CS 489/698: Introduction to Natural Language Processing

Lecture 4: Classification

Instructor: Freda Shi

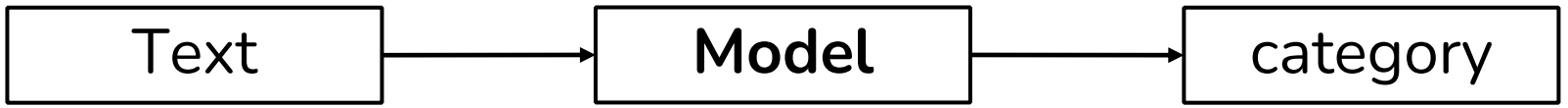
fhs@uwaterloo.ca

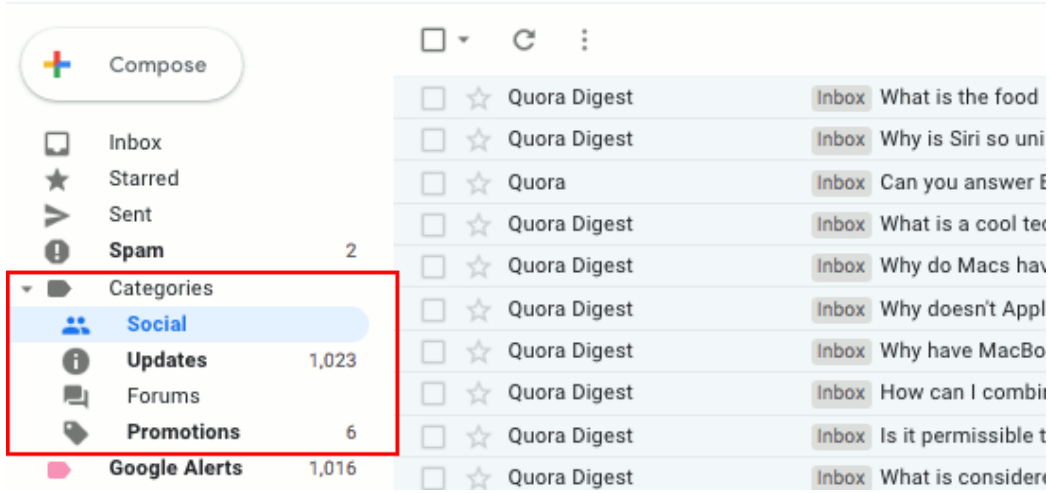
January 14th, 2026

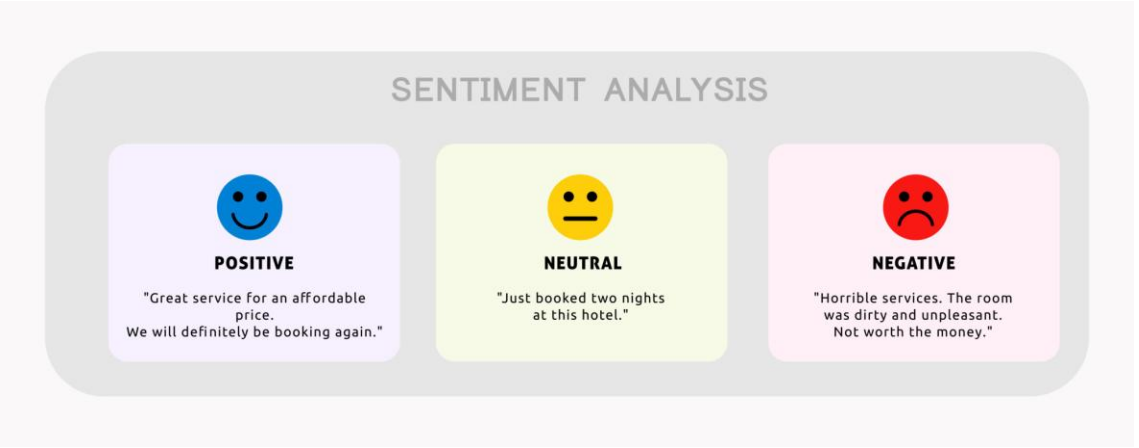
UNIVERSITY OF WATERLOO

Classification

Simplest user facing NLP application







Outline of Today's Lecture

Modeling approaches for classification

- Rule-based classification
- Naïve Bayes
- Logistic regression
- Neural networks

Rule-Based Classifier

Sentiment classification of sentence s (classes: positive, negative)

- If s contains words in [*good*, *excellent*, *extraordinary*, ...] return positive
- If s contains words in [*bad*, *terrible*, *awful*, ...] return negative


Nice interpretability


Can be very accurate (with carefully refined rules)

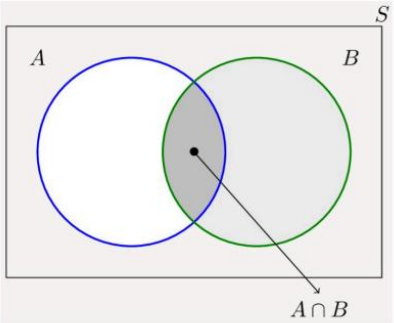

Rules are difficult to define


System can be very complicated

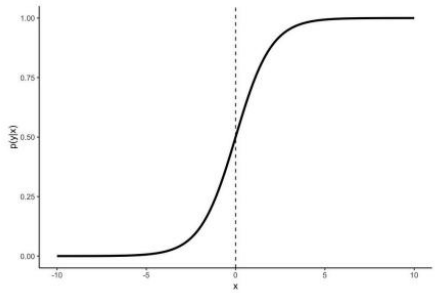

Hardly generalizable

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a lexicon and rule-based sentiment analysis tool that is *specifically attuned to sentiments expressed in social media*. It is fully open-sourced under the [MIT License] (we sincerely appreciate all attributions and readily accept most contributions, but please don't hold us liable).

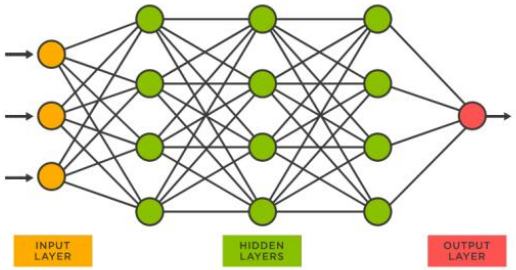
Statistical Classifiers



Naïve Bayes



Logistic Regression



Neural Networks

Statistical Classifiers: General Formulation

Data: A set of labeled sentences $\mathcal{D} = \{(s_1, y_1), (s_2, y_2), \ldots, (s_n, y_n)\}$.

- s_i: a sentence (or a piece of text).
- y_i: label, usually represented as an integer.

Deliverable: A classifier that takes an arbitrary sentence s as input, and predicts the label y.

$$\text{classify}(s) = \underset{\text{y}}{\text{argmax}}\ \text{score}(\text{s}, \text{y}; \Theta)$$

Θ: parameters of score function

Modeling, Learning, and Inference

Inference: solve arg max

Modeling: define score function

$$\text{classify}(s) = \underset{y}{\text{argmax}}\ \text{score}(s, y; \Theta)$$

Learning: choose parameter

Naïve Bayes: Modeling

Probabilistic model: $\text{score}(\text{s}, \text{y}; \Theta) \coloneqq P(y \mid s)$ estimated from the data.

How to estimate $P(\, y \mid s\,)$ to ensure sufficient generalizability?

The Bayes rule:

$$P(\, y \mid s\,) = \frac{P(\, s \mid y\,)P(y)}{P(s)} \propto P(\, s \mid y\,)P(y)$$

Unigram assumption:

$$P(\, s \mid y\,) = P(\, w_1 \mid y\,)P(\, w_2 \mid y\,) \ldots P(\, w_k \mid y\,) \quad \text{if } s = w_1 w_2 \ldots w_k$$

Θ in $\text{score}(\text{s}, \text{y}; \Theta)$: look-up table that stores $P(w \mid y)$ and $P(y)$.

Naïve Bayes: Learning

$$P(s \mid y) = P(w_1 \mid y)P(w_2 \mid y) \ldots P(w_k \mid y) \quad \text{if } s = w_1 w_2 \ldots w_k$$

Each $P(w \mid y)$ and $P(y)$ can be estimated by counting from the corpora.

$$P(w \mid y) = \frac{\text{count}(w, y)}{\sum_{w' \in V} \text{count}(w', y)} \qquad P(y) = \frac{\text{count(y)}}{n}$$

This is the estimation that maximizes dataset probability.

Issue: an unseen word in certain class will lead to a zero probability!

Solution: smoothing.

$$P(w \mid y) = \frac{\text{count}(w, y) + \alpha}{\sum_{w' \in V} \text{count}(w', y) + \alpha|V|}$$

“Bag-of-words” Features

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
sweet	1
genre	1
fairy	1
humor	1
have	1
great	1

...

Naïve Bayes: Inference

$$\text{classify}(s) = \underset{\text{y}}{\text{argmax}}\ \text{score}(\text{s}, \text{y}; \Theta) = \underset{\text{y}}{\text{argmax}}\ P(y \mid s)$$

$$P(\, y \mid s\,) \propto P(y)\ P(\, s \mid y\,) = P(y) P(\, w_1 \mid y\,) P(\, w_2 \mid y\,) \ldots P(\, w_k \mid y\,)$$

Consult the look-up tables for values of $P(y)$ and $P(\, w \mid y\,)$.

Naïve Bayes: Example

Type	Category	Sentence
Training	-	just plain boring
Training	-	entirely predictable and lacks energy
Training	-	no surprises and very few laughs
Training	+	very powerful
Training	+	the most fun film of the summer
Testing	?	predictable with no fun

- **Prior from training set:**

$$P(+) = \frac{2}{5}, P(-) = \frac{3}{5}$$

- $|V| = 20$ **from training set**
- **Conditional probabilities** ($\alpha = 1$):

$$P(\text{predictable} \mid +) = \frac{0+1}{9+20} = \frac{1}{29}$$

$$P(\text{no} \mid +) = \frac{0+1}{9+20} = \frac{1}{29}$$

$$P(\text{fun} \mid +) = \frac{1+1}{9+20} = \frac{2}{29}$$

$$P(\text{predictable} \mid -) = \frac{1+1}{14+20} = \frac{2}{34}$$

$$P(\text{no} \mid -) = \frac{1+1}{14+20} = \frac{2}{34}$$

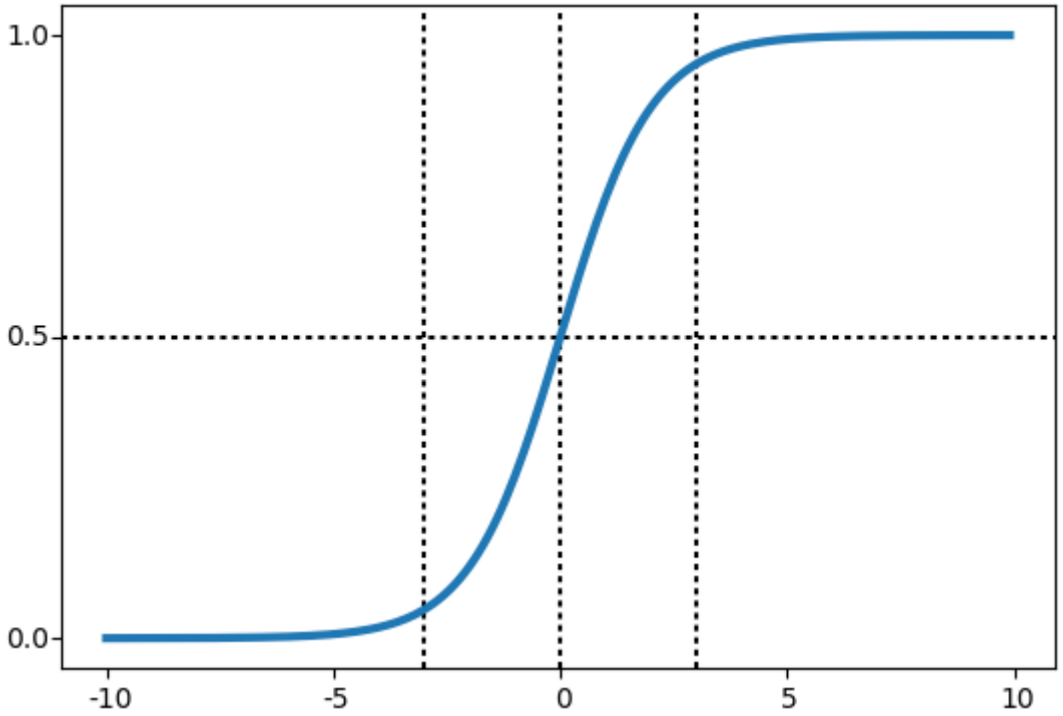
$$P(\text{fun} \mid -) = \frac{0+1}{14+20} = \frac{1}{34}$$

- **Scoring the examples**

$$P(+)P(\text{predictable} \mid +)P(\text{no} \mid +)P(\text{fun} \mid +) = 3.2 \times 10^{-5}$$

$$P(-)P(\text{predictable} \mid -)P(\text{no} \mid -)P(\text{fun} \mid -) = 6.1 \times 10^{-5}$$

Recap: Logistic Function



Logistic function: $\sigma(x) = \frac{1}{1+e^{-x}}, \quad \sigma: \mathbb{R} \to (0,1).$

"Native" for binary classification.

Logistic Regression: Modeling

Suppose we can represent sentence s with a vector $\boldsymbol{x}$.

Probabilistic model:

$$\text{score}(s, y = 1; \boldsymbol{\theta}) \coloneqq P(y = 1 \mid s) = \sigma(\boldsymbol{\theta} \cdot \boldsymbol{x} + b)$$
$$\text{score}(s, y = 0; \boldsymbol{\theta}) \coloneqq P(y = 0 \mid s) = 1 - \sigma(\boldsymbol{\theta} \cdot \boldsymbol{x} + b)$$

$\boldsymbol{\theta} \cdot \boldsymbol{x} + b$ can be written as $\boldsymbol{\theta} \cdot \boldsymbol{x}$ if $\boldsymbol{x}$ has a constant dimension.

Logistic Regression: Learning

Objective: maximizing the dataset probability, under the assumption that each example is sampled independently.

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{n} \sigma(\boldsymbol{\theta} \cdot \boldsymbol{x_i})^{y_i} \big(1 - \sigma(\boldsymbol{\theta} \cdot \boldsymbol{x_i})\big)^{1-y_i}$$

For better numerical representation, we usually take the negative logarithm of the probability as the loss and minimize it:

$$Loss(\boldsymbol{\theta}) = -\sum_{i=1}^{n} y_i \log \sigma(\boldsymbol{\theta} \cdot \boldsymbol{x_i}) + (1 - y_i) \log\big(1 - \sigma(\boldsymbol{\theta} \cdot \boldsymbol{x_i})\big)$$

Gradient descent:

$$\boldsymbol{\theta} = \boldsymbol{\theta} \; - \eta \nabla_{\boldsymbol{\theta}} Loss(\boldsymbol{\theta})$$

Logistic Regression: Inference

Compute whether class 0 or 1 has larger probability.

What if there are more than 2 classes?

- 1 vs. 1 for $\frac{C(C-1)}{2}$ class pairs and do voting; or
- 1 vs. all for C classes and do argmax.

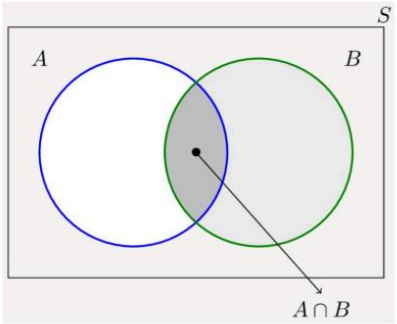
- Probabilistic interpretation (over classes) no longer hold for multi-class classification with logistic regression.

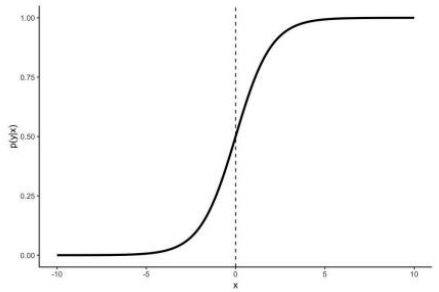
Generative vs. Discriminative Model for Classification

- Generative models: $P(s, y)$ is accessible when modeling $P(y \mid s)$.
- Discriminative models: $P(y \mid s)$ is directly modeled.

What are the differences?

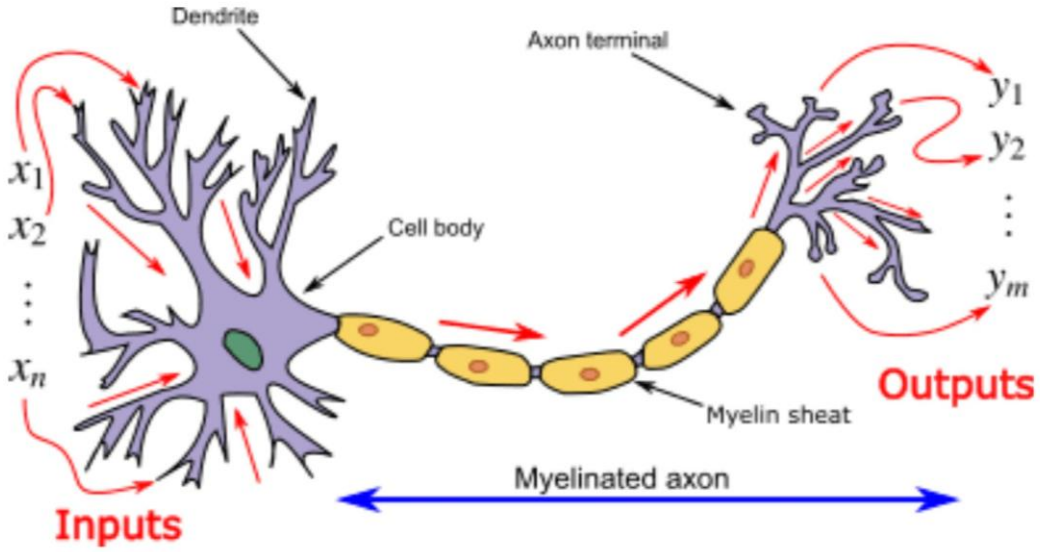
Key difference: if you can generate a new data example once you have the model.

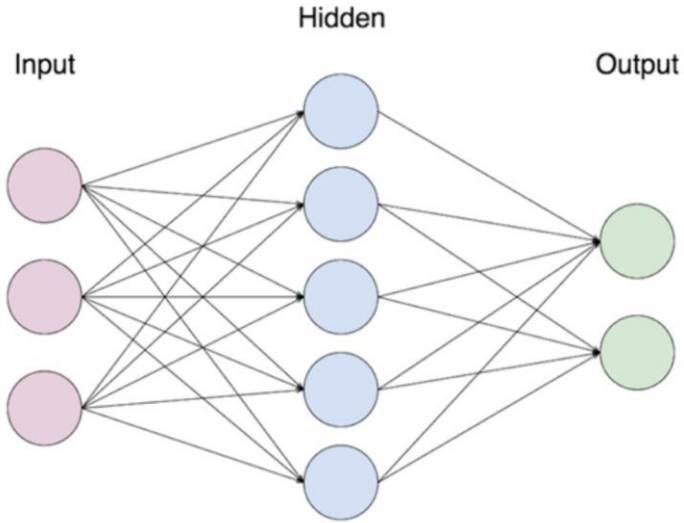




Neural-Network Classifier

- A neural network is a function
  - It has inputs and outputs.
  - “Neural modeling” now is better thought of as dense representation learning.





Neural Implementation of Scoring Function

Inference: solve arg max

Modeling: define score function

$$\text{classify}(s) = \underset{\text{y}}{\text{argmax}}\ \text{score}(\text{s}, \text{y}; \Theta)$$

Learning: choose parameter

With a neural network--based function $f_{\boldsymbol{\theta}}$, we input $\boldsymbol{x}$ and collects a vector $\hat{\boldsymbol{y}} = f_{\mathbf{w}}(\mathbf{x})$; $\text{score}(\boldsymbol{x}, y; \boldsymbol{\theta})$ is defined by selecting the corresponding entry in $\hat{\boldsymbol{y}}$.

Common Neural Network Notations

$\boldsymbol{u}$: a vector

u_i: entry i in the vector

$\boldsymbol{W}$: a matrix

$w_{i,j}$: entry (i, j) in the matrix

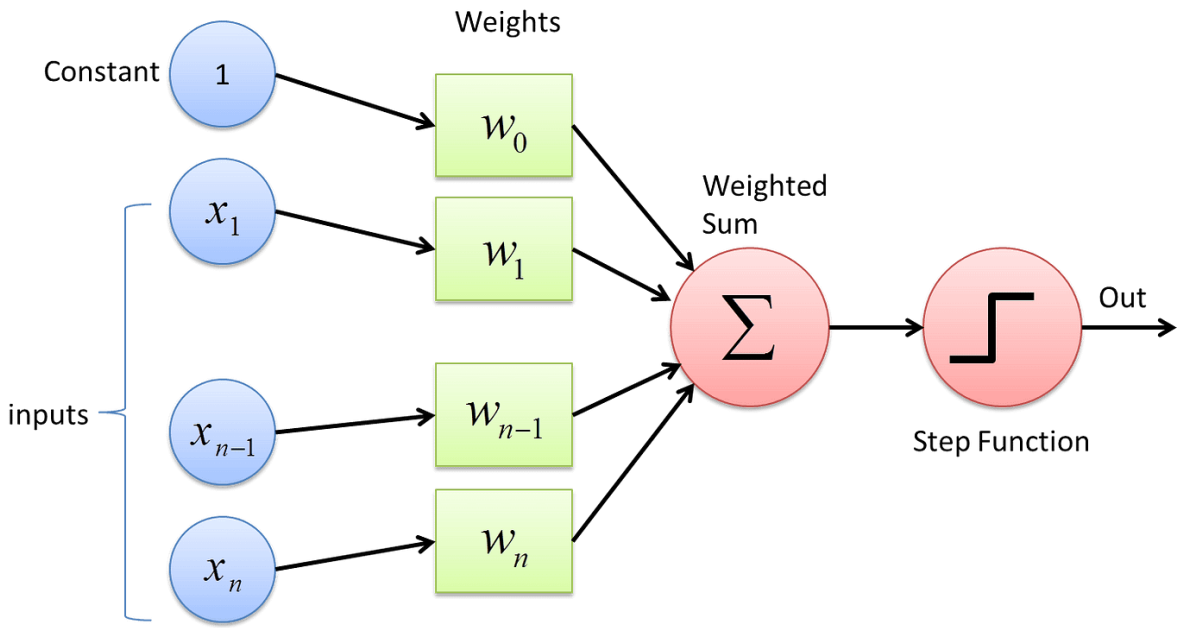
Perceptron

$$\text{perceptron}(\boldsymbol{x};\boldsymbol{\theta}) = \text{step}(\boldsymbol{\theta}^T\boldsymbol{x})$$

activation function

affine transform

$$\text{step}(z) = \begin{cases} 1 & if\ z \geq 0 \\ 0 & if\ z < 0 \end{cases}$$



Perceptron: Learning

$$\text{perceptron}(\boldsymbol{x}; \boldsymbol{\theta}) = \text{step}(\boldsymbol{\theta}^T \boldsymbol{x})$$

Predict the label: $\hat{y}^{(i)} = \text{perceptron}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$

Update weights: $\boldsymbol{\theta} = \boldsymbol{\theta} + \eta \cdot \left(y^{(i)} - \hat{y}^{(i)}\right)\boldsymbol{x}^{(i)}$

$\hat{y}^{(i)}$	$y^{(i)}$	What happens
0	0	nothing
1	1	nothing

Perceptron: Learning

$$\text{perceptron}(\boldsymbol{x}; \boldsymbol{\theta}) = \text{step}(\boldsymbol{\theta}^T \boldsymbol{x})$$

Predict the label: $\hat{y}^{(i)} = \text{perceptron}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$

Update weights: $\boldsymbol{\theta} = \boldsymbol{\theta} + \eta \cdot \left(y^{(i)} - \hat{y}^{(i)}\right)\boldsymbol{x}^{(i)}$

$y^{(i)}$	$\hat{y}^{(i)}$	What happens
0	0	nothing
1	1	nothing
0	1	$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \boldsymbol{x}^{(i)}$

Perceptron: Learning

$$\text{perceptron}(\boldsymbol{x}; \boldsymbol{\theta}) = \text{step}(\boldsymbol{\theta}^T \boldsymbol{x})$$

Predict the label: $\hat{y}^{(i)} = \text{perceptron}\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$

Update weights: $\boldsymbol{\theta} = \boldsymbol{\theta} + \eta \cdot \left(y^{(i)} - \hat{y}^{(i)}\right)\boldsymbol{x}^{(i)}$

$y^{(i)}$	$\hat{y}^{(i)}$	What happens
0	0	nothing
1	1	nothing
0	1	$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \boldsymbol{x}^{(i)}$
1	0	$\boldsymbol{\theta} = \boldsymbol{\theta} + \eta \boldsymbol{x}^{(i)}$

Neural Layer: Generalized Perceptron

A neural layer = affine transformation + nonlinearity

$$\text{perceptron}(\boldsymbol{x}) = \text{step}(\boldsymbol{\theta}^T \boldsymbol{x} + b)$$

activation function

affine transform

$$\text{neural\_layer}(\boldsymbol{x}) = g(\boldsymbol{W}^T \boldsymbol{x} + \boldsymbol{b})$$

Output is a vector (results from multiple independent perceptrons).

Can have other activation functions for nonlinearity.

Stacking Neural Layers

Multiple neural layers can be stacked together.

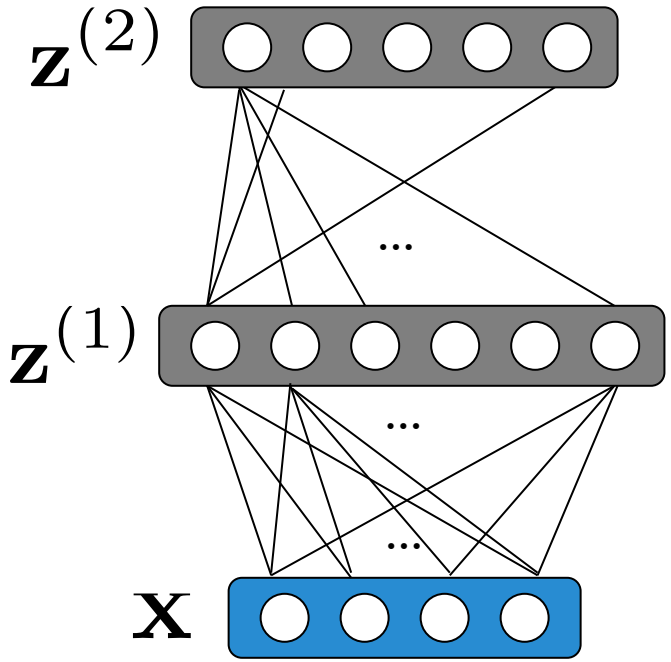
hidden units

$$\mathbf{z}^{(1)} = g\left(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

layer index

$$\mathbf{z}^{(2)} = g\left(\mathbf{W}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}\right)$$

$$\ldots$$



- Use output of one layer as input to the next
- *Feed-forward* and/or *fully-connected* layers
- Also called *multi-layer perceptron (MLP)*

Nonlinearities (activation function)

$$\mathbf{z}^{(1)} = g\left(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

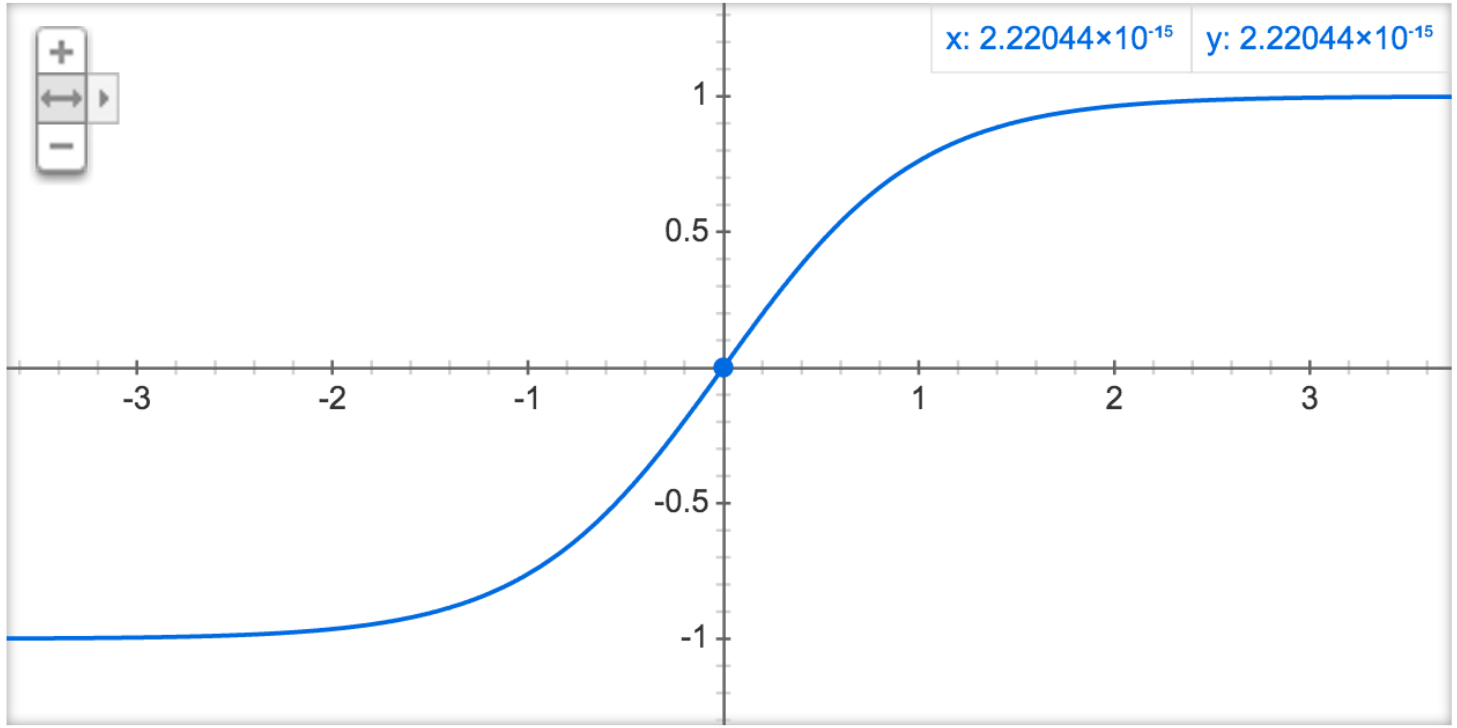
- g can be applied to each entry in a vector in an element-wise manner
- Common activation functions: tanh, sigmoid, and ReLU

Why nonlinearities?

- Otherwise stacking neural layers results in a simple affine transform.

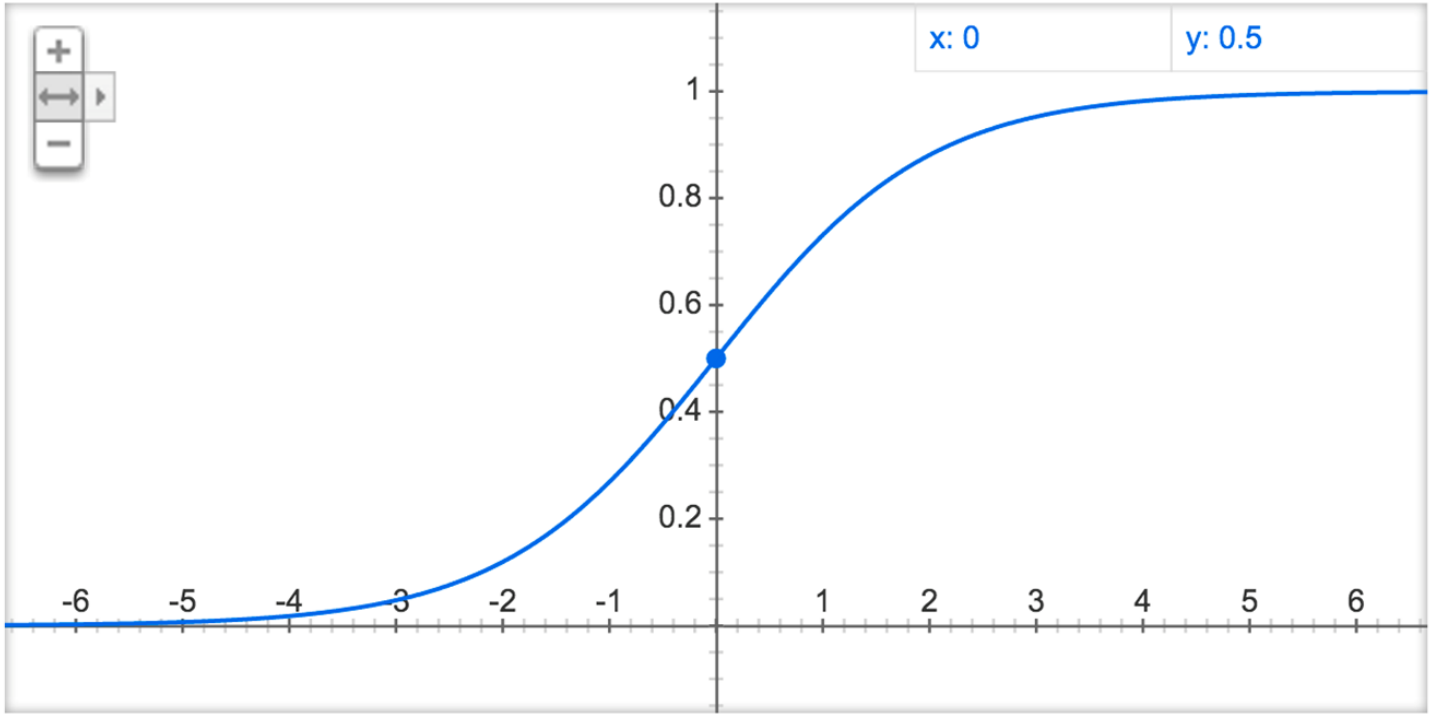
Nonlinearities: tanh

$$y = \tanh(x) = \frac{e^x + e^{-x}}{e^x - e^{-x}}$$



Nonlinearities: sigmoid

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



Sigmoid and tanh

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

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$$

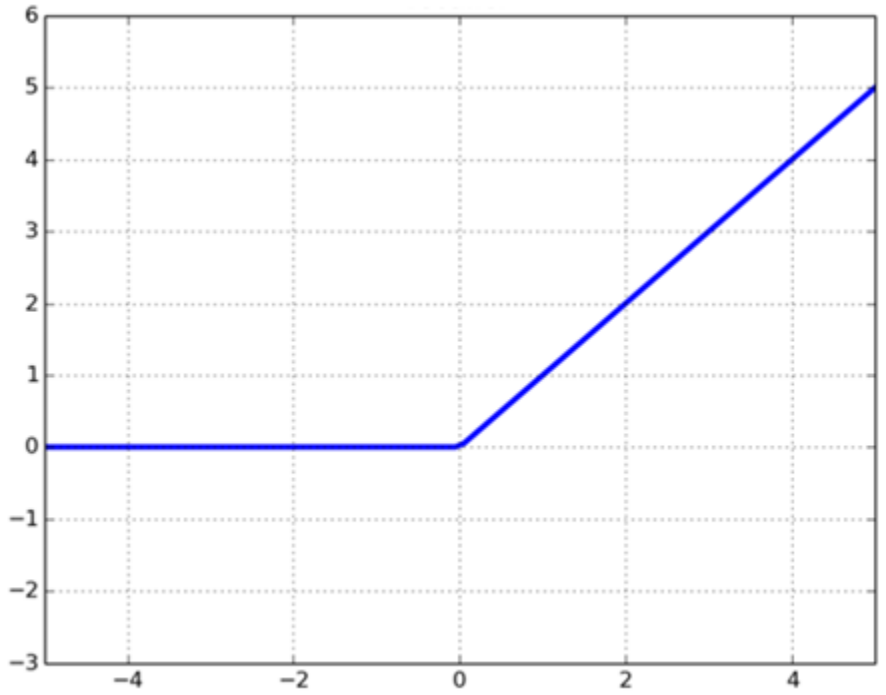
divide both sides by e^{x}

$$= \frac{1-e^{-2x}}{1+e^{-2x}}$$

$$= \sigma(2x) - (1-\sigma(2x)) = 2\sigma(2x) - 1$$

Nonlinearity: Rectified Linear Unit (ReLU)

$$y = \mathrm{ReLU}(x) = \max\{0, x\}$$



Sentiment Classification with Neural Network

$$\mathbf{z}^{(1)} = g\left(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{s} = \mathbf{W}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$

We empirically don't pass the final layer into an activation function

How can we get $\boldsymbol{x}$ for a sentence?

- Average word embeddings
- More complicated neural network structures

Neural Network: Training

$$\mathbf{s} = \begin{bmatrix} \text{score}(\mathbf{x}, 0; \mathbf{w}) \\ \text{score}(\mathbf{x}, 1; \mathbf{w}) \end{bmatrix} \overset{\text{softmax}}{\Rightarrow} \mathbf{p} = \begin{bmatrix} \frac{e^{\text{score}(\mathbf{x},0;\mathbf{w})}}{Z} \\ \frac{e^{\text{score}(\mathbf{x},1;\mathbf{w})}}{Z} \end{bmatrix}$$

$$Z = e^{\text{score}(\mathbf{x},0;\mathbf{w})} + e^{\text{score}(\mathbf{x},1;\mathbf{w})}$$

Maximize the probability (minimize negative log probability) of gold standard label

$$\text{loss} = -\log P(y \mid \mathbf{x}) = -\log \left(\frac{e^{\text{score}(\mathbf{x},y;\mathbf{w})}}{Z} \right)$$

Also called *cross entropy* (between $\mathbf{p}$ and the 1-hot gold standard distribution) loss.

Backpropagation

- Chain rule: suppose $y = f(x), z = g(y)$, then $\frac{\mathrm{d}z}{\mathrm{d}x} = \frac{\mathrm{d}z}{\mathrm{d}y} \cdot \frac{\mathrm{d}y}{\mathrm{d}x}$

$$\mathbf{z}^{(1)} = g\left(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

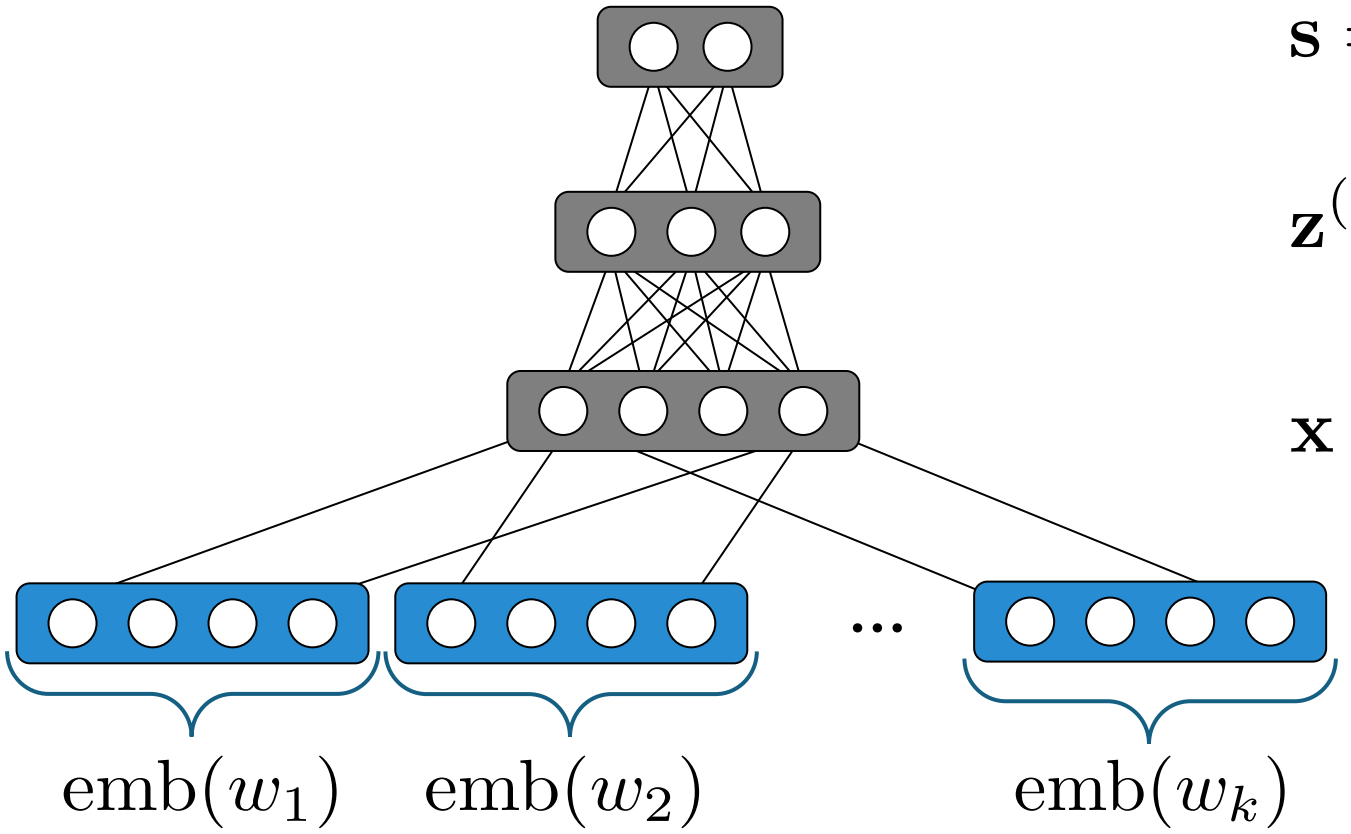
$$\mathbf{s} = \mathbf{W}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$

$$\mathrm{loss}(\mathbf{s}; \mathbf{x}, y) = -\log\left(\frac{e^{s_y}}{Z}\right)$$

Now we have $\frac{\partial \mathrm{loss}}{\partial \mathbf{s}}$, how should we update $\mathbf{W}^{(0)}$?

$$\frac{\partial \mathrm{loss}}{\partial \mathbf{W}^{(0)}} = \frac{\partial \mathrm{loss}}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \mathbf{z}^{(1)}} \frac{\partial \mathbf{z}^{(1)}}{\partial \mathbf{W}^{(0)}}$$

Visualization of an Average Word Vector Classifier



$$\mathbf{s} = \mathbf{W}^{(1)}\mathbf{z}^{(1)} + \mathbf{b}^{(1)}$$

$$\mathbf{z}^{(1)} = g\left(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\mathbf{x} = \frac{1}{k}\sum_{i=1}^{k}\text{emb}(w_k)$$

Next

Common neural network architectures for NLP (for more powerful models).