TEXT MINING

L03. VECTOR SEMANTICS

SUZAN VERBERNE 2022

Universiteit Leiden

TODAY'S LECTURE

- Quiz about week 2
- Preliminaries
  - distributional hypothesis
  - vector space model
- Introduction to neural networks
- Word embeddings
  - word2vec

QUIZ ABOUT WEEK 2

- What is Optical Character Recognition (OCR)?
  a. A technique for recognizing the correct character encoding
  b. A technique for converting handwritten text to digital text
  c. A technique for converting the image of a printed text to a digital text
  d. A technique for converting a formatted text to a plain text

QUIZ ABOUT WEEK 2

- What is Optical Character Recognition (OCR)?
  a. A technique for recognizing the correct character encoding
  b. A technique for converting handwritten text to digital text
  c. A technique for converting the image of a printed text to a digital text
  d. A technique for converting a formatted text to a plain text

QUIZ ABOUT WEEK 2

- What is the main limitation of ASCII?
  a. It can only encode letters that occur in the American alphabet
  b. It is not compatible with other encodings
  c. It compresses text files and causes quality loss

QUIZ ABOUT WEEK 2

- What is the main limitation of ASCII?
  a. It can only encode letters that occur in the American alphabet
  b. It is not compatible with other encodings
  c. It compresses text files and causes quality loss

QUIZ ABOUT WEEK 2

- What kind of strings does this regular expression match: /<[^>]+>/
  a. Smaller than / larger than
  b. Any HTML/XML tag
  c. Any URL

QUIZ ABOUT WEEK 2

- What kind of strings does this regular expression match: /<[^>]+>/
  a. Smaller than / larger than
  b. Any HTML/XML tag
  c. Any URL

QUIZ ABOUT WEEK 2

- Do you think it is possible to make a language independent lemmatizer?
  a. Yes, because a lemmatizer uses a dictionary
  b. Yes, because a lemmatizer uses patterns
  c. No, because a lemmatizer uses a dictionary
  d. No, because a lemmatizer uses patterns

QUIZ ABOUT WEEK 2

- Do you think it is possible to make a language independent lemmatizer?

a. Yes, because a lemmatizer uses a dictionary

b. Yes, because a lemmatizer uses patterns

c. No, because a lemmatizer uses a dictionary

d. No, because a lemmatizer uses patterns

QUIZ ABOUT WEEK 2

- What is the Levenshtein distance between ‘sheep’ and ‘ship’?
  a. 1
  b. 2
  c. 3
  d. 4

QUIZ ABOUT WEEK 2

- What is the Levenshtein distance between ‘sheep’ and ‘ship’?
  a. 1
  b. 2
  c. 3
  d. 4

QUIZ ABOUT WEEK 2

- Did you work on the exercise of week 2? (pre-processing with Spacy)
   a. No
   b. Yes, I started
   c. Yes, I completed at least half of it
   d. Yes, I completed it

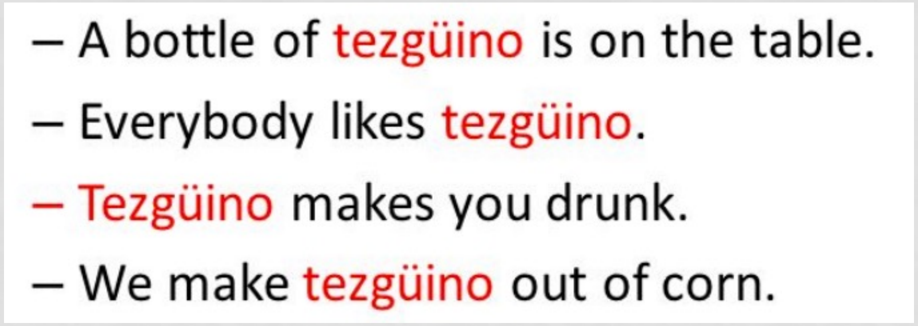
VECTOR SEMANTICS PRELIMINARIES

WHERE TO START

- Linguistics: Distributional hypothesis
- Information Retrieval: Vector Space Model (VSM)

DISTRIBUTIONAL HYPOTHESIS

- Harris, Z. (1954). “Distributional structure”. *Word*. 10 (23): 146–162
- The context of a word defines its meaning

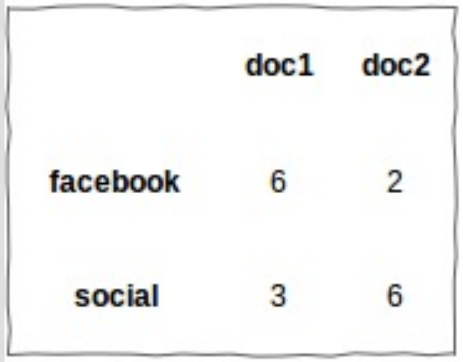


– A bottle of tezgüino is on the table.
– Everybody likes tezgüino.
– Tezgüino makes you drunk.
– We make tezgüino out of corn.

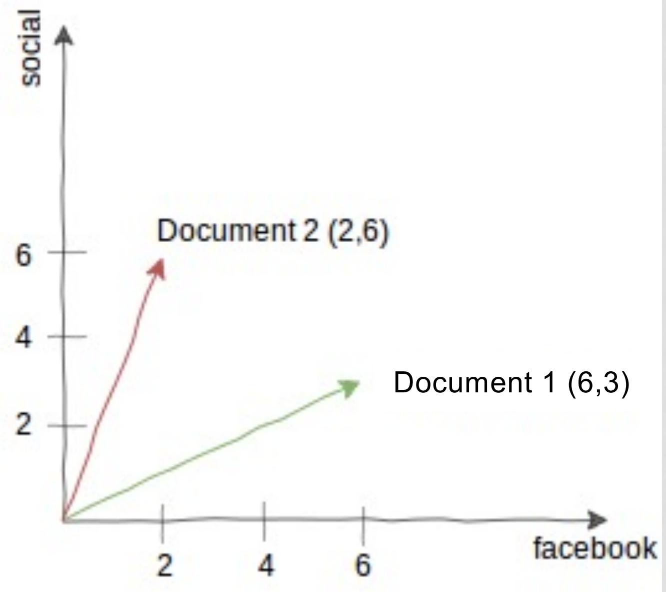
- Words that occur in similar contexts tend to be similar

VECTOR SPACE MODEL

- Traditional Vector Space Model (Information Retrieval):
  - documents and queries represented in a vector space
  - where the dimensions are the words



	doc1	doc2
facebook	6	2
social	3	6



VECTOR SPACE MODEL

Two problems with the vector space model:

- synonymy: many ways to refer to the same object, e.g. bicycle and bike
- polysemy: many words have more than one distinct meaning, e.g. bank, bass, chips

ALTERNATIVE SEMANTICS

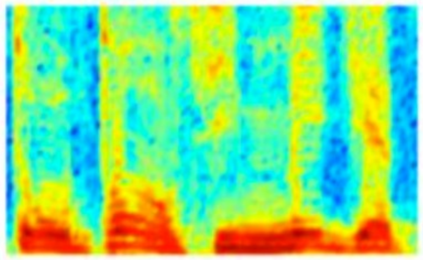
- Alternative: do not represent documents by words, but by
  - topics → topic modelling (lecture 8)
  - concepts → word embeddings
    - Idea: represent words by dense vectors, and place more similar words closer to each other in the vector space

WORD EMBEDDINGS

VECTOR SPACE MODEL

Image and audio processing systems work with rich, high-dimensional datasets encoded as vectors.

AUDIO



Audio Spectrogram

DENSE

IMAGES

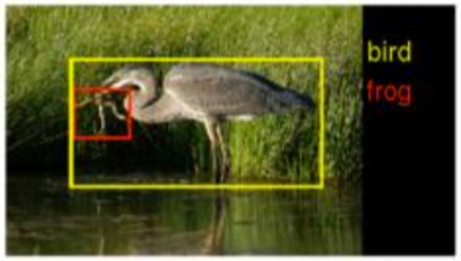


Image pixels

DENSE

TEXT

0	0	0	0.2	0	0.7	0	0	0	...	...

Word, context, or document vectors

SPARSE

VECTOR SPACE MODEL

However, natural language processing systems traditionally treat words as discrete atomic symbols.

- Encodings are arbitrary
- Provide no useful information to the system
- Leads to data sparsity (So, we need more data)

Word as Atomic Units

cat		[0,...,0,1,0,...,0]
dog		[0,...,1,0,0,...,0]

WORD EMBEDDINGS

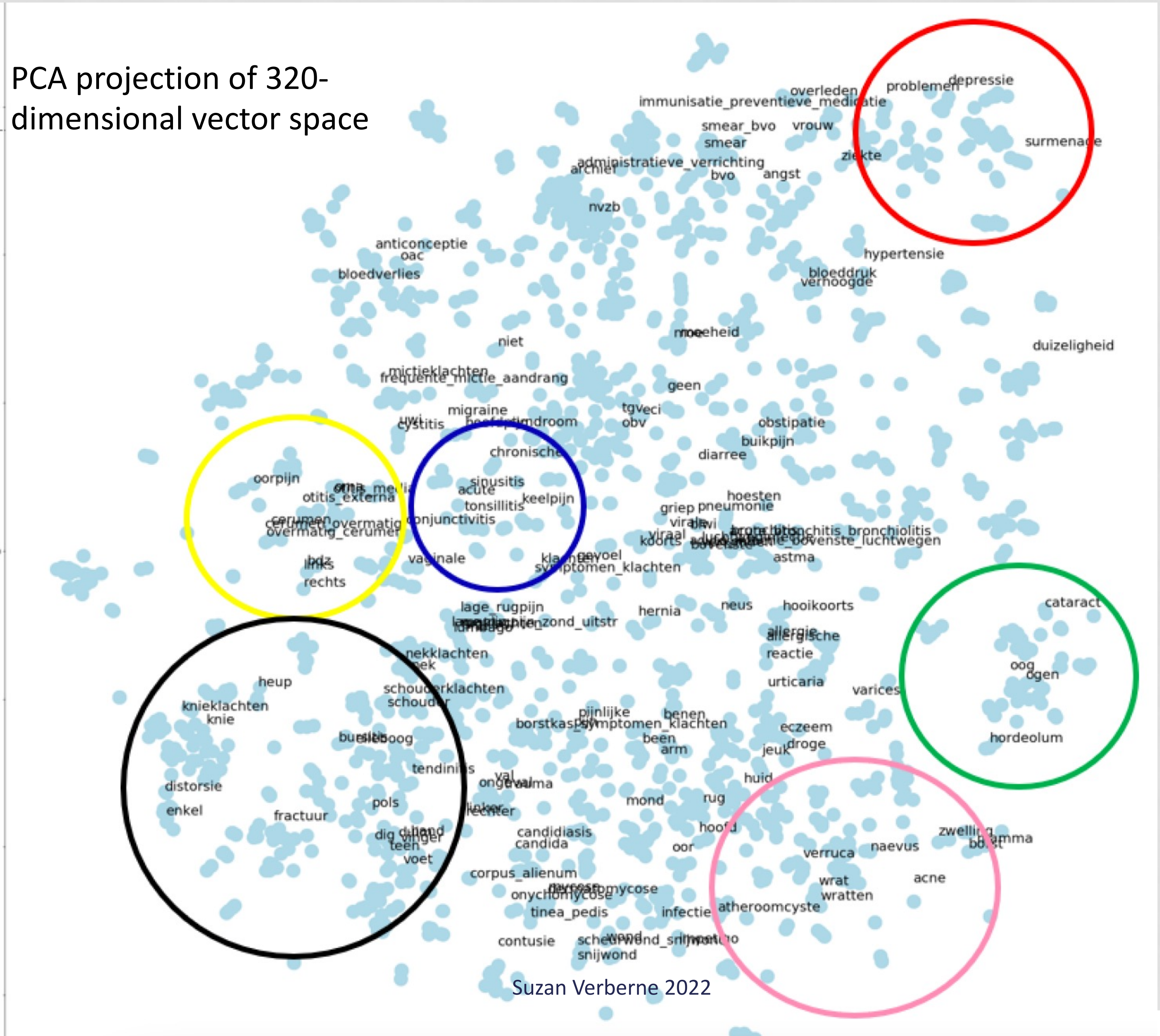
- Word embeddings are dense representations of words

$$cat \longrightarrow [0.122,0.81,0.405,\ldots,0.77]$$

WORD EMBEDDINGS

- Word embeddings models represent (embed) words in a continuous vector space
- The vector space is relatively low-dimensional (100 – 320 dimensions instead of 10,000s)
- Semantically and syntactically similar words are mapped to nearby points (Distributional Hypothesis)

PCA projection of 320-dimensional vector space



INTRODUCTION TO NEURAL NETWORKS

J&M CHAPTER 7

INTRODUCTION

Two options:

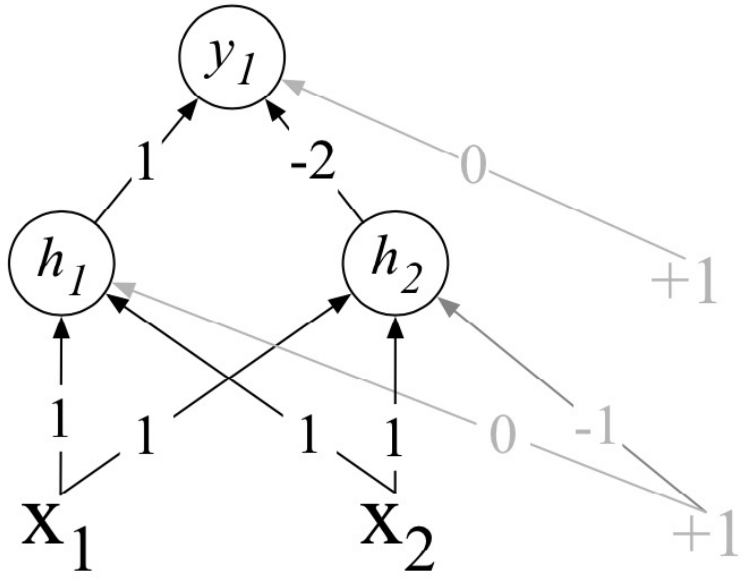
- If you are familiar with feedforward neural networks and the role of the hidden layer, I have an exercise for you (the XOR problem)
- If your knowledge is a bit rusty, you can listen to my explanation

EXERCISE: THE XOR PROBLEM

- The XOR (exclusive OR) function is a non-linear function
- Unlike AND and OR it cannot be solved with one neuron
- A feedforward network with one hidden layer:

Small neural network that solves the XOR problem

Inputvector (x1,x2)
x1 ∈ {0,1}
x2 ∈ {0,1}



What is the output for each of these vectors:
[0,0]
[0,1]
[1,0]
[1,1]

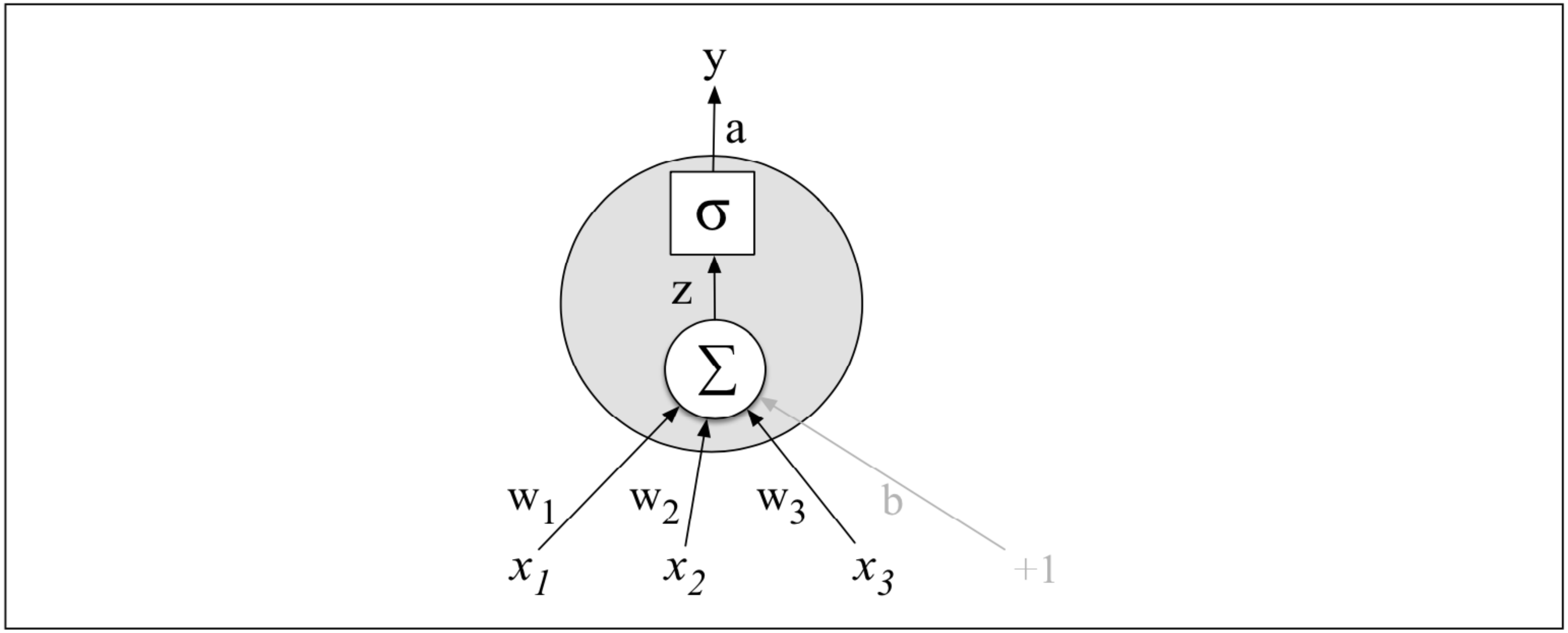
J&M, section 7.2: https://web.stanford.edu/~jurafsky/slp3/7.pdf

INTRODUCTION

- Neural networks share much of the same mathematics as logistic regression*
- But neural networks are a more powerful classifier than logistic regression:
  - multiple nodes = multiple functions = non-linearity
  - multiple layers = multiple abstractions over the input data

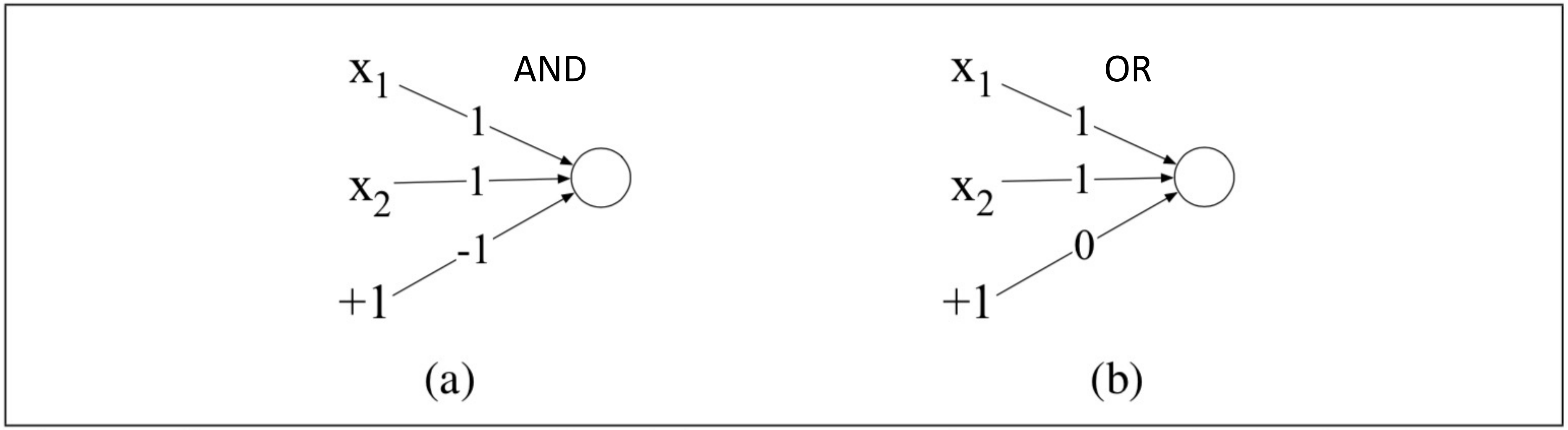
* If you are not familiar with Logistic Regression you can read J&M chapter 5

NEURAL NETWORK UNITS (J&M 7.1)



- Figure of one single unit with one function for three input variables (x1,x2,x3)
- σ: non-linear activation function
- Commonly used activation function : ReLU (Rectifier Linear Unit): $y = \max(x, 0)$

FUNCTIONS WITH ONE NEURON



- the numbers on the arrows are weights
- $y = w_1x_1 + w_2x_2 + b$
- the input vector is (x_1, x_2), we need to determine the weight vector (w_1, w_2)
- +1 is the bias (with weight -1 in (a) and weight 0 in (b))

LEARNING THE WEIGHTS

- In this example we manually added the weights
- When trained on data, the weights for neural networks are learned automatically (supervised learning using the true labels)
- By learning the weights to optimize the output classification the hidden layers will learn to form useful representations

THE XOR PROBLEM (J&M 7.2)

- Why one unit is not sufficient for non-linear functions ('exclusive OR')

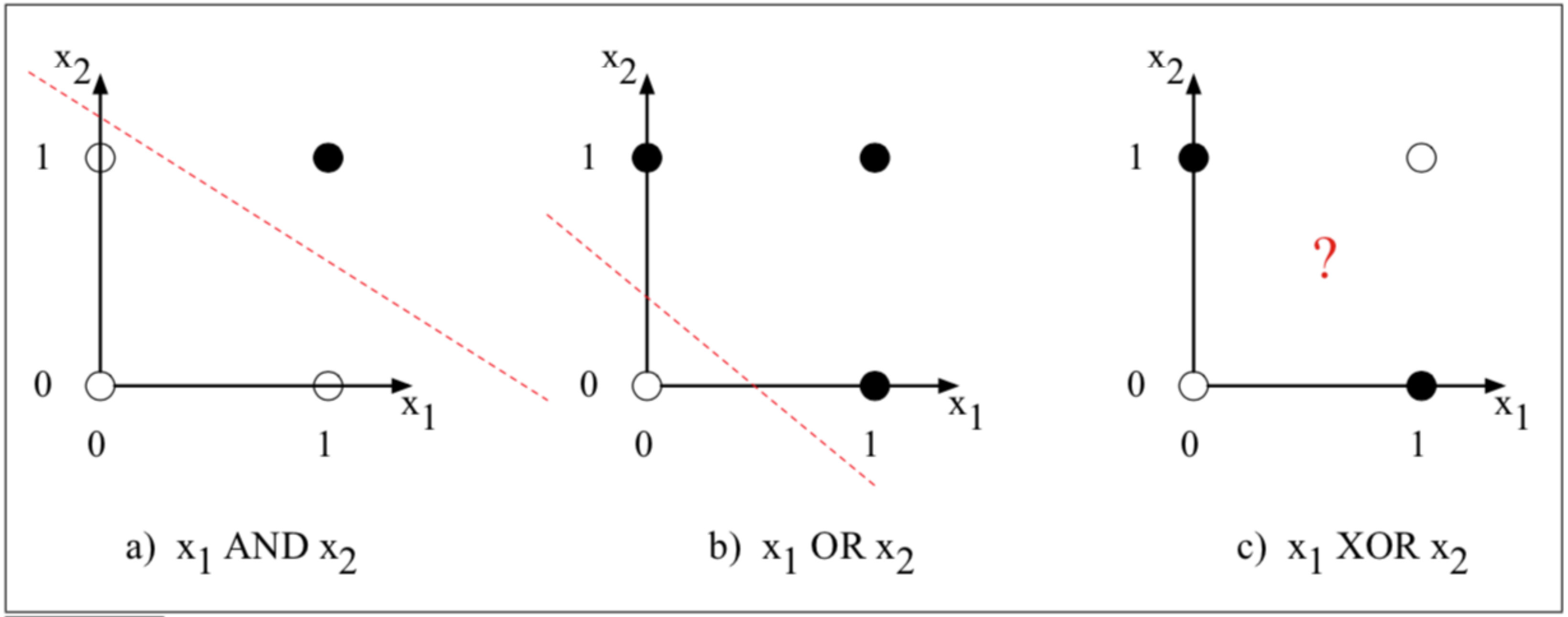


Figure 7.5 The functions AND, OR, and XOR, represented with input x_1 on the x-axis and input x_2 on the y axis. Filled circles represent perceptron outputs of 1, and white circles perceptron outputs of 0. There is no way to draw a line that correctly separates the two categories for XOR. Figure styled after Russell and Norvig (2002).

ADDING A HIDDEN LAYER

Small neural network for solving the XOR problem

Inputvector (x1,x2)
x1 ∈ {0,1}
x2 ∈ {0,1}

h_1, h_2, and y_1 are ReLU units

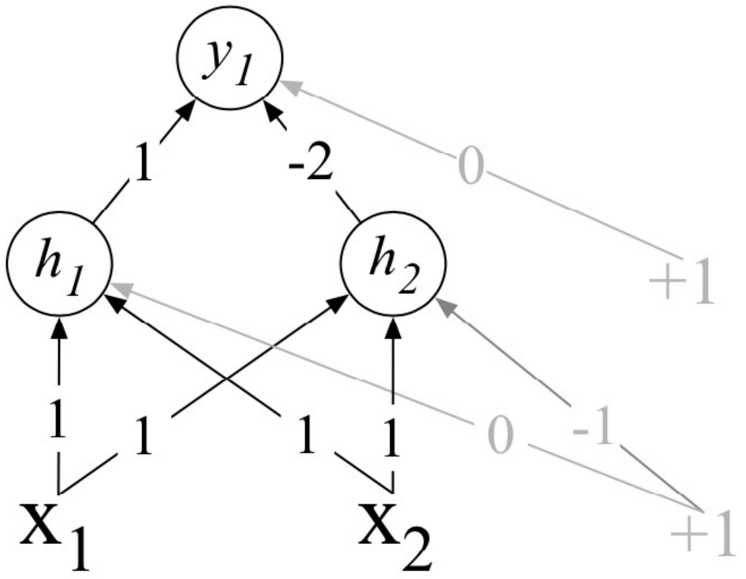


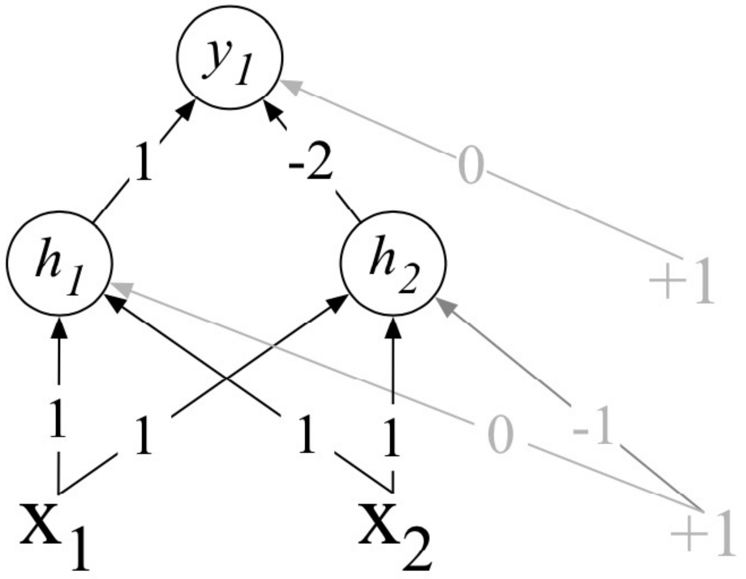
Figure 7.6 XOR solution after Goodfellow et al. (2016). There are three ReLU units, in two layers; we've called them h_1, h_2 (h for "hidden layer") and y_1. As before, the numbers on the arrows represent the weights w for each unit, and we represent the bias b as a weight on a unit clamped to +1, with the bias weights/units in gray.

ADDING A HIDDEN LAYER

Small neural network for solving the XOR problem

Inputvector (x1,x2)
x1 ∈ {0,1}
x2 ∈ {0,1}

h_1, h_2, and y_1 are ReLU units



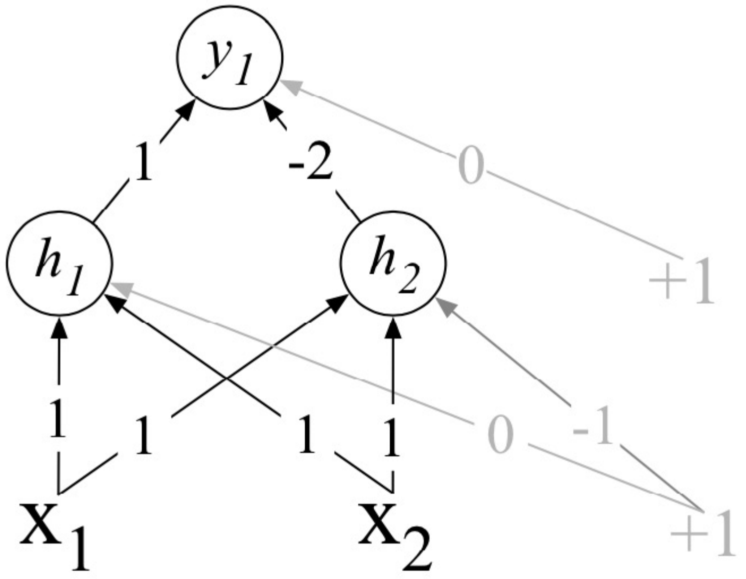
What is the output for each of these vectors:
[0,0]
[0,1]
[1,0]
[1,1]

ADDING A HIDDEN LAYER

Small neural network for solving the XOR problem

Inputvector (x1,x2)
x1 ∈ {0,1}
x2 ∈ {0,1}

h_1, h_2, and y_1 are ReLU units



What is the output for each of these vectors:
[0,0]
[0,1]
[1,0]
[1,1]

[0, 0] -> [1*0 +1*0 + 0*1, 1*0 + 1*0 -1*1] = [0, -1] ~>ReLU: [0, 0] -> 1*0 - 2*0 +0*1 = 0-0+0 = 0
[0, 1] -> [1*0 +1*1 + 0*1, 1*0 + 1*1 -1*1] = [1, 0] ~>ReLU: [1, 0] -> 1*1 - 2*0 +0*1 = 1-0+0 = 1
[1, 0] -> [1*1 +1*0 + 0*1, 1*1 + 1*0 -1*1] = [1, 0] ~>ReLU: [1, 0] -> 1*1 - 2*0 +0*1 = 1-0+0 = 1
[1, 1] -> [1*1 + 1*1 + 0*1, 1*1 + 1*1 -1*1] = [2, 1] ~>ReLU: [2, 1] -> 1*2 -2*1 + 0*1 = 2-2+0 = 0

- The hidden layer is a new representation that maps [0,1] and [1,0] to the same point

THE REPRESENTATION ON THE HIDDEN LAYER

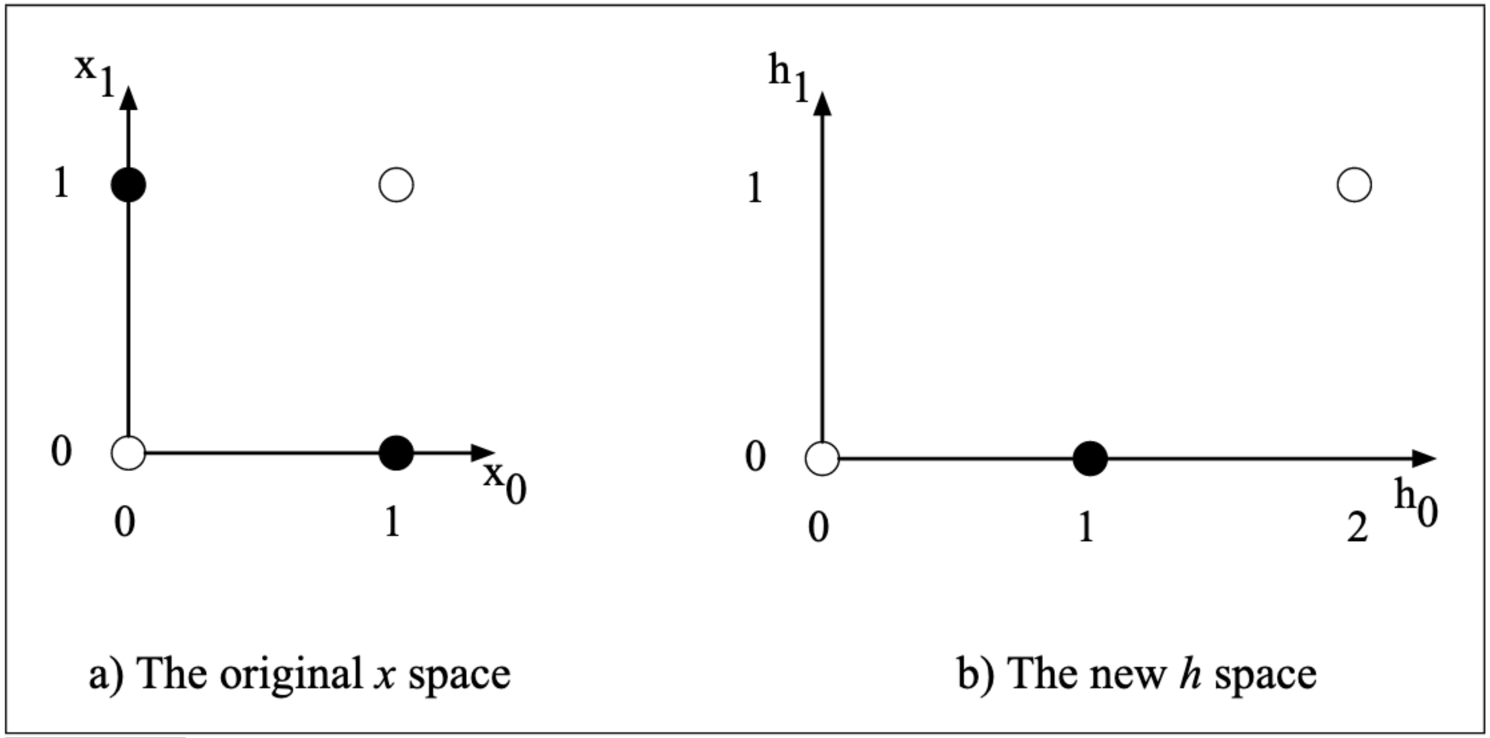
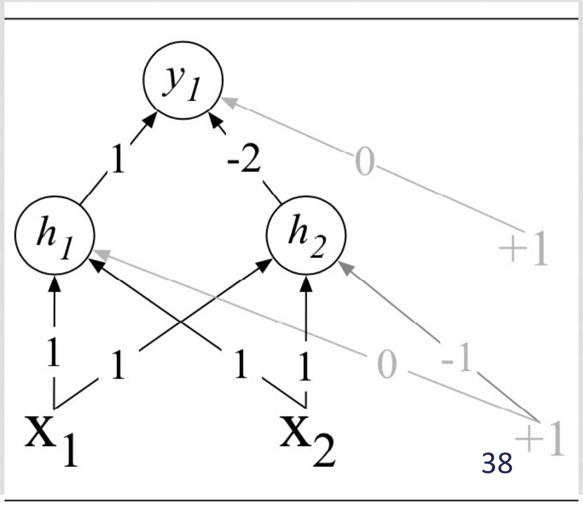


Figure 7.7 The hidden layer forming a new representation of the input. Here is the representation of the hidden layer, *h*, compared to the original input representation *x*. Notice that the input point [0 1] has been collapsed with the input point [1 0], making it possible to linearly separate the positive and negative cases of XOR. After Goodfellow et al. (2016).

FEEDFORWARD NETWORKS (J&M 7.3)

- A feedforward network is a multilayer network in which the units are connected with no cycles
- Simple feedforward networks have three kinds of nodes: input units, hidden units, and output units
- In the standard architecture, each layer is fully-connected



FEEDFORWARD NETWORK AS CLASSIFIER

- Binary classification (e.g. yes/no): single output node
  - *y* is the probability of postive output (yes)
- Multi-class classification: one output node for each category,
  - y is the probability of that category
  - The output layer thus gives a probability distribution

FEEDFORWARD NETWORK AS CLASSIFIER

- To get the probability distribution:
- The output of the classifier z (real-valued numbers) is transformed to a probability distribution using the softmax function

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

d=dimensionality

Example:
z = [0.5,3.6,-1.2]
-> softmax(z) = [0.043,0.949,0.0078]

TRAINING NEURAL NETS

- Feedforward neural net: supervised learning with input x and correct output y
- The system produces *ŷ*, an estimation of the true *y*
- The goal of the training procedure is to learn weights *W*[*i*] and bias *b*[*i*] for each layer *i* that make *ŷ* for each training observation as close as possible to the true *y*
- Find the parameters that minimize this loss function: gradient descent optimization algorithm

(see Introduction to Deep Learning, or J&M section 7.6)

TRAINING NEURAL NETS

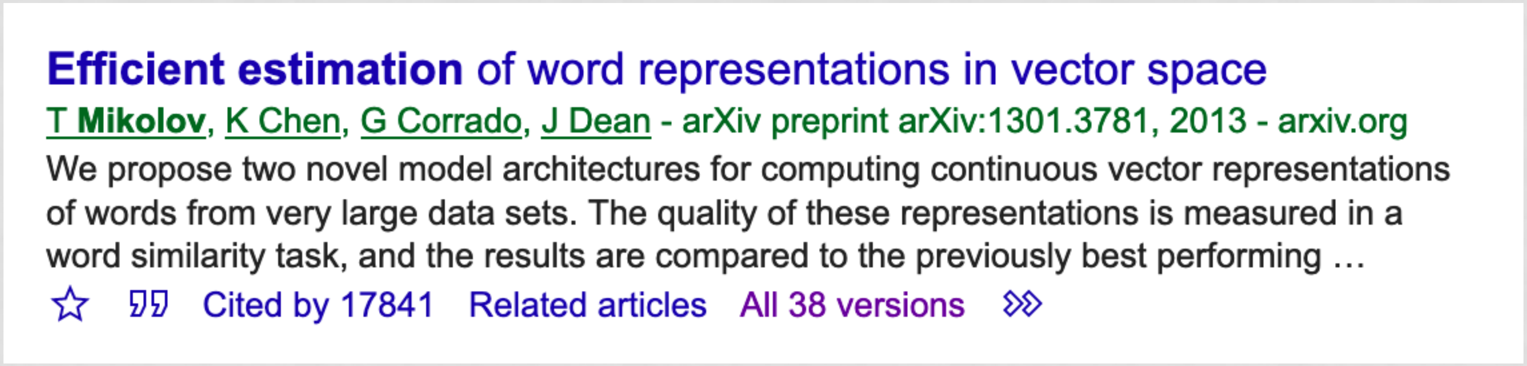
- Challenge: optimizing the weight parameters in all layers of the network, even though the loss is computed only at the very end of the network
- The solution is the error backpropagation algorithm
  - iterative, recursive and efficient method for computing the weights updates to improve the network

WORD2VEC

J&M, SECTION 6.8

WHAT IS WORD2VEC?

- Word2vec is a particularly computationally-efficient predictive model for learning word embeddings from raw text



Efficient estimation of word representations in vector space

T **Mikolov**, K Chen, G Corrado, J Dean - arXiv preprint arXiv:1301.3781, 2013 - arxiv.org

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing …

WHAT IS WORD2VEC?

- Intuition:
  - Train a neural classifier on a binary prediction task (on a text without labels!): “Is word *w* likely to show up near the word *bicycle*?”
  - We don’t actually care about this prediction task; instead we’ll take the learned classifier weights on the hidden layer as the word embeddings

WHERE DOES IT COME FROM

- Neural network language model (NNLM) (Bengio et al., 2003)
- Mikolov proposed to learn word vectors using a neural network with a single hidden layer (Mikolov et al 2013): word2vec
- Many neural architectures and models have been proposed for computing word vectors
  - GloVe (2014) - Global Vectors for Word Representation
  - FastText (2017) - Enriching Word Vectors with Subword Information
  - ELMo (2018) - Deep contextualized word representations
  - BERT (2019) - Bidirectional Encoder Representations from Transformers (lecture 7)

WORD2VEC

- Starting point: large collection (e.g. 10 Million words)
- First step: extract the vocabulary (e.g. 10,000 terms)
- Goal: to represent each of these 10,000 terms as a dense, lower-dimensional vector (typically 100-400 dimensions)

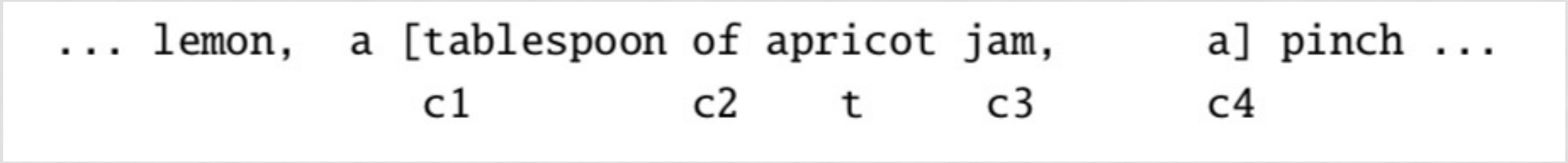
TRAINING WORD2VEC

- Training method: skip-gram with negative sampling

- Training task: binary classification of words in the text
  1. Treat the target word and a neighboring context word as positive examples
  2. Randomly sample other words in the lexicon to get negative samples
  3. Train a classifier to distinguish those two cases
  4. The learned weights are the embeddings

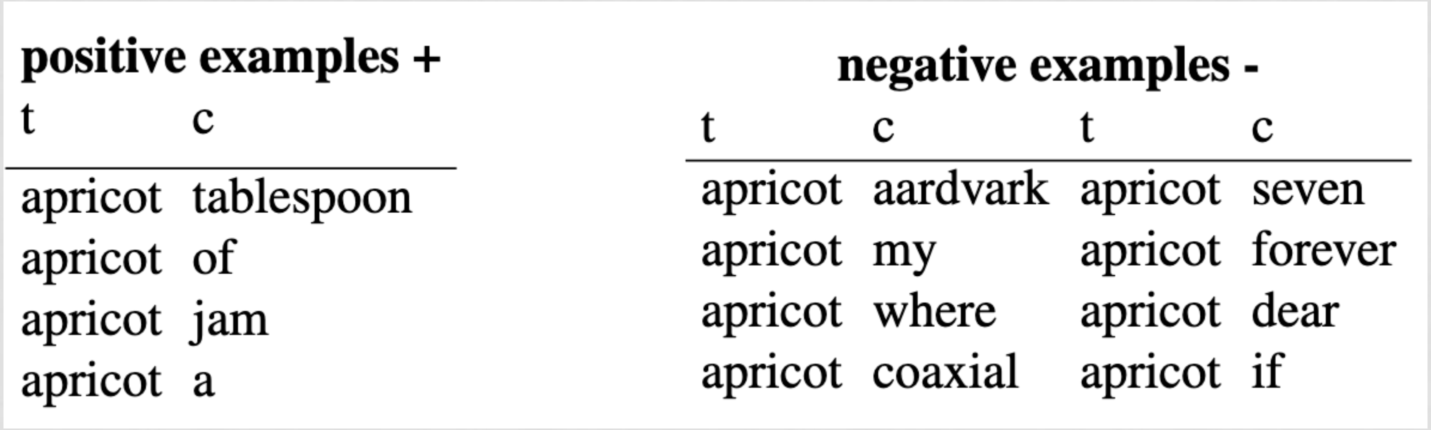
- It is a supervised learning problem on unlabeled data (running text). This is called self-supervision

TRAINING WORD2VEC



```
... lemon,  a [tablespoon of apricot jam,      a] pinch ...
               c1         c2    t      c3       c4
```

- This example has a target word *t* (apricot), and 4 context words in the *L* = ±2 window, resulting in 4 positive training instances
- Negative examples are randomly generated:



positive examples +

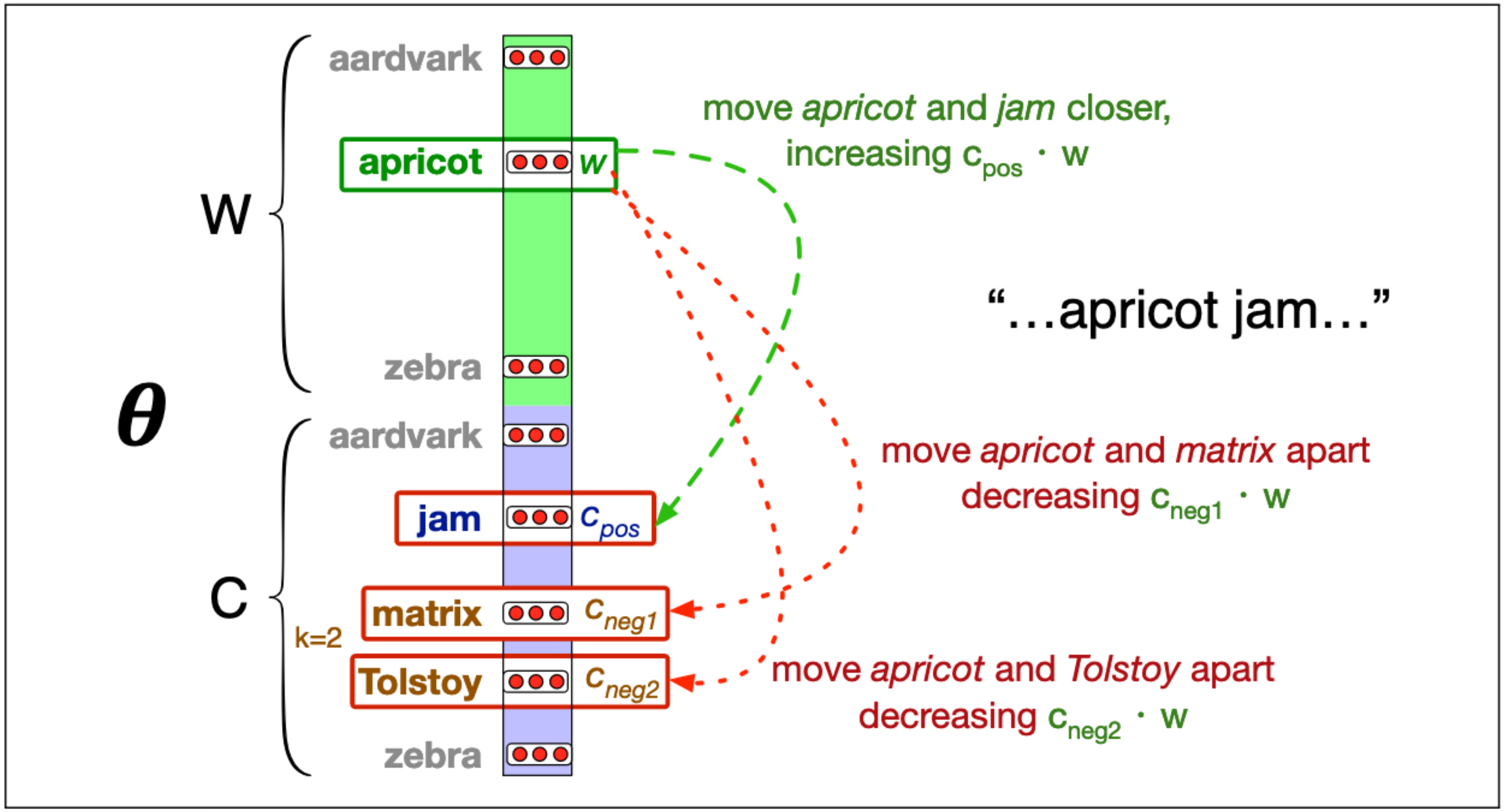
t	c
apricot	tablespoon
apricot	of
apricot	jam
apricot	a

negative examples -

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

TRAINING WORD2VEC

- The weights on the nodes in the hidden layer get random initializations
- The goal of the learning algorithm is to adjust those embeddings to
  - maximize the similarity of the target word, context word pairs (w,c_{pos}) drawn from the positive examples
  - minimize the similarity of the (w,c_{neg}) pairs from the negative examples
- The weights get updated while the model processes the collection
  - maximize the dot product of the word with the actual context words, and minimize the dot products of the word with the negative sampled non-context words (Gradient descent optimization)



aardvark
apricot
w
W
zebra
θ
aardvark
jam
c_{pos}
C
k=2
matrix
c_{neg1}
Tolstoy
c_{neg2}
zebra
move *apricot* and *jam* closer, increasing $c_{pos} \cdot w$
"…apricot jam…"
move *apricot* and *matrix* apart decreasing $c_{neg1} \cdot w$
move *apricot* and *Tolstoy* apart decreasing $c_{neg2} \cdot w$

Figure 6.14 Intuition of one step of gradient descent. The skip-gram model tries to shift embeddings so the target embeddings (here for *apricot*) are closer to (have a higher dot product with) context embeddings for nearby words (here *jam*) and further from (lower dot product with) context embeddings for noise words that don't occur nearby (here *Tolstoy* and *matrix*).

TRAINING WORD2VEC

- Summary:
  - Move through the training corpus with a sliding window. Each word + context is a classification problem
  - The outcome of the classification determines whether we adjust the current word vector. Gradually, vectors converge to (hopefully) optimal values
  - It is important that the context classification here is not an aim in itself: it is just a proxy to learn vector representations good for other tasks

ADVANTAGES OF WORD2VEC

- It scales
  - Can be trained on billion word corpora
  - In limited time
  - Possibility of parallel training
- Pre-trained word embeddings trained by one can be used by others
  - For entirely different tasks
- Incremental training
  - Train on one piece of data, save results, continue training later on

DO IT YOURSELF

- Implementation in Python package gensim

```
import gensim

model = gensim.models.Word2Vec(sentences, size=100,
                               window=5, min_count=5,
                               workers=4)
```

size: the dimensionality of the feature vectors (common: 200 or 320)

window: the maximum distance between the current and predicted word within a sentence

min_count: minimum number of occurrences of a word in the corpus to be included in the model

workers: for parallellization with multicore machines

DO IT YOURSELF

model.most_similar(‘apple’)

- [(’banana’, 0.8571481704711914), ...]

model.doesnt_match("breakfast cereal dinner lunch".split())

- ‘cereal’

model.similarity(‘woman’, ‘man’)

- 0.73723527

WHAT CAN YOU DO WITH IT?

- Mining knowledge about natural language
- Improve NLP applications
- Improve Text Mining applications

WHAT CAN YOU DO WITH IT?

- Mining knowledge about natural language
  - Learning semantic and semantic relations

WHAT CAN YOU DO WITH IT?

- A is to B as C is to ?
- This is the famous example:

  vector(king) – vector(man) + vector(woman) =

WHAT CAN YOU DO WITH IT?

- A is to B as C is to ?
- This is the famous example:

  vector(king) – vector(man) + vector(woman) = vector(queen)

  - Actually, what the original paper says is: if you substract the vector for ‘man’ from the one for ‘king’ and add the vector for ‘woman’, the vector closest to the one you end up with turns out to be the one for ‘queen’
  - More interesting:

    France is to Paris as Germany is to …

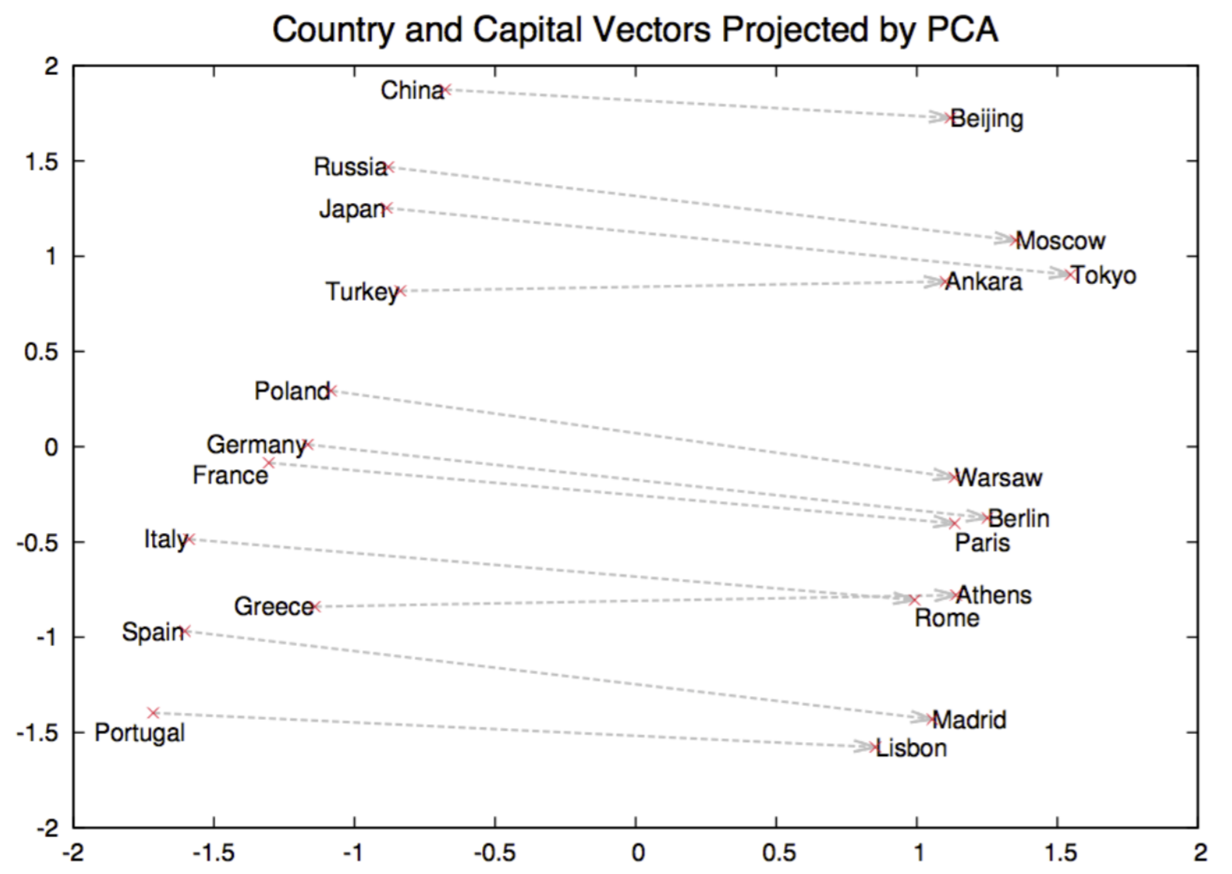


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Figure

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems, pages 3111–3119, 2013.

WHAT CAN YOU DO WITH IT?

- A is to B as C is to ?
- It also works for syntactic relations:
  - vector(biggest) - vector(big) + vector(small) =

WHAT CAN YOU DO WITH IT?

- A is to B as C is to ?
- It also works for syntactic relations:
  - vector(biggest) - vector(big) + vector(small) = vector(smallest)

WHAT CAN YOU DO WITH IT?

- Improve Text Mining applications:
  - (Near-)Synonym detection (→ query expansion)
  - Richer word and context representations for Named Entity Recognition
  - Document similarity
    - Example: cluster news articles per news event
  - Finding word associations / word clusters
  - Visualize word meanings for a domain collection

CLASSIFIERS WITH WORD EMBEDDINGS

- Neural Networks for text data take word embeddings as low-level representation of words
  - Word embeddings as input for convolutional neural networks in text categorization
  - Word embeddings as input for recurrent neural networks in sequence labelling

- Since 2018: word embeddings are used as language models that can be fine-tuned towards any natural language processing task (transfer learning, lecture 7)

CONCLUSIONS

Universiteit Leiden

HOMEWORK

- Read:
  - Jurafsky & Martin chapter 6. Vector semantics and embeddings (not: 6.7)
  - Jurafsky & Martin sections 7.1, 7.2, 7.3. Units, the XOR problem, Feedforward Neural Networks

- Exercise week 3: word2vec tutorial
  https://www.guru99.com/word-embedding-word2vec.html

AFTER THIS LECTURE…

- You can explain the role of the hidden layer in feedforward neural networks
- You can define the distributional hypothesis and its role in word embeddings
- You can define word embeddings
- You can explain how a word2vec model is trained
- You can describe 2 applications of word embeddings in text mining tasks
- You can train and use a word2vec model yourself