WEB INTELLIGENCE

Lecture 7: Word Embeddings and Neural Models

Russa Biswas

October 28, 2025

AALBORG UNIVERSITY

(Based on slides of Dan Jurafsky)

Word2vec – Recap

Quiz: https://www.menti.com/al8v8hicam3n

Word2vec – Recap

...lemon, a [tablespoon of apricot jam, a] pinch...

c1 c2 [target] c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

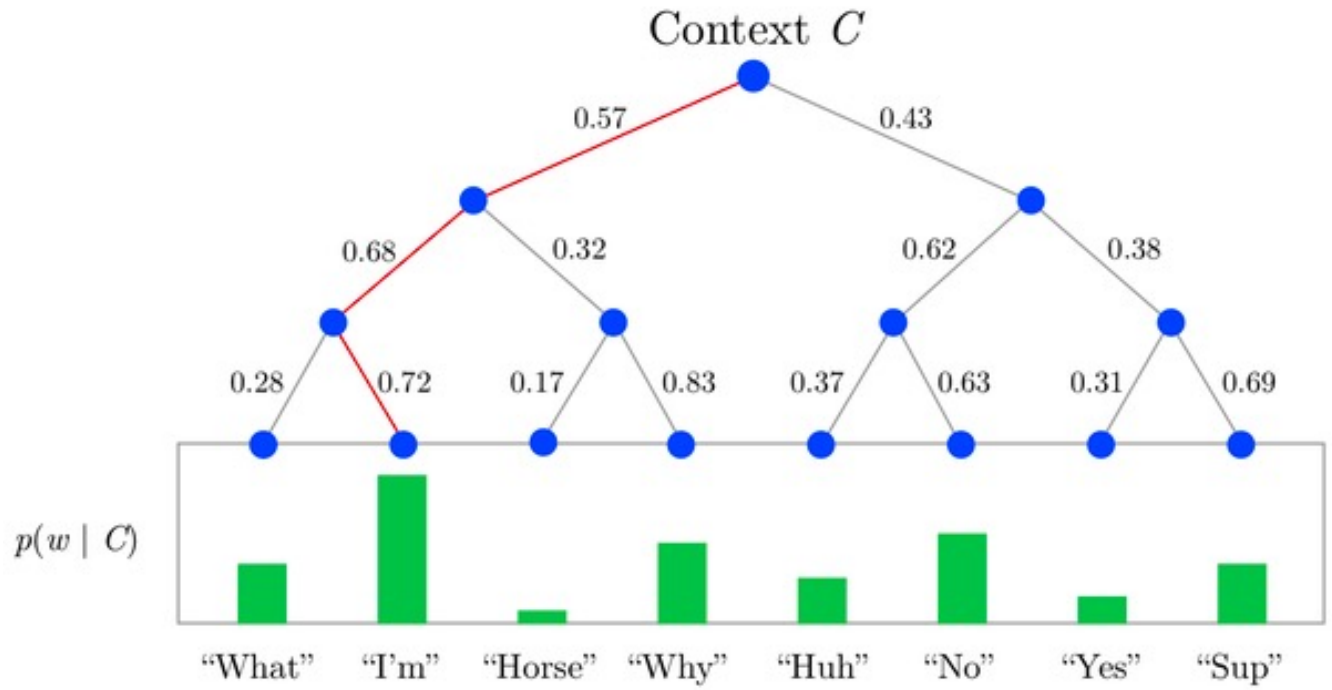
negative examples -

t	c	t	c
apricot	aardvark	apricot	seven
apricot	my	apricot	forever
apricot	where	apricot	dear
apricot	coaxial	apricot	if

Maximize the similarity of the target with the actual context words, and **minimize** the similarity of the target with the *k* negative sampled non-neighbor words.

$$
\begin{aligned}
L_{CE} &= -\log\left[P(+|w,c_{pos})\prod_{i=1}^{k}P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log P(-|w,c_{neg_i})\right] \\
&= -\left[\log P(+|w,c_{pos})+\sum_{i=1}^{k}\log\left(1-P(+|w,c_{neg_i})\right)\right] \\
&= -\left[\log \sigma(c_{pos}\cdot w)+\sum_{i=1}^{k}\log\sigma(-c_{neg_i}\cdot w)\right]
\end{aligned}
$$

Word2vec – Hierarchical Softmax



- **Hierarchical Softmax** is an alternative to softmax that is faster to evaluate:
  - it is O(log n)
  - time to evaluate compared to O(n) for softmax.
  - It utilises a multi-layer binary tree, where the probability of a word
  - is calculated through the product of probabilities on each edge on the path to that node.

Word2vec – Contd.

- The kinds of neighbors depend on window size
- **Small windows** (C= +/- 2) : nearest words are syntactically similar words in same taxonomy
  - *Hogwarts* nearest neighbors are other fictional schools: *Sunnydale, Evernight, Blandings*
- **Large windows** (C= +/- 5) : nearest words are related words in same semantic field
  - *Hogwarts* nearest neighbors are Harry Potter world: *Dumbledore, half-blood, Malfoy*
- Optimal Window size for large corpus:
- Optimal Window size for smaller corpus:

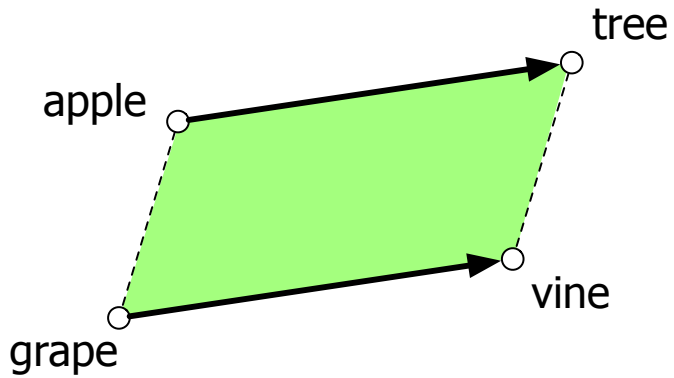
Properties of Word Embeddings

Analogical relations

The classic parallelogram model of analogical reasoning (Rumelhart and Abrahamson 1973)

To solve: *"apple is to tree as grape is to _____"*

Add tree – apple to grape to get vine



Analogical relations via parallelogram:

The parallelogram method can solve analogies with both sparse and dense embeddings (Turney and Littman 2005, Mikolov et al. 2013b)

king – man + woman is close to queen

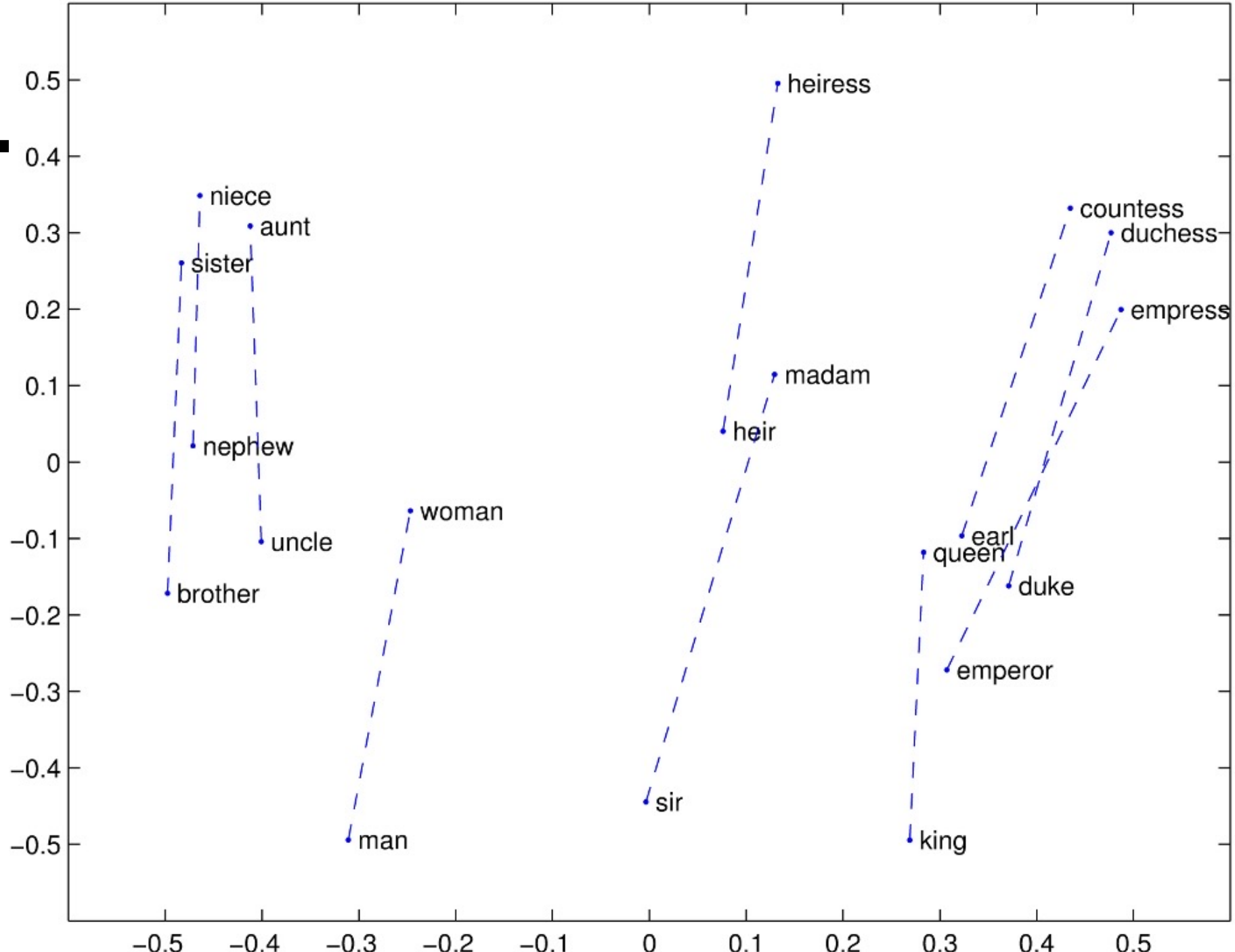
Paris – France + Italy is close to Rome

For a problem a:a*::b:b*, the parallelogram method is:

$$\hat{b}^* = \underset{x}{\text{argmax}}\ \text{distance}(x, a^* - a + b)$$

It only seems to work for frequent words, small distances and certain relations (relating countries to capitals, or parts of speech), but not others. (Linzen 2016, Gladkova et al. 2016, Ethayarajh et al. 2019a)

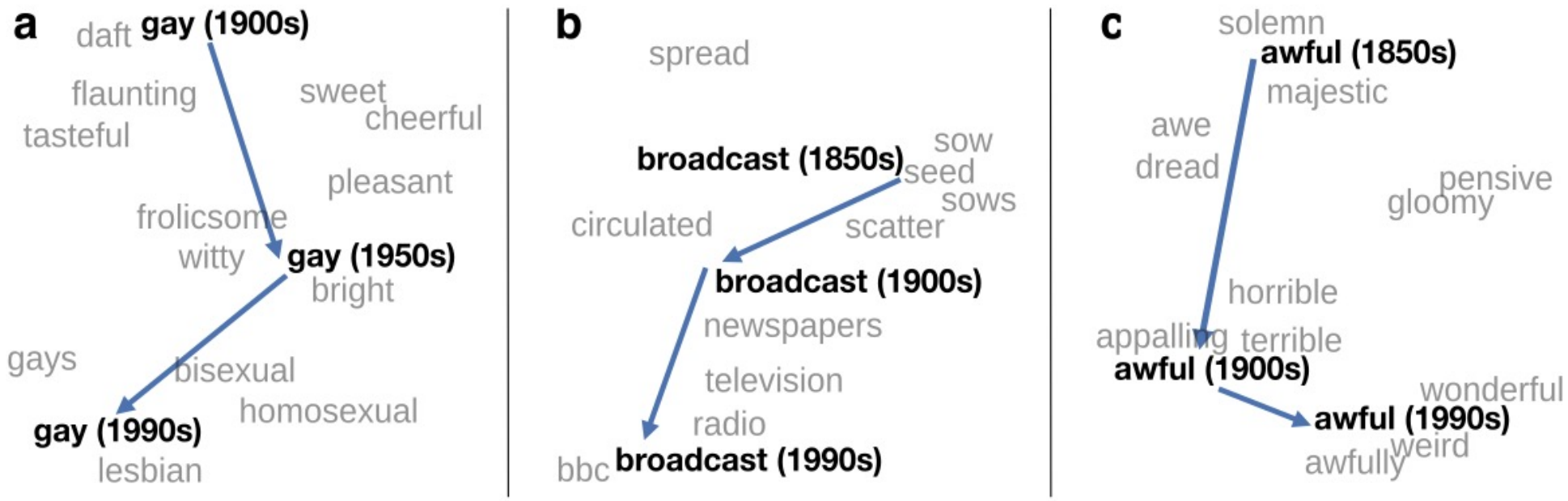
Understanding analogy is an open area of research (Peterson et al. 2020)


0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
niece
aunt
sister
nephew
uncle
brother
woman
man
heiress
madam
heir
sir
countess
duchess
empress
earl
queen
duke
emperor
king

Word Embeddings – Use Cases

Train embeddings on different decades of historical text to see meanings shift

~30 million books, 1850-1990, Google Books data



William L. Hamilton, Jure Leskovec, and Dan Jurafsky. 2016. Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. Proceedings of ACL.

Word Embeddings – Use Cases

Embeddings to reflect Cultural Bias

- Ask “Paris : France :: Tokyo : x”
  - x = Japan
- Ask “father : doctor :: mother : x”
  - x = nurse
- Ask “man : computer programmer :: woman : x”
  - x = homemaker

Algorithms that use embeddings as part of e.g., hiring searches for programmers, might lead to bias in hiring

Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *NeurIPS*, pp. 4349-4357. 2016.

Word Embeddings – Use Cases

Historical embedding as a tool to study cultural biases

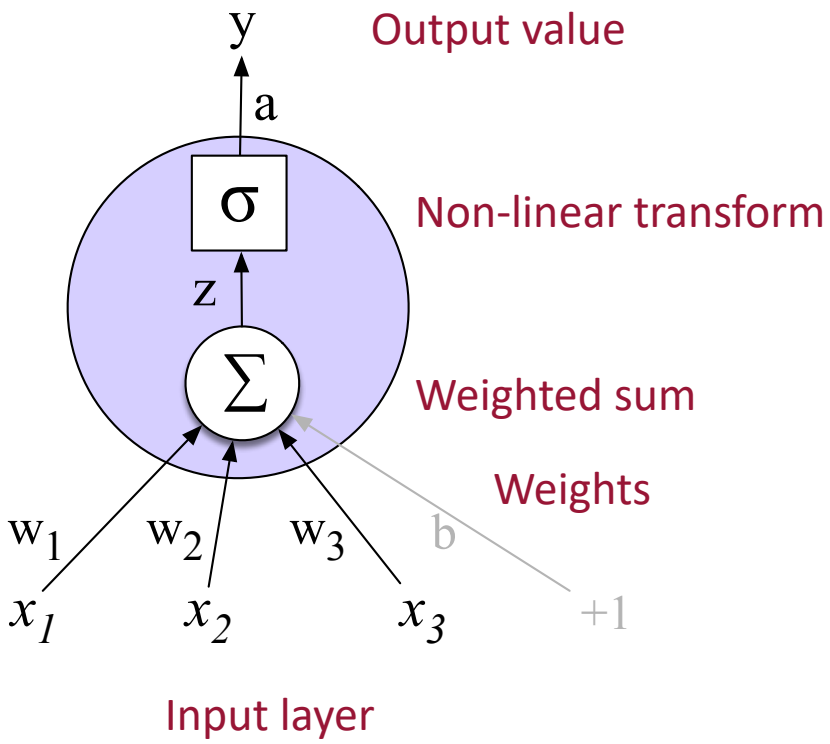
- Compute a **gender or ethnic bias** for each adjective: e.g., how much closer the adjective is to "woman" synonyms than "man" synonyms, or names of particular ethnicities
  - Embeddings for **competence** adjective (*smart, wise, brilliant, resourceful, thoughtful, logical)* are biased toward men, a bias slowly decreasing 1960-1990
  - Embeddings for **dehumanizing** adjectives (barbaric, monstrous, bizarre) were biased toward Asians in the 1930s, bias decreasing over the 20th century.
- These match the results of old surveys done in the 1930s

Garg, N., Schiebinger, L., Jurafsky, D., and Zou, J. (2018). Word embeddings quantify 100 years of gender and ethnic stereotypes. Proceedings of the National Academy of Sciences 115(16), E3635–E3644.

Neural Networks

https://www.menti.com/alijgjqntwsa

Neural Network Unit



- Take weighted sum of inputs, plus a bias

$$z = b + \sum_i w_i x_i$$

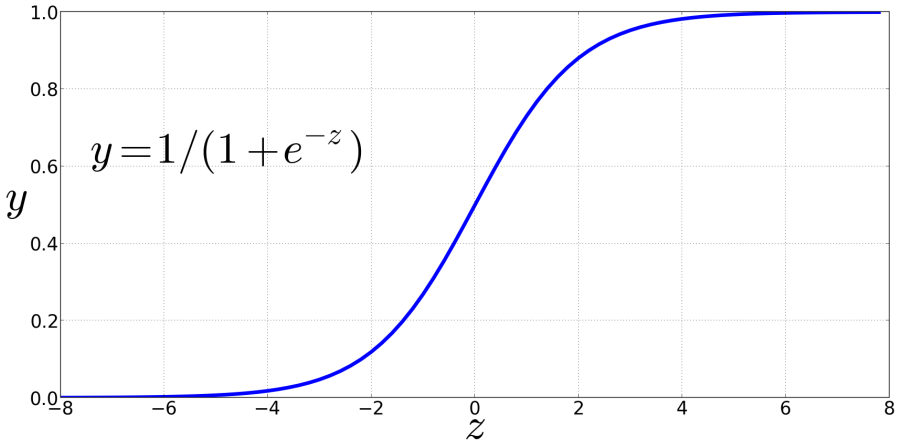
$$z = w \cdot x + b$$

- Instead of just using z, we'll apply a nonlinear activation function f:

$$y = a = f(z)$$

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

Non-linear Activation Function

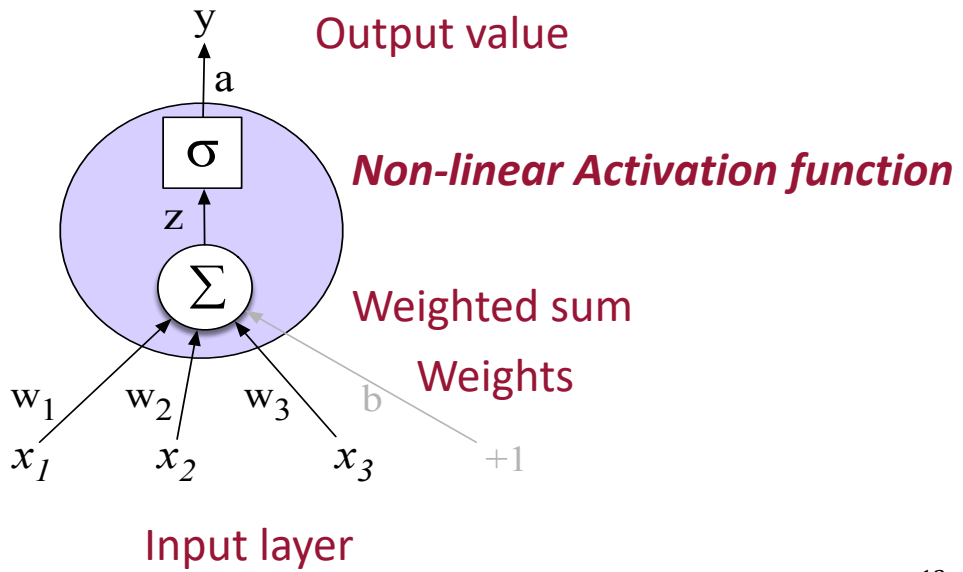


Non Linear Activation Function

- Sigmoid has a number of advantages;
  - it maps the output into the range [0,1], which is useful in squashing outliers toward 0 or 1.
  - It's differentiable, which is handy for learning

- Function that a simple unit with a sigmoid activation function is computing from an input x, a weight vector w, and a bias term b is given by

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + \exp(-(w \cdot x + b))}$$



Non Linear Activation Function - Example

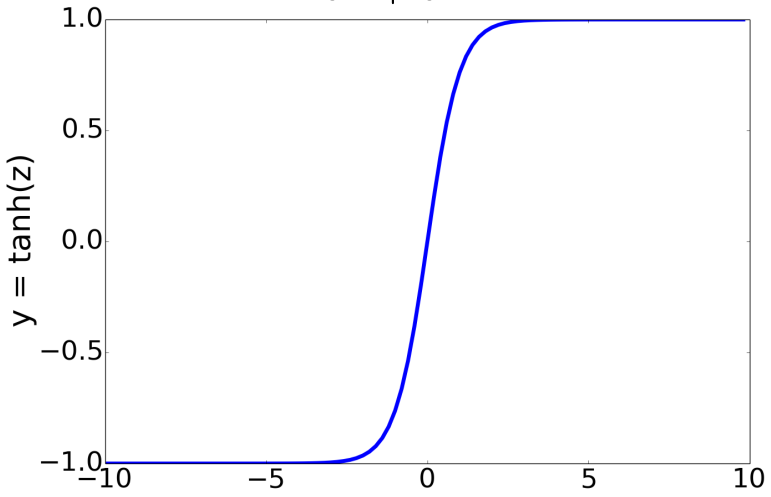
- *Suppose a unit has:*
- `w = [0.2,0.3,0.9]`
- `b = 0.5`
- What happens with input x:
- `x = [0.5,0.6,0.1]`

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

$$\frac{1}{1 + e^{-(.5*.2+.6*.3+.1*.9+.5)}} = \frac{1}{1 + e^{-0.87}} = .70$$

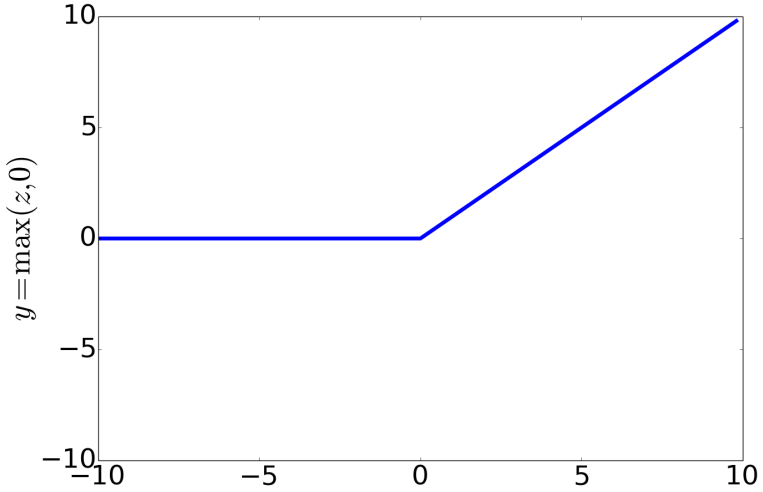
Other Non Linear Activation Function

$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



tanh [-1,+1]

$$y = max(z, 0)$$



ReLU - Rectified Linear Unit

The tanh function has the nice properties of being smoothly differentiable and mapping outlier values toward the mean.

In the sigmoid or tanh functions, very high values of *z* result in values of *y* that are **saturated**, i.e., extremely close to 1 and have derivatives very close to 0

Zero derivatives cause problems for learning: gradients that are almost 0 cause the error signal to get smaller and smaller until it is too small to be used for training, a problem called the **vanishing gradient problem**

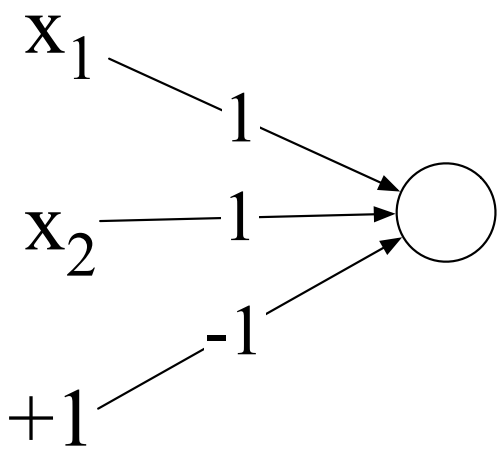
The derivative of ReLU for high values of *z* is 1 rather than very close to 0.

Perceptrons

- A very simple neural unit
  - Binary output (0 or 1)
  - No non-linear activation function

$$y = \begin{cases} 0, & \text{if } w \cdot x + b \leq 0 \\ 1, & \text{if } w \cdot x + b > 0 \end{cases}$$

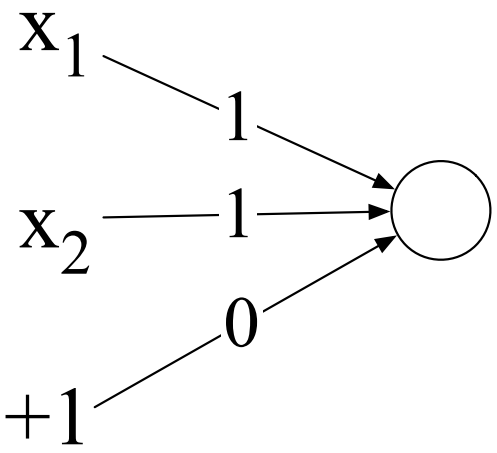
AND



AND

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1

OR



OR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	1

Perceptrons

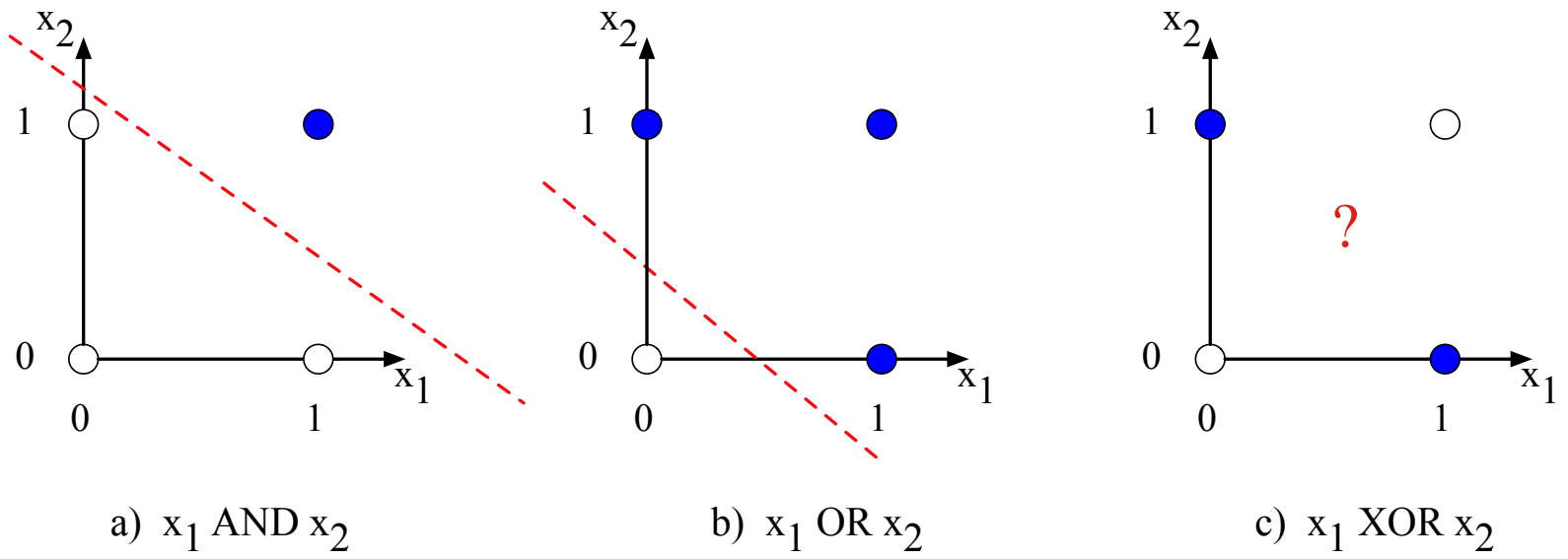
- Compute for the XOR function

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

XOR Problem

- Perceptron equation given x_1 and x_2, is the equation of a line. $w_1x_1 + w_2x_2 + b = 0$
- (in standard linear format: $x_2 = (-w_1/w_2)x_1 + (-b/w_2)$)
- This line acts as a **decision boundary**
  - 0 if input is on one side of the line
  - 1 if on the other side of the line

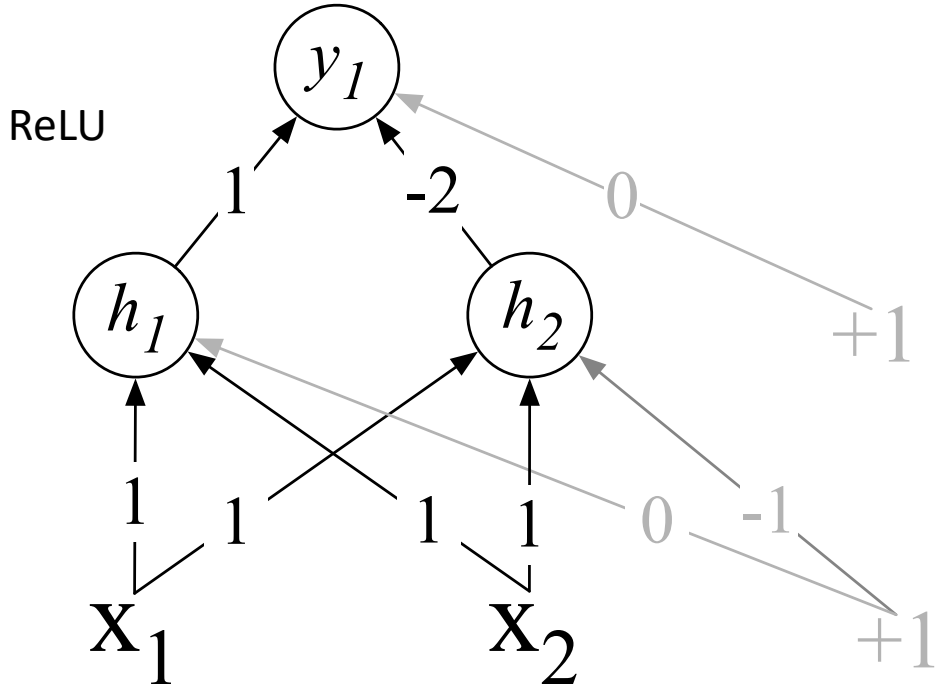
XOR is not a **linearly separable** function



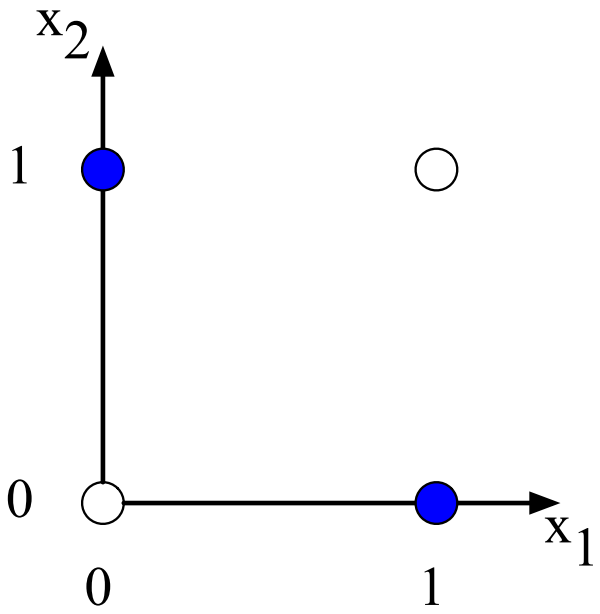
Solution of XOR

- XOR **can't** be calculated by a single perceptron
- XOR **can** be calculated by a layered network of units.

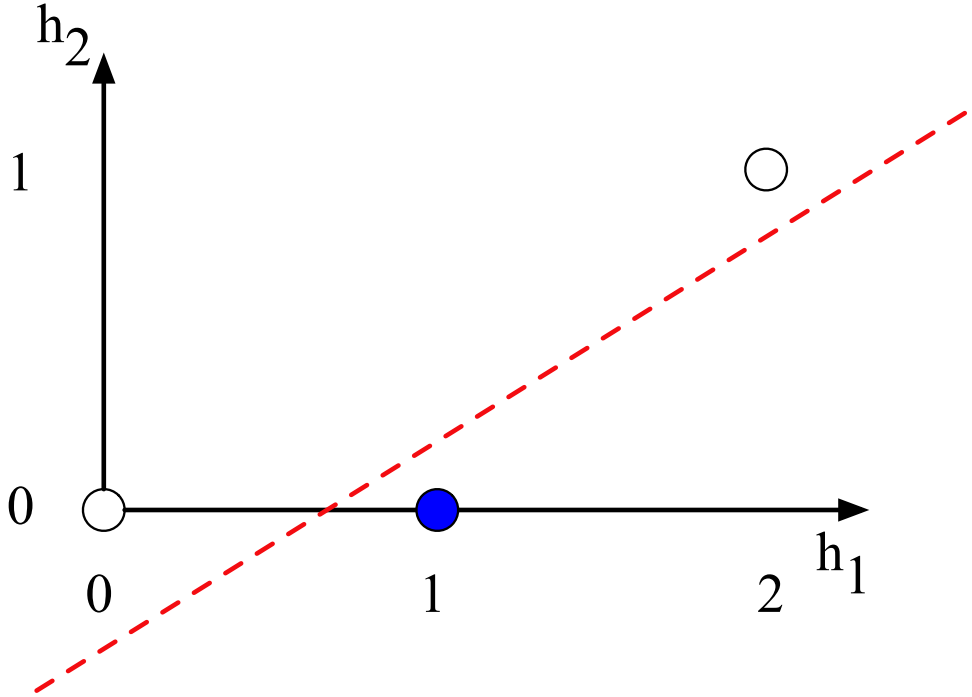
XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



XOR



a) The original *x* space



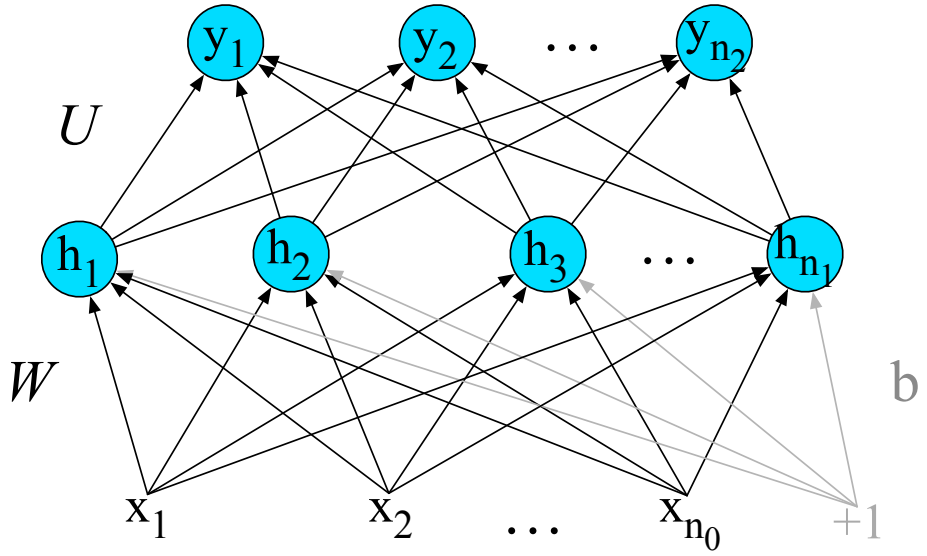
b) The new (linearly separable) *h* space

(With learning: hidden layers will learn to form useful representations)

Feed Forward Neural Networks

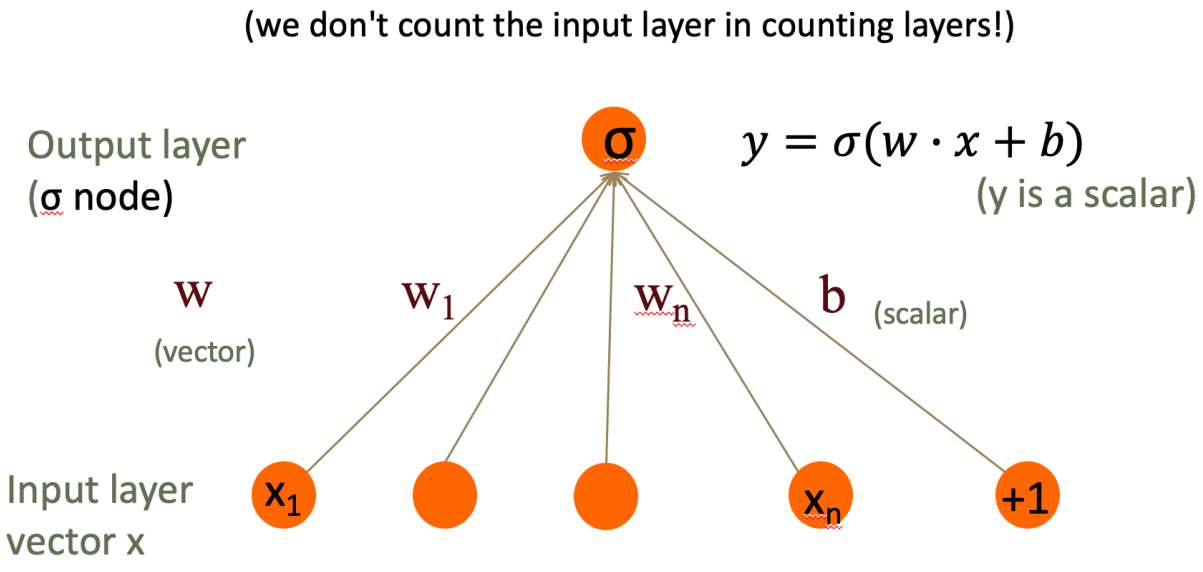
Can also be called **multi-layer perceptrons** (or **MLPs**) for historical reasons

A feedforward network is a multilayer network in which the units are connected **with no cycles**; the outputs from units in each layer are passed to units in the next higher layer, and no outputs are passed back to lower layers

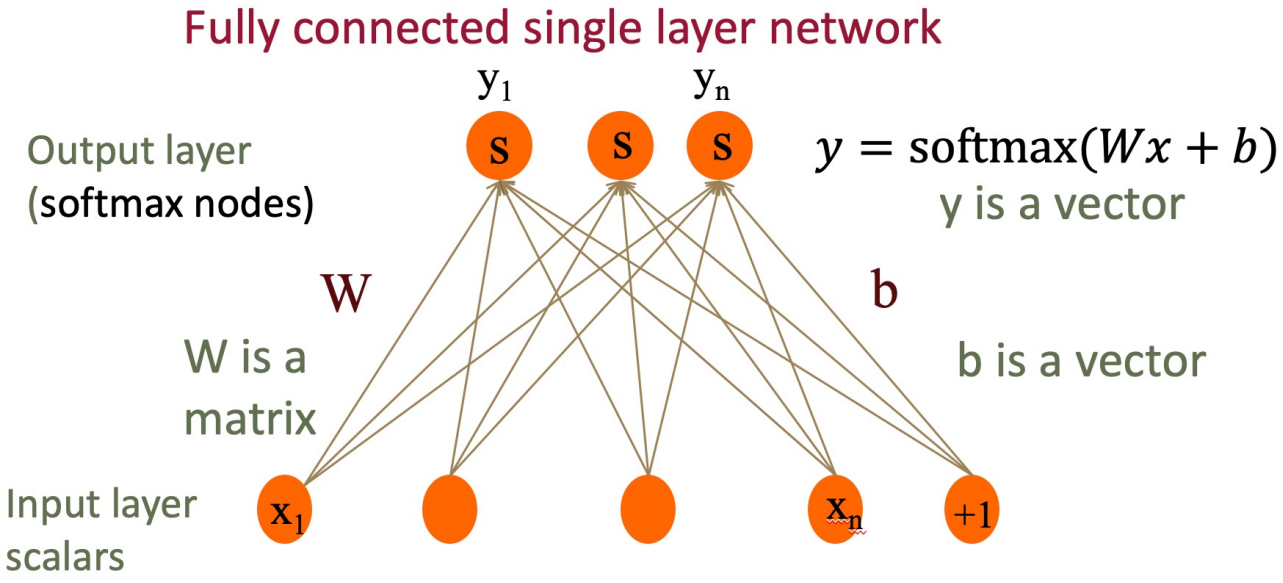


Logistic Regression and Neural Networks

Logistic Regression as 1-layer Network



Multinomial Logistic Regression as a 1-layer Network



Softmax: A Generalization of Sigmoid

For a vector *z* of dimensionality *k*, the softmax is:

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \leq i \leq k$$

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

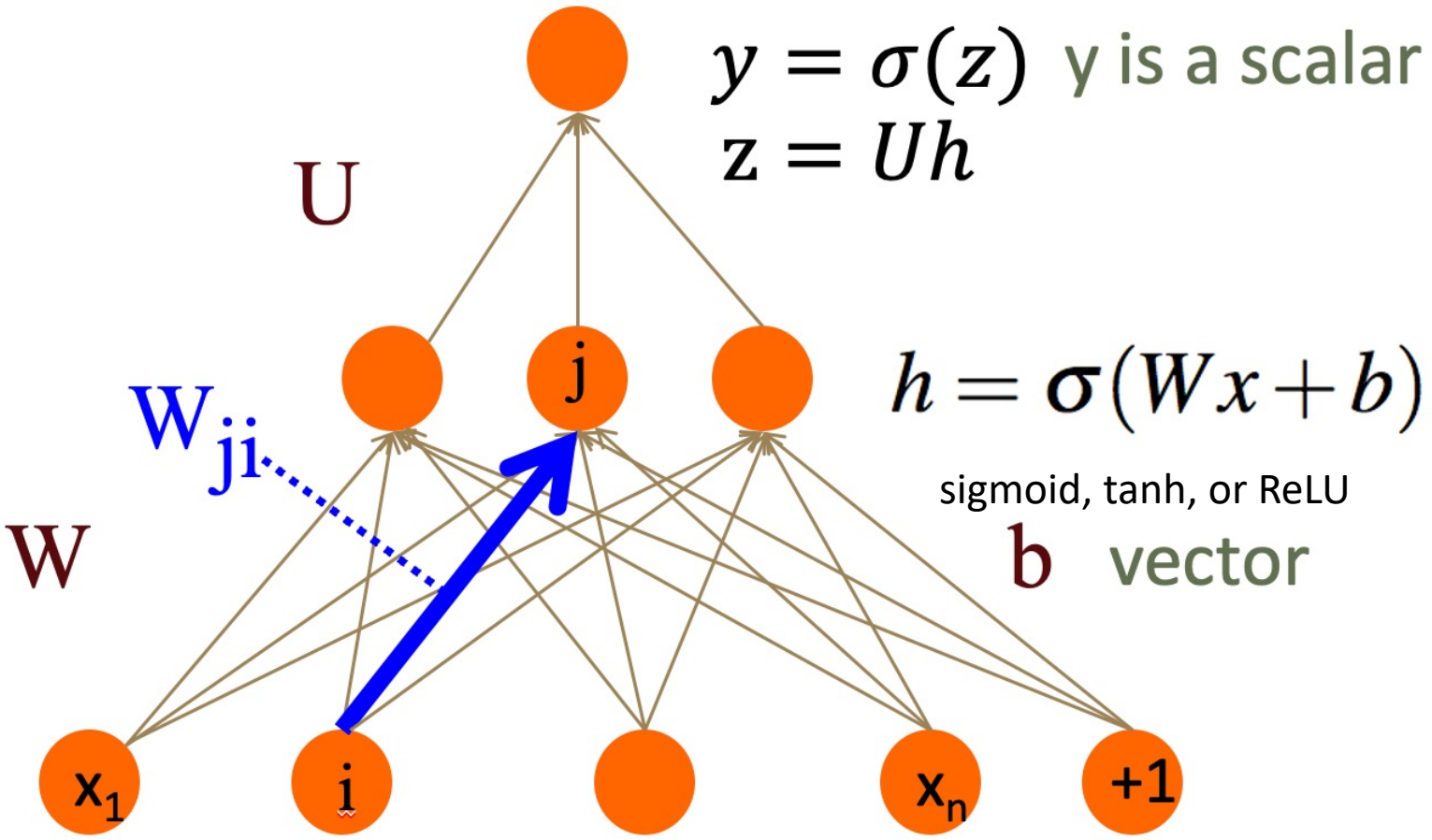
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Two-layer Network

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



$y = \sigma(z)$ y is a scalar

$z = Uh$

$h = \sigma(Wx + b)$

sigmoid, tanh, or ReLU

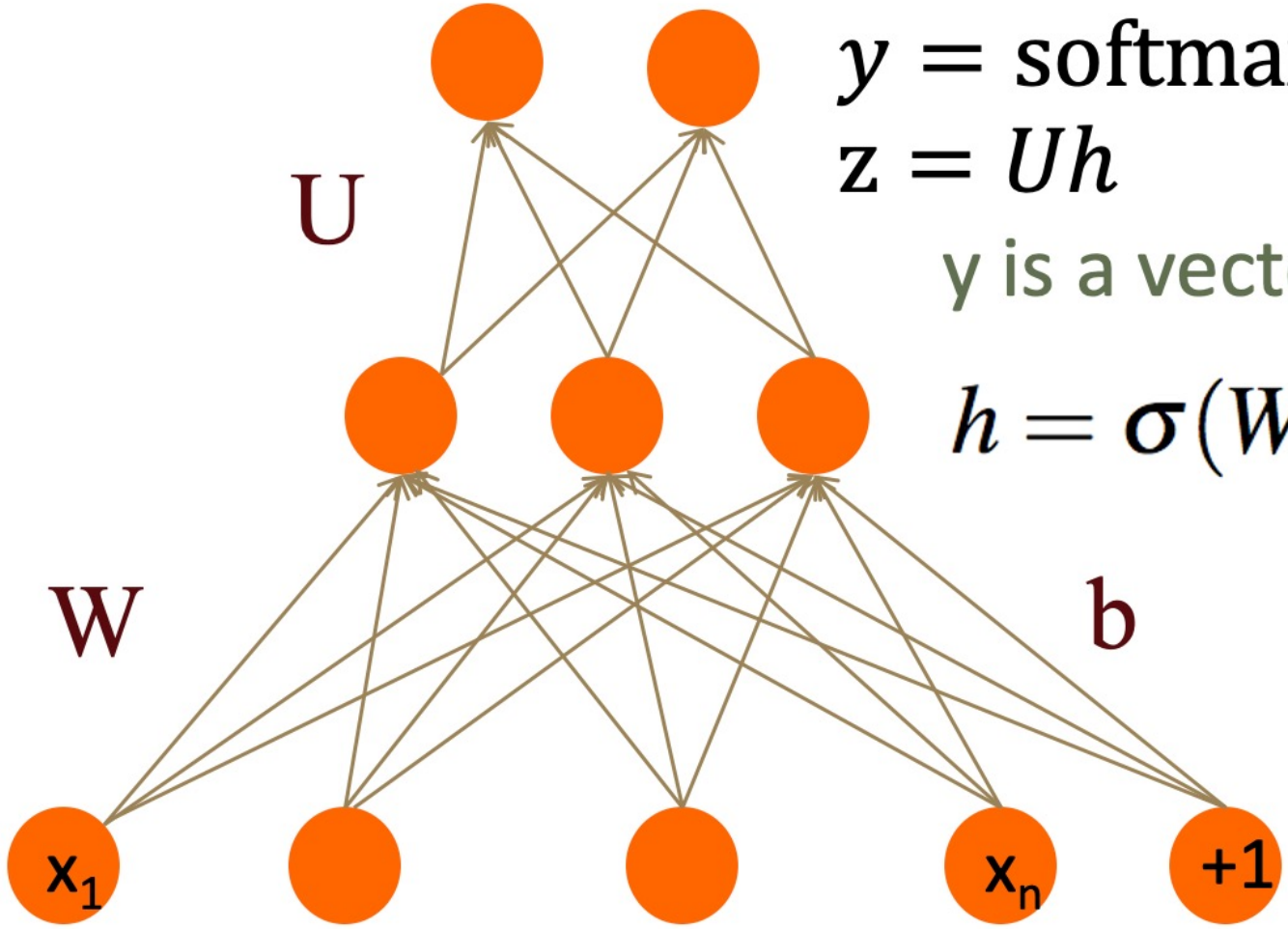
b vector

Two Layer Network

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



$$y = \text{softmax}(z)$$
$$z = Uh$$

y is a vector

$$h = \sigma(Wx + b)$$

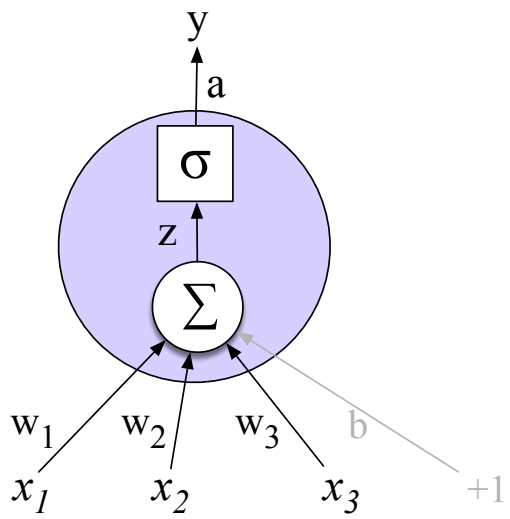
Could be ReLU
Or tanh

Multi-layer Notion

$$
\begin{aligned}
z^{[1]} &= W^{[1]} a^{[0]} + b^{[1]} \\
a^{[1]} &= g^{[1]}(z^{[1]}) \\
z^{[2]} &= W^{[2]} a^{[1]} + b^{[2]} \\
a^{[2]} &= g^{[2]}(z^{[2]}) \\
\hat{y} &= a^{[2]}
\end{aligned}
$$

for i **in** 1..n

$$
\begin{aligned}
z^{[i]} &= W^{[i]} a^{[i-1]} + b^{[i]} \\
a^{[i]} &= g^{[i]}(z^{[i]}) \\
\hat{y} &= a^{[n]}
\end{aligned}
$$



Replacing the bias : Let's switch to a notation without the bias unit

Just a notational change

1. Add a dummy node a_0=1 to each layer
2. Its weight w_0 will be the bias
3. So input layer $a^{[0]}_0$=1,
   - And $a^{[1]}_0$=1 , $a^{[2]}_0$=1,…

Replacing Bias

Instead of:

$$x = x_1, x_2, \ldots, x_{n0}$$

$$h = \sigma(Wx + b)$$

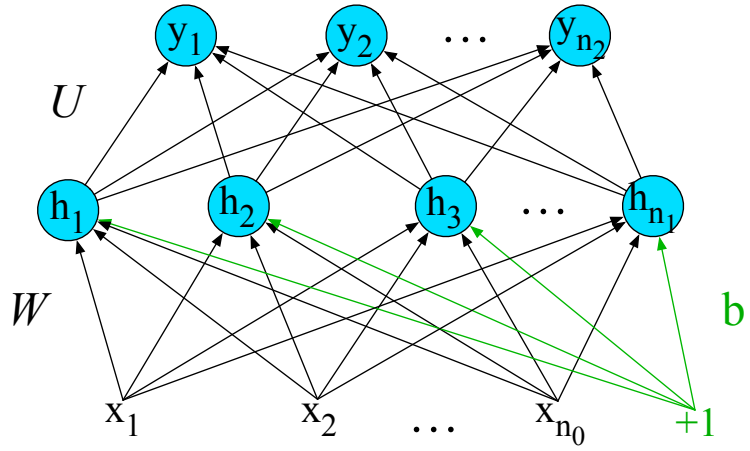
$$h_j = \sigma\left(\sum_{i=1}^{n_0} W_{ji}x_i + b_j\right)$$

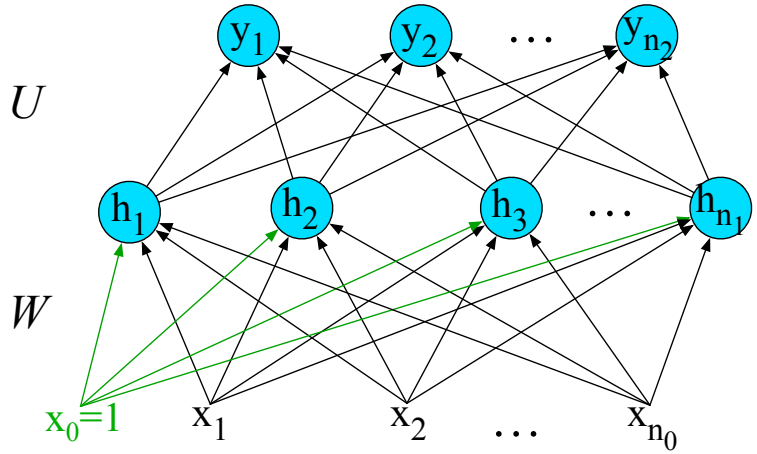
We'll do this:

$$x = x_0, x_1, x_2, \ldots, x_{n0}$$

$$h = \sigma(Wx)$$

$$\sigma\left(\sum_{i=0}^{n_0} W_{ji}x_i\right)$$





Feed Forward Networks – Use Cases

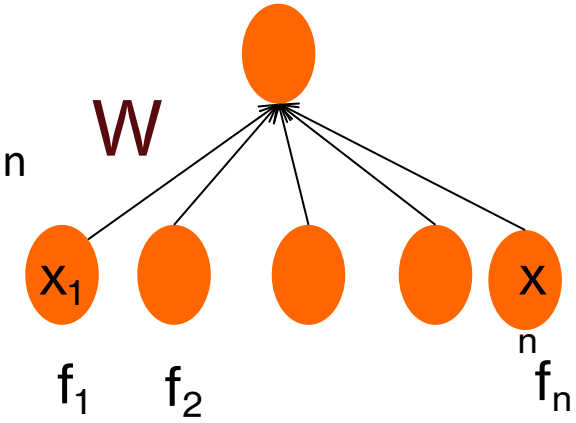
Text Classification

- We could do exactly what we did with logistic regression
- Input layer are binary features as before
- Output layer is 0 or 1

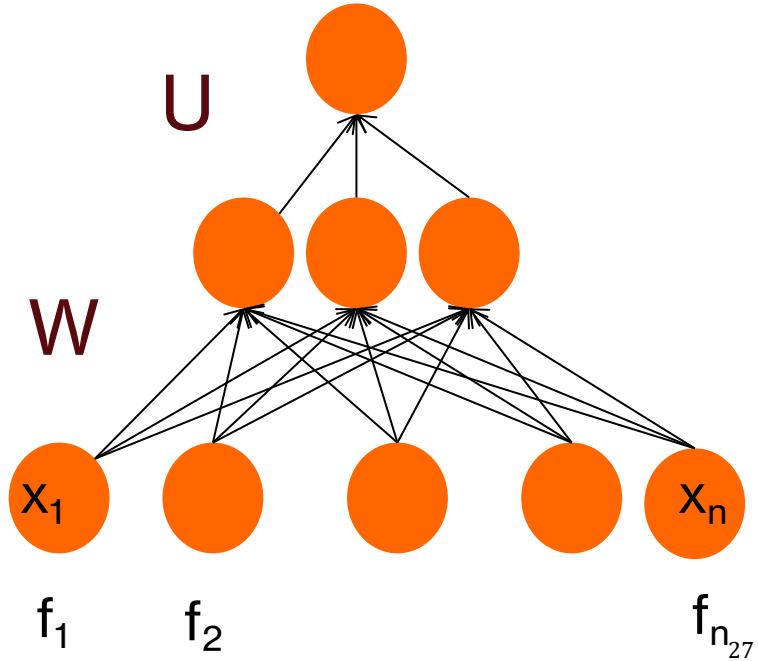
Problem: Sentiment Analysis

Var	Definition
x_1	count(positive lexicon) $\in$ doc)
x_2	count(negative lexicon) $\in$ doc)
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_4	count(1st and 2nd pronouns $\in$ doc)
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$
x_6	log(word count of doc)

Logistic Regression



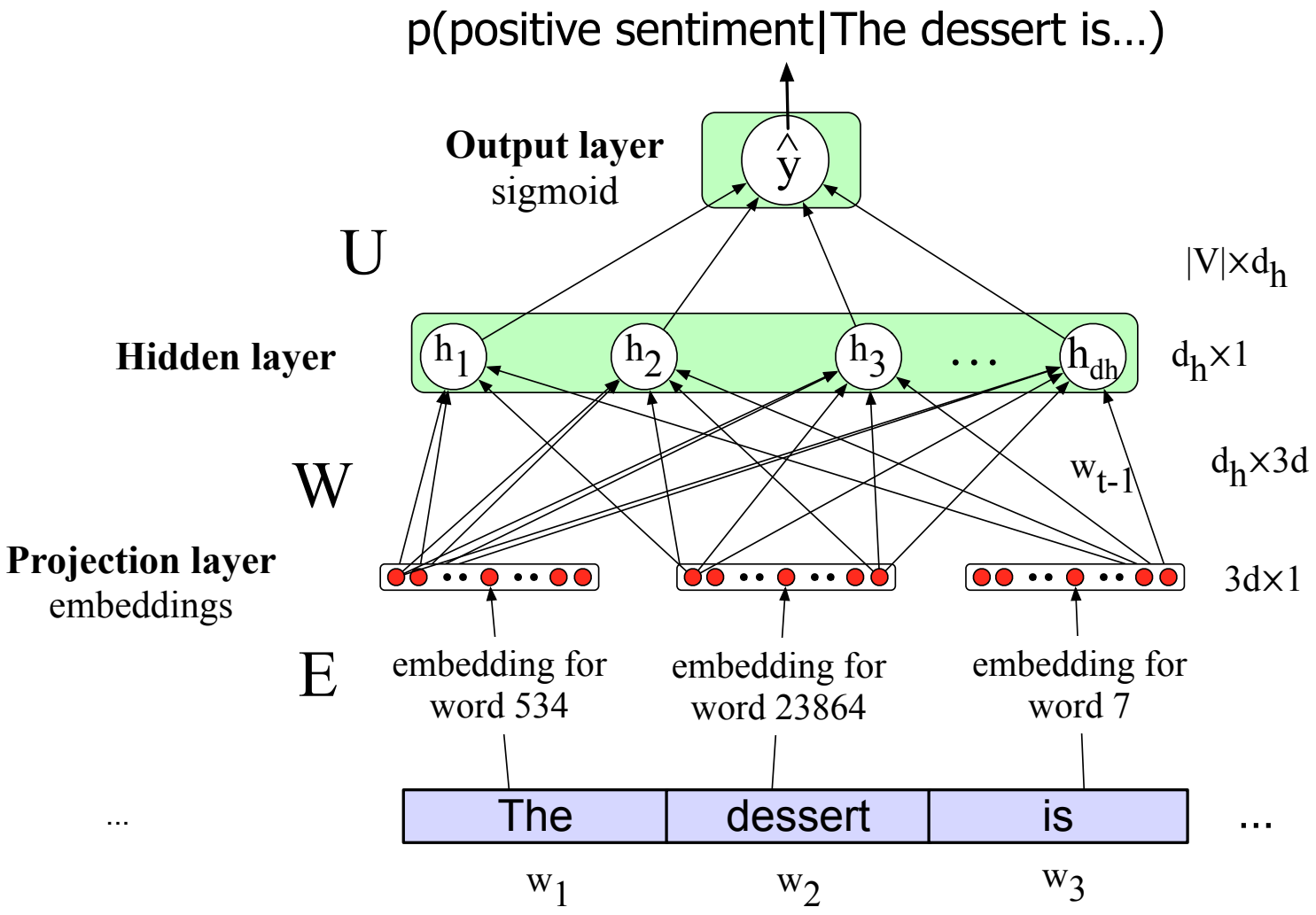
2-layer feedforward network



Text Classification

- Just adding a hidden layer to logistic regression
- allows the network to use non-linear interactions between features
- which may (or may not) improve performance.
- Instead of using hand-built human-engineered features for classification**:** ability to **learn** features from the data
  - **Use learned representations like embeddings**!

Text Classification with Embeddings as inputs

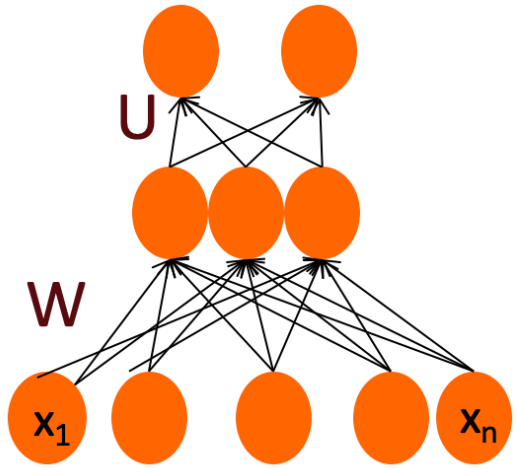


Challenges

- This assumes a fixed size length (3)!
- Kind of unrealistic.
- Some simple solutions (more sophisticated solutions later)

1. Make the input the length of the longest review
   - If shorter then pad with zero embeddings
   - Truncate if you get longer reviews at test time
2. Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
   - Take the mean of all the word embeddings
   - Take the element-wise max of all the word embeddings
     - For each dimension, pick the max value from all words

- What if you have more than two output classes?
  - Add more output units (one for each class)
  - And use a “softmax layer”



Neural Language Models (LMs)

- **Language Modeling**: Calculating the probability of the next word in a sequence given some history.
  - We've seen N-gram based LMs
  - But neural network LMs far **outperform** n-gram language models
- State-of-the-art neural LMs are based on more powerful neural network technology like Transformers
- But **simple feedforward LMs** can do almost as well!

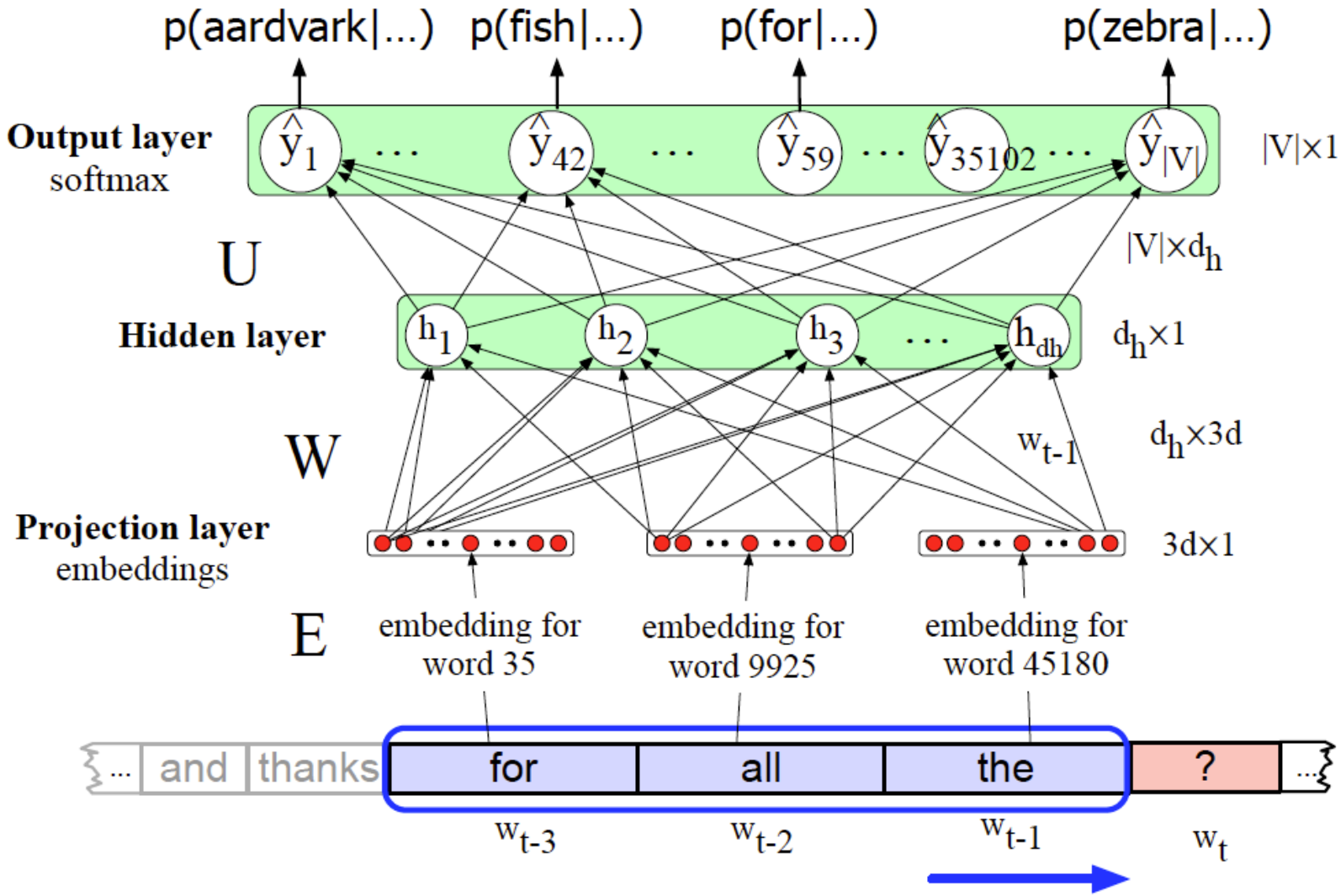
Task: predict next word w_t given prior words w_{t-1}, w_{t-2}, w_{t-3}, …

Problem: Now we're dealing with sequences of arbitrary length.

Solution: Sliding windows (of fixed length)

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

Neural Language Model



Neural Language Models – Why better?

Training data: I have to make sure that the cat gets fed.

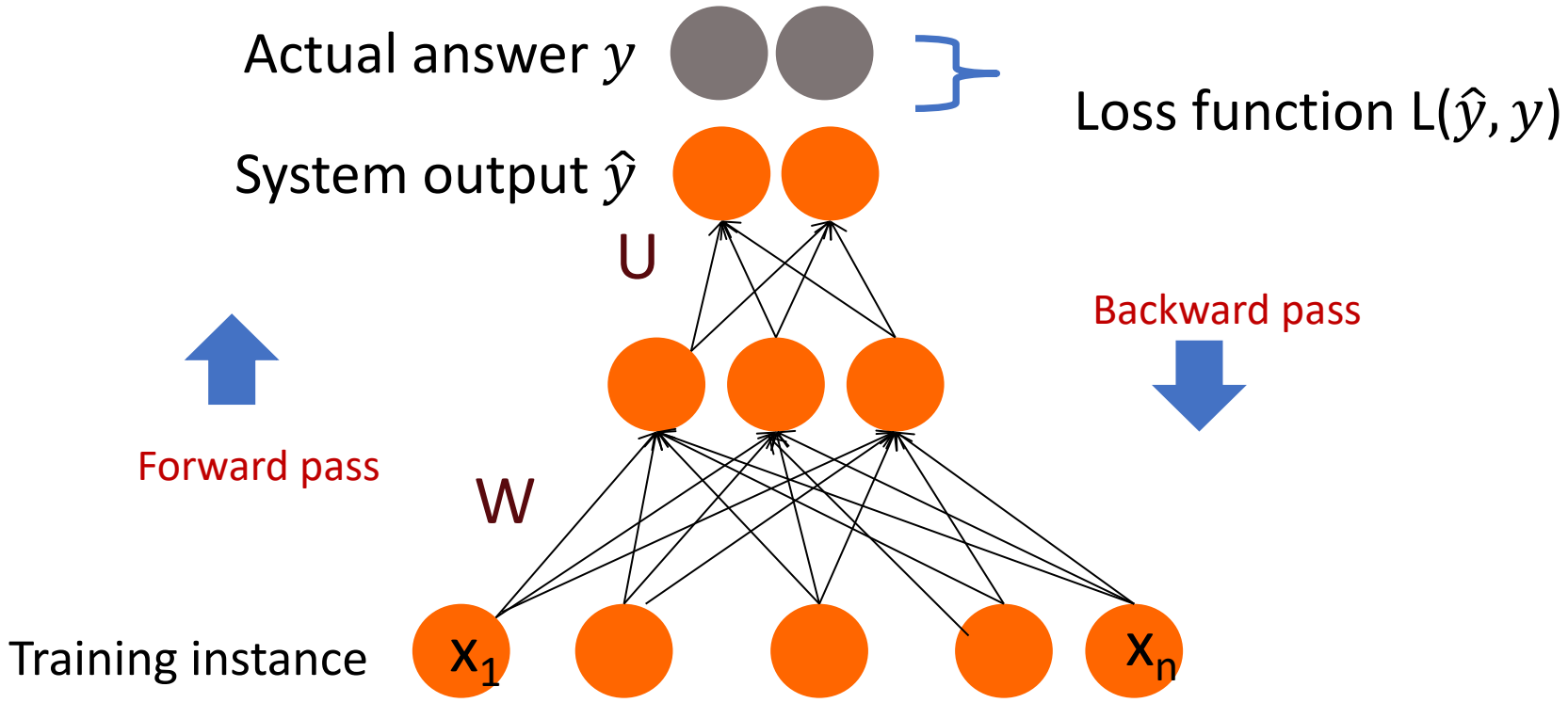
Test data: I forgot to make sure that the dog gets ___

Never seen: dog gets fed

N-gram LM can't predict "fed"!

Neural LM can use similarity of "cat" and "dog" embeddings to generalize and predict “fed” after dog

Training 2-layer Networks



Training 2-layer Networks

- For every training tuple (x, y)
  - Run *forward* computation to find our estimate $\hat{y}$
  - Run *backward* computation to update weights:
    - For every output node
      - Compute loss L between true y and the estimated $\hat{y}$
      - For every weight w from hidden layer to the output layer
        - Update the weight
    - For every hidden node
      - Assess how much blame it deserves for the current answer
      - For every weight w from input layer to the hidden layer
        - Update the weight

Loss function

- A measure for how far off the current answer is to the right answer
- Cross entropy loss for logistic regression:

$$L_{\text{CE}}(\hat{y}, y) = -\log p(y|x) = -[y \log \hat{y} + (1-y)\log(1-\hat{y})]$$

$$= -[y \log \sigma(w \cdot x + b) + (1-y)\log(1-\sigma(w \cdot x + b))]$$

Literature

- Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press. 2008.
- https://stanford-cs324.github.io/winter2022/lectures/introduction/
- https://web.stanford.edu/~jurafsky/slp3/ed3bookaug20_2024.pdf