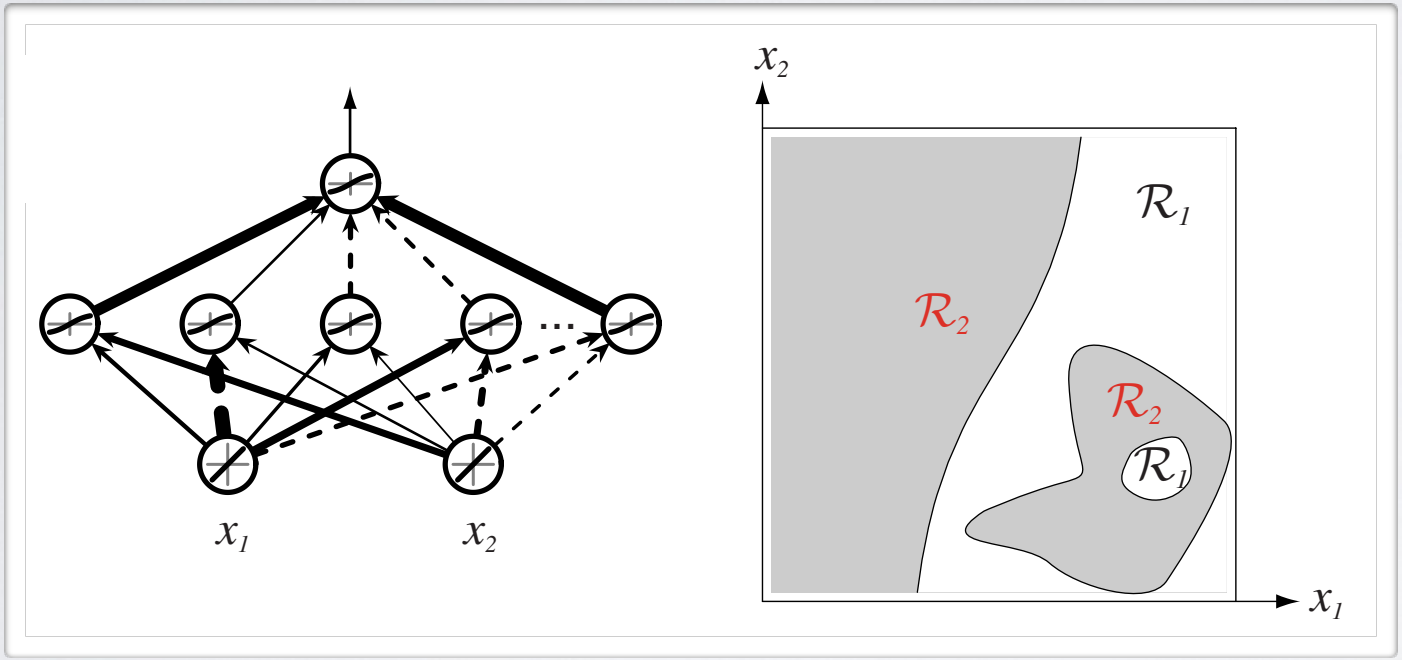


(from Pascal Vincent's slides)

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

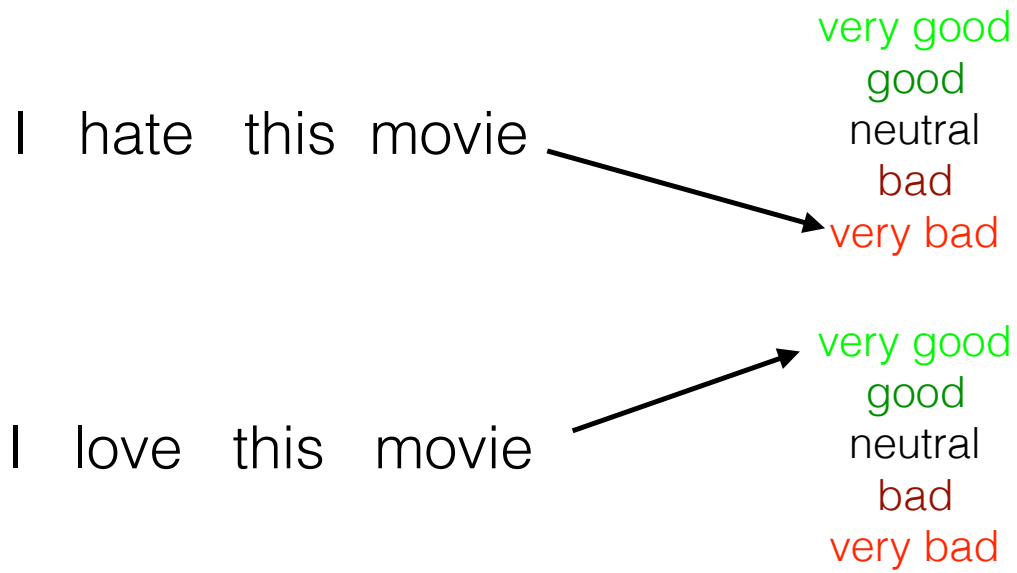
- Universal approximation theorem (Hornik, 1991):
  - "a single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units"
- The result applies for sigmoid, tanh and many other hidden layer activation functions
- This is 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

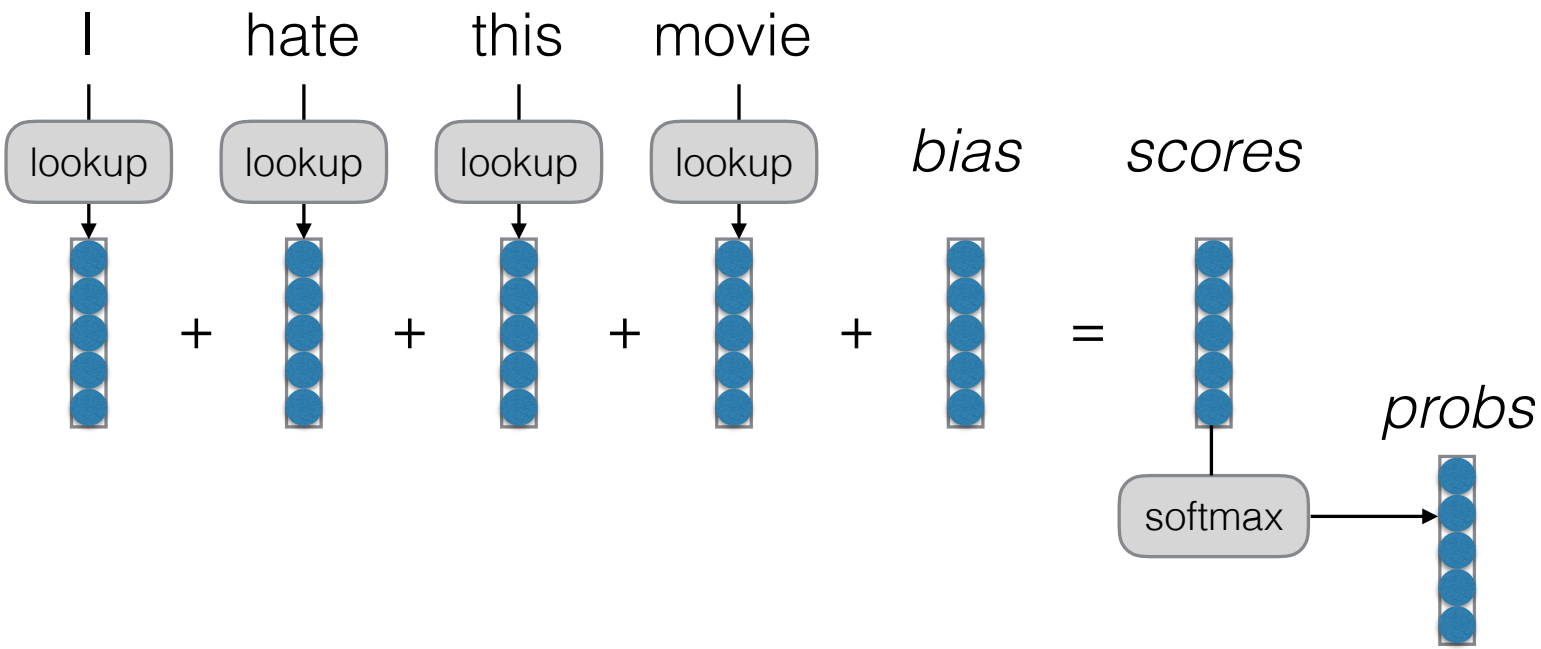
Content

An Example Prediction Problem: Sentence Classification



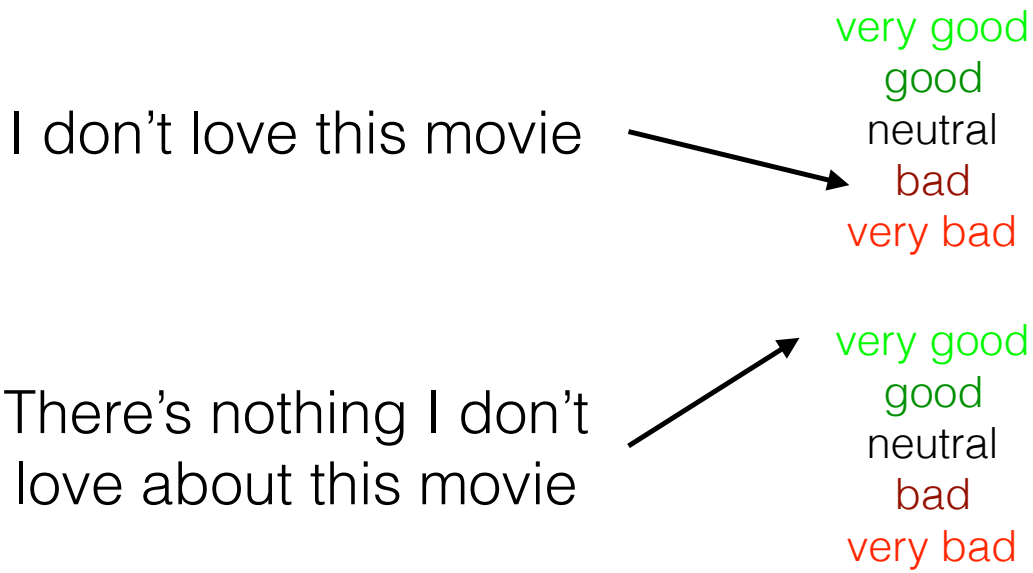
A slide from Graham Neubig's CS11-747 Neural Networks for NLP in CMU2019

A First Try: Bag of Words (BOW)



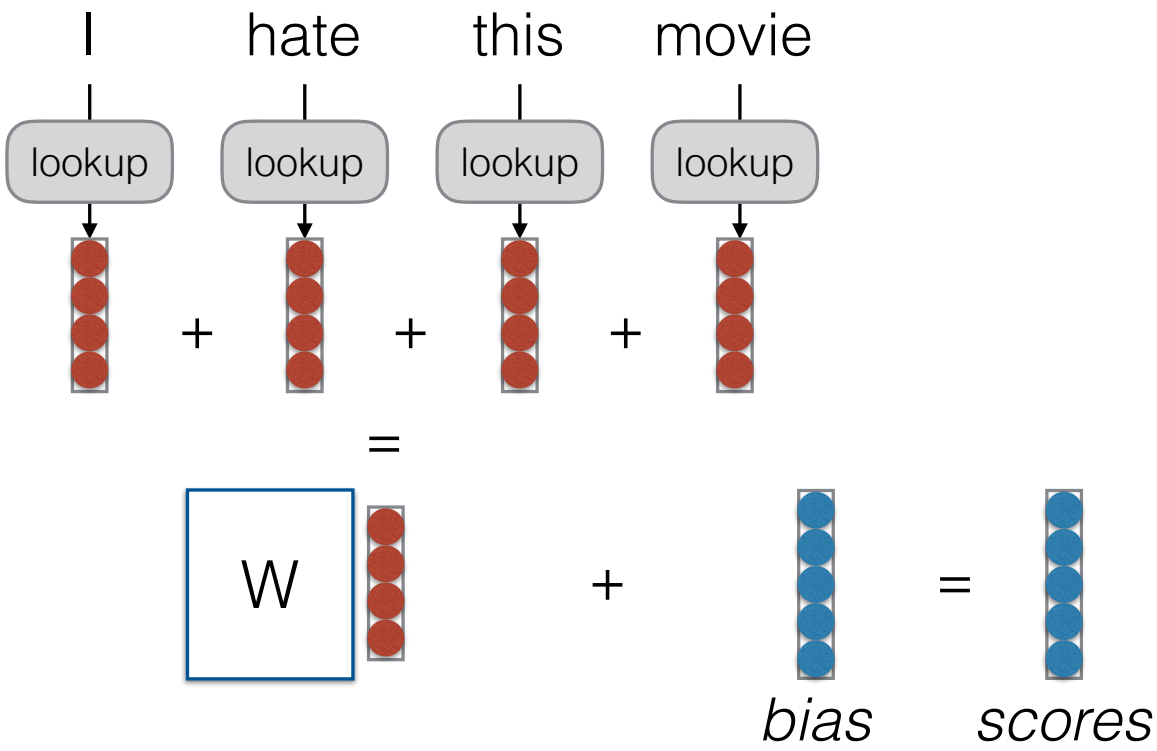
A slide from Graham Neubig's CS11-747 Neural Networks for NLP in CMU2019

Build It, Break It



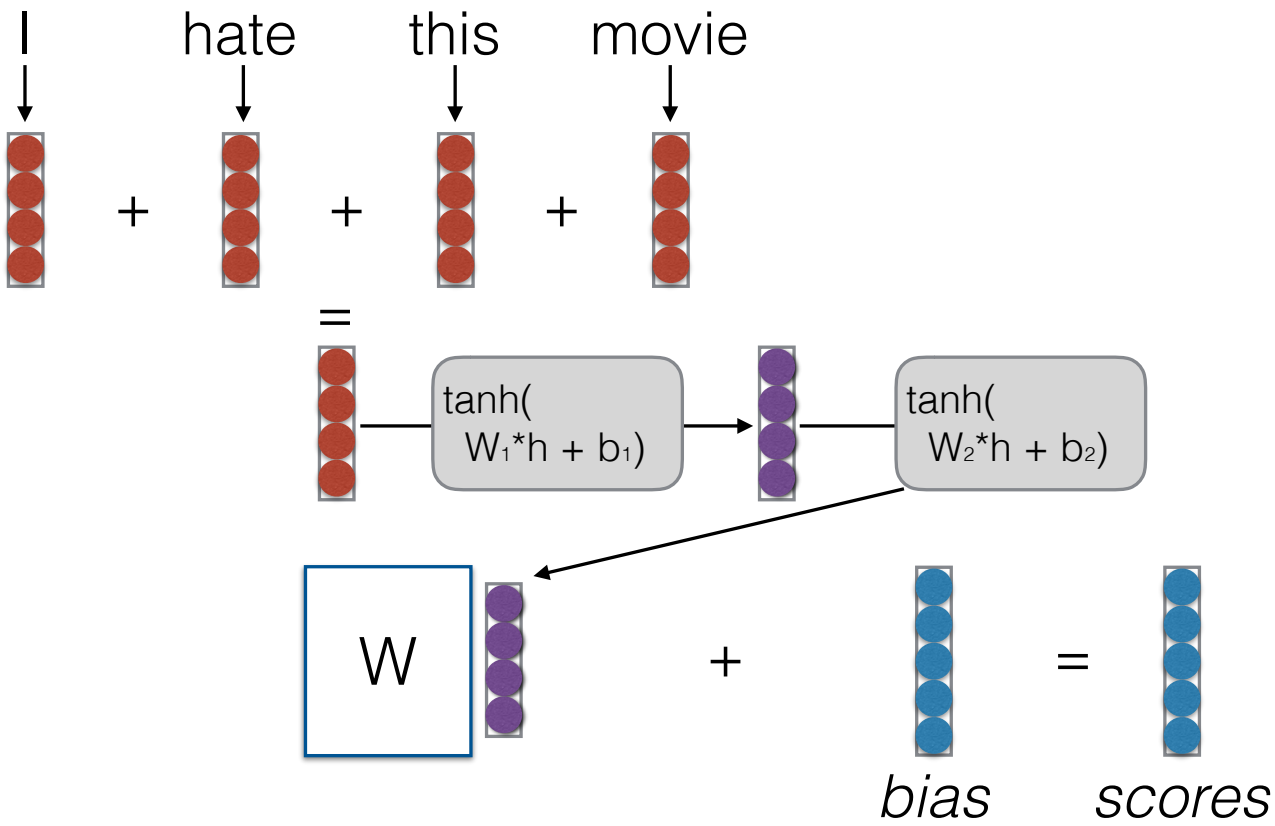
A slide from Graham Neubig's CS11-747 Neural Networks for NLP in CMU2019

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

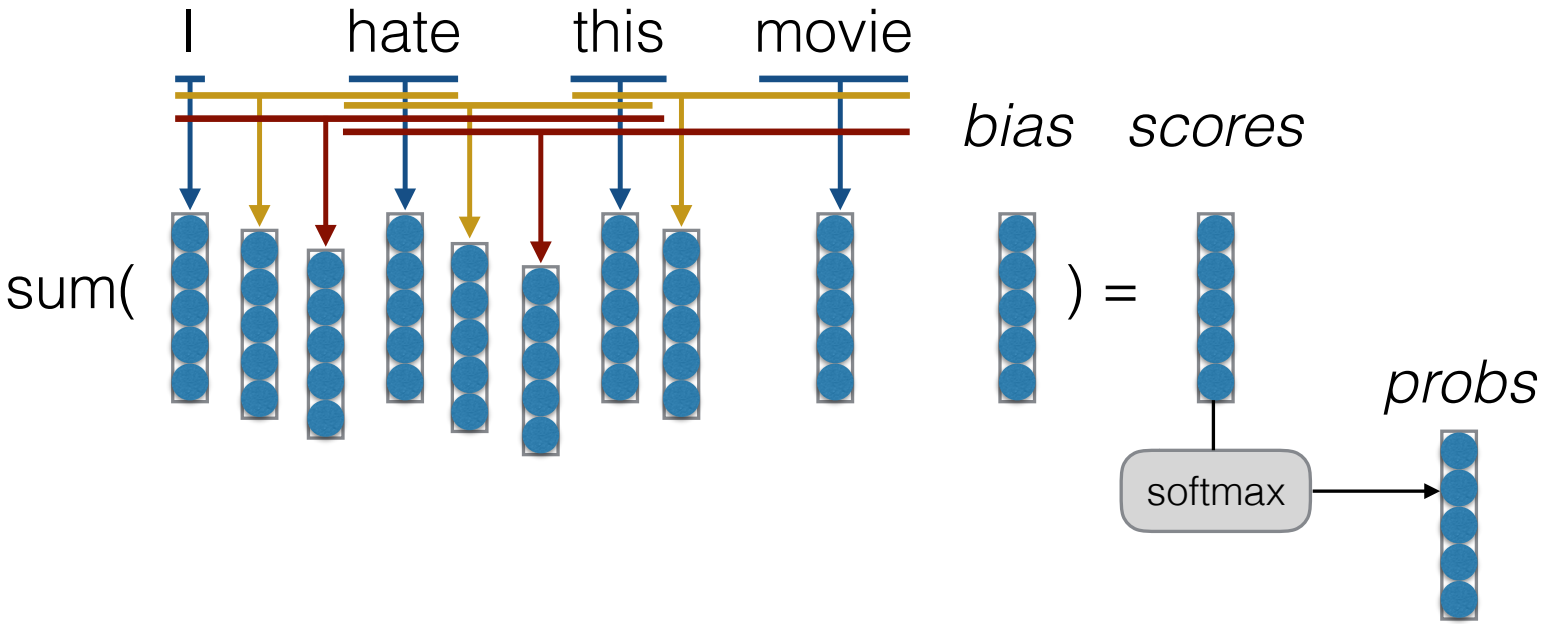
What do Our Vectors Represent?

- 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

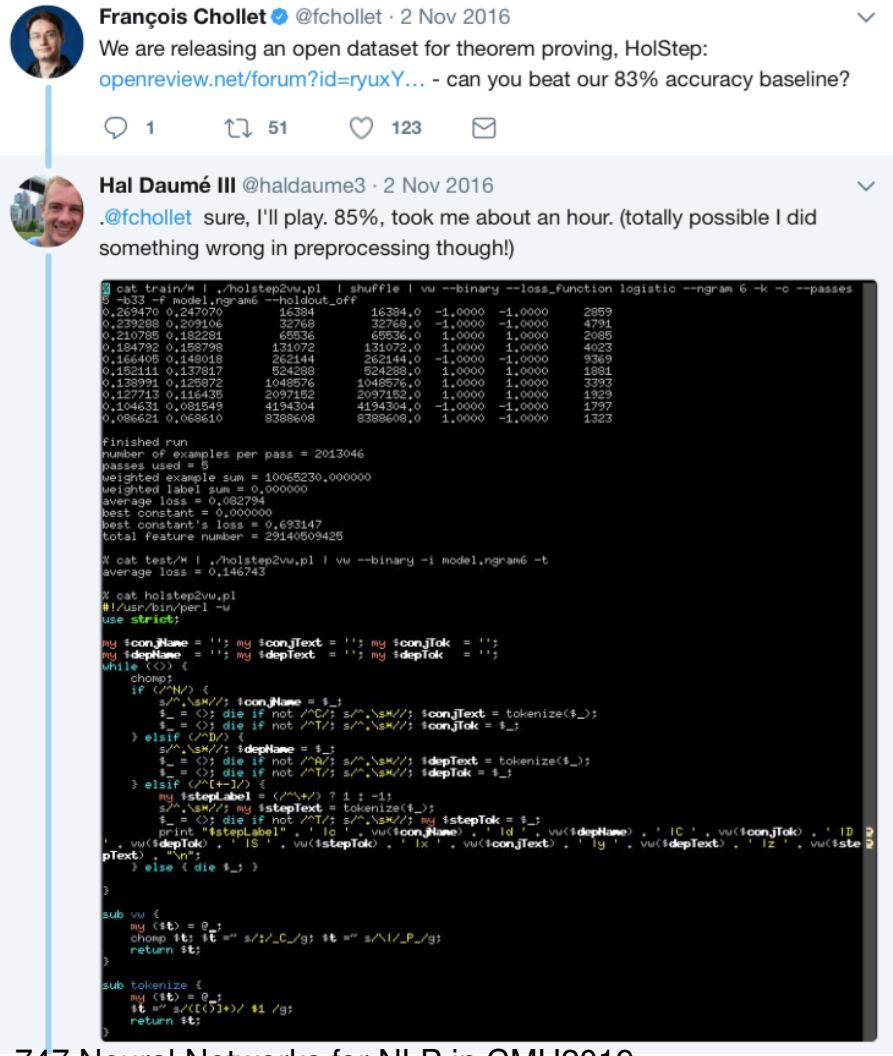
Bag of n-grams



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



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

What Problems w/ Bag of n-grams?

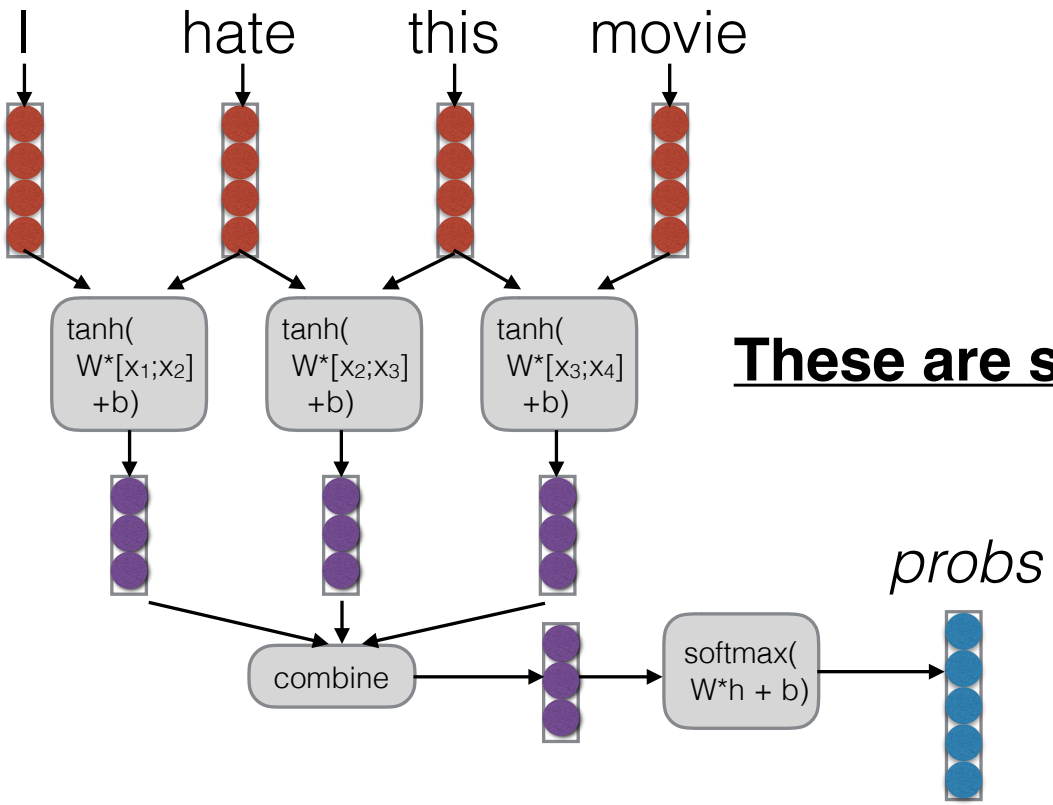
- Same as before: parameter explosion
- No sharing between similar words/n-grams

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

Content

1-dimensional Convolutions / Time-delay Networks

(Waibel et al. 1989)

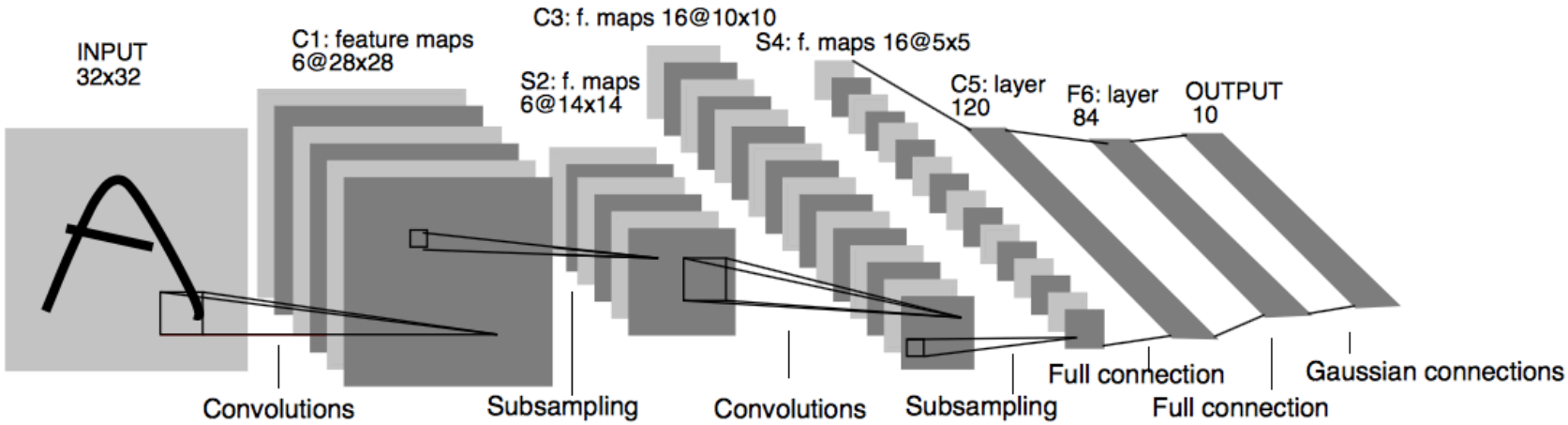


These are soft 2-grams!

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

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

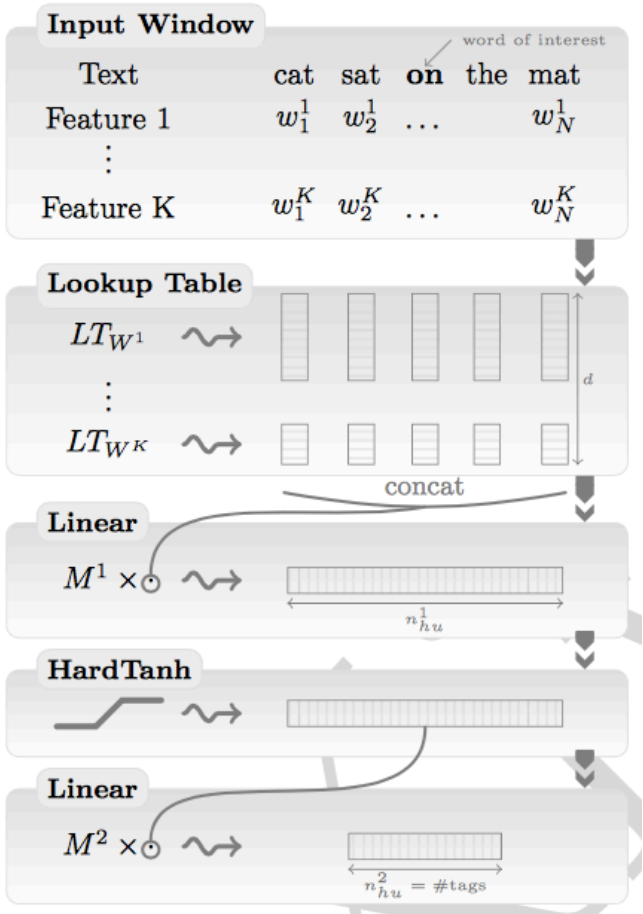
CNNs for Text
(Collobert and Weston 2011)

- 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

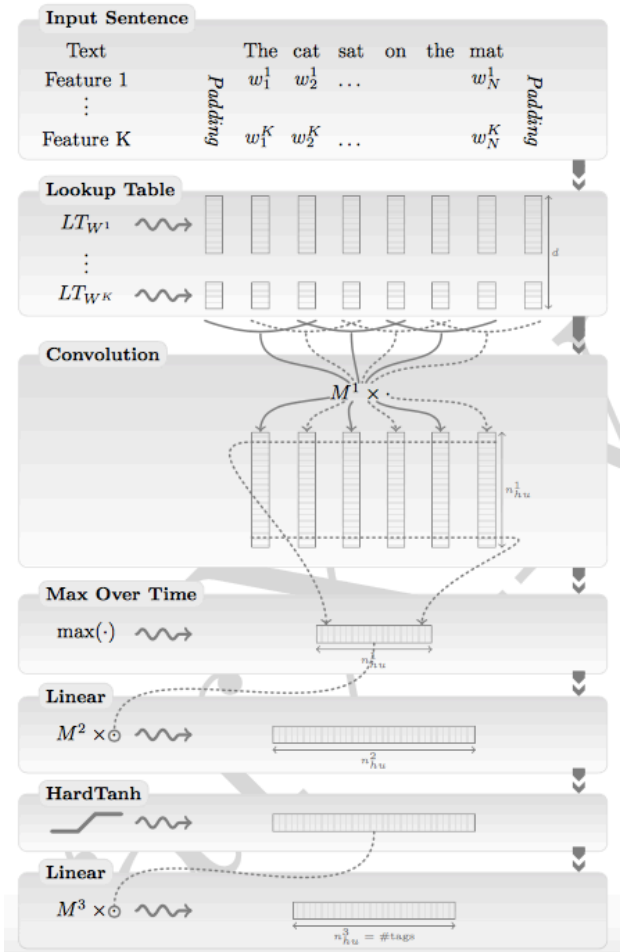
(Collobert and Weston 2011)



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

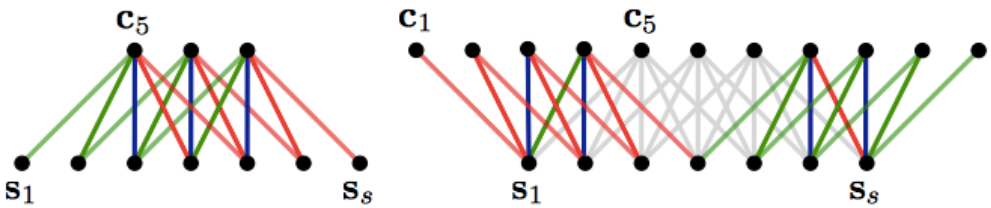
- 2D convolution function takes input + parameters
- **Input:** 3D tensor
  - rows (e.g. words), columns, features ("channels")
- **Parameters/Filters:** 4D tensor
  - rows, columns, input features, output features

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

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"*)

Narrow →



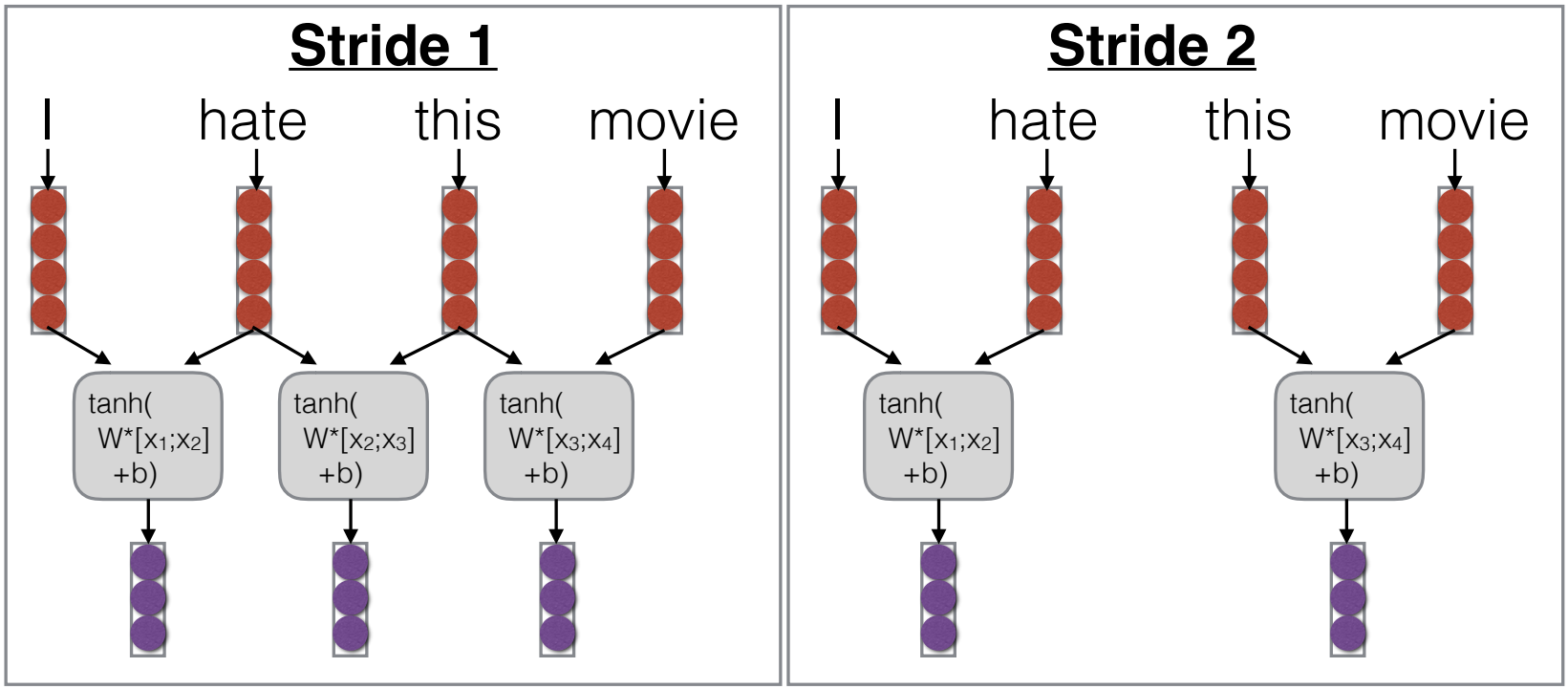
← Wide

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?"

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

Content

Stacked Convolution

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

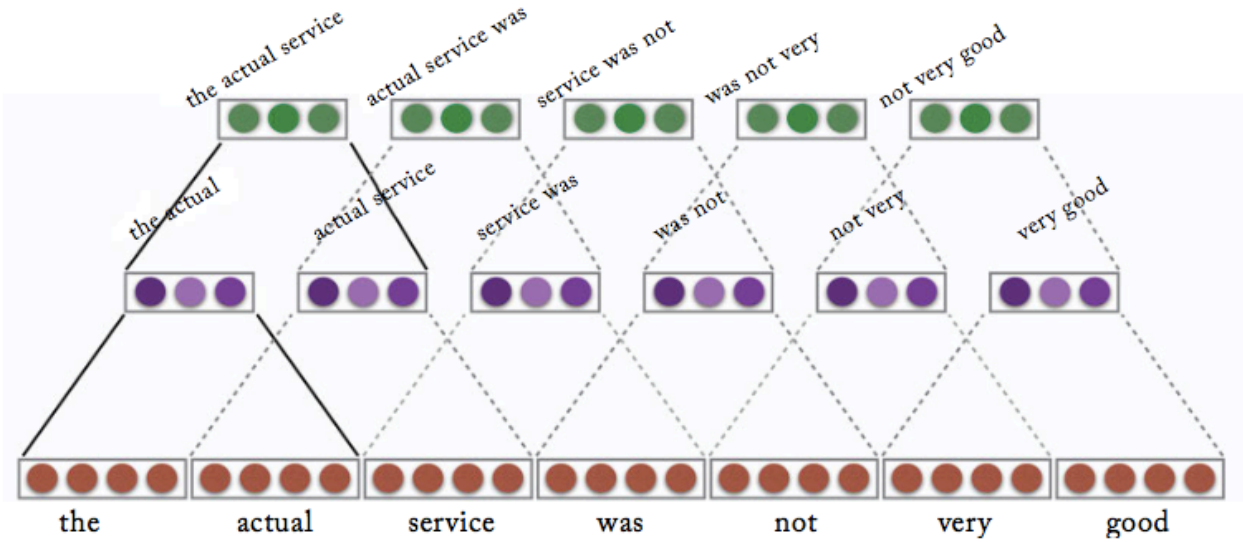


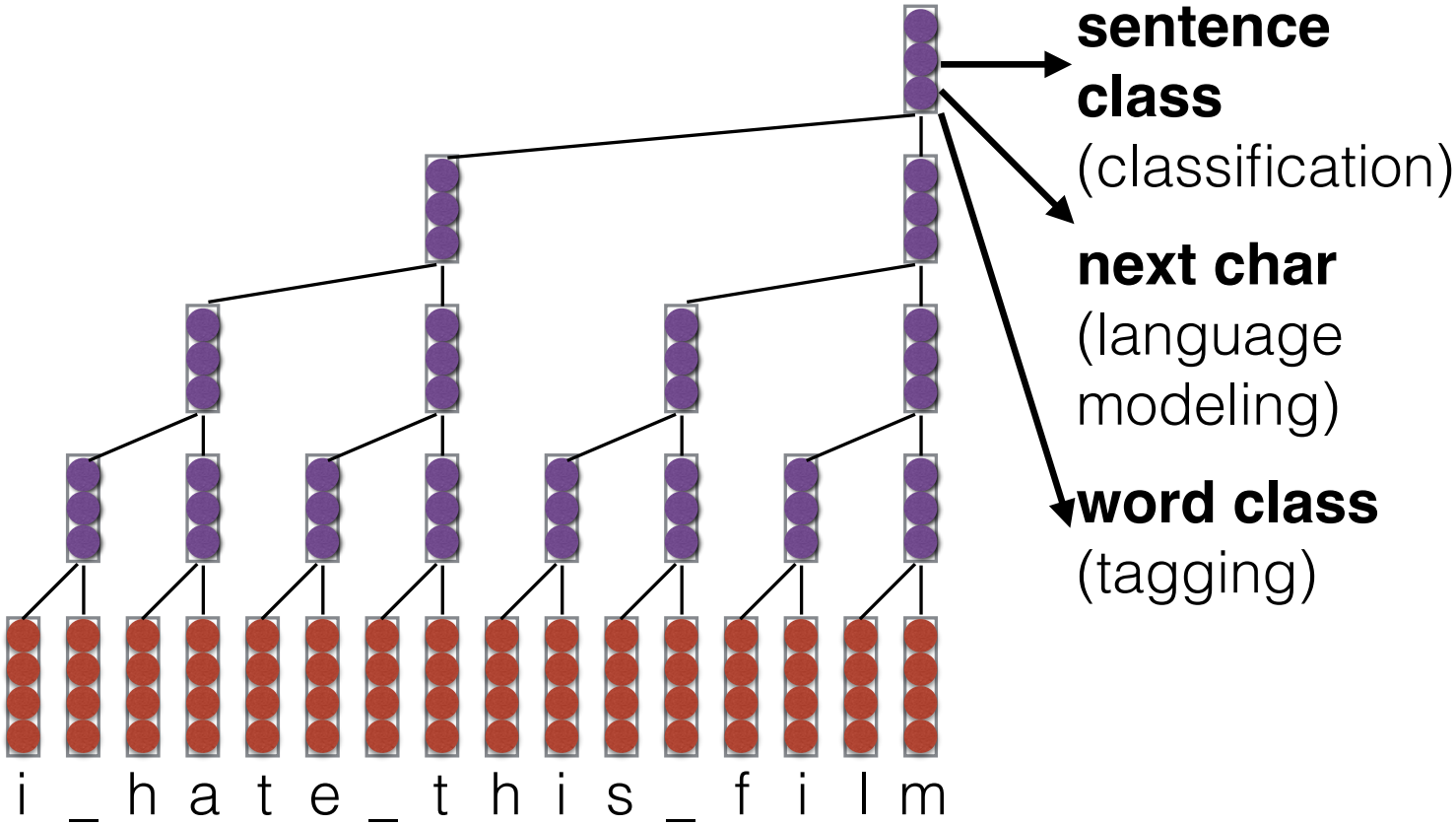
Image Credit: Goldberg Book

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

Dilated Convolution

(e.g. Kalchbrenner et al. 2016)

- **Gradually increase stride**, every time step (no reduction in length)



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

Why (Dilated) Convolution for Modeling Sentences?

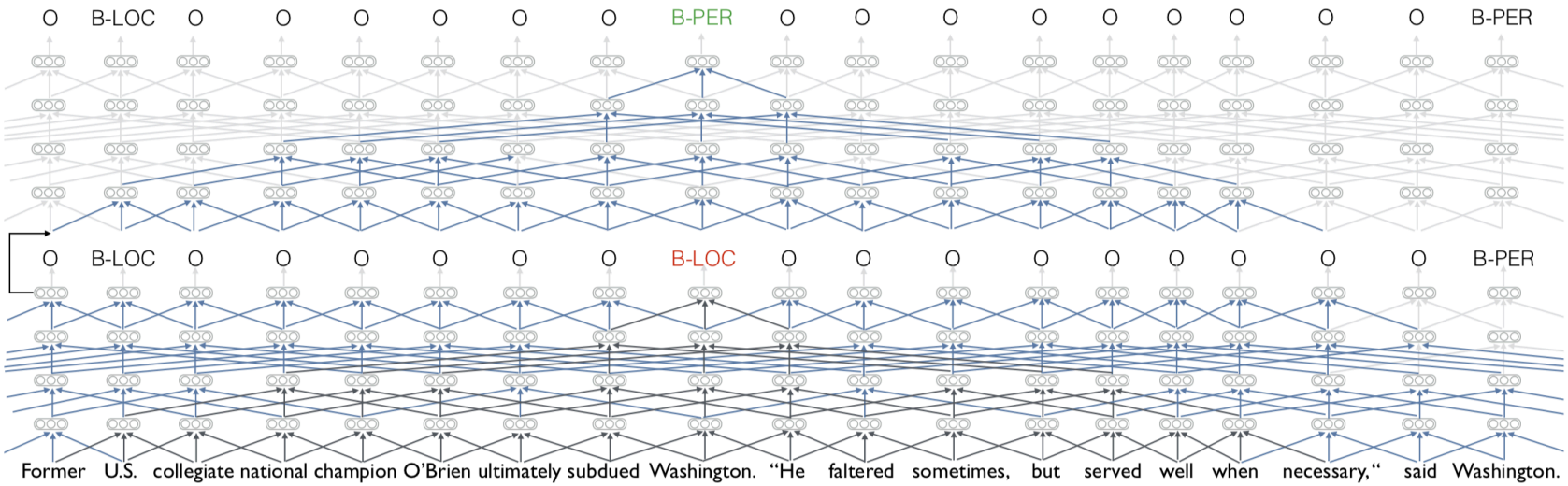
- In contrast to recurrent neural networks (next class)
- **+** Fewer steps from each word to the final representation: RNN O(N), Dilated CNN O(log N)
- **+** Easier to parallelize on GPU
- **-** Slightly less natural for arbitrary-length dependencies
- **-** A bit slower on CPU?

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

Iterated Dilated Convolution

(Strubell+ 2017)

- Multiple iterations of the same stack of dilated convolutions



- Wider context, more parameter efficient

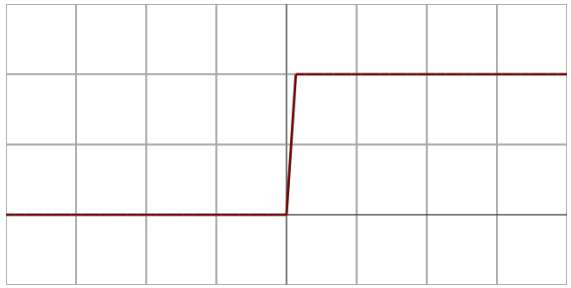
A slide from Graham Neubig's CS11-747 Neural Networks for NLP in CMU2019

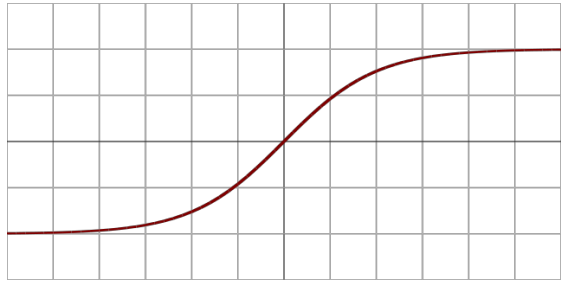
Content

Non-linear Functions

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

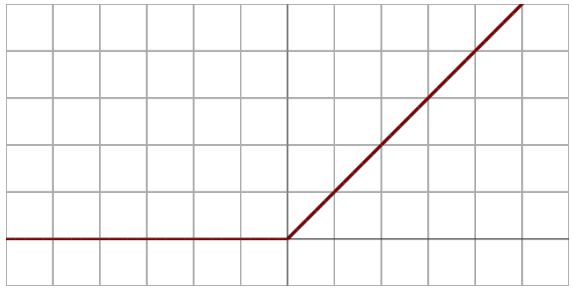
step

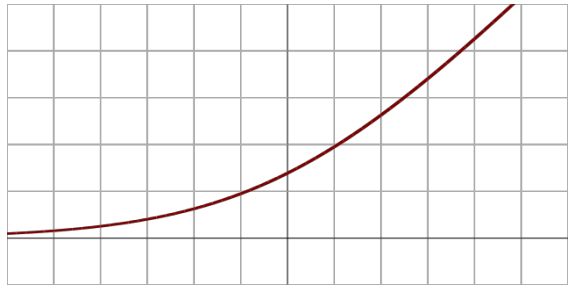




tanh

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

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

Ø	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
Ø	0.0	0.0	0.0	0.0

Ø,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,Ø	−0.5

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

3	1	2	−3
−1	2	1	−3
1	1	−1	1

11

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

3 channel 1D convolution with padding = 1

Ø	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
Ø	0.0	0.0	0.0	0.0

Ø,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,Ø	−0.5	−0.9	0.1

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

Could also use (zero) padding = 2

Also called “wide convolution”

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

conv1d, padded with max pooling over time

Ø	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
Ø	0.0	0.0	0.0	0.0

Ø,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,Ø	−0.5	−0.9	0.1
max p	0.3	1.6	1.4

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

13

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

conv1d, padded with ave pooling over time

Ø	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
Ø	0.0	0.0	0.0	0.0

Ø,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,Ø	−0.5	−0.9	0.1
ave p	−0.87	0.26	0.53

Apply 3 **filters** of size 3

	3	1	2	−3
	−1	2	1	−3
14	1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

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

Other less useful notions: stride = 2

Ø	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
Ø	0.0	0.0	0.0	0.0

Ø,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,Ø	−0.5	−0.9	0.1

Apply 3 **filters** of size 3

	3	1	2	−3
	−1	2	1	−3
16	1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

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

Less useful: local max pool, stride = 2

Ø	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
Ø	0.0	0.0	0.0	0.0

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

Ø,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,Ø	−0.5	−0.9	0.1
Ø	−Inf	−Inf	−Inf

Ø,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,Ø,Ø	−0.5	−0.9	0.1

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

conv1d, *k*-max pooling over time, *k* = 2

Ø	**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
Ø	**0.0**	**0.0**	**0.0**	**0.0**

Ø,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,Ø	−0.5	−0.9	0.1

2-max p	−0.2	1.6	1.4
	0.3	0.6	1.2

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

18

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

Other somewhat useful notions: dilation = 2

Ø	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
Ø	0.0	0.0	0.0	0.0

Apply 3 **filters** of size 3

3	1	2	−3
−1	2	1	−3
1	1	−1	1

1	0	0	1
1	0	−1	−1
0	1	0	1

1	−1	2	−1
1	0	−1	3
0	2	2	1

Ø,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,Ø	−0.5	−0.9	0.1

1,3,5	0.3	0.0
2,4,6		
3,5,7		

2	3	1
1	−1	−1
3	1	0

1	3	1
1	−1	−1
3	1	−1

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

Content

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)

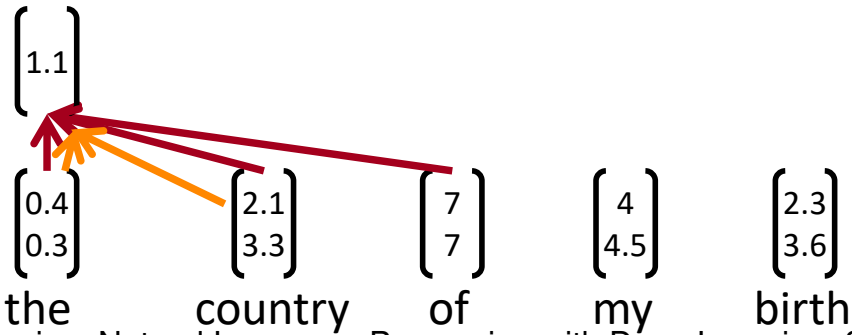
- Goal: Sentence classification:
  - Mainly positive or negative sentiment of a sentence
  - Other tasks like:
    - Subjective or objective language sentence
    - Question classification: about person, location, number, …

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

20

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: $\mathbf{x}_{1:n} = \mathbf{x}_1 \oplus x_2 \oplus \cdots \oplus \mathbf{x}_n$ (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, Standford U. CS224n

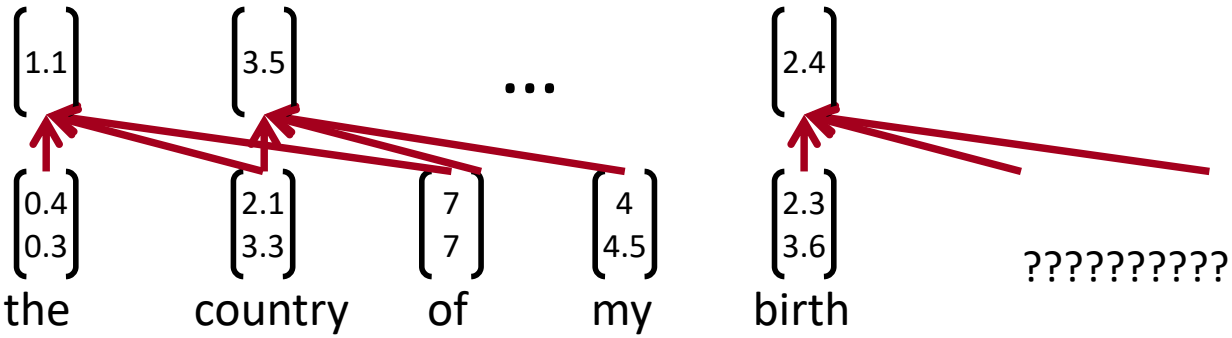
21

Single layer CNN

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

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

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



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

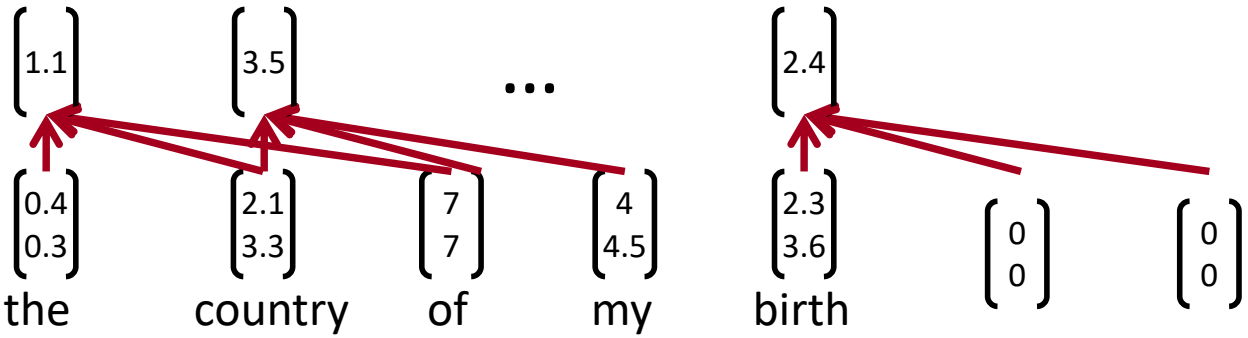
22

Single layer CNN

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

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

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



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 $\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$
- Pooled single number: $\hat{c} = \max\{\mathbf{c}\}$

- Use multiple filter weights **w**
- Useful to have different window sizes *h*
- Because of max pooling $\hat{c} = \max\{\mathbf{c}\}$, length of **c** irrelevant

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}] \in \mathbb{R}^{n-h+1}$$

- So we could have some filters that look at unigrams, bigrams, tri-grams, 4-grams, etc.

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

24

Multi-channel input idea

- Initialize with pre-trained word vectors (word2vec or Glove)
- Start with two copies
- Backprop into only one set, keep other “static”
- Both channel sets are added to c_i before max-pooling

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

25

Classification after one CNN layer

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

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

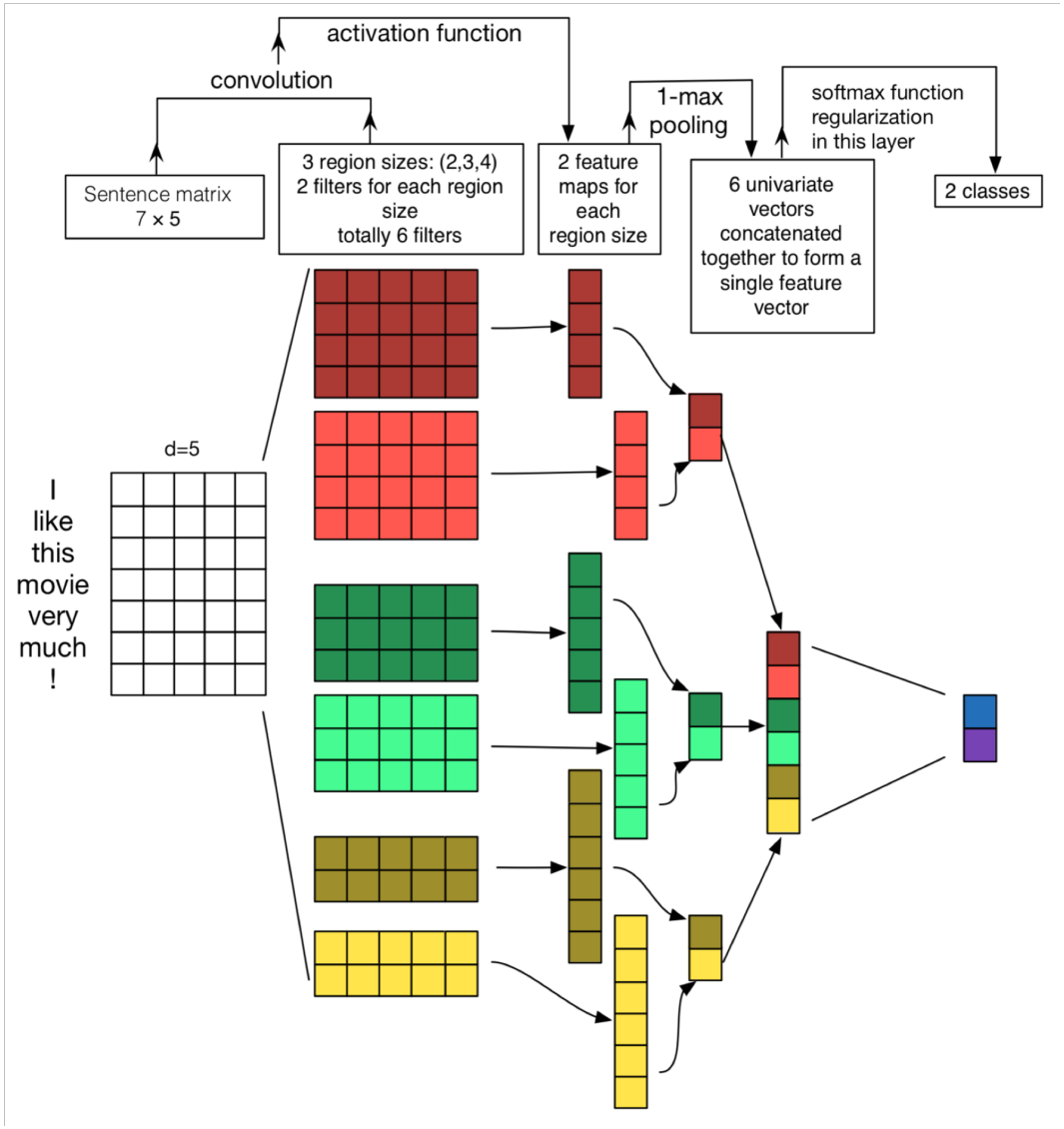
26

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)



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

Regularization

- Use **Dropout**: Create masking vector *r* of Bernoulli random variables with probability *p* (a hyperparameter) of being 1
- Delete features during training:

$$y = softmax\left(W^{(S)}(r \circ z) + b\right)$$

- Reasoning: Prevents co-adaptation (overfitting to seeing specific feature constellations) (Srivastava, Hinton, et al. 2014)
- At test time, no dropout, scale final vector by probability *p*

$$\hat{W}^{(S)} = pW^{(S)}$$

- **Also:** Constrain l_2 norms of weight vectors of each class (row in softmax weight $W^{(S)}$) to fixed number *s* (also a hyperparameter)
- If $\|W_{c\cdot}^{(S)}\| > s$, then rescale it so that: $\|W_{c\cdot}^{(S)}\| = s$
- Not very common

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

28

All hyperparameters in Kim (2014)

- Find hyperparameters based on dev set
- Nonlinearity: ReLU
- Window filter sizes h = 3, 4, 5
- Each filter size has 100 feature maps
- Dropout p = 0.5
  - Kim (2014) reports **2–4%** accuracy improvement from dropout
- L2 constraint *s* for rows of softmax, *s* = 3
- Mini batch size for SGD training: 50
- Word vectors: pre-trained with word2vec, *k* = 300

- During training, keep checking performance on dev set and pick highest accuracy weights for final evaluation

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

29

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	**89.6**
CNN-non-static	**81.5**	48.0	87.2	93.4	93.6	84.3	89.5
CNN-multichannel	81.1	47.4	**88.1**	93.2	92.2	**85.0**	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)	–	**48.7**	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	–	–	**93.6**	–	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	–	–	**93.6**	–	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_S (Silva et al., 2011)	–	–	–	–	**95.0**	–	–

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

30

Content

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