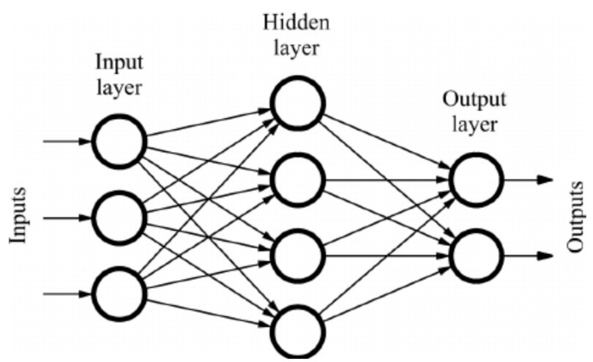
# CSE 4392 Special Topics
# Natural Language Processing

# Neural Network Basics

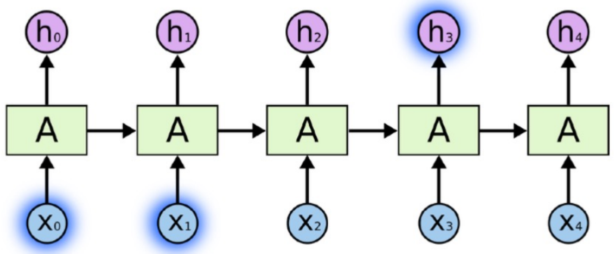
2024 Spring

# Neural Networks for NLP

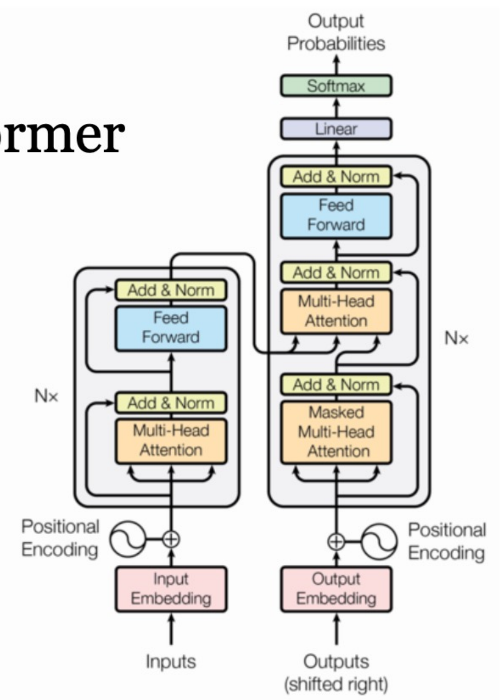
Feed-forward NNs



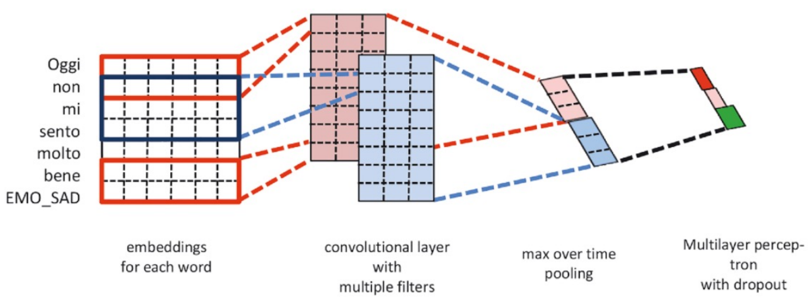

Recurrent NNs




Transformer




Convolutional NNs




***Always coupled with word embeddings…***

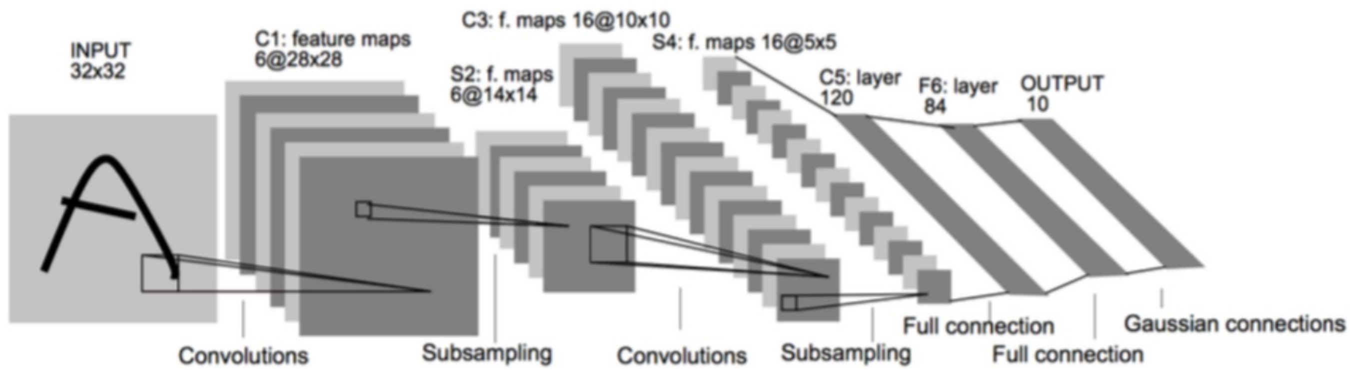
# In This Lecture

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

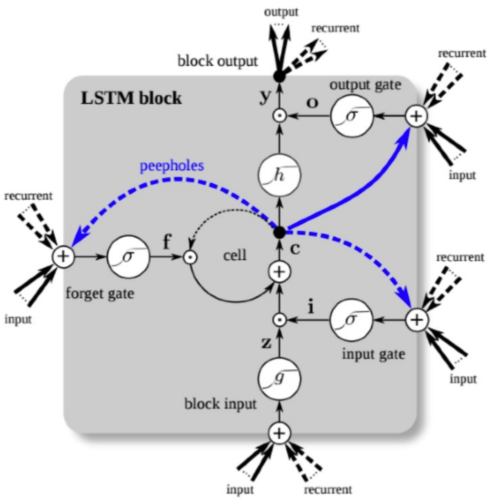
# The History of Neural Networks

# NN "Dark Ages"

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

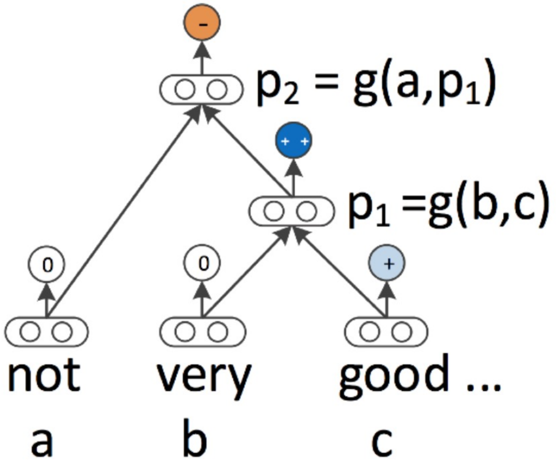



- Long Short-term Memory Networks (LSTMs): Hochreiter and Schmidhuber 1997
- Henderson 2003: neural shift-reduce parser, not SOTA



# 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

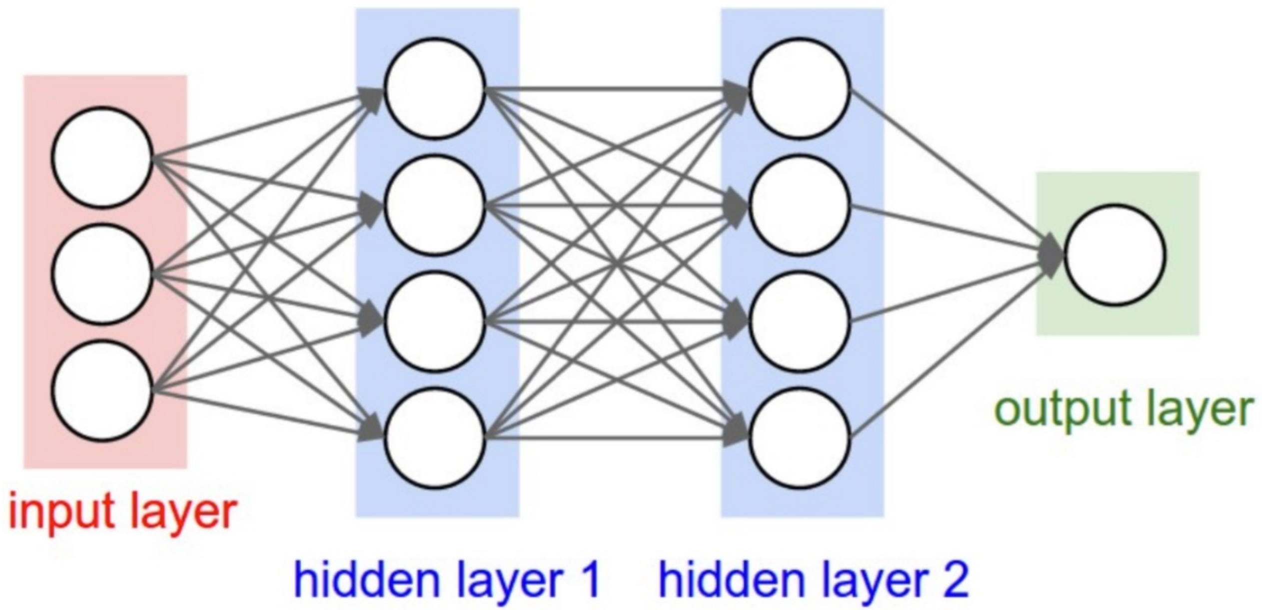
| Var | Definition | Value in Fig. 5.2 |
|---|---|---|
| $x_1$ | count(positive lexicon) $\in$ doc) | 3 |
| $x_2$ | count(negative lexicon) $\in$ doc) | 2 |
| $x_3$ | $\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$ | 1 |
| $x_4$ | count(1st and 2nd pronouns $\in$ 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

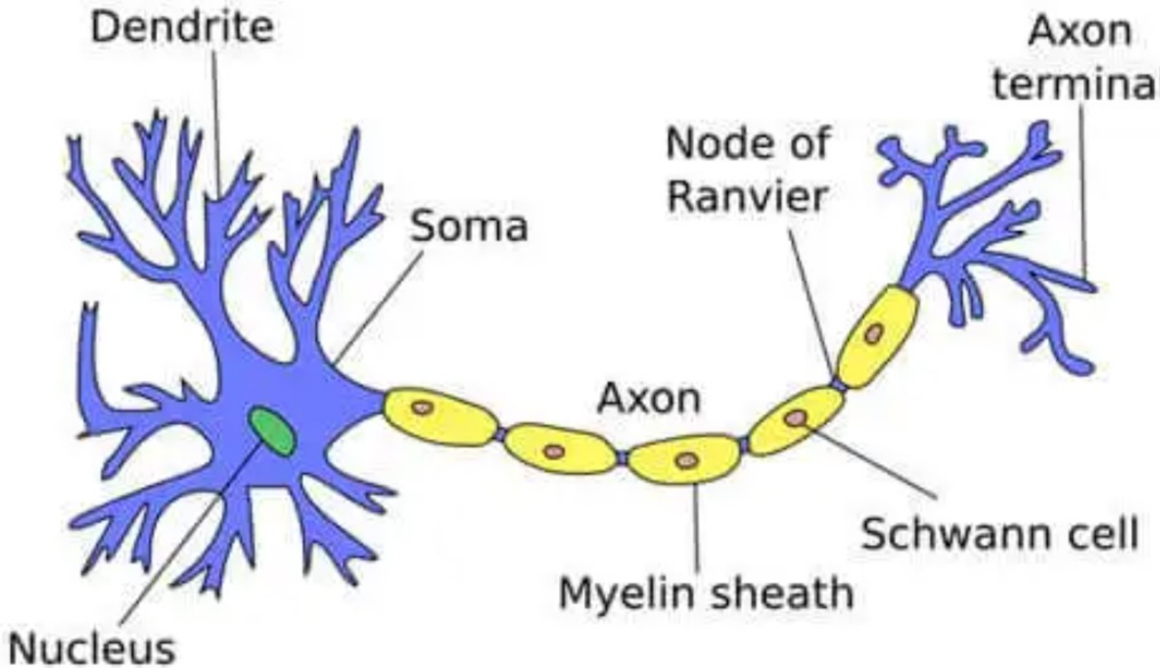
# Feed-forward Neural networks

# 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:

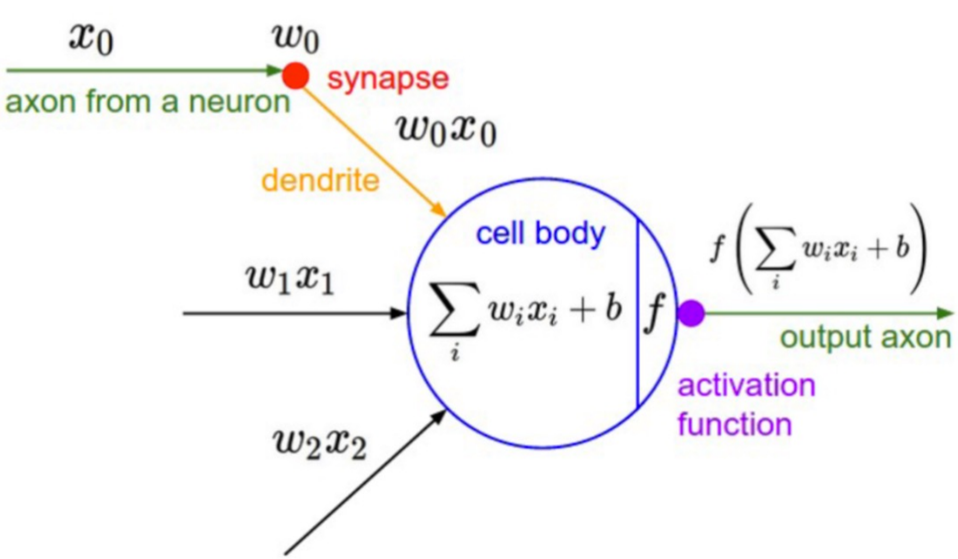
***numerous dendrites*** (the cell's inputs—carrying information toward the cell body) and ***a single axon*** (the cell's output—carrying information away).

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.

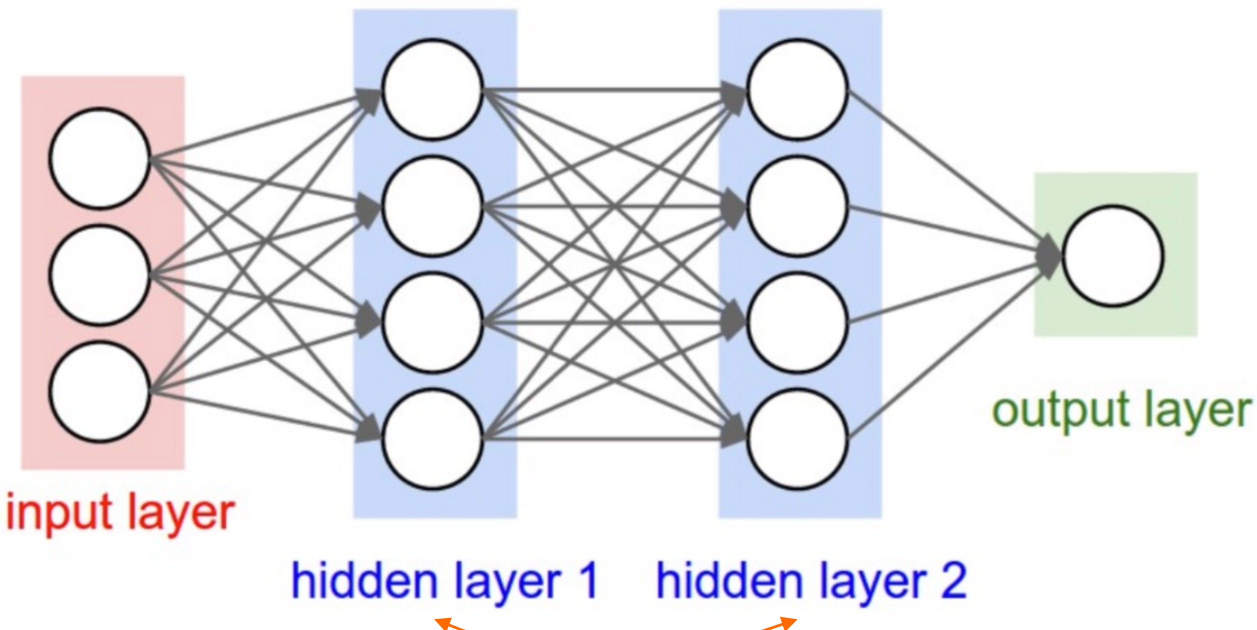
# AN ARTIFICIAL NEURON

- 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, applied a **nonlinear function** to the result, and passes it to its output.



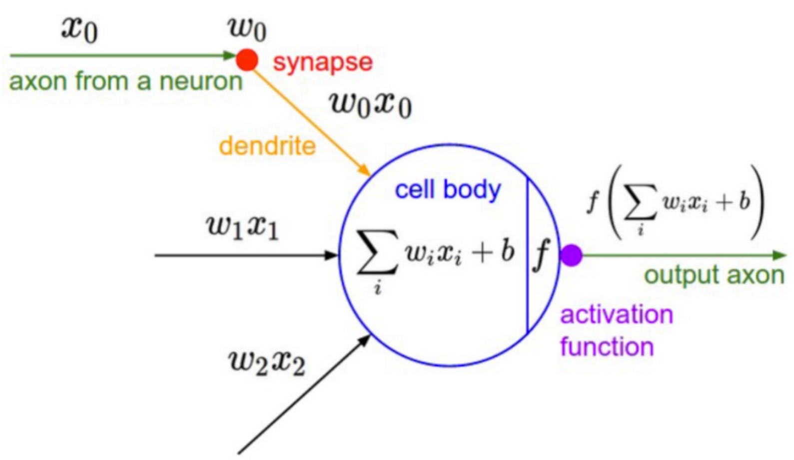
# A Neural Network

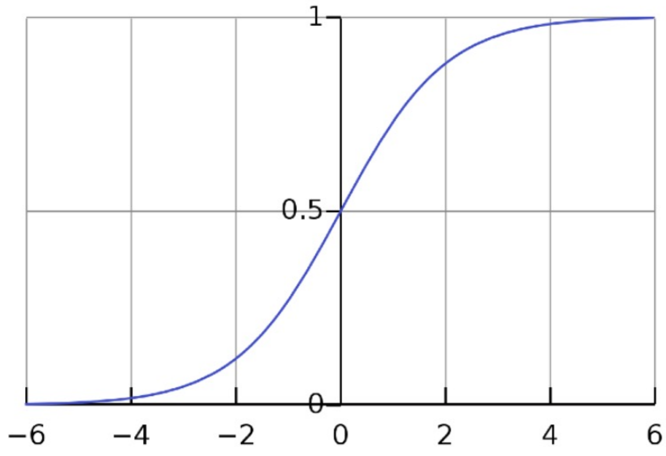
- The neurons are connected to each other, forming a **network**
- The output of a neuron may feed into the inputs of other neurons




fully connected layers

# A Neuron Can be a Binary Logistic Regression Unit

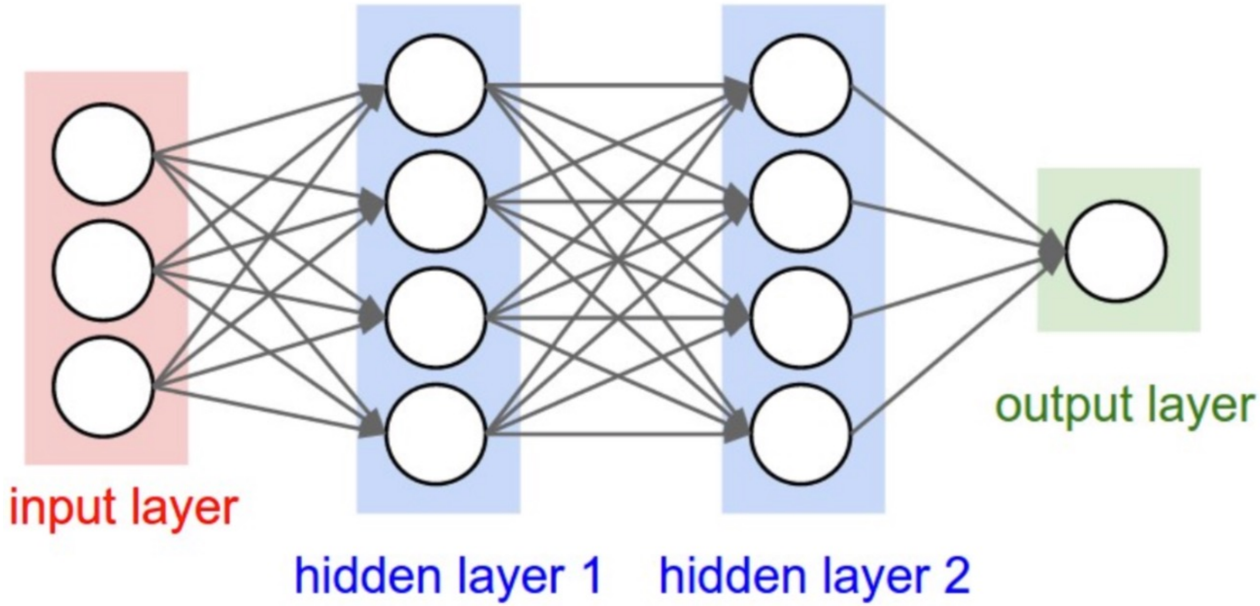



$$f(z) = \frac{1}{1 + e^{-z}}$$

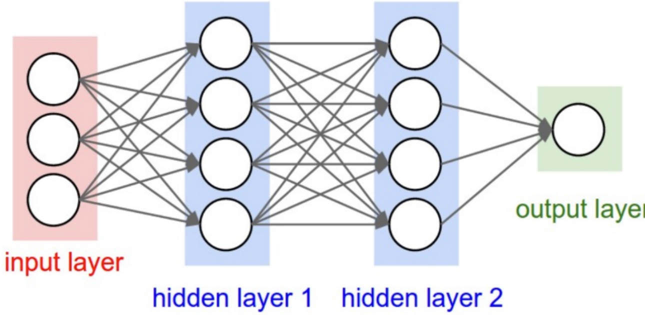
$$h_{\mathbf{w},b}(\mathbf{x}) = f(\mathbf{w}^\intercal \mathbf{x} + b)$$

# 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: $h_1^{(2)}, h_2^{(2)}, \ldots, h_{d_2}^{(2)}$

$$h_1^{(2)} = f(W_{1,1}^{(2)} h_1^{(1)} + W_{1,2}^{(2)} h_2^{(1)} + \ldots + W_{1,d_1}^{(2)} h_{d_1}^{(1)} + b_1^{(2)})$$
$$h_2^{(2)} = f(W_{2,1}^{(2)} h_1^{(1)} + W_{2,2}^{(2)} h_2^{(1)} + \ldots + W_{2,d_1}^{(2)} h_{d_1}^{(1)} + b_2^{(2)})$$
$$\cdots$$

- 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:

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

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

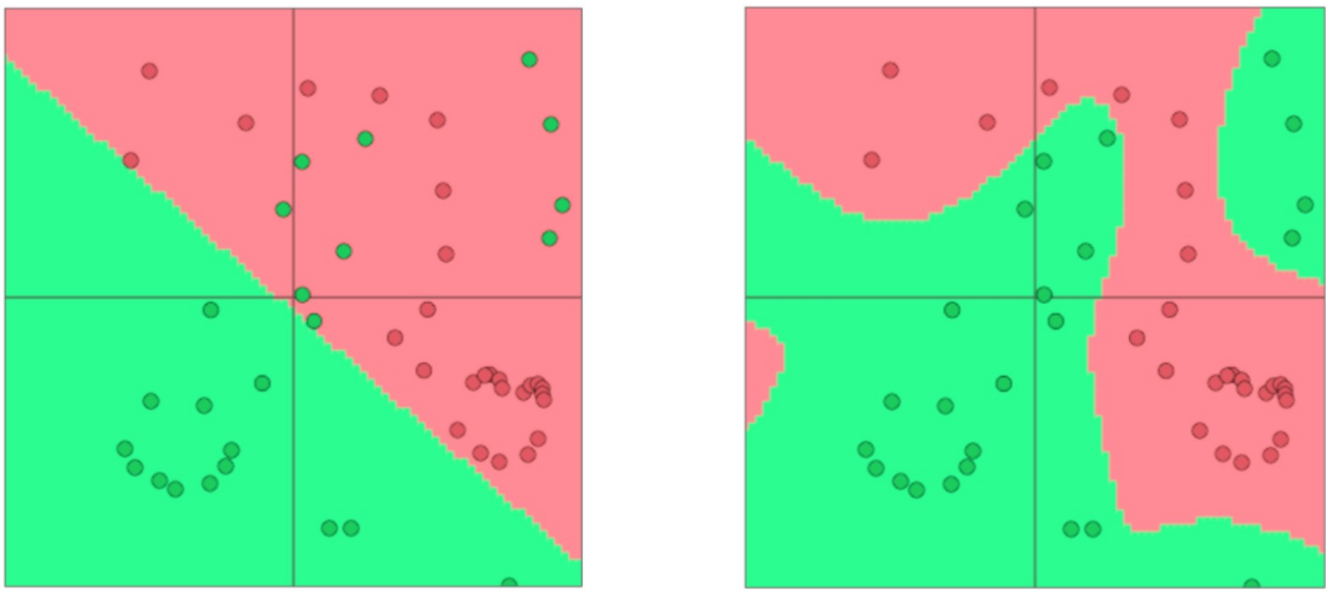
$f(.)$ is scalar function applied element wise: $f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$

- Output layer:

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

# WHY NON-LINEAR FUNCTION?

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

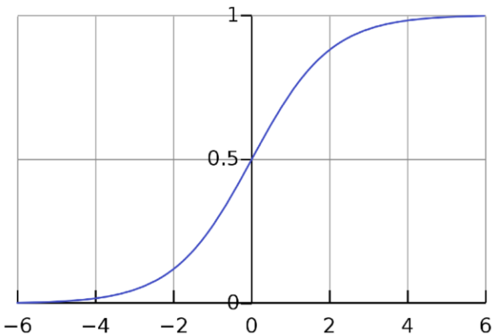
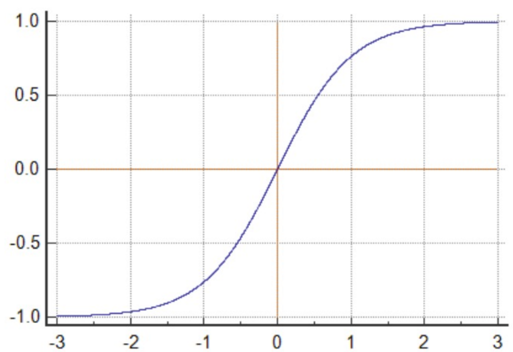
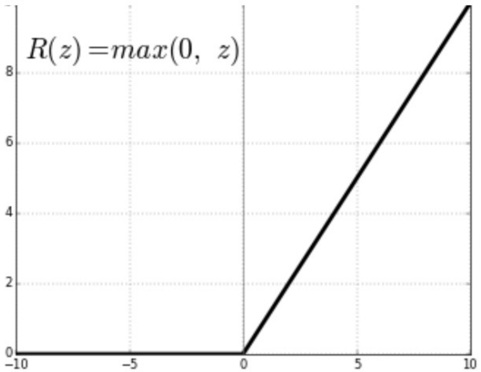


Capcity of the network increases with more hidden units and more hidden layers

# Quiz: Activation Function

- What if we remove the activation function?

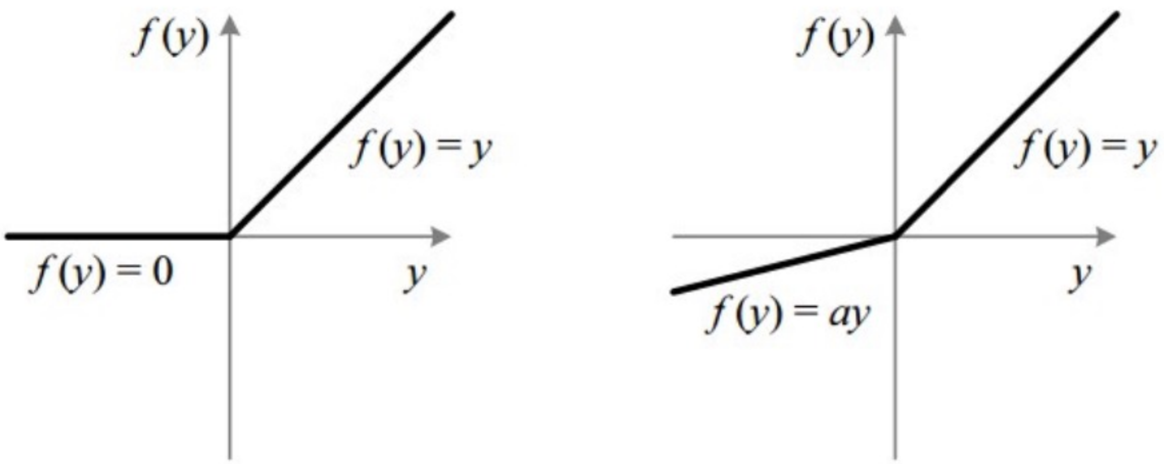
# Activation Functions

| sigmoid | tanh | ReLU (rectified linear unit) |
|---|---|---|
| $f(z) = \frac{1}{1+e^{-z}}$ | $f(z) = \frac{e^{2z}-1}{e^{2z}+1}$ | $f(z) = \max(0, z)$ |
|  |  |  |
| $f'(z) = f(z) \times (1 - f(z))$ | $f'(z) = 1 - f(z)^2$ | $f'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$ |

# 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

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

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

- Multi-class classification

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

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

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
  - ...