Neural networks (Just some basics…)

IN4080

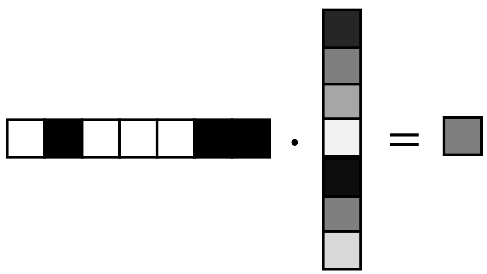
Natural Language Processing

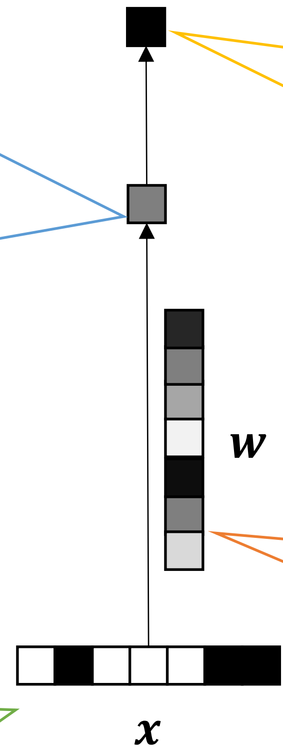
Yves Scherrer

Linear models for text classification

Example: Determine if a text has positive sentiment.

The prediction score corresponds to the dot product: $z = \boldsymbol{x} \cdot \boldsymbol{w}$





Logistic regression models add a sigmoid function:

$$P(y = 1|x) = \frac{1}{1 + \exp(-z)}$$

The features and weights can be represented as vectors $\boldsymbol{x}$ and $\boldsymbol{w}$.

The features are typically defined manually (e.g. bag of words).

Example

- $\boldsymbol{w} = [0.2, 0.3, 0.9, 0.5]$
- $\boldsymbol{x} = [0.5, 0.6, 0.1, 1]$

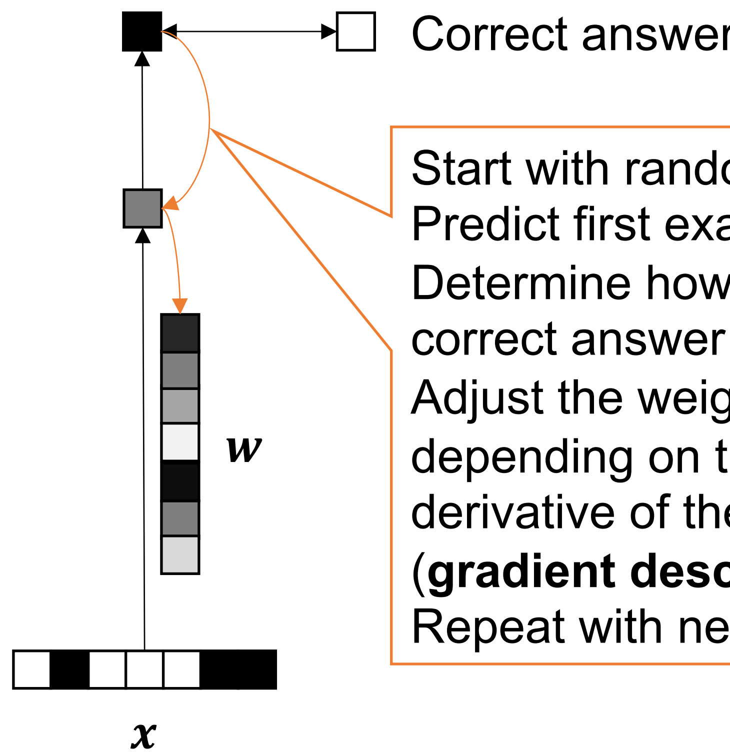
The last (or first) element of the feature vector is typically always set to 1. This is called the **bias term**.

- $\boldsymbol{w} \cdot \boldsymbol{x} = 0.1 + 0.18 + 0.09 + 0.5 = 0.87$

- $y = \sigma(\boldsymbol{w} \cdot \boldsymbol{x}) = \frac{1}{1+e^{-(\boldsymbol{w} \cdot \boldsymbol{x})}} = \frac{1}{1+e^{-0.87}} = 0.70$

Linear models
Training

Example: Determine if a text has positive sentiment.

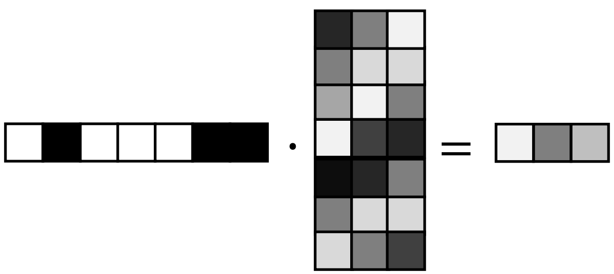


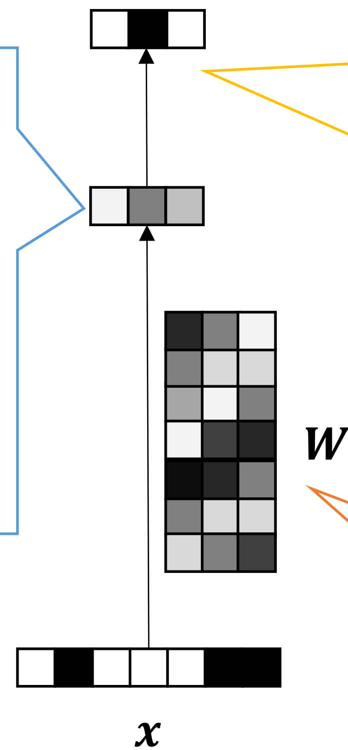
Start with random weights.
Predict first example.
Determine how far off the correct answer is (**loss**).
Adjust the weights depending on the derivative of the loss (**gradient descent**).
Repeat with next example.

Linear models
Multi-class prediction

Example: Determine the dominant sentiment of a text.

Compute the matrix-vector product: $\boldsymbol{z} = \boldsymbol{x} \cdot \boldsymbol{W}$





Probability distribution over three classes (e.g. positive, negative, neutral) with **softmax**:

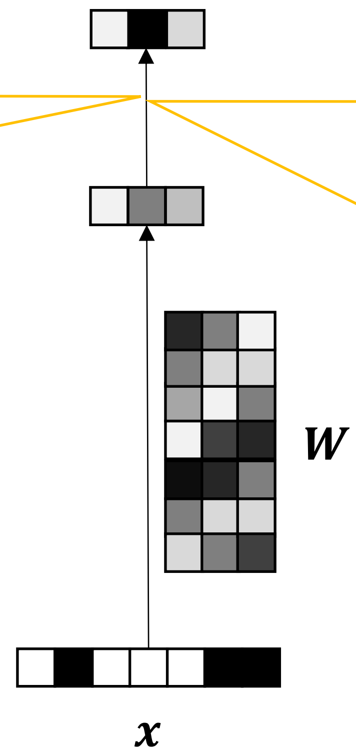
$$\frac{\exp(\boldsymbol{w}_y \cdot \boldsymbol{x})}{\sum_{y' \in Y} \exp(\boldsymbol{w}_{y'} \cdot \boldsymbol{x})}$$

Every class gets its own weight vector, resulting in a matrix.

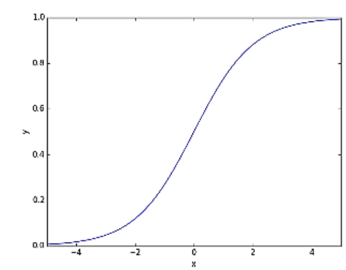
Activation functions

Example: Determine the dominant sentiment of a text.

This part is also called **activation function**.



Several types of S-shaped (“sigmoid”) functions can be used here.



Two-step classification

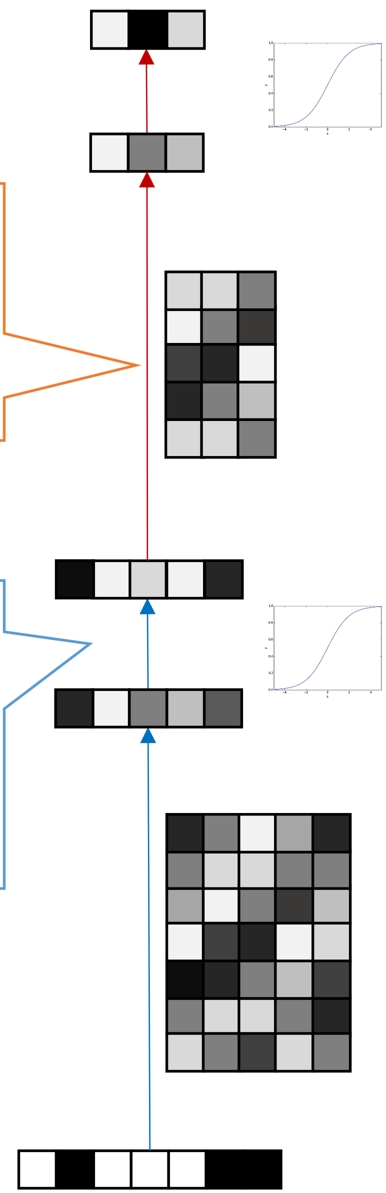
Idea:

- Partition the collection into e.g. 100 text classes
- Base the sentiment decision on the text class

Let's use the output of the first model as the input features of the second model.

Both weight matrices can be trained in a single pass: **backpropagation**.

This is called a **hidden layer**. Models with hidden layers are called **neural networks**.



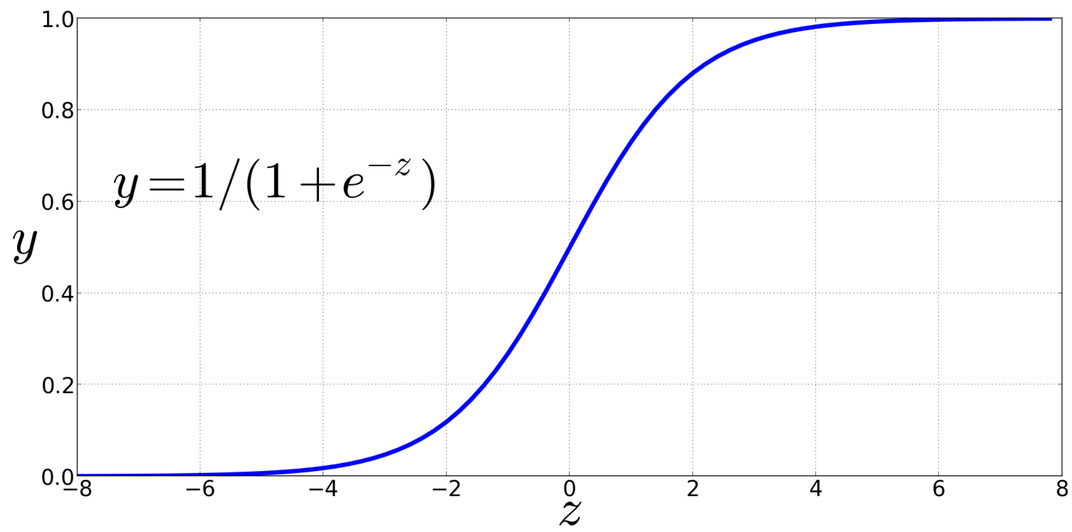
Activation functions

The **softmax** function is costly due to the normalization.

- In the hidden layers, we don't care about proper probability distributions and can use simpler activation functions (i.e. element-wise functions).

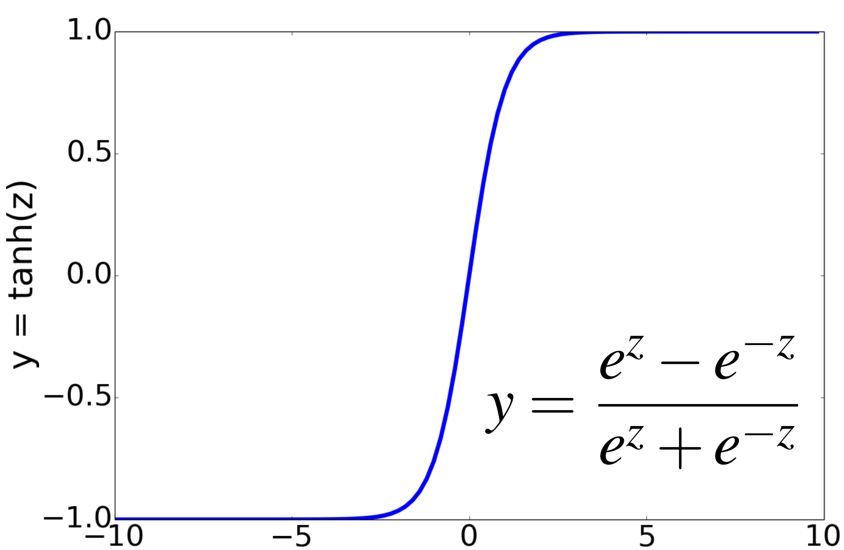
Sigmoid:

$y = \sigma(z)$



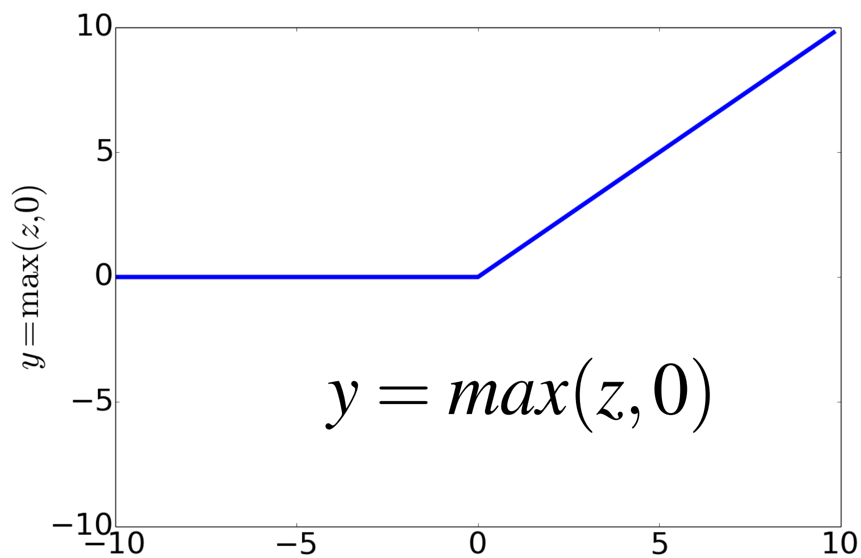
Activation functions

tanh



Similar to sigmoid, but often works better.

ReLu (Rectified Linear Unit)

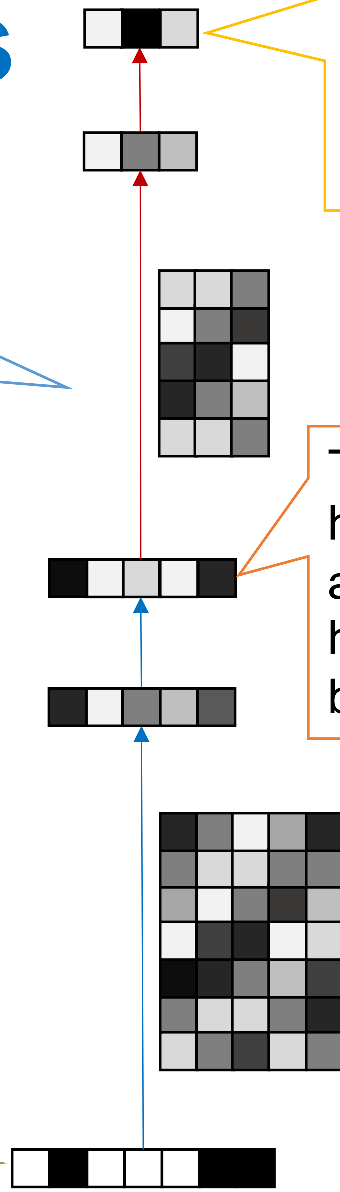


Very simple and often used.

Neural networks

The size of the prediction vector is defined by the number of classes.

Several hidden layers can be added. Neural networks with more than one hidden layer are called **deep models**.

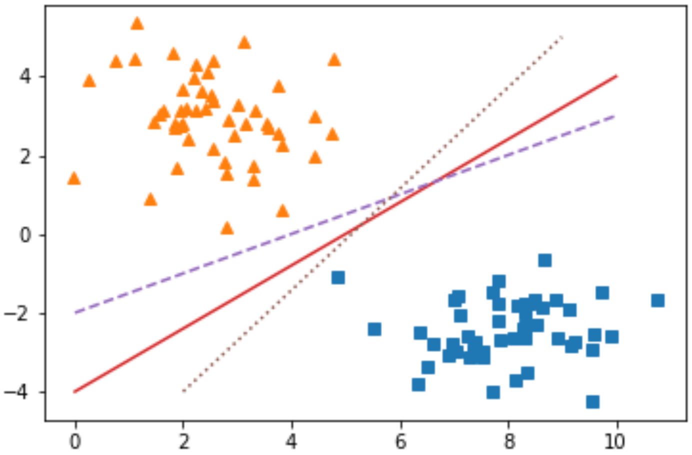


The size of the hidden layer vectors and the number of hidden layers can be freely chosen.

The size of the input vector is defined by the number of distinct words in the training corpus.

Why hidden layers?

- Perceptron and LR are linear classifiers
  - The decision boundary (for a binary prediction problem) is one straight line

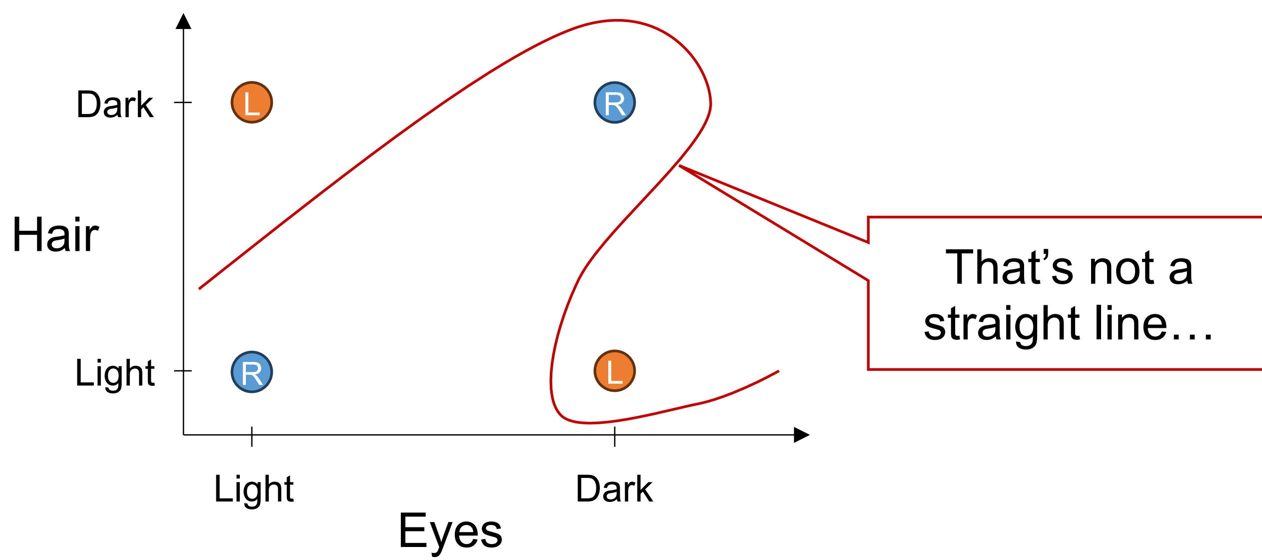


- There are relatively simple problems that cannot be solved with a linear classifier
  - For example, the XOR problem

The XOR problem

Let's imagine a world where:

- People have either light or dark hair
- People have either light or dark eyes
- People are right-handed whenever their hair and eye color matches, and left-handed when it doesn't match



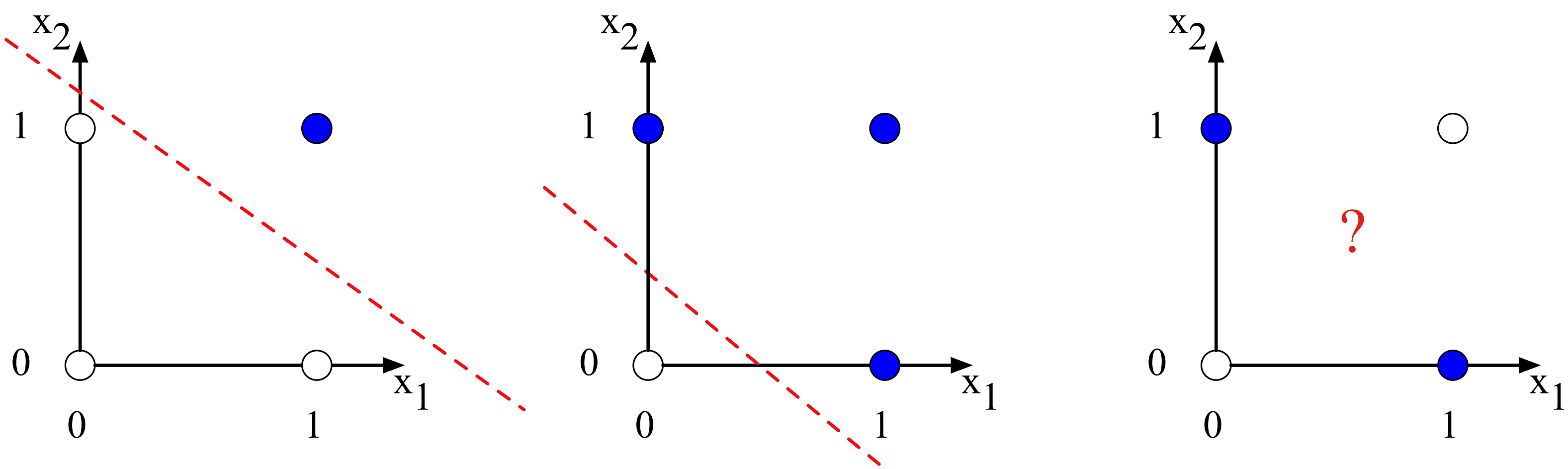
The XOR problem

0 = light, 1 = dark
x_1 = eyes, x_2 = hair

$$x = [0,0] \Longrightarrow y = 0$$
$$x = [0,1] \Longrightarrow y = 1$$
$$x = [1,0] \Longrightarrow y = 1$$
$$x = [1,1] \Longrightarrow y = 0$$

0 = right-handed
1 = left-handed

It is impossible to define a weight vector $\boldsymbol{w}$ that produces the correct predictions.



a) x_1 AND x_2 b) x_1 OR x_2 c) x_1 XOR x_2

Extended features

We can extend the feature vector with combinations of existing features:

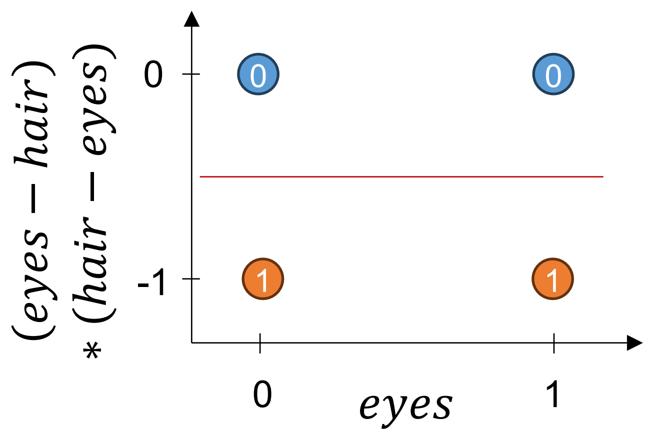
$$[x_1, x_2] \implies [x_1, x_2, x_1 * x_2, x_1^2 + x_2^2, \dots]$$

- Datasets which are not linearly separable may become separable when using extended features.

Let's try:

$$x = [eyes, hair, (eyes - hair) * (hair - eyes)]$$

- $x = [0, 0, 0]$ $\quad y = 0$
- $x = [0, 1, -1]$ $\quad y = 1$
- $x = [1, 0, -1]$ $\quad y = 1$
- $x = [1, 1, 0]$ $\quad y = 0$



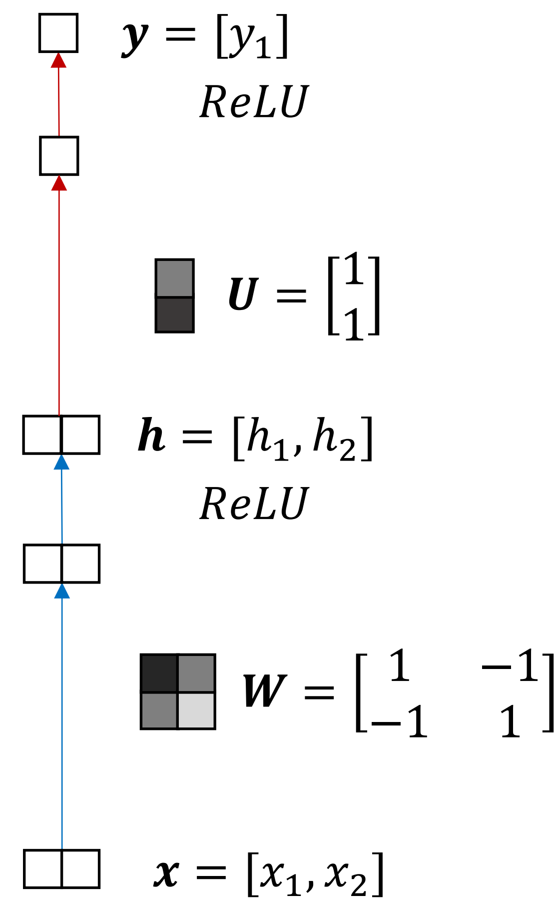
Extended features

$$x = [eyes, hair, (eyes - hair) * (hair - eyes)]$$

- How did we figure that out???
  - XOR can be defined as $(A \vee \neg B) \wedge (B \vee \neg A)$
- Features have to be defined manually
- Real-life features generally don't correspond to propositional logic functions…

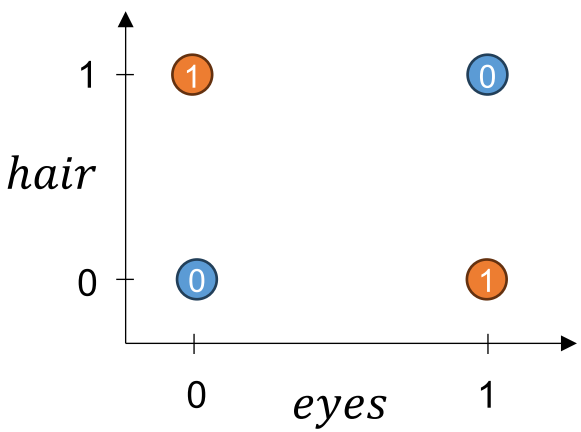
The XOR problem in a neural network

y	0	1	1	0
$h \cdot U$	0	1	1	0
h	$[0\ \ 0]$	$[0\ \ 1]$	$[1\ \ 0]$	$[0\ \ 0]$
$x \cdot W$	$[0\ \ 0]$	$[-1\ 1]$	$[1\ \ -1]$	$[0\ \ 0]$
x	$[0\ \ 0]$	$[0\ \ 1]$	$[1\ \ 0]$	$[1\ \ 1]$

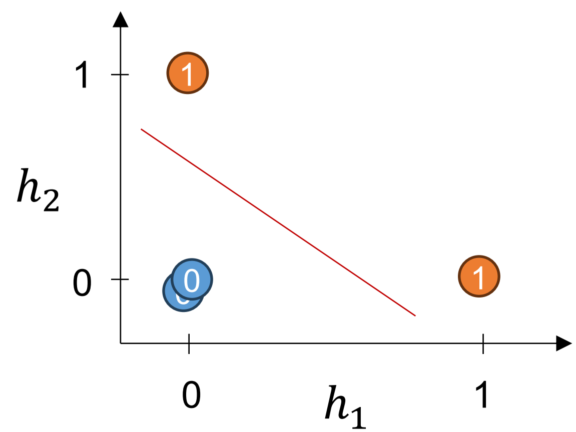


Why does this work?

The original space:



The new hidden space:

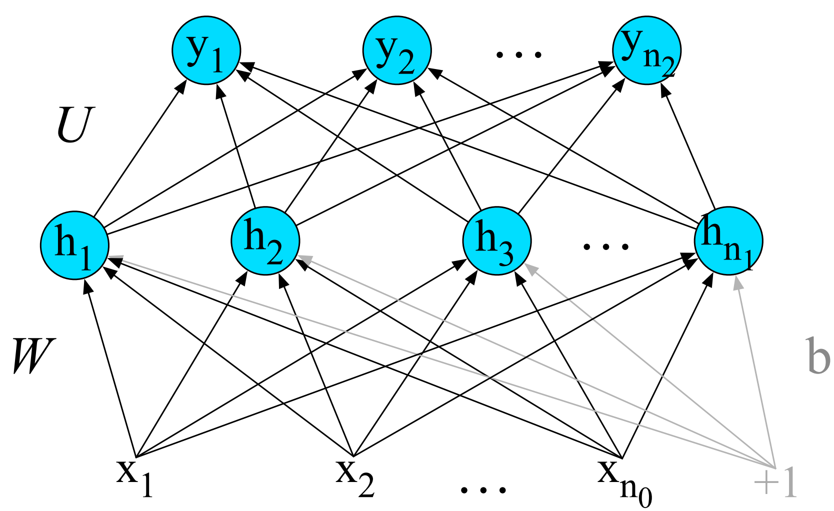


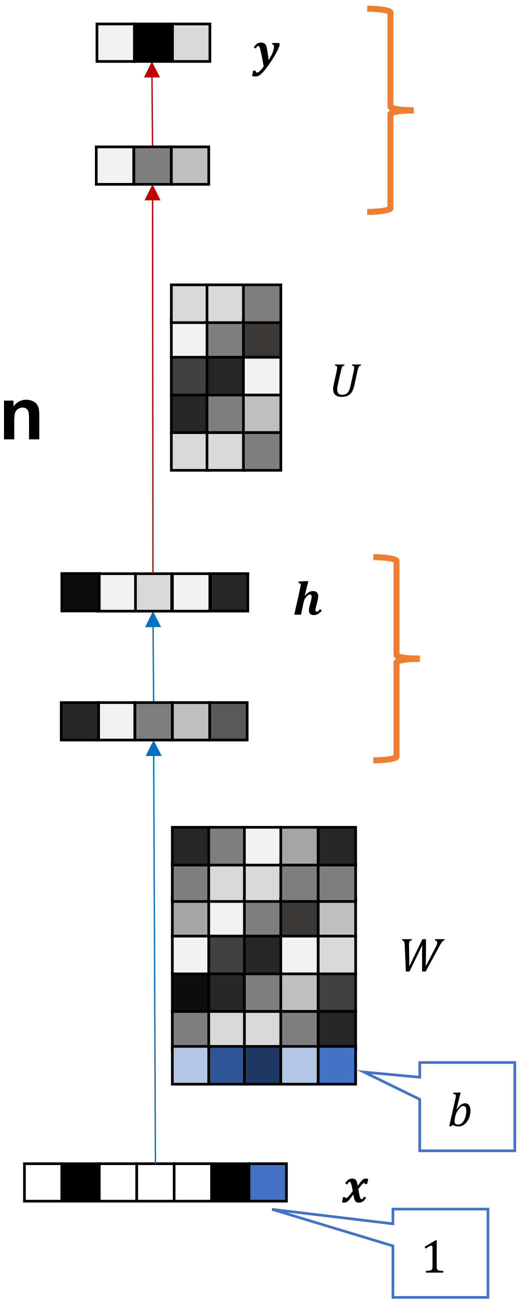
- This only works for a particular set of weight values in W and U. But these can be learned automatically.

Feed-forward neural networks

Feed-forward neural networks

- All arrows go "upwards"
- Also called **multi-layer perceptron** (for historical reasons)
- Input units x_i, hidden units h_i, output units y_i

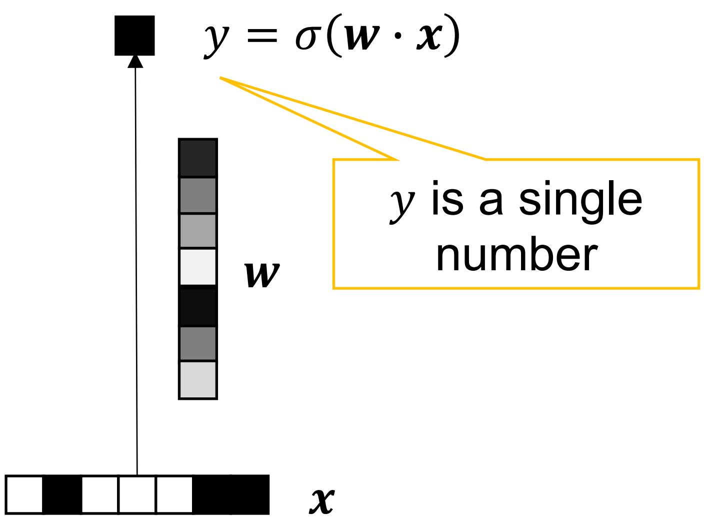




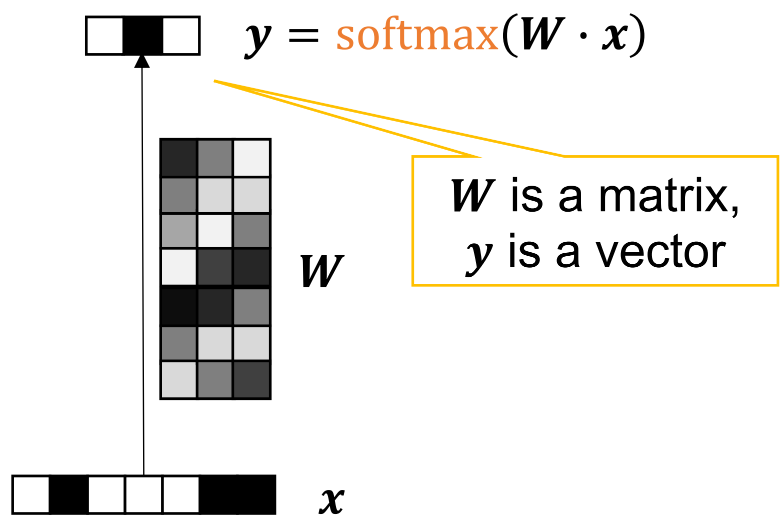
Feed-forward neural networks

FFN with 1 layer = logistic regression

Binary LR:



Multinomial LR:

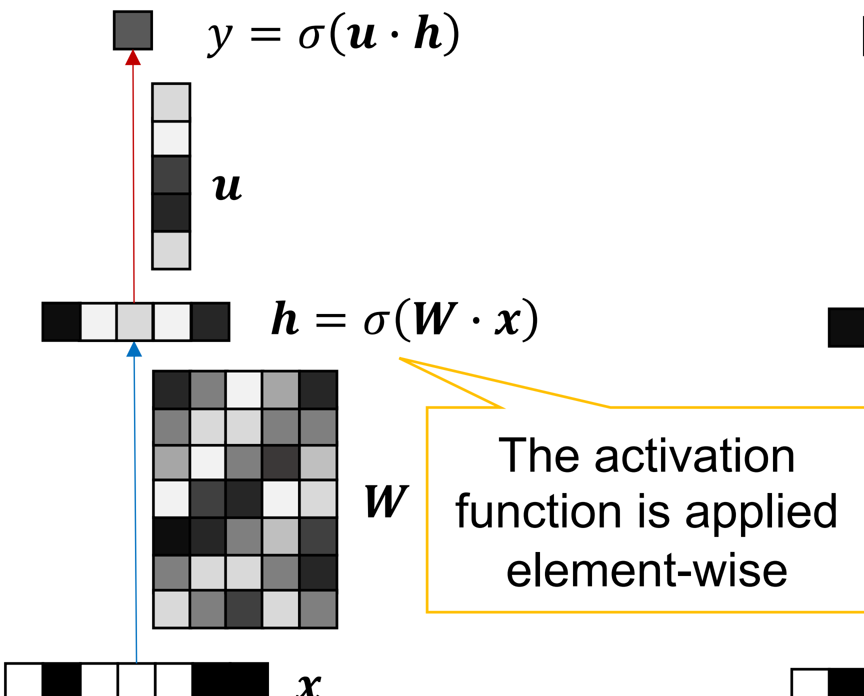


Feed-forward neural networks

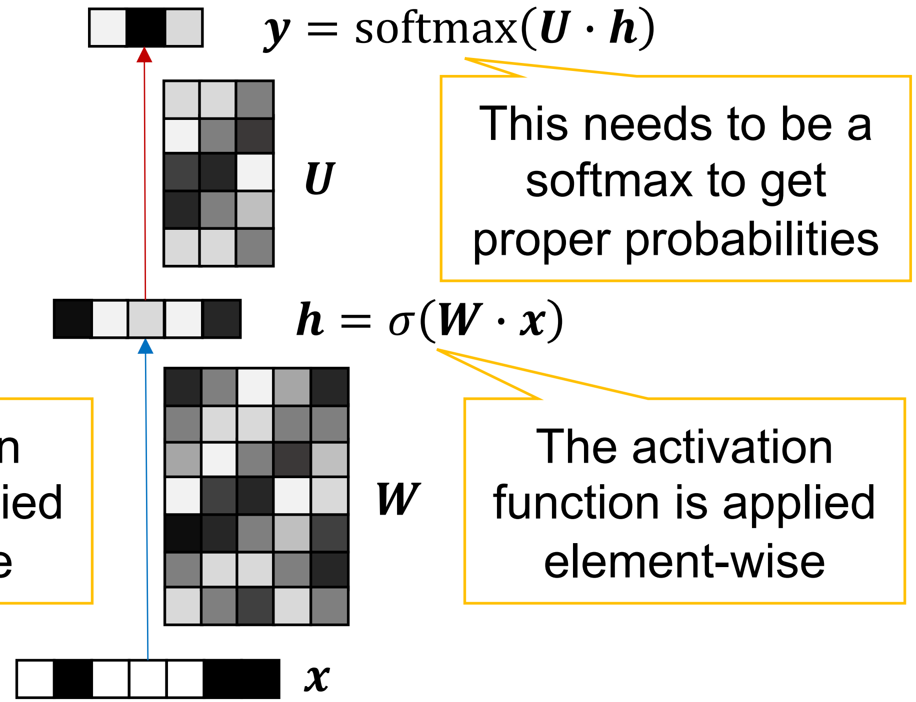
Two-layer networks

More layers can be added analogously

Binary:



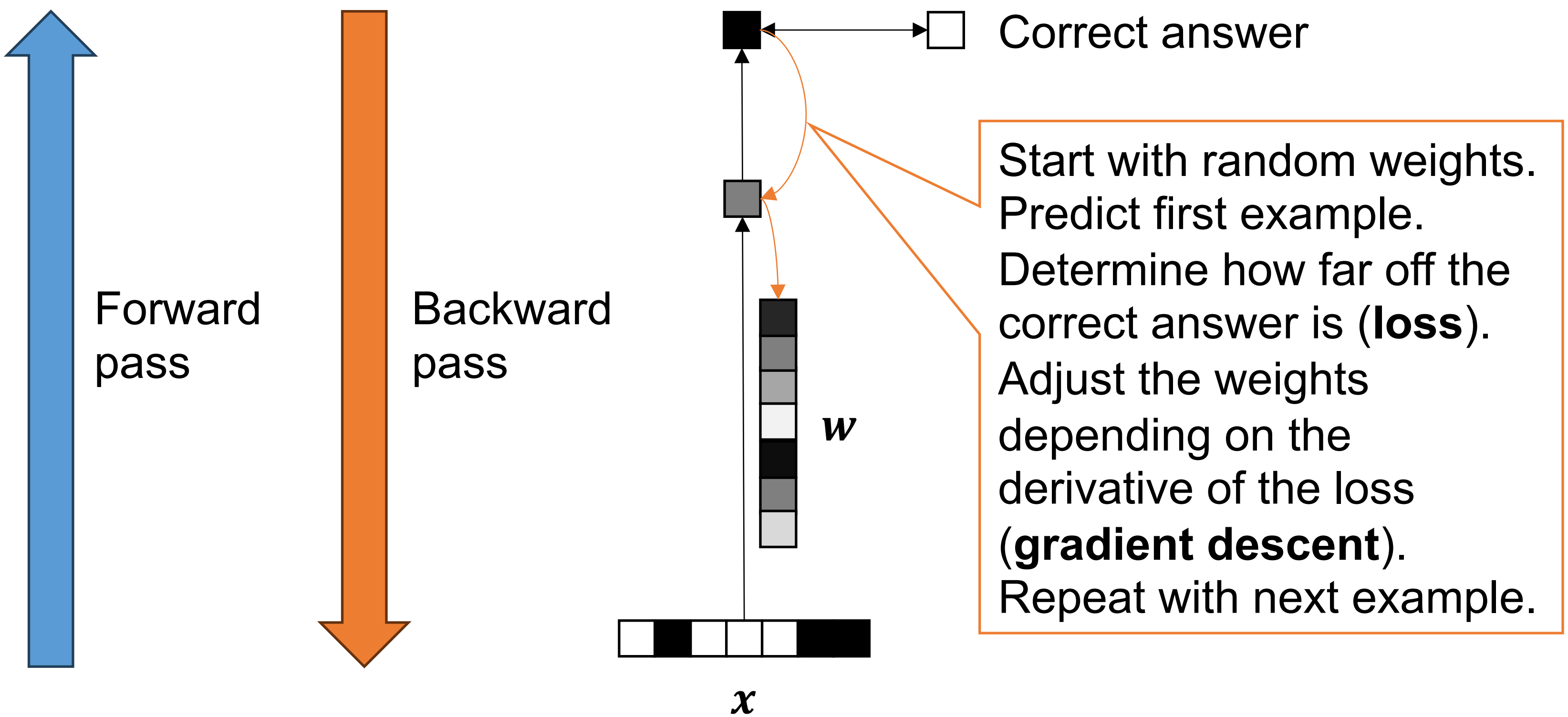
Multinomial:



Training neural networks

Linear models – Training

Example: Determine if a text has positive sentiment.



Training a 2-layer network

- 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 u 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 for LR

For binary logistic regression, we typically use the **cross-entropy loss**:

Goal: maximize probability of the correct label

Binary classification: only two options $\hat{y}$ and $1-\hat{y}$

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

$\hat{y} = \sigma(w \cdot x)$

Gradient descent for LR

Use the **derivative** of the loss function with respect to weights

$$\frac{d}{dw} L(\hat{y}, y) = \frac{d}{dw} L(f(x; w), y)$$

to tell us how to adjust weights for each training item:

$$w_{i,j} \leftarrow w_{i,j} - \lambda \frac{d}{dw_{i,j}} L(\hat{y}\,, y)$$

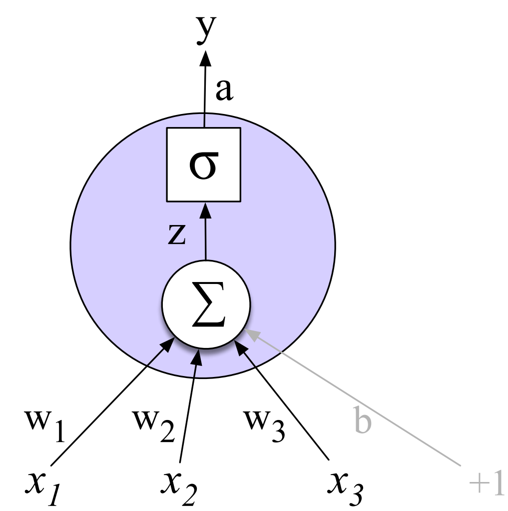
For logistic regression:

$$\frac{d}{dw_j} L_{CE}(\hat{y}, y) = (\sigma(w \cdot x) - y) \cdot x_j$$

Gradient descent

A neural network layer is essentially the same as a logistic regression classifier.

- **Chain rule:** if $f(x) = u(v(x))$ then $\frac{df}{dx} = \frac{du}{dv} \cdot \frac{dv}{dx}$



- Idea:
    1. Compute the derivative of the loss
    2. Compute the derivative of the activation function
    3. Compute the derivative of the dot product

Computation graphs

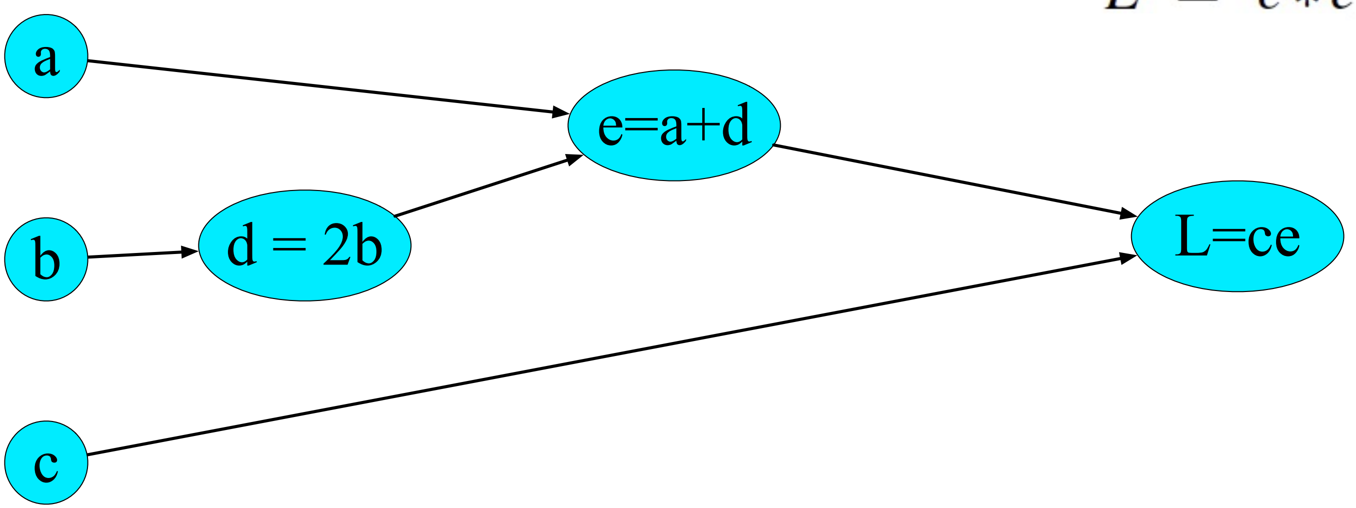
- What if we have a neural network with several layers?
  - We need the derivative of the loss with respect to each weight in **every layer** of the network.
  - But the loss can only be computed at the end of the network.
- Solution: **backpropagation**
  - “Distributes” the loss gradient over all the layers.
  - Relies on **computation graphs**.
  - A computation graph represents the process of computing a mathematical expression.

Example

$$L(a,b,c) = c(a+2b)$$

Computations:

$$\begin{aligned} d &= 2 * b \\ e &= a + d \\ L &= c * e \end{aligned}$$

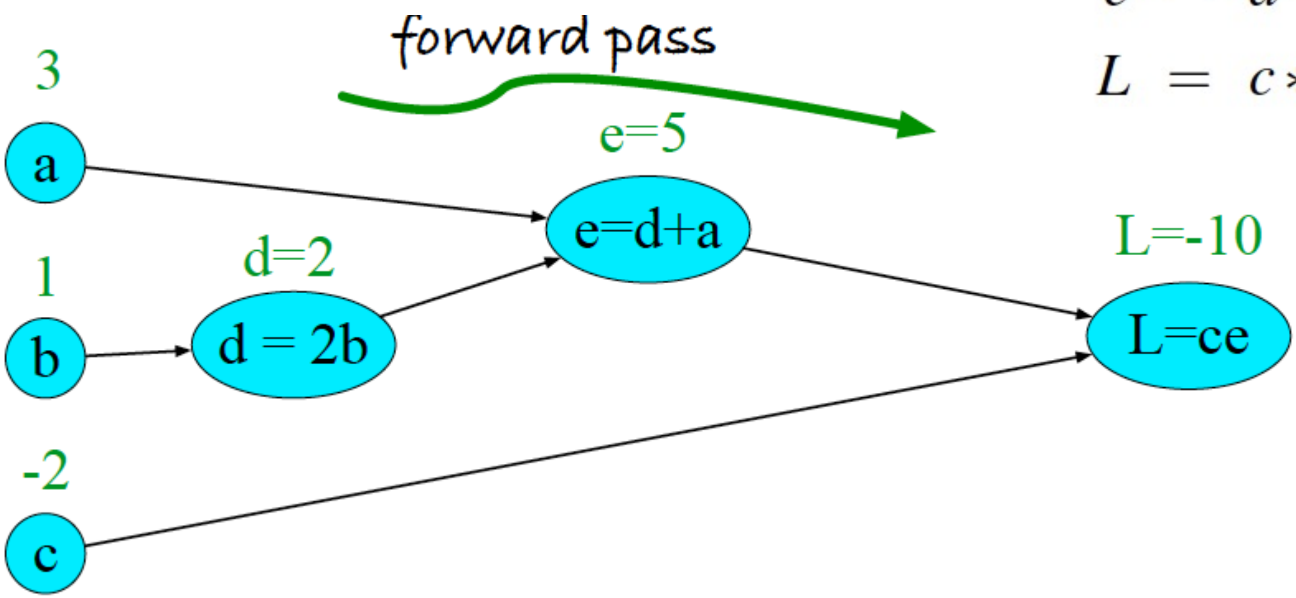


Example

$$L(a,b,c) = c(a+2b)$$

Computations:

$$\begin{aligned} d &= 2*b \\ e &= a+d \\ L &= c*e \end{aligned}$$



Example

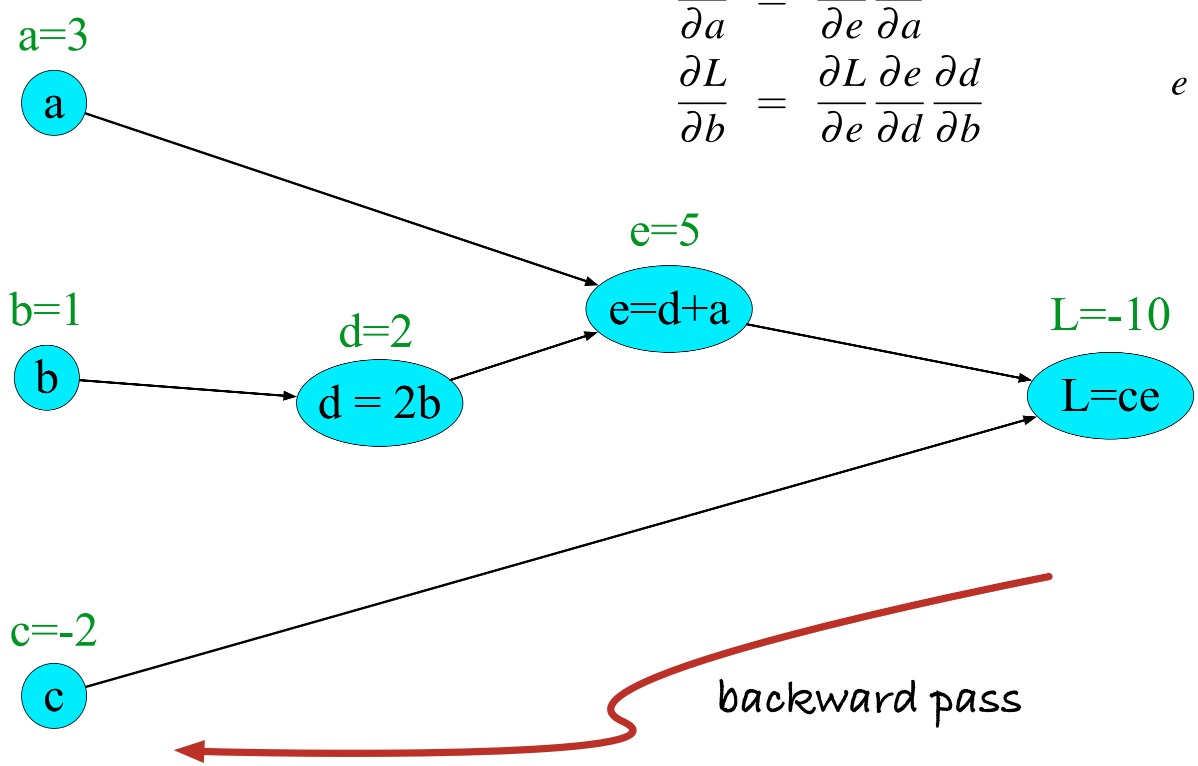
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

$$L = ce \ : \quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a + d \ : \quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

$$d = 2b \ : \quad \frac{\partial d}{\partial b} = 2$$



Example

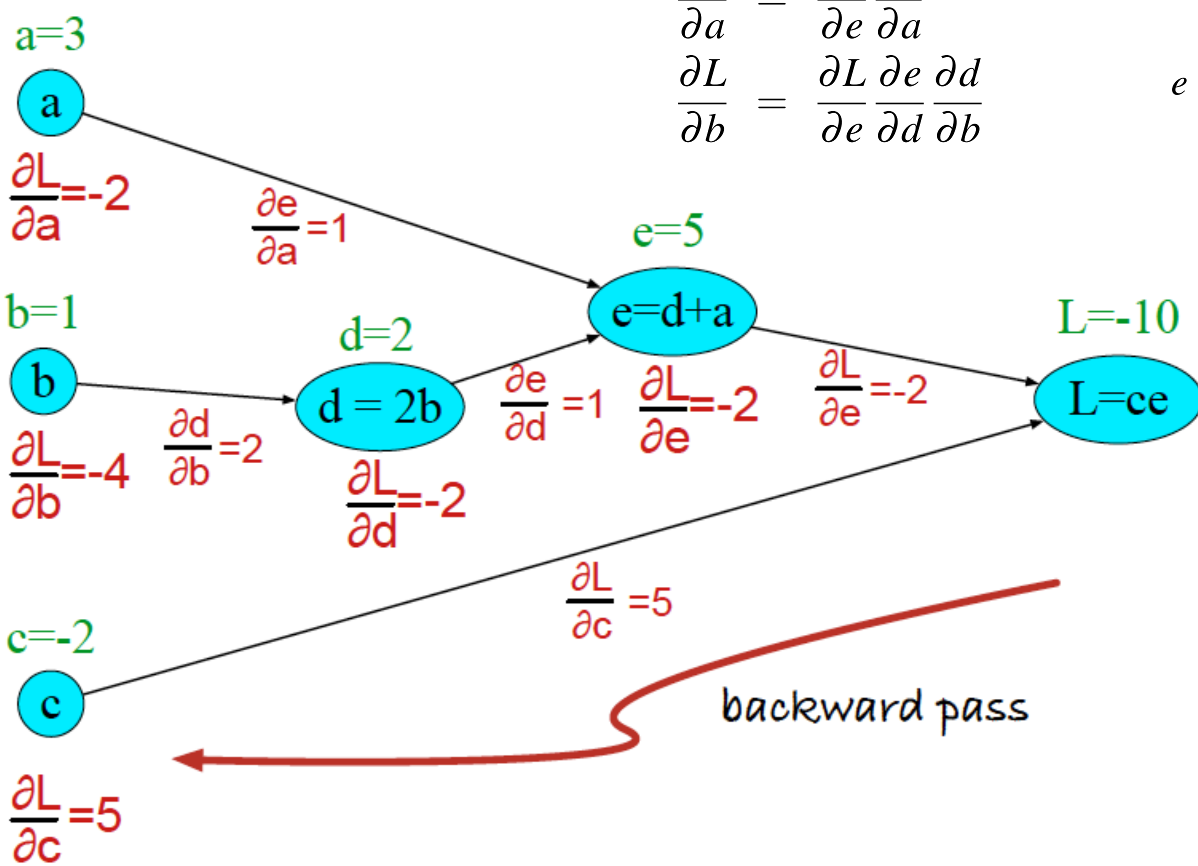
$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

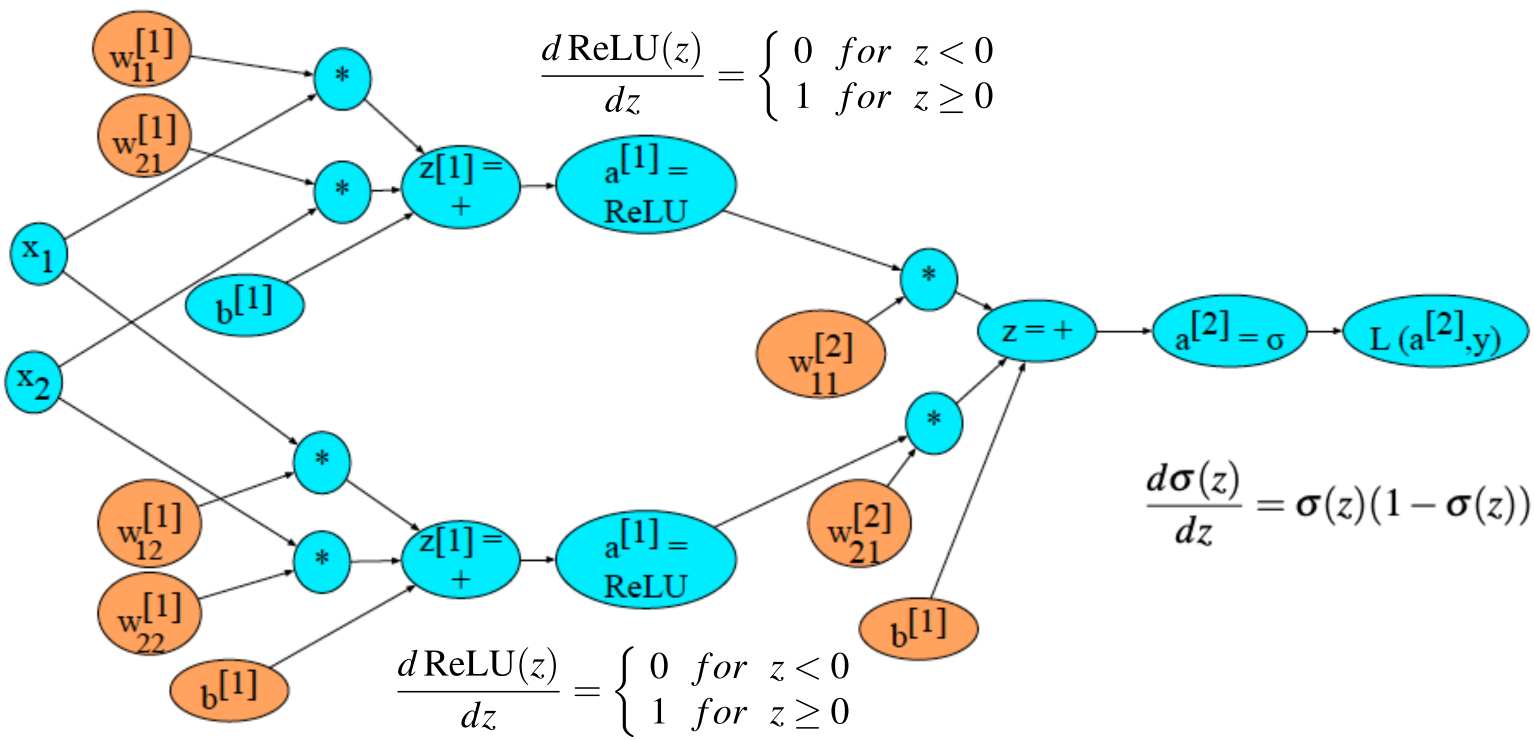
$$L = ce \;:\quad \frac{\partial L}{\partial e} = c, \frac{\partial L}{\partial c} = e$$

$$e = a + d \;:\quad \frac{\partial e}{\partial a} = 1, \frac{\partial e}{\partial d} = 1$$

$$d = 2b \;:\quad \frac{\partial d}{\partial b} = 2$$



Backpropagation in a two-layer network



Summary

To train a neural network, we need to:

- Be able to represent the network as a computation graph
- Use only differentiable operations
  - Dot product is ok
  - Sigmoid, tanh and ReLU are also ok
  - Cross-entropy loss is usually fine
- Use a toolkit that knows how to do the complicated differentiation stuff automatically
  - Write the forward pass function and let it determine the backward pass function

Word vectors and embeddings

IN4080

Natural Language Processing

Yves Scherrer

Vector semantics

How do we get there?

- Word-document (or term-document) matrices based on co-occurrence counts
  - Count weighting with tf-idf
- Word-context matrices based on co-occurrence counts
  - Dimensionality reduction (SVD, LSA)
    - Makes the vectors short and dense
- An alternative approach: word2vec
    - Creates short and dense vectors directly

Why short and dense vectors?

- Easier to use as feature vectors in machine learning (fewer weights to tune)
- Generalize better than explicit counts
- Capture semantic relations (synonymy, …) better

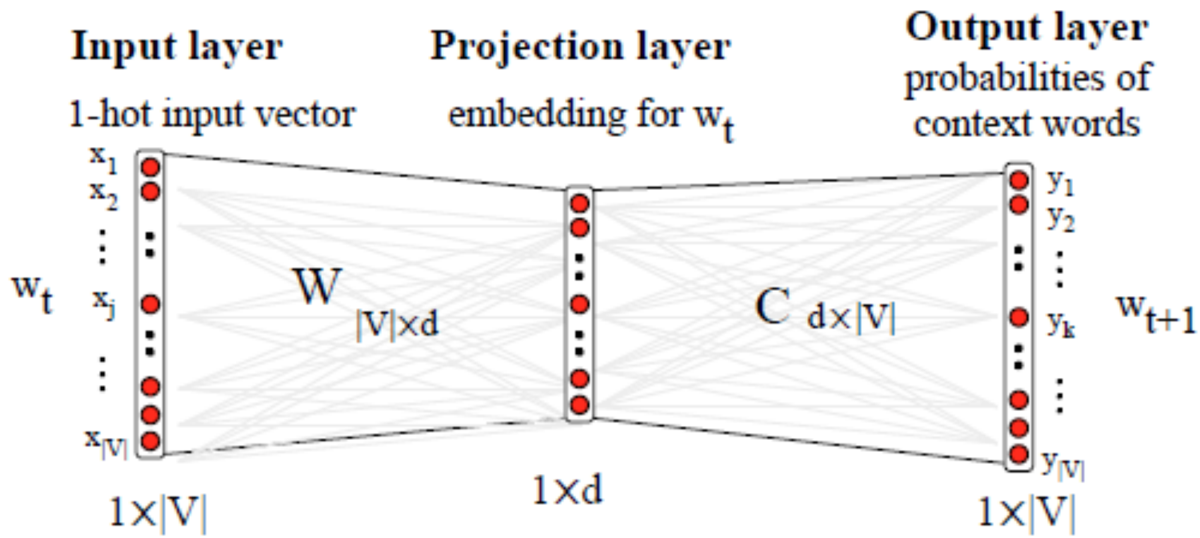
Word embeddings

- Count-based methods:
  - Count how often each word w occurs near "apricot"
- Alternative: **predict rather than count**
  - How likely is word w to show up near "apricot"?
- We don't actually care about this task, but we take the hidden representations of the network as the word embeddings.
- This is an instance of **self-supervision**:
  - Words that occur near "apricot" in the corpus can act as likely answers.
  - Words that never occur near "apricot" in the corpus can act as unlikely answers.
  - No need for human annotation.

Word embeddings

First idea: a neural bigram language model

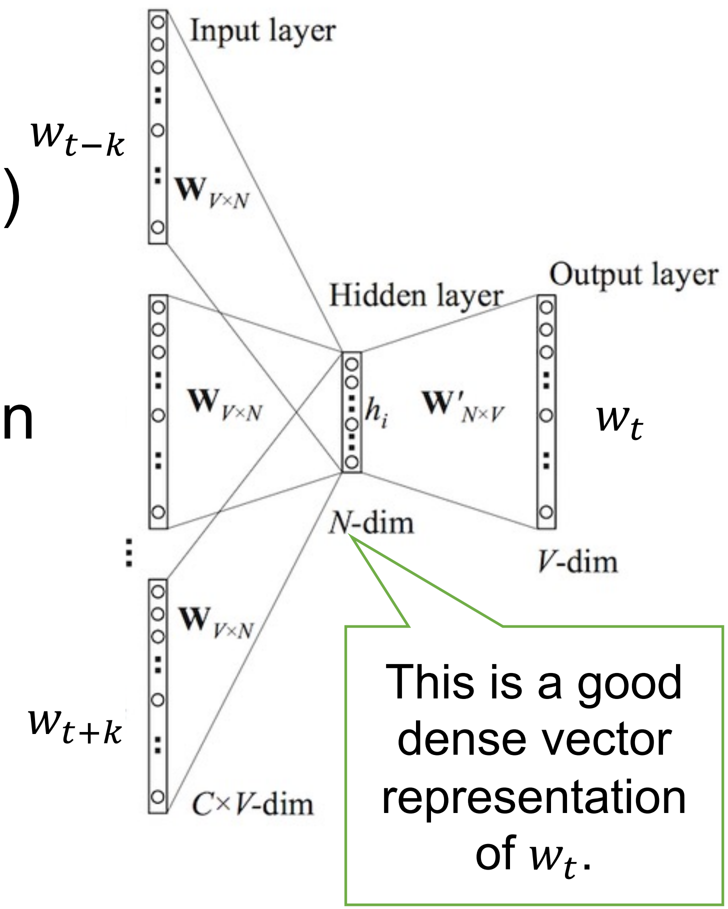
- Input: one-hot vector of word w_t
- One hidden layer
- Output: probability distribution over w_{t+1}



Word embeddings

Continuous bag-of-words (CBOW):

- Input: concatenation (or sum) of context word vectors $w_{t-k}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+k}$
- Output: probability distribution for target word w_t

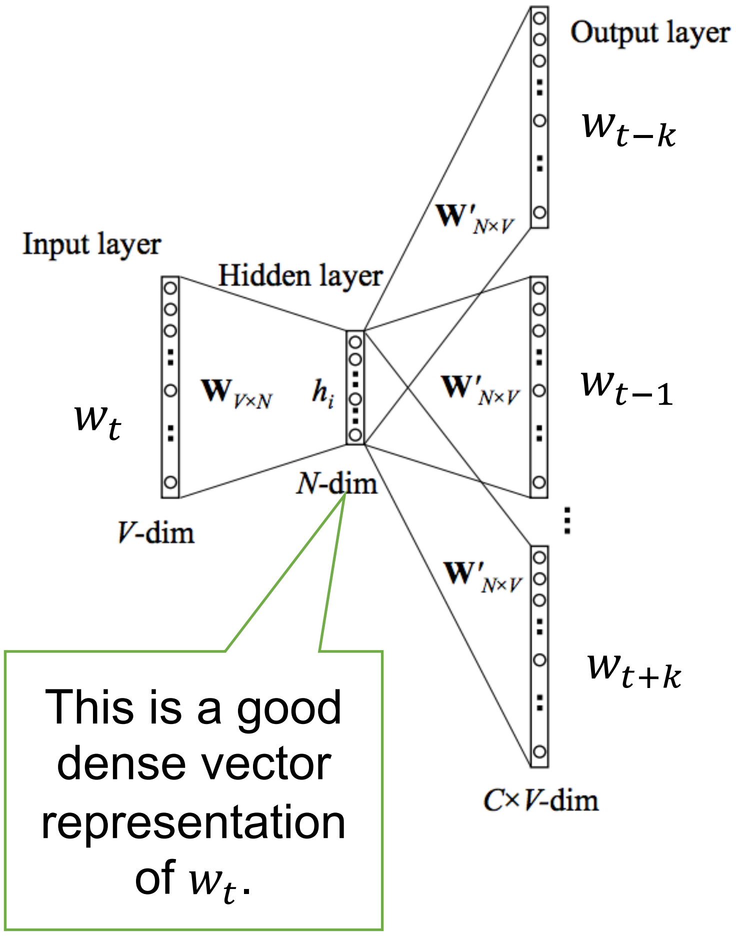


Word embeddings

Skip-gram:

- The opposite of CBOW
- Target word as input
- Context words as output

- This requires $2k$ softmax calculations
  - Complicated
  - Expensive (vocabulary vectors are very large)



Skip-gram with negative sampling

- Reformulate model as a binary classification task:
  - Input: one-hot vectors of two words w_t and w_c
  - Output: probability that w_c occurs in the context of w_t
- We need two types of training examples:
  - Word pairs with high probability (positive examples)
  - Word pairs with low probability (negative examples)
- Train a binary classifier on this data
  - Feed-forward neural network
  - A logistic regression classifier works just fine…

Training data

```
... lemon, a tablespoon of apricot preserves   or   a ...
                c1        c2     t         c3         c4
```

- For each positive example, we create k negative examples
  - Use any random word that isn't w_t

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples -

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

Computing context probabilities

- Given two vectors $\boldsymbol{w}$ and $\boldsymbol{c}$, we can compute their similarity using the dot product: $\boldsymbol{w} \cdot \boldsymbol{c}$
  - Note: cosine similarity is just a normalized dot product, and we don't need this type of normalization here.
- The dot product gives us any number. How can we convert it into a probability?
  - Sigmoid function (binary logistic regression):

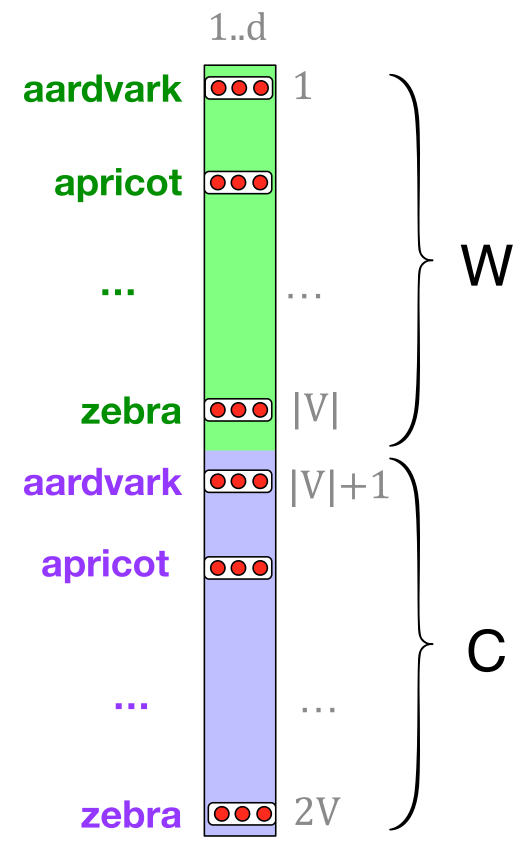
$$P(+|\boldsymbol{w}, \boldsymbol{c}) = \sigma(\boldsymbol{w} \cdot \boldsymbol{c}) = \frac{1}{1 + e^{-(\boldsymbol{w} \cdot \boldsymbol{c})}}$$

- Wait a minute: we don't *have* these vectors – wasn't the whole point to *create* them???

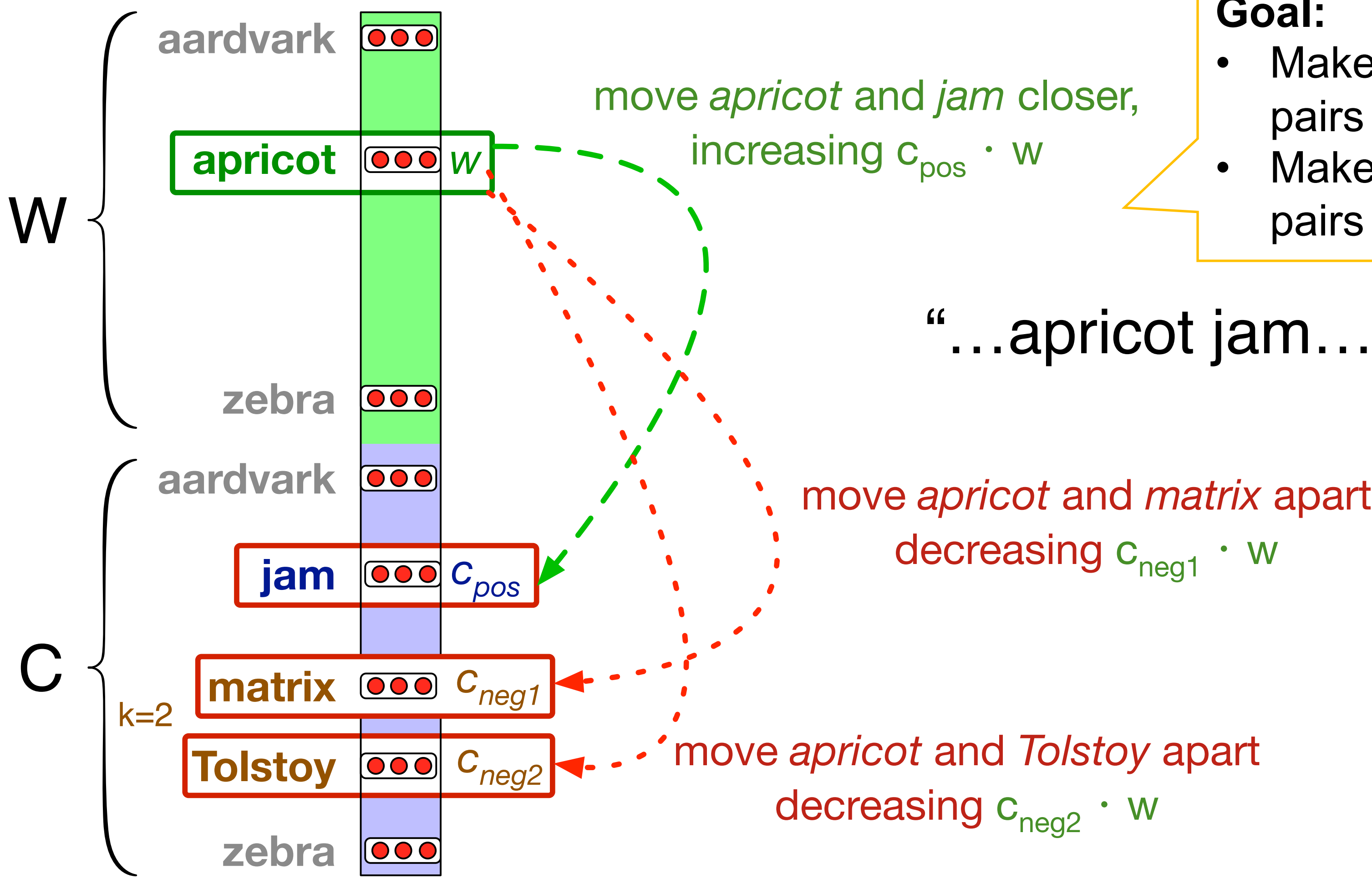
Computing context probabilities

Training procedure:

- Randomly initialize two vectors for each word of the dataset
  - One for its use as target word ($\boldsymbol{W}$)
  - One for its use as context word ($\boldsymbol{C}$)
- Pick an example from the training data (positive or negative)
- Predict probability
- Compute loss and update vectors based on gradient
- Pick next example



Training word embeddings



Goal:
- Make positive pairs more likely
- Make negative pairs less likely

"…apricot jam…"

Two sets of embeddings

- SGNS learns two sets of embeddings:
  - Embeddings for the target words w_i
  - Embeddings for the context words c_i
- It is common to just add them together, so word i is represented by the vector $w_i + c_i$.
  - That's what is commonly referred to by **word2vec**.

Effect of window size

- Small windows (± 2 context words):
  - The nearest words are **syntactically similar** words in the same taxonomy (e.g., same parts of speech).
- Large windows (± 5 context words):
  - The nearest words are **related** words in the same semantic fields.

Applications of word vectors

Word vectors or embeddings

- Each word is represented by a vector of real numbers
- Similarity between words can be measured by cosine similarity
- A word can be similar to one word in some dimensions and other words in other dimensions

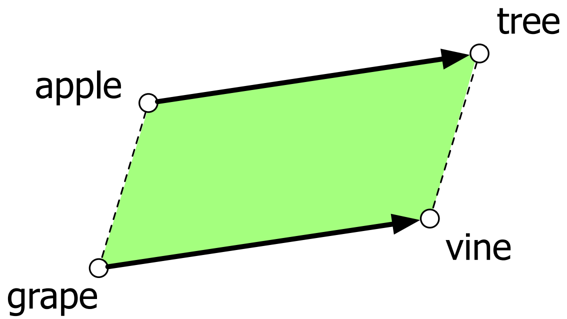
Dimensions

Word vectors				
dog	-0.4	0.37	0.02	-0.34
cat	-0.15	-0.02	-0.23	-0.23
lion	0.19	-0.4	0.35	-0.48
tiger	-0.08	0.31	0.56	0.07
elephant	-0.04	-0.09	0.11	-0.06
cheetah	0.27	-0.28	-0.2	-0.43
monkey	-0.02	-0.67	-0.21	-0.48
rabbit	-0.04	-0.3	-0.18	-0.47
mouse	0.09	-0.46	-0.35	-0.24
rat	0.21	-0.48	-0.56	-0.37

https://medium.com/@jayeshbahire

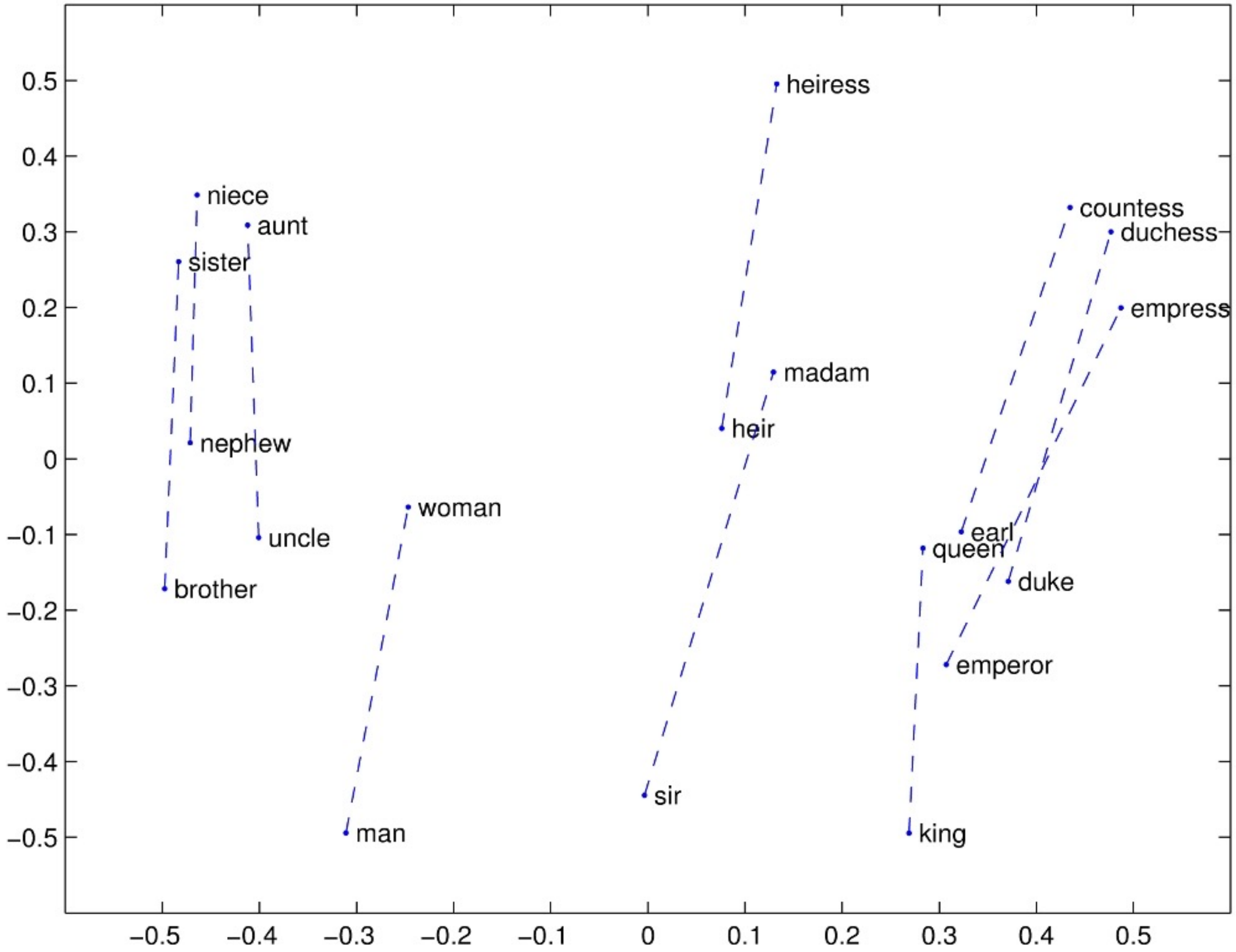
Analogical relations

- "Apple is to tree as grape is to vine."



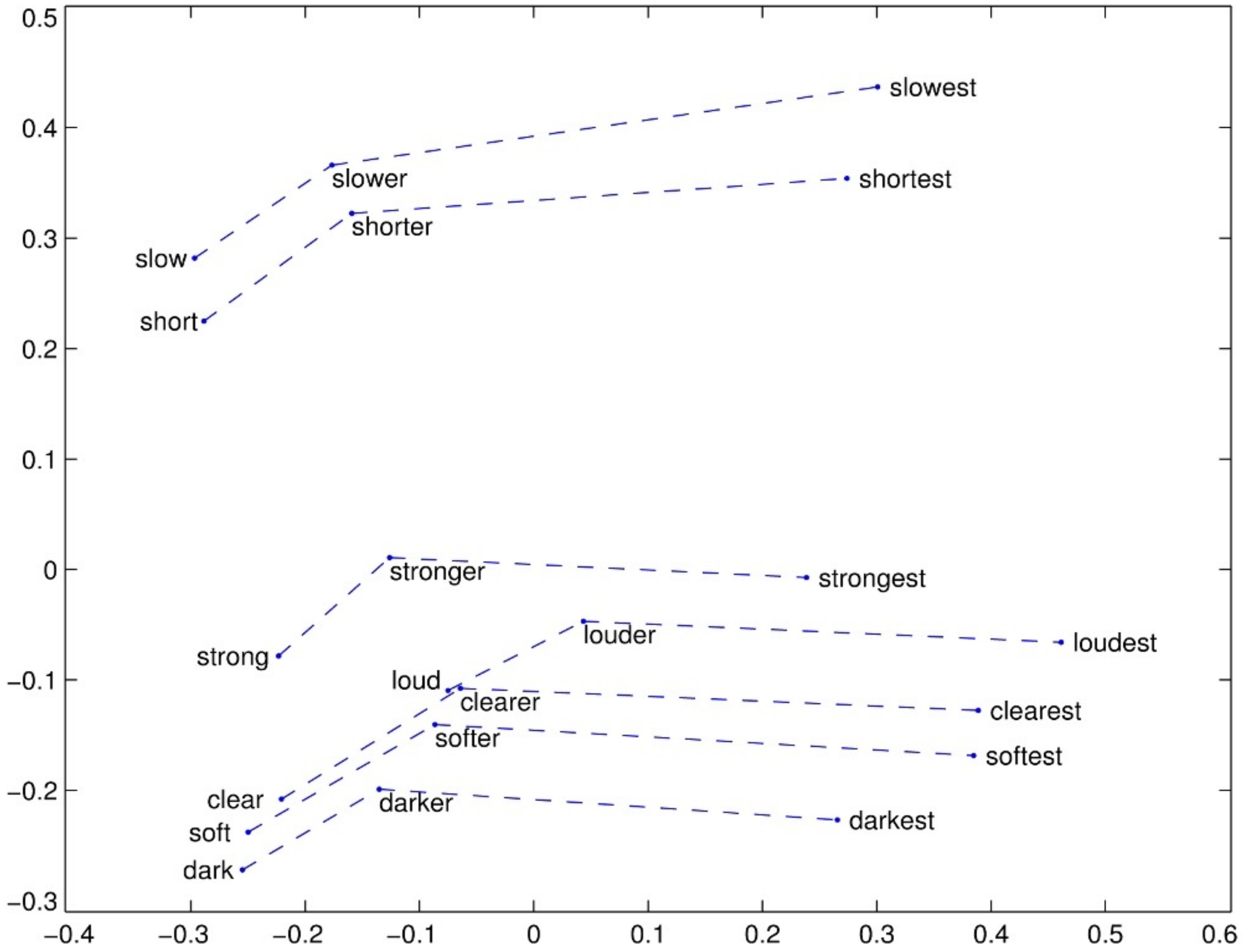
- "Man is to king as women is to ___."
  - $v_{king} - v_{man} + v_{woman} \approx v_{queen}$
- "Paris is to France as Rome is to ___."
  - $v_{France} - v_{Paris} + v_{Rome} \approx v_{Italy}$

Analogical relations



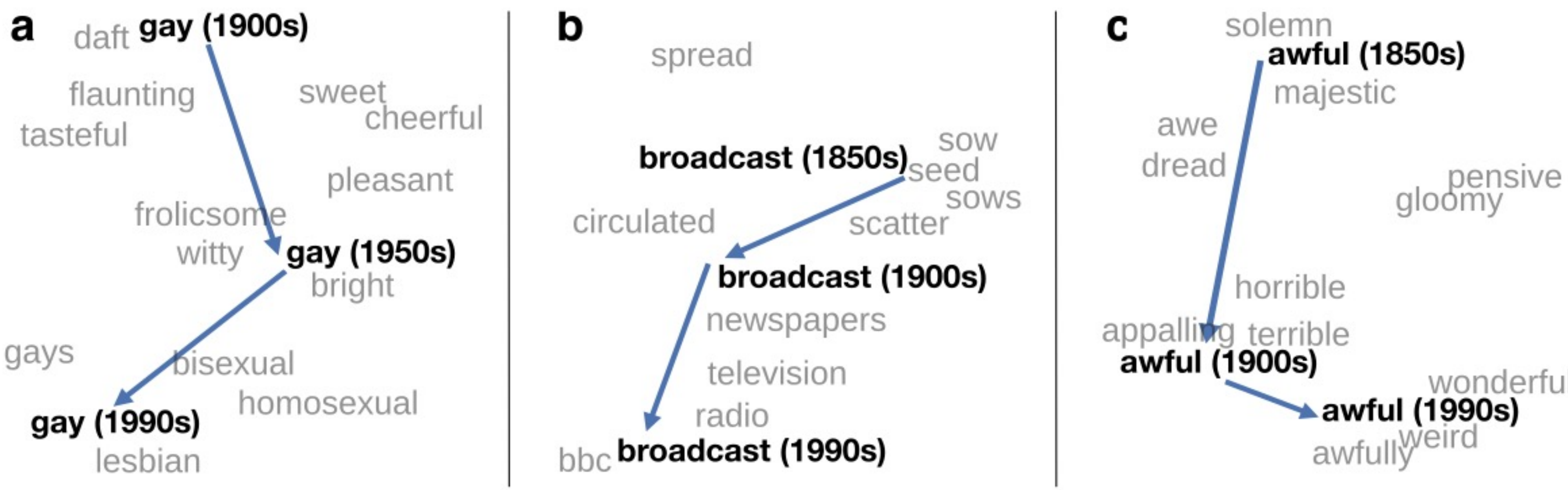
J&M

Analogical relations



J&M

Lexical semantic change



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

J&M

Bias

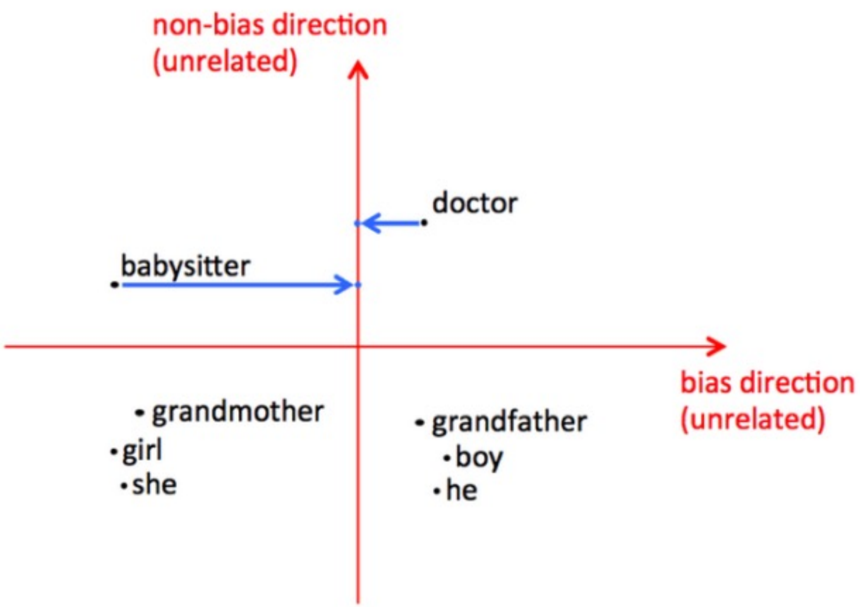
- *Man is to computer programmer as woman is to ___.*
  - $v_{programmer} - v_{man} + v_{woman} \approx v_{homemaker}$
- Different adjectives are associated with:
  - male and female terms
  - typical black names and typical white names
- Male and female terms end up relatively far apart in the vector space, even if their meaning is the same.

Bias

Debiasing:

- Neutralize the biases

- Should we debias?
- When should we (not) debias?



https://vagdevik.wordpress.com/2018/07/08/debiasing-word-embeddings

Demo

- Collection of pretrained word embeddings for various languages:
  - http://vectors.nlpl.eu/

- Interactive visualization of word similarities
  - http://vectors.nlpl.eu/explore/embeddings/en/

Evaluation of embeddings

Intrinsic evaluation:

- WordSim-353:
  - Broader “semantic relatedness”
- SimLex-999:
  - Narrower: similarity
  - Manually annotated for similarity

Word1	Word2	POS	Sim-score
old	new	A	1.58
smart	intelligent	A	9.2
plane	jet	N	8.1
woman	man	N	3.33
word	dictionary	N	3.68
create	build	V	8.48
get	put	V	1.98
keep	protect	V	5.4

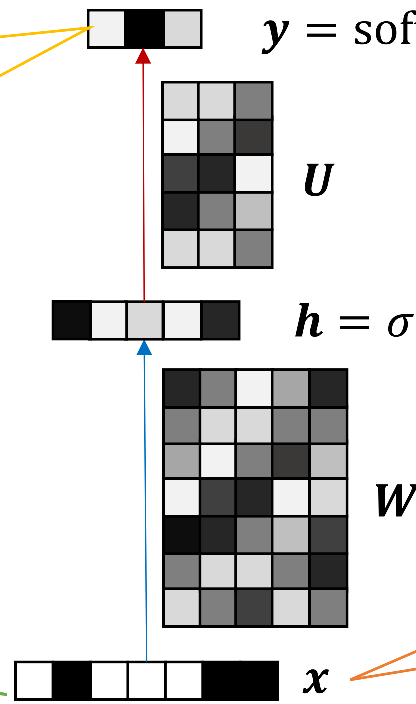
Use cases for word embeddings

- Document classification
- Language modeling

Neural document classification

The size of the prediction vector is defined by the number of classes.

$$\boldsymbol{y} = \mathrm{softmax}(\boldsymbol{U} \cdot \boldsymbol{h})$$



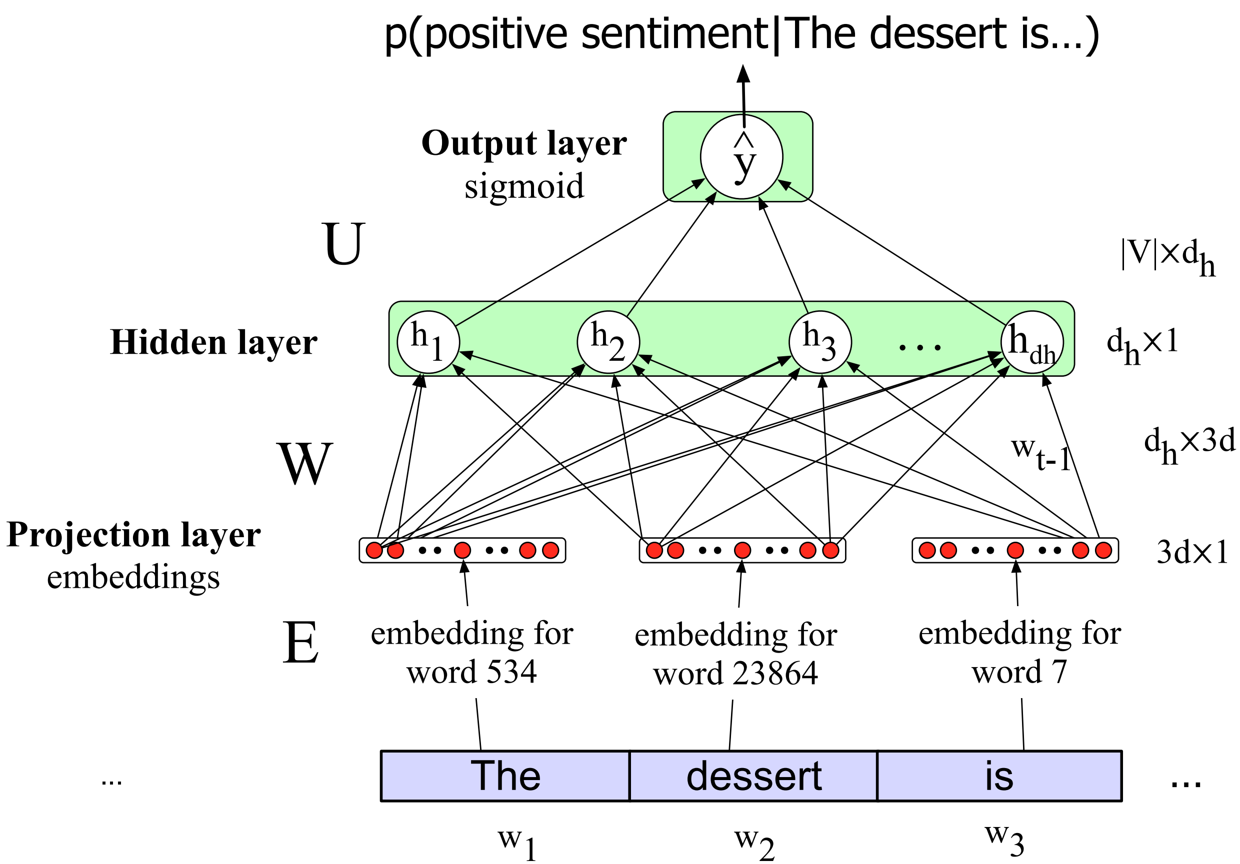
$$\boldsymbol{h} = \sigma(\boldsymbol{W} \cdot \boldsymbol{x})$$

The size of the input vector is defined by the number of distinct words in the training corpus (bag of words).

We can use pretrained word embeddings here. **How?**

Neural document classification

Maybe like this?

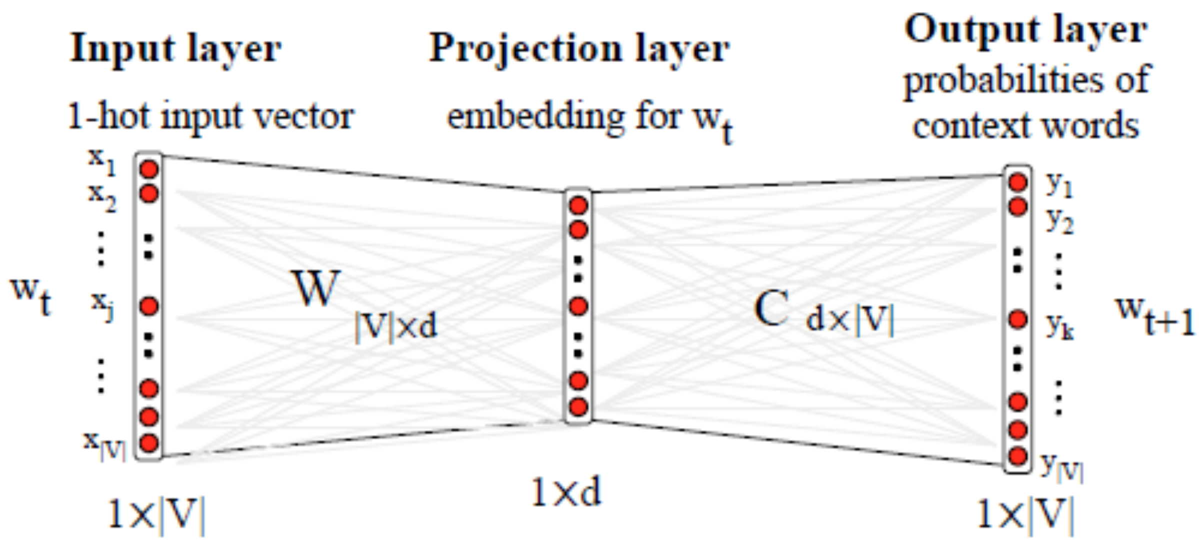


Neural document classification

- Issue: texts come in different lengths…
  - The graph in the previous slide uses only the first three words.
- Solution 1:
  - Use as many vectors as the length of the longest document.
  - Set vector values of “unused” words to zero.
  - That will be a very long vector…
- Solution 2 (**pooling**):
  - Create a single ”sentence embedding” that combines the embeddings of its words.
  - Take the mean of all the word embeddings
  - Take the element-wise maximum of all embeddings

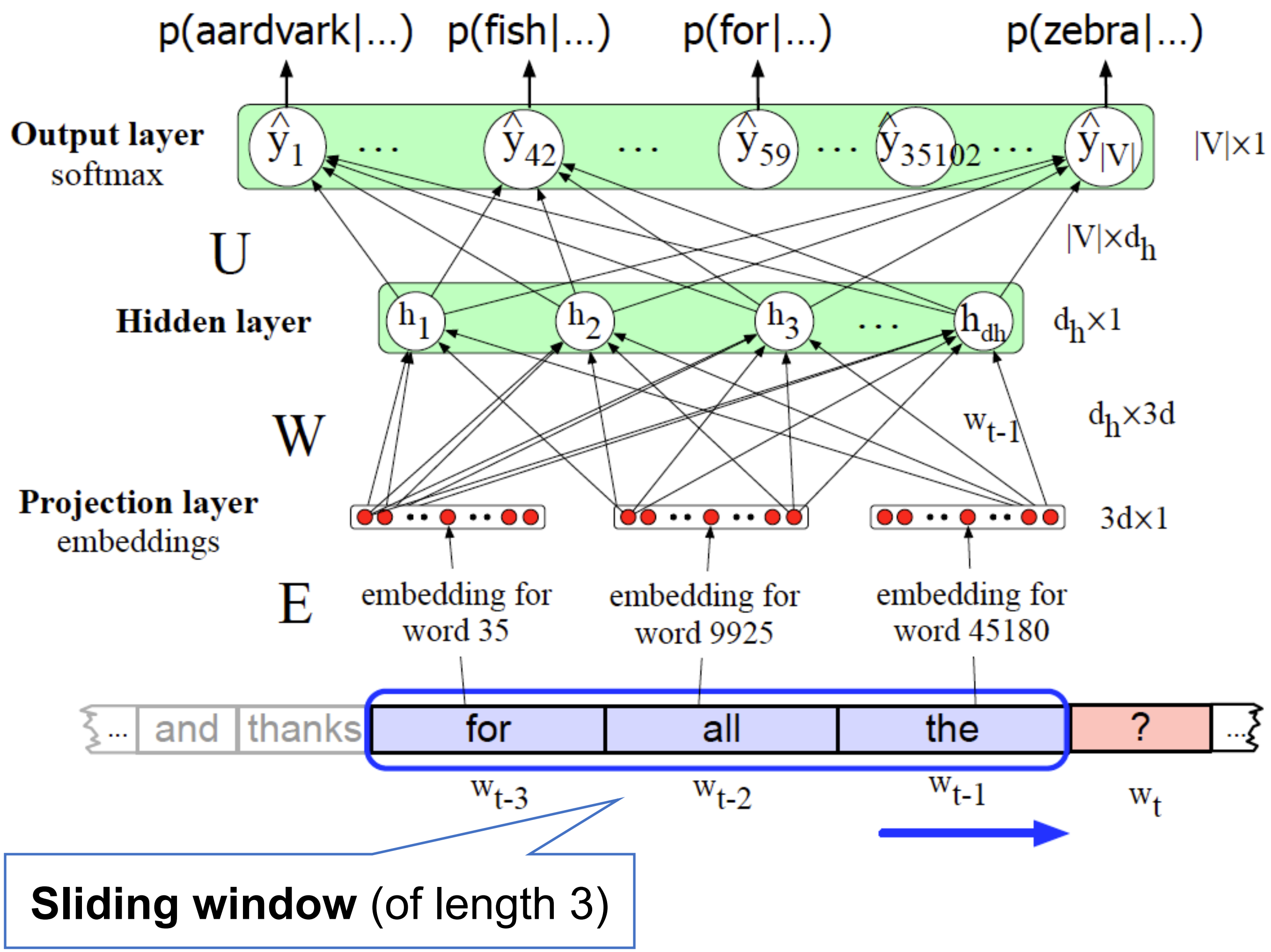
Neural language models

Recall our first attempt:



- This was for producing word embeddings. Now we assume that we have them.
- We can use more than one input word.

Neural language models



Neural language models

- This model has serious drawbacks:
  - Not efficient, need to run the same embeddings several times through the network.
  - Context is limited to window size.
- But neural LMs still work better than probabilistic ones.

Example:

- Training data:
  - Seen: I have to make sure that the cat gets fed.
  - Not seen: dog gets fed
- Test data:
  - I forgot to make sure that the dog gets ___
  - A probabilistic LM can't predict fed
  - A neural LM can use the similarity of cat and dog embeddings to generalize and predict fed after dog.