CS5740: Natural Language Processing
Spring 2018

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

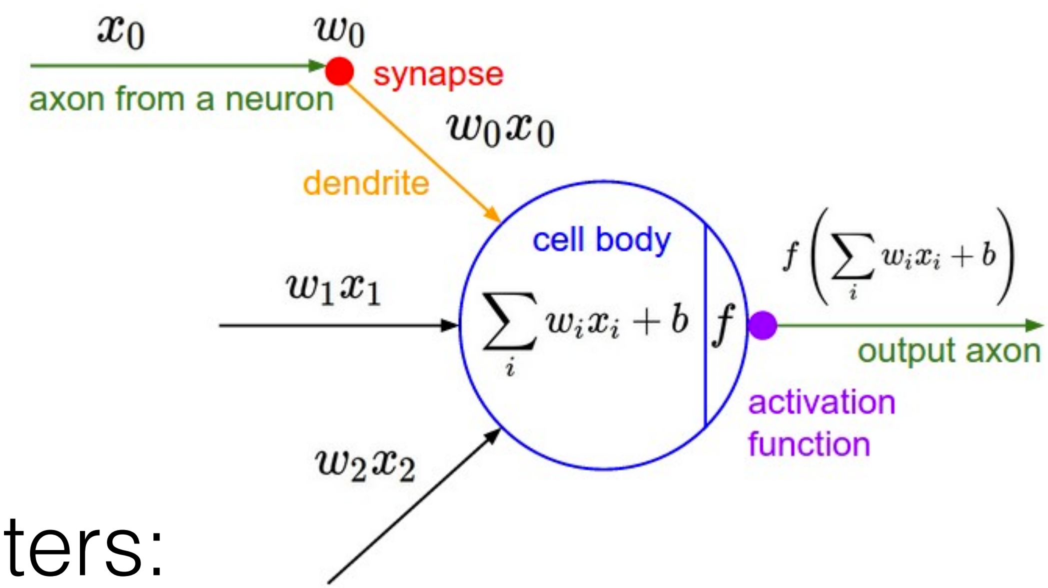
- Neural network algorithms date from the 80's
  - Originally inspired by early neuroscience
- Historically slow, complex, and unwieldy
- Now: term is abstract enough to encompass almost any model – but useful!
- Dramatic shift in last 2-3 years away from MaxEnt (linear, convex) to "neural net" (non-linear architecture)

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

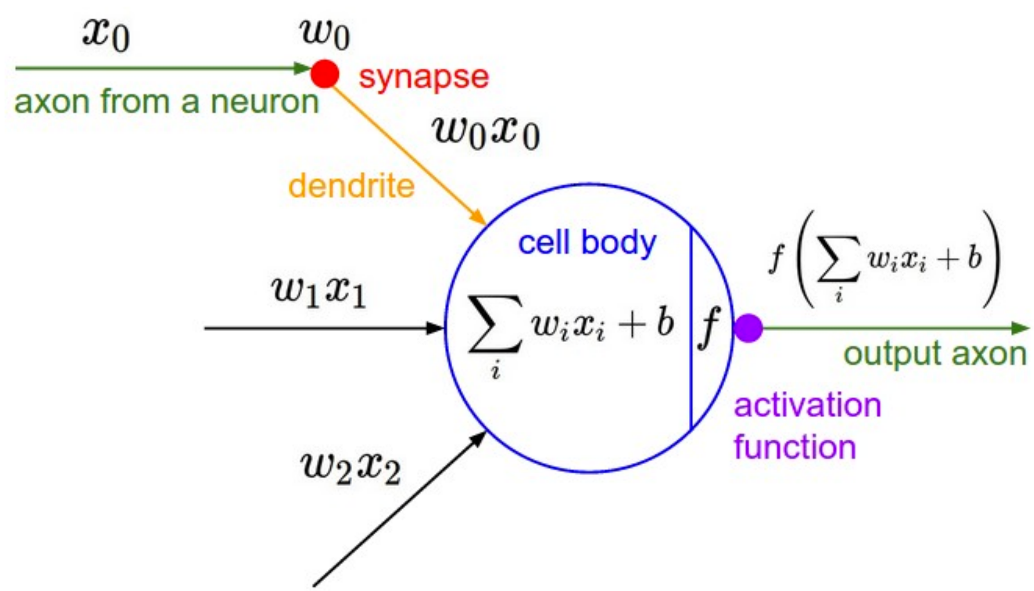
Neuron

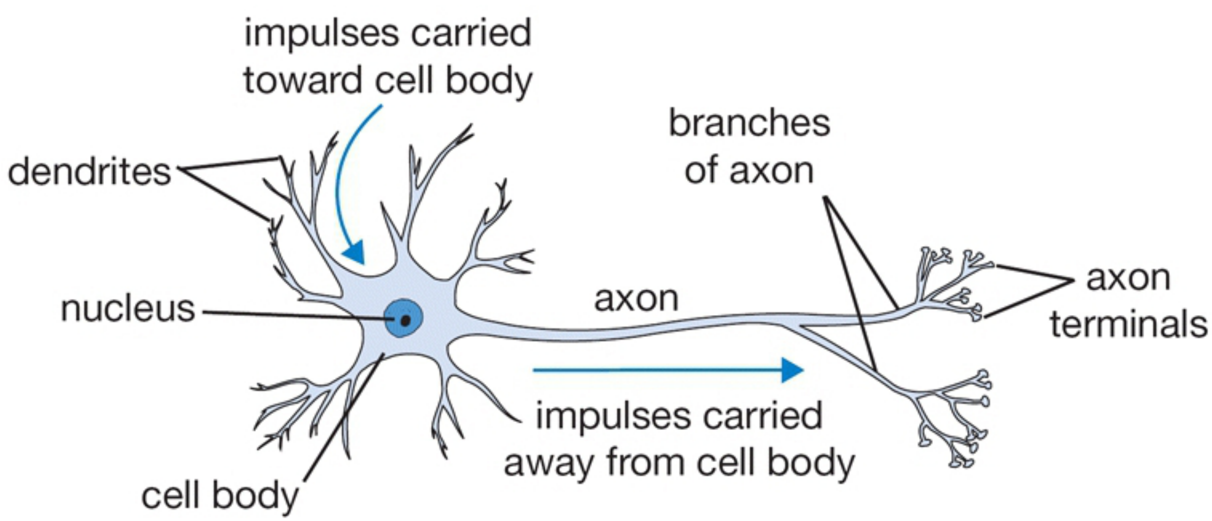
- Neural networks comes with their terminological baggage



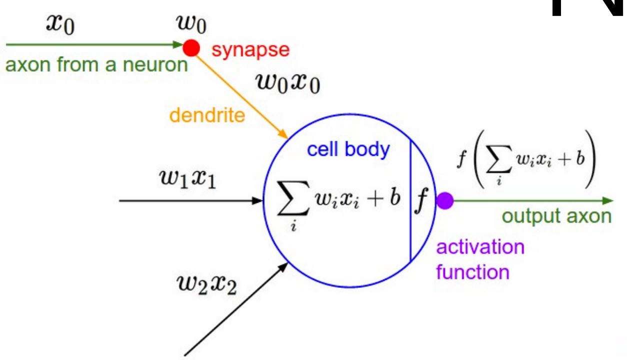
- Parameters:
  - Weights: w_i and b
  - Activation function
- If we drop the activation function, reminds you of something?

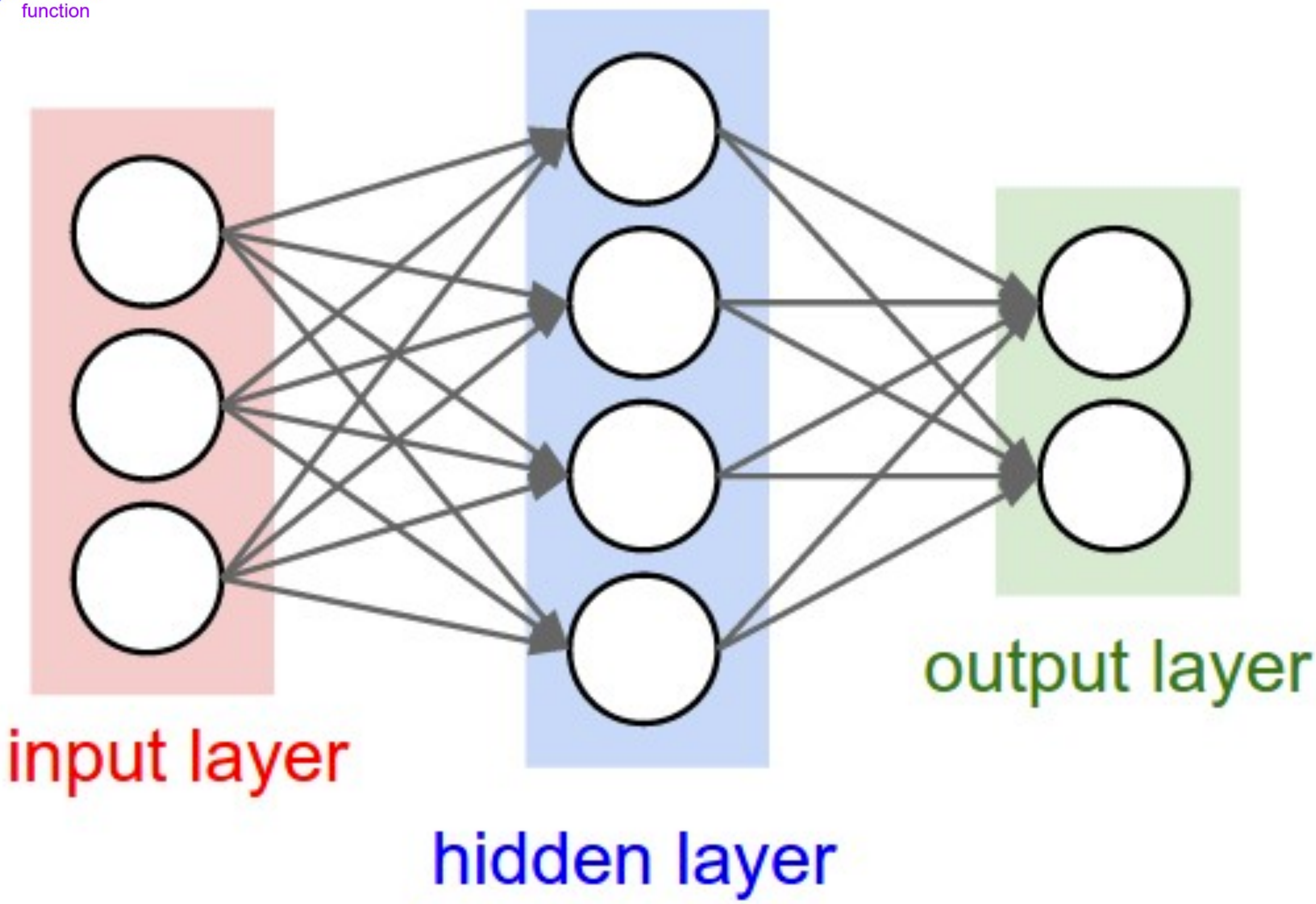
Biological “Inspiration”



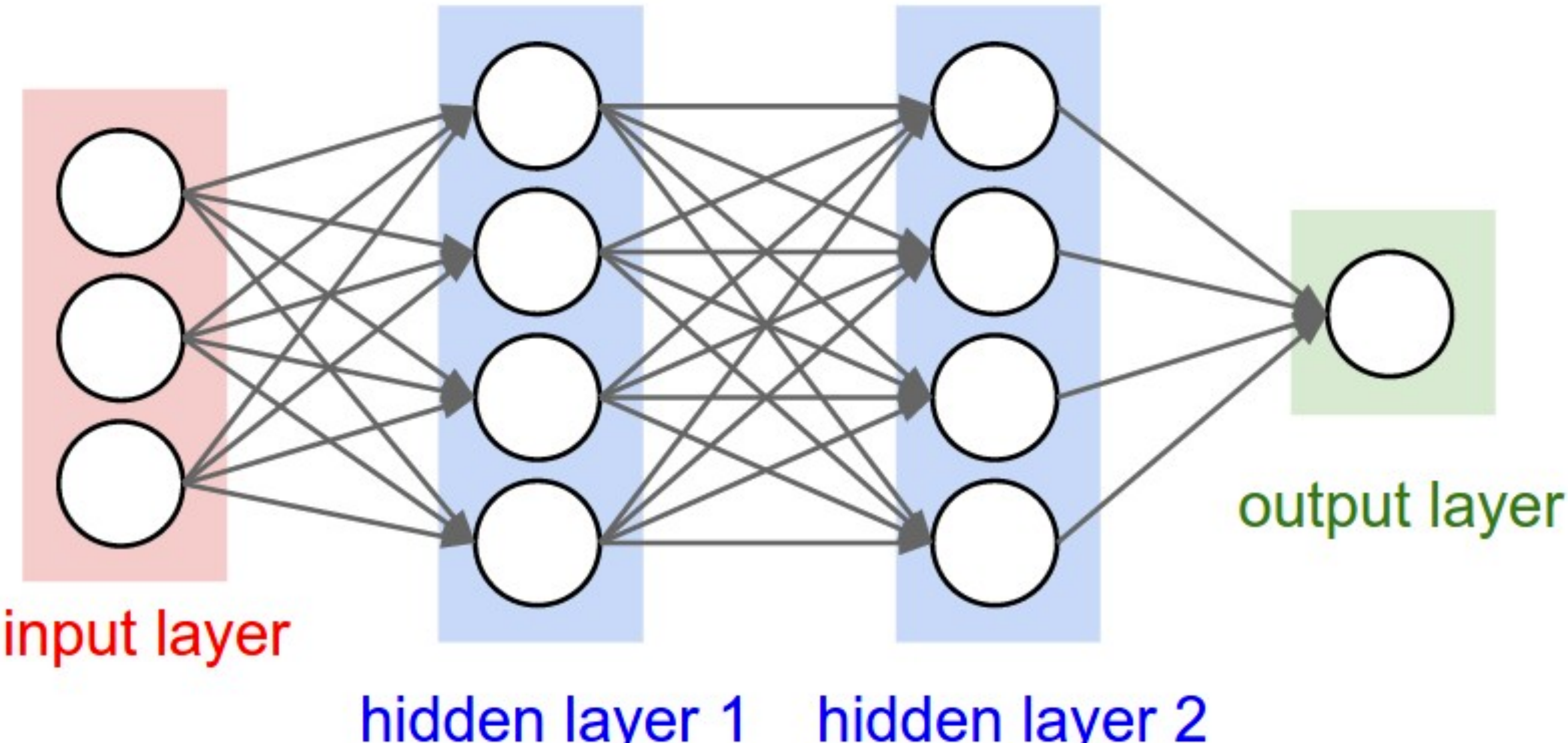


Neural Network





Neural Network

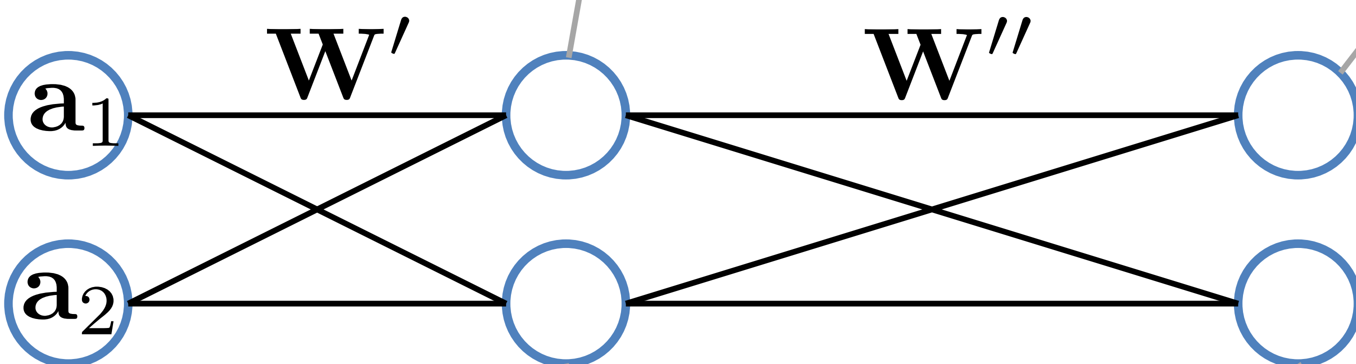


Matrix Notation

$$\mathbf{W}''(\mathbf{W}'\mathbf{a} + \mathbf{b}') + \mathbf{b}''$$

$$\mathbf{o}_1 = \mathbf{W}''_{11}\mathbf{h}_1 + \mathbf{W}''_{12}\mathbf{h}_2 + \mathbf{b}''_1$$

$$\mathbf{h}_1 = \mathbf{W}'_{11}\mathbf{a}_1 + \mathbf{W}'_{12}\mathbf{a}_2 + \mathbf{b}'_1$$



$$\mathbf{h}_2 = \mathbf{W}'_{21}\mathbf{a}_1 + \mathbf{W}'_{22}\mathbf{a}_2 + \mathbf{b}'_2$$

$$\mathbf{o}_2 = \mathbf{W}''_{21}\mathbf{h}_1 + \mathbf{W}''_{22}\mathbf{h}_2 + \mathbf{b}''_2$$

Neuron and Other Models

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

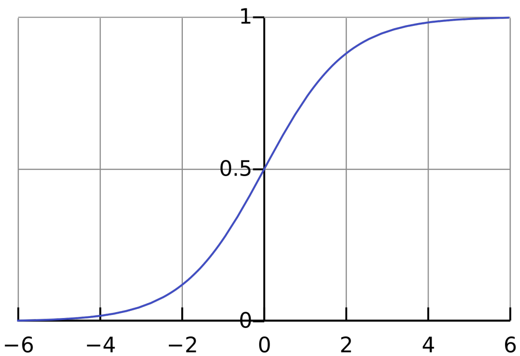
From MaxEnt to Neural Nets

- 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:

$$\begin{aligned}
P(y_1|x;w) &= \frac{e^{w^\top \phi(x,y_1)}}{e^{w^\top \phi(x,y_1)} + e^{w^\top \phi(x,y_2)}} \\
&= \frac{e^{w^\top \phi(x,y_1)}}{e^{w^\top \phi(x,y_1)} + e^{w^\top \phi(x,y_2)}} \frac{e^{-w^\top \phi(x,y_1)}}{e^{-w^\top \phi(x,y_1)}} \\
&= \frac{1}{1 + e^{w^\top (\phi(x,y_2) - \phi(x,y_2))}} \\
&= \frac{1}{1 + e^{-w^\top z}} = f(w^\top z)
\end{aligned}$$



Logisitc Function (sigmoid)

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

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:

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

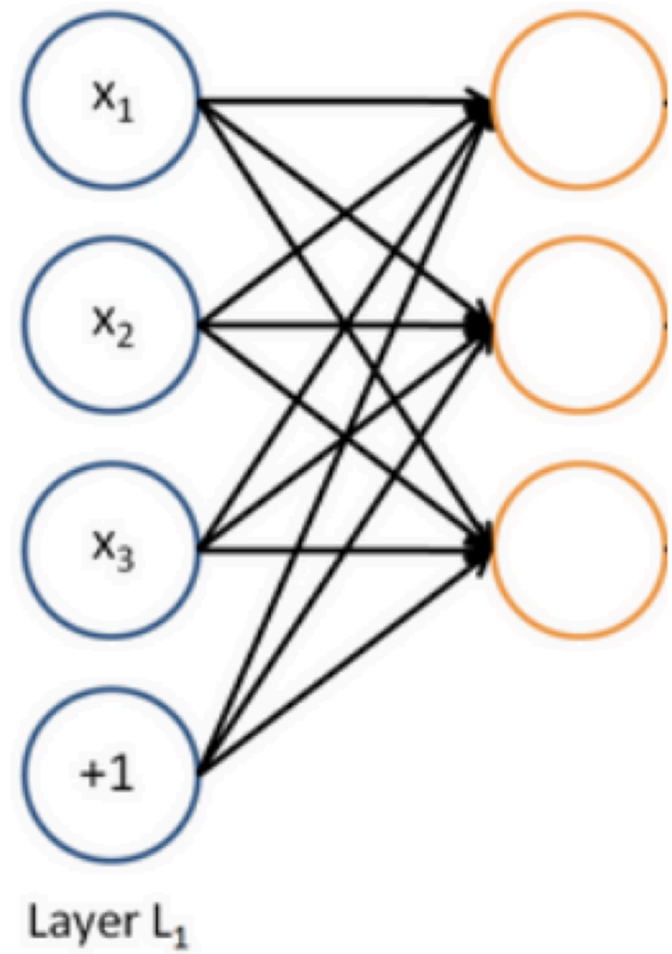
- Neuron:
  - Add an “always on” feature for class prior → bias term (b)

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

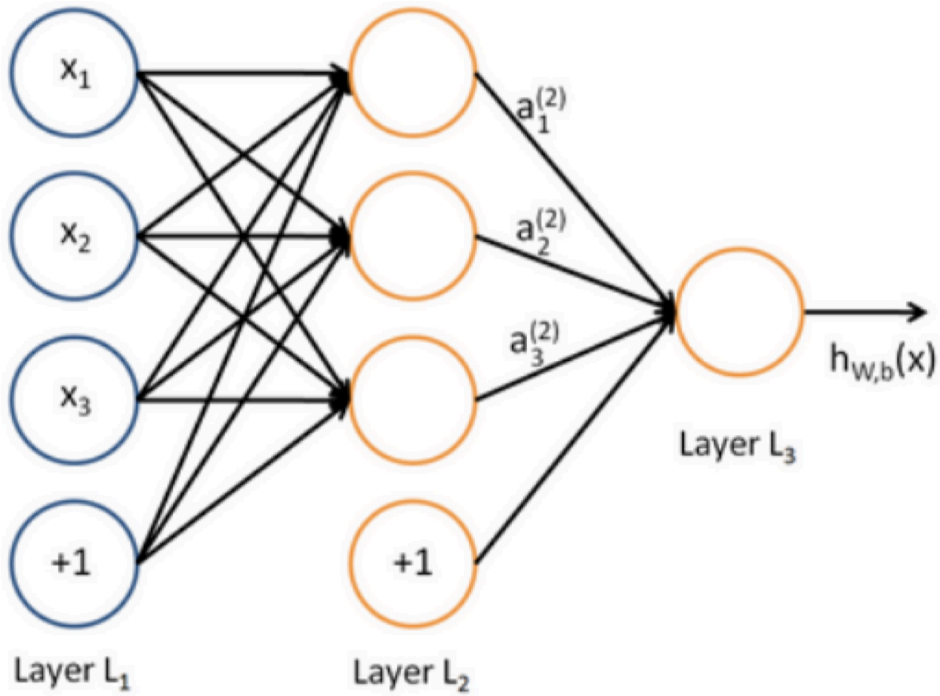
$$f(u) = \frac{1}{1+e^{-u}}$$

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

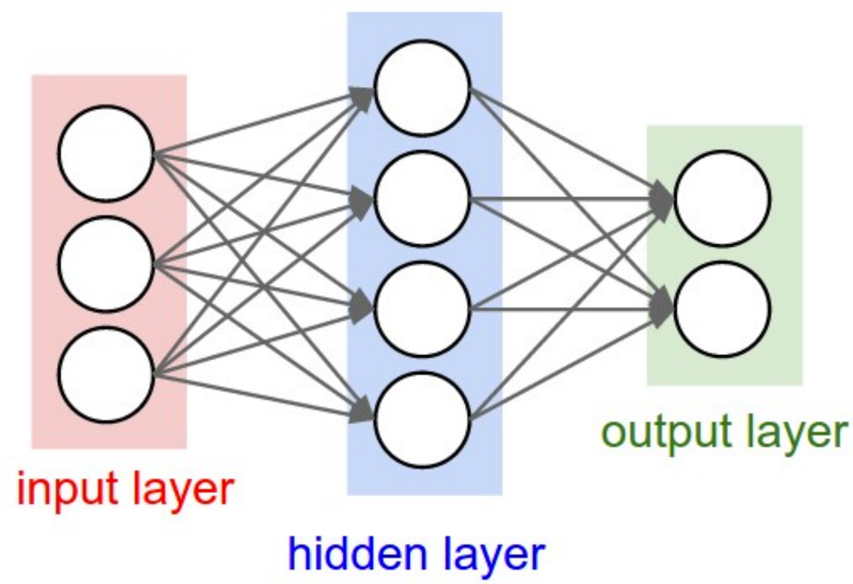
Probabilistic Output from Neural Nets

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

$$y = \mathrm{softmax}(\mathrm{W} \cdot \mathrm{z} + \mathrm{b})$$

$$\mathrm{softmax}(\mathrm{q}) = \frac{\mathrm{e}^{\mathrm{q}}}{\sum_{\mathrm{j}=1}^{\mathrm{k}} \mathrm{e}^{\mathrm{q}_\mathrm{j}}}$$

  - Where q is the output layer



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

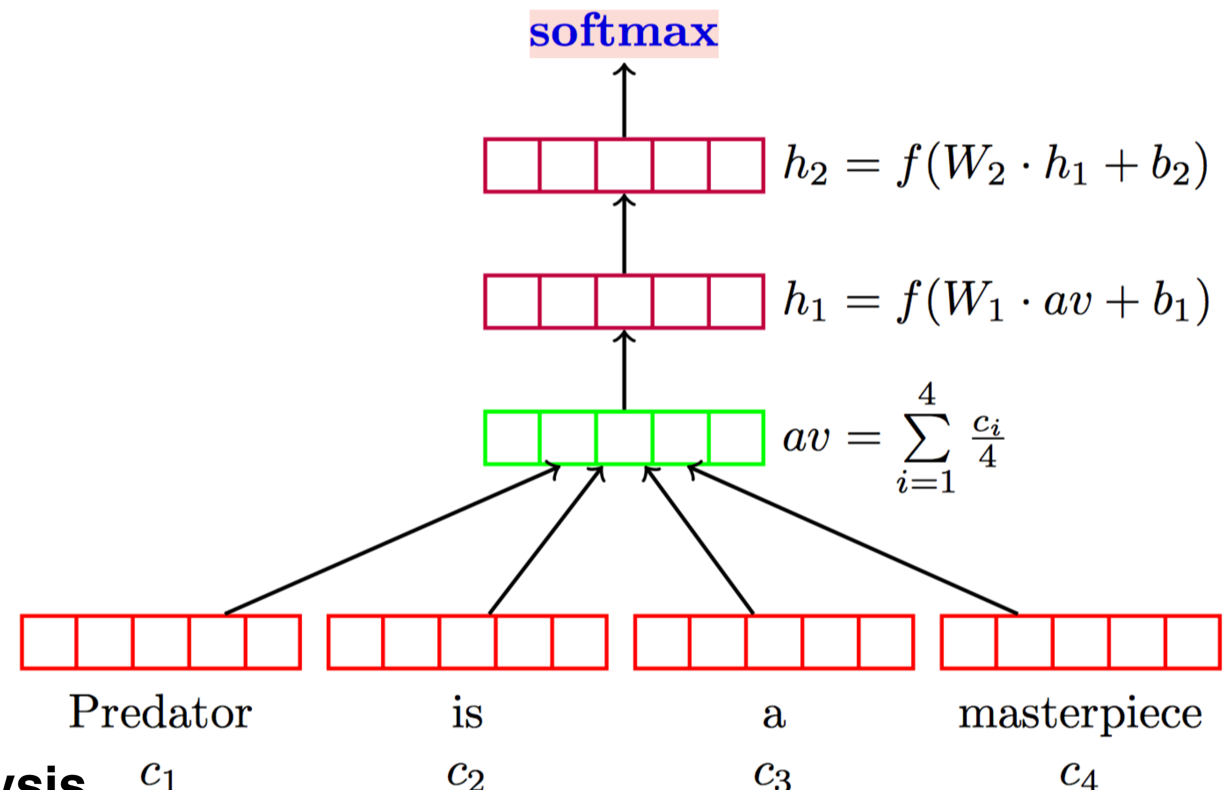
- Each word is represented using a dense low-dimensional vector
  - Low-dimensional << vocabulary size
- If trained well, similar words will have similar vectors
- How to train? What objective to maximize?
  - Soon …

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



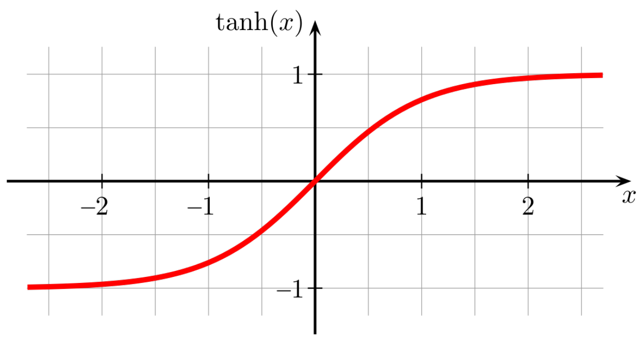
IMDB sentiment analysis

BOW + fancy smoothing + SVM	
NBOW + DAN	

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

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)



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

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