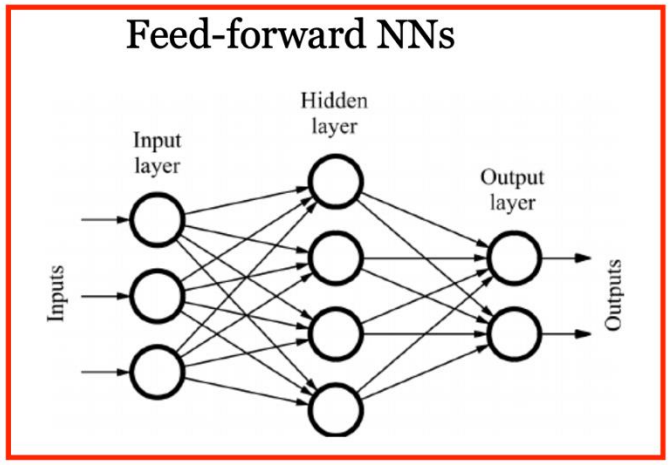
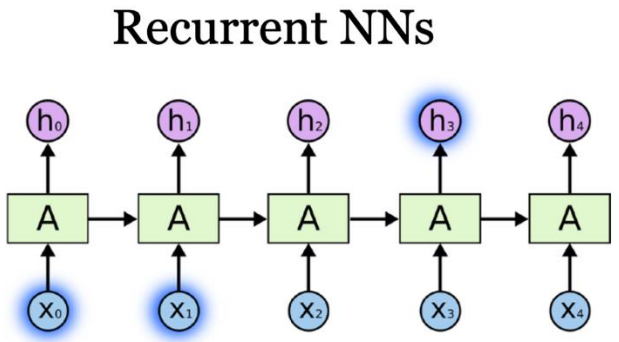
CSE 4392 Special Topics
Natural Language Processing

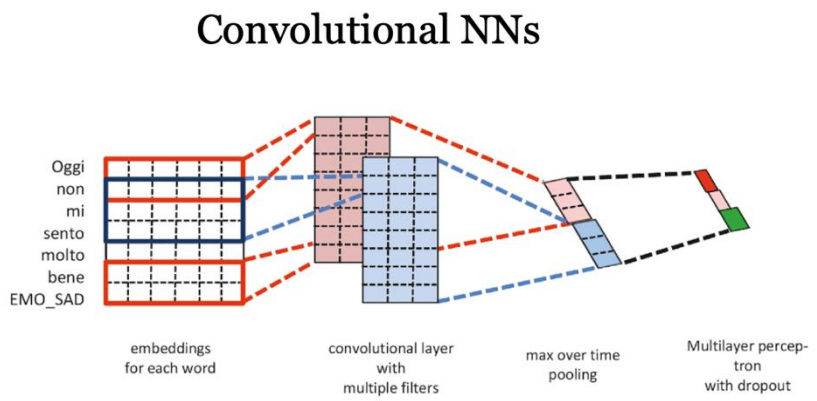
Neural Network Basics

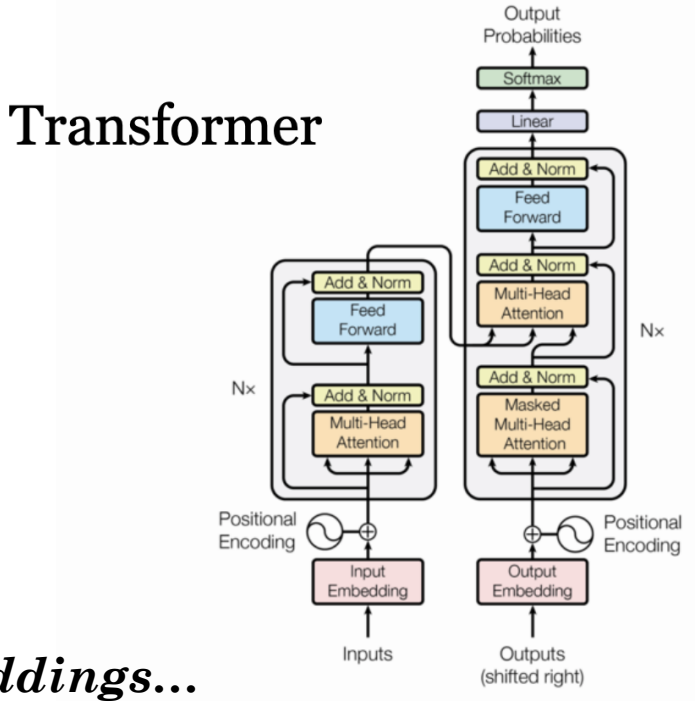
2024 Spring

Neural Networks for NLP









Always coupled with word embeddings…

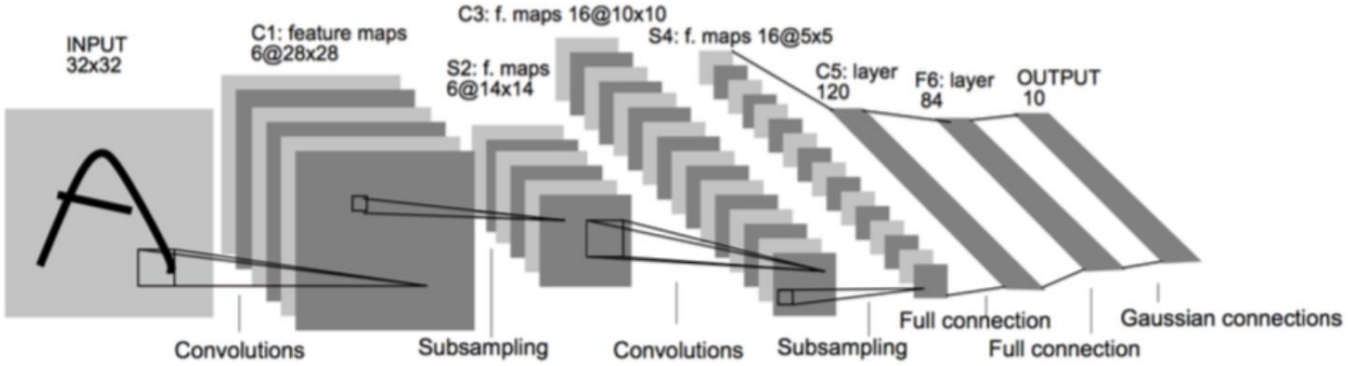
In This Lecture

- Feedforward Neural Networks
- Applications
  - Neural Bag-of-Words Models
  - Feedforward Neural Language Models
- The training algorithm: Back-propagation

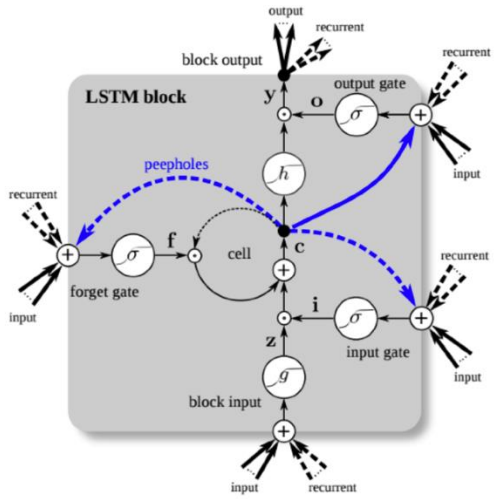
The History of Neural Networks

NN "Dark Ages"

- Neural network algorithms data back to 1980s
- ConvNets: Applied to MNIST by Yann LeCun in 1998

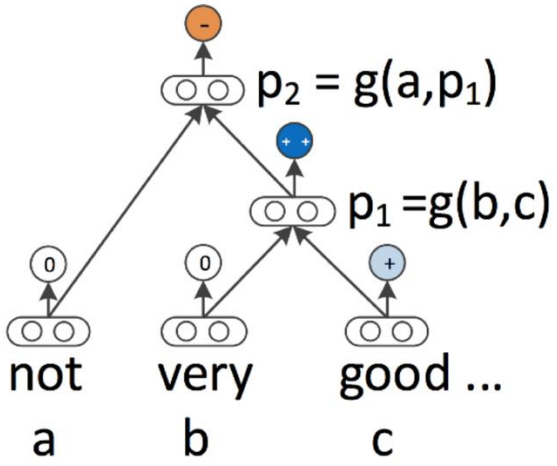


- Long Short-term Memory Networks (LSTMs): Hochreiter and Schmidhuber 1997
- Henderson 2003: neural shift-reduce parser, not SOTA



2008-2013: A Glimmer of Light

- Collobert and Weston 2011: "**NLP (almost) from Scratch**"
  - Feedforward NNs can replace "feature engineering"
  - 2008 version was marred by bad experiments, claimed SOTA but wasn't, 2011 version tied SOTA
- Krizhevskey et al, 2012: AlexNet for ImageNet Classification
- Socher 2011-2014: tree-structured RNNs working okay



2014: Things start to work

- Kim (2014) + Kalchbrenner et al, 2014: sentence classification
  - ConvNets work for NLP!
- Sutskever et al, 2014: sequence-to-sequence for neural MT
  - LSTMs work for NLP!
- Chen and Manning 2014: dependency parsing
  - Even feedforward networks work well for NLP!
- 2015: explosion of neural networks for everything under the sun

Why Didn't They Work Before?

- **Datasets too small**: for MT, not really better until you have 1M+ parallel sentences (and really need a lot more)
- **Optimization not well understood**: good initialization, per-feature scaling + momentum (Adagrad/Adam) work best out-of-the-box
  - Regularization: dropout is pretty helpful
  - Computers not big enough: can't run for enough iterations
- Inputs: need **word embeddings** to represent continuous semantics

The "Promise"

- Most NLP works in the past focused on human-designed representations and input features

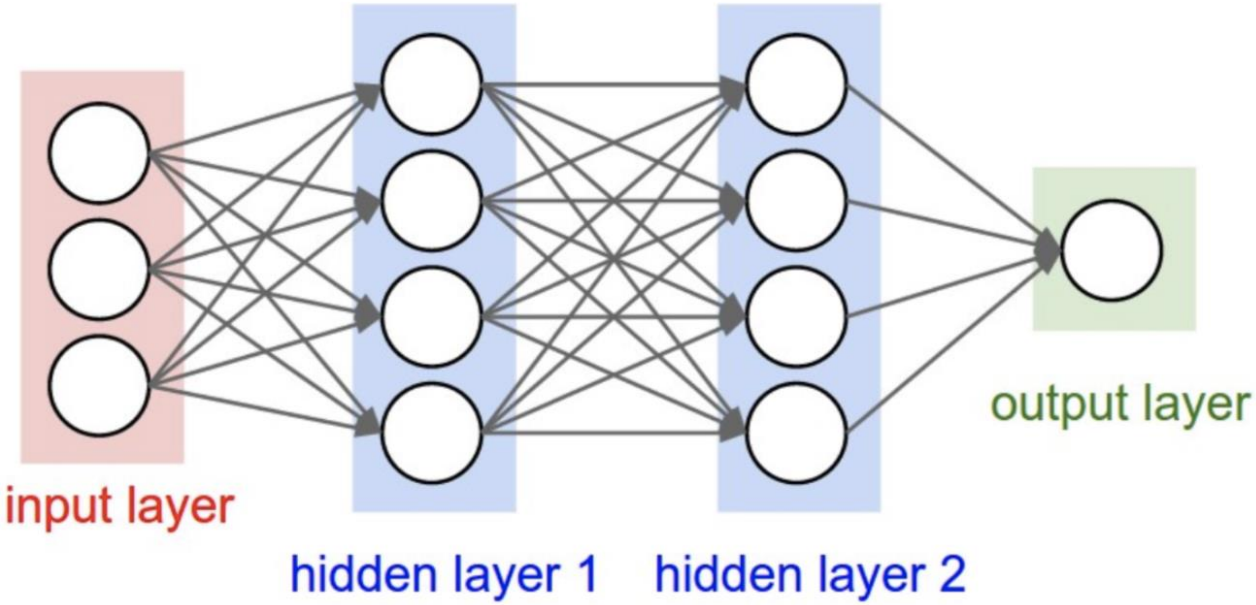
Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

- **Representation learning** attempts to automatically learn good features and representations
- **Deep learning** attempts to learn multiple levels of representation on increasing complexity/abstraction

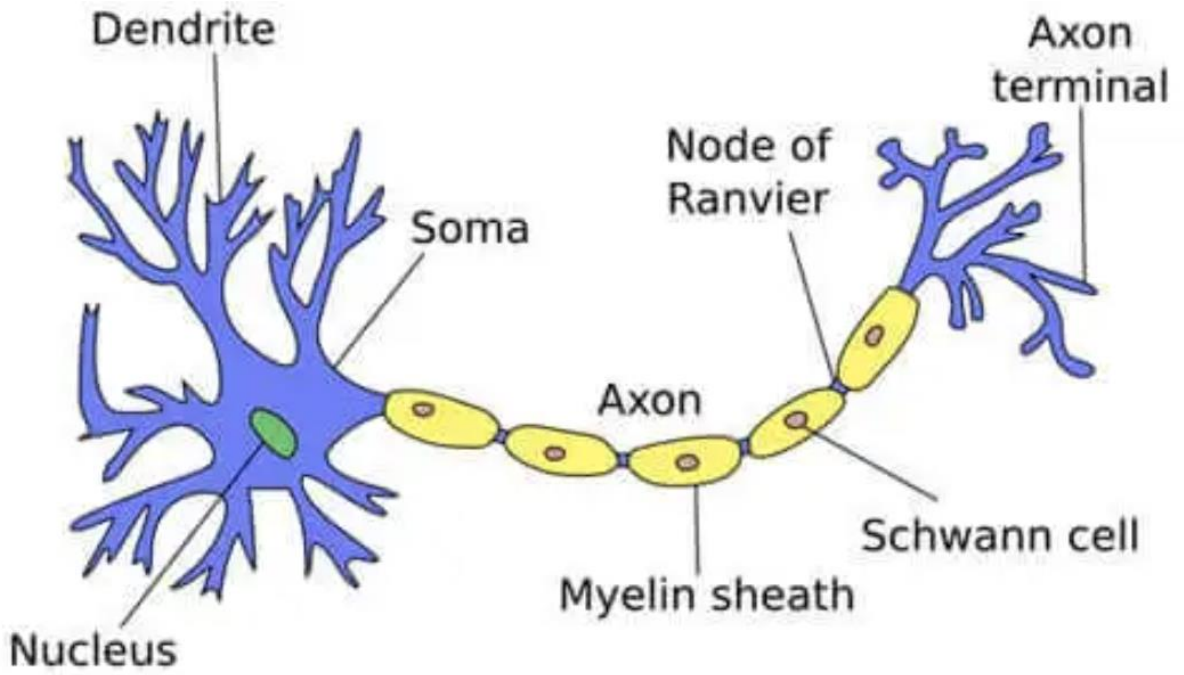
Feed-forward Neural networks

Feed-forward NNs

- Input: $x_1, \ldots, x_d$
- Output: $y \in \{0,1\}$



A Neuron in Human Brain



Each neuron is made up of a cell body with some connections coming off it:

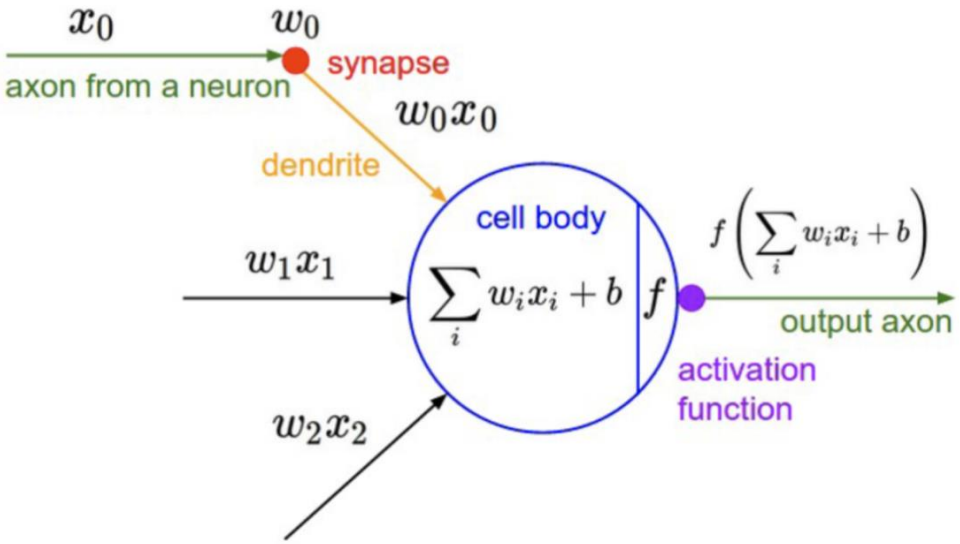
numerous dendrites (the cell's inputs—carrying information toward the cell body) and ***a single axon*** (the cell's output—carrying information away).

Dendrites extend from the neuron cell body and receive messages from other neurons.

When neurons receive or send messages, they transmit electrical impulses along their axons that aid in carrying out functions such as *storing memories*, *controlling muscles*, and more.

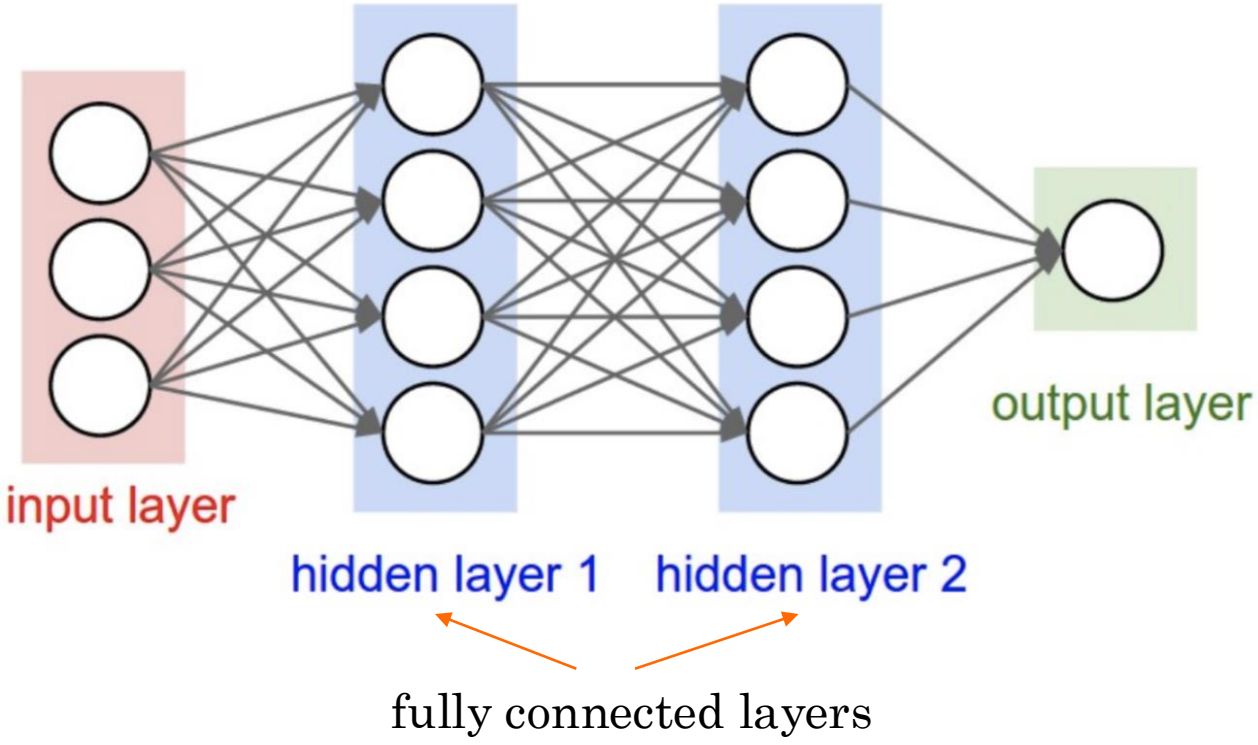
An Artificial Neuron

- A neuron is a computational unit that has scalar inputs and an output
- Each input has an associated weight.
- The neuron multiples each input by its weight, sums them, applies a **nonlinear function** to the result, and passes it to its output.

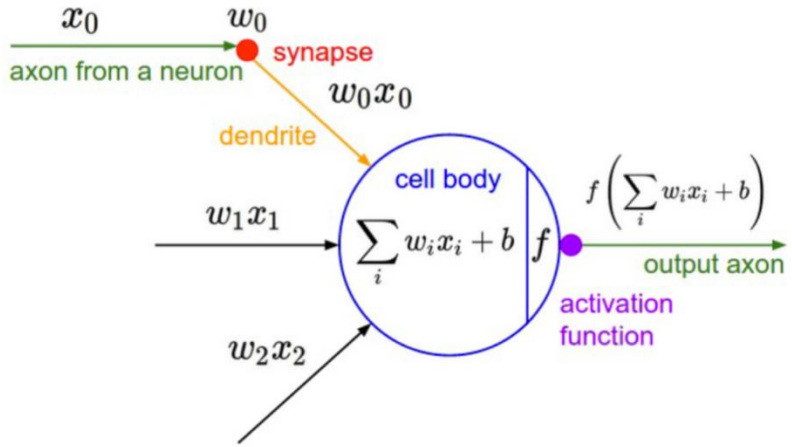


A Neural Network

- The neurons are connected to each other, forming a **network**
- The output of a neuron may feed into the inputs of other neurons

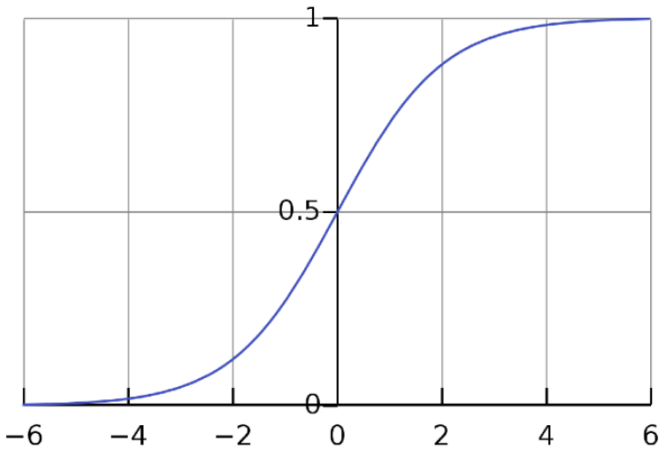


A Neuron Can be a Binary Logistic Regression Unit

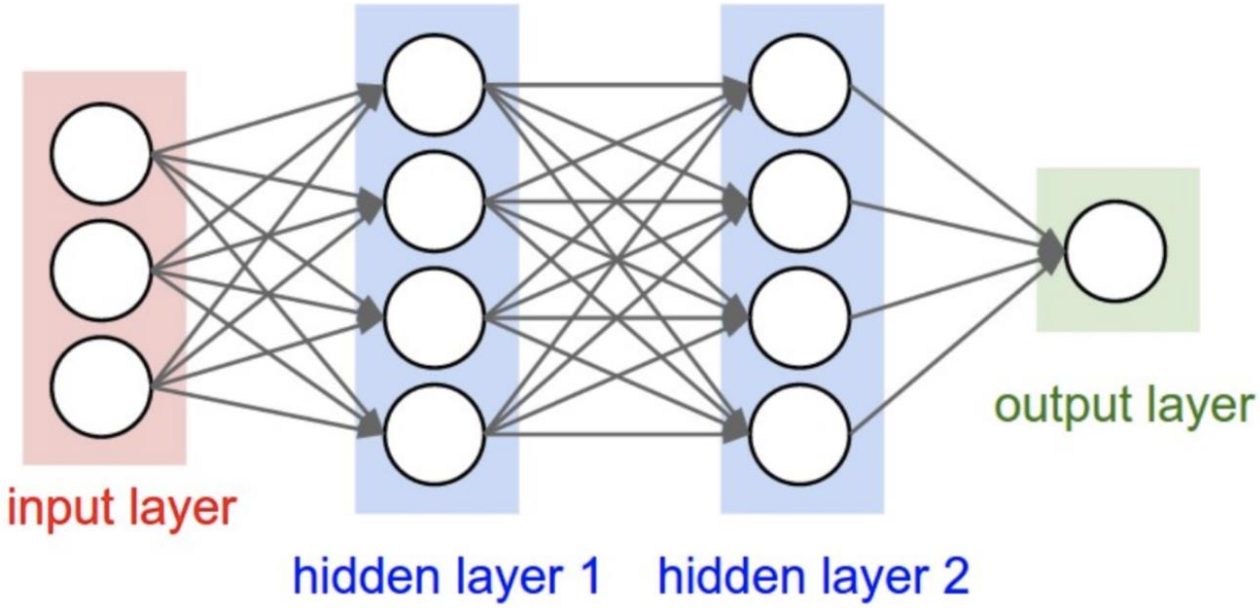


$$f(z) = \frac{1}{1 + e^{-z}}$$

$$h_{\mathbf{w},b}(\mathbf{x}) = f(\mathbf{w}^\intercal \mathbf{x} + b)$$

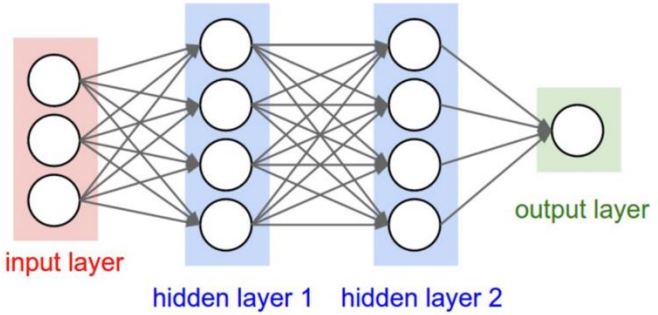


A NEURAL NETWORK IS SEVERAL LOGISTIC REGRESSION RUNNING SIMULTANEOUSLY



- If we feed a vector of inputs through a bunch of logistic regression functions, then we get a vector of outputs…
- which we can feed into another logistic regression function

Mathematical Notations



- Input layer: $x_1, x_2, \ldots, x_d$
- Hidden layer 1: $h_1^{(1)}, h_2^{(1)}, \ldots, h_{d_1}^{(1)}$

$$h_1^{(1)} = f(W_{1,1}^{(1)} x_1 + W_{1,2}^{(1)} x_2 + \cdots + W_{1,d}^{(1)} x_d + b_1^{(1)})$$
$$h_2^{(1)} = f(W_{2,1}^{(1)} x_1 + W_{2,2}^{(1)} x_2 + \cdots + W_{2,d}^{(1)} x_d + b_2^{(1)})$$

$\ldots$

- Hidden layer 2: $h_1^{(2)}, h_2^{(2)}, \ldots, h_{d_2}^{(2)}$

$$h_1^{(2)} = f(W_{1,1}^{(2)} h_1^{(1)} + W_{1,2}^{(2)} h_2^{(1)} + \ldots + W_{1,d_1}^{(2)} h_{d_1}^{(1)} + b_1^{(2)})$$

$$h_2^{(2)} = f(W_{2,1}^{(2)} h_1^{(1)} + W_{2,2}^{(2)} h_2^{(1)} + \ldots + W_{2,d_1}^{(2)} h_{d_1}^{(1)} + b_2^{(2)})$$

$\ldots$

- Output layer:

$$y = \sigma(w_1^{(o)} h_1^{(2)} + w_2^{(o)} h_2^{(2)} + \ldots + w_{d_2}^{(o)} h_{d_2}^{(2)} + b^{(o)})$$

Matrix Notations

- Input layer: $x \in \mathbb{R}^d$
- Hidden layer 1:

$$\mathbf{h}_1 = f(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \in \mathbb{R}^{d_1}$$

$$\mathbf{W}^{(1)} \in \mathbb{R}^{d_1 \times d}, \mathbf{b}^{(1)} \in \mathbb{R}^{d_1}$$

- Hidden layer 2:

$$\mathbf{h}_2 = f(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)}) \in \mathbb{R}^{d_2}$$

$$\mathbf{W}^{(2)} \in \mathbb{R}^{d_2 \times d_1}, \mathbf{b}^{(2)} \in \mathbb{R}^{d_2}$$

- Output layer:

$$y = \sigma(\mathbf{w}^{(o)} \cdot \mathbf{h}_2 + b^{(o)})$$

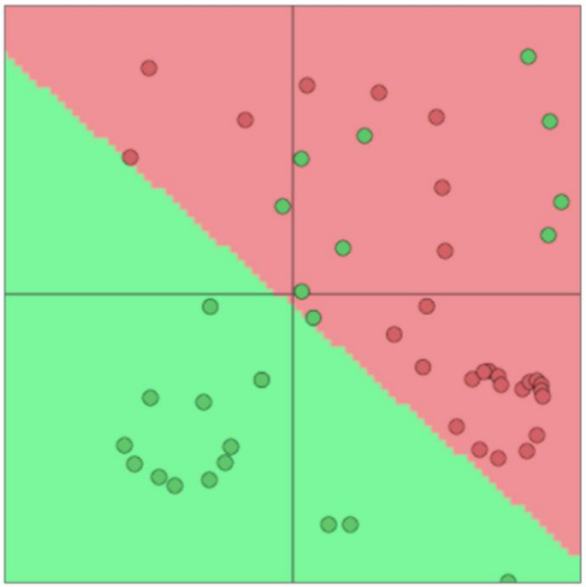
$f(.)$ is scalar function applied element wise:
$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$

Quiz: Human Neurons

- Human brain has many more neurons than our common ANNs. But human doesn't need to remember 1M translation pairs to be able to "train these neurons" and translate a language to another language well. Why?

Why Non-linear Function?

- Neural Networks can learn much more complex functions and non-linear decision boundaries

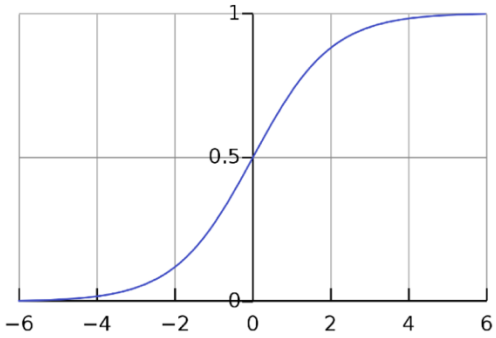


Capcity of the network increases with more hidden units and more hidden layers

Activation Functions

sigmoid

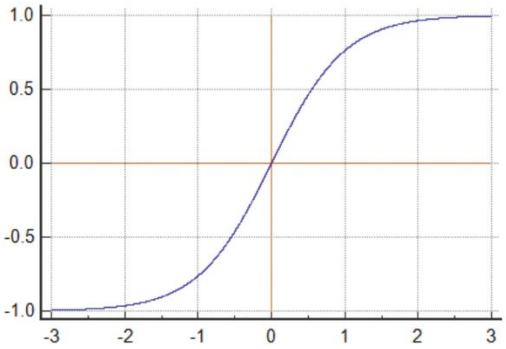
$$f(z) = \frac{1}{1 + e^{-z}}$$



$$f'(z) = f(z) \times (1 - f(z))$$

tanh

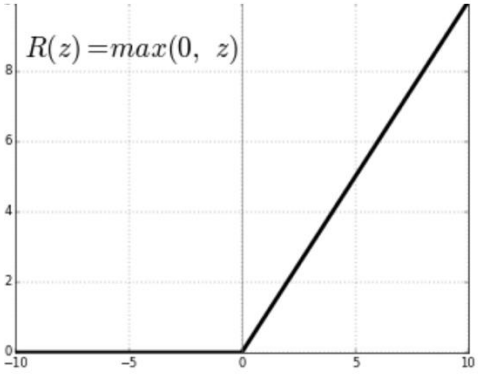
$$f(z) = \frac{e^{2z} - 1}{e^{2z} + 1}$$



$$f'(z) = 1 - f(z)^2$$

ReLU
(rectified linear unit)

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

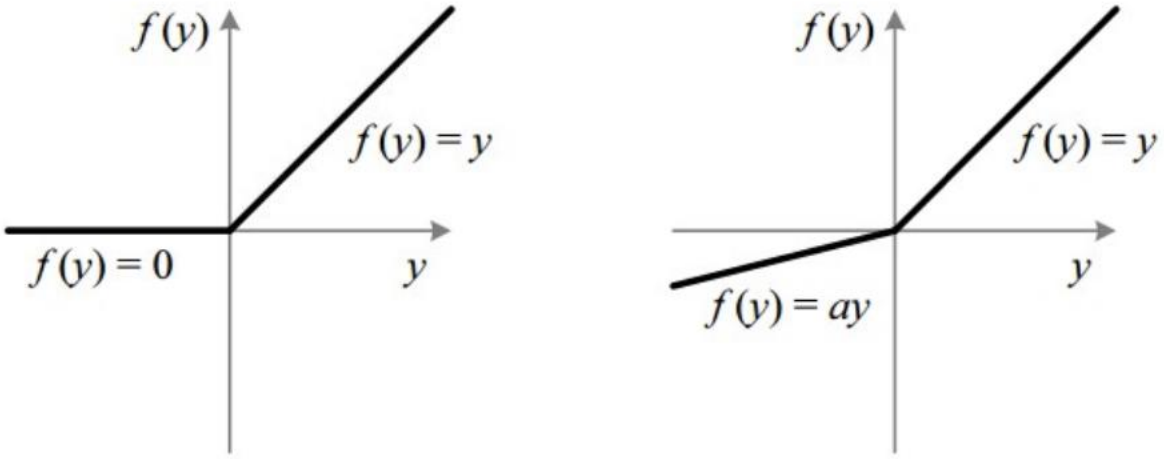


$$f'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$

Activation Functions

- Problem of ReLU? "Dead neurons" when z<0
- Leaky ReLU:

$$f(z) = \begin{cases} z & z \geq 0 \\ 0.01z & z < 0 \end{cases}$$



- Also: blowing up the activation (to infinity!)

Quiz: ReLu

- What are the advantages of ReLU over Sigmoid as an activation function?

Loss Function (at the output layer)

- Binary classification

$$y = \sigma(\mathbf{w}^{(o)} \cdot \mathbf{h}_2 + b^{(o)})$$

$$\mathcal{L}(y, y^*) = -y^* \log y - (1 - y^*) \log (1 - y)$$

- Regression

$$y = \mathbf{w}^{(o)} \cdot \mathbf{h}_2 + b^{(o)}$$

$$\mathcal{L}_{\text{MSE}}(y, y^*) = (y - y^*)^2$$

- Multi-class classification

$$y_i = \text{softmax}_i(\mathbf{W}^{(o)}\mathbf{h}_2 + \mathbf{b}^{(o)}) \quad \mathbf{W}^{(o)} \in \mathbb{R}^{C \times d_2}, \mathbf{b}^{(o)} \in \mathbb{R}^C$$

$$\mathcal{L}(y, y^*) = -\sum_{i=1}^{C} y_i^* \log y_i$$

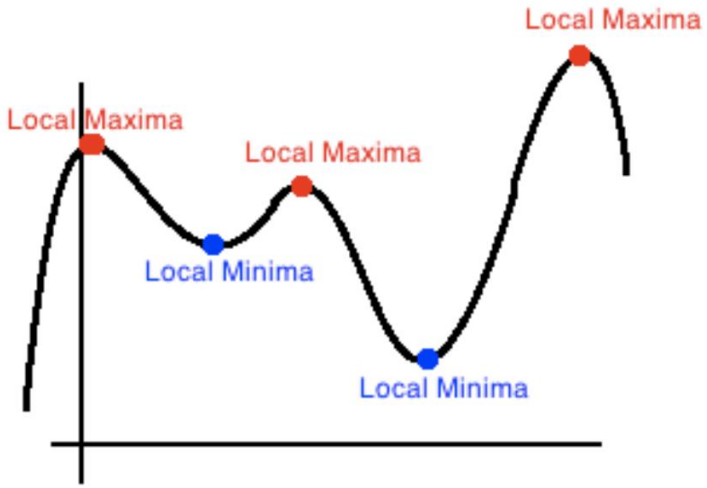
The question again becomes how to compute: $\nabla_\theta \mathcal{L}(\theta)$

$$\theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \mathbf{w}^{(o)}, b^{(o)}\}$$

Optimization

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta J(\theta)$$

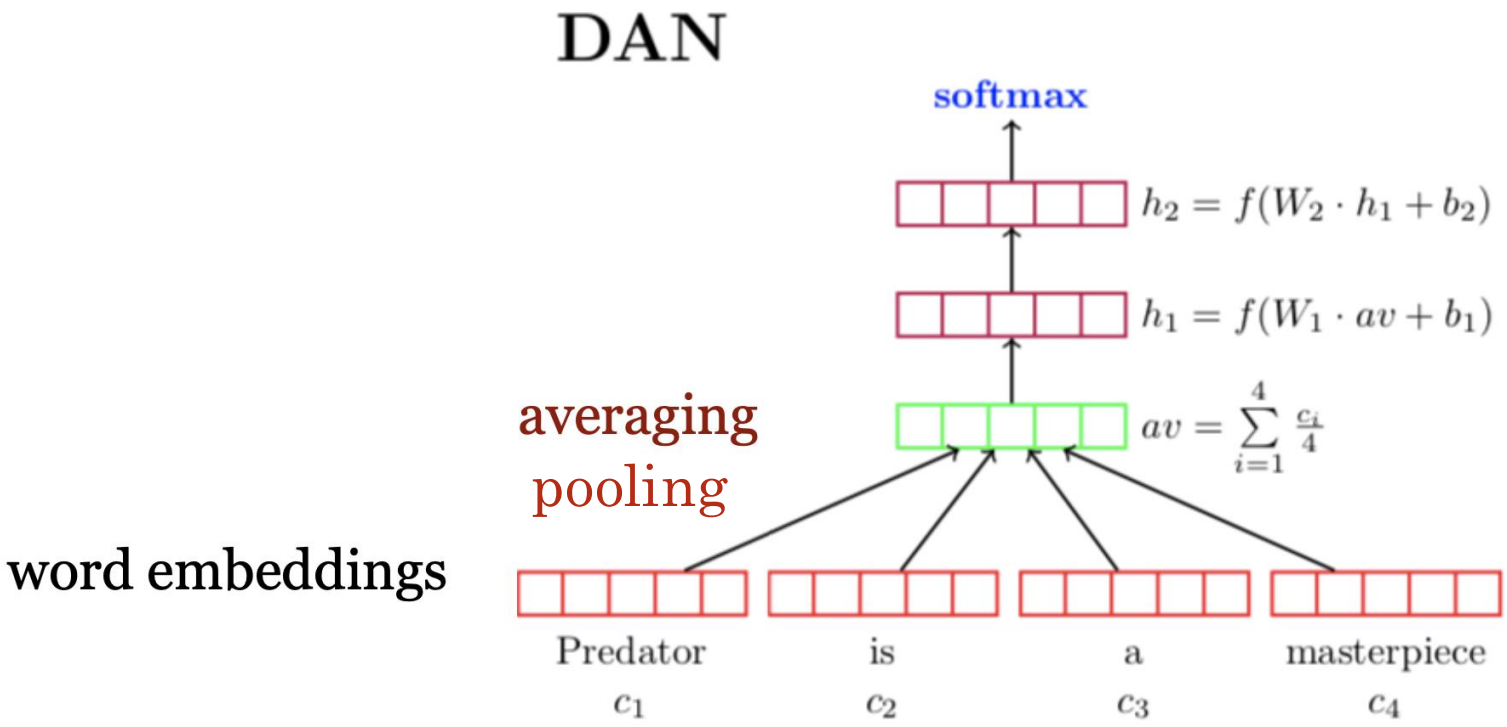
- Logitistic regression is convex: one global minimum.
- Neural networks are non-convex and not easy to optimize!
- A class of more sophisticated "adaptive" optimizers that scale the parameter adjustment by an accumulated gradient
  - Adam
  - Adagrad
  - RMSprop
  - …



Applications

Neural Bag-of-Words (NBOW)

- Deep Averaging Networks (DAN) for Text Classification



(Iyyer et 2015): Deep Unordered Composition Rivals Syntactic Methods for Text Classification

Word embedding: re-train or not?

- Word embeddings can be treated as parameters, too!

$$\theta = \{\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \mathbf{W}^{(2)}, \mathbf{b}^{(2)}, \mathbf{w}^{(o)}, b^{(o)}, \mathbf{E}_{emb}\}$$

- When the training set is small, don't re-train word embeddings (think of them as features!).

Why?

- Most cases: initialize word embeddings using pre-trained ones (word2vec, Glove) and re-train them for the task.

Good vs. bad

- When you have enough data, you can just randomly initialize them and train from scratch (e.g. machine translation)

Neural Bag-of-Words (NBOW)

Model	RT	SST fine	SST bin	IMDB	Time (s)
DAN-ROOT	—	46.9	85.7	—	**31**
DAN-RAND	77.3	45.4	83.2	88.8	136
DAN	80.3	47.7	86.3	89.4	136
NBOW-RAND	76.2	42.3	81.4	88.9	91
NBOW	79.0	43.6	83.6	89.0	91
BiNB	—	41.9	83.1	—	—
NBSVM-bi	79.4	—	—	91.2	—

Feed-forward Neural Networks

- N-gram models: $P(mat \mid the\ cat\ sat\ on\ the)$
  - Input layer (context size **n=5**):

$$\mathbf{x} = [\mathbf{e}_{\text{the}}; \mathbf{e}_{\text{cat}}; \mathbf{e}_{\text{sat}}; \mathbf{e}_{\text{on}}; \mathbf{e}_{\text{the}}] \in \mathbb{R}^{dn}$$

concatenation

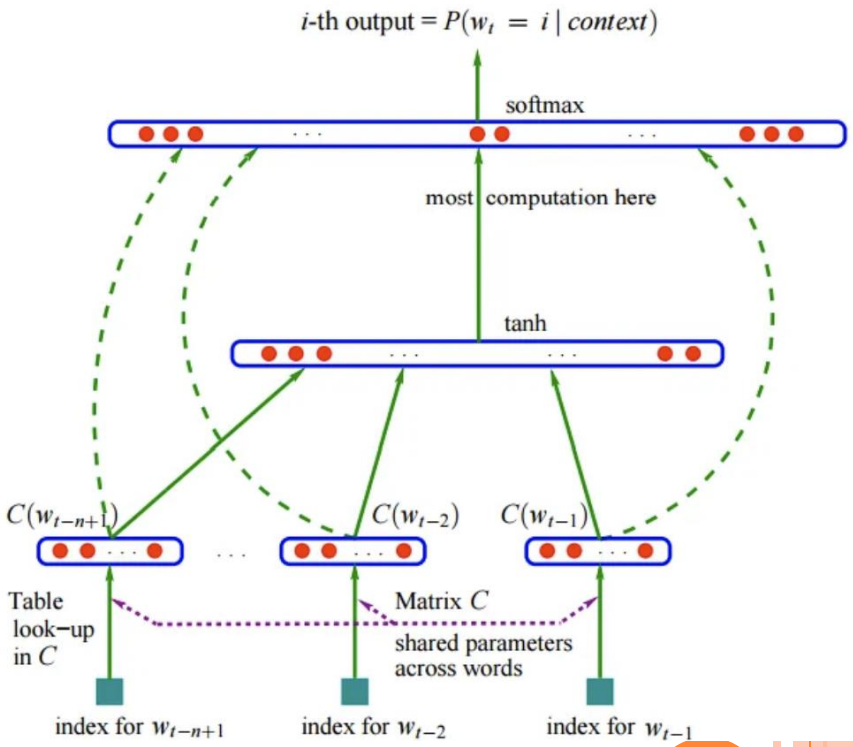
  - Hidden layer:

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^{h}$$

  - Output layer:

$$\mathbf{z} = \mathbf{U}\mathbf{h} \in \mathbb{R}^{|V|}$$

$$P(w = i \mid \text{context}) = softmax_i(\mathbf{z})$$



(Bengio et 2003): A Neural Probabilistic Language Model

Backpropagation

How to compute gradients

Backpropagation

- It's taking derivatives and applying chain rule!
- We'll **re-use** derivatives computed for higher layers in computing derivatives for lower layers so as to minimize computation
- Good news is that modern automatic differentiation tools did all for you!
  - Implementing backprop by hand is like programming in assembly language.

Deriving Gradients for Feed-forward NNs

Input: $\mathbf{x}$ $\quad \mathbf{x} \in \mathbb{R}^d$

$\mathbf{h}_1 = \tanh(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$ $\quad \mathbf{W}_1 \in \mathbb{R}^{d_1 \times d}$ $\quad \mathbf{b}_1 \in \mathbb{R}^{d_1}$

$\mathbf{h}_2 = \tanh(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$ $\quad \mathbf{W}_2 \in \mathbb{R}^{d_2 \times d_1}$ $\quad \mathbf{b}_2 \in \mathbb{R}^{d_2}$

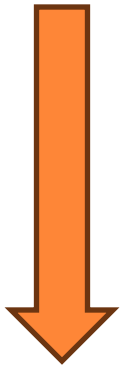
$y = \sigma(\mathbf{w}^\intercal \mathbf{h_2} + b)$ $\quad \mathbf{w} \in \mathbb{R}^{d_2}$

$$\mathcal{L}(y, y^*) = -y^* \log y - (1 - y^*) \log (1 - y)$$

$$\frac{\partial L}{\partial \mathbf{w}} = ? \qquad \frac{\partial L}{\partial b} = ?$$

$$\frac{\partial L}{\partial \mathbf{W}_2} = ? \qquad \frac{\partial L}{\partial \mathbf{b}_2} = ?$$

$$\frac{\partial L}{\partial \mathbf{W}_1} = ? \qquad \frac{\partial L}{\partial \mathbf{b}_1} = ?$$



Going backward!

Deriving Gradients for Feed-forward NNs

$$\mathbf{z}_1 = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \qquad \mathbf{h}_1 = \tanh(\mathbf{z}_1)$$

$$\mathbf{z}_2 = \mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2 \qquad \mathbf{h}_2 = \tanh(\mathbf{z}_2)$$

$$y = \sigma(\mathbf{w}^\intercal \mathbf{h_2} + b)$$

Forward Propagation

$$\frac{\partial \mathcal{L}}{\partial b} = y - y^* \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{w}} = (y - y^*)\mathbf{h}_2 \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{h}_2} = (y - y^*)\mathbf{w}$$

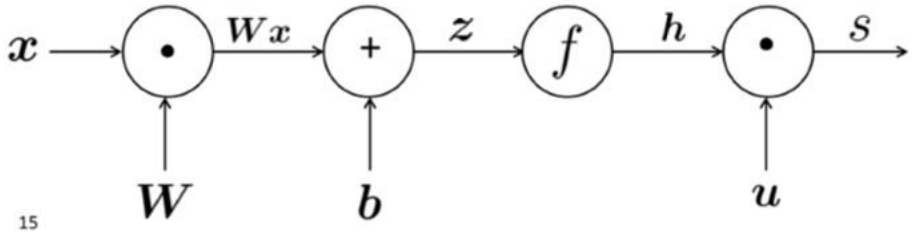
$$\frac{\partial L}{\partial \mathbf{z}_2} = (1 - \mathbf{h}_2^2) \circ \frac{\partial L}{\partial \mathbf{h}_2}$$

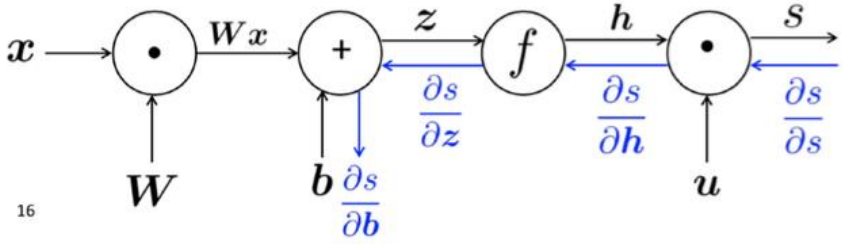
Backward Propagation

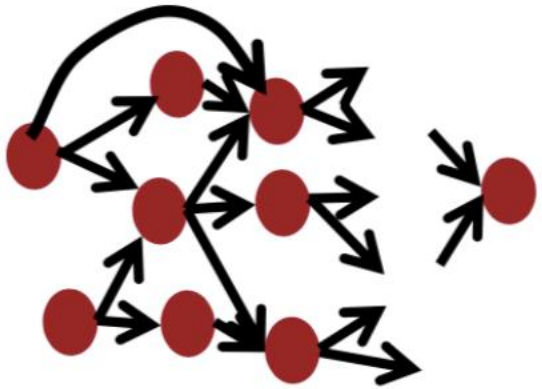
$$\frac{\partial L}{\partial \mathbf{W}_2} = \frac{\partial L}{\partial \mathbf{z}_2}\mathbf{h}_1^\intercal \qquad \frac{\partial L}{\partial \mathbf{b}_2} = \frac{\partial L}{\partial \mathbf{z}_2} \qquad \frac{\partial L}{\partial \mathbf{h}_1} = \mathbf{W}_2^\intercal \frac{\partial L}{\partial \mathbf{z}_2}$$

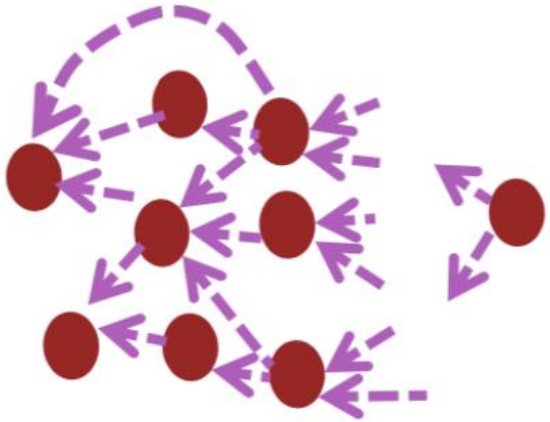
$$\frac{\partial L}{\partial \mathbf{z}_1} = (1 - \mathbf{h}_1^2) \circ \frac{\partial L}{\partial \mathbf{h}_1} \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{z_1}}\mathbf{x}^\intercal \qquad \frac{\partial \mathcal{L}}{\partial \mathbf{b}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{z_1}}$$

Computation Graphs





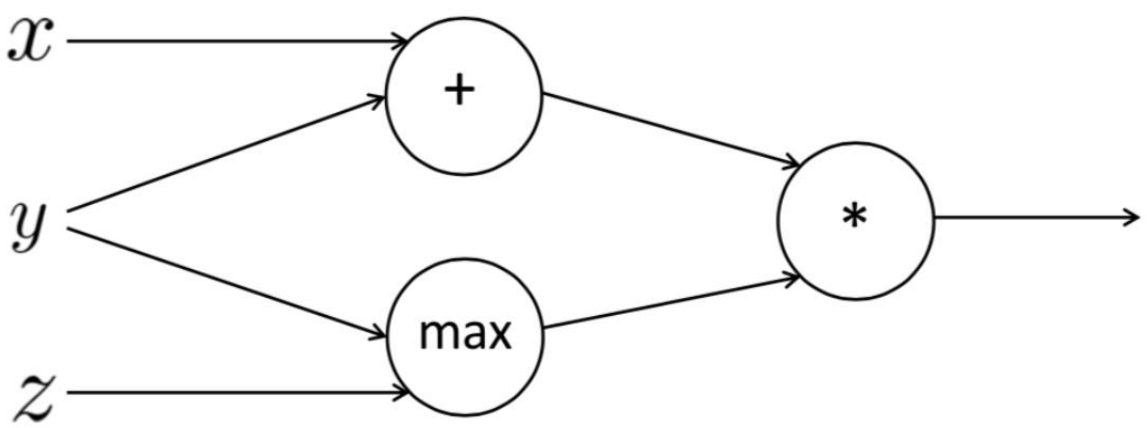




An Example

$$f(x, y, z) = (x + y)\max(y, z)$$
$$x = 1, y = 2, z = 0$$

$$a = x + y$$
$$b = \max(y, z)$$
$$f = ab$$



Backpropagation in General Computational Graphs

- Forward propagation: visit nodes in topological sort order
  - Compute value of node given predecessors
- Backward propagation:
  - Initialize output gradient as 1
  - Visit nodes in reverse order and compute gradient w.r.t. each node using gradient w.r.t. successors

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

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