Deep Learning for Natural Language Processing

Cornelia Caragea

Computer Science
University of Illinois at Chicago

Credits for slides: Manning, Socher

Neural Networks and Word Window Classification

Today

Lecture Structure (Part 1):

- Classification review/introduction
- Updating word vectors for classification
- Neural networks introduction
- Word2Vec as neural nets

Lecture Structure (Part 2):

- Named Entity Recognition
- Binary true vs. corrupted word window classification
- Backpropagation

Named Entity Recognition (NER)

- The task: find and classify names in text, for example:

> **The European Commission [ORG] said on Thursday it disagreed with German [MISC] advice.**
>
> **Only France [LOC] and Britain [LOC] backed Fischler [PER] 's proposal .**
>
> **“What we have to be extremely careful of is how other countries are going to take Germany 's lead”, Welsh National Farmers ' Union [ORG] (NFU [ORG]) chairman John Lloyd Jones [PER] said on BBC [ORG] radio .**

- Possible purposes:
  - Tracking mentions of particular entities in documents
  - For question answering, answers are usually named entities
  - A lot of wanted information is really associations between named entities
  - The same techniques can be extended to other slot-filling classifications
- Often followed by Named Entity Linking/Canonicalization into Knowledge Base

Named Entity Recognition on word sequences

We predict entities by classifying words in context and then extracting entities as word subsequences

Foreign	ORG	B-ORG
Ministry	ORG	I-ORG
spokesman	O	O
Shen	PER	B-PER
Guofang	PER	I-PER
told	O	O
Reuters	ORG	B-ORG
that	O	O
:	:	👆 BIO encoding

Why might NER be hard?

- Hard to work out boundaries of entity

  Is the first entity "First National Bank" or "National Bank"

- Hard to know if something is an entity
  Is there a school called "Future School" or is it a future school?
- Hard to know class of unknown/novel entity:

> To find out more about Zig Ziglar and read features by other Creators Syndicate writers and

  What class is "Zig Ziglar"? (A person.)

- Entity class is ambiguous and depends on context "Charles Schwab" is PER not ORG here!

> where Larry Ellison and Charles Schwab can live discreetly amongst wooded estates. And

Binary word window classification

- In general, classifying single words is rarely done
- Interesting problems like ambiguity arise in context!
- Example: auto-antonyms:
  - "To sanction" can mean "to permit" or "to punish"
  - "To seed" can mean "to place seeds" or "to remove seeds"
- Example: resolving linking of ambiguous named entities:
  - Paris $\rightarrow$ Paris, France vs. Paris Hilton vs. Paris, Texas
  - Hathaway $\rightarrow$ Berkshire Hathaway vs. Anne Hathaway

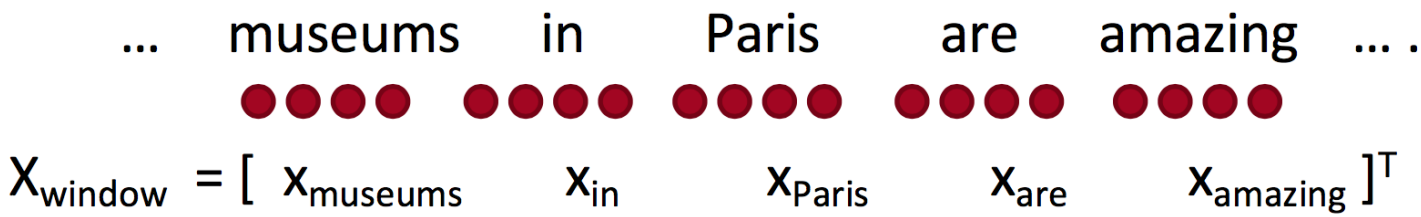
Window classification

- **Idea:** classify a word in its context window of neighboring words.
- For example, **Named Entity Classification** of a word in context
  - Person, Location, Organization, None
- A simple way to classify a word in context might be to average the word vectors in a window and to classify the average vector
  - Problem: that would lose position information

Window classification: Softmax

- Train softmax classifier to classify a center word by taking concatenation of word vectors surrounding it in a window

- Example: Classify "Paris" in the context of this sentence with window length 2:



- Resulting vector $x_{window} = \boxed{x \in \mathbb{R}^{5d}}$, a column vector!

Simplest window classifier: Softmax

With $x = x_{window}$, we can use the same softmax classifier as before

predicted model output probability

$$\boxed{\hat{y}_y} = p(y|x) = \frac{\exp(\boxed{W_{y\cdot}x})}{\sum_{c=1}^{C}\exp(W_{c\cdot}x)}$$

With cross entropy error as before:

same

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

- How do you update the word vectors?
- Short answer: Just take derivatives and optimize

Binary classification with unnormalized scores

Method used by Collobert & Weston (2008, 2011)

- Just recently won ICML 2018 Test of time award
- For our previous example:

$$x_{window} = [\ x_{museums} \quad x_{in} \quad x_{Paris} \quad x_{are} \quad x_{amazing}\ ]$$

- Assume we want to classify whether the center word is a Location
- Similar to word2vec, we will go over all positions in a corpus. But this time, it will be supervised and only some positions should get a high score.
- E.g., the positions that have an actual NER Location in their center are "true" positions and get a high score

Binary classification for NER Location

- Example: Not all museums in Paris are amazing .
- Here: one true window, the one with Paris in its center and all other windows are "corrupt" in terms of not having a named entity location in their center.

  museums in Paris are amazing
- "Corrupt" windows are easy to find and there are many: Any window whose center word is not specifically labeled as NER location in our corpus

  Not all museums in Paris

Neural Network Feed-forward Computation

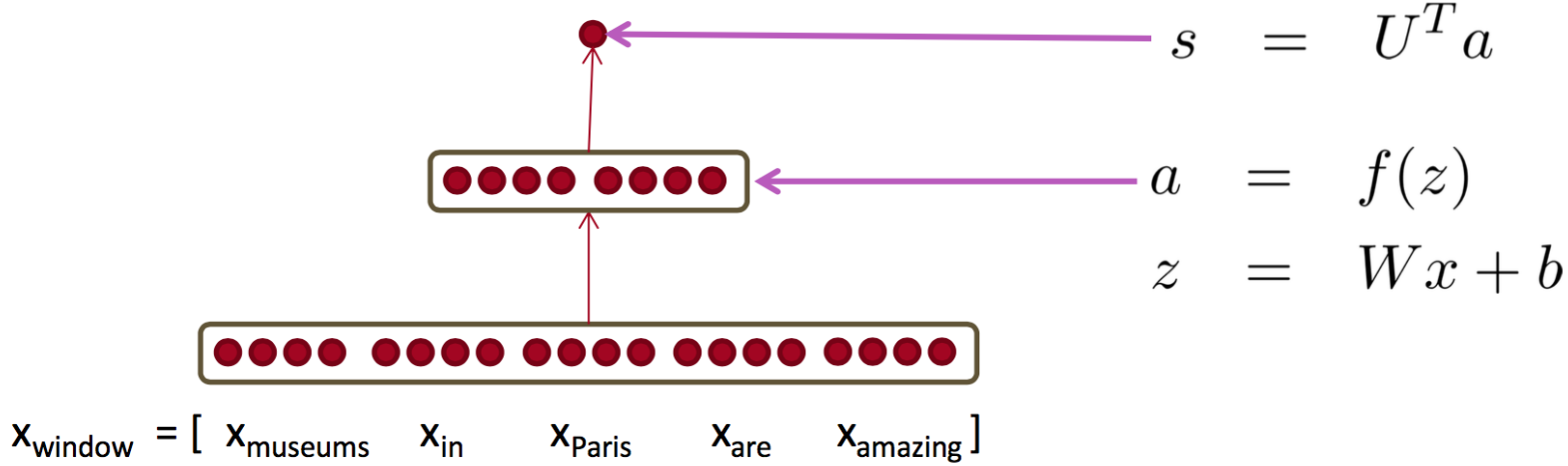
Use neural activation a (or h) simply to give an unnormalized score

$$score(x) = U^T a \in \mathbb{R}$$

We compute a window's score with a 3-layer neural net:

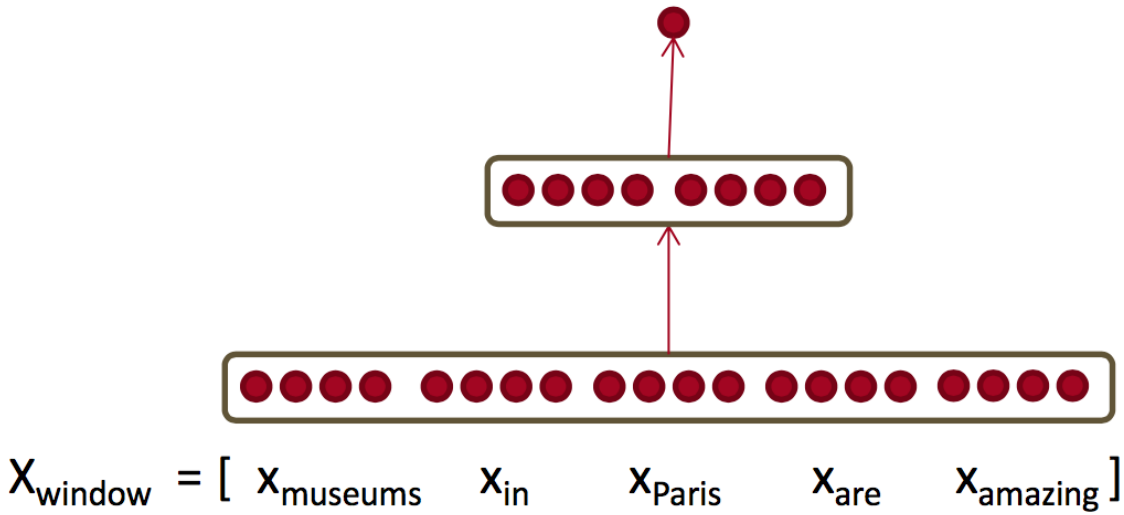
- s = *score*("museums in Paris are amazing")

$s = U^T f(Wx + b)$
$x \in \mathbb{R}^{20\times 1}, W \in \mathbb{R}^{8\times 20}, U \in \mathbb{R}^{8\times 1}$



Main intuition for extra layer

- The middle layer learns non-linear interactions between the input word vectors.



Example: only if "museums" is first vector should it matter that "in" is in the second position

The max-margin loss

- Idea for training objective: Make true window's score larger and corrupt window's score lower (until they're good enough)

- s = score(museums in Paris are amazing)
- s_c = score(Not all museums in Paris)

- Minimize

$$J = \max(0, 1 - s + s_c)$$

- This is continuous $\rightarrow$ we can use SGD.

Max-margin loss

- Objective for a single window:

$$J = \max(0, 1 - s + s_c)$$

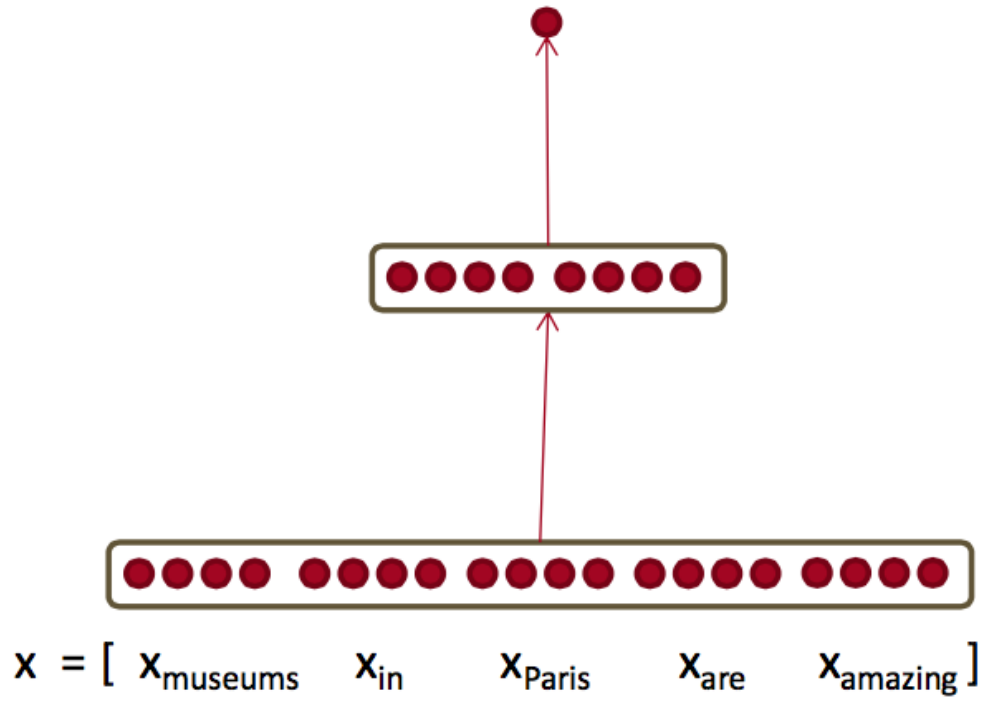
- Each window with an NER location at its center should have a score $+1$ higher than any window without a location at its center
- xxx | $\leftarrow$ 1 $\rightarrow$ | ooo
- For full objective function: Sample several corrupt windows per true one. Sum over all training windows.
- Similar to negative sampling in word2vec

Simple network for score

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{x} \quad \text{(input)}$$



Remember: Stochastic Gradient Descent

- Update equation (in matrix notation):

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

α = step size or learning rate

- How do we compute $\nabla_\theta J(\theta)$
  - The backpropagation algorithm.

Computing Gradients

- Review of multivariable derivatives
- Matrix calculus: Fully vectorized gradients

Gradients

- Given a function with 1 output and 1 input

$$f(x) = x^3$$

- Its gradient (slope) is its derivative

$$\frac{df}{dx} = 3x^2$$

Gradients

- Given a function with 1 output and n inputs

$$f(\mathbf{x}) = f(x_1, x_2, \cdots, x_n)$$

- Its gradient is a vector of partial derivatives with respect to each input

$$\frac{\partial f}{\partial \mathbf{x}} = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \cdots, \frac{\partial f}{\partial x_n}\right]$$

Jacobian Matrix: Generalization of the Gradient

- Given a function with m **outputs** and n **inputs**

$$\boldsymbol{f}(\boldsymbol{x}) = [f_1(x_1, x_2, ..., x_n), ..., f_m(x_1, x_2, ..., x_n)]$$

- Its Jacobian is an $m \times n$ **matrix** of partial derivatives

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\left(\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right)_{ij} = \frac{\partial f_i}{\partial x_j}$$

Chain Rule

- For one-variable functions: multiply derivatives

$$z = 3y$$

$$y = x^2$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} = (3)(2x) = 6x$$

- For multiple variables at once: multiply Jacobians

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = \ldots$$

Example Jacobian: Elementwise activation Function

$\boldsymbol{h} = f(\boldsymbol{z})$, what is $\dfrac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}$? $\qquad\qquad$ $\boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$

$h_i = f(z_i)$

Example Jacobian: Elementwise activation Function

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$

$$h_i = f(z_i)$$

- Function has n outputs and n inputs $\rightarrow$ n by n Jacobian

Example Jacobian: Elementwise activation Function

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$

$$h_i = f(z_i)$$

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \qquad \text{definition of Jacobian}$$

$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \qquad \text{regular 1-variable derivative}$$

Example Jacobian: Elementwise activation Function

$$\boldsymbol{h} = f(\boldsymbol{z}), \text{what is } \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}? \qquad \boldsymbol{h}, \boldsymbol{z} \in \mathbb{R}^n$$

$$h_i = f(z_i)$$

$$\left(\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}}\right)_{ij} = \frac{\partial h_i}{\partial z_j} = \frac{\partial}{\partial z_j} f(z_i) \qquad \text{definition of Jacobian}$$

$$= \begin{cases} f'(z_i) & \text{if } i = j \\ 0 & \text{if otherwise} \end{cases} \qquad \text{regular 1-variable derivative}$$

$$\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \begin{pmatrix} f'(z_1) & & 0 \\ & \ddots & \\ 0 & & f'(z_n) \end{pmatrix} = \text{diag}(\boldsymbol{f}'(\boldsymbol{z}))$$

Other Jacobians

$$\frac{\partial}{\partial \boldsymbol{x}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{W}$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x}+\boldsymbol{b}) = \boldsymbol{I} \text{ (Identity matrix)}$$

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T\boldsymbol{h}) = \boldsymbol{h}^{\boldsymbol{T}}$$

- Compute these at home for practice!

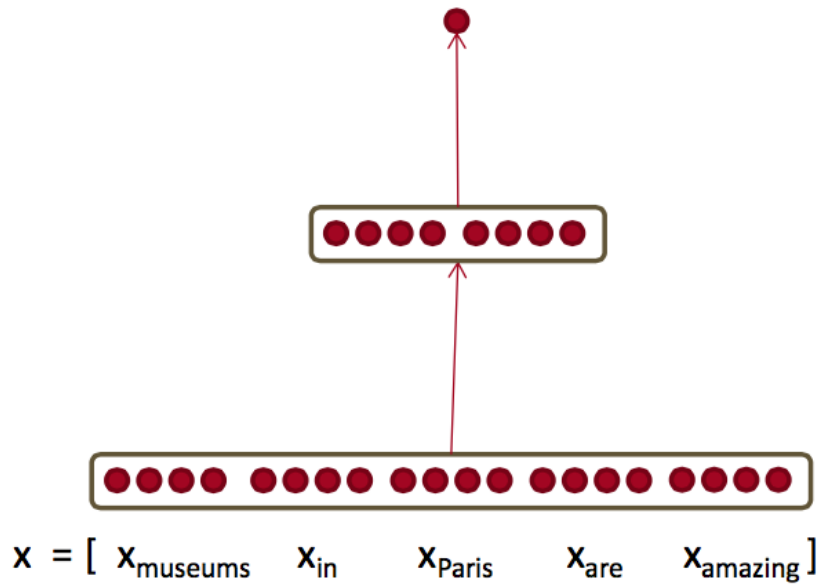
Back to our Neural Net!

- Let's find $\frac{\partial s}{\partial b}$
  - In practice we care about the gradient of the loss, but we will compute the gradient of the score for simplicity

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{x} \quad \text{(input)}$$



Break up equations into simple pieces

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{x} \quad \text{(input)}$$

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad \text{(input)}$$

Apply the chain rule

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad (\text{input})$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

Apply the chain rule

$$\boxed{s = \boldsymbol{u}^T \boldsymbol{h}}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \boxed{\frac{\partial s}{\partial \boldsymbol{h}}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

Apply the chain rule

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad (\text{input})$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

Apply the chain rule

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

Write out the Jacobians

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \quad \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$
$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \text{diag}(f'(\boldsymbol{z}))$$
$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

Write out the Jacobians

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

$$\downarrow$$

$$\boldsymbol{u}^T$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$

$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \text{diag}(f'(\boldsymbol{z}))$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

Write out the Jacobians

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

$$\boldsymbol{u}^T \operatorname{diag}(f'(\boldsymbol{z}))$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$

$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \operatorname{diag}(f'(\boldsymbol{z}))$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

Write out the Jacobians

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{z})$$

$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

$$= \boldsymbol{u}^T \operatorname{diag}(f'(\boldsymbol{z}))\boldsymbol{I}$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$

$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \operatorname{diag}(f'(\boldsymbol{z}))$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

Write out the Jacobians

$$s = \boldsymbol{u}^T \boldsymbol{h}$$
$$\boldsymbol{h} = f(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$
$$\boldsymbol{x} \quad \text{(input)}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

$$= \boldsymbol{u}^T \operatorname{diag}(f'(\boldsymbol{z}))\boldsymbol{I}$$

$$= \boldsymbol{u}^T \circ f'(\boldsymbol{z})$$

Useful Jacobians from previous slide

$$\frac{\partial}{\partial \boldsymbol{u}}(\boldsymbol{u}^T \boldsymbol{h}) = \boldsymbol{h}^T$$

$$\frac{\partial}{\partial \boldsymbol{z}}(f(\boldsymbol{z})) = \operatorname{diag}(f'(\boldsymbol{z}))$$

$$\frac{\partial}{\partial \boldsymbol{b}}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \boldsymbol{I}$$

Re-using Computation

- Suppose we now want to compute $\frac{\partial s}{\partial W}$
  - Using the chain rule again:

$$\frac{\partial s}{\partial \boldsymbol{W}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

Re-using Computation

- Suppose we now want to compute $\frac{\partial s}{\partial W}$
  - Using the chain rule again:

$$\frac{\partial s}{\partial \boldsymbol{W}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

The blue part is the same as before! Let's avoid duplicated computation...

Re-using Computation

- Suppose we now want to compute $\frac{\partial s}{\partial W}$
  - Using the chain rule again:

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$$

$$\frac{\partial s}{\partial \boldsymbol{b}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} = \boldsymbol{\delta}$$

$$\boldsymbol{\delta} = \frac{\partial s}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}} = \boldsymbol{u}^T \circ f'(\boldsymbol{z})$$

δ is local error signal

Derivative with respect to Matrix: Output shape

- What does $\frac{\partial s}{\partial W}$ look like? $W \in \mathbb{R}^{n \times m}$
- 1 output, nm inputs: 1 by nm Jacobian?
  - Inconvenient to do $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$

Derivative with respect to Matrix: Output shape

- What does $\frac{\partial s}{\partial W}$ look like? $W \in \mathbb{R}^{n \times m}$
- 1 output, nm inputs: 1 by nm Jacobian?
  - Inconvenient to do $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$
- Instead follow convention: shape of the gradient is shape of parameters
  - So $\frac{\partial s}{\partial W}$ is n by m:

$$\begin{bmatrix} \frac{\partial s}{\partial W_{11}} & \cdots & \frac{\partial s}{\partial W_{1m}} \\ \vdots & \ddots & \vdots \\ \frac{\partial s}{\partial W_{n1}} & \cdots & \frac{\partial s}{\partial W_{nm}} \end{bmatrix}$$

Derivative with respect to Matrix

Remember $\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}$

- $\boldsymbol{\delta}$ is going to be in our answer
- The other term should be $\boldsymbol{x}$ because $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$

It turns out $\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \boldsymbol{x}^T$

δ is local error signal at z
x is local input signal

Why the Transposes? This makes the dimensions work out!

$$\frac{\partial s}{\partial \boldsymbol{W}} = \boldsymbol{\delta}^T \quad \boldsymbol{x}^T$$
$$[n \times m] \quad [n \times 1][1 \times m]$$

Backpropagation

Backpropagation

- Computing gradients algorithmically and efficiently
- Converting what we just did by hand into an algorithm
- Used by deep learning software frameworks (TensorFlow, PyTorch, etc.)

Training with Backpropagation

$$J = \max(0, 1 - s + s_c)$$

$$s = U^T f(Wx + b)$$
$$s_c = U^T f(Wx_c + b)$$

Assuming cost *J* is > 0,

compute the derivatives of *s* and s_c wrt all the involved variables: *U, W, b, x*

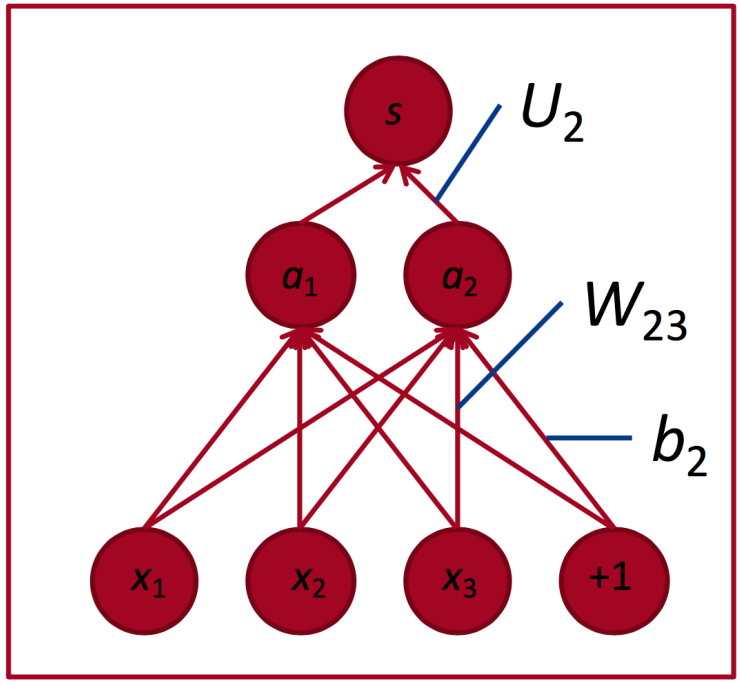
$$\frac{\partial s}{\partial U} = \frac{\partial}{\partial U} U^T a \qquad \frac{\partial s}{\partial U} = a$$

Training with Backpropagation

- Let's consider the derivative of a single weight W_{ij}

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W} U^T a = \frac{\partial}{\partial W} U^T f(z) = \frac{\partial}{\partial W} U^T f(Wx + b)$$

- This only appears inside a_i
- For example: W_{23} is only used to compute a_2



Training with Backpropagation

$$\frac{\partial s}{\partial W} = \frac{\partial}{\partial W} U^T a = \frac{\partial}{\partial W} U^T f(z) = \frac{\partial}{\partial W} U^T f(Wx + b)$$

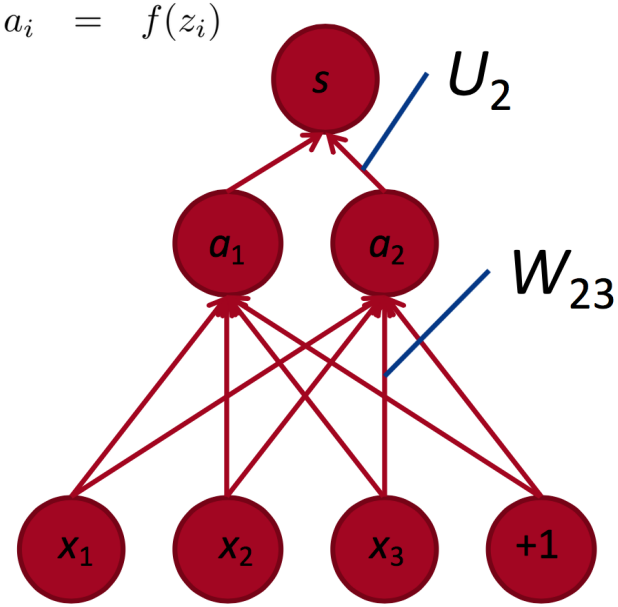
Derivative of weight W_{ij}:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$$

$$\frac{\partial}{\partial W_{ij}} U^T a \quad \rightarrow \quad \frac{\partial}{\partial W_{ij}} U_i a_i$$

$$\begin{aligned} U_i \frac{\partial}{\partial W_{ij}} a_i &= U_i \frac{\partial a_i}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i \frac{\partial f(z_i)}{\partial z_i} \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial z_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial W_{i.} x + b_i}{\partial W_{ij}} \end{aligned}$$

$$\begin{aligned} z_i &= W_{i.} x + b_i = \sum_{j=1}^{3} W_{ij} x_j + b_i \\ a_i &= f(z_i) \end{aligned}$$



Training with Backpropagation

Derivative of single weight W_{ij} :

$$z_i = W_i.x + b_i = \sum_{j=1}^{3} W_{ij}x_j + b_i$$

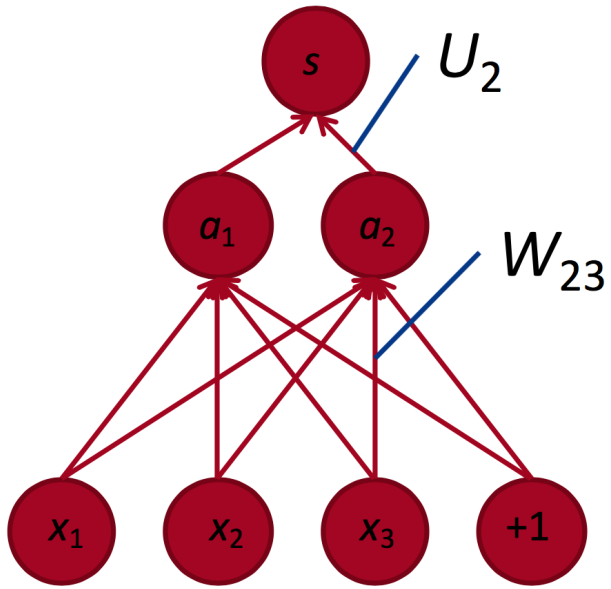
$$a_i = f(z_i)$$

$$\begin{aligned} U_i \frac{\partial}{\partial W_{ij}} a_i &= U_i f'(z_i) \frac{\partial W_i.x + b_i}{\partial W_{ij}} \\ &= U_i f'(z_i) \frac{\partial}{\partial W_{ij}} \sum_k W_{ik}x_k \\ &= \underbrace{U_i f'(z_i)}\, x_j \\ &= \delta_i \; x_j \end{aligned}$$

Local error signal

Local input signal

where $f'(z) = f(z)(1 - f(z))$ for logistic f



Training with Backpropagation

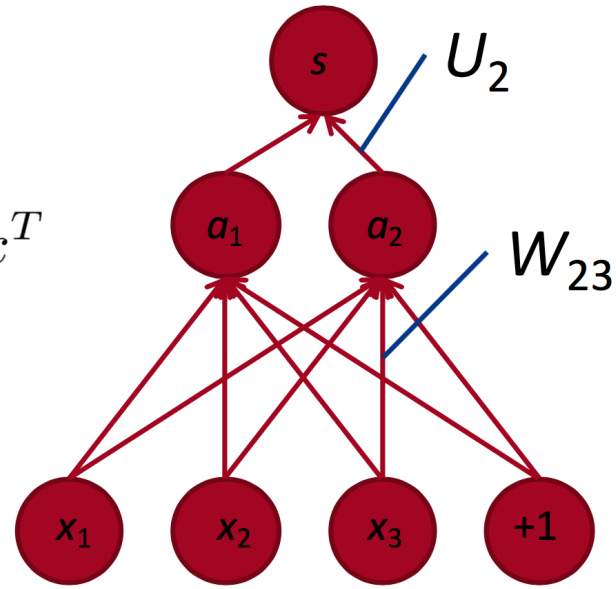
- From single weight W_{ij} to full W:

$$\frac{\partial s}{\partial W_{ij}} = \underbrace{U_i f'(z_i)}_{\delta_i} x_j$$

$$z_i = W_{i\cdot}x + b_i = \sum_{j=1}^{3} W_{ij}x_j + b_i$$

$$a_i = f(z_i)$$

- We want all combinations of $i = 1, 2$ and $j = 1, 2, 3$ → ?
- Solution: Outer product: $\frac{\partial J}{\partial W} = \delta x^T$ where $\delta \in \mathbb{R}^{2\times 1}$ is the "responsibility" or error message coming from each activation a

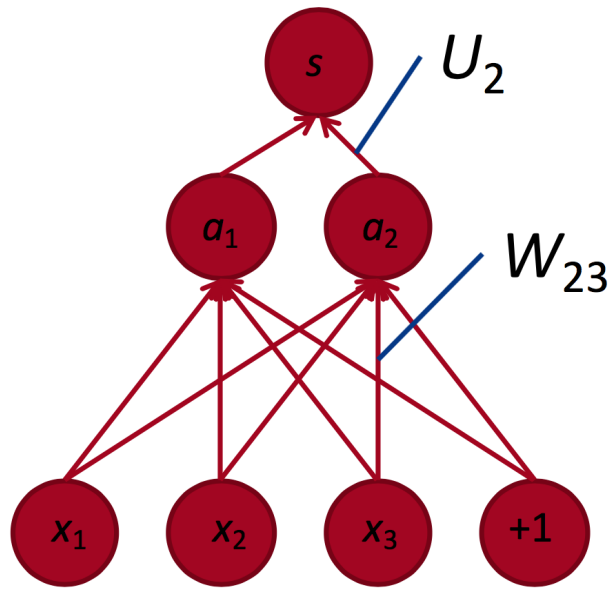


Training with Backpropagation

- For biases *b*, we get:

$$\begin{aligned} z_i &= W_i.x + b_i = \sum_{j=1}^{3} W_{ij}x_j + b_i \\ a_i &= f(z_i) \end{aligned}$$

$$\begin{aligned} & U_i \frac{\partial}{\partial b_i} a_i \\ = \; & U_i f'(z_i) \frac{\partial W_i.x + b_i}{\partial b_i} \\ = \; & \delta_i \end{aligned}$$



Training with Backpropagation

That's almost backpropagation

It's simply taking derivatives and using the chain rule!

Remaining trick: we can **re-use** derivatives computed for higher layers in computing derivatives for lower layers!

Example: last derivatives of model, the word vectors in x

Training with Backpropagation

- Take derivative of score with respect to single element of word vector
- Now, we cannot just take into consideration one a_i because each x_j is connected to all the neurons above and hence x_j influences the overall score through all of these, hence:

$$
\begin{aligned}
\frac{\partial s}{\partial x_j} &= \sum_{i=1}^{2} \frac{\partial s}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{2} \frac{\partial U^T a}{\partial a_i} \frac{\partial a_i}{\partial x_j} \\
&= \sum_{i=1}^{2} U_i \frac{\partial f(W_i.x + b)}{\partial x_j} \\
&= \sum_{i=1}^{2} \underbrace{U_i f'(W_i.x + b)}_{\delta} \frac{\partial W_i.x}{\partial x_j} \\
&= \sum_{i=1}^{2} \delta_i W_{ij} \\
&= W_{.j}^T \delta
\end{aligned}
$$

Re-used part of previous derivative

Training with Backpropagation

- With $\frac{\partial s}{\partial x_j} = W_{\cdot j}^T \delta$,what is the full gradient? →

$$\frac{\partial s}{\partial x} = W^T \delta$$

- Observations: The error message $\pm$ that arrives at a hidden layer has the same dimensionality as that hidden layer

Putting all gradients together:

- Remember: Full objective function for each window was:

$$J = \max(0, 1 - s + s_c)$$

$$s = U^T f(Wx + b)$$
$$s_c = U^T f(Wx_c + b)$$

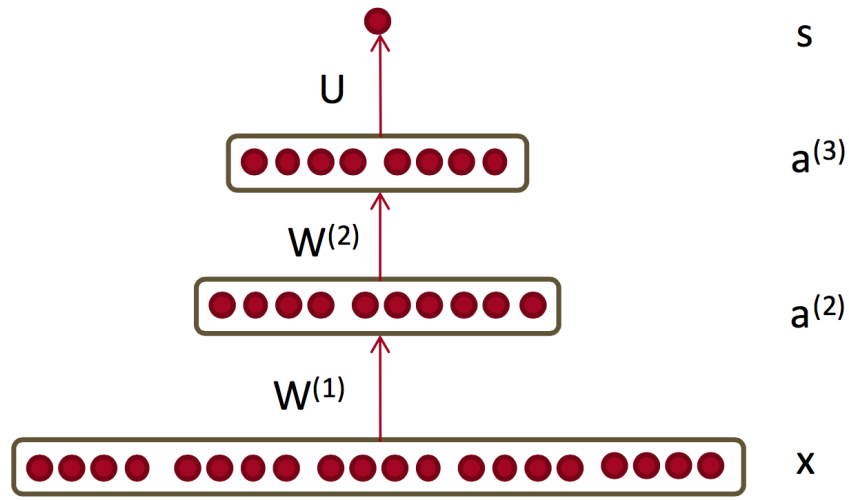
- For example: gradient for U:

$$\frac{\partial s}{\partial U} = 1\{1 - s + s_c > 0\} \left(-f(Wx + b) + f(Wx_c + b)\right)$$
$$\frac{\partial s}{\partial U} = 1\{1 - s + s_c > 0\} \left(-a + a_c\right)$$

Two layer neural nets and full backprop

- Let's look at a 2 layer neural network
- Same window definition for x
- Same scoring function
- 2 hidden layers (carefully not superscripts now!)

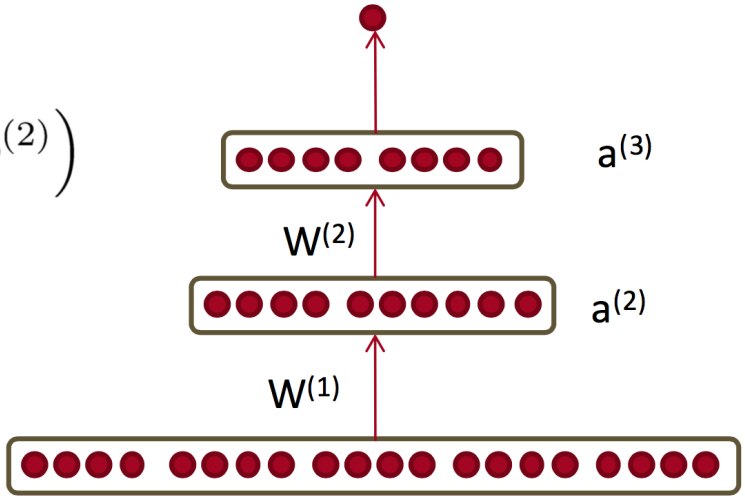
$$
\begin{aligned}
x &= z^{(1)} = a^{(1)} \\
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f\left(z^{(2)}\right) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
a^{(3)} &= f\left(z^{(3)}\right) \\
s &= U^T a^{(3)}
\end{aligned}
$$



Two layer neural nets and full backprop

- Fully written out as one function:

$$\begin{aligned} s &= U^T f\left(W^{(2)} f\left(W^{(1)} x + b^{(1)}\right) + b^{(2)}\right) \\ &= U^T f\left(W^{(2)} a^{(2)} + b^{(2)}\right) \\ &= U^T a^{(3)} \end{aligned}$$



- Same derivation as before for $W^{(2)}$ (now sitting on $a^{(1)}$)

$$\begin{aligned} \frac{\partial\, s}{\partial W_{ij}} &= \underbrace{U_i f'(z_i)}\, x_j \\ &= \quad \delta_i \quad\; x_j \end{aligned} \qquad \begin{aligned} \frac{\partial s}{\partial W^{(2)}_{ij}} &= \underbrace{U_i f'\left(z_i^{(3)}\right)}\, a_j^{(1)} \\ &= \quad \delta_i^{(3)} \quad\; a_j^{(2)} \end{aligned}$$

Two layer neural nets and full backprop

- Same derivation as before for top $W^{(2)}$:

$$\begin{aligned}\frac{\partial s}{\partial W_{ij}^{(2)}} &= \underbrace{U_i f'\left(z_i^{(3)}\right)}_{\delta_i^{(3)}} a_j^{(1)} \\ &= \delta_i^{(3)} \quad a_j^{(2)}\end{aligned}$$

$$\begin{aligned}x &= z^{(1)} = a^{(1)} \\ z^{(2)} &= W^{(1)}x + b^{(1)} \\ a^{(2)} &= f\left(z^{(2)}\right) \\ z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\ a^{(3)} &= f\left(z^{(3)}\right) \\ s &= U^T a^{(3)}\end{aligned}$$

- In matrix notation: $\frac{\partial s}{\partial W^{(2)}} = \delta^{(3)} a^{(2)T}$

where $\delta^{(3)} = U \circ f'\left(z^{(3)}\right)$ and $\pm$ is the element-wise product
also called Hadamard product

- Last missing piece for understanding general backprop: $\frac{\partial s}{\partial W^{(1)}}$

Two layer neural nets and full backprop

- Last missing piece: $\frac{\partial s}{\partial W^{(1)}}$

- What's the bottom layer's error message $\pm^{(2)}$?

$$
\begin{aligned}
x &= z^{(1)} = a^{(1)} \\
z^{(2)} &= W^{(1)} x + b^{(1)} \\
a^{(2)} &= f\left(z^{(2)}\right) \\
z^{(3)} &= W^{(2)} a^{(2)} + b^{(2)} \\
a^{(3)} &= f\left(z^{(3)}\right) \\
s &= U^T a^{(3)}
\end{aligned}
$$

- Similar derivation to single layer model

- Main difference, we already have ${W^{(2)}}^T \delta^{(3)}$ and need to apply the chain rule again on $f'(z^{(2)})$

Two layer neural nets and full backprop

- Chain rule for: $s = U^T f\left(W^{(2)} f\left(W^{(1)} x + b^{(1)}\right) + b^{(2)}\right)$
- Get intuition by deriving $\frac{\partial s}{\partial W^{(1)}}$ as if it was a scalar
- Intuitively, we have to sum over all the nodes coming into layer
- Putting it all together: $\delta^{(2)} = \left(W^{(2)^T} \delta^{(3)}\right) \circ f'\left(z^{(2)}\right)$

Two layer neural nets and full backprop

- Last missing piece: $\frac{\partial s}{\partial W^{(1)}} = \delta^{(2)} x^T$

- In general for any matrix $W^{(l)}$ at internal layer l and any error with regularization E_R all backprop in standard multilayer neural networks boils down to 2 equations:

$$
\begin{aligned}
x &= z^{(1)} = a^{(1)} \\
z^{(2)} &= W^{(1)} x + b^{(1)} \\
a^{(2)} &= f\left(z^{(2)}\right) \\
z^{(3)} &= W^{(2)} a^{(2)} + b^{(2)} \\
a^{(3)} &= f\left(z^{(3)}\right) \\
s &= U^T a^{(3)}
\end{aligned}
$$

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \circ f'(z^{(l)}),$$

$$\frac{\partial}{\partial W^{(l)}} E_R = \delta^{(l+1)} (a^{(l)})^T + \lambda W^{(l)}$$

- Top and bottom layers have simpler ±

Backpropagation (Another explanation)

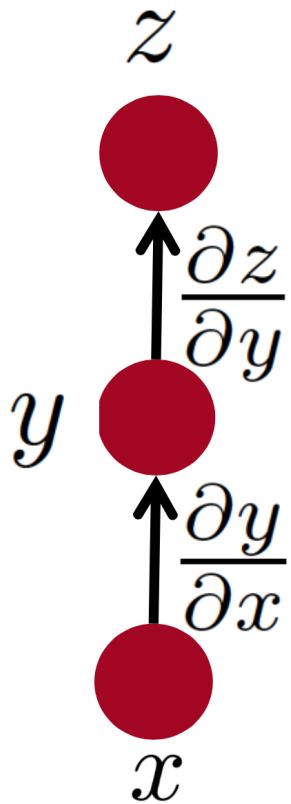
- Compute gradient of example-wise loss wrt parameters

- Simply applying the derivative chain rule wisely

$$z = f(y) \quad y = g(x) \quad \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

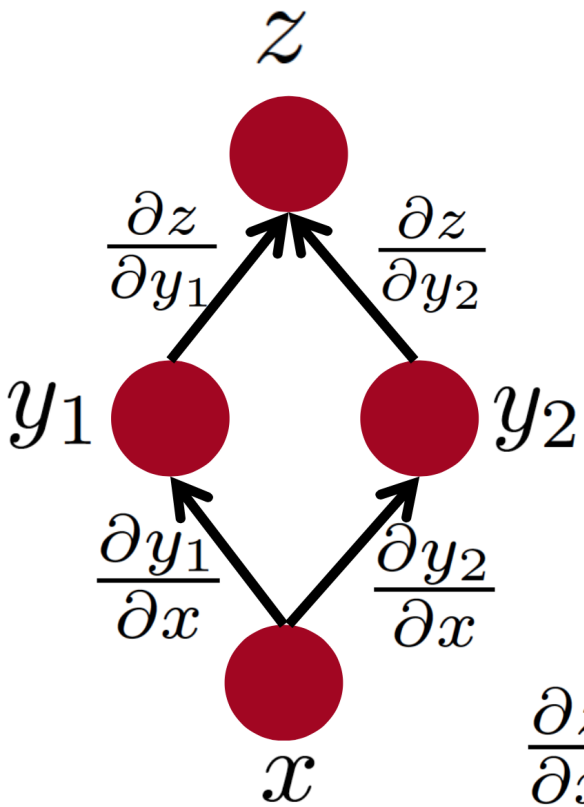
- If computing the loss(example, parameters) is $O(n)$ computation, then so is computing the gradient

Simple Chain Rule



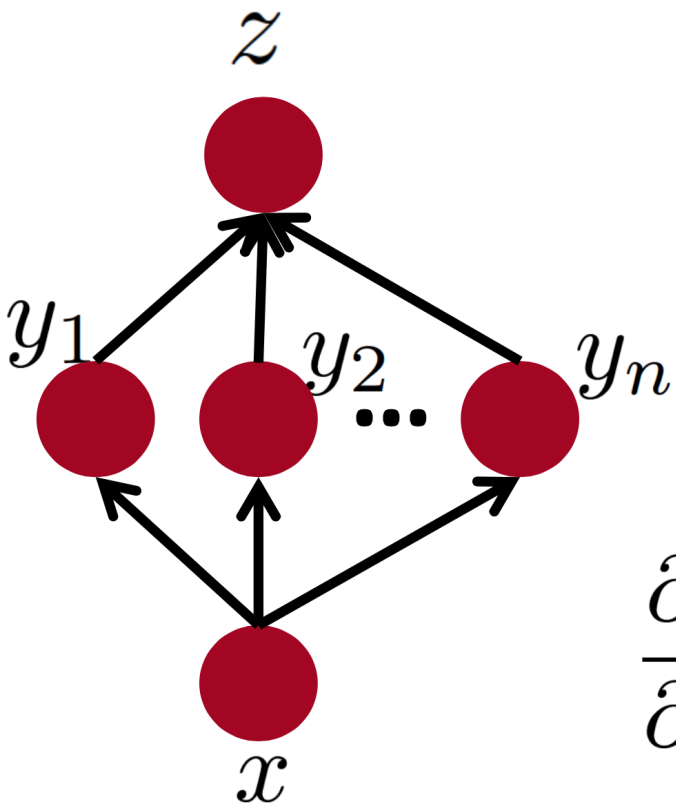
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

Multiple Paths Chain Rule



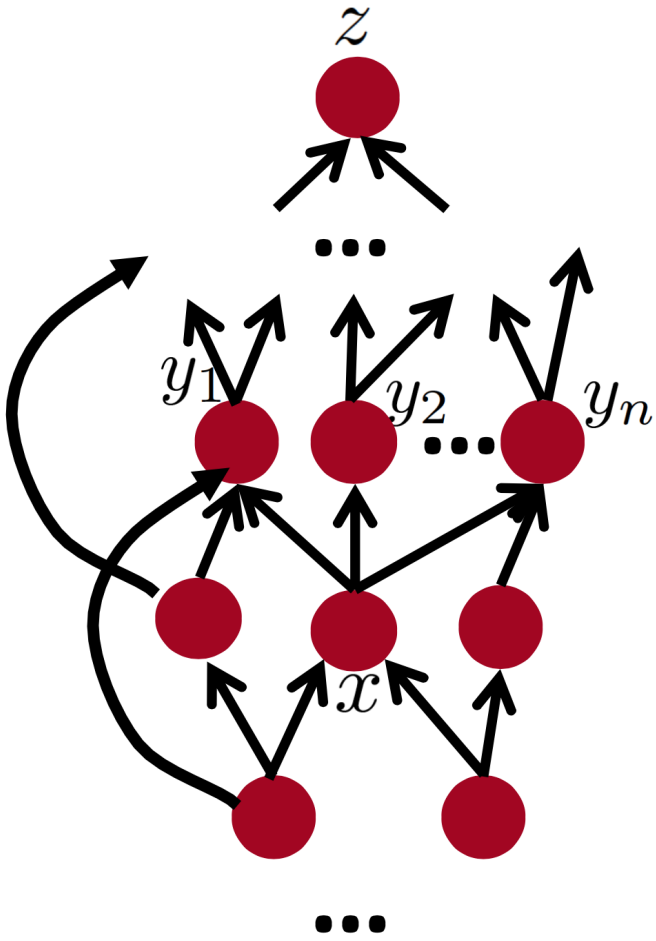
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

Multiple Paths Chain Rule - General



$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Chain Rule in Flow Graph

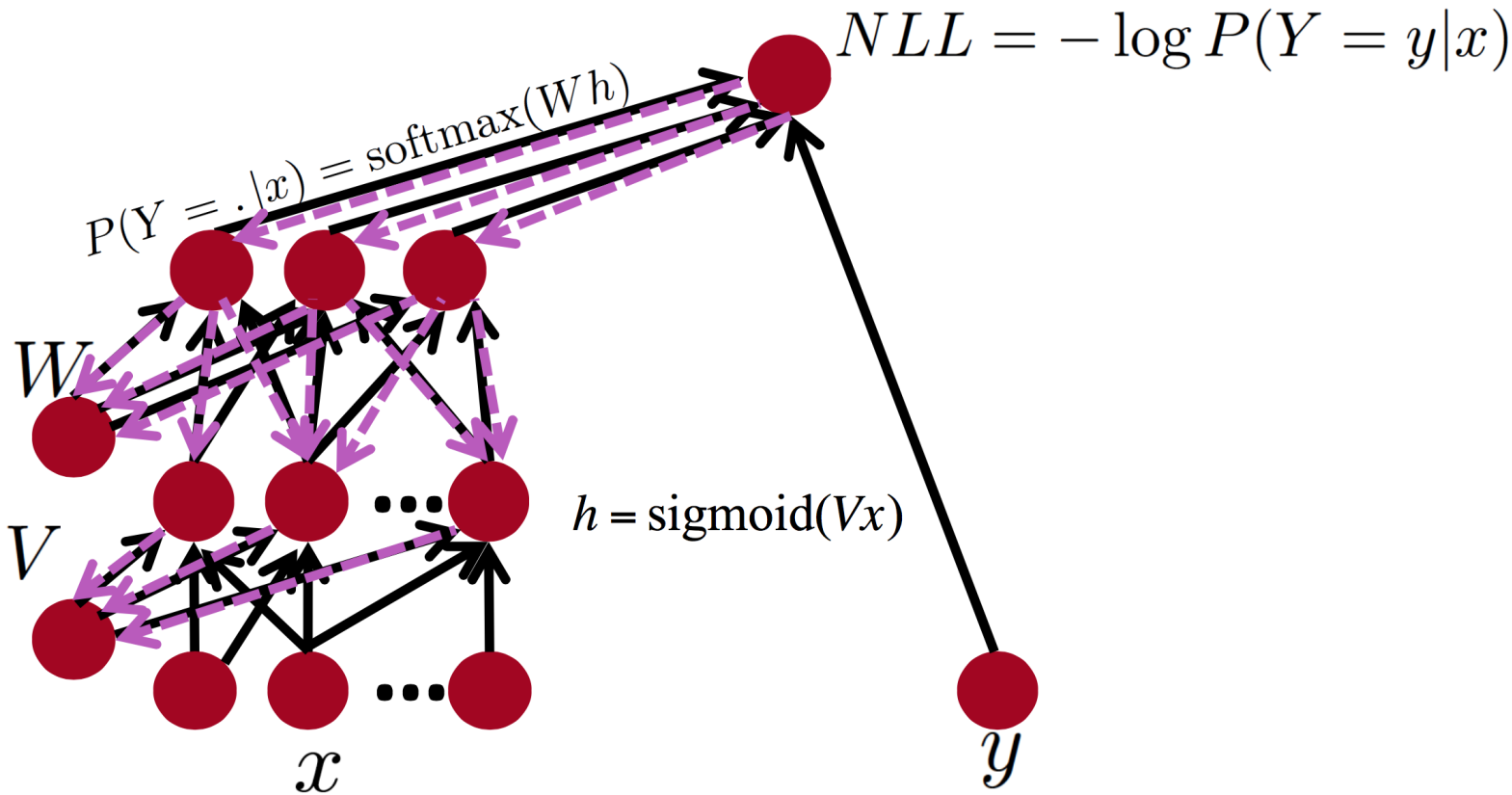


Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

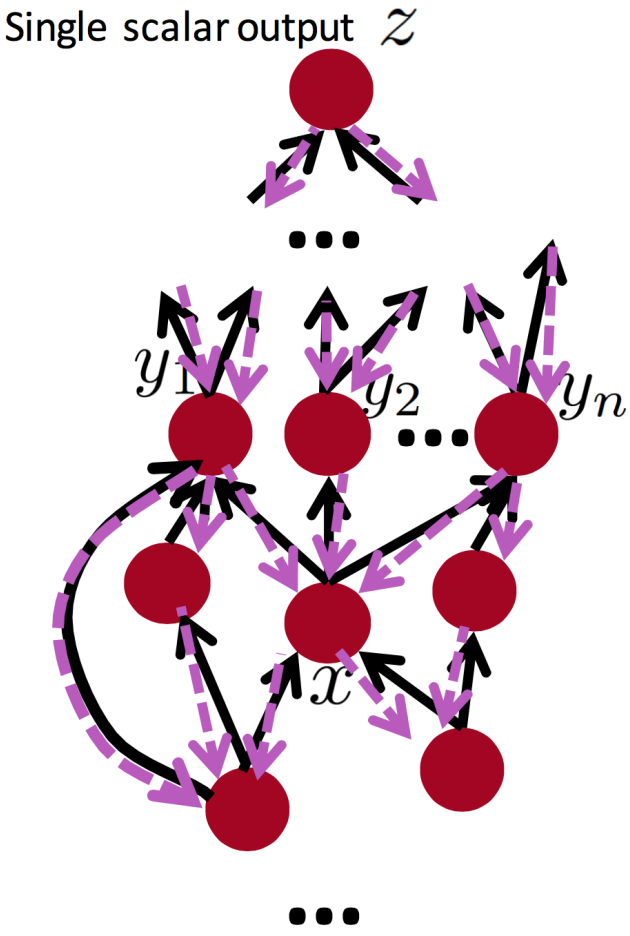
$\{y_1, y_2, \ldots y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Back-Prop in Multi-Layer Net



Back-Prop in General Flow Graph

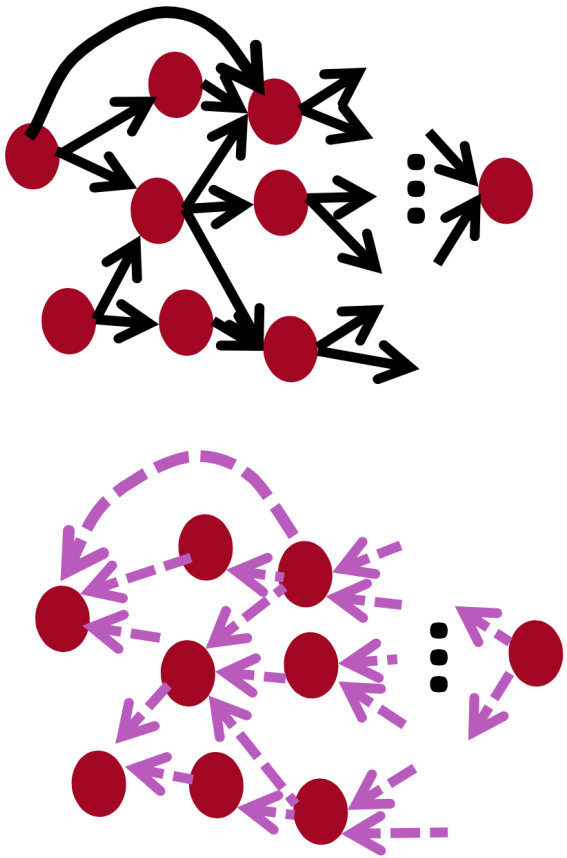


1. Fprop: visit nodes in topo-sort order
   - Compute value of node given predecessors
2. Bprop:
   - initialize output gradient = 1
   - visit nodes in reverse order:
     Compute gradient wrt each node using gradient wrt successors

$\{y_1, y_2, \ldots y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Automatic Differentiation



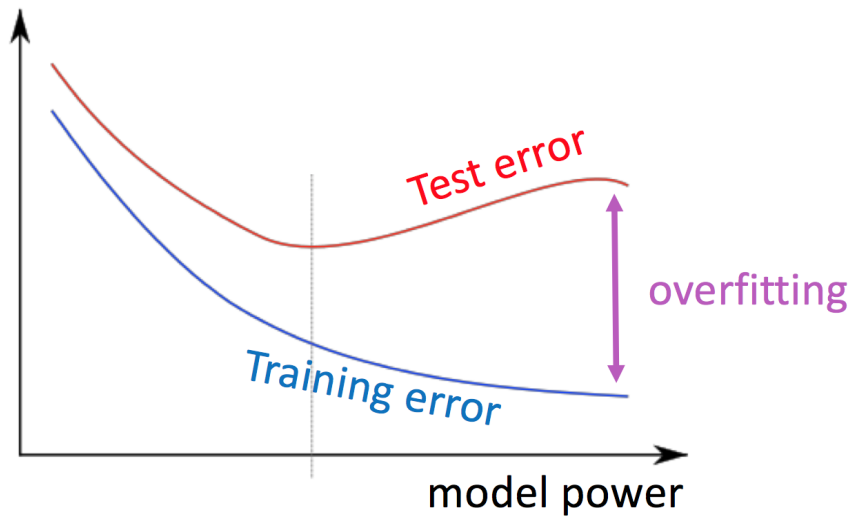
- The gradient computation can be **automatically inferred** from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.
- Easy and fast prototyping

We have models with many params! Regularization!

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

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

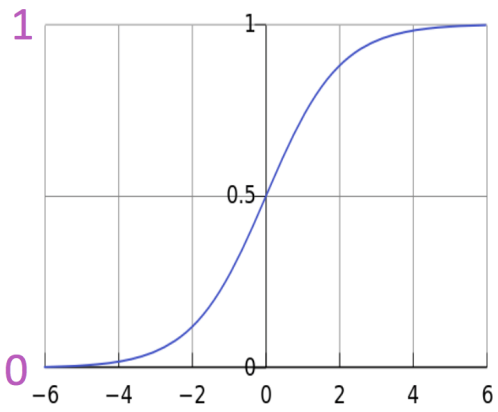
- Regularization (largely) prevents overfitting when we have a lot of features (or later a very powerful/deep model, ++)



Non-linearities: The starting points

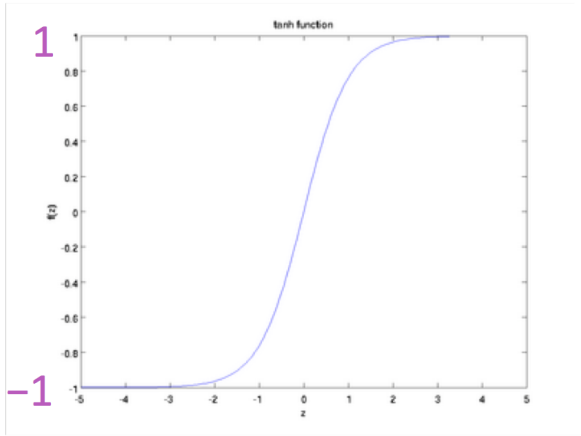
logistic ("sigmoid")

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



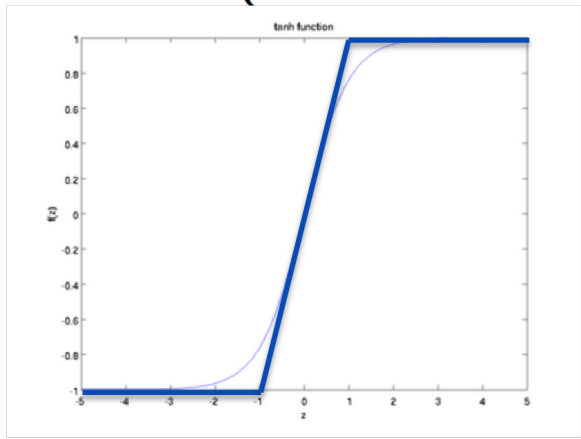
tanh

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



hard tanh

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



tanh is just a rescaled and shifted sigmoid (2 × 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

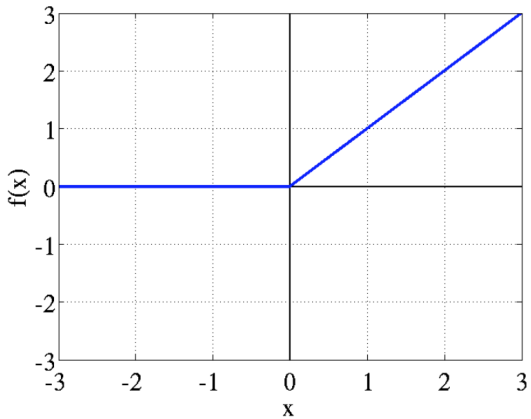
Non-linearities: The new world order

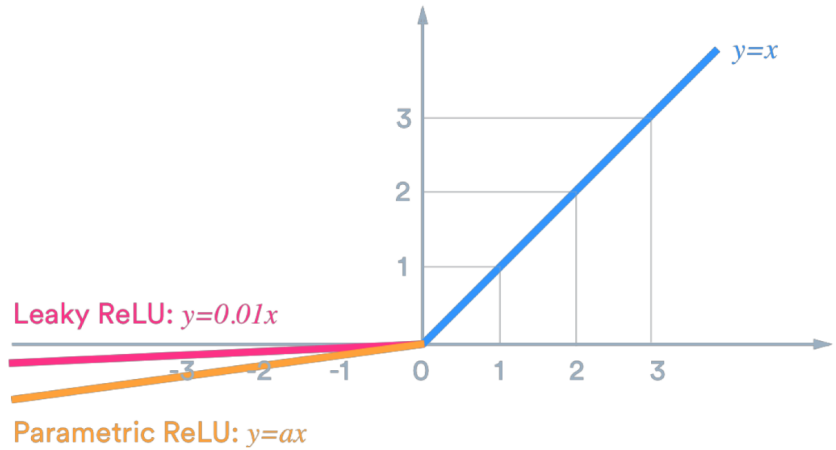
ReLU (rectified linear unit) hard tanh

Leaky ReLU

Parametric ReLU

$$\text{rect}(z) = \max(z, 0)$$





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