CS378: Natural Language Processing

Lecture 4: Feedforward Neural Network

The University of Texas at Austin

Eunsol Choi

Slides adapted from Greg Durrett, Yoav Artzi, Yejin Choi, Princeton NLP course

Logistics

- Course modality survey is on the Piazza, please complete it.
- From next week, lectures will be in person at GDC 1.304.
- LectureOnline will be available asynchronously.
- Final Project guideline will be updated later this week, so stay tuned!

Perceptron

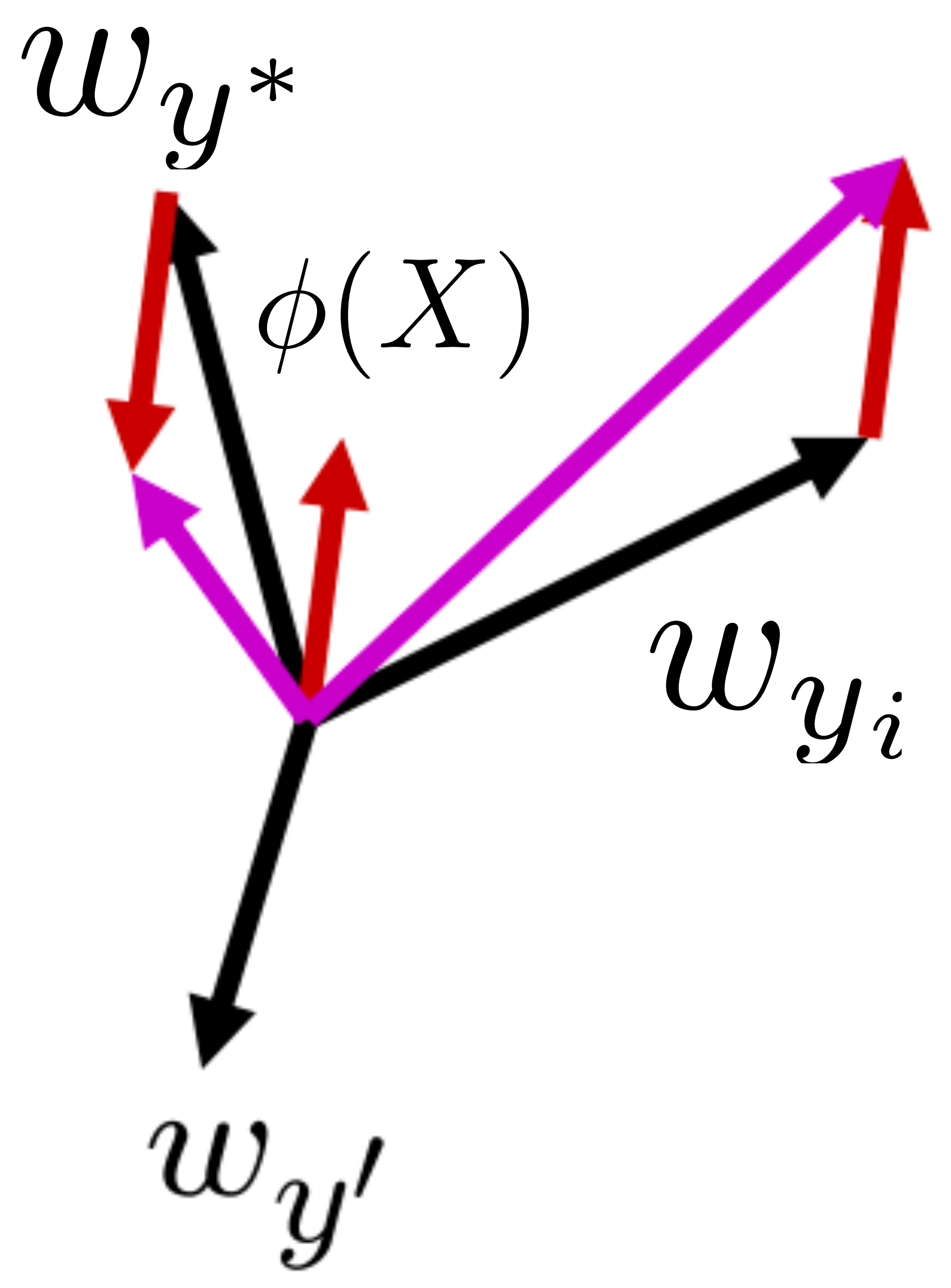
- Simple error-driven learning approach similar to logistic regression
- Start with zero weight vector.
- Visit training examples one by one.
- Decision rule: $w \cdot \phi(x) > 0$
  - If correct: do nothing!
  - If incorrect: if label is positive, $w \leftarrow w + \phi(x)$
    negative, $w \leftarrow w - \phi(x)$

Multi-class Perceptron

- A weight vector for each class: w_y
- Start with zero weights
- Visit training instances one by one
  - Make a prediction

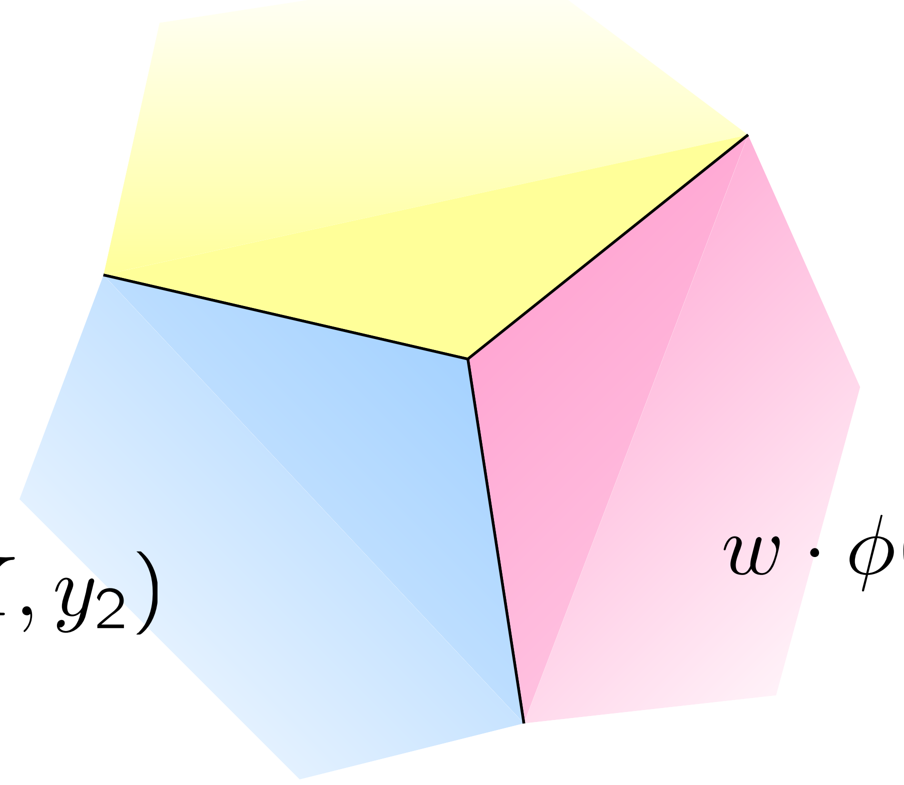
$$y^* = \text{argmax}_{y \in C} w_y \cdot \phi(x_i)$$

  - If correct ($y^* == y^{(i)}$): no change, continue!
  - If wrong: adjust weights



Multi-class Perceptron: Rewrite

- Now feature vector encodes label as well
- Start with zero weights
- Visit training instances one by one
  - Make a prediction



$$y^* = \textbf{argmax}_{y \in C} w \cdot \phi(x^i, y^i)$$

- If correct ($y^* == y^i$): no change, go to next example!
- If wrong: adjust weights

$$w = w + \phi(x^i, y^i) - \phi(x^i, y^*)$$

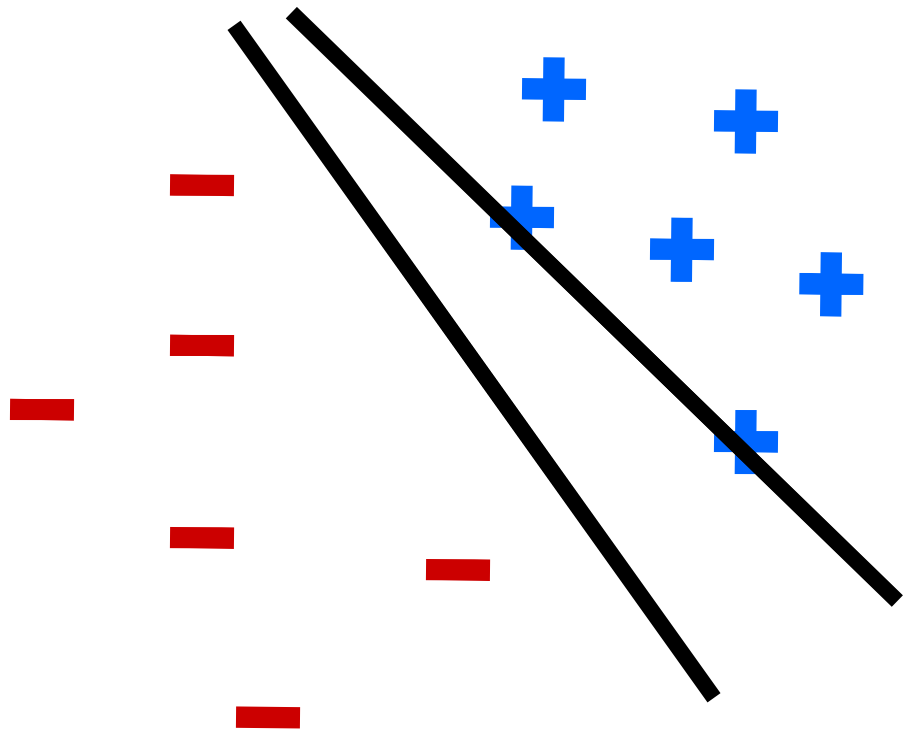
Different Weights vs. Different Features

- Different weights: $y^* = \text{argmax}_{y \in C} w_y \cdot \phi(x_i)$
  - Generalizes to neural networks: $\phi(x)$ is the first *n*-1 layers of the network, then you multiply by a final linear layer at the end

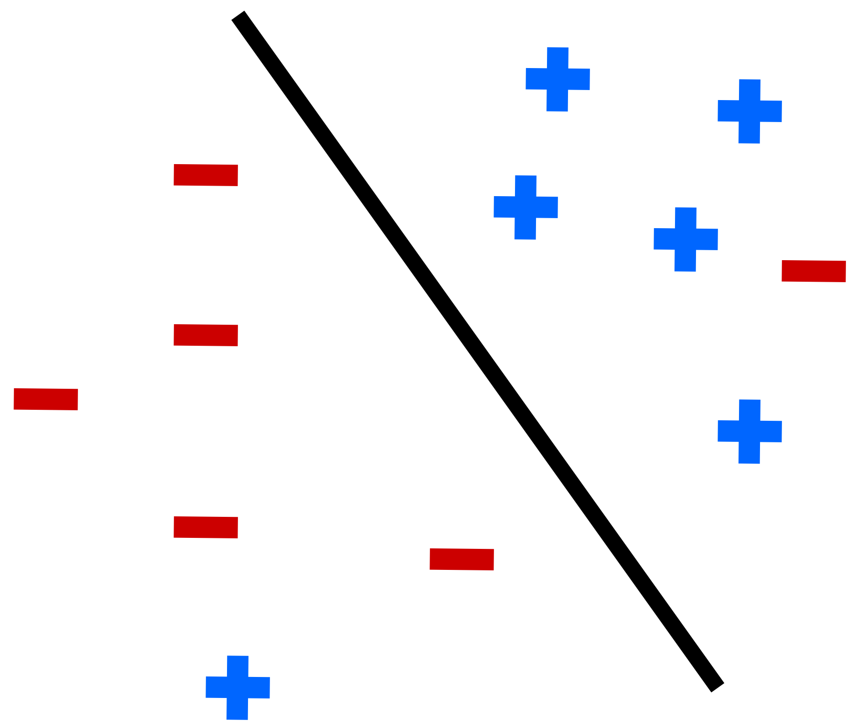
- Different features: $y^* = \text{argmax}_{y \in C} w \cdot \phi(x_i, y)$
  - Advantage? Can make feature dependent on the label
    - Suppose C is a structured label space (part-of-speech tags for each word in a sentence). $\phi(x,y)$ extracts features over shared parts of these.

Perceptron Learning

- No counting or computing probabilities on training set
- Separability: some parameters get the training set perfectly correct
- Convergence: if the training is separable, perceptron will eventually converge
- Mistake Bound: the maximum number of mistakes (binary case) related to the margin or degree of separability



Separable



Non-Separable

Logistic Regression Updates

Gradient: $$\frac{dL(w)}{dw} = [y - \sigma(w \cdot \phi(x)]\phi(x)$$ $$w^{t+1} \leftarrow w^t + \eta \frac{d}{dw} L(w)$$

Assuming learning rate $\eta = 1$

if label is positive, $$\frac{dL(w)}{dw} = [1 - p(y = 1 \,|\, x, w)]\phi(x)$$ $$w^{t+1} \leftarrow w^t + (1 - P(y = 1 \,|\, x, w))\phi(x)$$

negative, $$\frac{dL(w)}{dw} = [-p(y = 1 \,|\, x, w)]\phi(x)$$ $$w^{t+1} \leftarrow w^t - P(y = 1 \,|\, x, w)\phi(x)$$

Comparison

Perceptron

- Decision rule: $y* = 1$ If $w \cdot \phi(x) > 0$

  $y* = 0$ Otherwise

  - If correct: do nothing!
  - If incorrect:

    if label is positive,

    $$w \leftarrow w + \phi(x)$$

    negative,

    $$w \leftarrow w - \phi(x)$$

Logistic Regression

- Decision rule:

  $$y* = \text{argmax}_{y \in 0,1} p(y \mid x, w)$$

- Always:

  if label is positive,

  $$w \leftarrow w + (1 - P(y = 1 \mid x, w))\phi(x)$$

  negative,

  $$w \leftarrow w - P(y = 1 \mid x, w)\phi(x)$$

Three views of classification

- Naïve Bayes:
  - Parameters from data statistics
  - Parameters: probabilistic interpretation
  - Training: one pass through the data
- Log-linear models:
  - Parameters from gradient ascent
  - Parameters: linear, probabilistic model, and discriminative
  - Training: gradient ascent, regularize to stop overfitting
- The Perceptron:
  - Parameters from reactions to mistakes
  - Parameters: discriminative interpretation
  - Training: go through the data until validation accuracy maxes out

Overview

- Classification Problem
- Learning a classifier
  - Naive Bayes Classifier
  - Log-linear classifier (maximum entropy models)
  - Perceptron
  - **Feedforward Neural Network**

What makes neural network different from classifiers we learned so far?

Why Neural Network?

- Linear classification: $\text{argmax}_{y \in 0,1} w \cdot \phi(x)$
- Want to learn intermediate **conjunctive** features of the input

the movie was ***not*** *all that* ***good***

I[contains *not* & contains *good*]

- How do we learn this if our feature vector is just the unigram indicators?

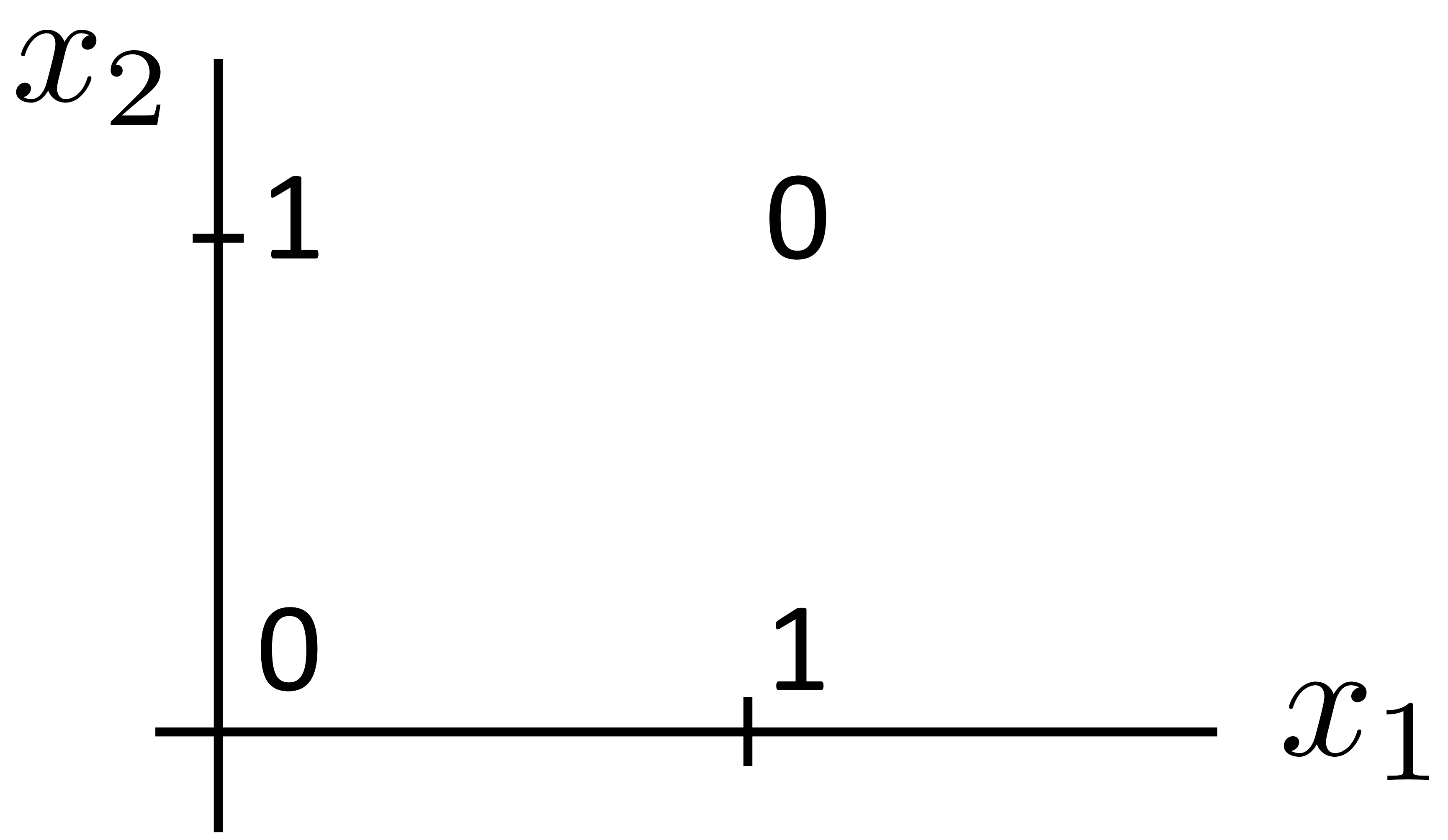
I[contains *not*], I[contains *good*]

Neural Networks: XOR

- Let's see how we can use neural nets to learn a simple nonlinear function
- Inputs $x_1, \; x_2$

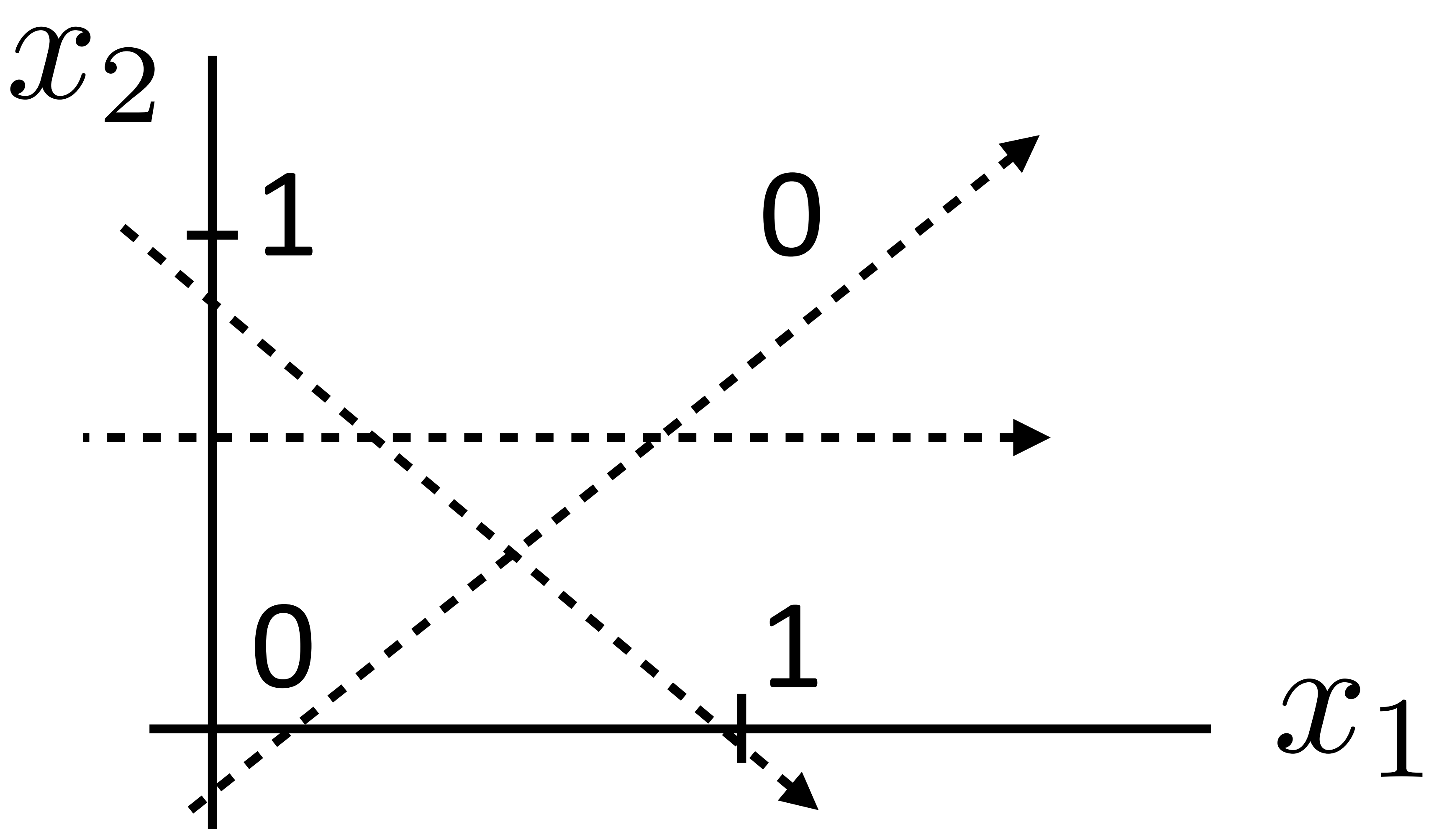
  $(\text{generally } \mathbf{x} = (x_1, \ldots, x_m))$
- Output y

  $(\text{generally } \mathbf{y} = (y_1, \ldots, y_n))$



x_1	x_2	$y = x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Neural Networks: XOR



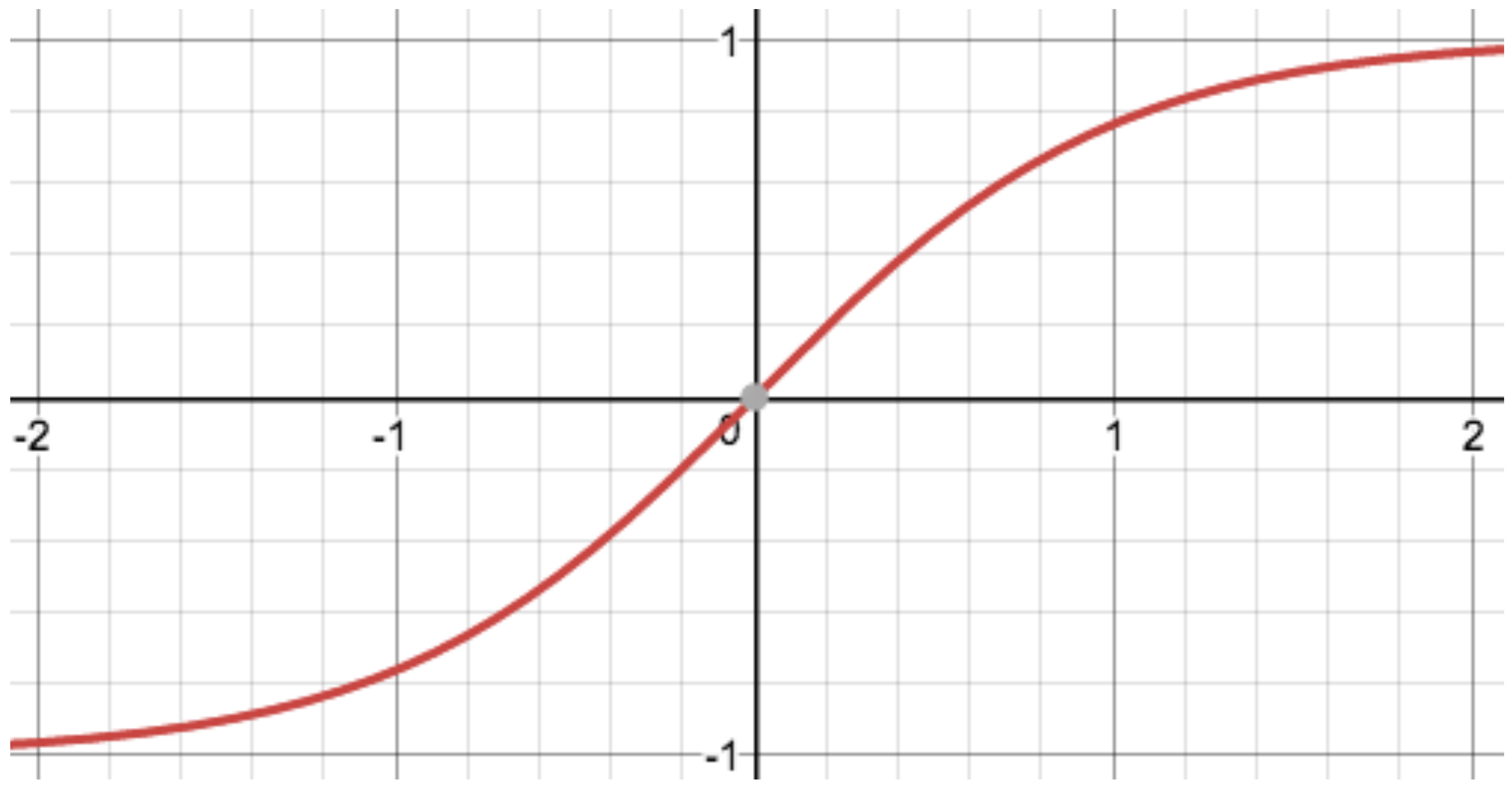
$$y = a_1 x_1 + a_2 x_2$$ ✗

$$y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)$$ ✓

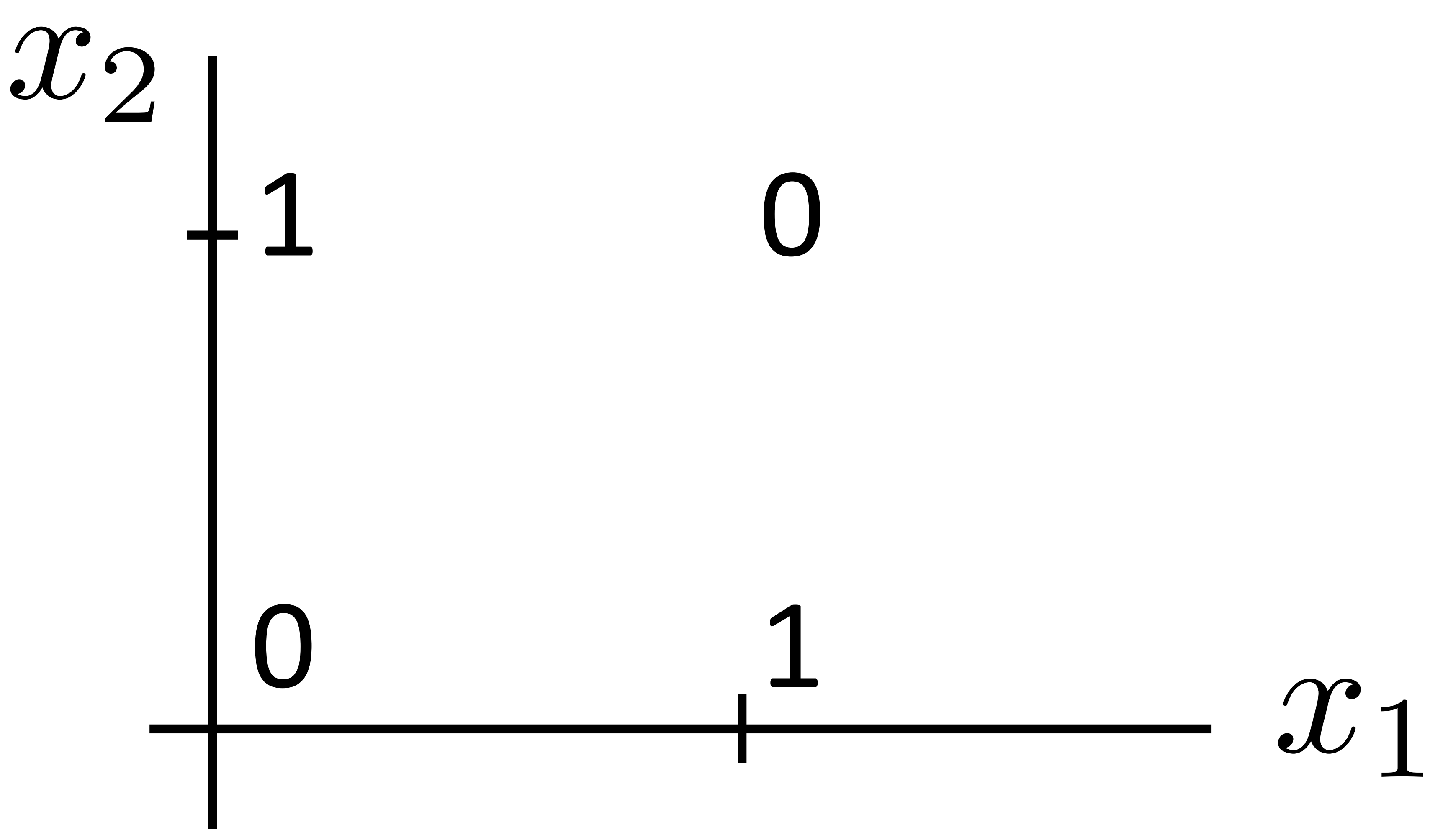
"or"

(looks like action potential in neuron)

x_1	x_2	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0



Neural Networks: XOR



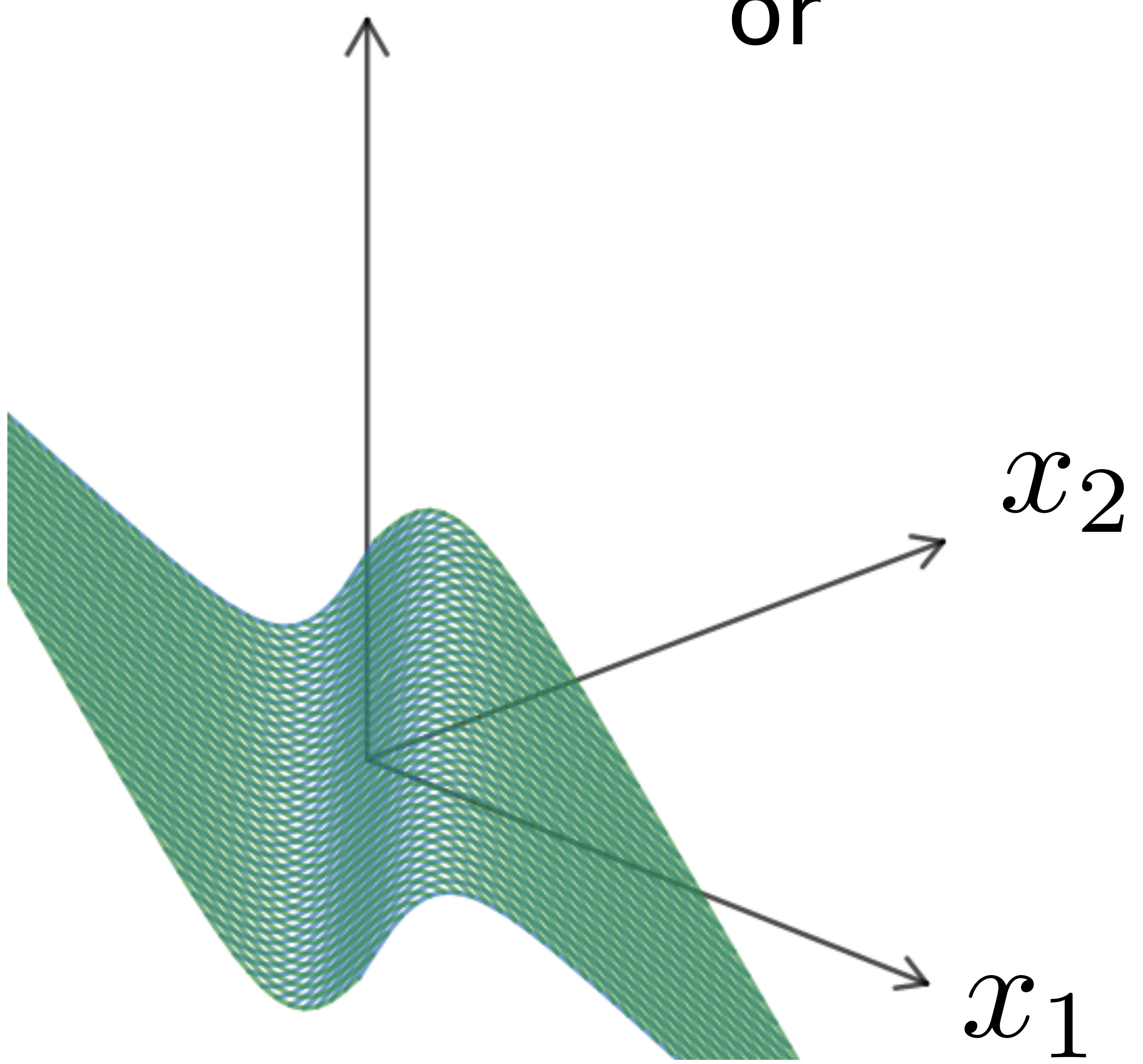
x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

$$y = a_1 x_1 + a_2 x_2$$ X

$$y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)$$ ✓

$$y = -x_1 - x_2 + 2 \tanh(x_1 + x_2)$$

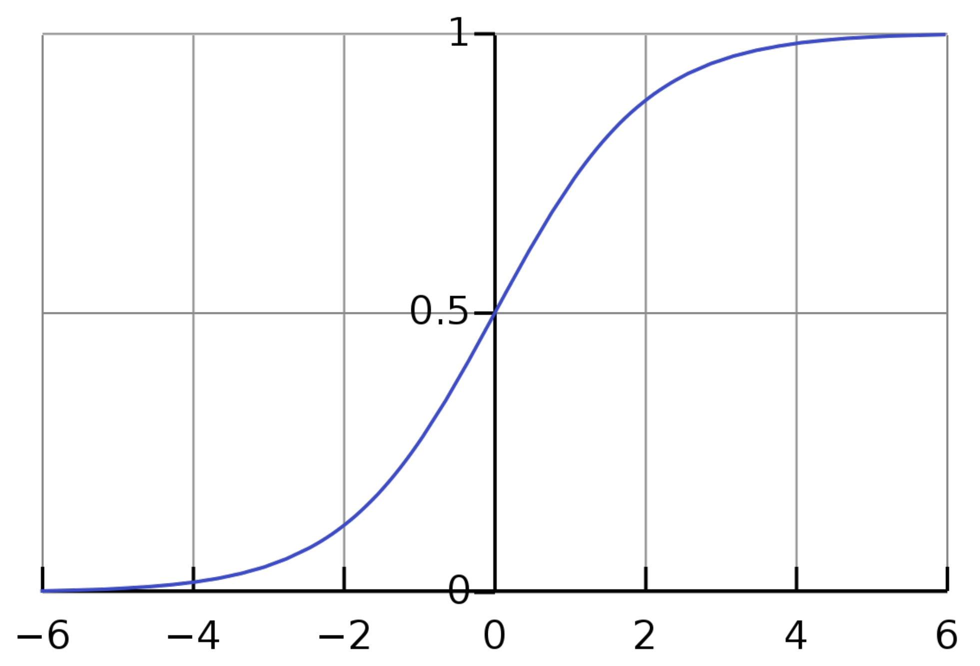
"or"

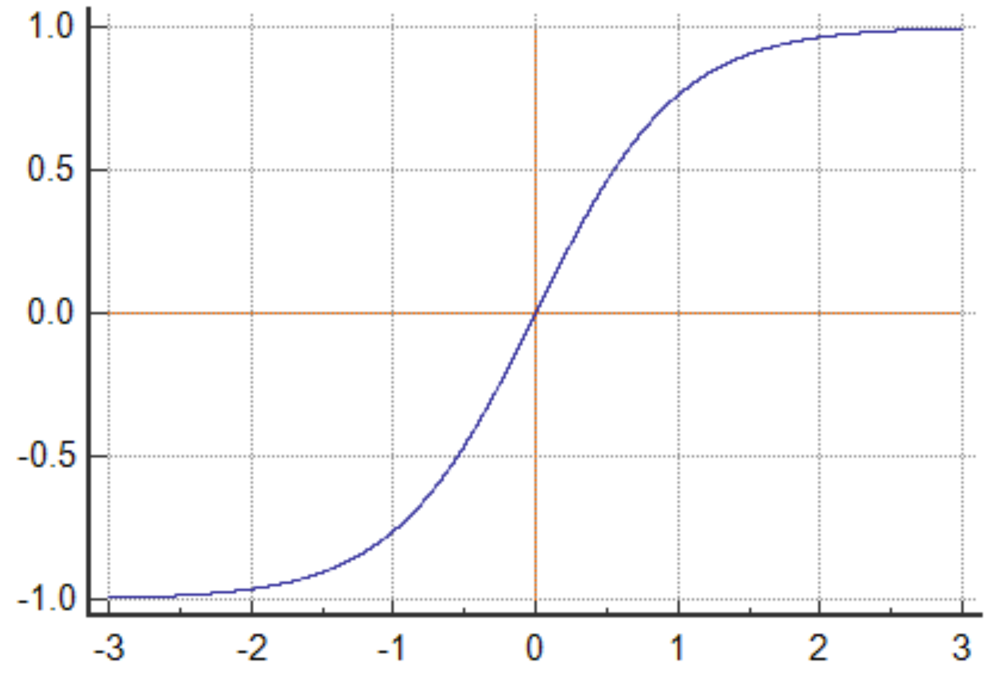


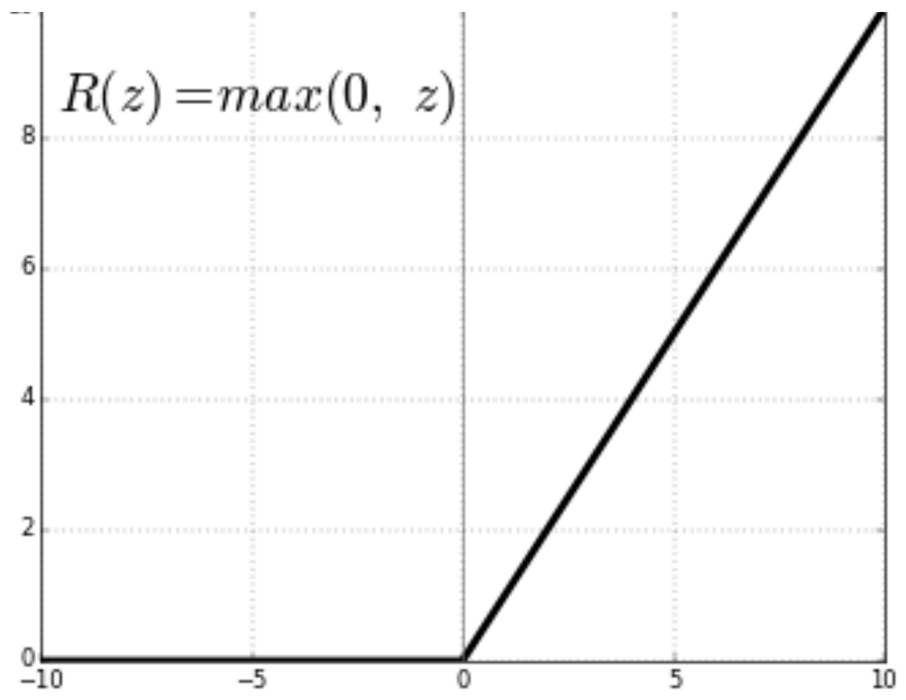
Non linear activation function

- Entry-wise function: $g : \mathbb{R} \to \mathbb{R}$

sigmoid	tanh	ReLU (rectified linear unit)
$g(z) = \frac{1}{1+e^{-z}}$	$g(z) = \frac{e^{2z}-1}{e^{2z}+1}$	$g(z) = \max(0, z)$







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

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

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

Advantages of ReLU?

Neural Networks

Linear model: $y = \mathbf{w} \cdot \mathbf{x} + b$

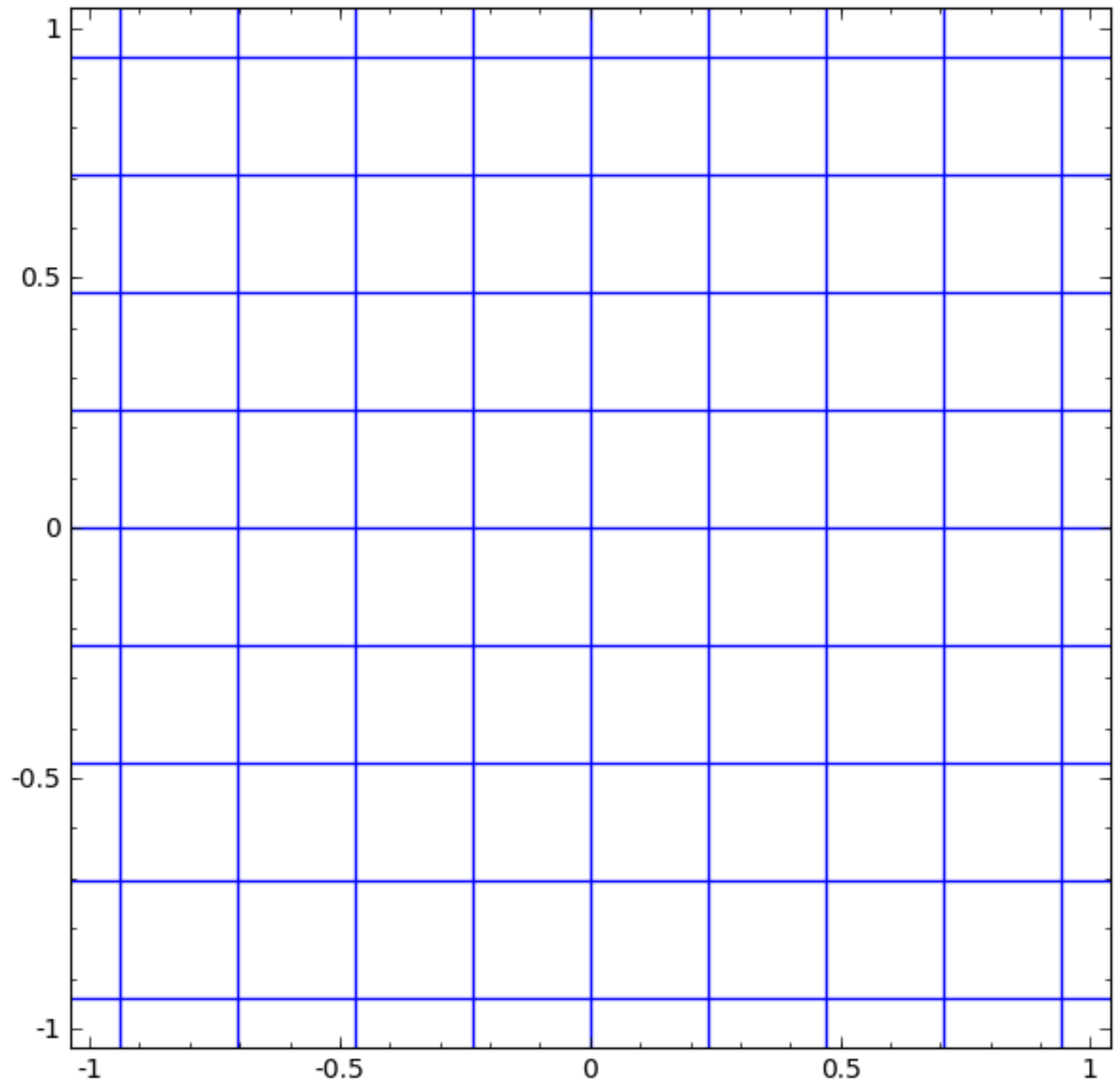
$$y = g(\mathbf{w} \cdot \mathbf{x} + b)$$

$$\mathbf{y} = g(\mathbf{W}\mathbf{x} + \mathbf{b})$$

Nonlinear function

Linear transformation

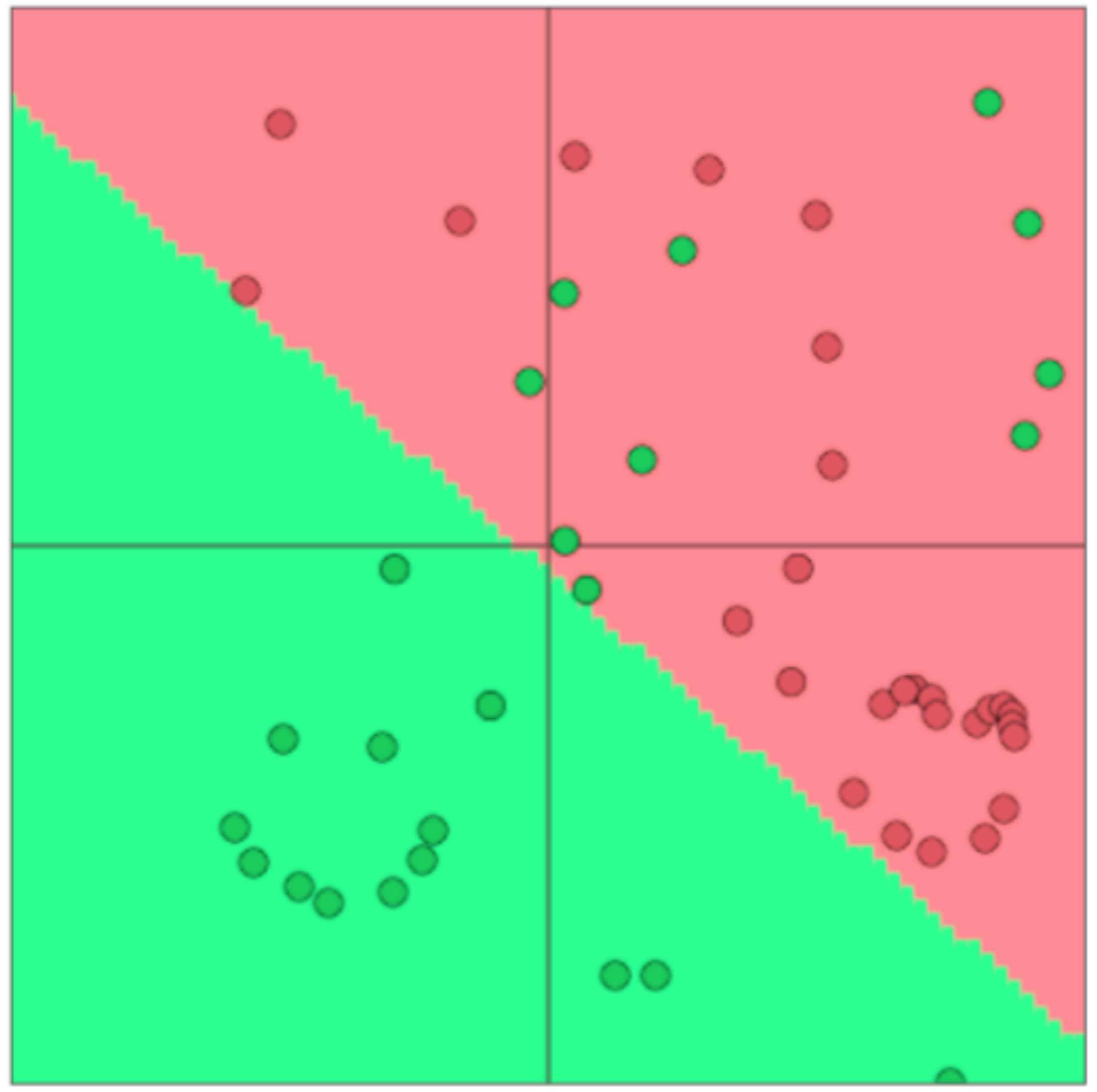
Shift

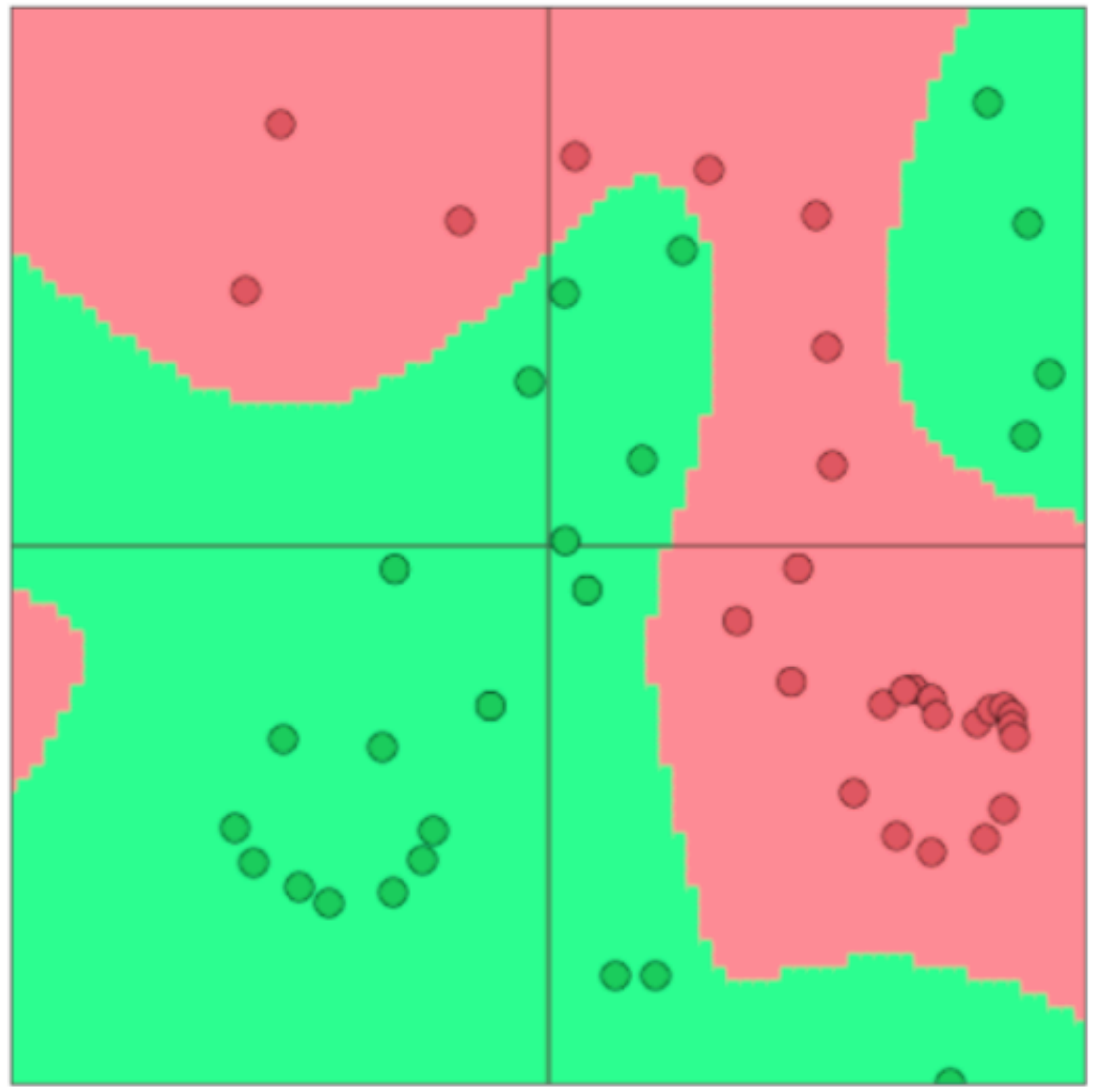


Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Non-linearity & Deep network

- Neural network can learn much more complex functions and nonlinear decision boundaries

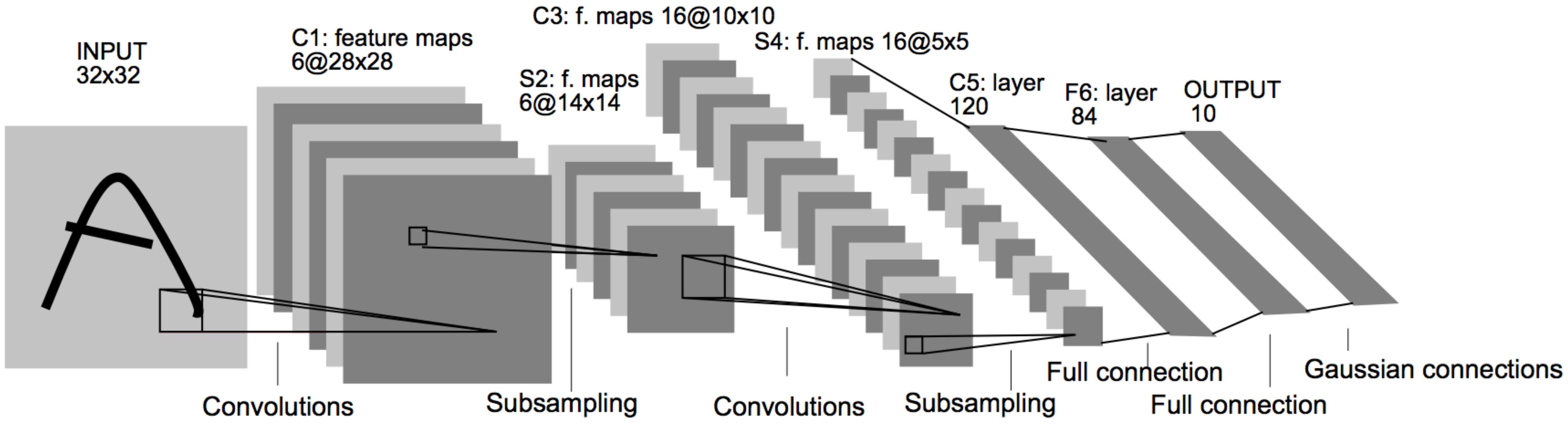




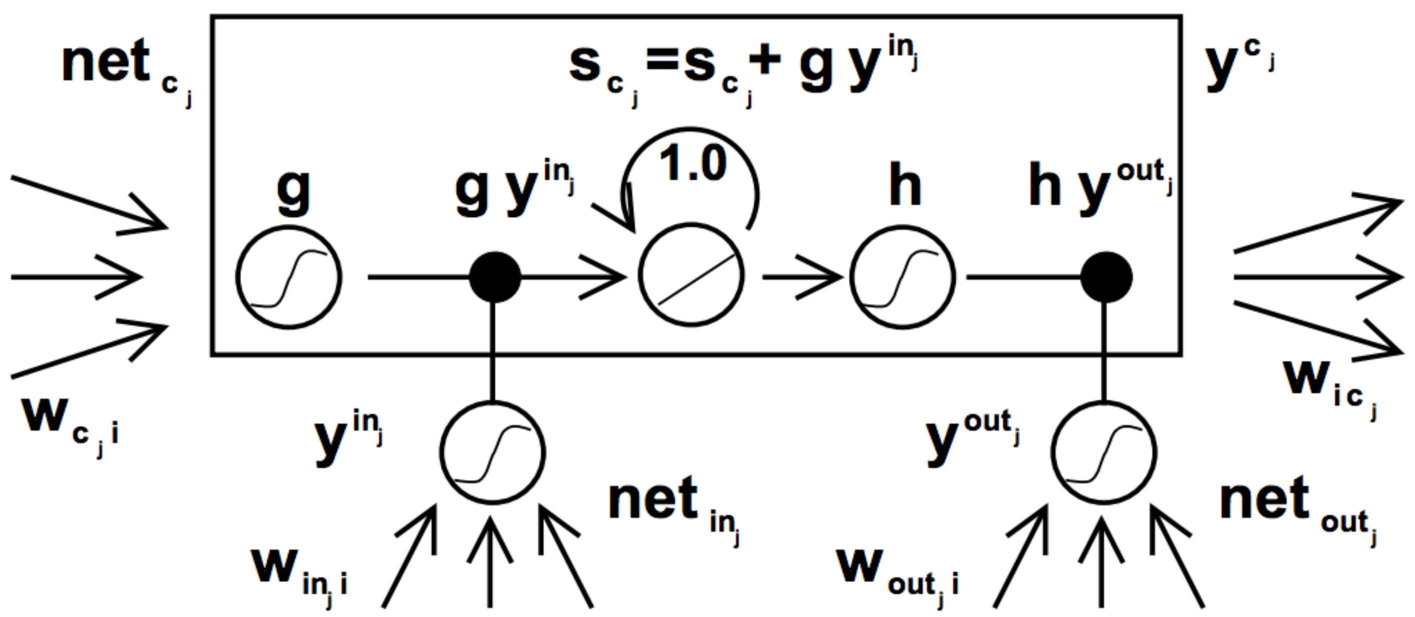
Brief History of Neural Network in NLP

History: Early Times

- Convnets: applied to digit recognition by LeCun in 1998



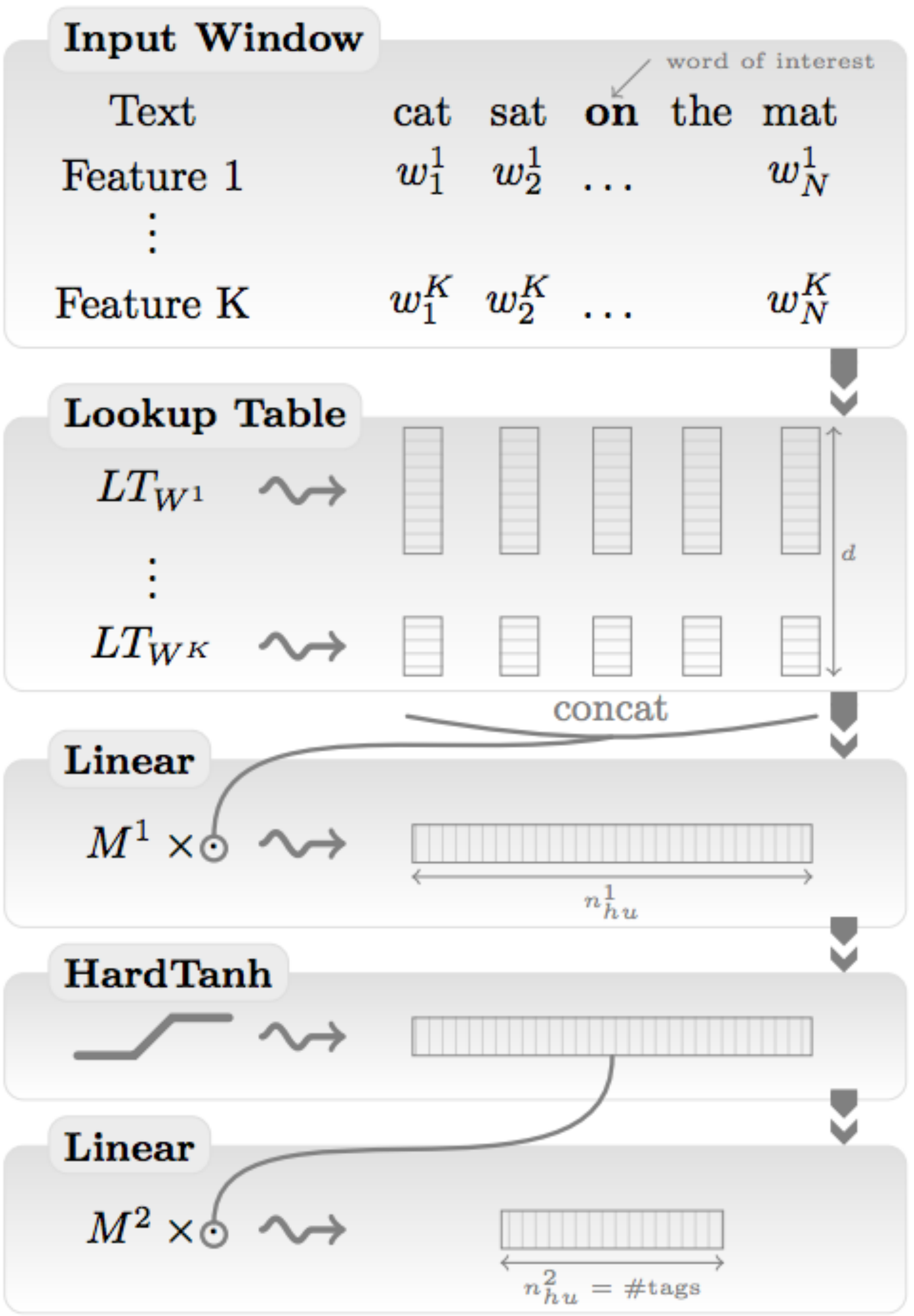
- Long short term memory network (LSTM): Hochreiter and Schmidhuber (1997)



- Henderson (2003): applied to nlp task (parsing) not SOTA

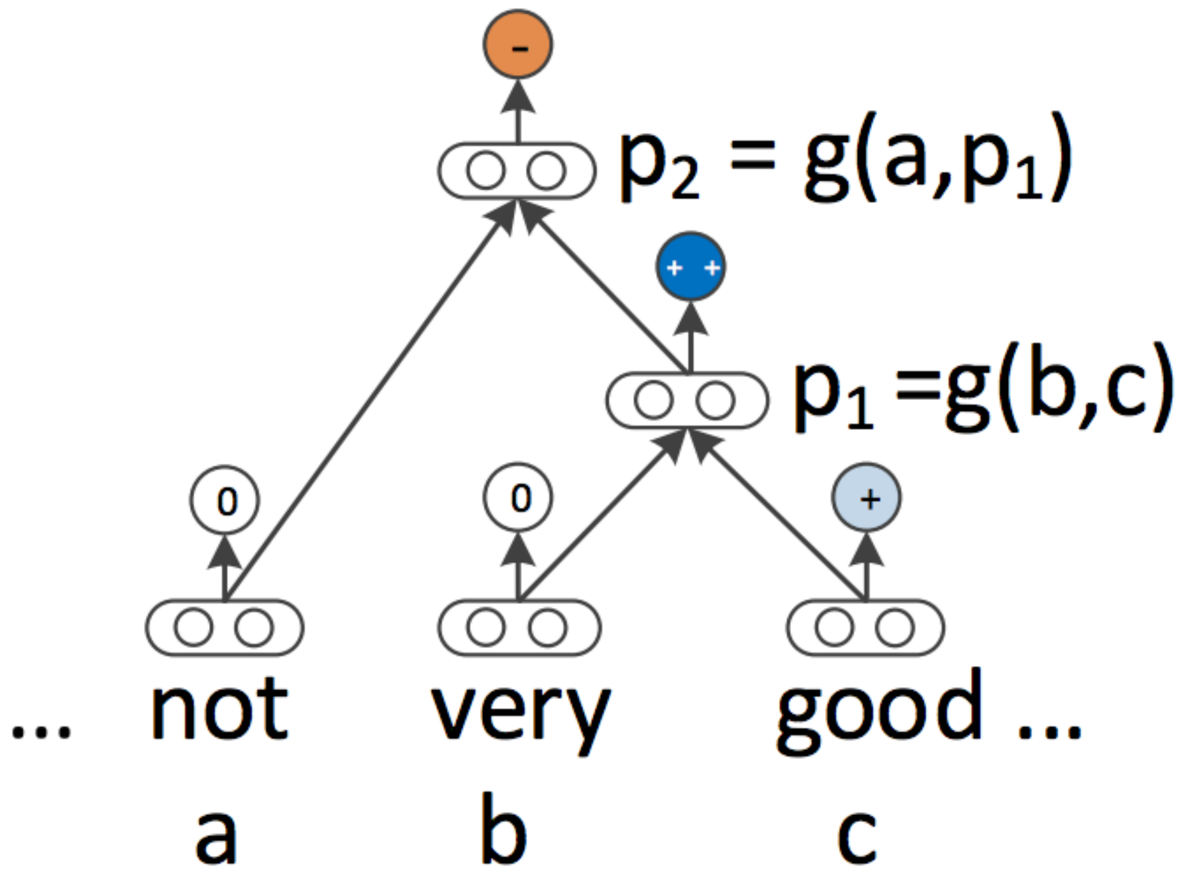
2008-2013: A glimmer of light…

- Collobert and Weston 2011: "NLP (almost) from scratch"
  - Feedforward neural nets induce features



- Krizhevskey et al. (2012): AlexNet for vision (image classification)

- Socher 2011-2014: tree-structured recursive neural networks working okay (for sentiment classification)



2014: Stuff starts working

- Kim (2014) + Kalchbrenner et al. (2014): sentence classification / sentiment
  - Applying convolutional NN
- Sutskever et al. + Bahdanau et al.: Neural MT
  - LSTMs
- Chen and Manning transition-based dependency parser
  - Feedforward neural network
- 2015: explosion of neural nets 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 / Adadelta / Adam) work best out-of-the-box
  - Regularization: dropout is pretty helpful
  - Computers not big enough: can't run for enough iterations

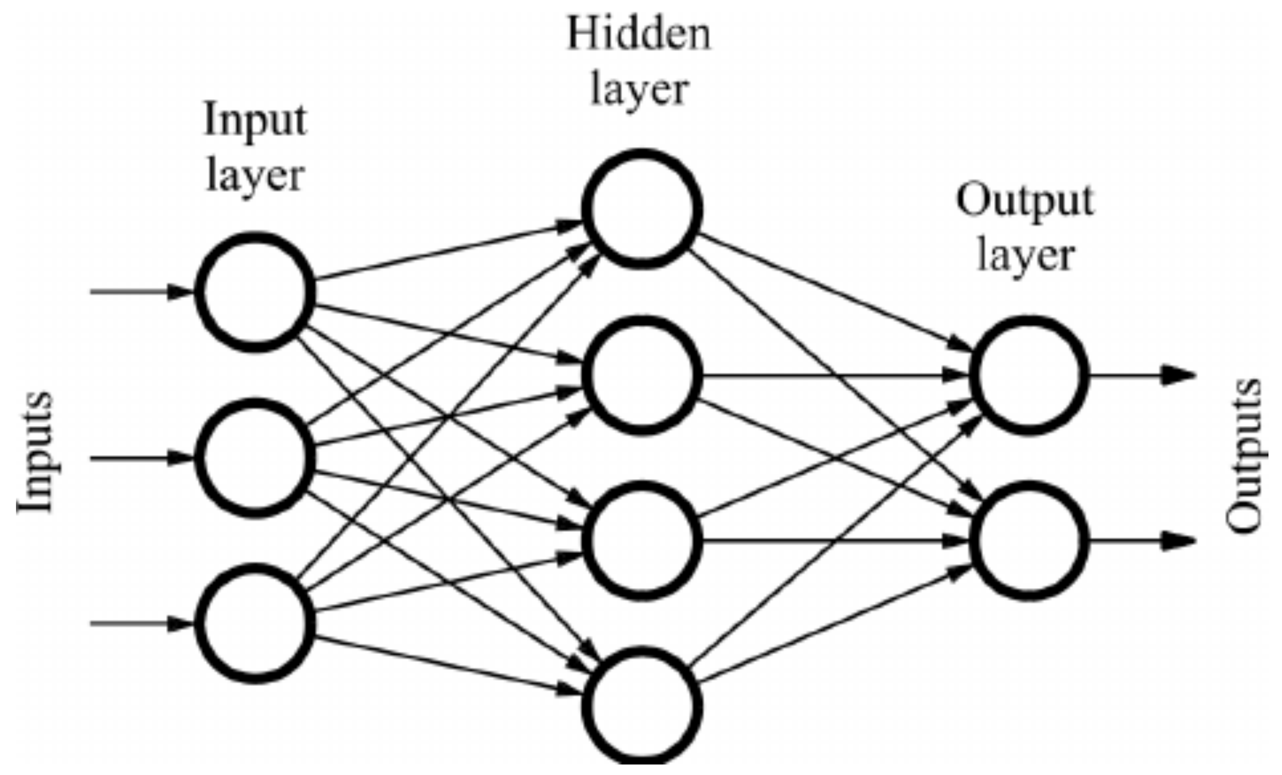
- Inputs: need word representations to have the right continuous semantics

The “Promise”

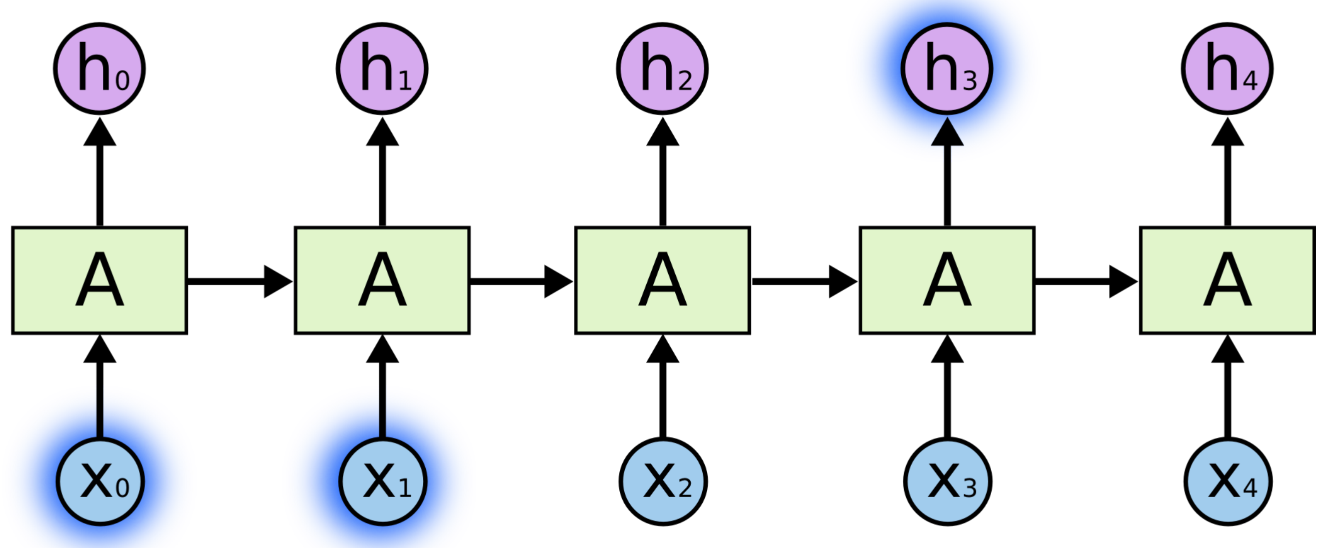
- Most ML works with human-designed feature representations
- ML becomes optimizing weights
- Representation Learning: automatically learn good features and representations
- Deep Learning: attempts to learn multilevel of representation of increasing complexity / abstraction

Neural Networks in NLP

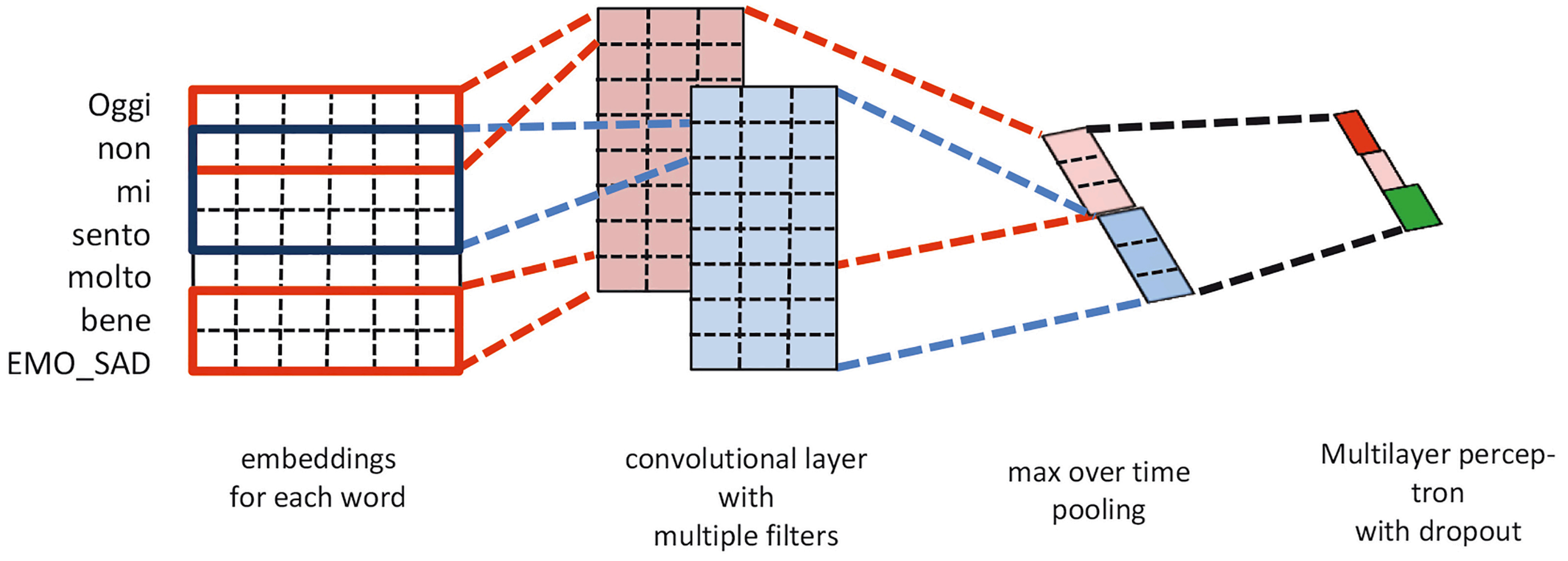
Feed-forward NNs



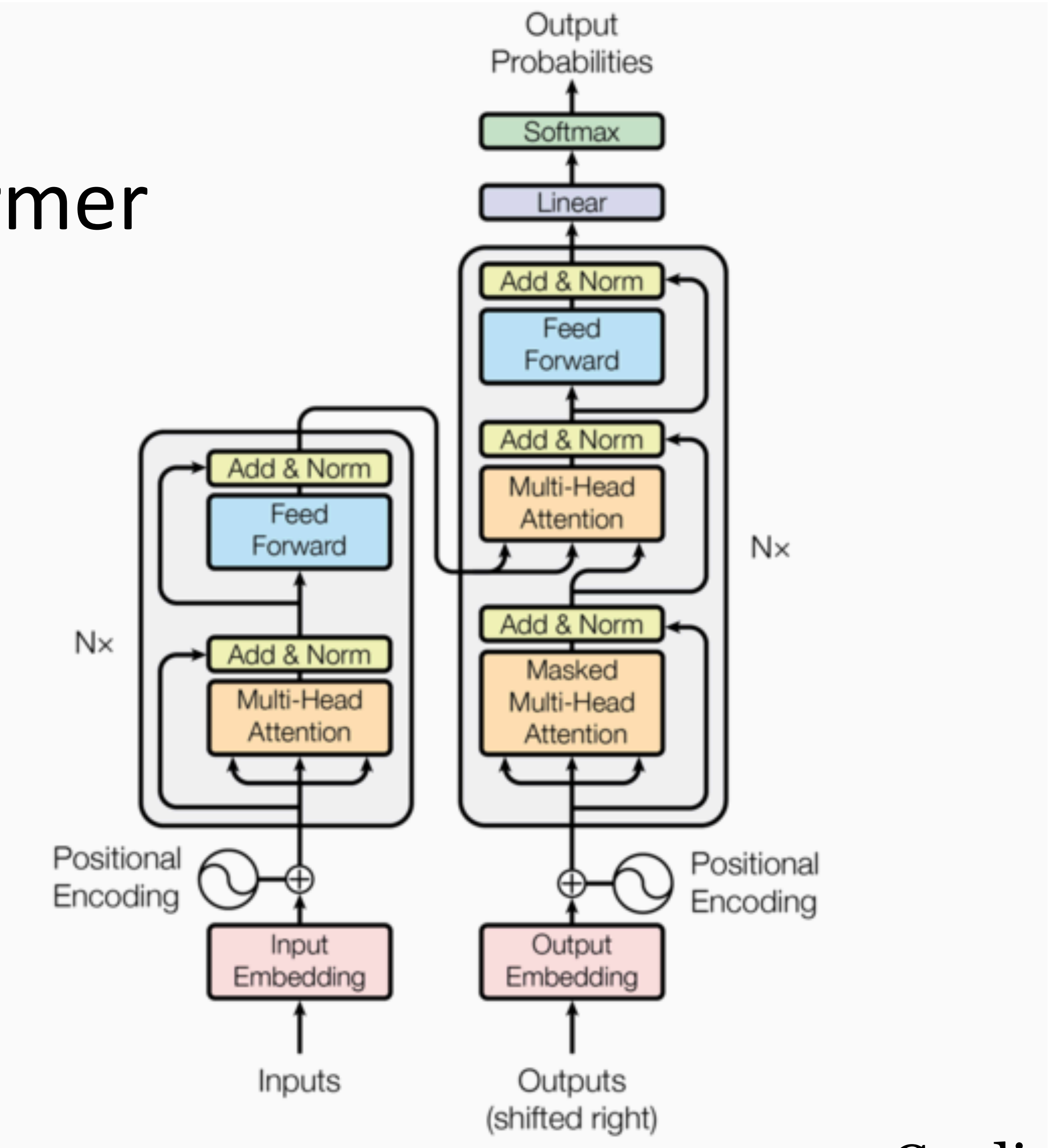
Recurrent NNs



Convolutional NNs



Transformer



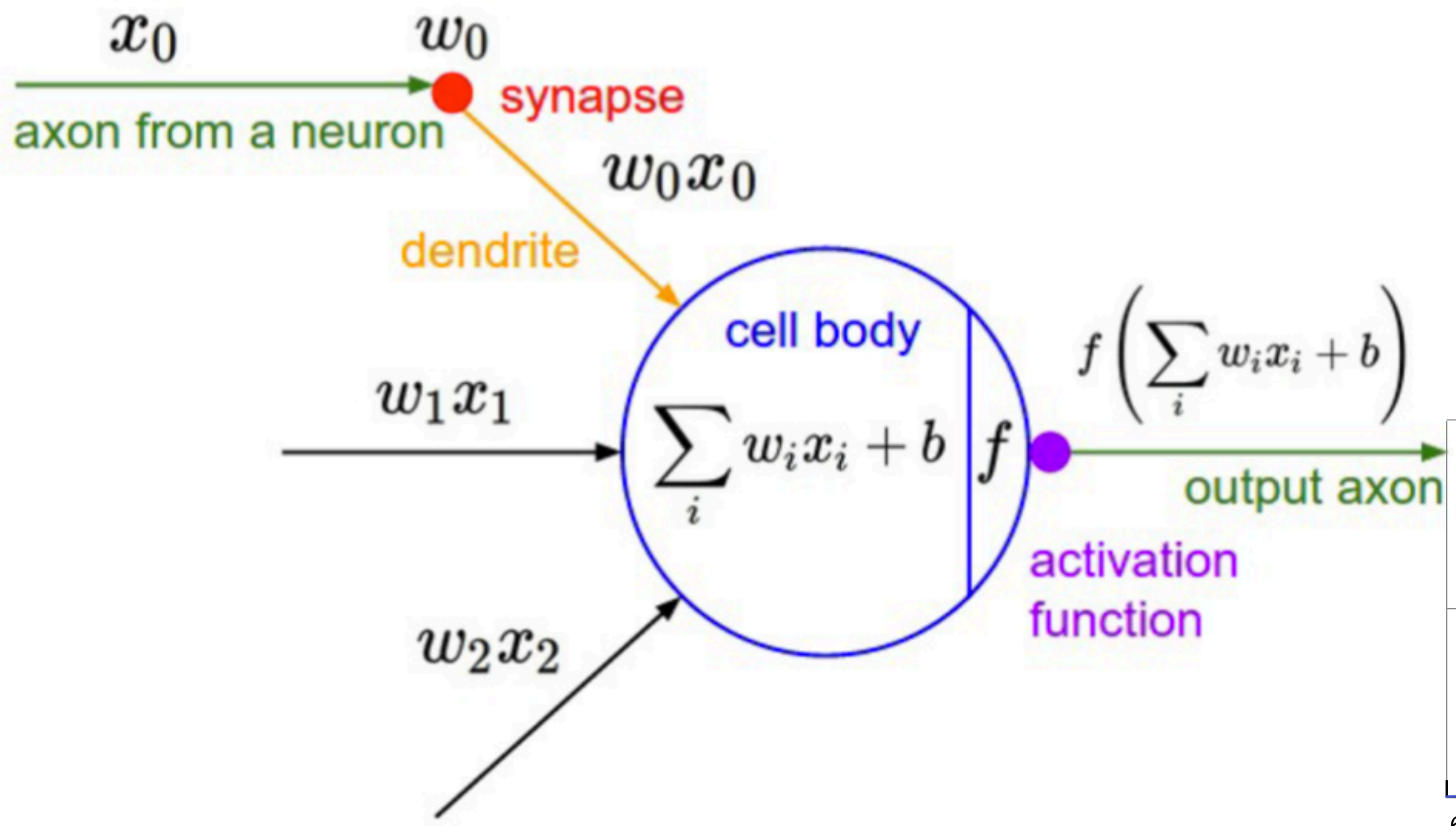
Always coupled with word embeddings…

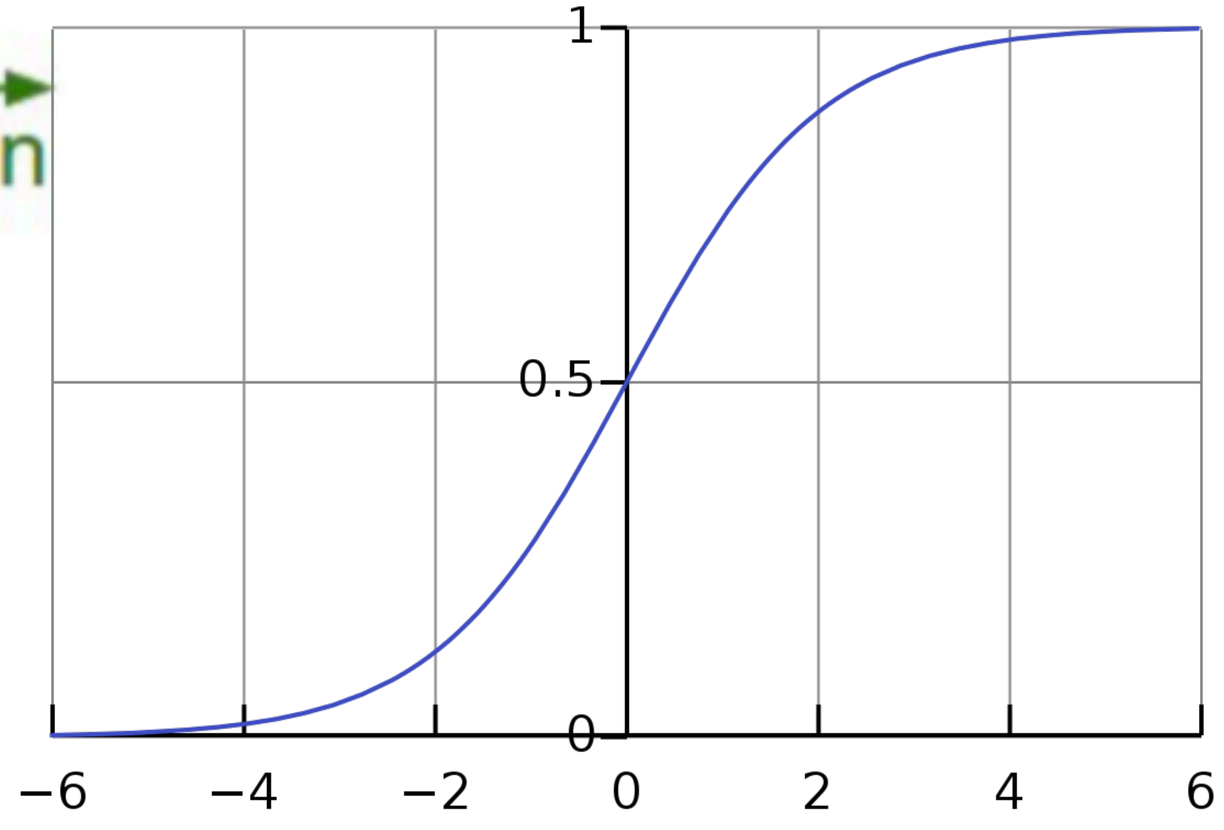
Credits: Princeton NLP course

Feedforward Networks

Sigmoid Neuron in Neural Network

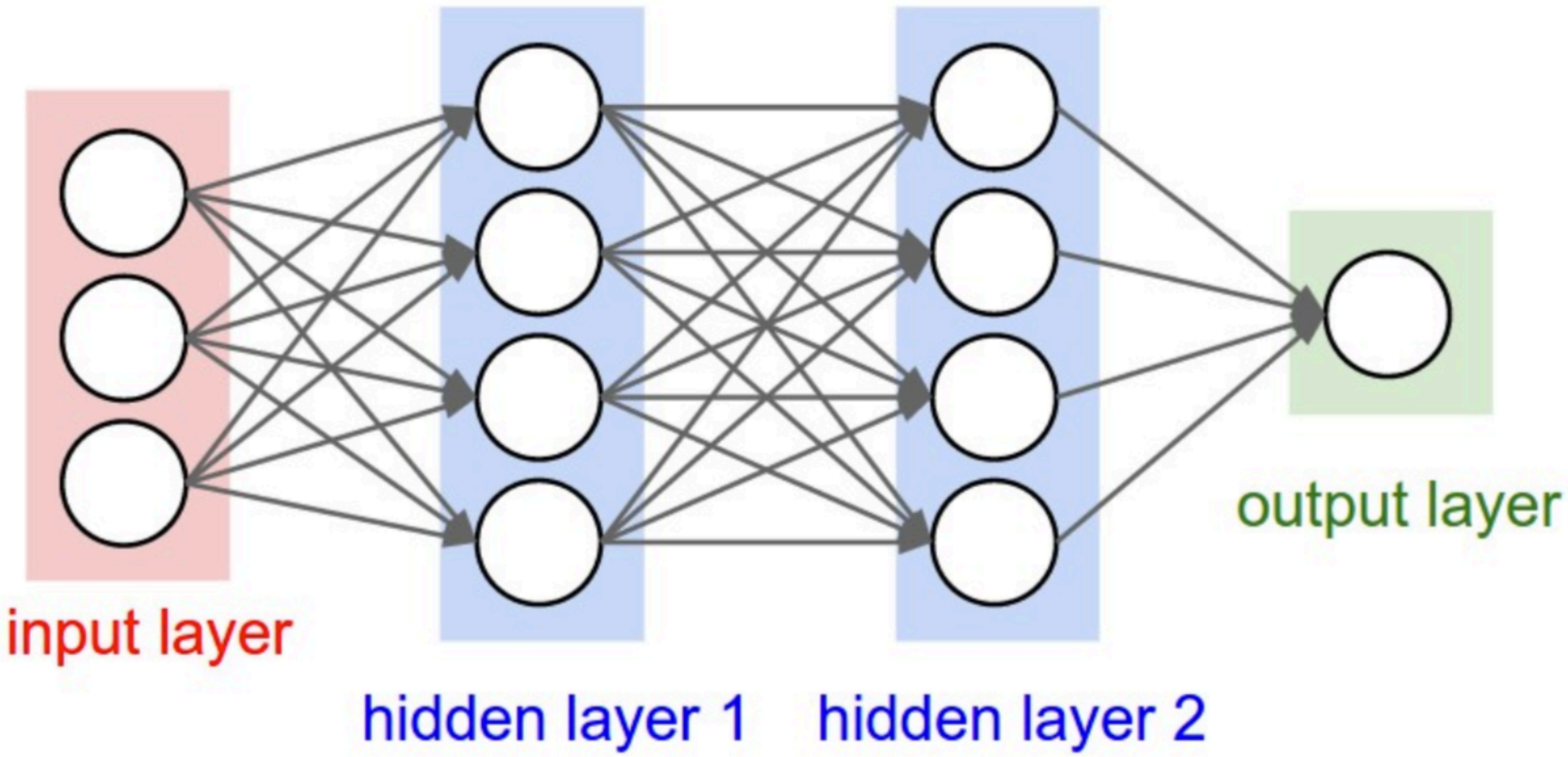
- A single neuron is a computational unit
- The neuron multiples each input by its weight, sums them, applied a **nonlinear function** to the result, and passes it to its output.





A neural network

- If we feed inputs through multiple logistic regression functions, then we can construct a output vector…
- which we can feed into another logistic regression function as an input.



Recap: Multinomial Logistic Regression

$$P(y|\mathbf{x}) = \frac{\exp(\mathbf{w}_y^\top f(\mathbf{x}))}{\sum_{y' \in \mathcal{Y}} \exp(\mathbf{w}_{y'}^\top f(\mathbf{x}))}$$

- Three classes, "different weights"

$$\begin{matrix} \mathbf{w}_1^\top f(\mathbf{x}) \\ \mathbf{w}_2^\top f(\mathbf{x}) \\ \mathbf{w}_3^\top f(\mathbf{x}) \end{matrix} = \begin{matrix} -1.1 \\ 2.1 \\ -0.4 \end{matrix} \xrightarrow{\text{softmax}} \begin{matrix} 0.036 \\ 0.89 \\ 0.07 \end{matrix} \quad \text{class probs}$$

- Softmax operation = "exponentiate and normalize"
- We write this as: $\mathrm{softmax}(W f(\mathbf{x}))$

Logistic Regression with NNs

$$P(y|\mathbf{x}) = \frac{\exp(\mathbf{w}_y^\top f(\mathbf{x}))}{\sum_{y' \in \mathcal{Y}} \exp(\mathbf{w}_{y'}^\top f(\mathbf{x}))}$$

- Single scalar probability

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(W f(\mathbf{x}))$$

- Weight vector per class;
  W is [num classes x num feats]

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(W g(V f(\mathbf{x})))$$

- Now one hidden layer