Department of Linguistics and Translation
香港城市大學
City University of Hong Kong

# Computational Linguistics
# LT3233

Jixing Li

## Lecture 11: Feedforward Neural Network with Word Embeddings

Slides adapted from Dan Jurafsky

# Lecture plan

- Classification with FFNN
- Language model with FFNN
- PyTorch implementation
- Short break (15 mins)
- Hands-on exercises

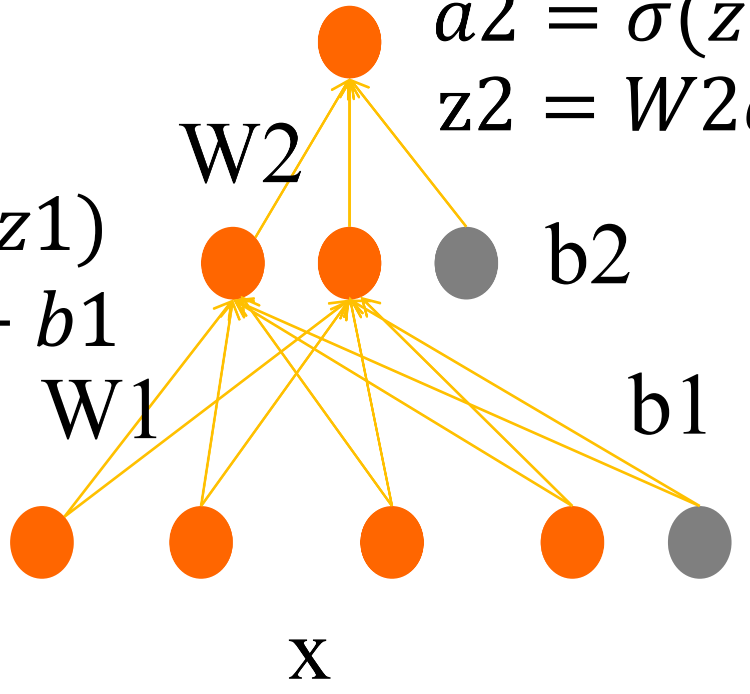
# Use cases for feedforward neural networks

1. **Text classification**
2. **Language modeling**

State-of-the-art systems use more powerful neural architectures, but simple models are useful to consider!

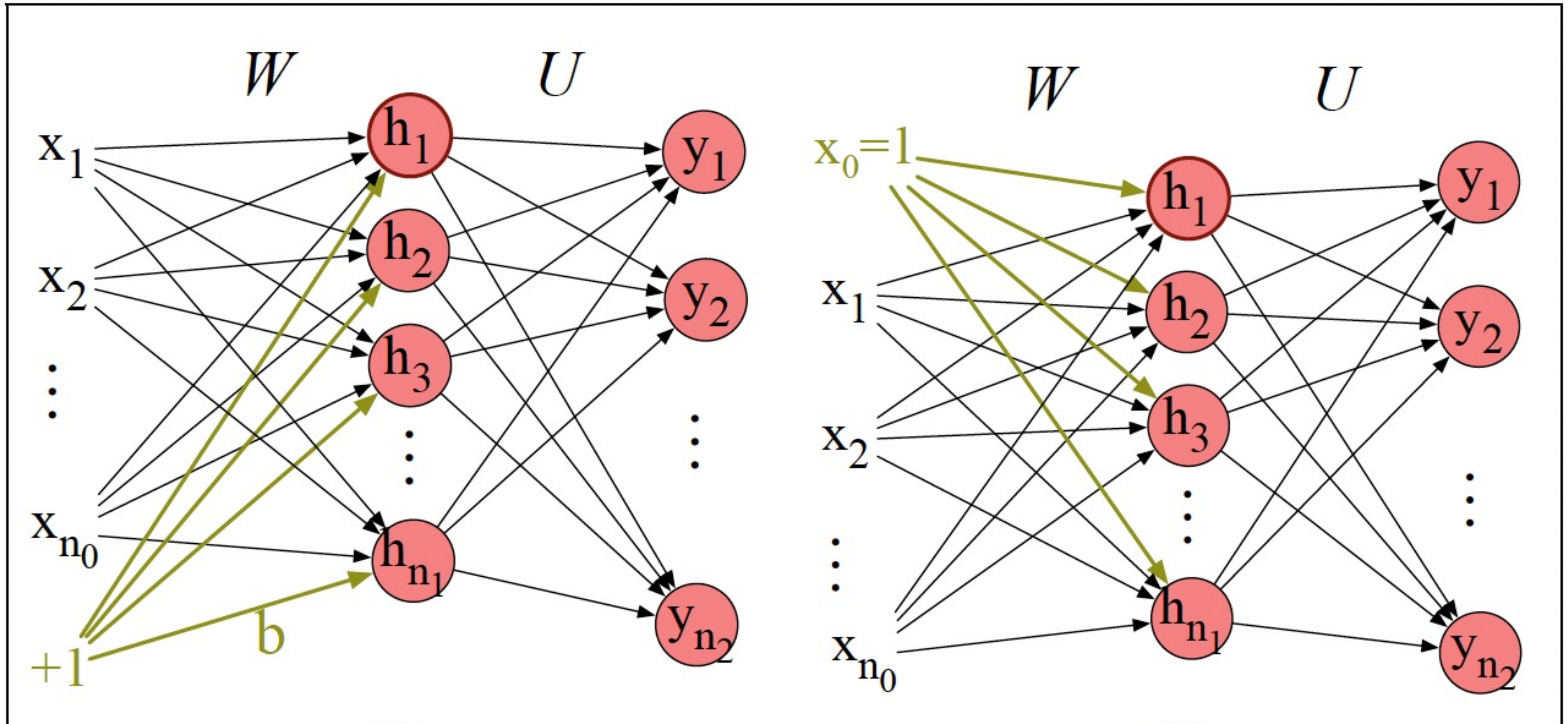
$$a2 = \sigma(z2)$$
$$z2 = W2a1 + b2$$

$$a1 = \mathrm{ReLu}(z1)$$
$$z1 = W1x + b1$$



# Replacing the bias unit

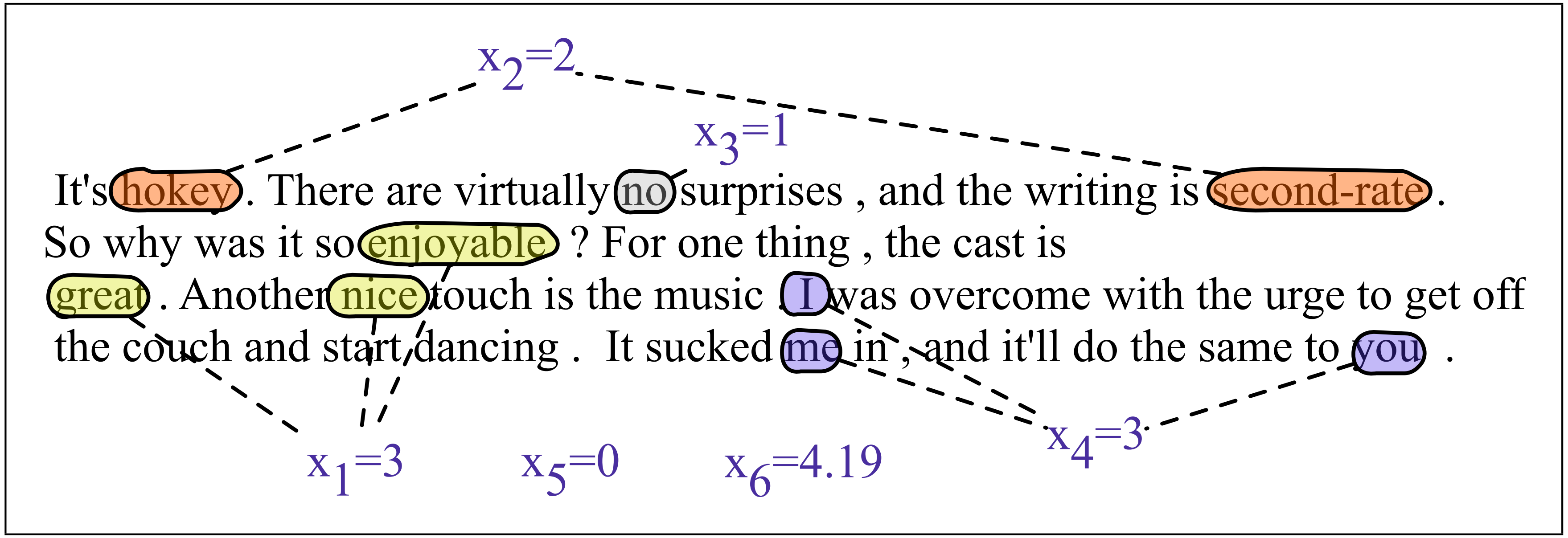
Let's switch to a notation without the bias unit *b*
Add a dummy node $a_0=1$ to each layer, its weight $w_0$ will be the bias
So input layer $a^{[0]}_0=1$, $a^{[1]}_0=1$ , $a^{[2]}_0=1$,…



(a)
(b)

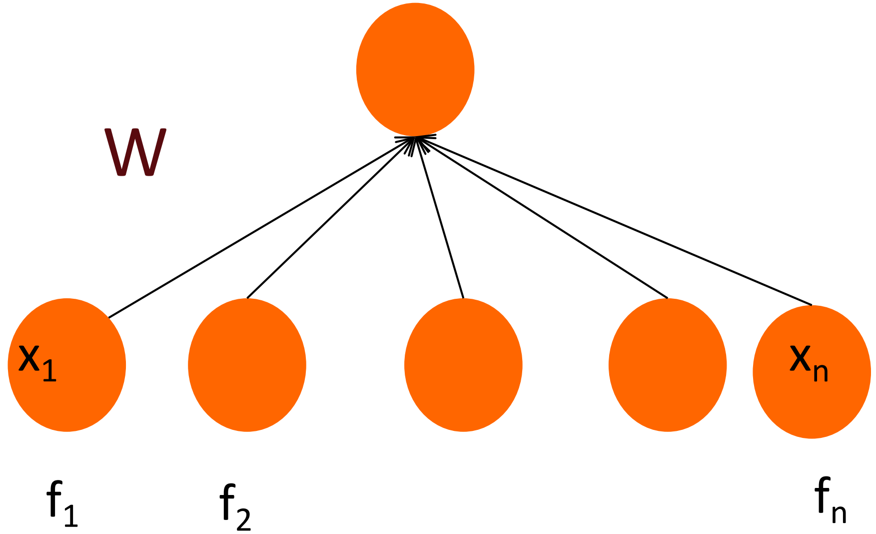
# Sentiment analysis

## Using hand-built features




| | | |
|---|---|---|
| $x_1$ | count(positive lexicon) $\in$ doc) | 3 |
| $x_2$ | count(negative lexicon) $\in$ doc) | 2 |
| $x_3$ | $\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$ | 1 |
| $x_4$ | count(1st and 2nd pronouns $\in$ doc) | 3 |
| $x_5$ | $\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$ | 0 |
| $x_6$ | log(word count of doc) | $\ln(66) = 4.19$ |

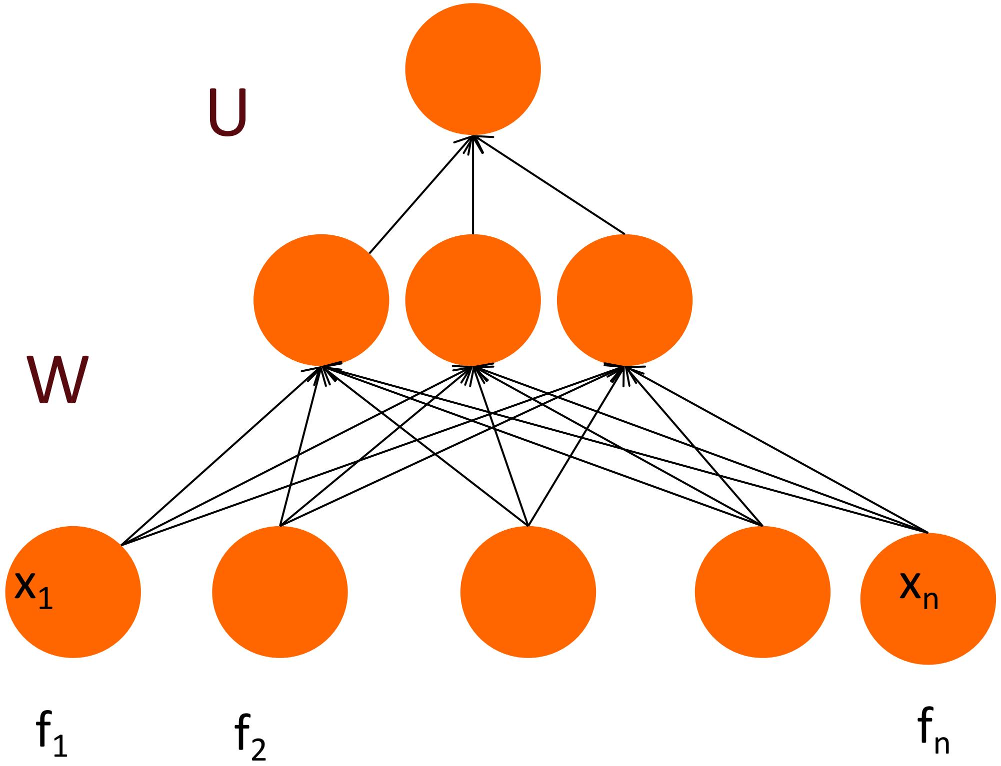
# Logistic regression




| | | |
|---|---|---|
| $x_1$ | count(positive lexicon) $\in$ doc) | 3 |
| $x_2$ | count(negative lexicon) $\in$ doc) | 2 |
| $x_3$ | $\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$ | 1 |
| $x_4$ | count(1st and 2nd pronouns $\in$ doc) | 3 |
| $x_5$ | $\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$ | 0 |
| $x_6$ | log(word count of doc) | $\ln(66) = 4.19$ |

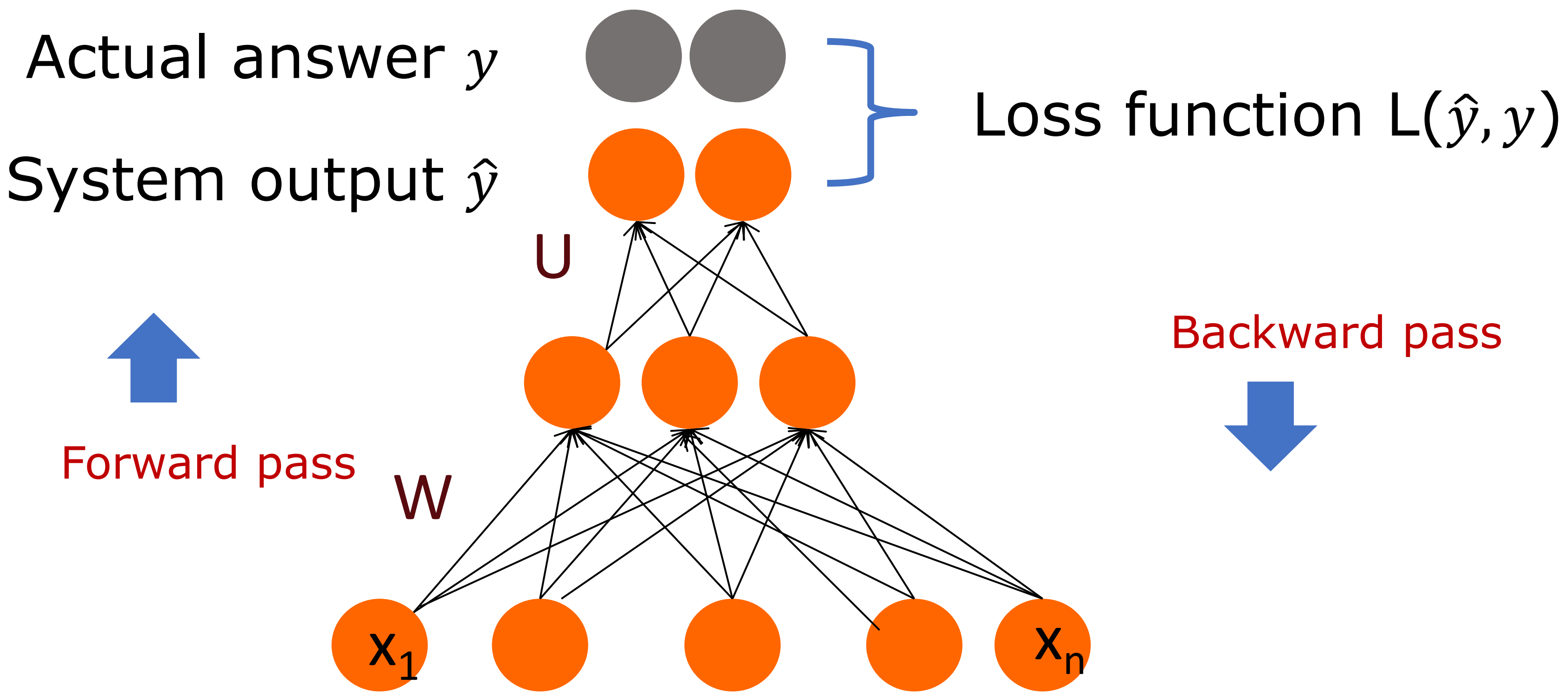
$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\
&= \sigma(.833) \\
&= 0.70
\end{aligned}
$$

# Feedforward neural network

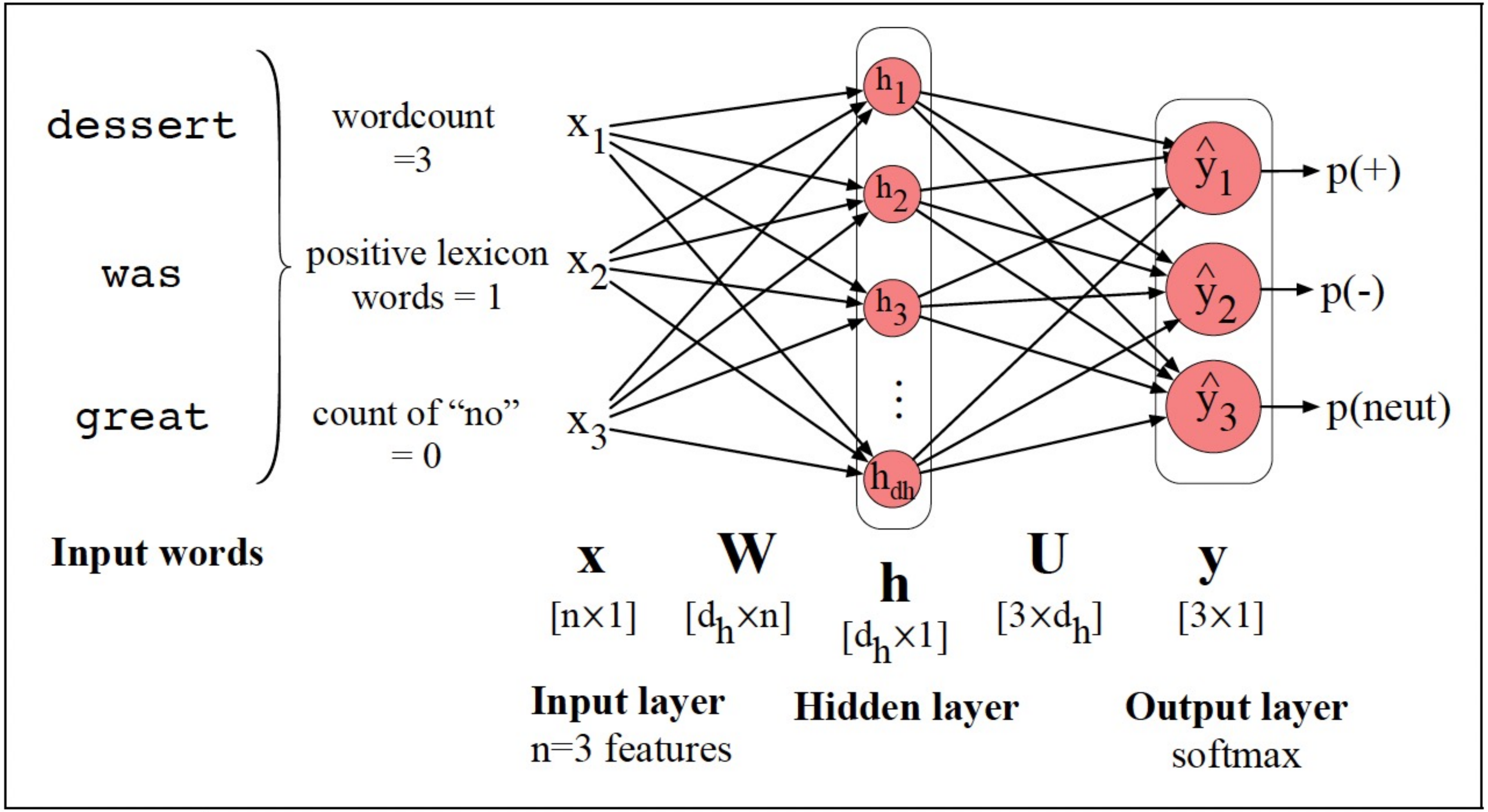



| | | |
|---|---|---|
| $x_1$ | count(positive lexicon) $\in$ doc) | 3 |
| $x_2$ | count(negative lexicon) $\in$ doc) | 2 |
| $x_3$ | $\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$ | 1 |
| $x_4$ | count(1st and 2nd pronouns $\in$ doc) | 3 |
| $x_5$ | $\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$ | 0 |
| $x_6$ | log(word count of doc) | $\ln(66) = 4.19$ |

# Training a neural network model



# FFNN with hand-built features

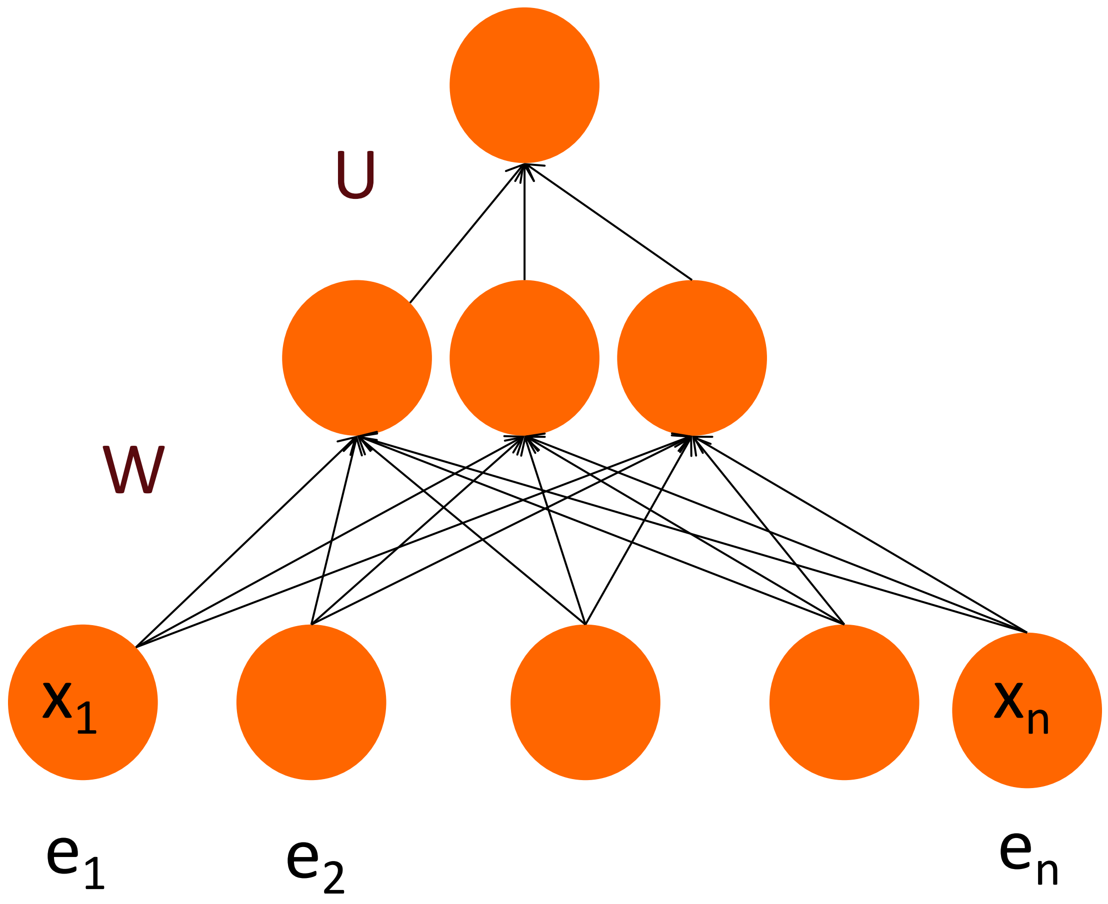



$$
\begin{aligned}
\mathbf{x} &= [\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_N] \\
\mathbf{h} &= \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \\
\mathbf{z} &= \mathbf{U}\mathbf{h} \\
\hat{\mathbf{y}} &= \text{softmax}(\mathbf{z})
\end{aligned}
$$

# Even better: Representation learning

The real power of deep learning comes from the ability to **learn features** from the data

Instead of using hand-built human-engineered features for classification, use learned representations like embeddings!



# Word embeddings

Two words are similar in meaning if their contexts are similar

| | | |
|---|---|---|
| is traditionally followed by | **cherry** | pie, a traditional dessert |
| often mixed, such as | **strawberry** | rhubarb pie. Apple pie |
| computer peripherals and personal | **digital** | assistants. These devices usually |
| a computer. This includes | **information** | available on the internet |

| | **aardvark** | ... | **computer** | **data** | **result** | **pie** | **sugar** | ... |
|---|---|---|---|---|---|---|---|---|
| **cherry** | 0 | ... | 2 | 8 | 9 | 442 | 25 | ... |
| **strawberry** | 0 | ... | 0 | 0 | 1 | 60 | 19 | ... |
| **digital** | 0 | ... | 1670 | 1683 | 85 | 5 | 4 | ... |
| **information** | 0 | ... | 3325 | 3982 | 378 | 5 | 13 | ... |

# Cosine similarity

$$\cos\left(\frac{\vec{v}\cdot\vec{w}}{|\vec{v}||\vec{w}|}\right)=\frac{\vec{v}}{|\vec{v}|}\cdot\frac{\vec{w}}{|\vec{w}|}=\frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$

| | pie | data | computer |
|---|---|---|---|
| **cherry** | 6.76 | 2.42 | 1.22 |
| **digital** | 1.65 | 6.85 | 6.83 |
| **information** | 1.44 | 6.62 | 6.48 |

$$\cos(cherry, information)$$
$$=\frac{6.76*1.44+2.42*6.62+1.22*6.48}{\sqrt{6.76^2+2.42^2+1.22^2}\sqrt{1.44^2+6.62^2+6.48^2}}=0.49$$

$$\cos(digital, information)$$
$$=\frac{1.65*1.44+6.85*6.62+6.83*6.48}{\sqrt{1.65^2+6.85^2+6.83^2}\sqrt{1.44^2+6.62^2+6.48^2}}=0.99$$

semantically-related words have higher cosine similarity

# Word2Vec: Skip-gram algorithm

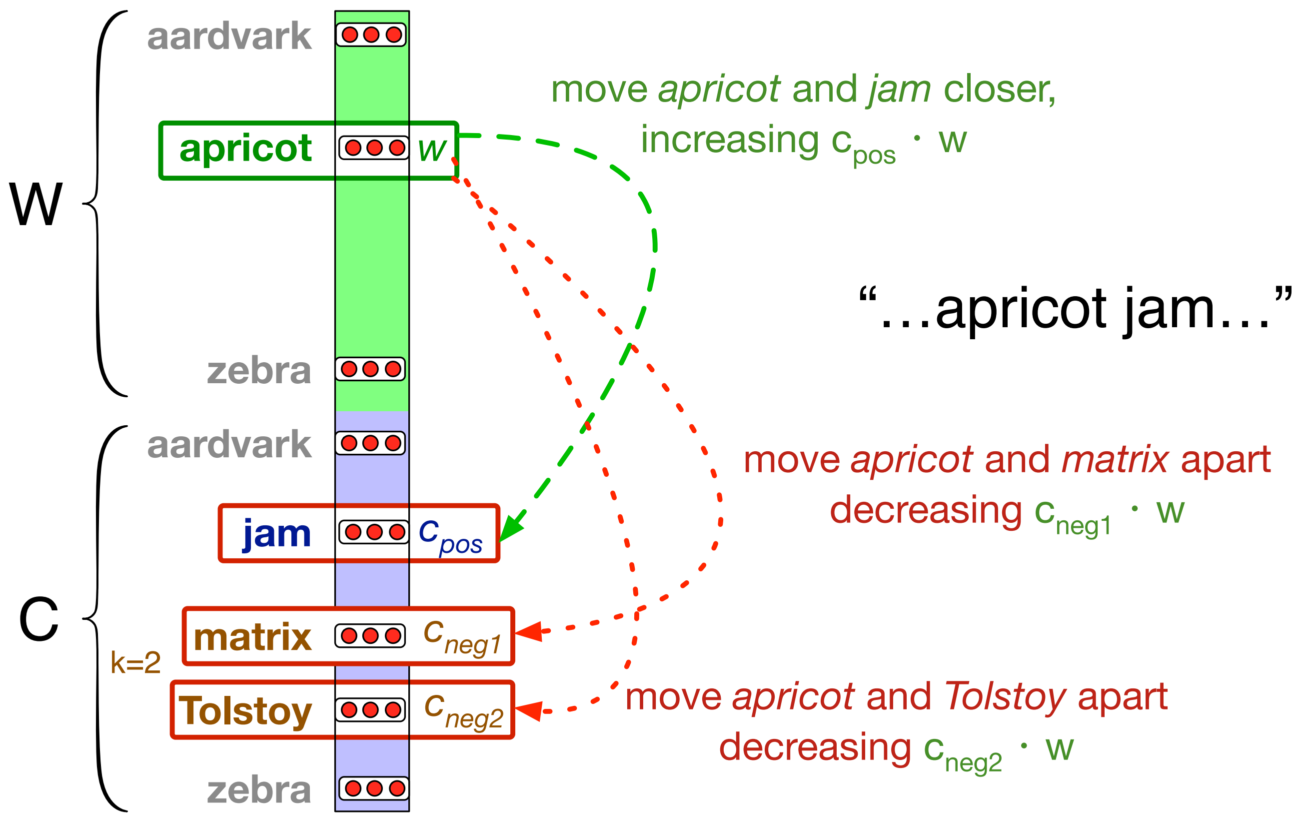
How to learn?

**Gradient descent!**

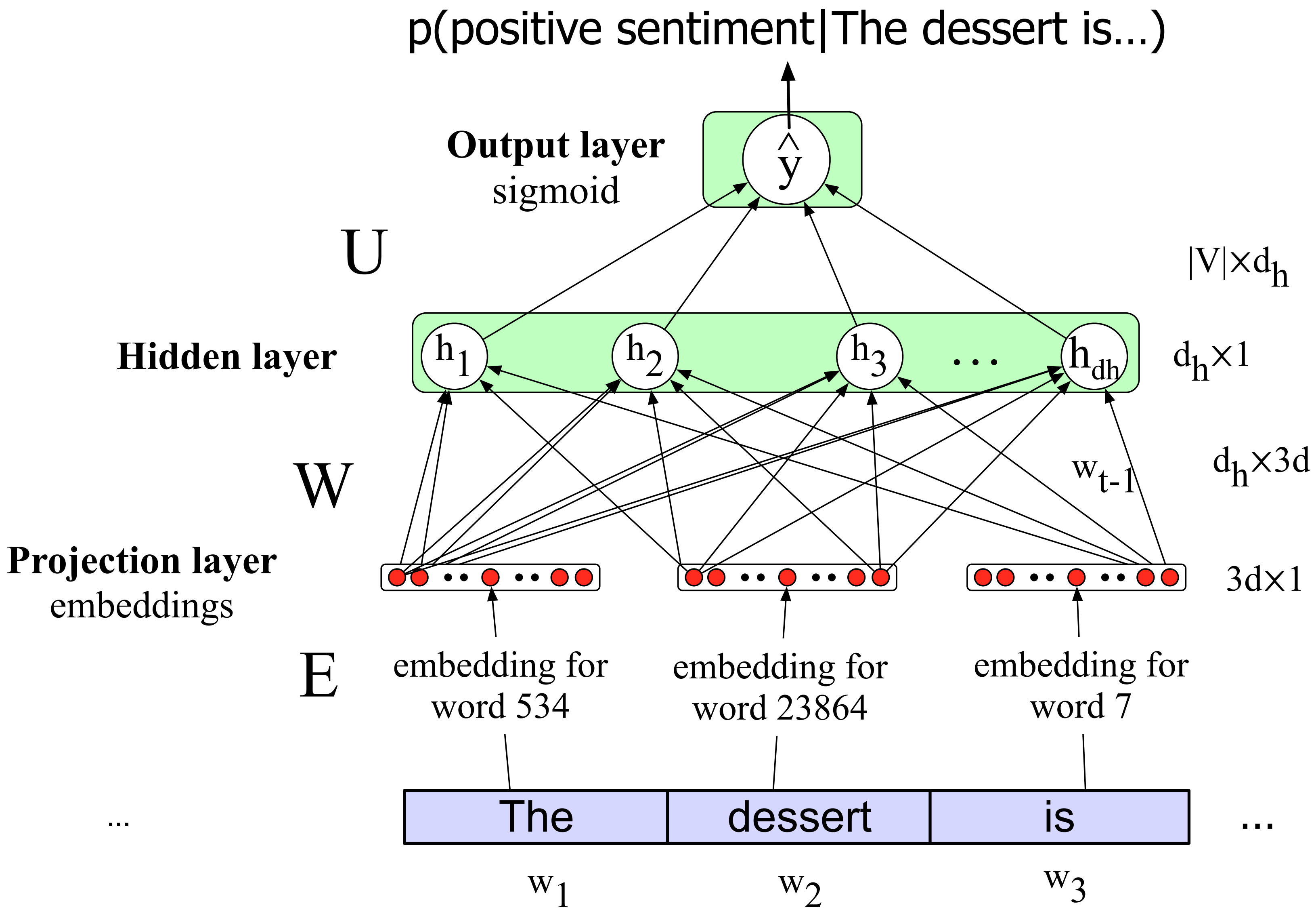
We'll adjust the word weights to

- make the positive pairs more likely
- and the negative pairs less likely,
- over the entire training set.

$\boldsymbol{\theta}$



# FFNN with word embeddings

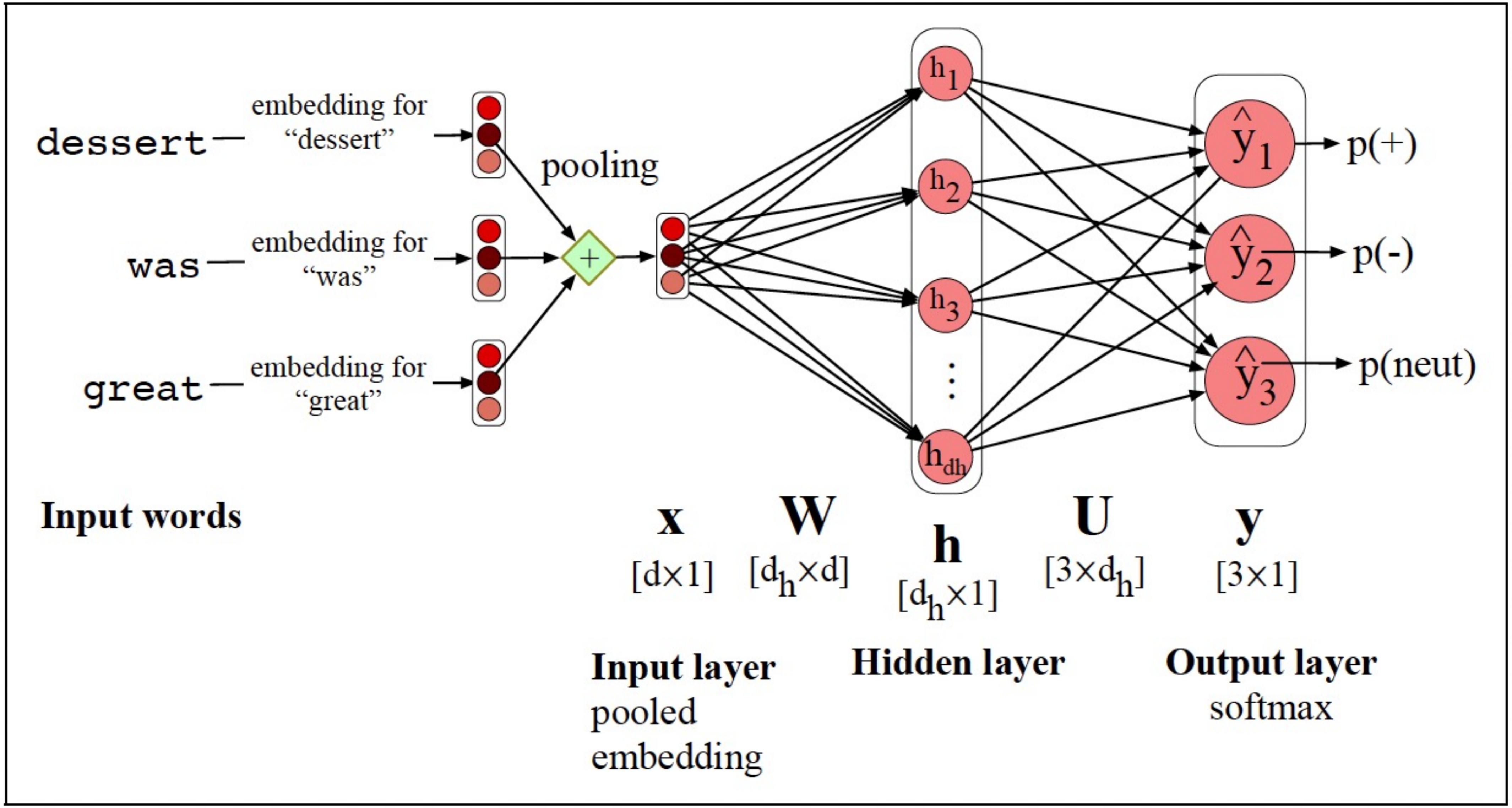


# Issue: Texts come in different sizes

- This assumes a fixed size length (3)! → unrealistic

- One simple solution (more sophisticated solutions later):
  Create a single "sentence embedding" (the same dimensionality as a word) to represent all the words
  - Averaging pooling:
    Take the mean of all the word embeddings
  - Max pooling:
    Take the element-wise max of all the word embeddings
    →For each dimension, pick the max value from all words

# Averaged word embedding



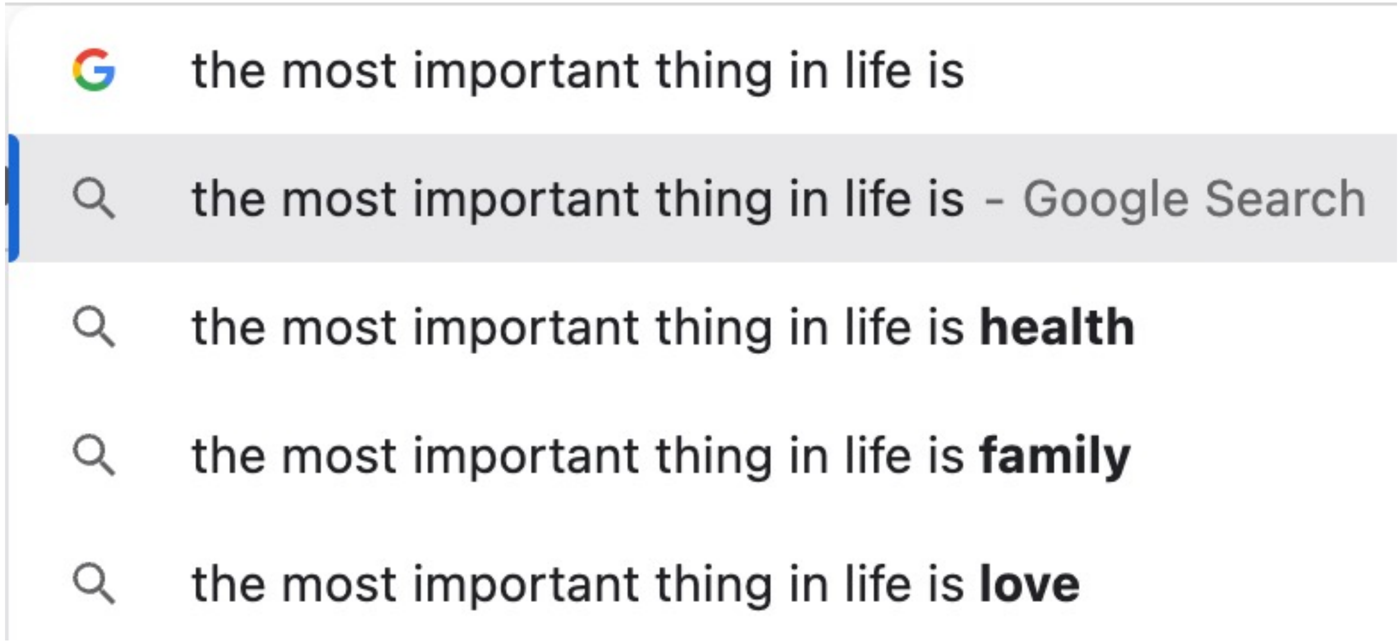
# Language model

**Goal:** Compute the probability of a sentence or sequence of words

**P(***a pile of shaving cream***)** **or** **P(***cream*|*a pile of shaving***)**

$\mathbf{P}(W) = \mathbf{P}(w_1,w_2,w_3,\ldots w_n)$ $\mathbf{P}(w_n|w_1,w_2,w_3,\ldots w_{n-1})$

**→ Language Model (LM)**



# N-gram language model

Bigram model using the **Maximum Likelihood Estimate (MLE)**

$$\mathbf{P}(w_i|w_{i-1}) = \frac{\mathbf{Count}(w_{i-1}, w_i)}{\mathbf{Count}(w_{i-1})}$$

*<s> I am Sam </s>*
*<s> Sam I am </s>*
*<s> I do not like green eggs and ham </s>*

$\mathbf{P}(I|\text{<s>})=\frac{2}{3}=0.67$ $\mathbf{P}(am|I)=\frac{2}{3}=0.67$ $\mathbf{P}(Sam|am)=\frac{1}{2}=0.5$

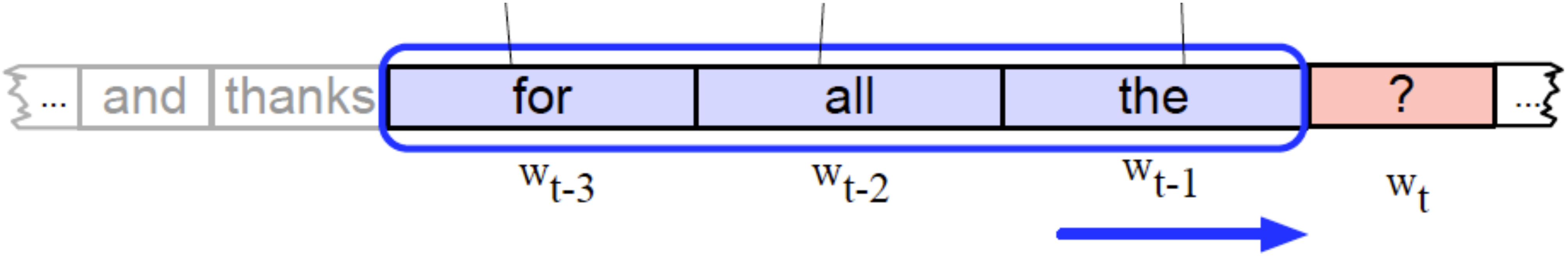
$\mathbf{P}(\text{</s>}|Sam)=\frac{1}{2}=0.5$ $\mathbf{P}(Sam|\text{<s>})=\frac{1}{3}=0.33$ $\mathbf{P}(do|I)=\frac{1}{3}=0.33$

# Simple FFNN language model

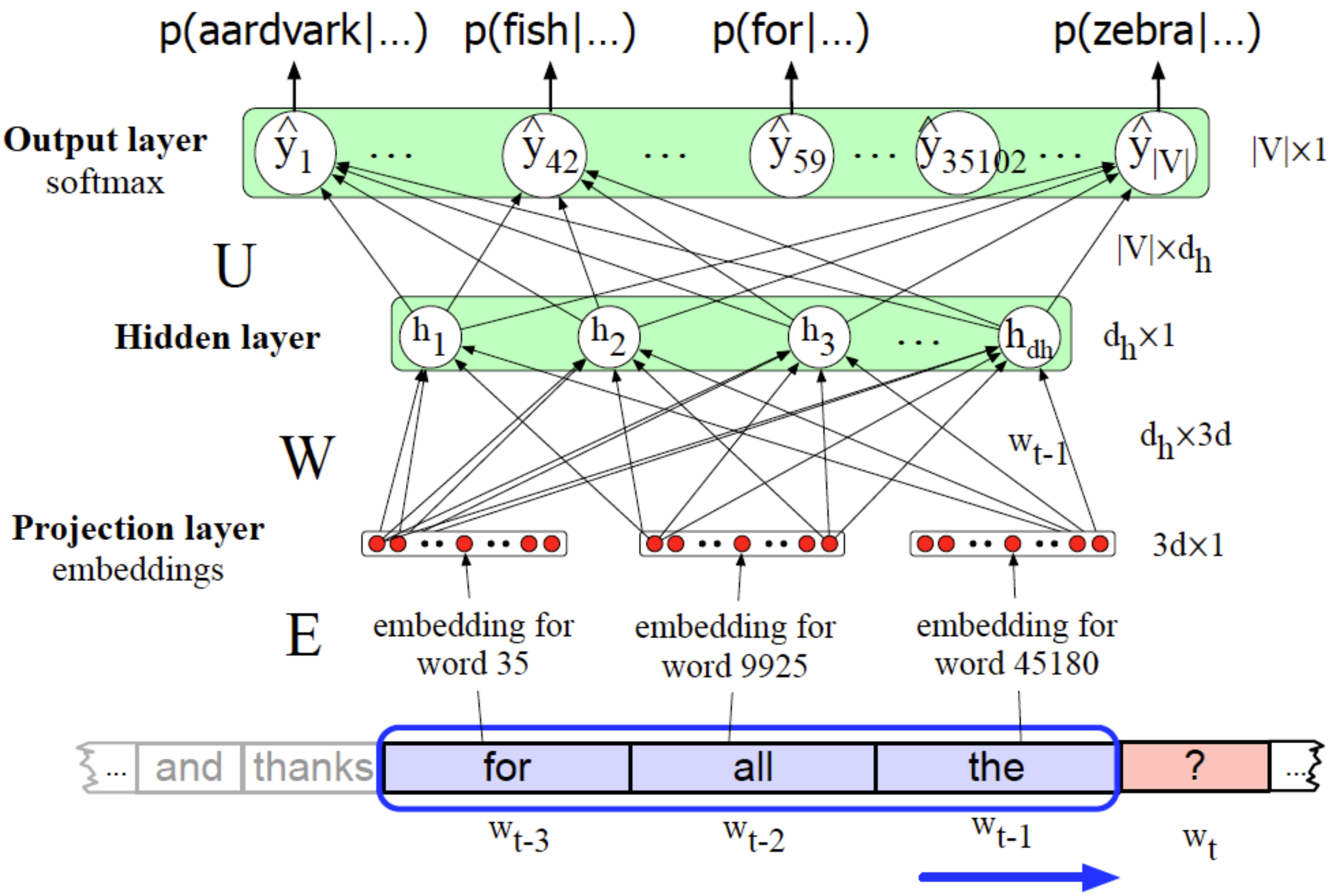
**Task**: Predict next word $w_t$ given prior words $w_{t-1}$, $w_{t-2}$, $w_{t-3}$, …

**Problem**: Sequences of arbitrary length

**Solution**: Sliding windows (of fixed length)



# Model architecture



# Why Neural LMs work better than N-gram LMs

**Training data:**

We've seen: I have to make sure that the cat gets fed.

Never seen: dog gets fed

**Test data:**

I forgot to make sure that the dog gets ___

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

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

# To do

- Optional reading: **SLP** Ch7
- Review Colab