Lecture 3:
Quick Intro to Classification, Log-linear Models and Deep Networks

Supervised Learning in a Nutshell

- The task:
  - **Input:** training samples $x_1,x_2,x_3\ldots,x_n$ drawn from some distribution *D*, the learner is provided with their labels $y_1,\ldots,y_n$
  - **Goal:** correctly predict the label of a new sample drawn from *D*
- Evaluation:
  - Take an annotated corpus and partition it as follows:

Training Data	Development Data	Held-out Test Data

  - Development data → for exploration; test data → for reporting results

Classification

- Automatically make a decision about inputs
- **Examples:**
  - Document → category
  - Image of digit → digit
  - Image of object → object type (object recognition)
  - Query + webpage → best match
  - Symptoms → diagnosis
- Three main ideas:
  - Representation in a feature space
  - Scoring by linear functions
  - Learning by optimizing

Classification (1st example)

Task: predict whether a word is

1. function word (e.g., “the”, “in”, “than”)
2. content word (e.g., “dog”, “run”, “city”)

Representation: every word is represented by

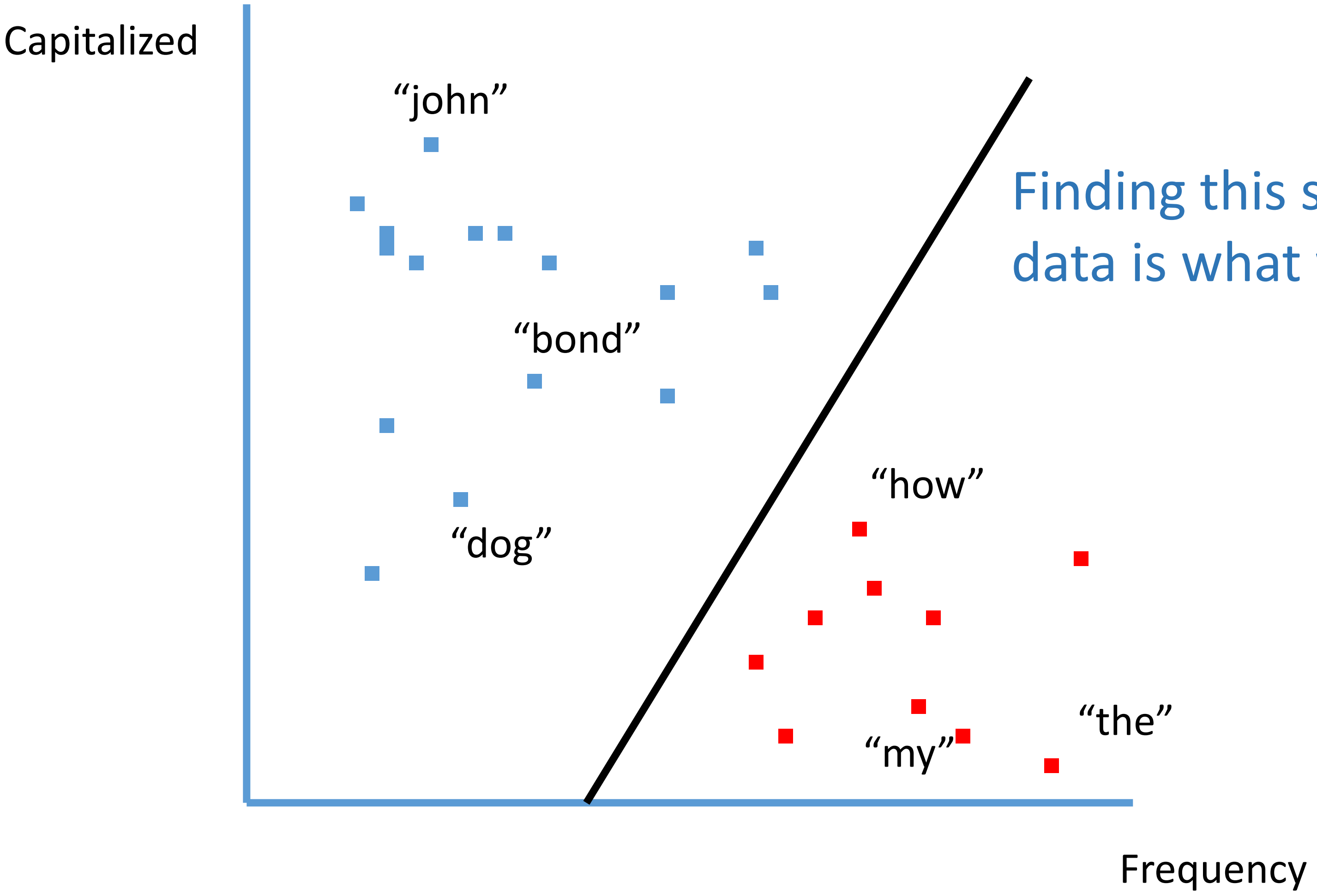
(1) its frequency

(2) how frequently it is capitalized

Model: there is a line that separates function words from content words

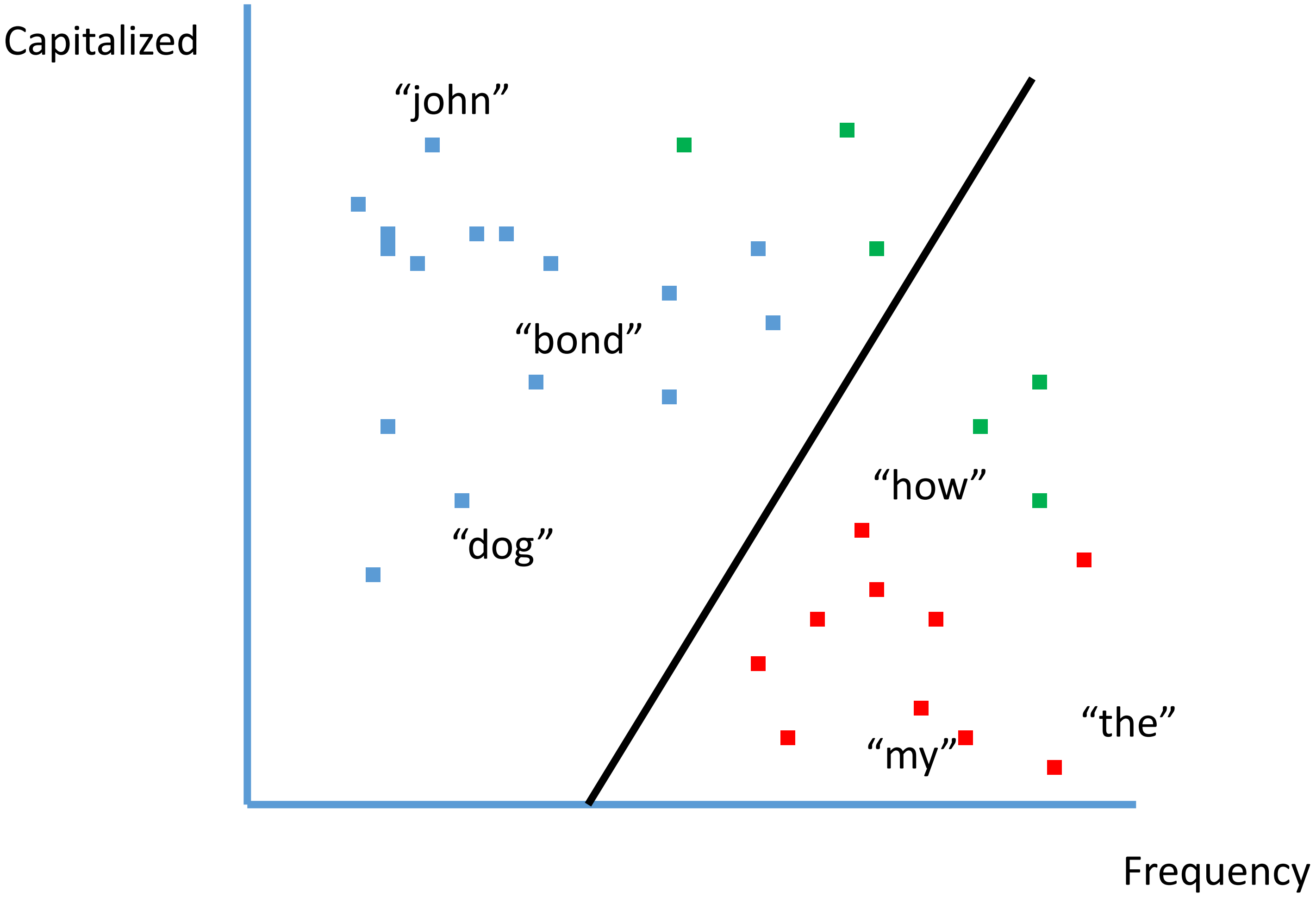
Learning: find that separating line

Classification (1^{st} example)

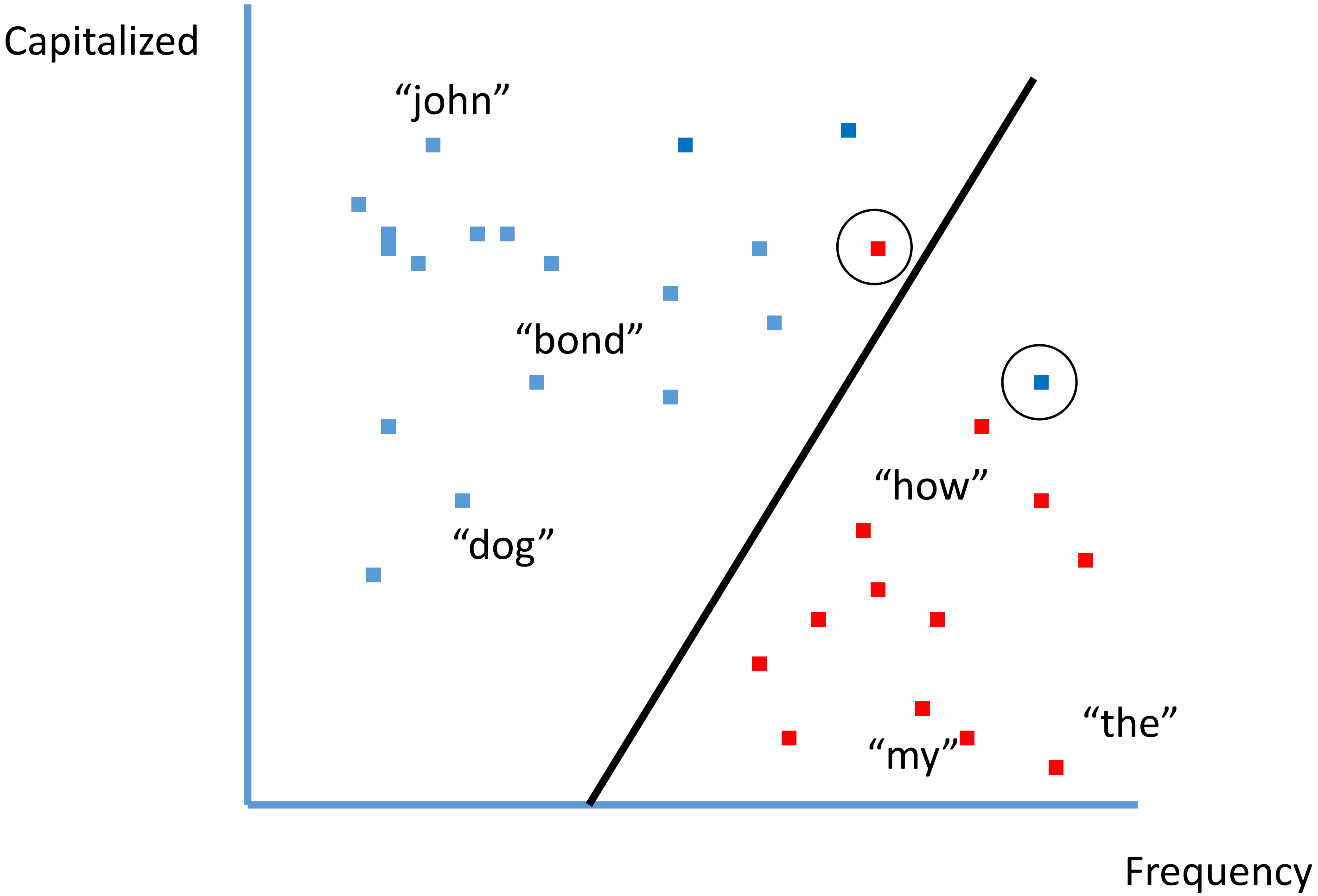


Finding this separator given the data is what we call "learning"

Classification (1st example)



Classification (1st example)



Probabilistic Classification

- Two broad approaches to predicting classes y*
  - Joint / Generative: work with a joint probabilistic model of the data
    - Assume functional form for P(X|Y), P(Y) and estimate parameters from the data
    - Use Bayes rules to calculate P(Y|X)
    - Prediction:

      $$y^*=\operatorname{argmax}_y P(y,x)= \operatorname{argmax}_y P(y)P(x|y)$$

    - Advantages: learning is easy, smoothing is well-understood, a complete model

  - Conditional / Discriminative (e.g., Logistic Regression)
    - Only model conditional probability $P(y|x)$
    - Prediction:

      $$y^* = \operatorname{argmax}_y P(y|x)$$

    - Advantages: no need to model $P(x)$, easier to develop feature-rich models for $P(y|x)$

Maximum Likelihood Estimation

- The likelihood is the probability of the observed data given the parameters:

$$P(x, y; \theta)$$

- In discriminative models we often talk about the conditional likelihood

$$P(y|x; \theta)$$

- The maximum estimator (MLE) is given as $\theta_{MLE} = argmax_{\theta} P(x, y; \theta)$

- And in discriminative models as $\theta_{MLE} = argmax_{\theta} P(y|x; \theta)$

Simplest Generative Model: Naïve Bayes

- Represent each sample in a feature space $x_i \in R^d$
- Assume the label is discrete $y \in L$, where L is some finite set

- **Model:** the Naïve Bayes model is defined as

$$P(x, y) = \mathrm{P(y)} \prod_{j=1}^{d} P(x^{(j)}|y)$$

- **Learning:** the Maximum Likelihood estimators of this model is

$$\hat{p}(y) = \frac{\#\{yi = y\}}{N}; \quad \hat{p}\left(x^{(j)}|y\right) = \frac{\#\{x_i{}^{(j)} = x, yi = y\}}{\#\{yi = y\}}$$

- We need to apply some smoothing, obviously

Simplest Generative Model: Naïve Bayes

- **Prediction:** for an example $x_i \in R^d$

$$y^* = argmaxy\, P(y)P(x|y)$$

- As there are only so many values *y* can take, we just iterate over all of them and find the maximum

Example of a Naïve Bayes: Bag of Words

- Say we want to decide what the topic of some text is
- We assume that the topic is the label *y*, and represent the text as a count vector of the words in it
  - Each distinct wordform is a feature (dimension)
  - Values of features are counts

the ape likes the bananas
John likes apples

[2,1,1,1,0,0]
[0,0,1,0,1,1]
Features: *the, ape, likes, bananas, john, apples*

  - This works OK for text categorization if the topics are not too fine-grained

Discriminative Approach

- Where there are complex features, generative approaches are more difficult to use
  - For instance, highly correlated features

- Many feature-based discriminative classification techniques out there, but Log-linear models extremely popular in the NLP community!

Text Classification

- Goal: classify documents into categories

… win the election …	*POLITICS*
… win the game …	*SPORTS*
… see a movie …	*OTHER*

- Classically: based on bag of words in the document
- But other information sources are potentially relevant: document length, average word length, document's source, document layout

Feature Representation

Washington County jail served 11,166 meals last month - a figure that translates to feeding some 120 people three times daily for 31 days

context:jail = 1
context:county = 1
context:feeding = 1
context:game = 0

…

local-context:jail = 1
local-context:meals = 1

…

object-head:meals = 1
object-head:ball = 0

- Features are indicator functions which count the occurrences of certain patterns in the input
- We will have different feature values for every pair of input x and class y

Notation

INPUT x^i … win the election ...

OUTPUT SPACE $\mathcal{Y}$ SPORTS, POLITICS, OTHER

OUTPUTS y SPORTS

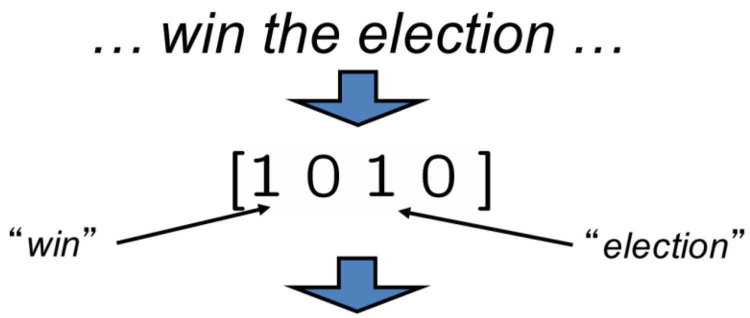
TRUE OUTPUTS y^i POLITICS

FEATURE VECTORS $\phi(x, y)$ [0 0 0 0 1 0 1 0 0 0 0 0]

SPORTS+"win" POLITICS+"win"

Block Notation

- We often think of the feature function as a mapping from a pair of input and label pair to a feature vector
  - In these cases, the feature vector will take a block form, as below



$$\phi(x, {\color{red}SPORTS}) = [{\color{red}1\ 0\ 1\ 0}\ {\color{blue}0\ 0\ 0\ 0}\ {\color{green}0\ 0\ 0\ 0}]$$
$$\phi(x, {\color{blue}POLITICS}) = [{\color{red}0\ 0\ 0\ 0}\ {\color{blue}1\ 0\ 1\ 0}\ {\color{green}0\ 0\ 0\ 0}]$$
$$\phi(x, {\color{green}OTHER}) = [{\color{red}0\ 0\ 0\ 0}\ {\color{blue}0\ 0\ 0\ 0}\ {\color{green}1\ 0\ 1\ 0}]$$

Prediction

- In a linear model, each feature gets a weight
  - Weight vector: *w*

$$\phi(x, SPORTS) = [1\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$\phi(x, POLITICS) = [0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 0]$$

$$w = [\ 1\ \ 1\ \ -1\ -2\ \ 1\ \ -1\ \ 1\ \ -2\ \ \ -2\ \ -1\ \ -1\ \ 1]$$

$$\phi(x, y)^T w$$

$$\phi(x, SPORTS) = [1\ 0\ 1\ 0 \ldots] \qquad \mathrm{score}(\mathrm{x^i}, \mathrm{SPORTS}, \mathrm{w}) = 1 \times 1 + (-1) \times 1 = 0$$

$$\phi(x, POLITICS) = [\ldots 1\ 0\ 1\ 0 \ldots] \qquad \mathrm{score}(\mathrm{x^i}, \mathrm{POLITICS}, \mathrm{w}) = 1 \times 1 + 1 \times 1 = 2$$

$$\phi(x, \mathrm{OTHER}) = [\ldots 1\ 0\ 1\ 0] \qquad \mathrm{score}(\mathrm{x^i}, \mathrm{OTHER}, \mathrm{w}) = (-2) \times 1 + (-1) \times 1 = -3$$

$$\mathrm{prediction}(\mathrm{x^i}, \mathrm{w}) = \mathrm{POLITICS}$$

- The prediction of *y* is the value that maximizes the score

Log-linear (MaxEnt) Models

- Model: use the scores as probabilities:

$score$(x,y,w)

$$p(y|x;w) = \frac{\exp\left(w \cdot \phi(x,y)\right)}{\sum_{y'} \exp\left(w \cdot \phi(x,y')\right)}$$

- Learning: maximize the (log) conditional likelihood of training data

$$L(w) = \log \prod_{i=1}^{n} P(y_i|x_i;w) = \sum_{i=1}^{n} \log P(y_i|x_i;w)$$

$$w^* = \arg\max_{w} L(w)$$

- Prediction: $argmax_y\, p(\mathrm{y}|\mathrm{x};\mathrm{w}) = argmax_y\, score(\mathrm{y},\mathrm{x};\mathrm{w})$

Log-linear Models

- The conditional likelihood is concave, which means that we can optimize it using standard convex optimization techniques
  - Like gradient ascent or quasi-Newton methods
  - The gradient is given as:

$$L(w) = \sum_{i=1}^{n} \log P(y_i | x_i; w) \qquad P(y|x;w) = \frac{e^{w \cdot \phi(x,y)}}{\sum_{y'} e^{w \cdot \phi(x,y')}}$$

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^{n} \left(\phi_j(x_i, y_i) - \sum_{y'} P(y'|x_i; w) \phi_j(x_i, y') \right)$$

Total count of feature j in correct candidates

Expected count of feature j in predicted candidates

Regularization

- The log-linear model doesn't have the same issue with zero counts as the generative model

- However, it may overfit the data by inflating *w*
  - If a certain feature appeared once with the class *SPORTS*, the model would have an incentive to place a very high weight for that feature and the label *SPORTS*
  - To combat this, we add a regularization term (often l_2 regularization)

$$L(w) = \sum_{i=1}^{n} \log p(y_i | x_i; w) - \frac{\lambda}{2} ||w||^2$$

Regularization: Modified Gradient

$$L(w) = \sum_{i=1}^{n} \left(w \cdot \phi(x_i, y_i) - \log \sum_{y} \exp(w \cdot \phi(x_i, y)) \right) - \frac{\lambda}{2} ||w||^2$$

$$\frac{\partial}{\partial w_j} L(w) = \sum_{i=1}^{n} \left(\phi_j(x_i, y_i) - \sum_{y} p(y|x_i; w) \phi_j(x_i, y) \right) - \lambda w_j$$

Total count of feature j in correct candidates

Expected count of feature j in predicted candidates

Big weights are bad

SGD for Log-Linear Models and Perceptron

1. $w^{(0)} \leftarrow 0$
2. **for** $r = 1 \ldots N_{iterations}$
3. **for** $i = 1 \ldots N$
4. $\hat{y} \leftarrow argmax_y \, \phi(x_i, y_i)$
5. $w^{((r-1)N+i)} \leftarrow w^{((r-1)N+i-1)} + \eta \cdot \Big(\Phi(x_i, y_i) - \Phi(x_i, \hat{y})\Big)$
6. **return** w

In Perceptron, a maximum rather than expectation

Example: Named Entity Recognition

- The task: finding all the names mentioned in a text and classifying them into their types (e.g., *Location, Organization, Person, Other*)
- Example:

Citing high fuel prices, [$_{\text{ORG}}$ **United Airlines**] said [$_{\text{TIME}}$ **Friday**] it has increased fares by [$_{\text{MONEY}}$ **$6**] per round trip on flights to some cities also served by lower-cost carriers. [$_{\text{ORG}}$ **American Airlines**], a unit of [$_{\text{ORG}}$ **AMR Corp.**], immediately matched the move, spokesman [$_{\text{PER}}$ **Tim Wagner**] said. [$_{\text{ORG}}$ **United**], a unit of [$_{\text{ORG}}$ **UAL Corp.**], said the increase took effect [$_{\text{TIME}}$ **Thursday**] and applies to most routes where it competes against discount carriers, such as [$_{\text{LOC}}$ **Chicago**] to [$_{\text{LOC}}$ **Dallas**] and [$_{\text{LOC}}$ **Denver**] to [$_{\text{LOC}}$ **San Francisco**].

From the Jurafsky & Martin textbook

Example: Regularization in NER

Because of regularization, the more common prefixes have larger weights even though entire-word features are more specific

Feature Weights

Feature Type	Feature	PER	LOC
Previous word	*at*	-0.73	0.94
Current word	*Grace*	0.03	0.00
Beginning bigram	*Gr*	0.45	-0.04
Current POS tag	NNP	0.47	0.45
Prev and cur tags	IN NNP	-0.10	0.14
Current signature	Xx	0.80	0.46
Prev-cur-next sig	x-Xx-Xx	-0.69	0.37
P. state - p-cur sig	O-x-Xx	-0.20	0.82
…			
Total:		**-0.58**	**2.68**

Local Context

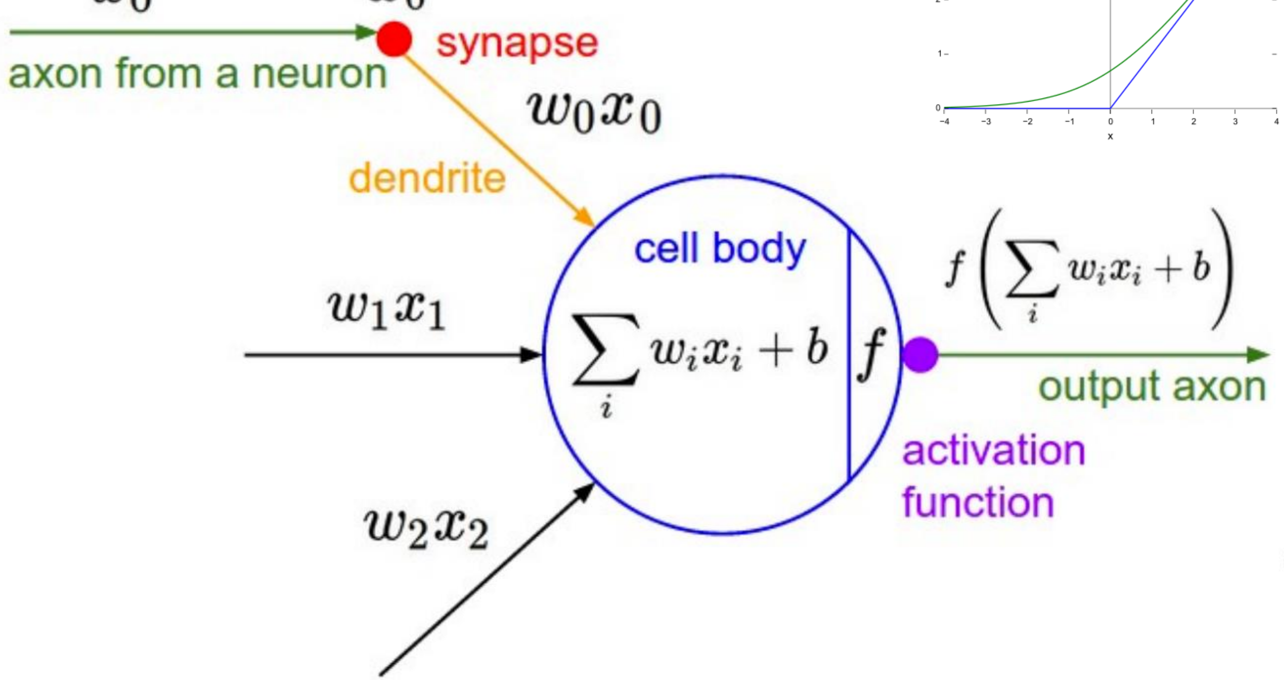
	Prev	Cur	Next
Word	at	Grace	Road
Tag	IN	NNP	NNP
Sig	x	Xx	Xx

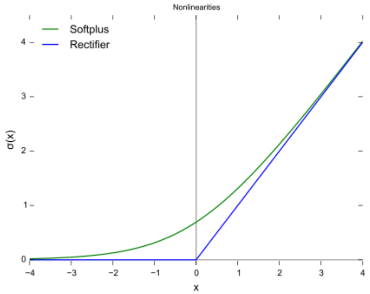
Briefly on Neural Networks

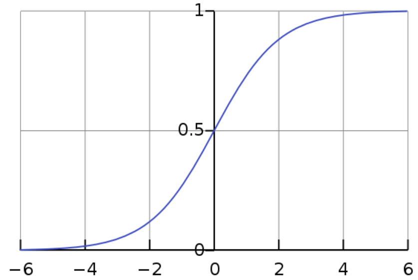
- Neural network algorithms date from the 70’s
- Originally inspired by early neuroscience
- Historically slow, complex, and unwieldy
- Now: term is abstract enough to encompass a wide variety of models
- Dramatic shift in NLP in the last 2-3 years away from log-linear models (linear, convex) to “neural net” (non-linear, non-convex architecture)

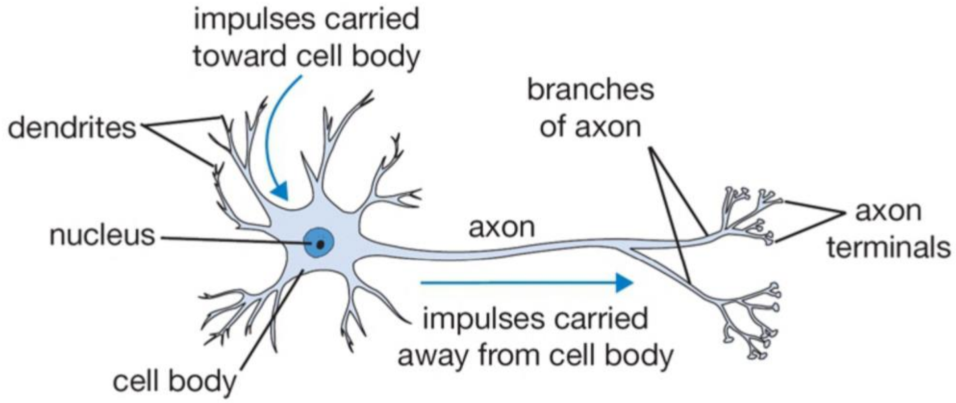
Nodes in Neural Networks

- Parameters: w_i and b

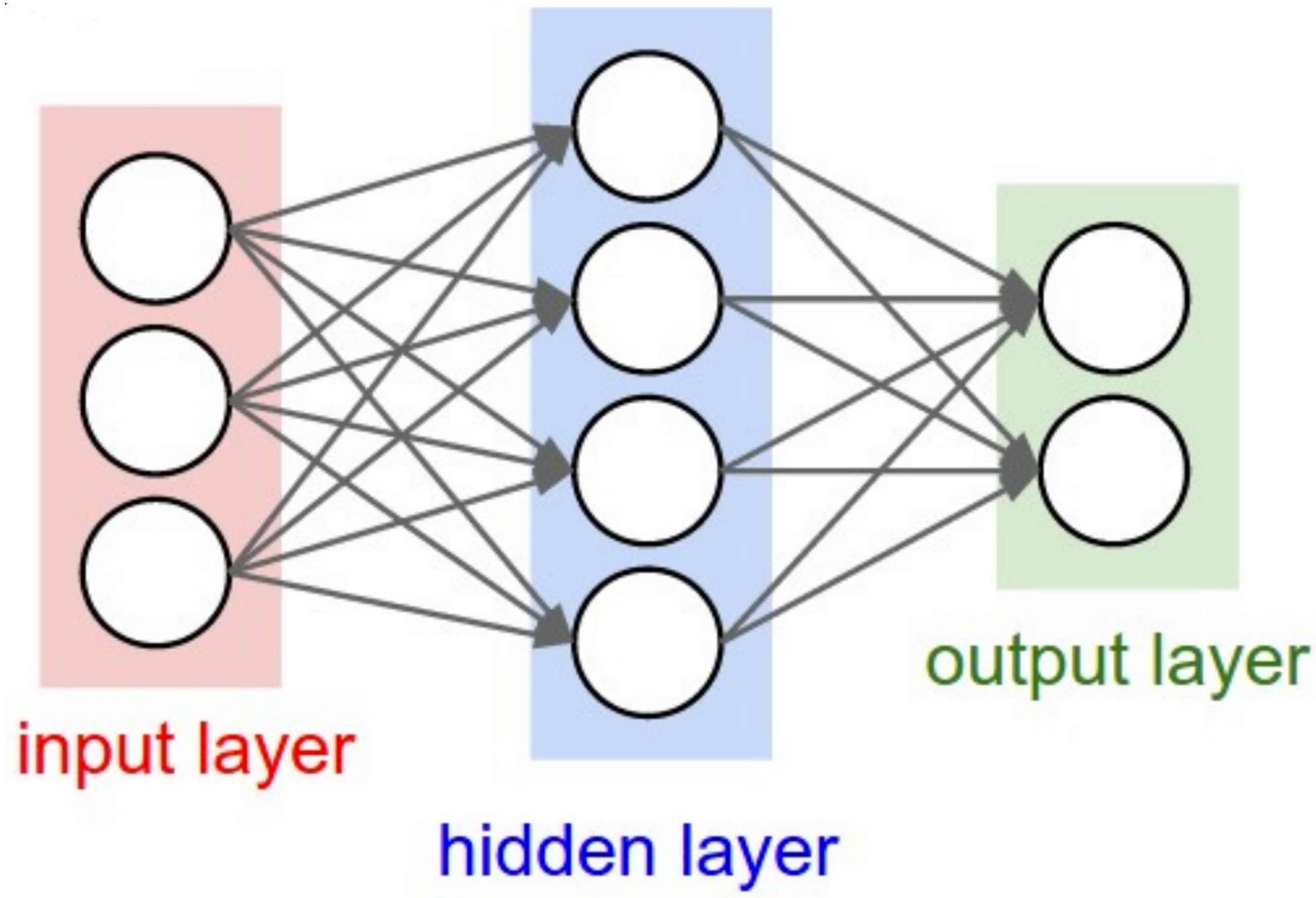


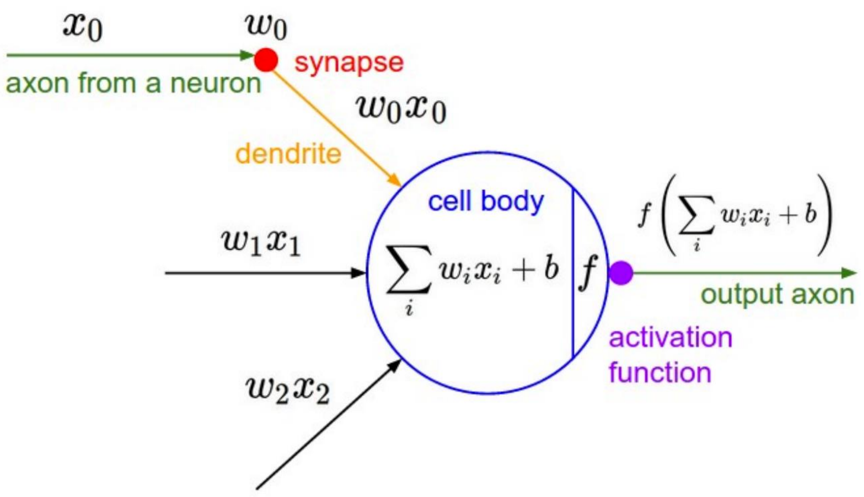




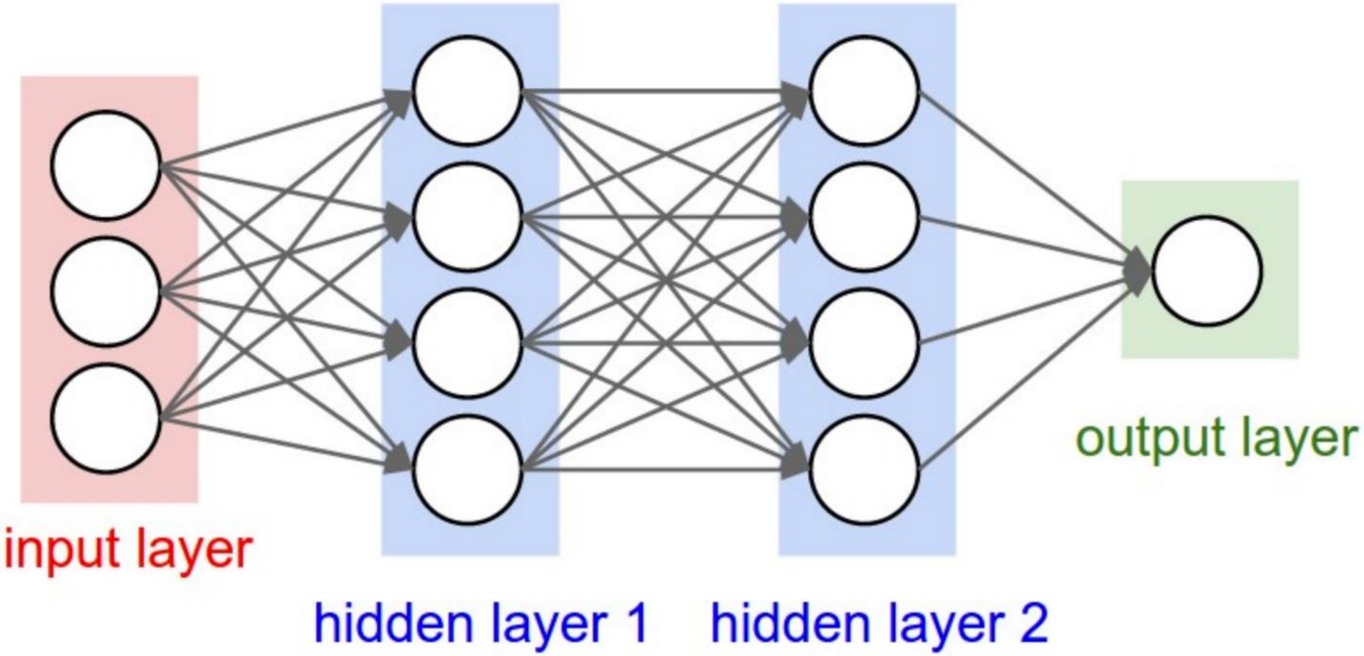


Feed-forward Neural Network





Feed-forward Neural Network



$$o = f(W_3 \cdot \vec{h_2} + b_3)$$

$$\vec{h_1} = f(W_1 \cdot \vec{x} + b_1) \qquad \vec{h_2} = f(W_2 \cdot \vec{h_1} + b_2)$$

Training: Backpropagation

- Training Neural Networks is generally done using the Backpropagation algorithm
  - You need to decide on a loss function ℓ, and then minimize the empirical risk

$$L_S(\theta) = \frac{1}{m} \sum_{i=1}^{m} \ell(\theta; (x_i, y_i))$$

- More details are given in IML and many good tutorials online, such as http://u.cs.biu.ac.il/~yogo/nnlp.pdf http://www.cs.cornell.edu/courses/cs5740/2016sp/resources/backprop.pdf
- In principle, all backpropagation does is (stochastic) gradient descent
  - This converges to a local minimum, which are often enough surprisingly good

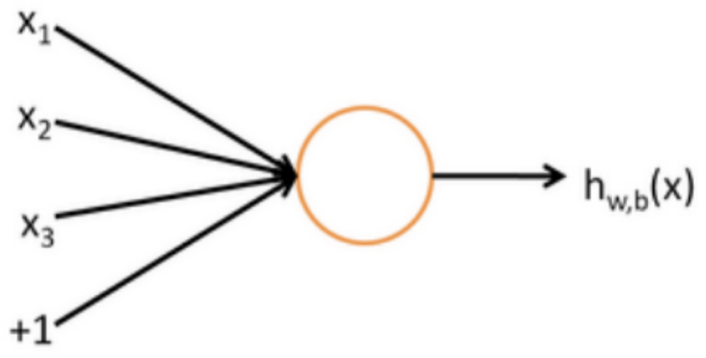
Log-linear Models and Neural Nets

- A single-layer NN with a sigmoid activation, and function is just a binary log-linear model
- If it's binary, we can assume: $\phi(x, CLASS_1) = -\phi(x, CLASS_2) := \phi(x)$

$$P(CLASS_1|x;w) = \frac{e^{w^T\phi(x,y)}}{e^{w^T\phi(x)} + e^{-w^T\phi(x)}} = \frac{1}{1+e^{-2\cdot w^T\phi(x)}}$$

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

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



One-hot vectors

- A vector of length |V|
- 1 for the target word and 0 for other words
- So if “popsicle” is vocabulary word #5, the **one-hot vector** is

  [0,0,0,0,1,0,0,0,0…….0]
- Often the vocabulary is truncated at some frequency threshold

Softmax Layers

- Softmax layers turn vector outputs into a probability distribution

$$SOFTMAX : \mathcal{R}^n \rightarrow \mathcal{R}^n$$

$$SOFTMAX(\vec{x})_i = \frac{e^{x_i}}{\Sigma_i e^{x_i}}$$

- Log-linear models (with *n* labels) is equivalent to a FF network with no hidden layers, and a softmax layer at the end
  - Simple exercise: show the formulations are equivalent