CS5740: Natural Language Processing

Neural Networks

Instructor: Yoav Artzi

Slides adapted from Dan Klein, Dan Jurafsky, Chris Manning, Michael Collins, Luke Zettlemoyer, Yejin Choi, and Slav Petrov

Overview

- Introduction to Neural Networks
- Word representations
- NN Optimization tricks

Some History

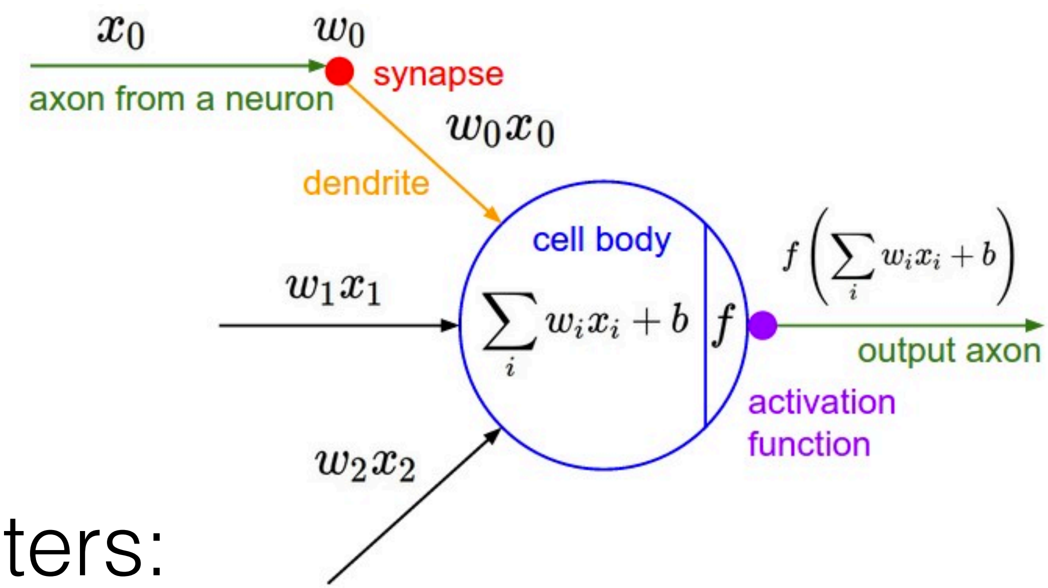
- Neural network algorithms date to the 80's
  - Originally inspired by early neuroscience
- Historically slow, complex, and unwieldy
- Now: term is abstract enough to encompass almost any model – but useful!
- Dramatic shift in last 3-4 years away from MaxEnt (linear, convex) to "neural net" (non-linear architecture, non-convex)

The “Promise”

- Most ML works well because of human-designed representations and input features
- ML becomes just optimizing weights
- Representation learning attempts to automatically learn good features and representations
- Deep learning attempts to learn multiple levels of representation of increasing complexity/abstraction

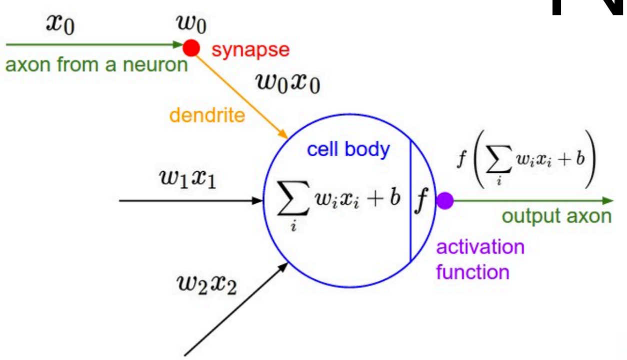
Neuron

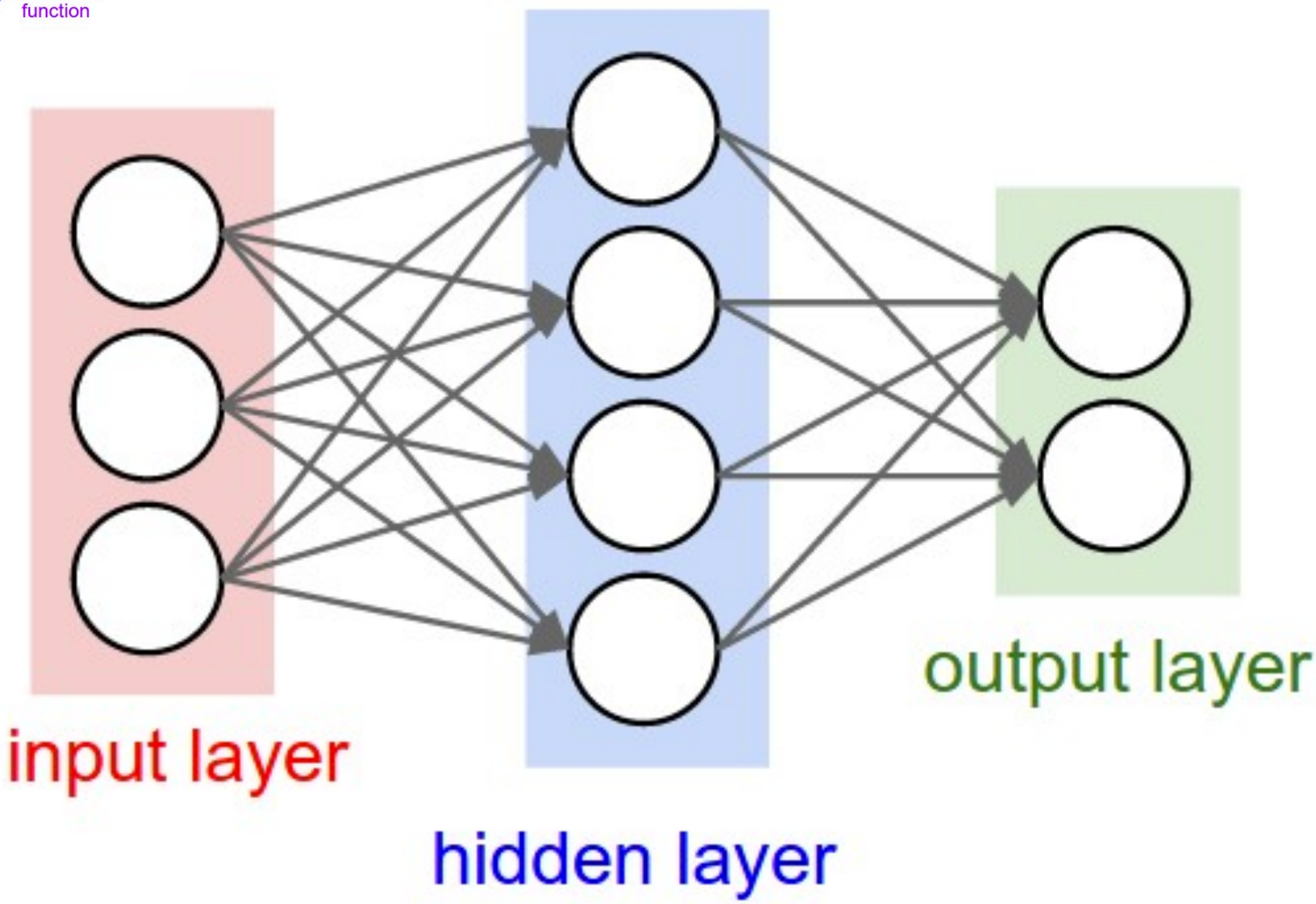
- Neural networks comes with their terminological baggage



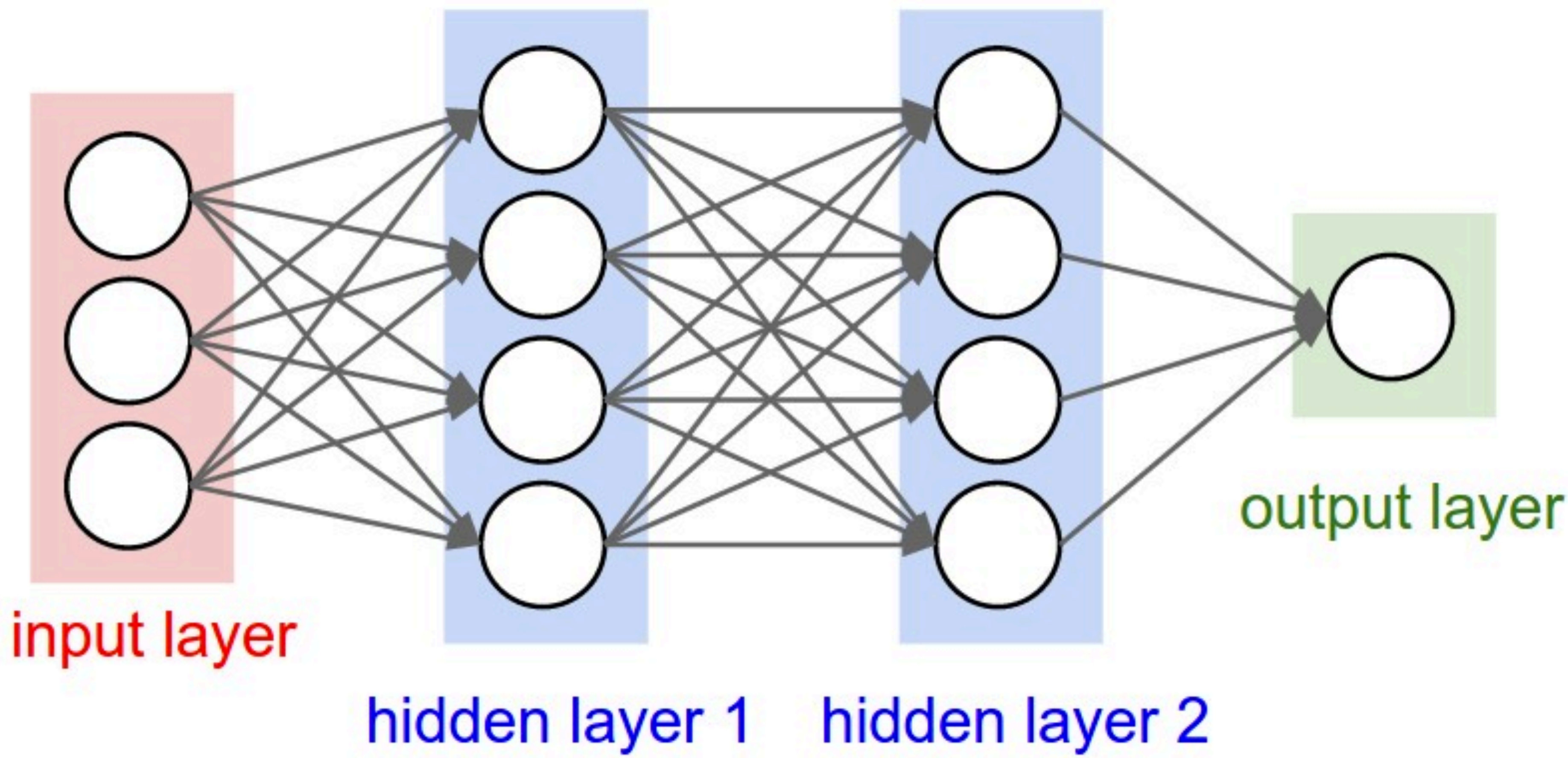
- Parameters:
  - Weights: w_i and b
  - Activation function
- If we drop the activation function, reminds you of something?

Neural Network

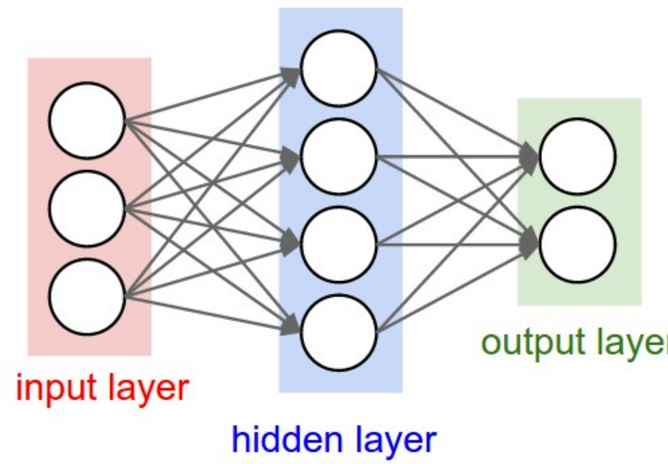




Neural Network



Matrix Notation



$$\mathbf{a} \in \mathbb{R}^{1\times 3}$$

$$\left.\begin{array}{r} \mathbf{W}' \in \mathbb{R}^{3\times 4} \\ \mathbf{W}'' \in \mathbb{R}^{4\times 2} \\ \mathbf{b}' \in \mathbb{R}^{1\times 4} \\ \mathbf{b}'' \in \mathbb{R}^{1\times 2} \end{array}\right\} \text{Learned parameters}$$

$$\mathbf{h} \in \mathbb{R}^{1\times 4}$$

$$\mathbf{o} \in \mathbb{R}^{1\times 2}$$

$$\mathbf{h} = \mathbf{a}\mathbf{W}' + \mathbf{b}'$$

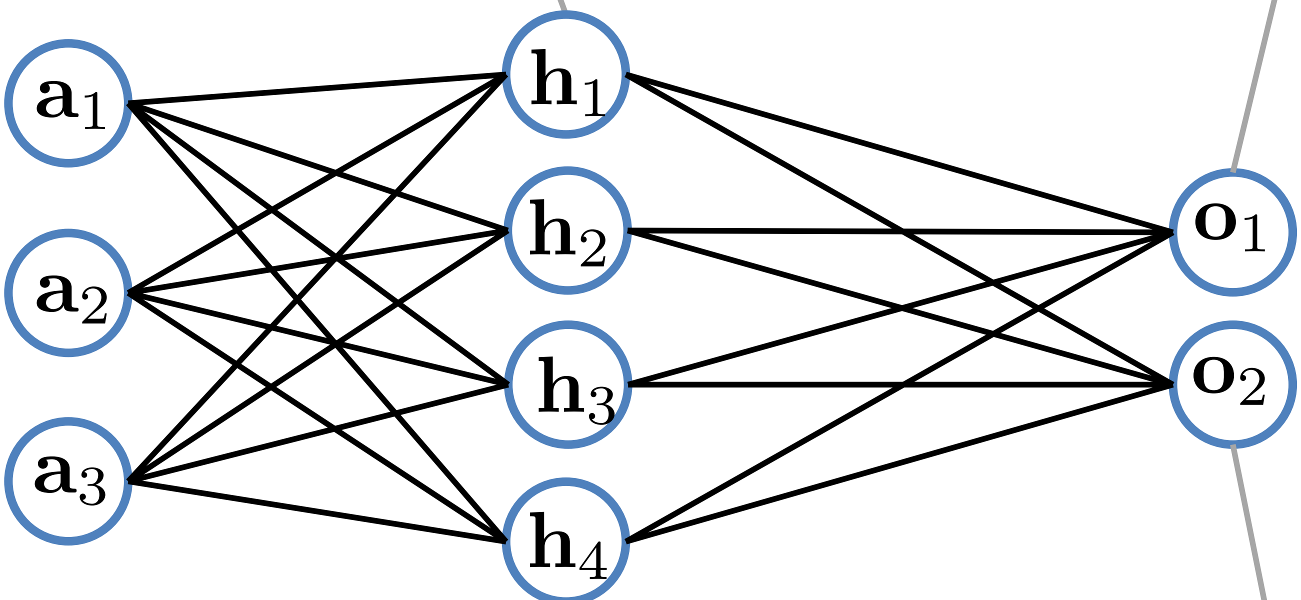
$$\begin{aligned} \mathbf{o} &= \mathbf{h}\mathbf{W}'' + \mathbf{b}'' \\ &= (\mathbf{a}\mathbf{W}' + \mathbf{b}')\mathbf{W}'' + \mathbf{b}'' \end{aligned}$$

No activation/non-linearity function

Matrix Notation

$$\mathbf{h}_1 = \mathbf{a}_1\mathbf{W}'_{11} + \mathbf{a}_2\mathbf{W}'_{21} + \mathbf{a}_3\mathbf{W}'_{31} + \mathbf{b}'_1$$

$$\mathbf{o}_1 = \mathbf{h}_1\mathbf{W}''_{11} + \mathbf{h}_2\mathbf{W}''_{21} + \mathbf{h}_3\mathbf{W}''_{31} + \mathbf{h}_4\mathbf{W}''_{41} + \mathbf{b}''_1$$



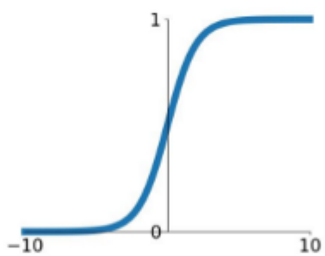
$$\mathbf{o}_1 = \mathbf{h}_1\mathbf{W}''_{12} + \mathbf{h}_2\mathbf{W}''_{22} + \mathbf{h}_3\mathbf{W}''_{32} + \mathbf{h}_4\mathbf{W}''_{42} + \mathbf{b}''_1$$

$$\mathbf{h}_2 = \mathbf{a}_1\mathbf{W}'_{14} + \mathbf{a}_2\mathbf{W}'_{24} + \mathbf{a}_3\mathbf{W}'_{34} + \mathbf{b}'_4$$

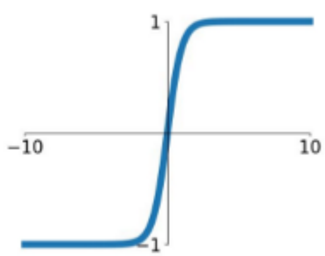
Activation Functions

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

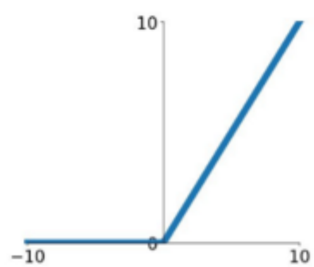
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



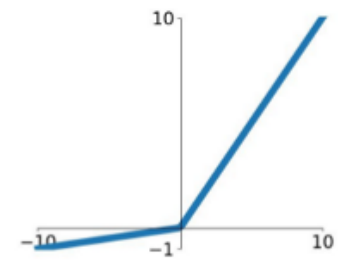
tanh
$\tanh(x)$



ReLU
$\max(0, x)$

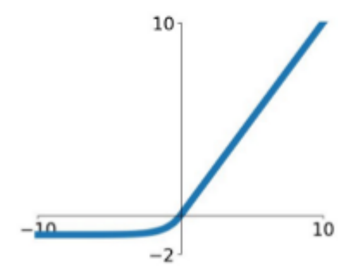


Leaky ReLU
$\max(0.1x, x)$



Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neurons and Other Models

- A single neuron is a perceptron
- Strong connection to MaxEnt – how?

From MaxEnt to Neural Nets

- Vector form MaxEnt:

$$P(y|x;w) = \frac{e^{w^\top \phi(x,y)}}{\sum_{y'} e^{w^\top \phi(x,y')}}$$

- For two classes:

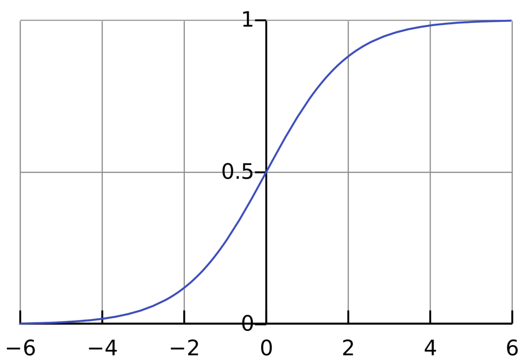
$$P(y_1|x;w) = \frac{e^{w^\top \phi(x,y_1)}}{e^{w^\top \phi(x,y_1)} + e^{w^\top \phi(x,y_2)}}$$

$$= \frac{e^{w^\top \phi(x,y_1)}}{e^{w^\top \phi(x,y_1)} + e^{w^\top \phi(x,y_2)}} \frac{e^{-w^\top \phi(x,y_1)}}{e^{-w^\top \phi(x,y_1)}}$$

$$= \frac{1}{1 + e^{w^\top (\phi(x,y_2) - \phi(x,y_2))}}$$

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

$$z = \phi(x, y_1) - \phi(x, y_2)$$



Logisitc Function (sigmoid)

From MaxEnt to Neural Nets

- Vector form MaxEnt:

$$P(y|x;w) = \frac{e^{w^\top \phi(x,y)}}{\sum_{y'} e^{w^\top \phi(x,y')}}$$

- For two classes:

$$P(y_1|x;w) = \frac{1}{1+e^{-w^\top z}} = f(w^\top z)$$

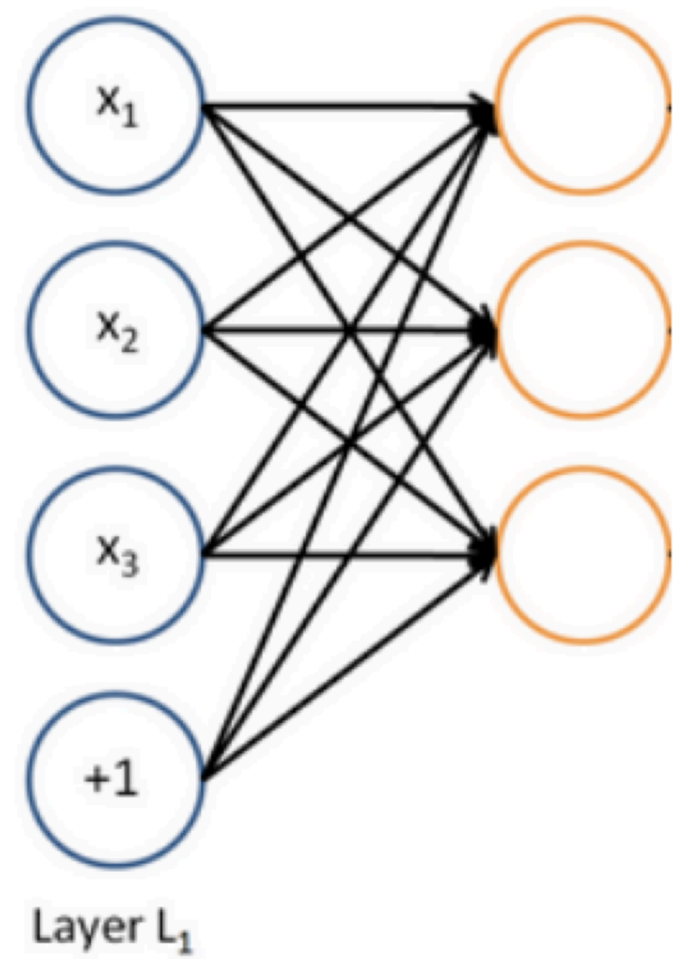
- Neuron:
  - Add an “always on” feature for class prior → bias term (b)

$$h_{w,b}(z) = f(w^\top z + b)$$

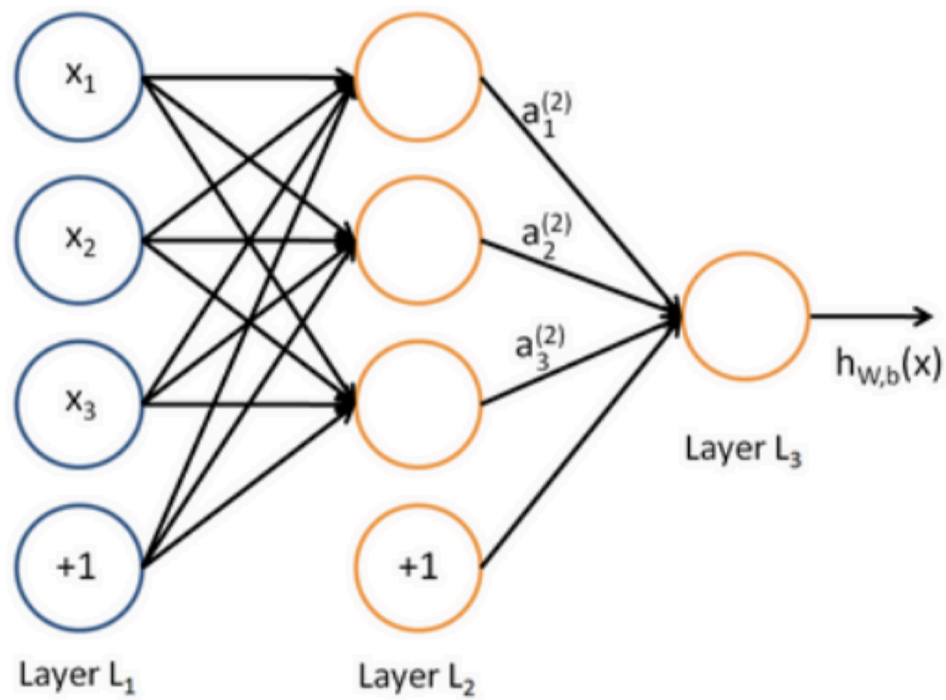
$$f(u) = \frac{1}{1+e^{-u}}$$

Neural Net = Several MaxEnt Models

- Feed a number of MaxEnt models → vector of outputs
- And repeat …



Neural Net = Several MaxEnt Models



- But: how do we tell the hidden layer what to do?
  - Learning will figure it out

How to Train?

- No hidden layer:
  - Supervised
  - Just like MaxEnt
- With hidden layers:
  - Latent units → not convex
  - What do we do?
    - Back-propagate the gradient
    - About the same, but no guarantees

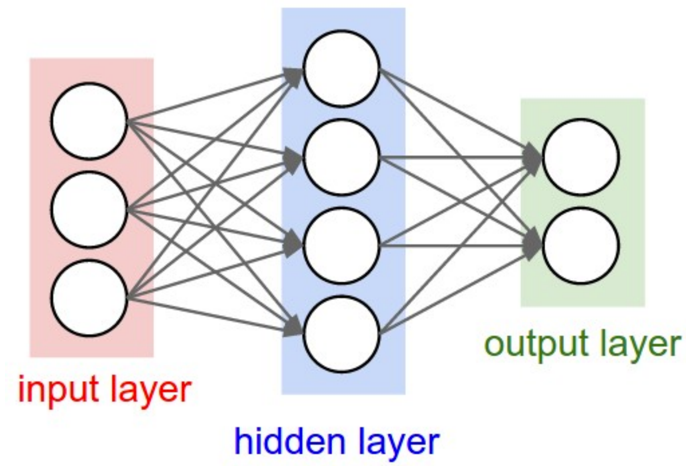
Probabilistic Output from Neural Nets

- What if we want the output to be a probability distribution over possible outputs?
- Normalize the output activations using **softmax**:

$$y = \mathrm{softmax}(\mathbf{o})$$

$$\mathrm{softmax}(\mathbf{o}_\mathrm{i}) = \frac{\exp(\mathbf{o}_\mathrm{i})}{\sum_{\mathrm{j}=1}^{\mathrm{k}} \exp(\mathbf{o}_\mathrm{j})}$$

  - Where $\boldsymbol{o}$ is the output layer
  - Usually: no non-linearity before softmax



Word Representations

- So far, atomic symbols:
  - “hotel”, “conference”, “walking”, “___ing”
- But neural networks take vector input
- How can we bridge the gap?
- One-hot vectors

hotel = [0 0 0 0 … 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
conference = [0 0 0 0 … 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]

  - Dimensionality:
    - Size of vocabulary
    - 20K for speech
    - 500K for broad-coverage domains
    - 13M for Google corpora

Word Representations

- One-hot vectors:

  hotel = [0 0 0 0 … 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
  conference = [0 0 0 0 … 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
  hotels = [0 0 0 0 … 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]

  – Problems?

  – Information sharing?

    - “hotel” vs. “hotels”

Word Embeddings

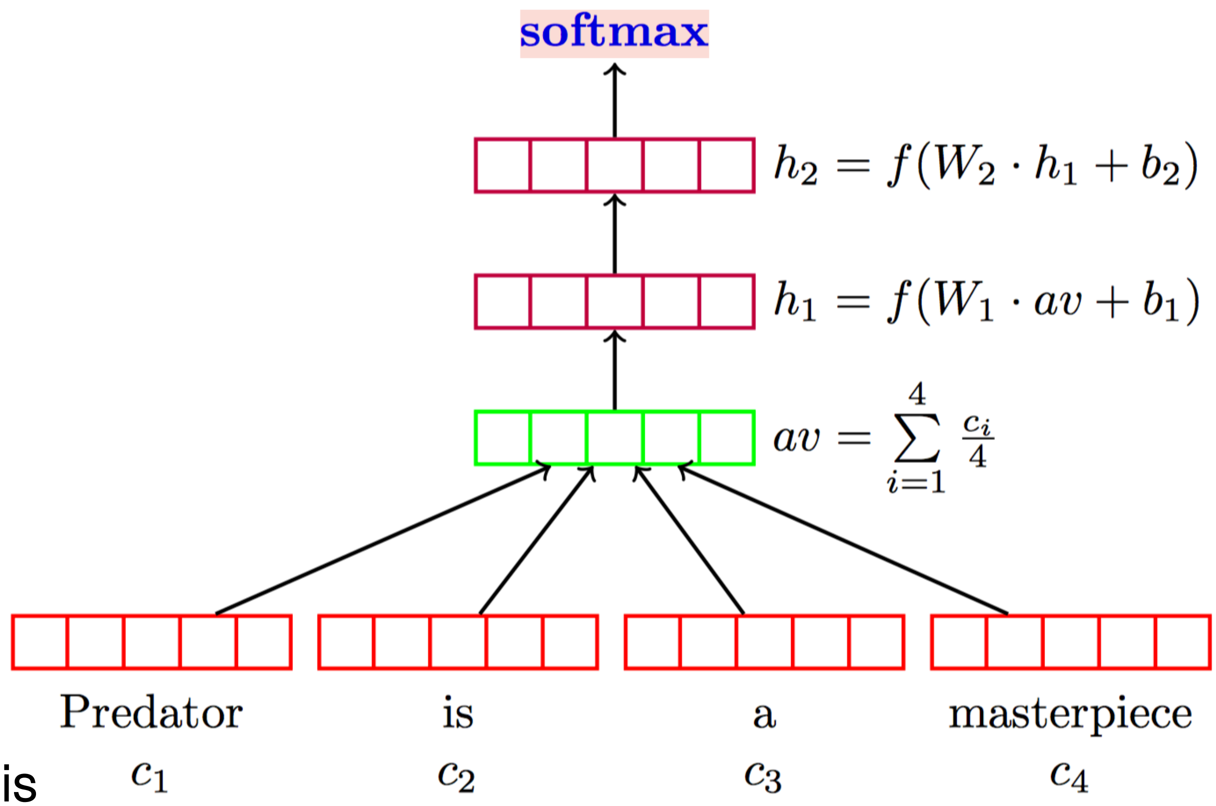
- Each word is represented using a dense low-dimensional vector
  - Low-dimensional << vocabulary size
- If trained well, similar words will have similar vectors
- How to train? What objective to maximize?
  - Soon …

Word Embeddings as Features

- Example: sentiment classification
  - very positive, positive, neutral, negative, very negative
- Feature-based models: bag of words
- Any good neural net architecture?
  - Concatenate all the vectors
    - Problem: different document → different length
  - Instead: sum, average, etc.

Neural Bag-of-words

Deep
Averaging
Networks



IMDB sentiment analysis

BOW + fancy smoothing + SVM	
NBOW + DAN	

[Iyyer et al. 2015; Wang and Manning 2012]

Classify Word Pair

- Goal: build a classifier that given a pair of words, classify if they are the full name of a person or not
- The classifier is a multi-layer-perceptron with three layers
- Make a drawing!
- Write the matrix notation, including dimensionality of matrices (choose as you wish, and as needed)
- What are the parameters to be learned

Inputs: x_l, x_r

Input vocabulary: $\mathcal{V}$

Embedding function: $\phi : \mathcal{V} \rightarrow \mathbb{R}^{256}$

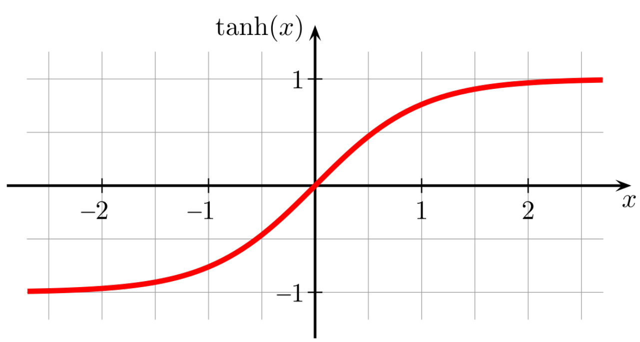
Weight matrices: $\mathbf{W}^1, \mathbf{W}^2, \mathbf{W}^3$

Bias vectors: $\mathbf{b}^1, \mathbf{b}^2, \mathbf{b}^3$

Operations: $2 \times \sigma : \mathbb{R}^* \rightarrow \mathbb{R}^*, 1 \times \text{softmax}$

Practical Tips

- Select network structure appropriate for the problem
  - Window vs. recurrent vs. recursive
  - Non-linearity function
- Gradient checks to identify bugs
  - If you build from scratch
- Parameter initialization
- Model is powerful enough?
  - If not, make it larger
  - Yes, so regularize, otherwise it will overfit
- Know your non-linearity function and its gradient
  - Example tanh(x)



$$\frac{\partial}{\partial x}\tanh(x) = 1 - \tanh^2(x)$$

Debugging

- Verify value of initial loss when using softmax
- Perfectly fit a single mini-batch
- If learning fails completely, maybe gradients stuck
  - Check learning rate
  - Verify parameter initialization
  - Change non-linearity functions

Avoiding Overfitting

- Reduce model size (but not too much)
- L1 and L2 regularization
- Early stopping (e.g., *patience*)
- Dropout (Hinton et al. 2012)
  - Randomly set 50% of inputs in each layer to 0