Deep Learning for NLP

The lecture starts at 13:15

Florina Piroi

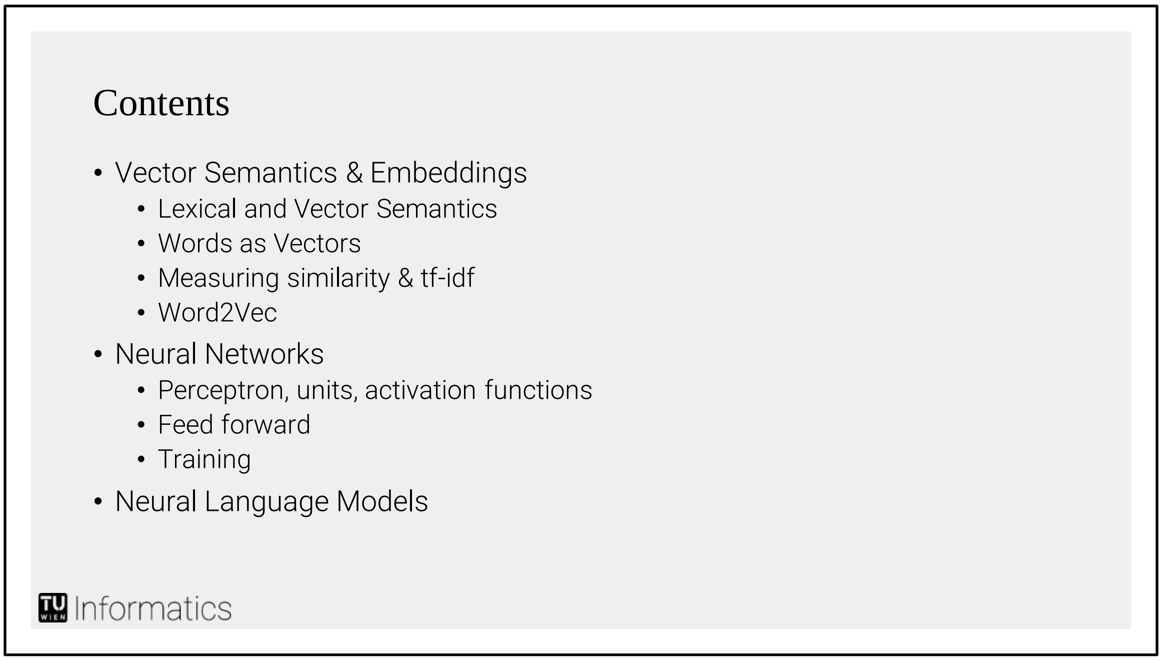
TU WIEN Informatics

Relevant Literature

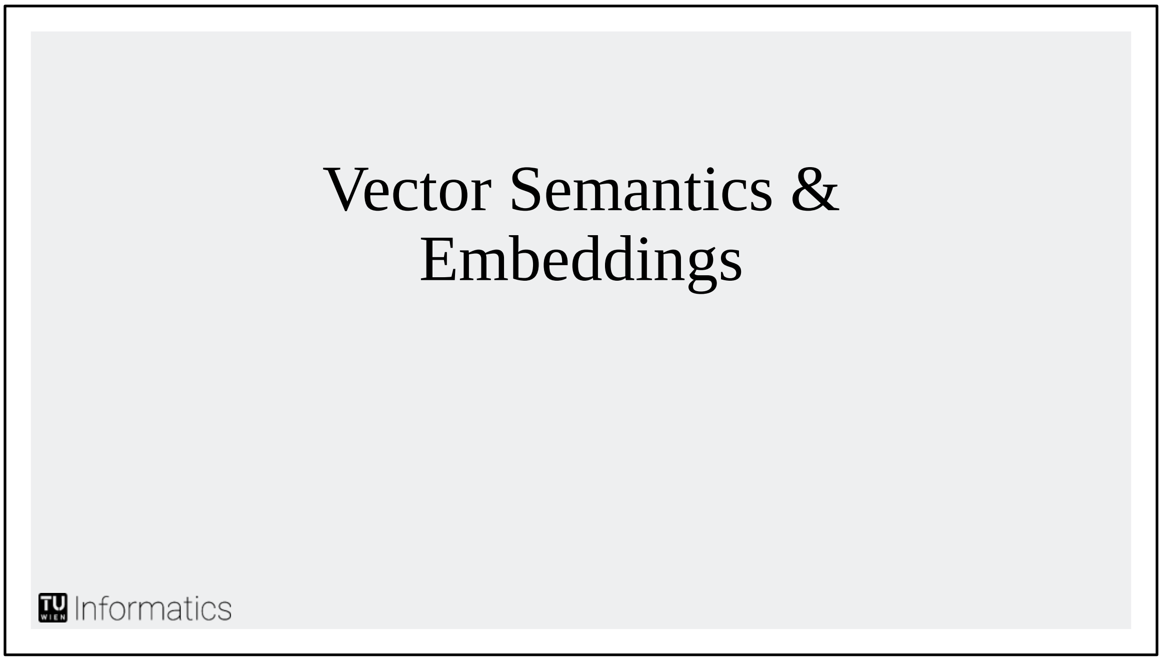
- Jurafsky & Martin, SLP, 3rd Edition: Chapters 6, 7
  - (including slides), references therein
- M. Nielsen, Neural Networks and Deep Learning, 2019

TU WIEN Informatics

https://web.stanford.edu/~jurafsky/slp3/

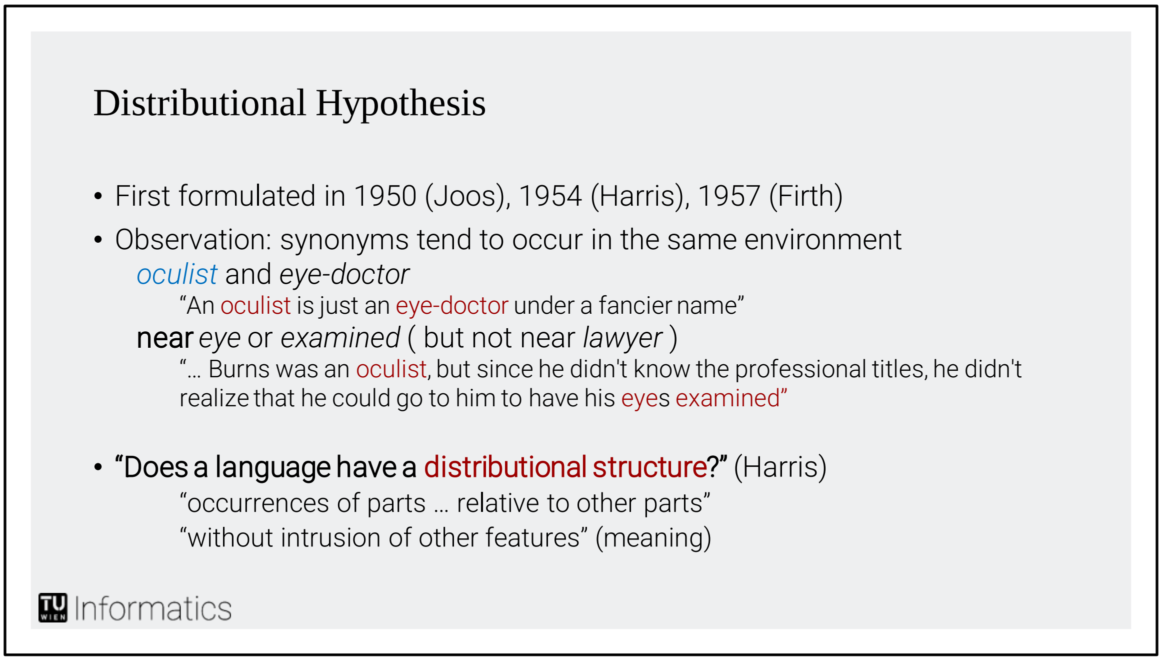


Contents



Vector Semantics & Embeddings

Informatics



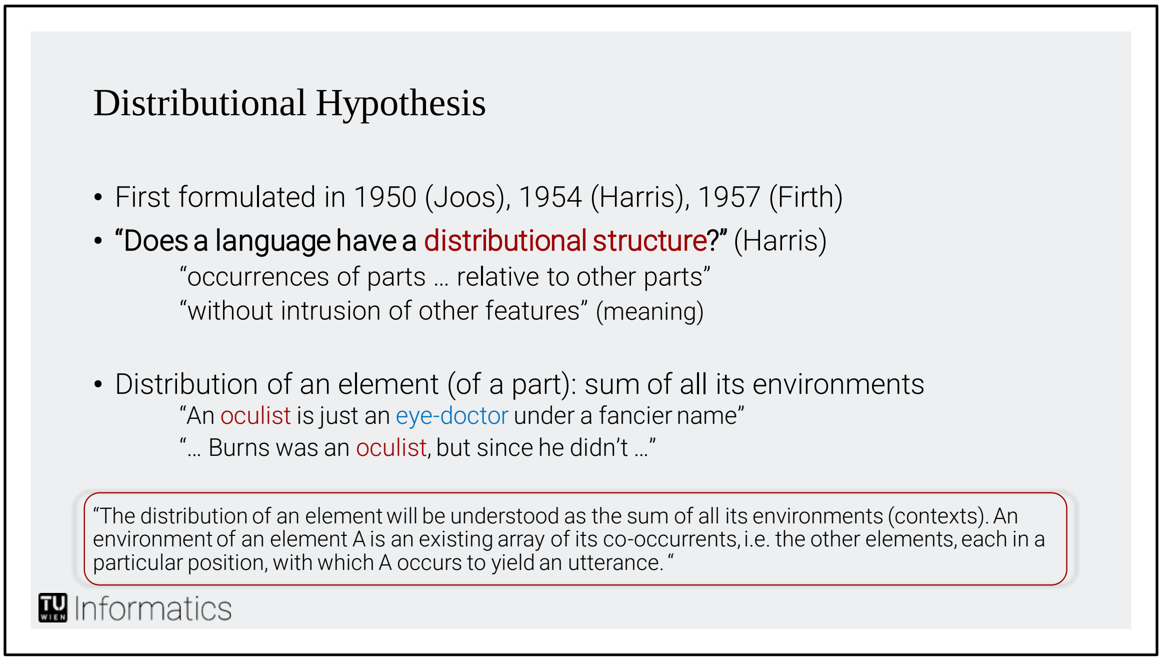
Distributional Hypothesis

- First formulated in 1950 (Joos), 1954 (Harris), 1957 (Firth)
- Observation: synonyms tend to occur in the same environment
  - *oculist* and *eye-doctor*
    - "An oculist is just an eye-doctor under a fancier name"
  - near *eye* or *examined* (but not near *lawyer*)
    - "… Burns was an oculist, but since he didn't know the professional titles, he didn't realize that he could go to him to have his eyes examined"
- "Does a language have a distributional structure?" (Harris)
  - "occurrences of parts … relative to other parts"
  - "without intrusion of other features" (meaning)

Informatics

The question was raised: can we describe languages in terms of how the words are distributed in the corpus? In terms of a distributional structure? More concretely: Can we describe a language by looking at occurrences of parts relative to other parts? „parts", here are linguistic elements (sounds ultimately).

Is this description complete? Without intrusion from other features such as meaning? I.e. what happens if I substitute *lawyer* instead of *oculist*?



Distributional Hypothesis

- First formulated in 1950 (Joos), 1954 (Harris), 1957 (Firth)
- “Does a language have a distributional structure?” (Harris)

  “occurrences of parts … relative to other parts”

  “without intrusion of other features” (meaning)

- Distribution of an element (of a part): sum of all its environments

  “An oculist is just an eye-doctor under a fancier name”

  “… Burns was an oculist, but since he didn’t …”

“The distribution of an element will be understood as the sum of all its environments (contexts). An environment of an element A is an existing array of its co-occurrents, i.e. the other elements, each in a particular position, with which A occurs to yield an utterance. “

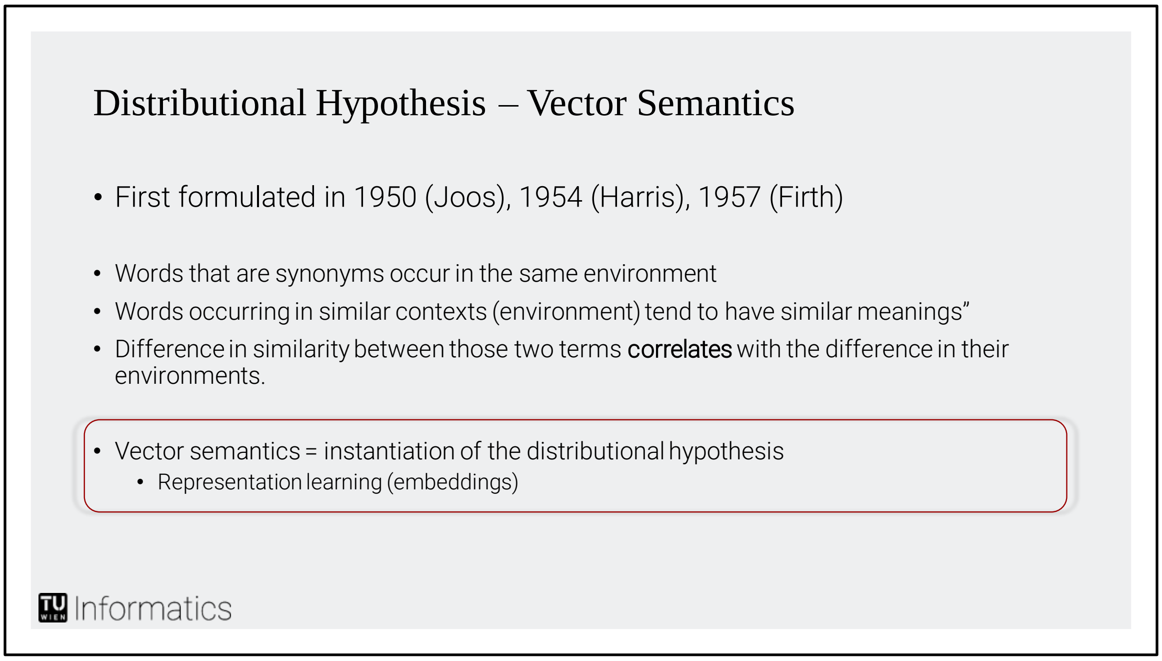
TU WIEN Informatics

Distributional Hypothesis

- First formulated in 1950 (Joos), 1954 (Harris), 1957 (Firth)
- "Does a language have a distributional structure?" (Harris)
  - "occurrences of parts … relative to other parts"
  - "without intrusion of other features" (meaning)

- Words that are synonyms occur in the same environment
- Words occurring in similar contexts (environment) tend to have similar meanings"
- Difference in similarity between those two terms **correlates** with the difference in their environments.

Informatics



Distributional Hypothesis – Vector Semantics

- First formulated in 1950 (Joos), 1954 (Harris), 1957 (Firth)

- Words that are synonyms occur in the same environment
- Words occurring in similar contexts (environment) tend to have similar meanings”
- Difference in similarity between those two terms **correlates** with the difference in their environments.

- Vector semantics = instantiation of the distributional hypothesis
  - Representation learning (embeddings)

Informatics

And how does the Vector Semantics instantiate the distributional hypothesis? by learning representations of the MEANING of the words, representations which are nowadays called ***embeddings***. The representations are learned directly from their distributions in the text.

These representations underlie the powerful **contextualized word representations** (ELMo, BERT) which we will be talking about in a few weeks.

Lexical Semantics

Q: How to represent the meaning of a word?

- N-Gram: string of letters/characters
- Index in a vocabulary list
- …

But:

- *cold* vs. *hot*
- *happy* vs. *sad*

The trophy doesn't fit into the brown suitcase because it's too small.

-> Model of the meaning

TU WIEN Informatics

Lexical Semantics

-> Model of the **meaning** – why do we need one?

The trophy doesn't fit into the brown suitcase because it's too small.

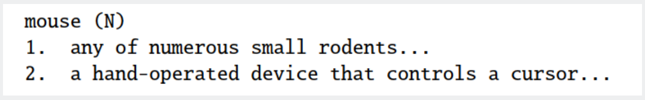
Draw useful inferences to help us solve meaning-related tasks:

- Q&A
- Plagiarism & paraphrasing
- Dialogue
- Summarization

Lexical semantics refers to the linguistic study of the word meaning

Lexical Semantics - Lemma and Senses

Lemma == dictionary form == citation form

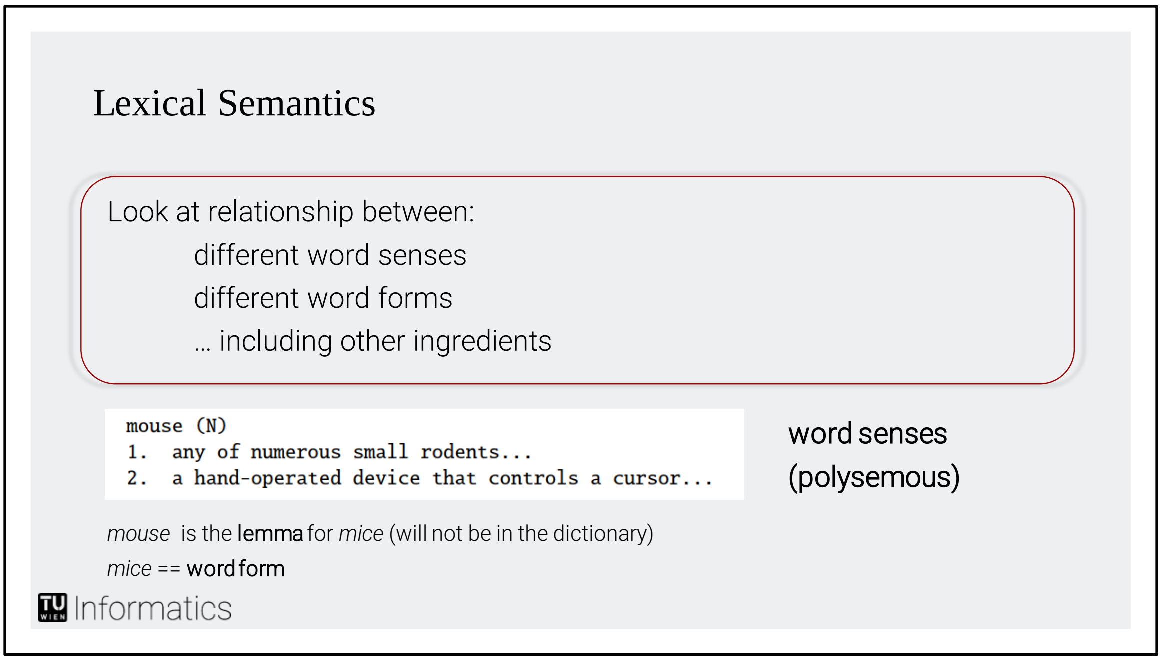


```
mouse (N)
1.  any of numerous small rodents...
2.  a hand-operated device that controls a cursor...
```

word senses
(polysemous)

mouse is the **lemma** for *mice* (will not be in the dictionary)

mice == **word form**



Lexical Semantics

Look at relationship between:
- different word senses
- different word forms
- … including other ingredients

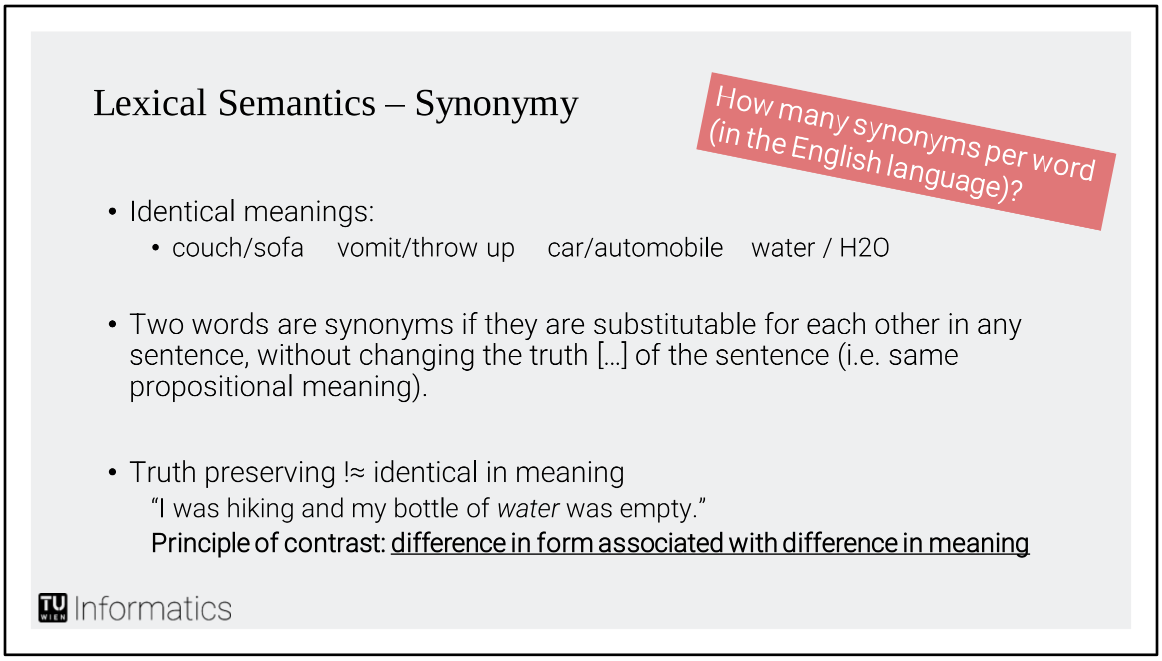
```
mouse (N)
1.  any of numerous small rodents...
2.  a hand-operated device that controls a cursor...
```

word senses
(polysemous)

mouse is the **lemma** for *mice* (will not be in the dictionary)

mice == **word form**

Informatics


Lexical Semantics – Synonymy

How many synonyms per word (in the English language)?

- Identical meanings:
  - couch/sofa vomit/throw up car/automobile water / H2O
- Two words are synonyms if they are substitutable for each other in any sentence, without changing the truth [...] of the sentence (i.e. same propositional meaning).
- Truth preserving !≈ identical in meaning

  "I was hiking and my bottle of *water* was empty."

  Principle of contrast: difference in form associated with difference in meaning

Informatics

Truth preserving is not really the same with identical in meaning. Indeed, probably no two words are absolutely identical in meaning.

For example "H2O" is used in a scientific context, but in this sentence "water" is more appropriate.

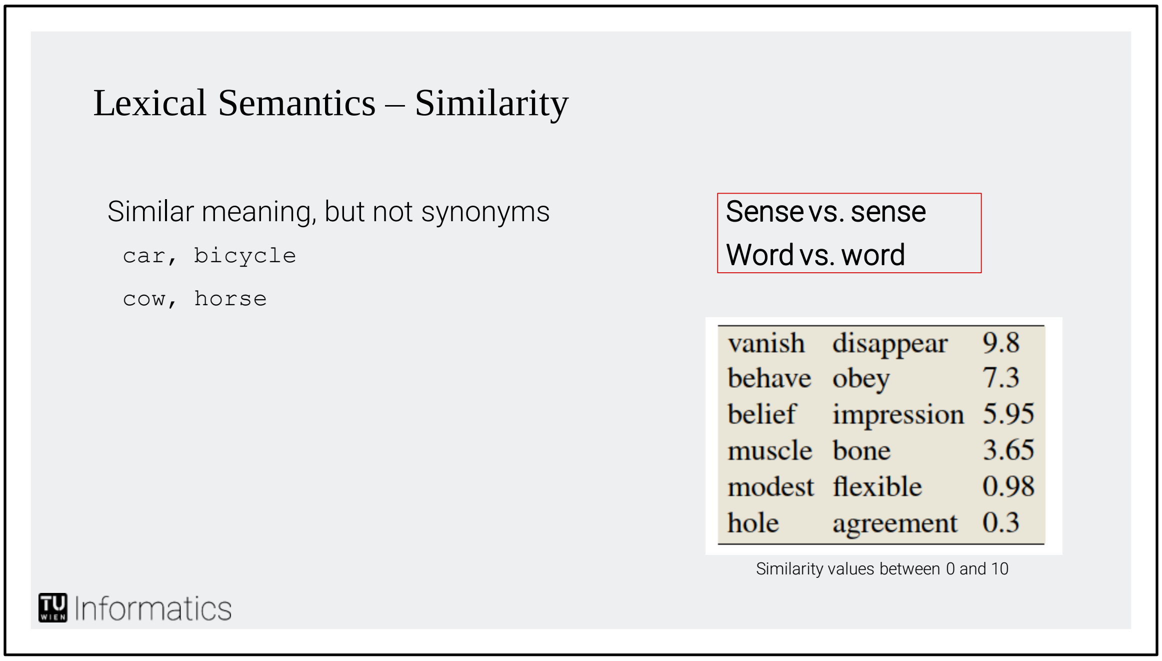
One of the fundamental tenets in semantics is the principle of contrast, which assumes that a difference in the linguistic form is **always** associated with some difference in meaning. The difference between H2O and water is also part of the meaning of the words.

Lexical Semantics – Antonymity

- Opposite senses, with respect to one feature of their meaning:
    - up / down   hot / cold   in / out

- Can define binary opposition
- Can be at the opposite ends of a scala (long / short)
- Can be reversives (rise / fall)

TU WIEN Informatics



Lexical Semantics – Similarity

Similar meaning, but not synonyms

`car, bicycle`

`cow, horse`

Sense vs. sense
Word vs. word

vanish	disappear	9.8
behave	obey	7.3
belief	impression	5.95
muscle	bone	3.65
modest	flexible	0.98
hole	agreement	0.3

Similarity values between 0 and 10

Informatics

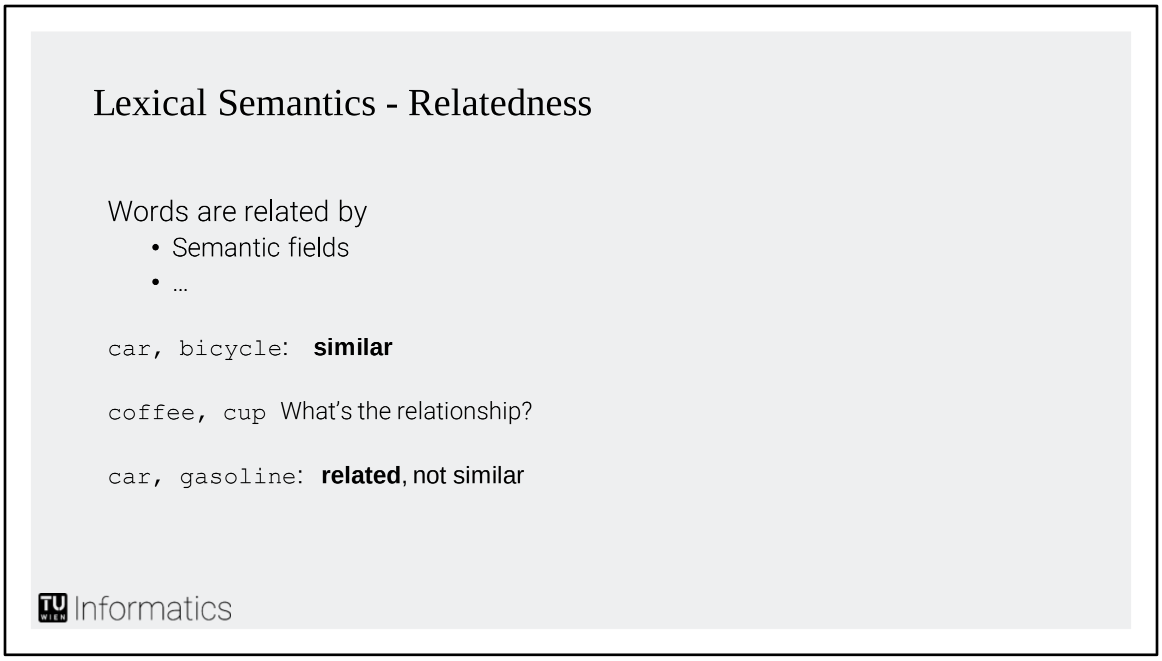
Words don't have many synonyms, but have lots of similar words!

Words with similar meanings are not synonyms, but share some element of meaning.

Cow and horse are not synonyms, but are similar. And here we move from relationships between word senses, to relations between words, and similarity is such a relationship.

The notion of word similarity is very useful in larger semantic tasks. Knowing how similar two words are can help in computing how similar the meaning of two phrases or sentences are, a very important component of natural language understanding tasks like question answering, paraphrasing, and summarization.

But how can we get these values for similarity? One way is to ask humans.
In this example, values between 0 and 10 were given to pairs of words, and you observe the scores of near synonyms, vs. scores of words that hardly have something in common (last line)



Lexical Semantics - Relatedness

Words are related by
- Semantic fields
- …

`car, bicycle`: **similar**

`coffee, cup` What's the relationship?

`car, gasoline`: **related**, not similar

Informatics

We can relate the meaning of words in other ways than similarity.

Word relatedness, also called "word association" is a way to look at the relationship between two words not by similarity.

Consider the meanings of the words coffee and cup. Coffee is not similar to cup; they share practically no features (coffee is a plant or a beverage, while a cup is a manufactured object with a particular shape). But coffee and cup are clearly related; they are associated by co-participating in an everyday event (the event of drinking coffee out of a cup).

Lexical Semantics - Relatedness

Words are related by

- Semantic fields (surgeon, scalpel, nurse, anaesthetic, hospital)
- … (linguistic frameworks)

`car, bicycle`: **similar**

`coffee, cup` What's the relationship?

`car, gasoline`: **related**, not similar

-> topic models: Latent Dirichlet Allocation, LDA – unsupervised learning on large set of texts, induce sets of associated words.

TU WIEN Informatics

Lexical Semantics – Superordinate / Subordinate

One sense is a **subordinate** of the other:
the first is more specific (subclass of the other)

- `car` subordinate of `vehicle`.
- `vehicle` **superordinate** of `car`.

TU WIEN Informatics

Lexical Semantics – Connotations

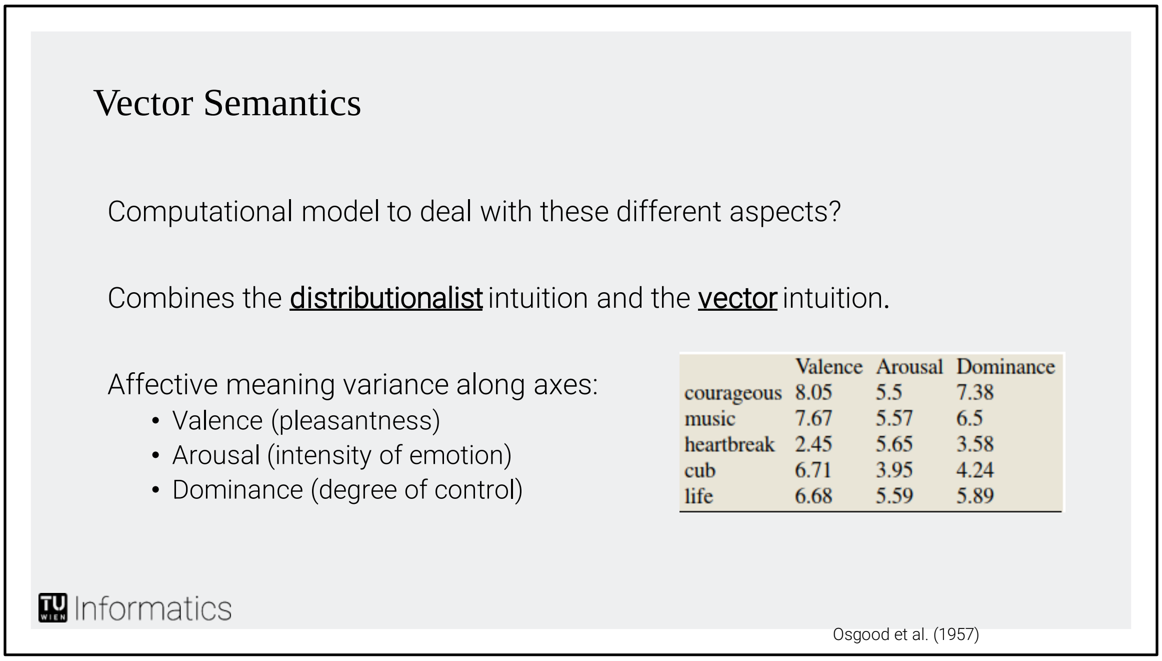
Words have affective meanings

- positive connotations (happy)
- negative connotations (sad)

positive evaluation (great, love)
negative evaluation (terrible, hate).

But: “terribly good!”

TU WIEN Informatics



Vector Semantics

Computational model to deal with these different aspects?

Combines the distributionalist intuition and the vector intuition.

Affective meaning variance along axes:
- Valence (pleasantness)
- Arousal (intensity of emotion)
- Dominance (degree of control)

	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24
life	6.68	5.59	5.89

TU WIEN Informatics

Osgood et al. (1957)

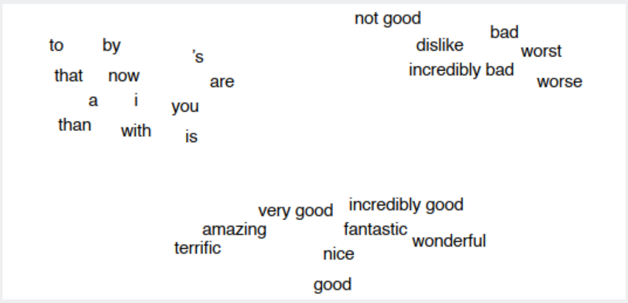
In connection with work on word connotations, Osgood and some colleagues looked at how words varied along three axes, and they attached values to words on each of these axes. Osgood et al. (1957) noticed that in using these 3 numbers to represent the meaning of a word, the model he created was representing each word as a point in a three-dimensional space. This revolutionary idea that word meaning word could be represented as a point in space was the first expression of the vector semantics models

Osgood, C. E., Suci, G. J., & Tannenbaum, P. H. (1957). *The measurement of meaning.* Univer. Illinois Press
https://gwern.net/doc/psychology/1957-osgood-themeasurementofmeaning.pdf

Vector Semantics

- Define words as vectors
- “embedding” – embedded into a (multi-dimentional) space

- Standard in NLP



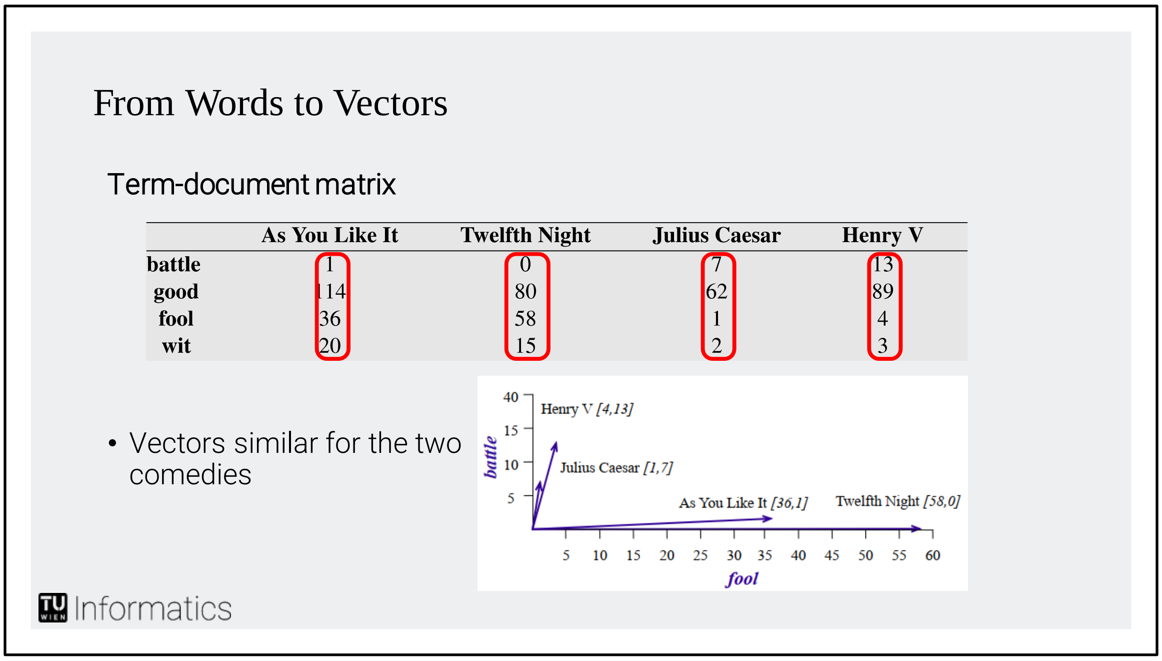
Informatics

Types of Embeddings

- TF-IDF
  - Common baseline
  - Sparse vectors
  - Words as function of counts

- Word2vec
  - Dense vectors
  - Representations distinguish between near/far words.

TU WIEN Informatics

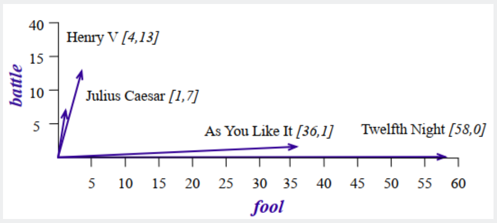


From Words to Vectors

Term-document matrix

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- Vectors similar for the two comedies



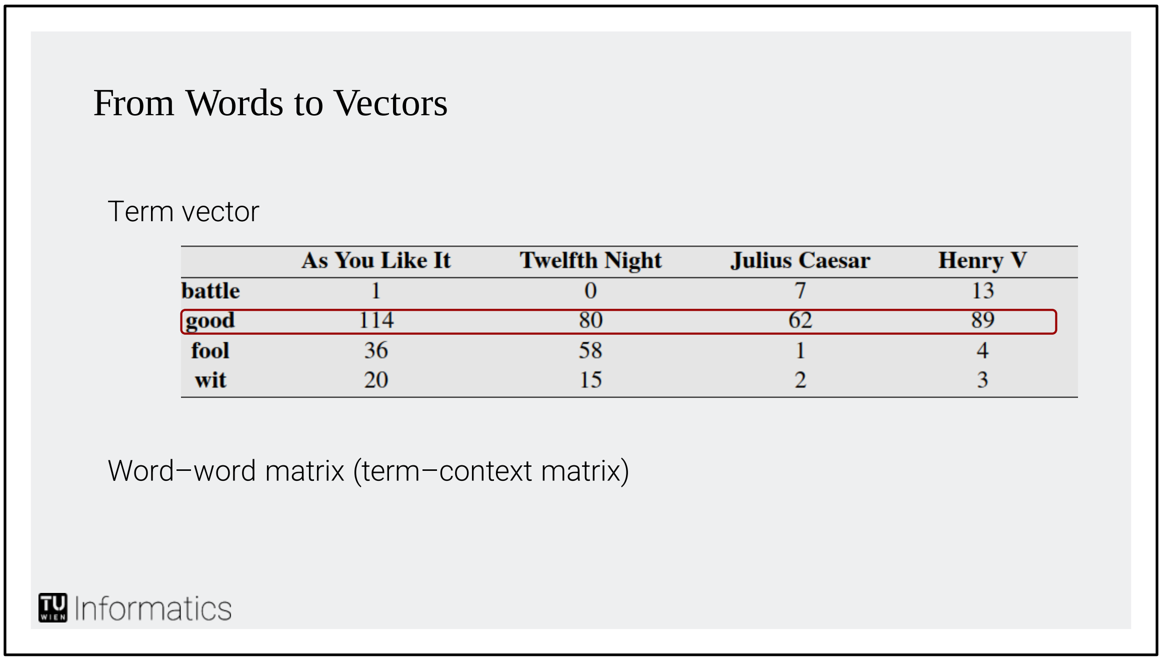
Informatics

Vector or distributional models of meaning are generally based on a co-occurrence matrix, a way of representing how often words co-occur. Such co-occurrences matrices can be constructed in different ways, we look here at the term-document matrix.

Take 4 Shakespeare plays, and to illustrate the process we pick these words occurring in their texts. For each term we count how often it occurs in one document. We obtain, thus, four vectors, circled in red, one for each play. In reality, such a term-doc matrix is much larger, and you know already that it has lots of We can think of the vector for a document as identifying a point in |V|-dimensional space; thus the documents in this table are points in 4-dimensional space.

Term-document matrices were originally defined as a means of finding similar documents for the task of document information retrieval. Two documents that are similar will tend to have similar words, and if two documents have similar words their column vectors will tend to be similar.

The comedy and history document vectors are non-similar.



From Words to Vectors

Term vector

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

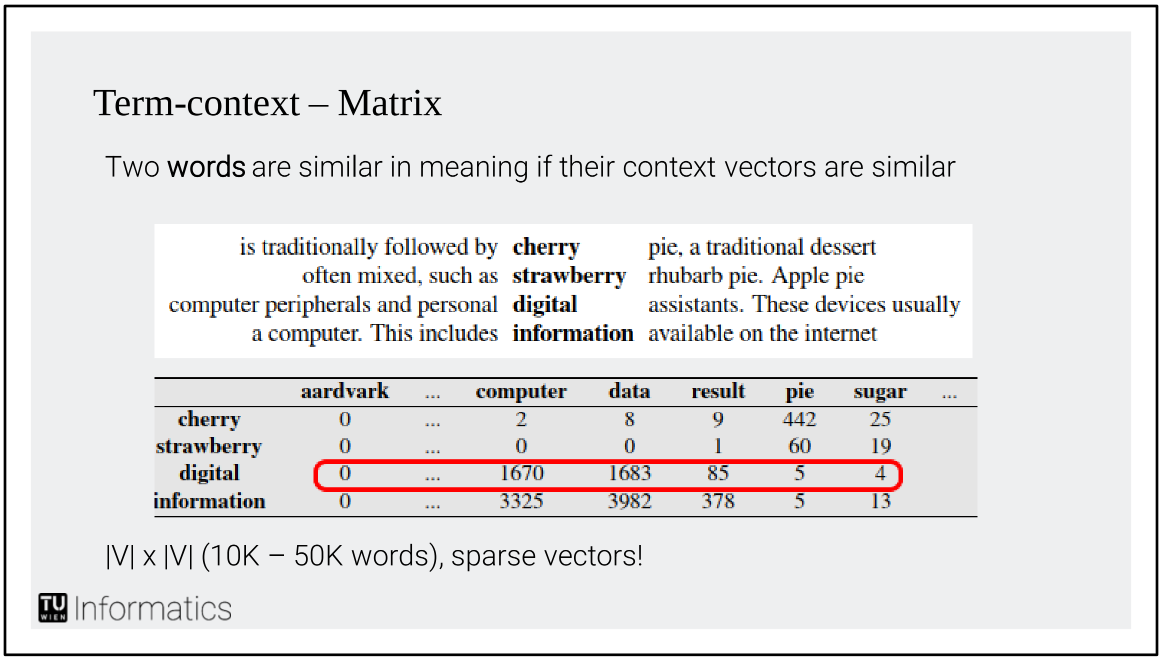
Word–word matrix (term–context matrix)

Informatics

But vector semantics can also be used to represent the meaning of words, by associating each word with a vector.

For documents, we saw that similar documents had similar vectors, because similar documents tend to have similar words. This same principle applies to words: similar words have similar vectors because they tend to occur in similar documents. The term-document matrix thus lets us represent the meaning of a word by the documents it tends to occur in

-



Term-context – Matrix

Two **words** are similar in meaning if their context vectors 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	

|V| x |V| (10K – 50K words), sparse vectors!

Informatics

More common are the word-word matrixes

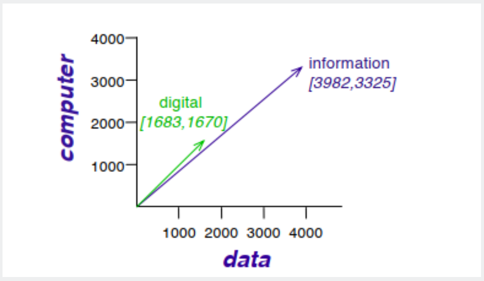
If we then take every occurrence of each word (say strawberry) and count the context words around it, we get a word-word co-occurrence matrix. Fig. shows a simplified subset of the word-word co-occurrence matrix for these four words computed from the Wikipedia corpus

Term-term matrix is in the size of the vocabulary of the corpus we are looking at.

Cosine for Similarity

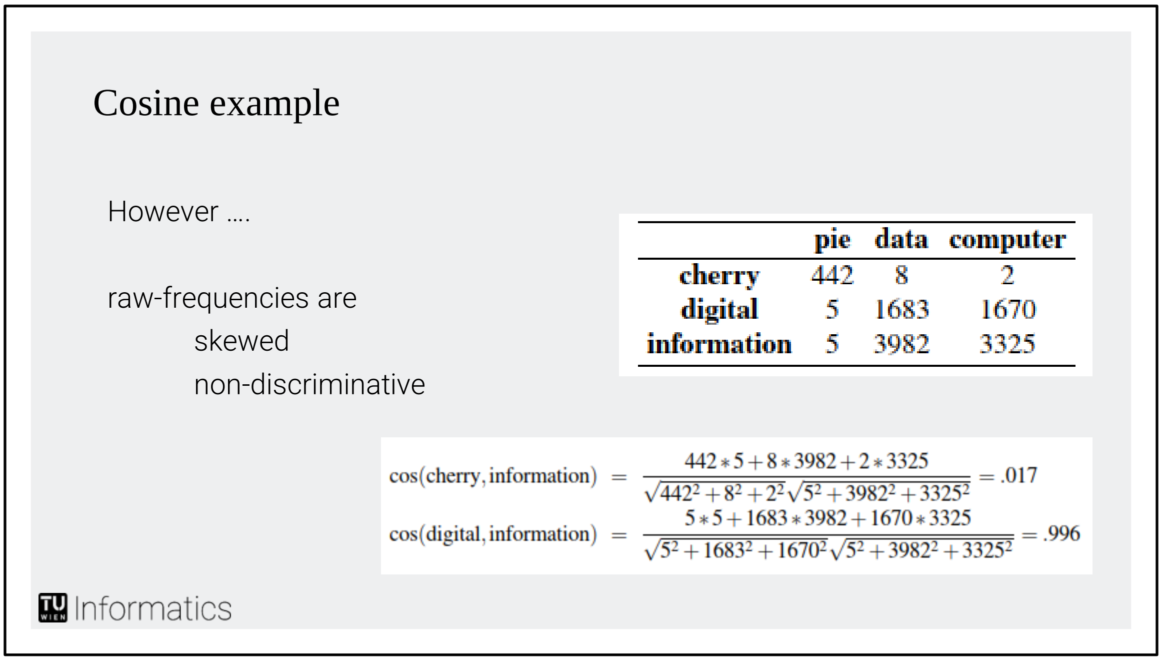
Measure the angle between vectors

$$\text{cosine}(\mathbf{v},\mathbf{w}) = \frac{\mathbf{v}\cdot\mathbf{w}}{|\mathbf{v}||\mathbf{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2}\sqrt{\sum_{i=1}^{N} w_i^2}}$$



	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	

Informatics



Cosine example

However ….

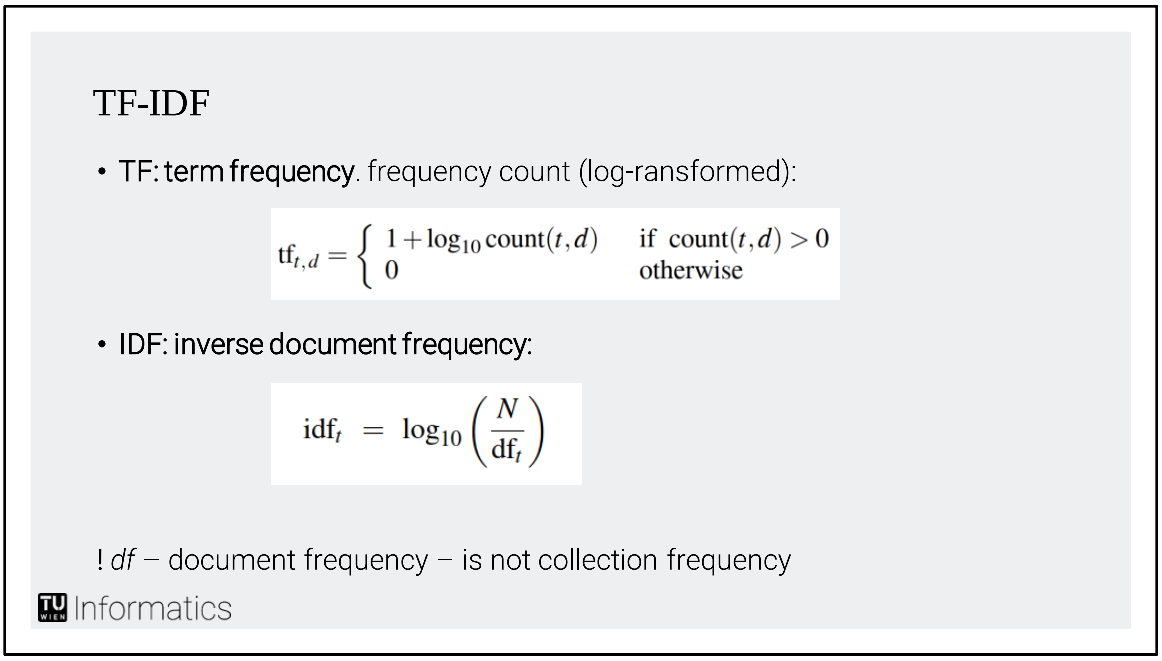
raw-frequencies are
- skewed
- non-discriminative

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325

$$\cos(\text{cherry}, \text{information}) = \frac{442 * 5 + 8 * 3982 + 2 * 3325}{\sqrt{442^2 + 8^2 + 2^2}\sqrt{5^2 + 3982^2 + 3325^2}} = .017$$

$$\cos(\text{digital}, \text{information}) = \frac{5 * 5 + 1683 * 3982 + 1670 * 3325}{\sqrt{5^2 + 1683^2 + 1670^2}\sqrt{5^2 + 3982^2 + 3325^2}} = .996$$

TU WIEN Informatics



TF-IDF

- TF: term frequency. frequency count (log-ransformed):

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10}\mathrm{count}(t,d) & \text{if count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- IDF: inverse document frequency:

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right)$$

! *df* – document frequency – is not collection frequency

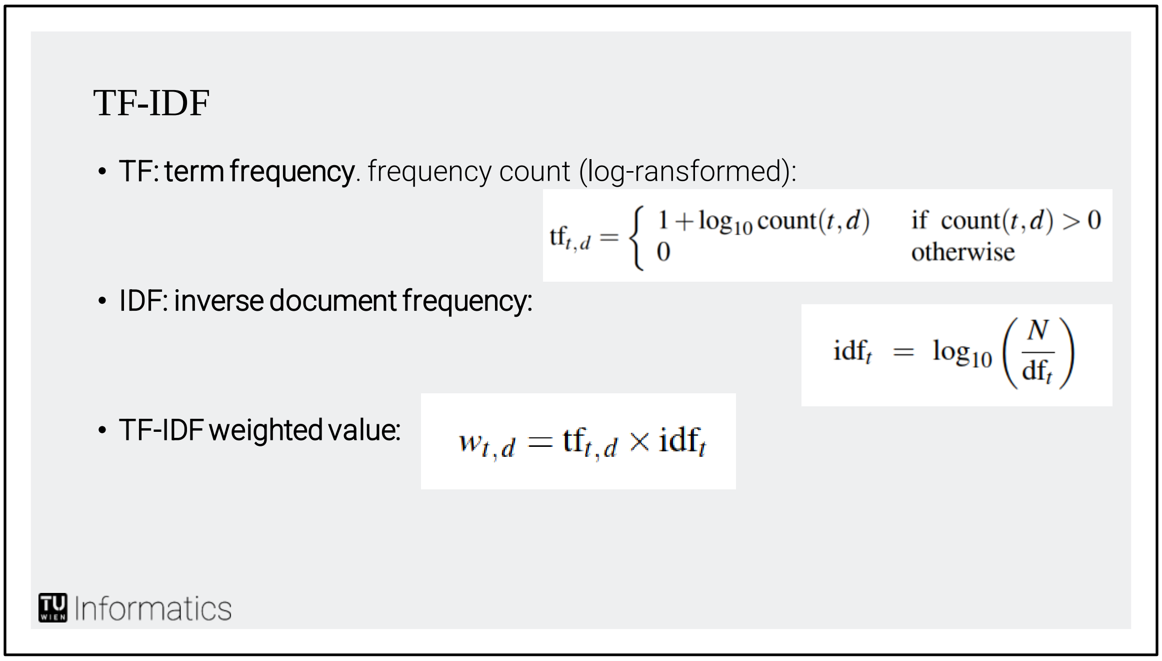
Informatics

TF-IDF combines two factors:

Term frequency – counts the frequency of the terms in a given document

And the Inverse Document Frequency, which looks at the number of documents in the collection where the word “i“ occurs. Here N is the number of documents in the collection, df is the document frequency of the word t

Log 10 - terms which occur 10 times in a document would have a tf=2, 100 times in a document tf=3, 1000 times tf=4, and so on. The intuition is that a word appearing 100 times in a document doesn’t make that word 100 times more likely to be relevant to the meaning of the document. Because we can’t take the log of 0, we normally add 1 to the count



TF-IDF

- TF: term frequency. frequency count (log-ransformed):

$$\mathrm{tf}_{t,d} = \begin{cases} 1+\log_{10}\mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

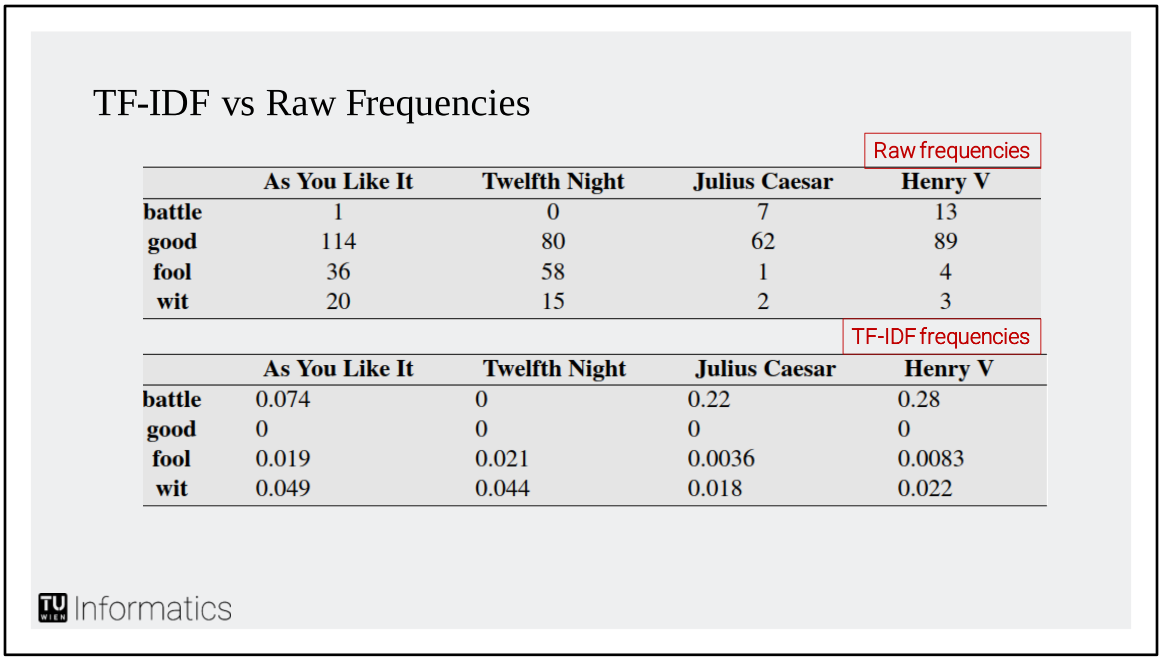
- IDF: inverse document frequency:

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right)$$

- TF-IDF weighted value:

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

TU WIEN Informatics



TF-IDF vs Raw Frequencies

Raw frequencies

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

TF-IDF frequencies

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

Informatics

Note that the tf-idf values for the dimension corresponding to the word good have now all become 0; since this word appears in every document, the tf-idf algorithm leads it to be ignored in any comparison of the plays.

So this is how MEANING is somehow modelled in the TF IDF encoding

Recap

- Vector Semantics & Embeddings
    - Lexical and Vector Semantics
    - Words as Vectors
    - Measuring similarity & tf-idf
        - Sparse
    - Word2Vec

Informatics

Dense Vectors – Word2Vec

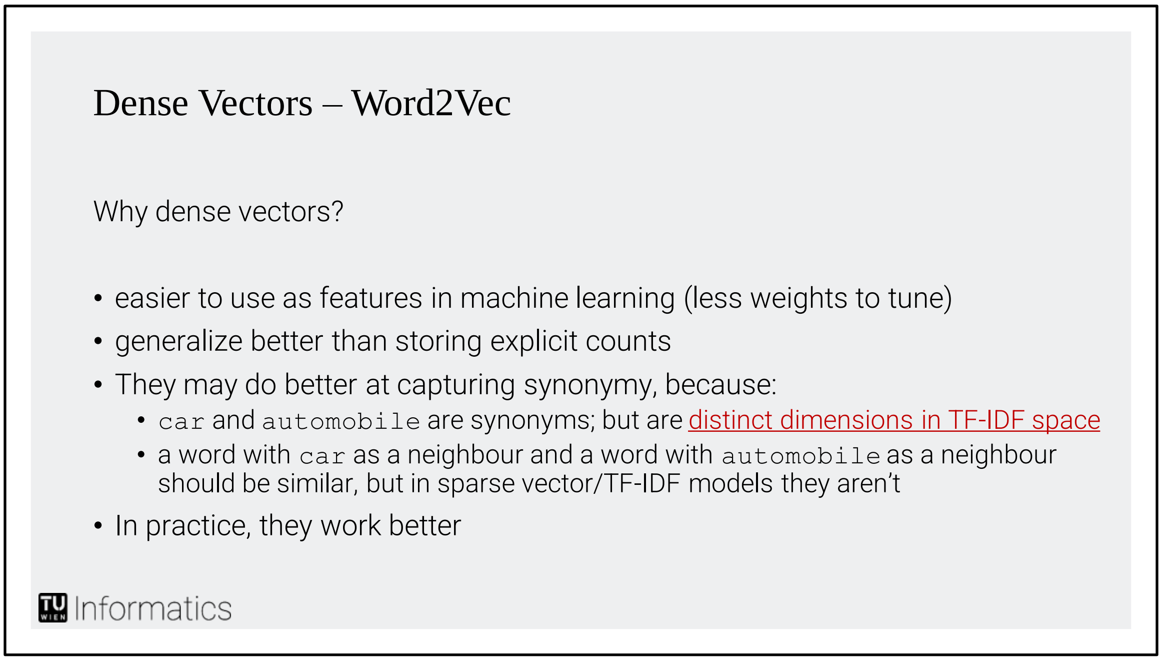
TF-IDF vectors are

- long (length |V|= 20,000 to 50,000)
- sparse (most elements are zero)

Want vectors which are

- short (length 50-1000)
- dense (most elements are non-zero)

Informatics



Dense Vectors – Word2Vec

Why dense vectors?

- easier to use as features in machine learning (less weights to tune)
- generalize better than storing explicit counts
- They may do better at capturing synonymy, because:
  - `car` and `automobile` are synonyms; but are distinct dimensions in TF-IDF space
  - a word with `car` as a neighbour and a word with `automobile` as a neighbour should be similar, but in sparse vector/TF-IDF models they aren't
- In practice, they work better

Informatics

In TF-IDF the relation between dimensions in the vector are not captured. In dense vectors, they are – although we don't really know why.

Where to look for Dense Embeddings

Word2vec (Mikolov et al.)
    https://code.google.com/archive/p/word2vec/

Fasttext
    http://www.fasttext.cc/

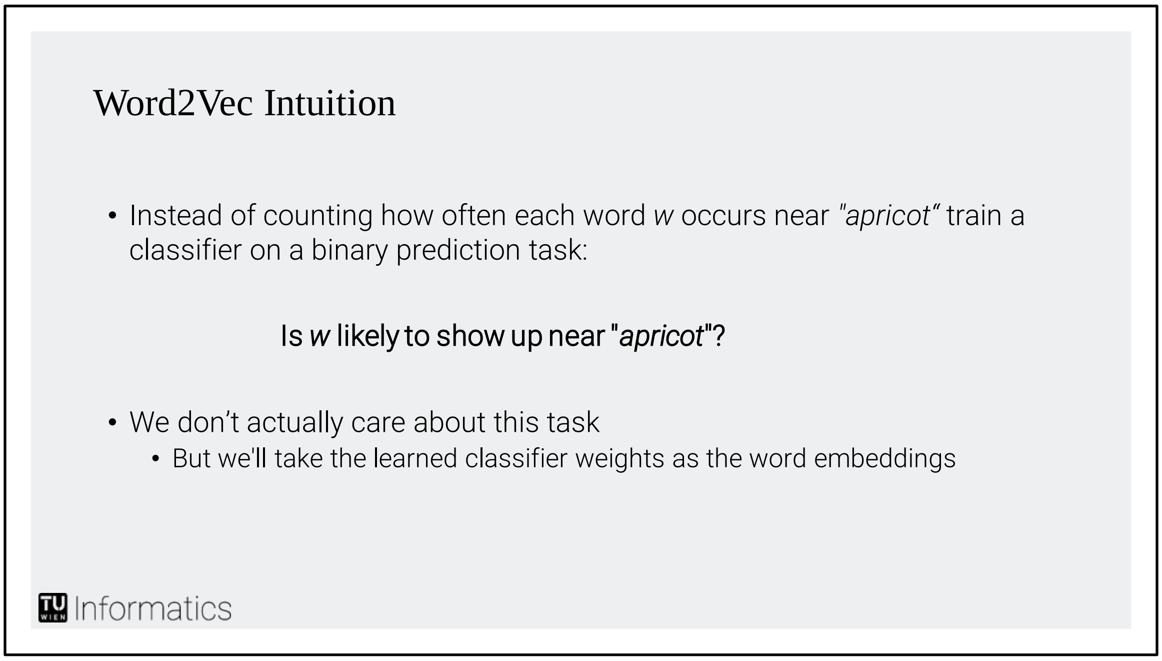
Glove (Pennington, Socher, Manning)
    http://nlp.stanford.edu/projects/glove/

TU WIEN Informatics

Word2Vec

- Popular embedding method
- Very fast to train
- Code available on the web

Idea: **predict** rather than **count**



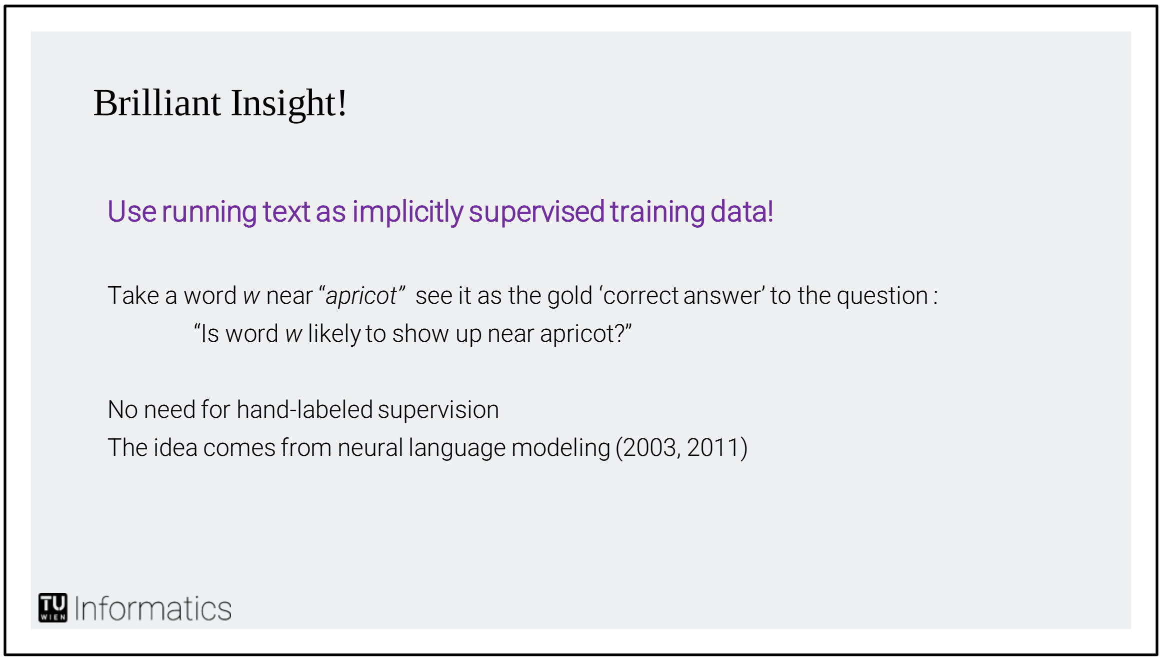
Word2Vec Intuition

- Instead of counting how often each word *w* occurs near *"apricot"* train a classifier on a binary prediction task:

Is *w* likely to show up near "*apricot*"?

- We don't actually care about this task
  - But we'll take the learned classifier weights as the word embeddings

Informatics



Brilliant Insight!

Use running text as implicitly supervised training data!

Take a word *w* near *"apricot"* see it as the gold 'correct answer' to the question :
"Is word *w* likely to show up near apricot?"

No need for hand-labeled supervision
The idea comes from neural language modeling (2003, 2011)

TU WIEN Informatics

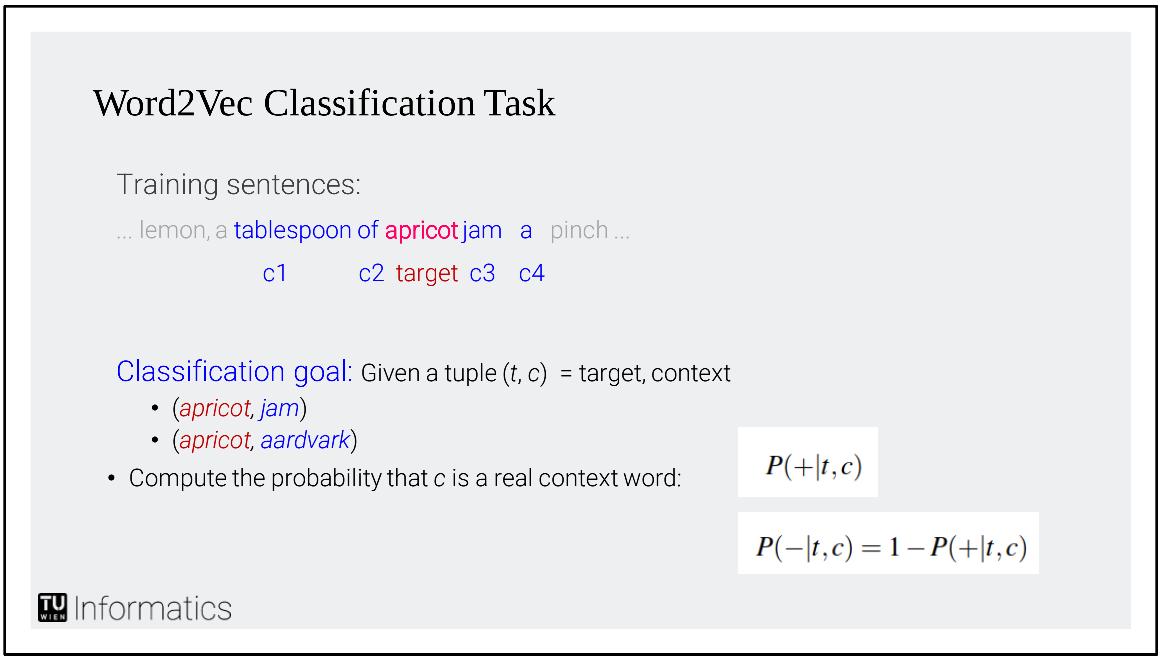
2003 – 2011
This idea was first proposed in the task of neural language modeling, when Bengio et al. (2003) and Collobert et al. (2011) showed that a neural language model (a neural network that learned to predict the next word from prior words) could just use the next word in running text as its supervision signal, and could be used to learn an embedding representation for each word as part of doing this prediction task.

Word2Vec
simplifies the prediction task by making it a binary classification task uses linear regression, instead of NNs – that would require sophisticated training algs.

Word2Vec: Skip-Gram

- "skip-gram with negative sampling" (SGNS)

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. Use logistic regression to train a classifier to distinguish those two cases
4. Use the weights as the embeddings



Word2Vec Classification Task

Training sentences:

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

c1 c2 target c3 c4

Classification goal: Given a tuple (t, c) = target, context

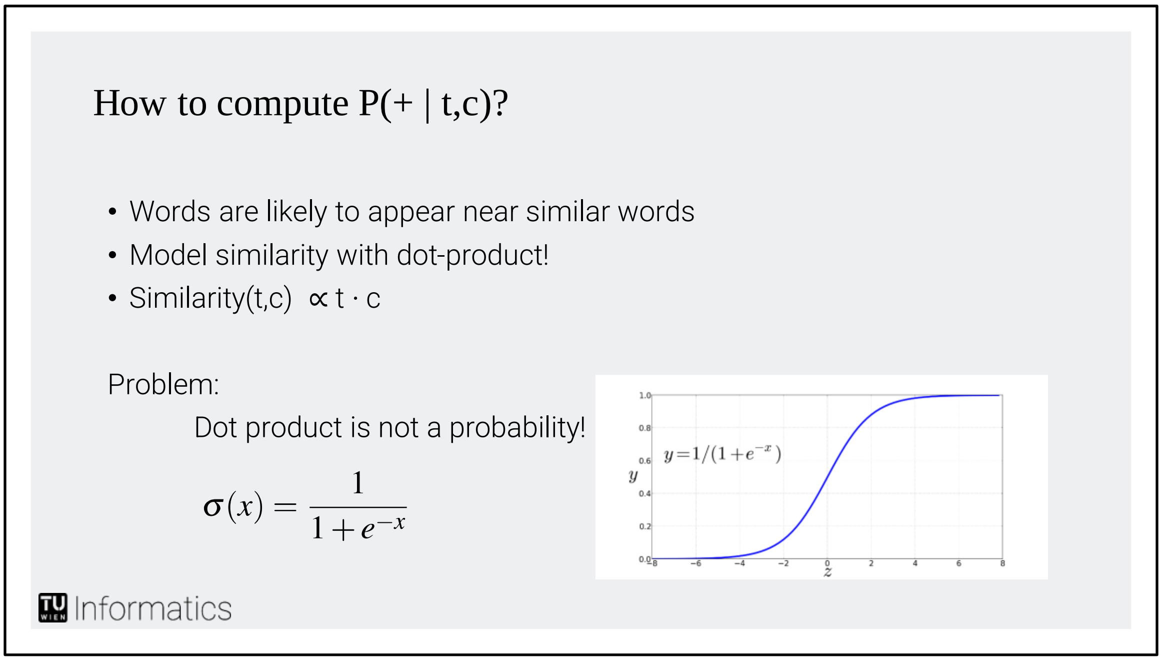
- (*apricot*, *jam*)
- (*apricot*, *aardvark*)

- Compute the probability that c is a real context word:

$$P(+|t,c)$$

$$P(-|t,c) = 1 - P(+|t,c)$$

Informatics



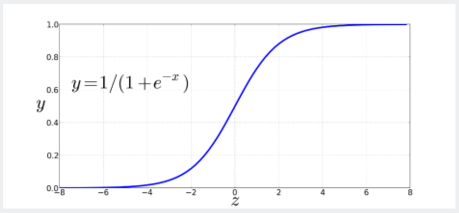
How to compute P(+ | t,c)?

- Words are likely to appear near similar words
- Model similarity with dot-product!
- Similarity(t,c) $\propto$ t · c

Problem:

Dot product is not a probability!

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

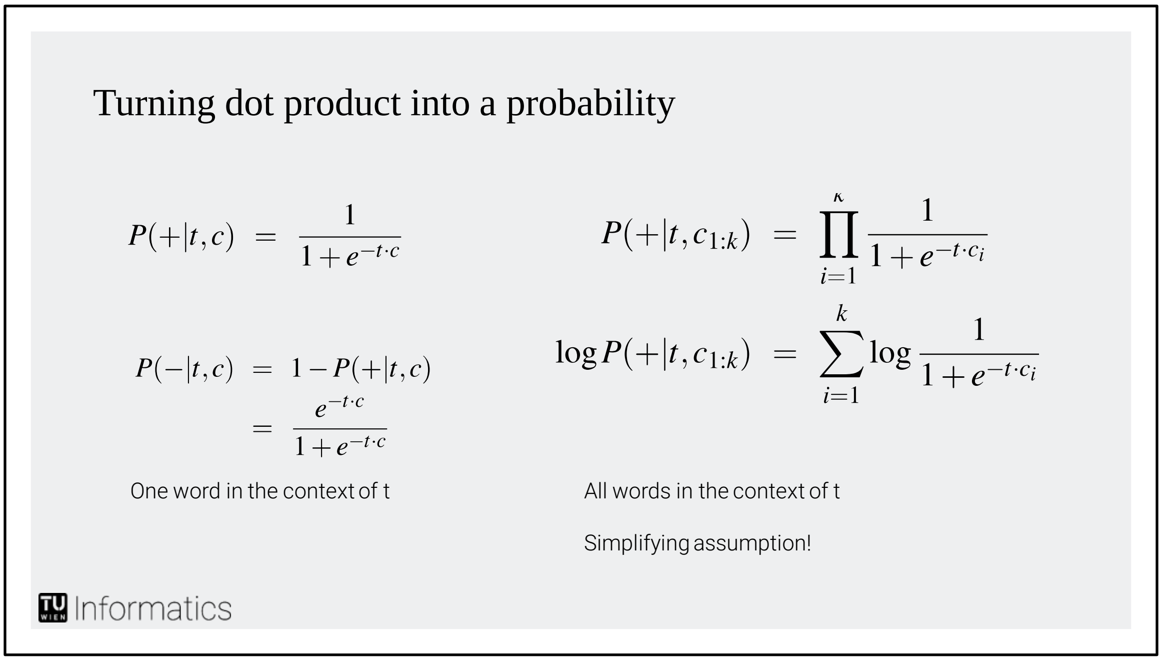


Informatics

How does the classifier compute the probability P? The intuition of the skip-gram model is to base this probability on similarity: a word is likely to occur near the target if its embedding is similar to the target embedding.

How can we compute similarity between embeddings? Recall that two vectors are similar if they have a high dot product (cosine, the most popular similarity metric, is just a normalized dot product). In other words: …

The logistic (sigmoid) function to squash the values between 0 and 1. The sigmoid function is fundamental to the logistic regression.



Turning dot product into a probability

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

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

One word in the context of t

$$P(+|t,c_{1:k}) = \prod_{i=1}^{k} \frac{1}{1+e^{-t\cdot c_i}}$$

$$\log P(+|t,c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1+e^{-t\cdot c_i}}$$

All words in the context of t

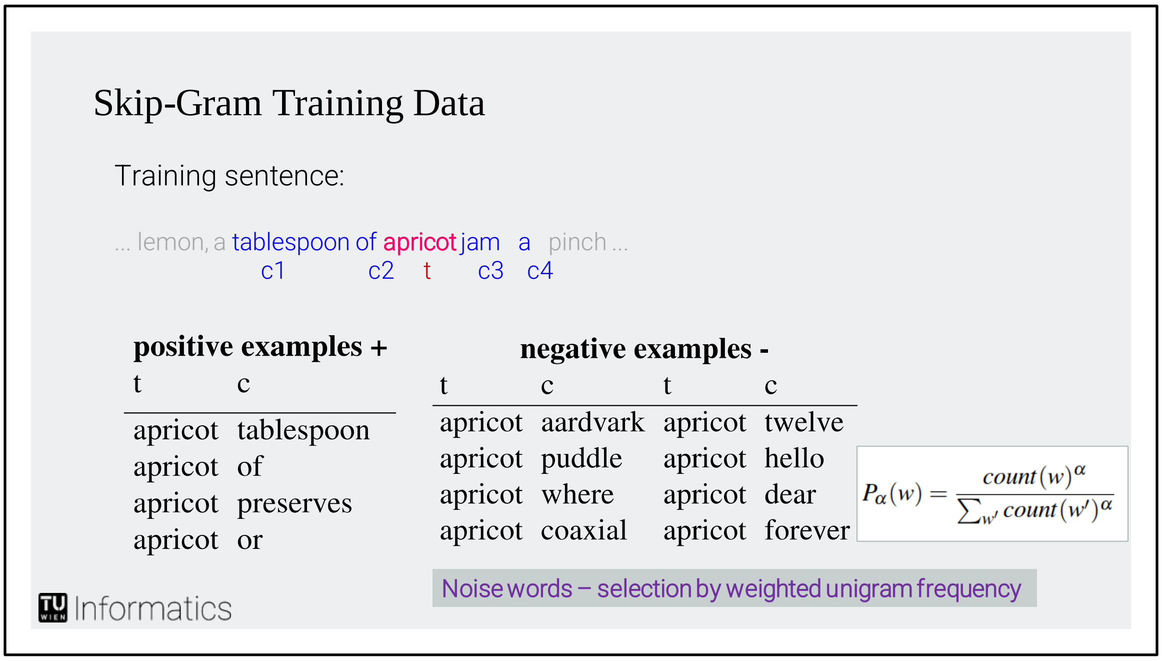
Simplifying assumption!

TU WIEN Informatics

We apply the sigmoid function to the output of the dot product, and we obtain now probabilities. Here is how to do it for one term.

On the left hand is the probability for one word only, but we need to take into account multiple context words. So to compute the probability of term t to have the context formed by words c_1 …c_k we user the formulae on the light hand side.

The Skip-gram model makes a very strong assumption that the context words are independent, so we can just multiply the probabilities of the individual context words (or – in log form – to sum up).



Skip-Gram Training Data

Training sentence:

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

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

$$P_\alpha(w) = \frac{count(w)^\alpha}{\sum_{w'} count(w')^\alpha}$$

Noise words – selection by weighted unigram frequency

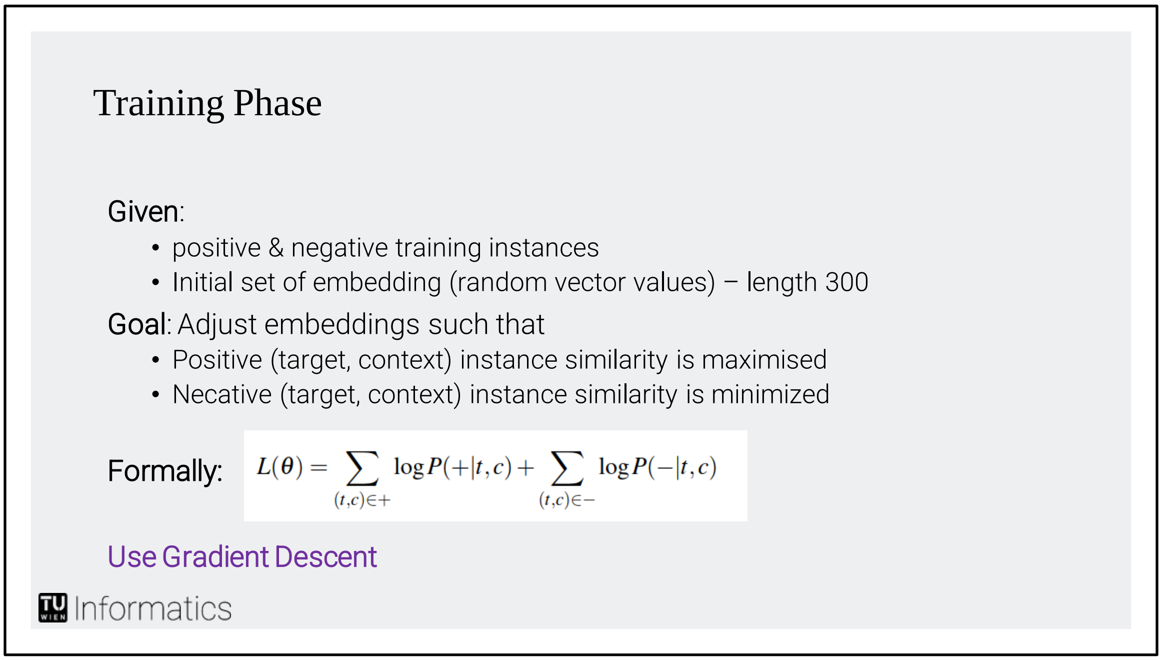
TU WIEN Informatics

Word2vec learns embeddings by starting with an initial set of embedding vectors and then iteratively shifting the embedding of each word w to be more like the embeddings of words that occur nearby in texts, and less like the embeddings of words that don’t occur nearby. Let’s start by considering a single piece of training data:

Let’s look at an example

- For each positive example, we'll create *k* negative examples.
- Using *noise* words
- Any random word that isn't *t*

How to choose the noise words: we do that by their unigram frequency, where the weight, alpha is usually set to .75



Training Phase

Given:

- positive & negative training instances
- Initial set of embedding (random vector values) – length 300

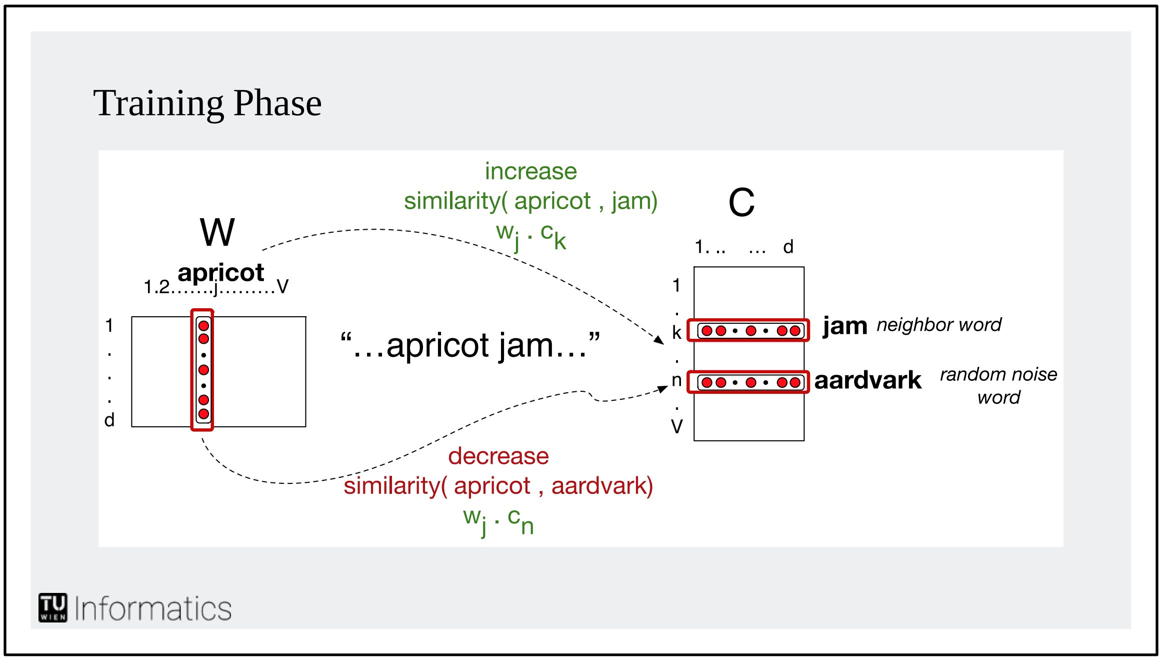
Goal: Adjust embeddings such that

- Positive (target, context) instance similarity is maximised
- Necative (target, context) instance similarity is minimized

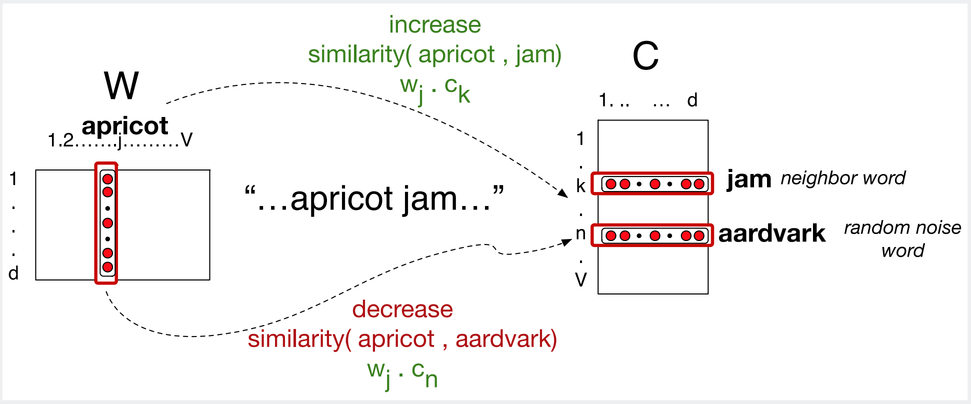
Formally: $$L(\theta) = \sum_{(t,c)\in +} \log P(+|t,c) + \sum_{(t,c)\in -} \log P(-|t,c)$$

Use Gradient Descent

TU WIEN Informatics

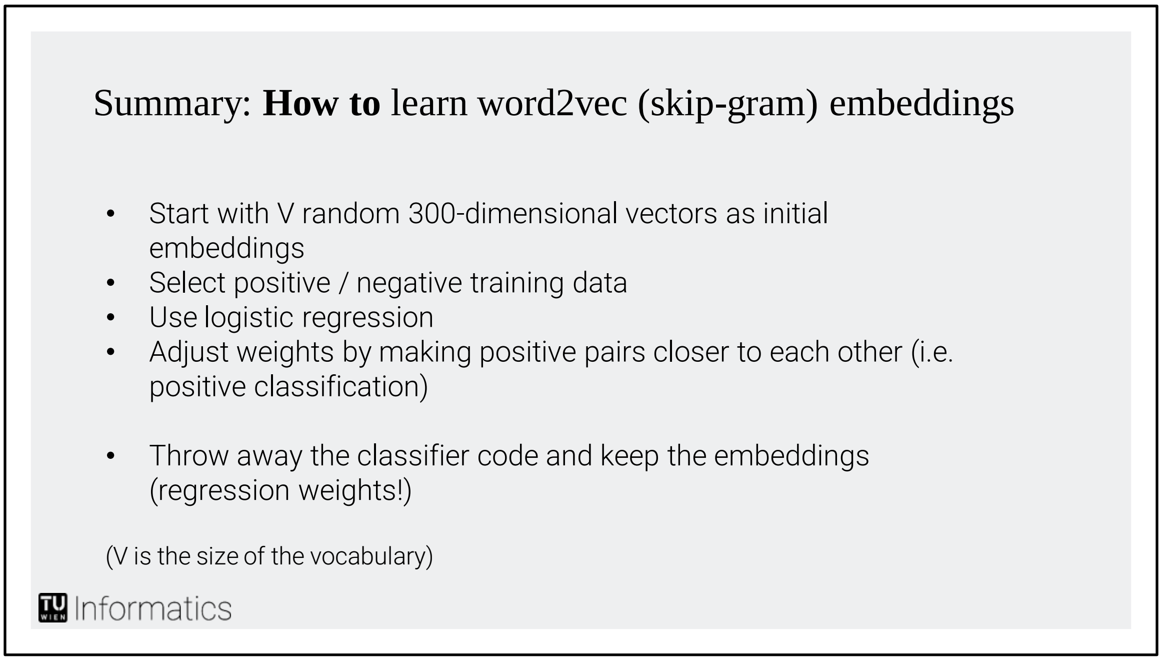


Training Phase



Informatics

Note that we learn, actually, two separate embeddings! W and C. We usually keep W, and through C away, but there are methods to combine them.



Summary: **How to** learn word2vec (skip-gram) embeddings

- Start with V random 300-dimensional vectors as initial embeddings
- Select positive / negative training data
- Use logistic regression
- Adjust weights by making positive pairs closer to each other (i.e. positive classification)

- Throw away the classifier code and keep the embeddings (regression weights!)

(V is the size of the vocabulary)

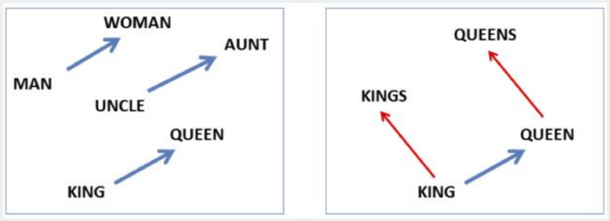
TU WIEN Informatics

Word2Vec Embeddings: Semantic Properties

Similarity depends on context window size:

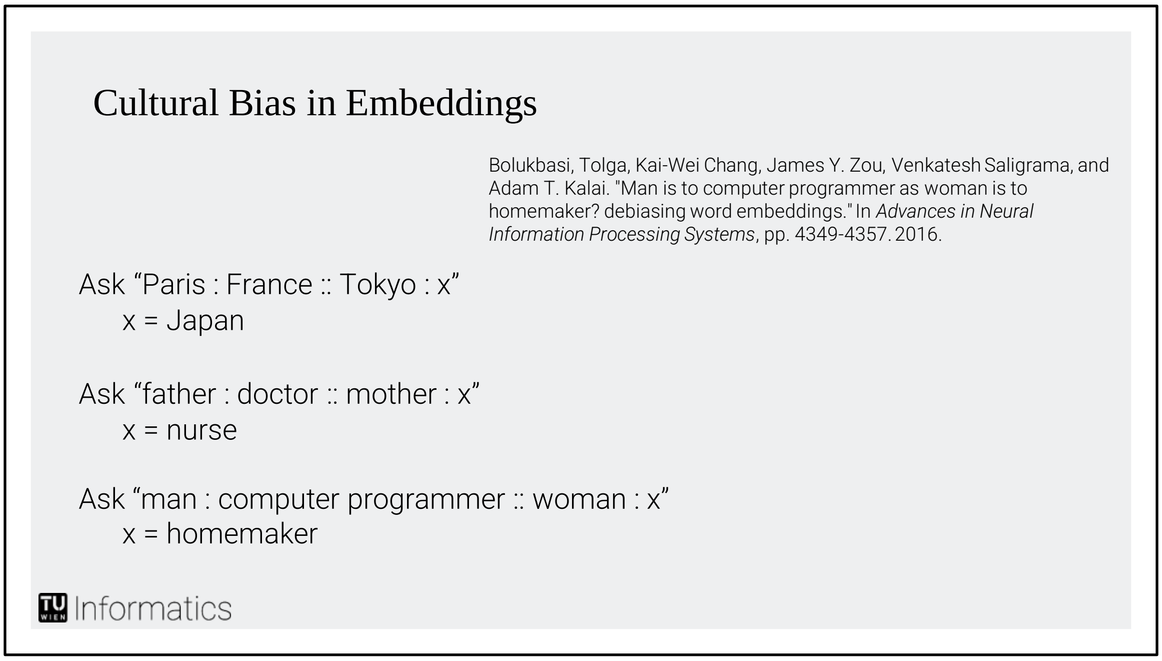
- Short context windows – similar words
- Long context windows – similar topics

Analogy: relational meaning appears to be captured



vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')
vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')

TU WIEN Informatics



Cultural Bias in Embeddings

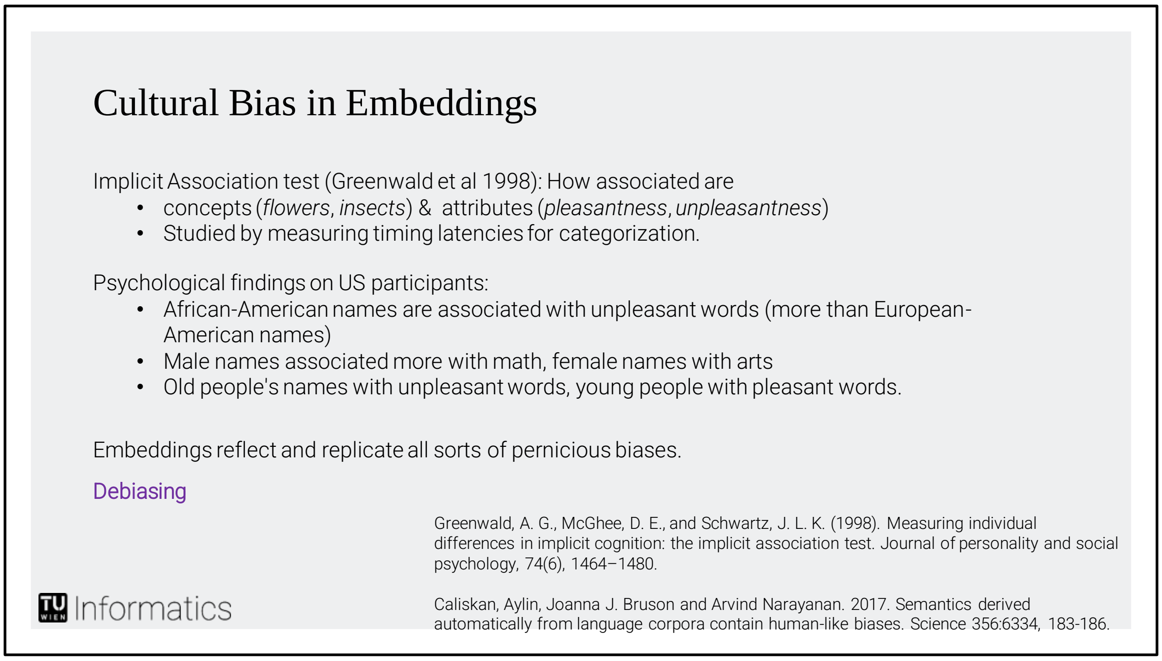
Bolukbasi, Tolga, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, and Adam T. Kalai. "Man is to computer programmer as woman is to homemaker? debiasing word embeddings." In *Advances in Neural Information Processing Systems*, pp. 4349-4357. 2016.

Ask "Paris : France :: Tokyo : x"
 x = Japan

Ask "father : doctor :: mother : x"
 x = nurse

Ask "man : computer programmer :: woman : x"
 x = homemaker

TU WIEN Informatics



Cultural Bias in Embeddings

Implicit Association test (Greenwald et al 1998): How associated are
- concepts (*flowers*, *insects*) & attributes (*pleasantness*, *unpleasantness*)
- Studied by measuring timing latencies for categorization.

Psychological findings on US participants:
- African-American names are associated with unpleasant words (more than European-American names)
- Male names associated more with math, female names with arts
- Old people's names with unpleasant words, young people with pleasant words.

Embeddings reflect and replicate all sorts of pernicious biases.

Debiasing

Greenwald, A. G., McGhee, D. E., and Schwartz, J. L. K. (1998). Measuring individual differences in implicit cognition: the implicit association test. Journal of personality and social psychology, 74(6), 1464–1480.

Caliskan, Aylin, Joanna J. Bruson and Arvind Narayanan. 2017. Semantics derived automatically from language corpora contain human-like biases. Science 356:6334, 183-186.

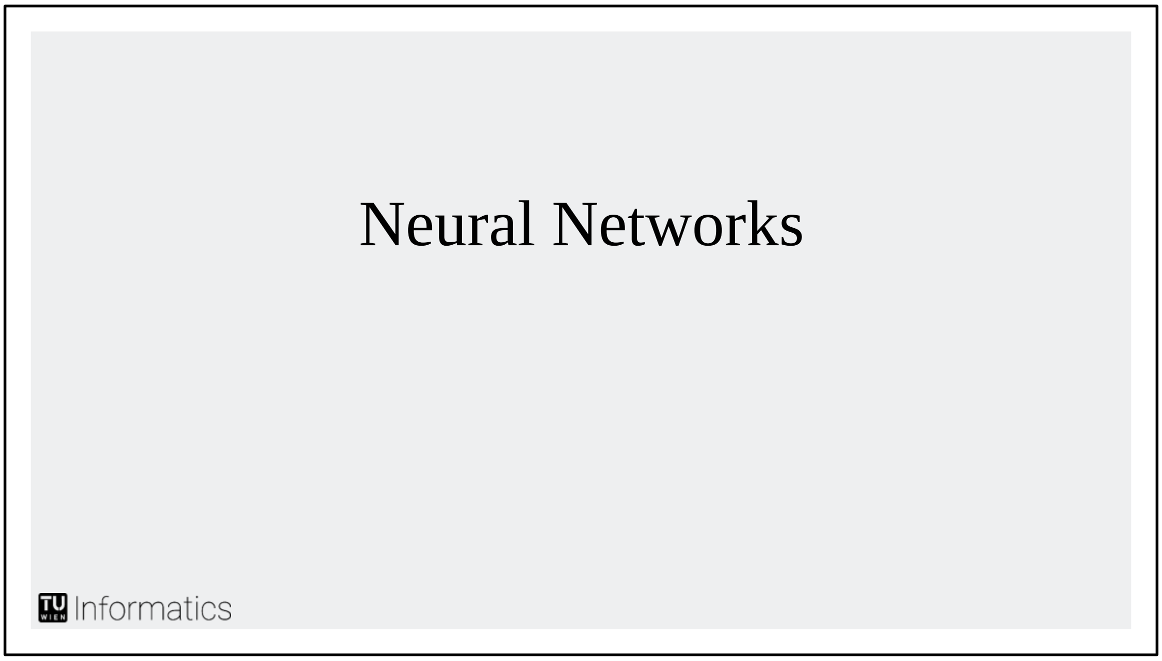
Informatics

Recent research focuses on ways to try to remove these kinds of biases, for example by developing a transformation of the embedding space that removes gender stereotypes but preserves definitional gender (Bolukbasi et al. 2016, Zhao et al. 2017) or changing the training procedure (Zhao et al., 2018b). However, although these debiasing sorts of debiasing may reduce bias in embeddings, they do not eliminate it (Gonen and Goldberg, 2019), and this remains an open problem.

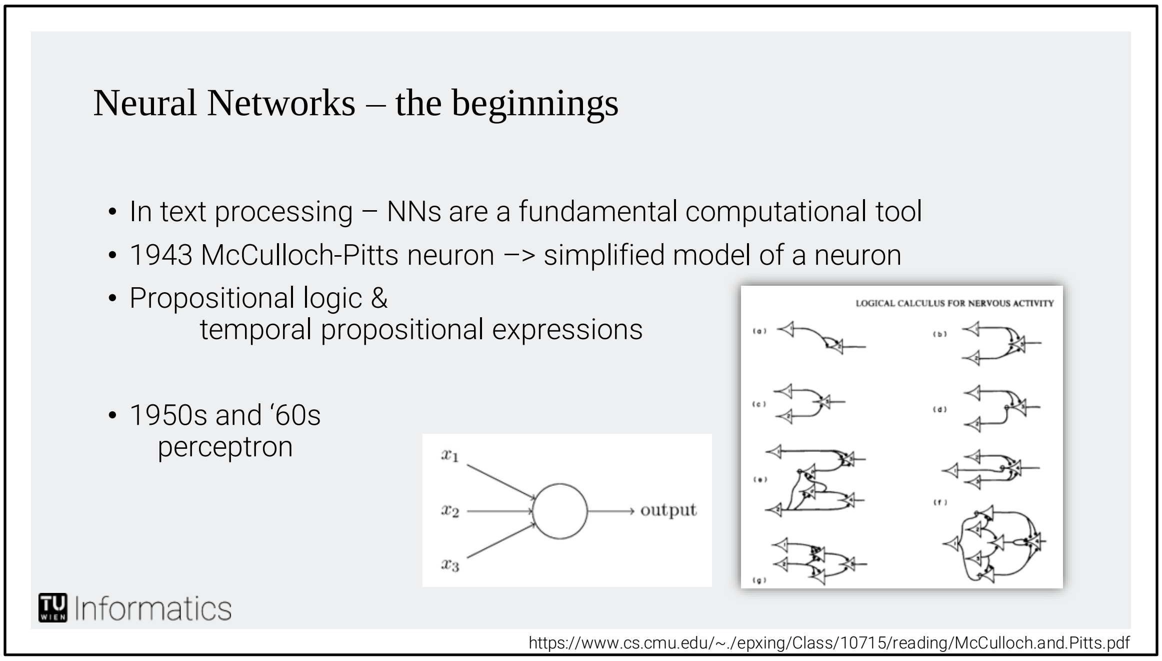
Recap

- Vector Semantics & Embeddings
  - Lexical and Vector Semantics
  - Words as Vectors
  - Measuring similarity & tf-idf
    - Sparse
  - Word2Vec
- Neural Networks

Informatics

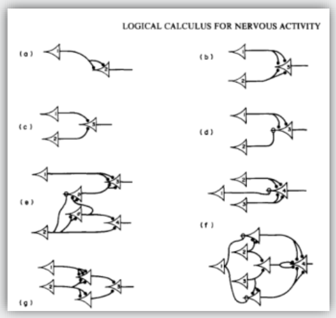


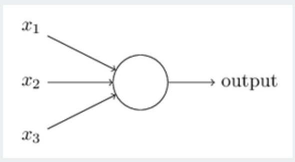
Neural Networks



Neural Networks – the beginnings

- In text processing – NNs are a fundamental computational tool
- 1943 McCulloch-Pitts neuron –> simplified model of a neuron
- Propositional logic &
  temporal propositional expressions
- 1950s and '60s
  perceptron





Informatics

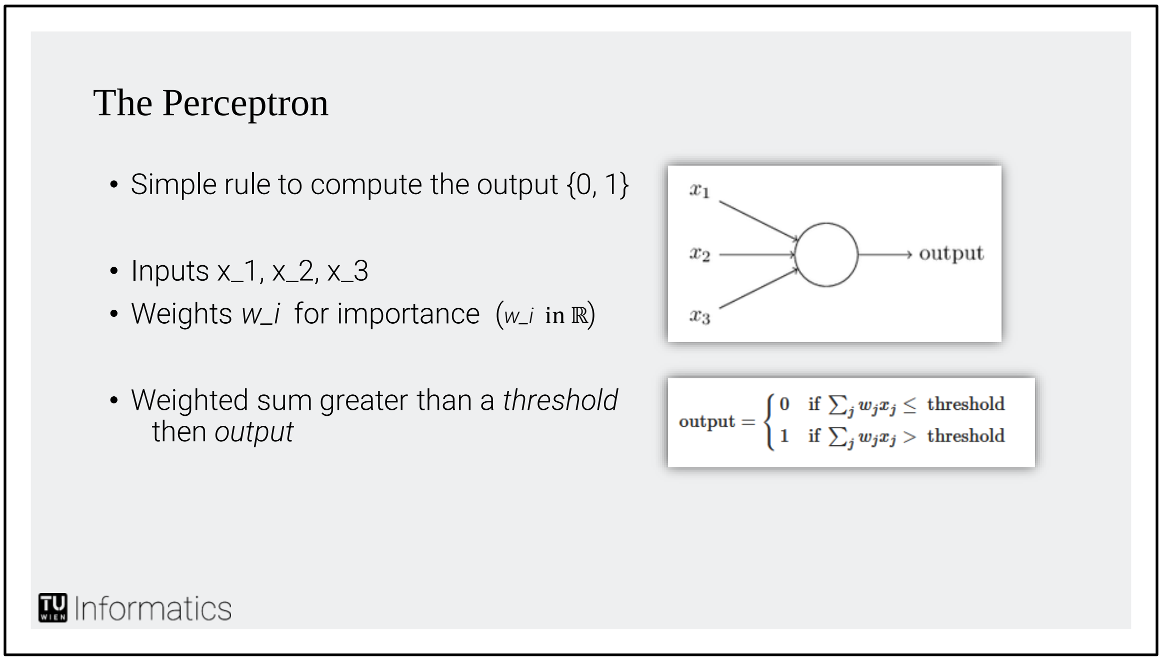
https://www.cs.cmu.edu/~./epxing/Class/10715/reading/McCulloch.and.Pitts.pdf

Neural networks are a fundamental computational tool for language processing, and a very old one.

A simplified model of the human neuron was introduced as a kind of computing element that could be described in terms of
propositional logic. The authors of that paper wrote in 1943: "Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic."

Propositional logic takes 1-2 inputs, if you have more inputs it is actually a combination of operations that take 1-2 inputs, only.

BUT the modern NN is network of small computing units, each taking a vector of input values and outputting a single value. And most of the NN introductions start with describing the perceptron, first introduced by Frank Rosenblatt who was inspired by McCulloghs and Pitts earlier work.

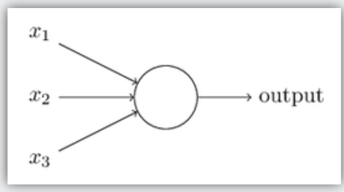


The Perceptron

- Simple rule to compute the output {0, 1}

- Inputs x_1, x_2, x_3
- Weights *w_i* for importance (*w_i* in $\mathbb{R}$)

- Weighted sum greater than a *threshold* then *output*

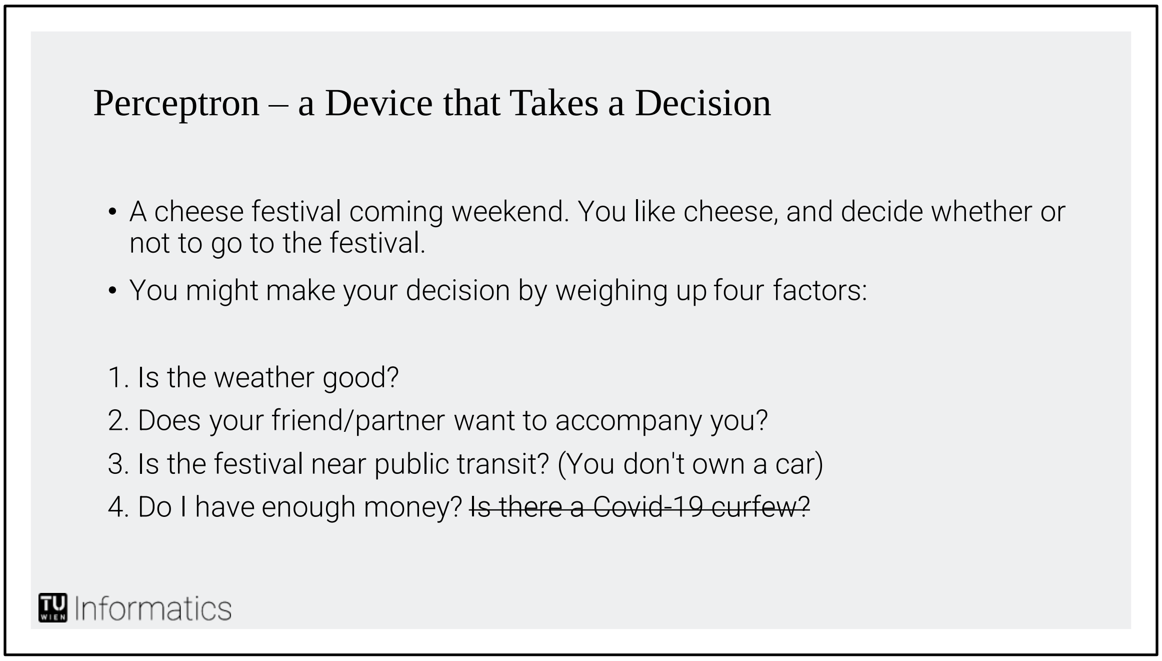


$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{threshold} \end{cases}$$

Informatics

In the example shown the perceptron has three inputs, *x*1, *x*2, *x*3.

In general it could have more or fewer inputs.
Rosenblatt proposed a simple rule to compute the output. He introduced *weights*, *w*1, *w*2,…, real numbers expressing the importance of the respective inputs to the output. The neuron's output, 0 or 1, is determined by whether the weighted sum ∑*jwjxj* is less than or greater than some *threshold value*. Just like the weights, the threshold is a real number which is a parameter of the neuron.



Perceptron – a Device that Takes a Decision

- A cheese festival coming weekend. You like cheese, and decide whether or not to go to the festival.
- You might make your decision by weighing up four factors:

1. Is the weather good?
2. Does your friend/partner want to accompany you?
3. Is the festival near public transit? (You don't own a car)
4. Do I have enough money? ~~Is there a Covid-19 curfew?~~

Perceptron – a Device that Takes a Decision

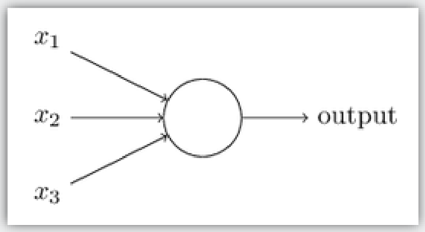
by weighting up evidence

1. Is the weather good?

2. Friend/partner Joining?

3. Public transportation

4. Do I have enough money?

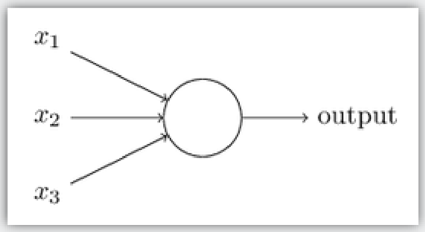


output = {1/go, 0/no_go}

Informatics

Perceptron – a Device that Takes a Decision

1. Is the weather good?
   0 – bad, 1 – good
2. Friend/partner Joining?
   0 – no, 1 – yes
3. Public transportation
   0 – no, 1 – yes
4. Do I have enough money?
   0 – no, 1 – yes



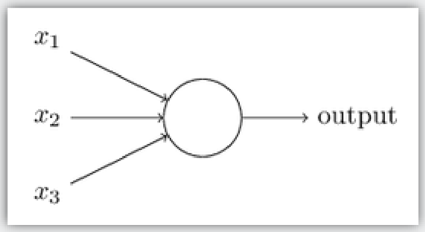
output = {1/go, 0/no_go}

Informatics

Perceptron – a Device that Takes a Decision

1. Is the weather good?
   0 – bad, 1 – good, *w_1 = 6*
2. Friend/partner Joining?
   0 – no, 1 – yes *w_2 = 2*
3. Public transportation
   0 – no, 1 – yes *w_3= 2*
4. Do I have enough money?
   0 – no, 1 – yes *w_4 = -5*

compute $\sum_j w_j x_j$

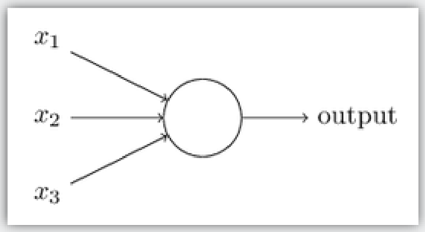


output = {1/go, 0/no_go}
threshold = 7

Informatics

Perceptron – a Device that Takes a Decision

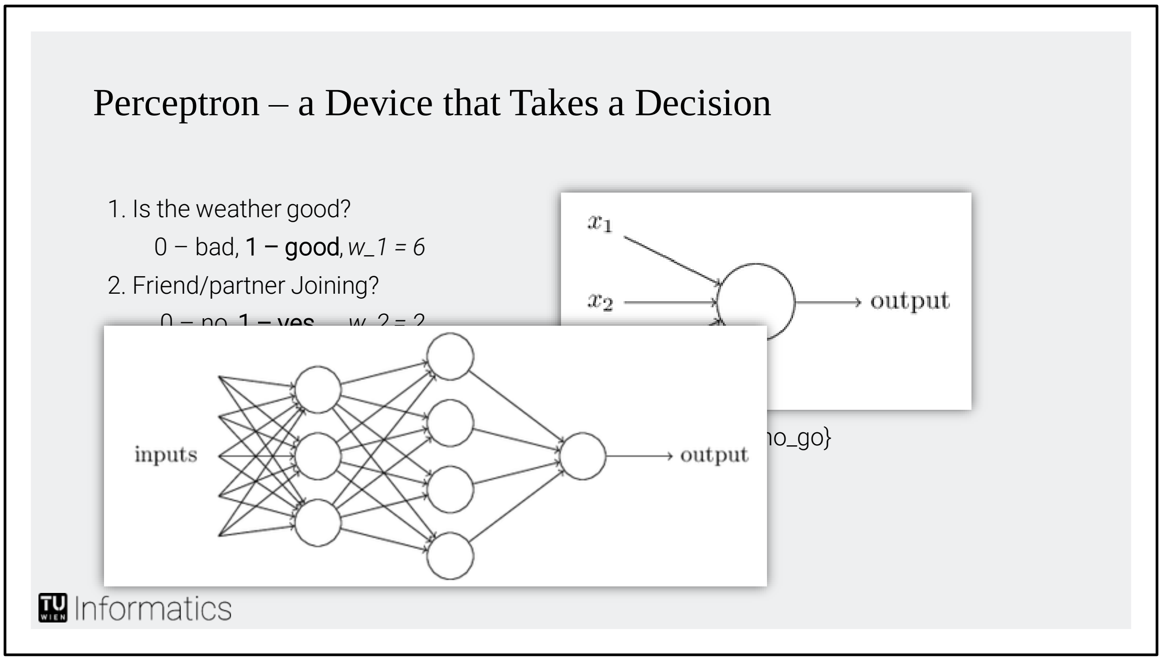
1. Is the weather good?
   0 – bad, 1 – good, *w_1 = 6*
2. Friend/partner Joining?
   0 – no, 1 – yes *w_2 = 2*
3. Public transportation
   0 – no, 1 – yes *w_3= 2*
4. Do I have enough money?
   0 – no, 1 – yes *w_4 = -5*



output = {1/go, 0/no_go}
threshold = 7

compute = $\sum_j w_j x_j$ 8 (first three factors are 1, go) = 3 (all factors are 1, no_go)

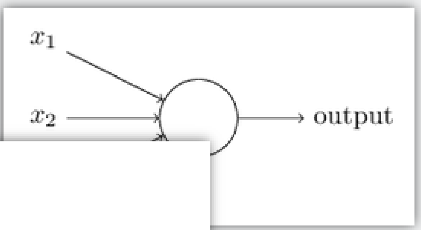
Informatics

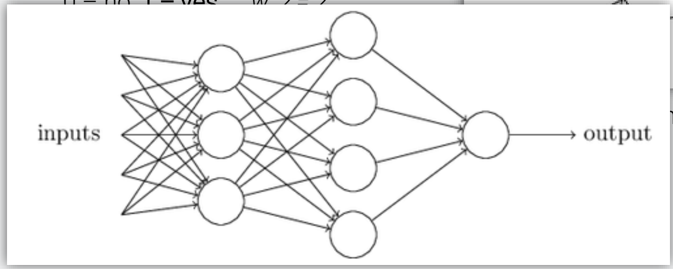


Perceptron – a Device that Takes a Decision

1. Is the weather good?
   0 – bad, 1 – good, *$w_1 = 6$*
2. Friend/partner Joining?
   0 – no, 1 – yes, *$w_2 = 2$*

no_go}





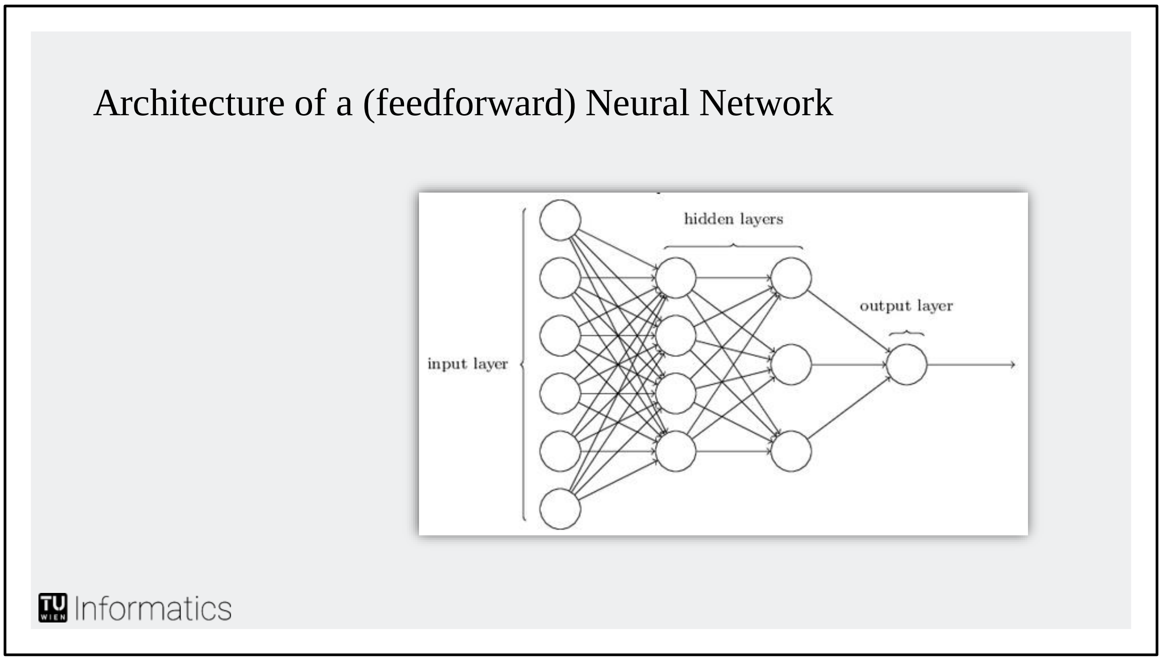
Obviously, such a perceptron is **not** a complete model for how we make decisions.

But what the example illustrates is how a perceptron can weigh up different kinds of evidence in order to make decisions. And it is maybe more evident to you, now, that a complex network of perceptrons can make subtle decisions.

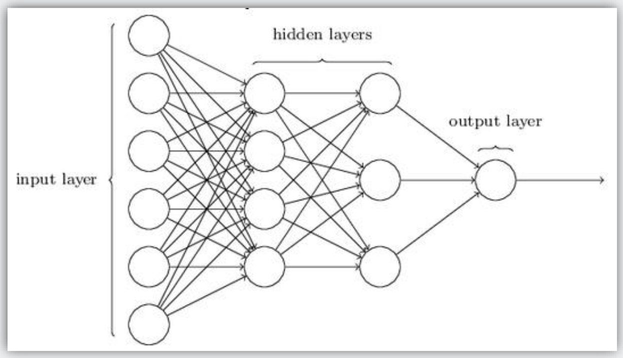
In this network, the first column of perceptrons - what we'll call the first *layer* of perceptrons - is making three very simple decisions, by ***weighing the input evidence***. Perceptrons in the second layer weigh up the results from the first layer of decision-making.

So they are taking decisions based on previous decisions, and at a more abstract level than the perceptrons in the first level. And so on.

Notice that some perceptrons seem to have multiple output arrows, even though we have defined them as having only one output. This is only meant to indicate that a singleoutput is being sent to multiple new perceptrons

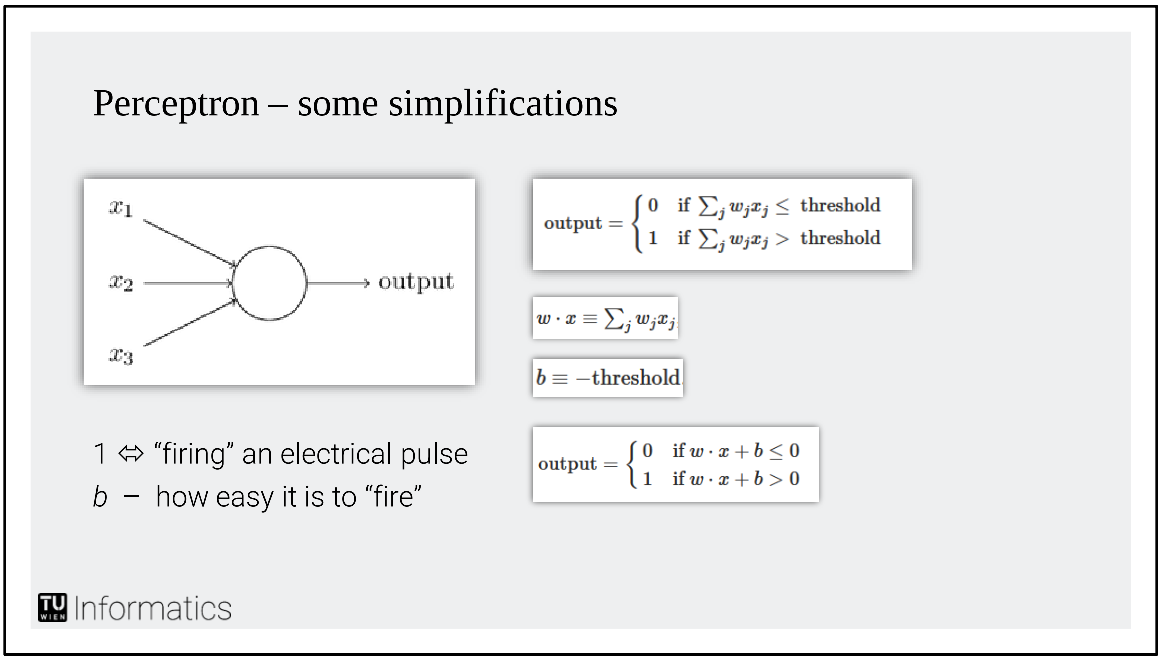


Architecture of a (feedforward) Neural Network

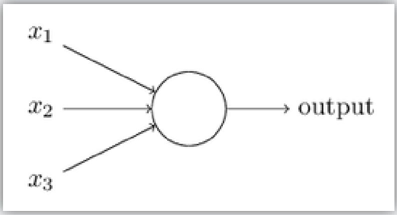


Informatics

The input and outputs are typically represented as their own neurons, with the other neurons named *hidden layers.* The term hidden only means – not an input and not an output – no other deep meaning is behind the notion of "hidden layer"



Perceptron – some simplifications



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

$$w \cdot x \equiv \sum_j w_j x_j$$

$$b \equiv -\text{threshold}$$

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

1 ⬄ “firing” an electrical pulse

b – how easy it is to “fire”

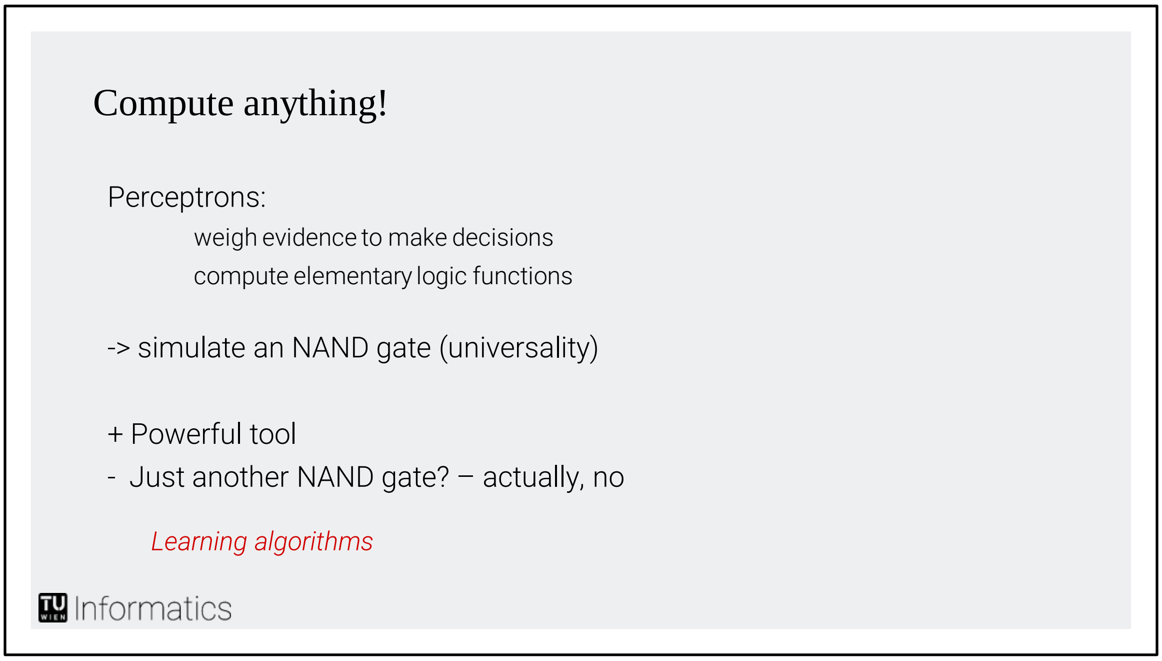
Informatics

Let’s simplify the description of the perceptron: The condition SUM >= threshold is cumbersome, and we can do some notational changes to improve it.

First, we can write the Sum term as a dot product (vectorization anyone) where w and x are vectors (weights and inputs)
The second is to move the threshold to the other side of the inequality, and replace it with what you know as the perceptron’s *bias*.

The biological interpretation of a perceptron is this: when it emits a 1 this is equivalent to ‘firing’ an electrical pulse, and when it is 0 this is when it is not firing.

The bias indicates how difficult it is for this particular node to send out a signal
-



Compute anything!

Perceptrons:

weigh evidence to make decisions

compute elementary logic functions

-> simulate an NAND gate (universality)

+ Powerful tool
- Just another NAND gate? – actually, no

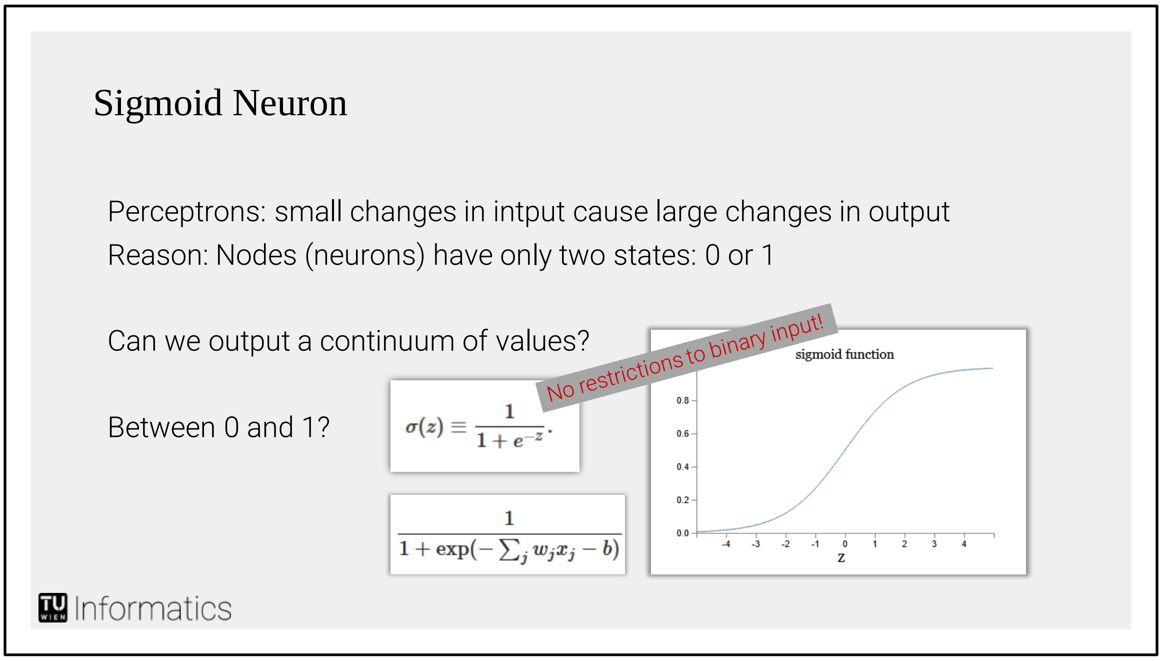
Learning algorithms

Informatics

In fact perceptrons are universal for computation.

The situation is better than this view suggests. It turns out that we can devise *learning algorithms* which can automatically tune the weights and biases of a network of artificial neurons. This tuning happens in response to external stimuli, without direct intervention by a programmer. These learning algorithms enable us to use artificial neurons in a way which is radically different to conventional logic gates. Instead of explicitly laying out a circuit of NAND and other gates, our neural networks can simply learn to solve problems, sometimes problems where it would be extremely difficult to directly design a conventional circuit.

-



Sigmoid Neuron

Perceptrons: small changes in intput cause large changes in output
Reason: Nodes (neurons) have only two states: 0 or 1

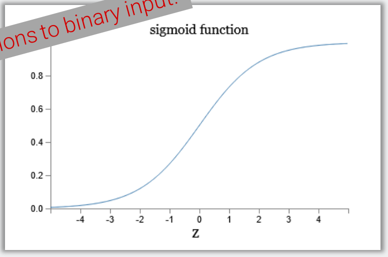
Can we output a continuum of values?

Between 0 and 1?

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}.$$

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}$$

No restrictions to binary input!



Informatics

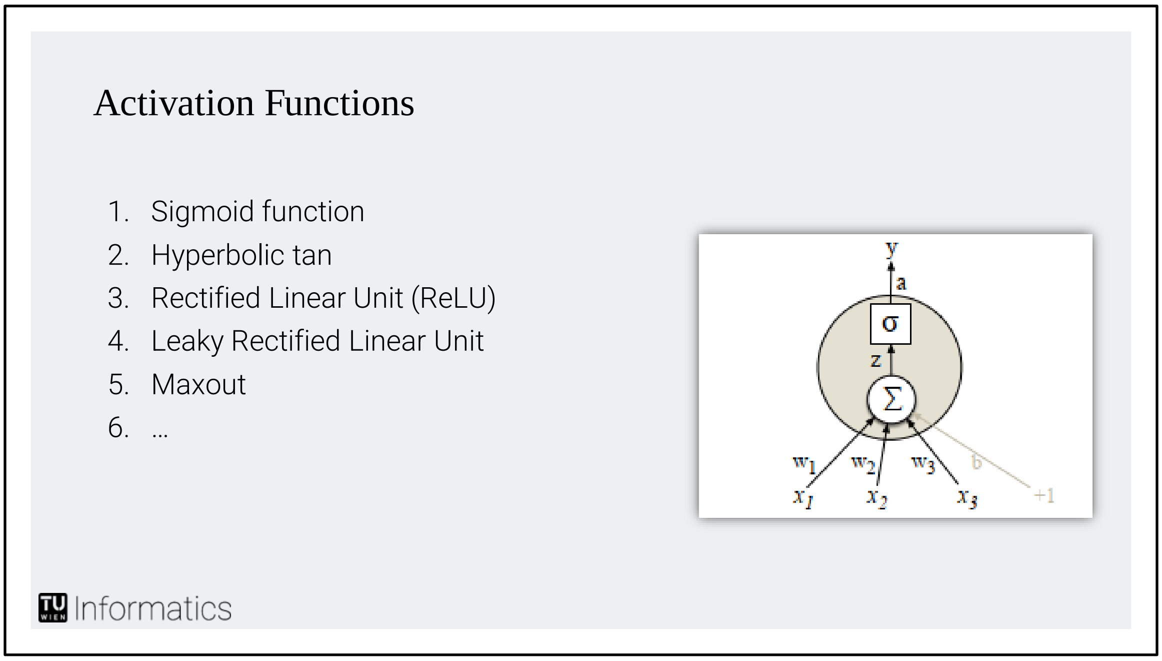
An important shortcoming of a perceptron is that a small change in the input values can cause a large change the output because each node (or neuron) only has two possible states: 0 or 1.

A better solution would be to output a continuum of values, say any number between 0 and1.Most tutorials spend a significant amount of time describing the conceptual leap from binary outputs to a continuous output.

I our case, we grab in our tool box of mathematical functions and choose one that transforms the input onto an [0, 1] interval.
The most used is this sigmoid function.

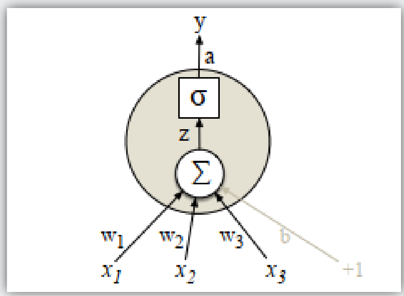
This function reflects much better the change in the input into changes in the output. It is very similar to the classic perceptron behaviour, as for very big input values, this function will be 1, and for very large negative values, the value of the output is 0.

Most notable: outliers are squashed towards 0 or 1 (very large pos or neg – are the outliers in these cases)



Activation Functions

1. Sigmoid function
2. Hyperbolic tan
3. Rectified Linear Unit (ReLU)
4. Leaky Rectified Linear Unit
5. Maxout
6. …



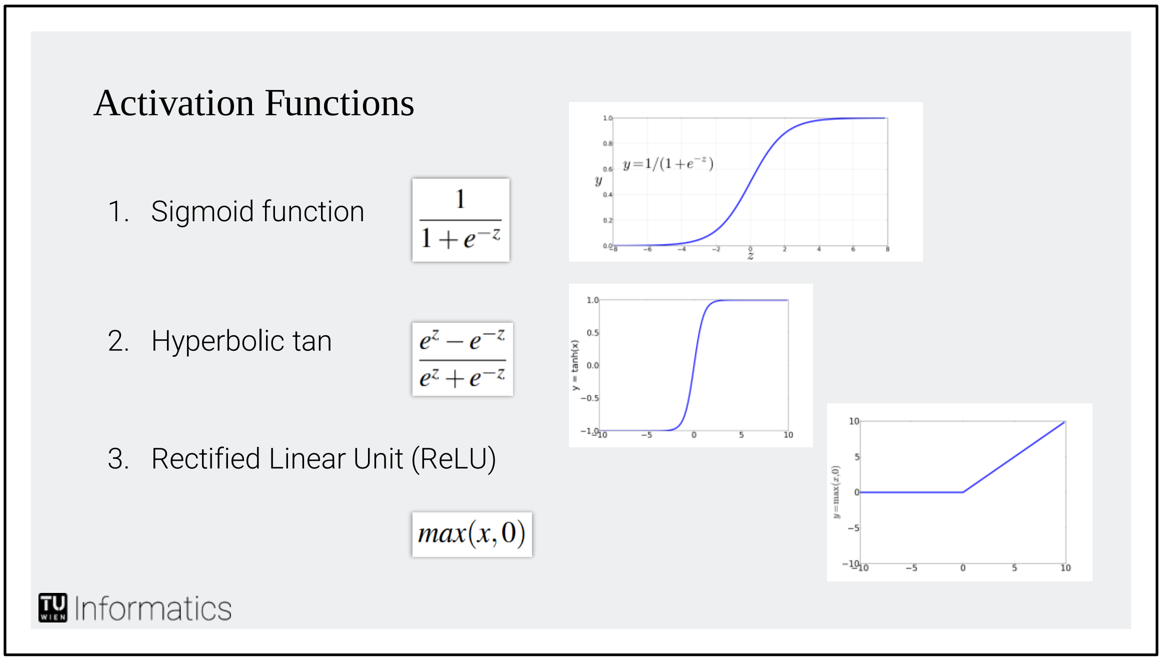
Informatics

In the sigmoid neuron example, the choice of what function to use to go from x.w+b to an output is called the activation function.

Using a logistic or a sigmoid activation function has some benefits in being able to easily take derivatives and the interpret them using logistic regression.

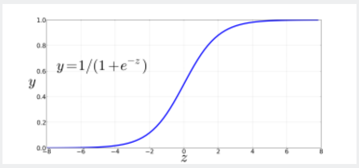
Other choices have certain benefits that have recently grown in popularity. Some of these include:

These activation functions have different properties that make them useful for different language applications or network architectures.

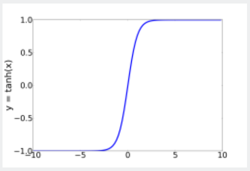


Activation Functions

1. Sigmoid function $\frac{1}{1+e^{-z}}$

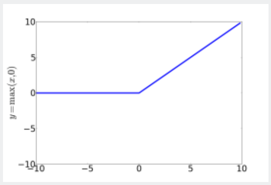


2. Hyperbolic tan $\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$



3. Rectified Linear Unit (ReLU)

$max(x,0)$



TU WIEN Informatics

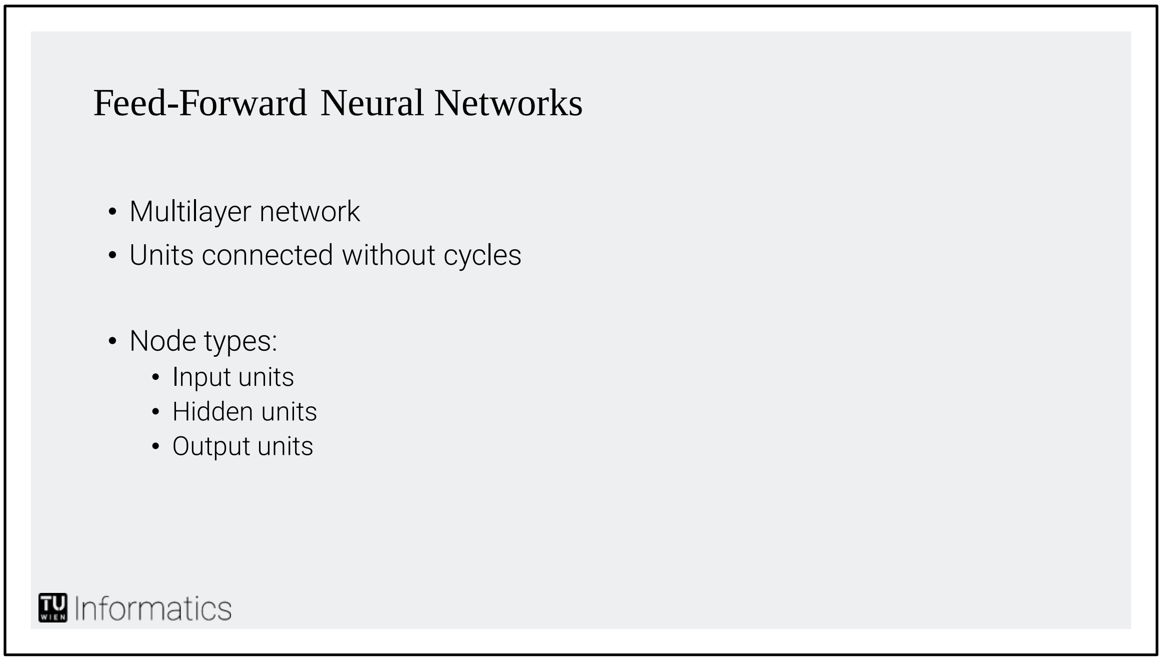
These activation functions have different properties that make them useful for different language applications or network architectures.

For example the rectifier function has nice properties that result from it being very close to linear.

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, which causes problems for learning.

Rectifiers don’t have this problem, since the output of values close to 1 also approaches 1 in a nice gentle linear way.

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



Feed-Forward Neural Networks

- Multilayer network
- Units connected without cycles
- Node types:
  - Input units
  - Hidden units
  - Output units

TU WIEN Informatics

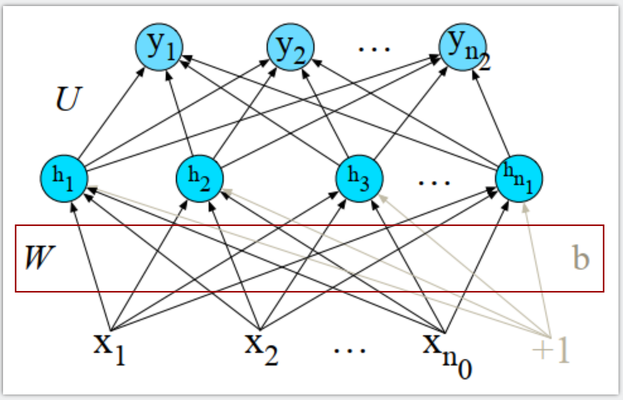
A feedforward network is a multilayer network in which the units are connected with no cycles.

-

Feed-forward Neural Network

- Fully connected
- Hidden units sum over all inputs
- $W_{i,j}$ link between x_i and h_j

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



TU WIEN Informatics

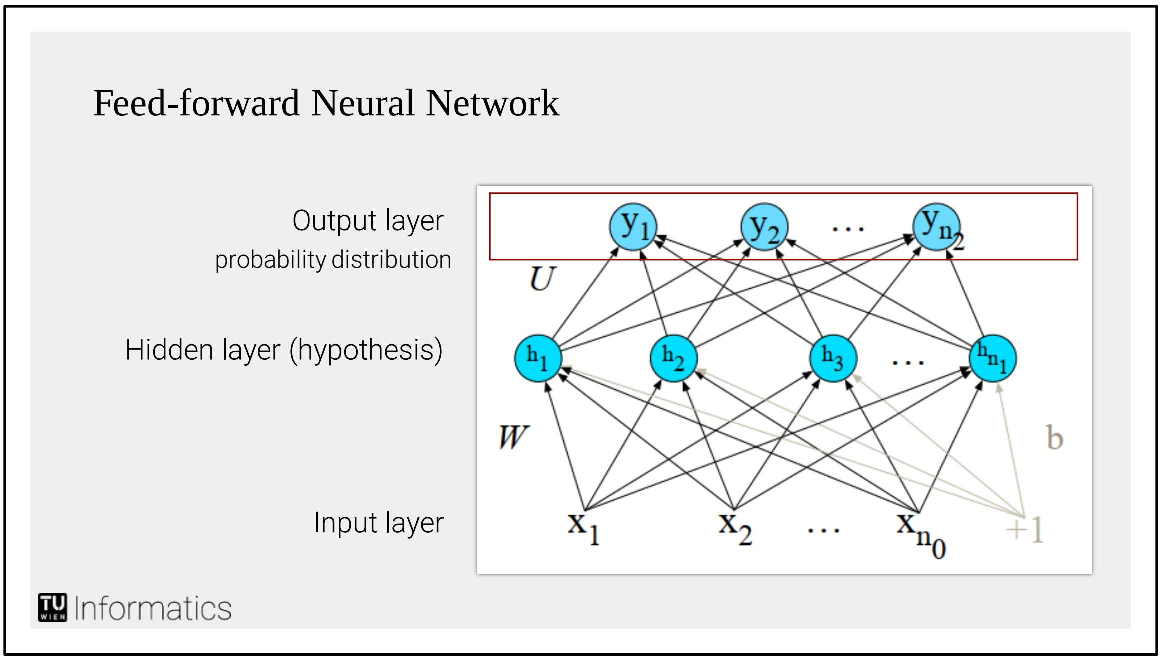
Here we have a FF NN with one hidden layer.

Recall that the hidden units have parameter w (weights) and b (the bias). We can represent the parameters for the ENTIRE hidden layer with a single matrix W which combines the weight vector and the bias for each unit i, and a single bias vector for the whole layer.

Having the weights stored as a matrix allows the use of efficient matrix computations

In fact, the computation only has three steps: multiplying the weight matrix by the input vector x, adding the bias vector b, and applying the activation function g

IN this example, the activation function is applied elementwise.



Feed-forward Neural Network

Output layer
probability distribution

Hidden layer (hypothesis)

Input layer

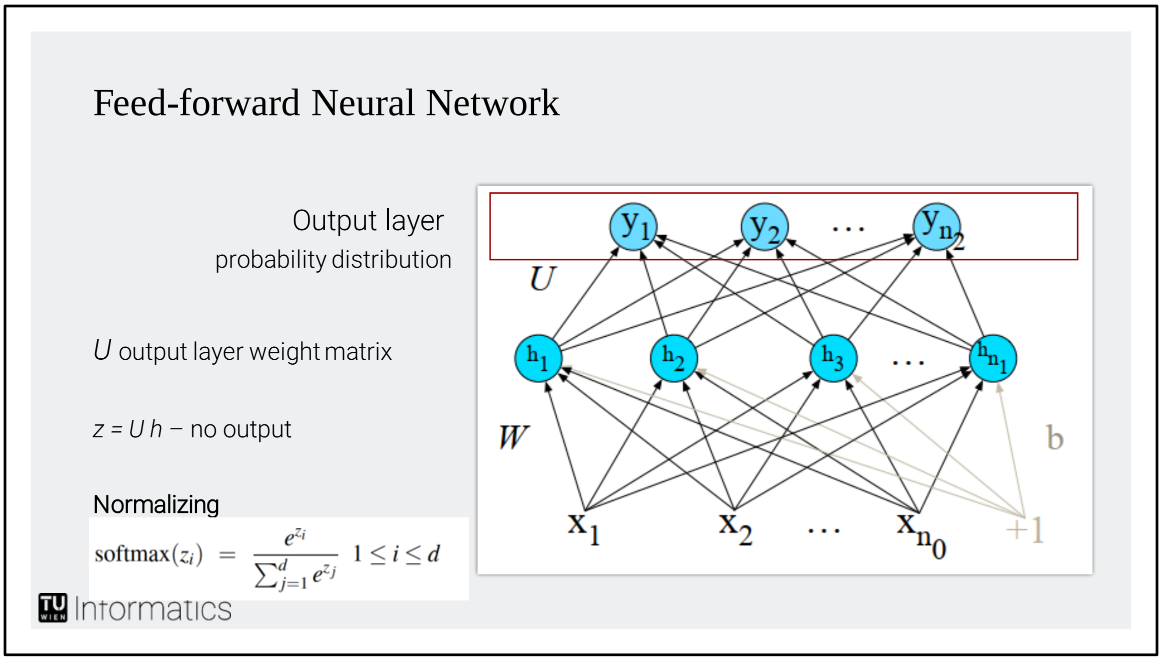
Informatics

The hidden layer takes the input and forms a representation of that input.
The output layer takes this new representation and computes a final output.

The output could be a single real value – like we‘ve seen in the simple feed forward network before, with one output node only.

But in many cases, the goal of the network is to make some sort of classification decision.
If we do a binary classification (e.g. sentiment analysis: positive vs. Negative sentences) one output node is sufficient.
For multinomial classification (POS) we need one output node for each potential POS. The output of these nodes are, then, probability values that an input is assigned a certain POS tag.

And since we talk about probabilities, all these values must sum to one.
The output layer thus SHOULD give a probability distribution across the output nodes.



Feed-forward Neural Network

Output layer
probability distribution

U output layer weight matrix

$z = U\,h$ – no output

Normalizing

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

Informatics

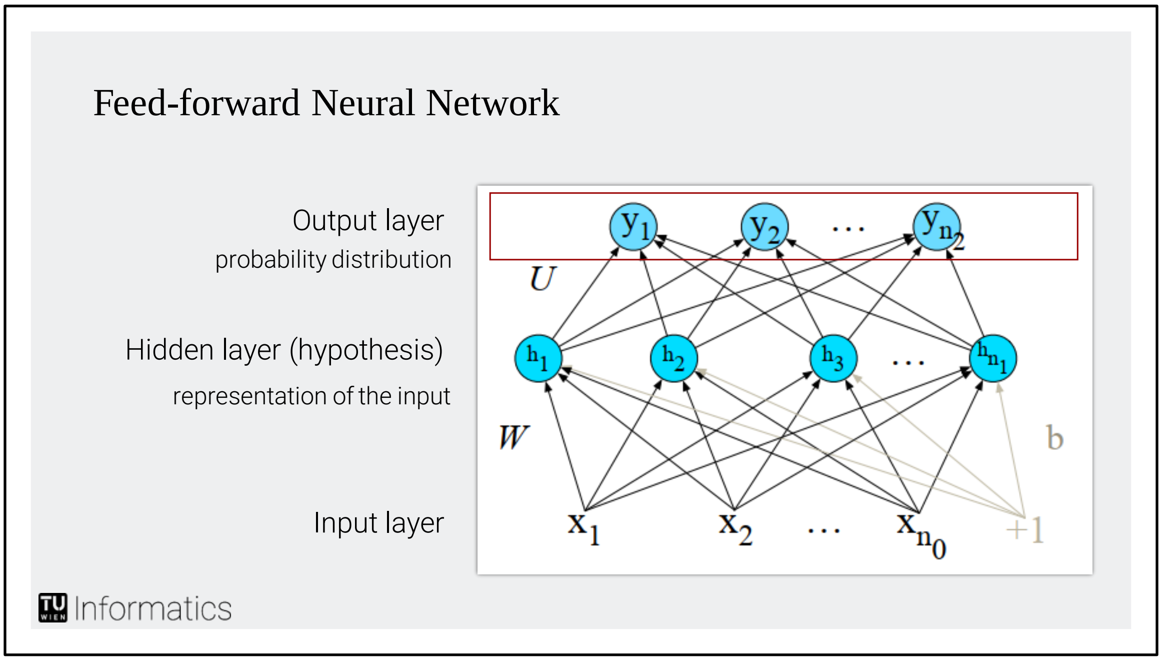
The weight matrix is multiplied by its input vector (h) to produce the intermediate output z.
z = Uh

There are n2 output nodes, so z ∈ R^n2 , weight matrix U has dimensionality U ∈ R^n2×n1 , and element Ui j is the weight from unit j in the hidden layer to unit i in the output layer.

However, z can't be the output of the classifier, since it's a vector of real-valued numbers, while what we need for classification is a vector of probabilities.

How do we change that into a vector or probabilities? By normalizing and transforming the vector z to a vector of probabilities with values between 0 and 1, and summing up to 1. For example: softmax

(d is the length of the vector z, on this picture d = n_2)

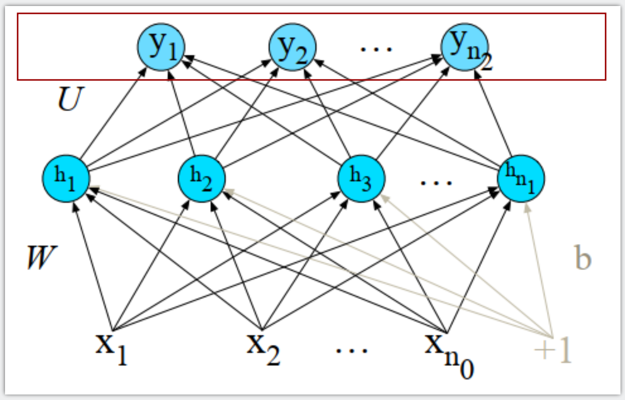


Feed-forward Neural Network

Output layer
probability distribution

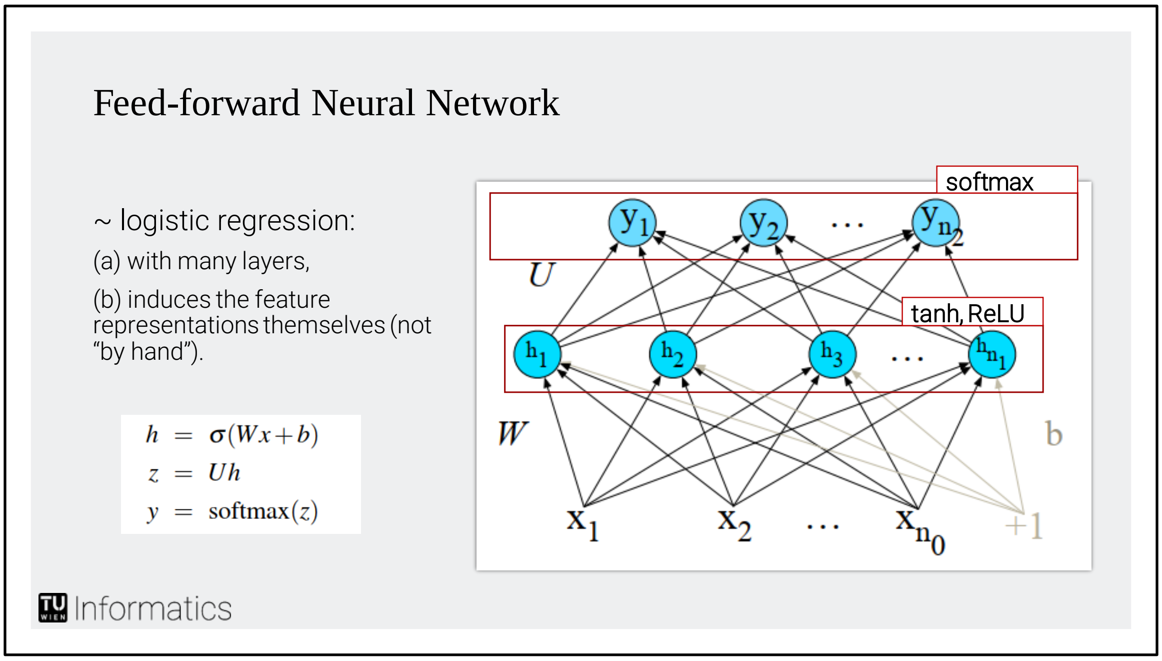
Hidden layer (hypothesis)
representation of the input

Input layer



Informatics

So we can think of a NN classifier with one hidden layer as building a vector h which is a hidden layer representation of the input, and then running standard logistic regression on the features that the network develops in h.



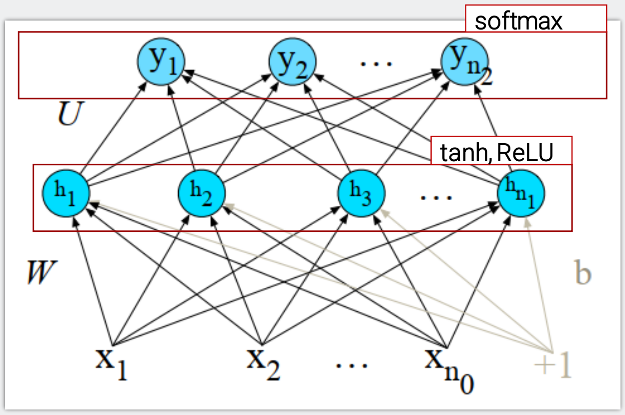
Feed-forward Neural Network

~ logistic regression:

(a) with many layers,

(b) induces the feature representations themselves (not “by hand”).

$$\begin{aligned} h &= \sigma(Wx+b) \\ z &= Uh \\ y &= \text{softmax}(z) \end{aligned}$$

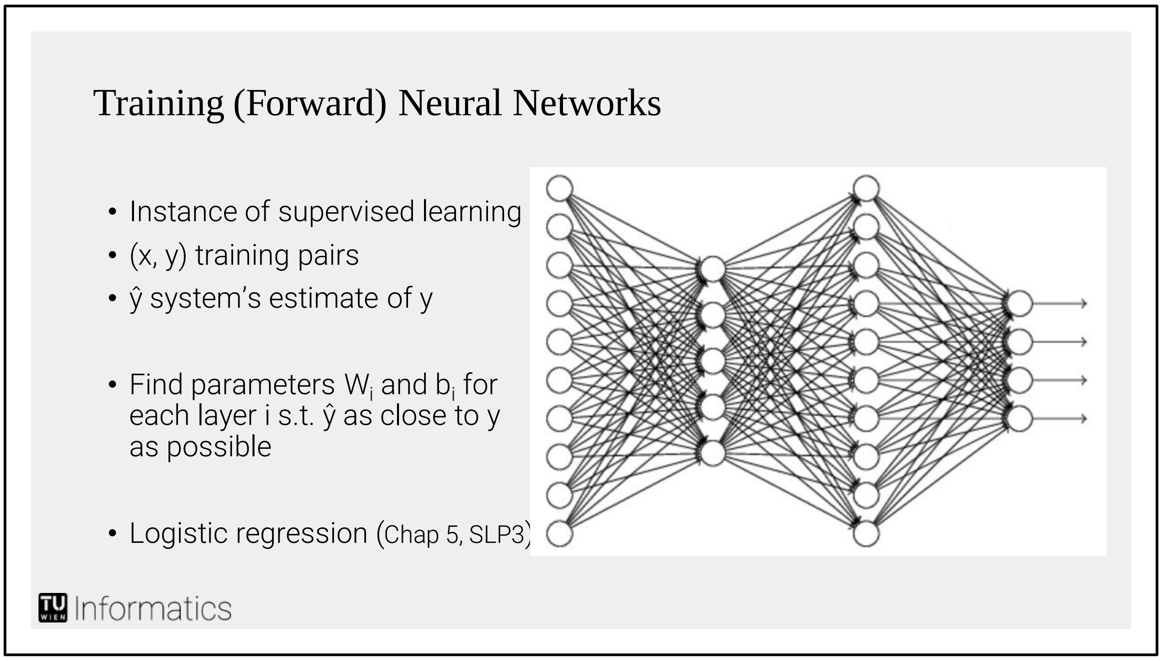


Informatics

So a neural network is like logistic regression, but (a) with many layers, since a deep neural network is like layer after layer of logistic regression classifiers, and (b) rather than forming the features by feature templates, the prior layers of the network induce the feature representations themselves.

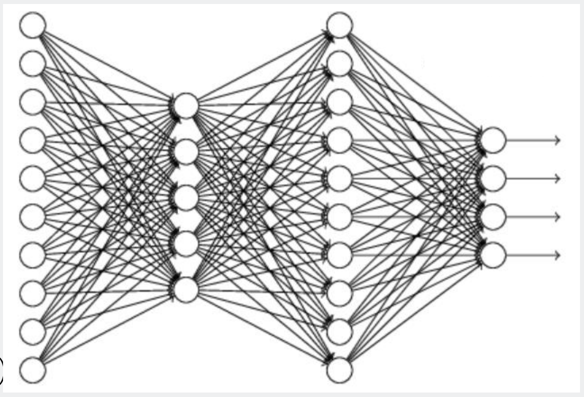
And to summarize: these are the final equations you need for a feedforward network with a single hidden layer, that takes input a vector x, outputs a vector y of probabilities, and is parametrized with the U and W weight matrices.

IN practice for hidden layers the activation function of choice is ReLU or tanh, and a softmax is applied to the output layer.



Training (Forward) Neural Networks

- Instance of supervised learning
- (x, y) training pairs
- ŷ system's estimate of y
- Find parameters W_i and b_i for each layer i s.t. ŷ as close to y as possible
- Logistic regression (Chap 5, SLP3)

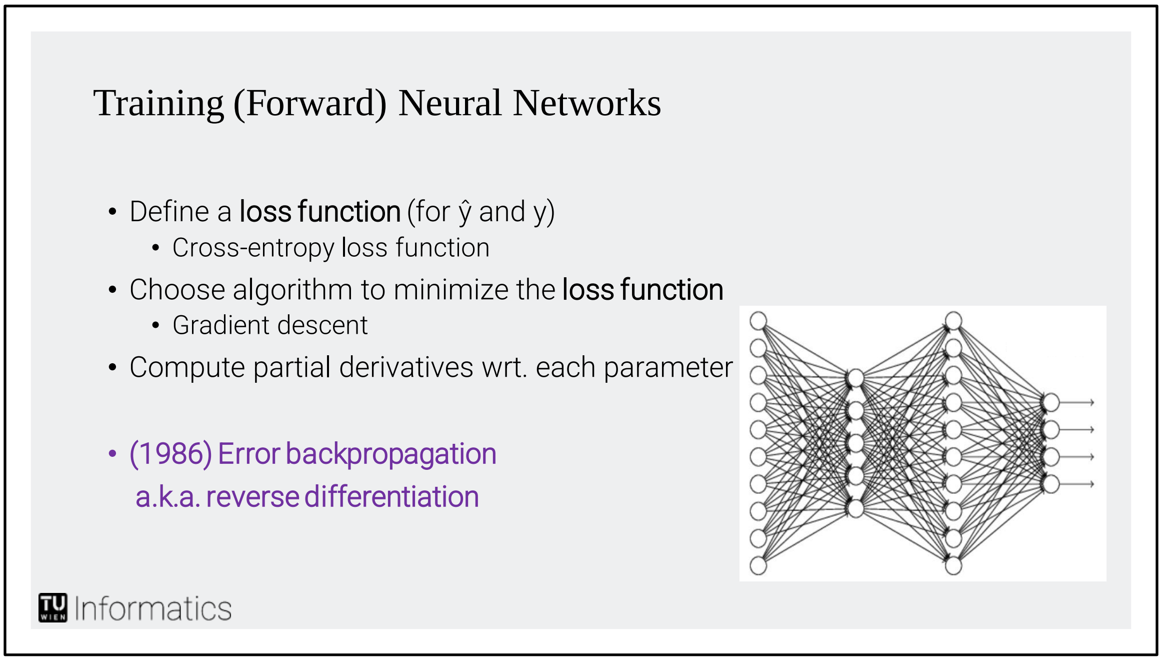


Informatics

In the previous slides we took the weights at given, we've seen how to compute with a neural network that was given to us. How do we learn the weights and the biases?

Training feedforward neural net is an instance of supervised machine learning in which we know the correct output y for each observation x. What the system produces, is yˆ, the system's estimate of the true y. The goal of the training procedure is to learn parameters W[i] and b[i] for each layer i that make yˆ for each training observation as close as possible to the true y.

This is logistic regression!



Training (Forward) Neural Networks

- Define a **loss function** (for ŷ and y)
  - Cross-entropy loss function
- Choose algorithm to minimize the **loss function**
  - Gradient descent
- Compute partial derivatives wrt. each parameter
- (1986) Error backpropagation
  a.k.a. reverse differentiation

Informatics

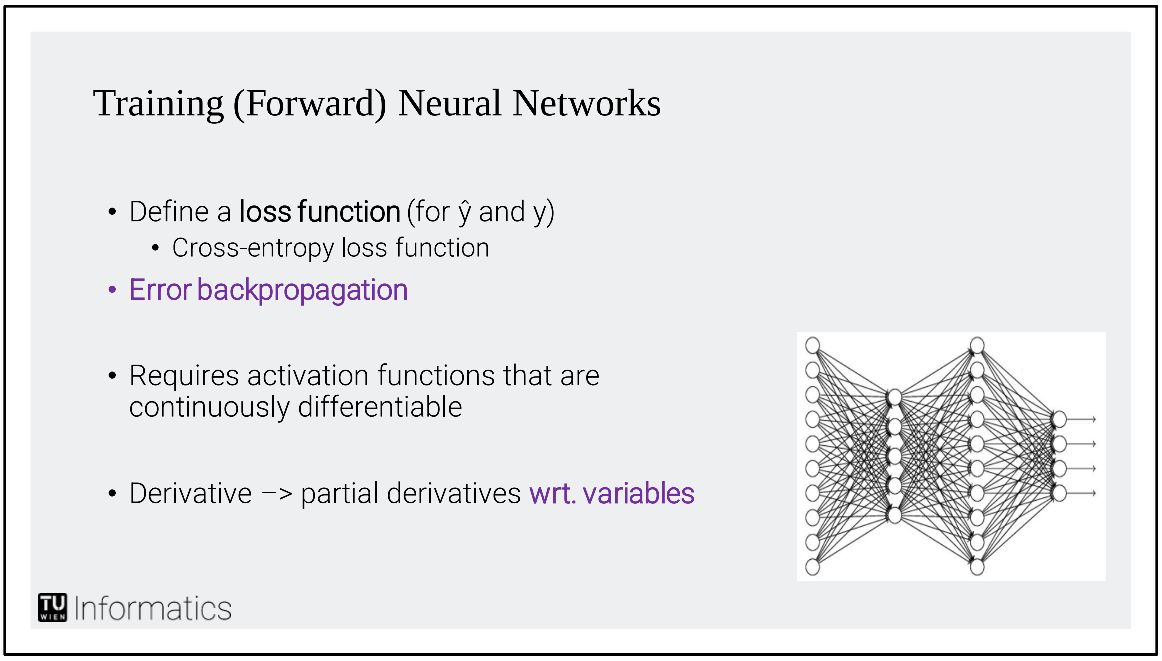
1. Select a loss function, usually the cross-entropy
2. To find the parameters that minimize this loss function, we'll use the gradient descent optimization algorithm.

But the **gradient descent** requires knowing the **gradient** of the loss function, the vector that contains the partial derivative of the loss function with respect to each of the parameters.

This is where things are a bit more complex compared to logistic regression. Because in LRegr you could directly compute the derivative of the loss function wrt. any parameter – so it was easy to tune the parameters of the regression such that the difference between ^y and y got smaller.

But for neural networks, with millions of parameters in many layers, it's much harder to see how to compute the partial derivative of some weight in layer 1 when the loss is attached to some much later layer.

How do we distribute the loss measured at the final output layer across the weights of the different layers? By **error back propagation (aka. Reverse differentiation)**

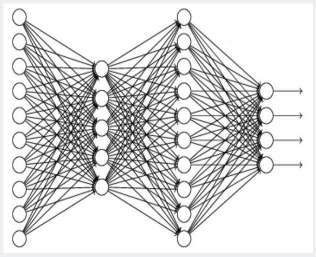


Training (Forward) Neural Networks

- Define a **loss function** (for ŷ and y)
  - Cross-entropy loss function
- Error backpropagation

- Requires activation functions that are continuously differentiable

- Derivative –> partial derivatives wrt. variables

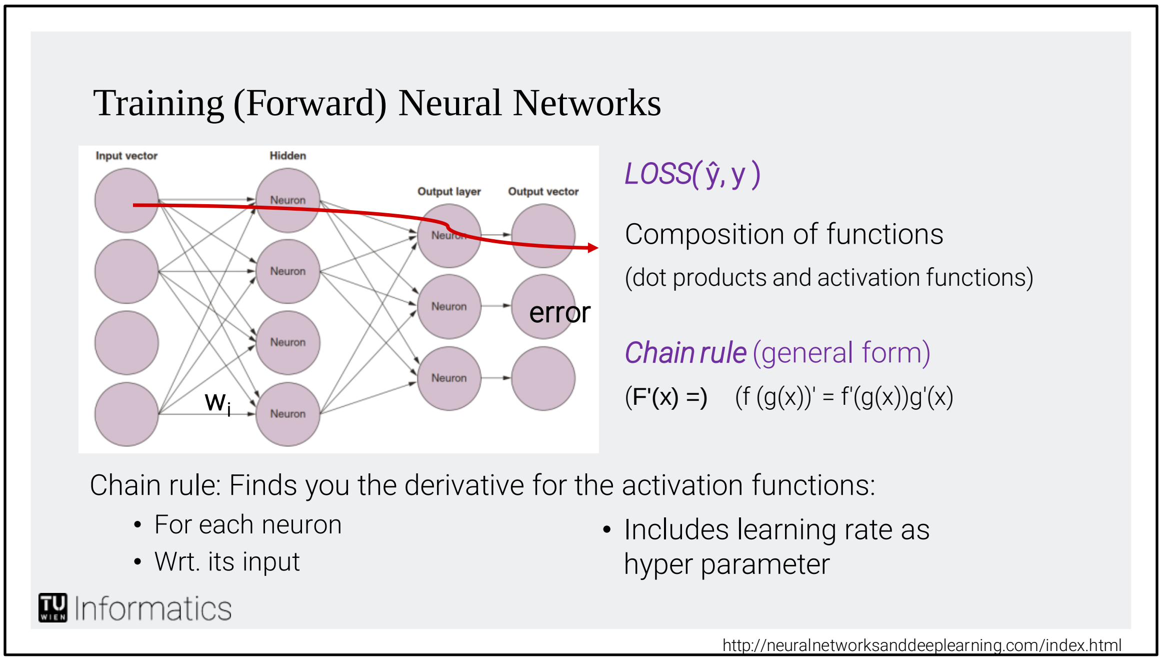


TU WIEN Informatics

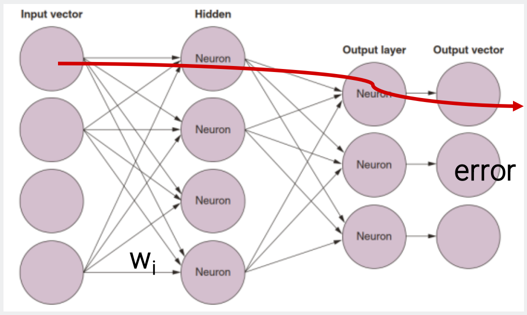
backpropagation requires an activation function that is nonlinear and continuously differentiable.

So why differentiable? If you can calculate the derivative of the function, you can also do partial derivatives of the function, with respect to various variables in the function itself.

The hint of the magic is “with respect to various variables.” You have a path toward updating a weight with respect to the amount of input it received!



Training (Forward) Neural Networks



LOSS(ŷ, y)

Composition of functions

(dot products and activation functions)

Chain rule (general form)

(F'(x) =) (f (g(x))' = f'(g(x))g'(x)

Chain rule: Finds you the derivative for the activation functions:

- For each neuron
- Wrt. its input

- Includes learning rate as hyper parameter

Informatics

http://neuralnetworksanddeeplearning.com/index.html

Usually start with the error of the network and apply a loss function to it (Mean Squared Error, or Cross Entropy Loss)

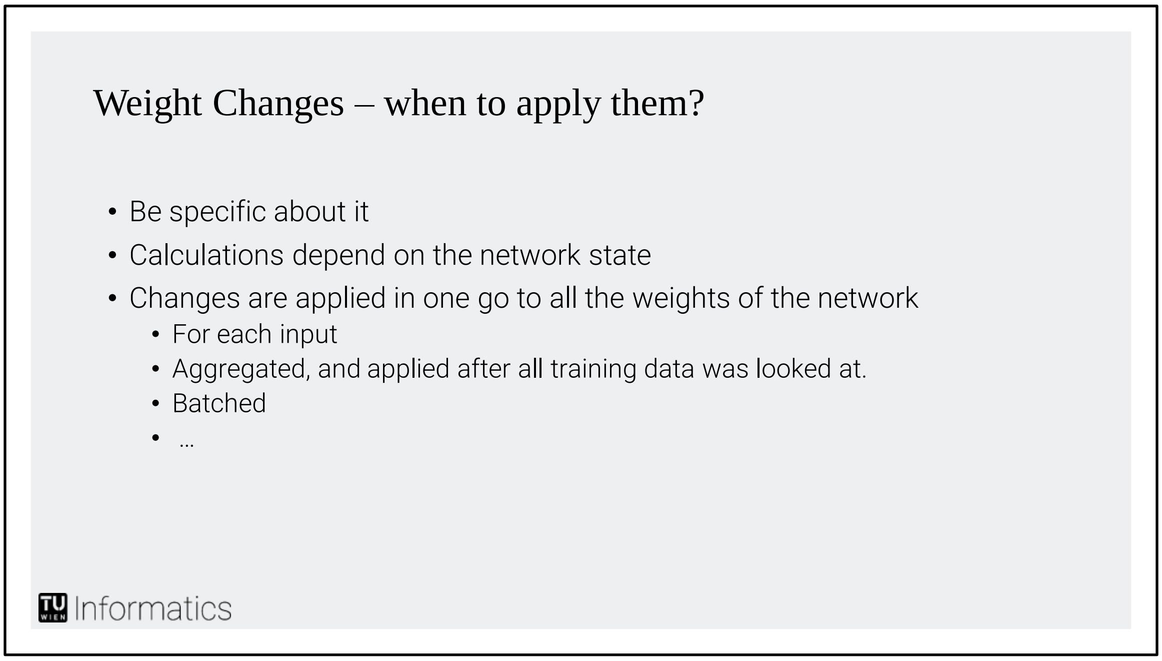
Note that the output is given by a composition of functions, more specifically dot products followed by activation functions, at each step! At each layer!

You can now use this formula to find the derivative of the activation function of **each neuron with respect to the input that fed it**.

I.e. You can calculate how much **that weight** contributed **to the final error** and adjust it appropriately.

I recommend you to study this algorithm in one of the indicated literature texts.

The learning rate parameter is part of the derivative computation.


Weight Changes – when to apply them?

- Be specific about it
- Calculations depend on the network state
- Changes are applied in one go to all the weights of the network
  - For each input
  - Aggregated, and applied after all training data was looked at.
  - Batched
  - …

TU WIEN Informatics

It’s important to be specific about when the changes are applied to the weights themselves.

As you calculate each weight update in each layer, the calculations all depend on the network’s state during the forward pass. That is on which training example is currently “in” computation by the network.

Once the error is calculated, you then calculate the proposed change to each weight in the network. But you don’t immediately apply then, otherwise, if you would update the weights at each derivation computation step (going backwards in the network) the derivatives for the lower levels will no longer be appropriate for that particular input.

You can aggregate all the ups and down for each weight based on each training sample, without updating any of the weights and instead update them at the end of all the training.

Training Feed-Forward Neural Networks

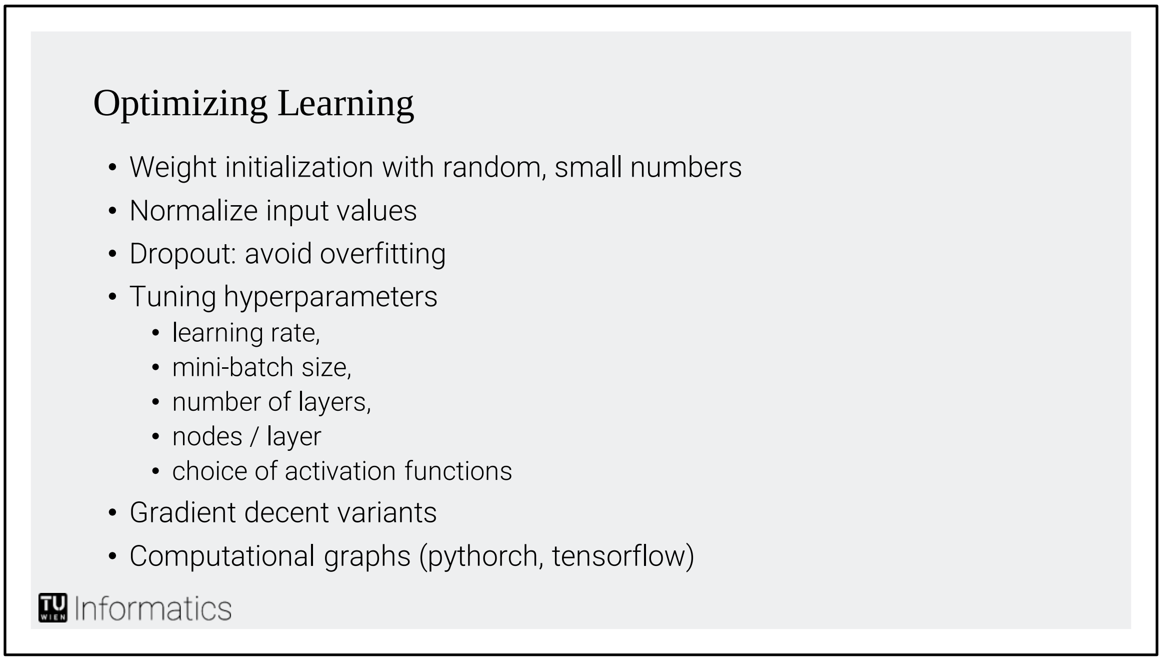
1. Pass in all the inputs.
2. Get error for each input.
3. Backpropagate errors to each of the weights.
4. Update each weight with the total change in error

Steps 1.-4. for all training data

- EPOCH
- Can pass the data again -> new refinements

- Overfitting!

TU WIEN Informatics


Optimizing Learning

- Weight initialization with random, small numbers
- Normalize input values
- Dropout: avoid overfitting
- Tuning hyperparameters
  - learning rate,
  - mini-batch size,
  - number of layers,
  - nodes / layer
  - choice of activation functions
- Gradient decent variants
- Computational graphs (pythorch, tensorflow)

Informatics

Optimizing NNs is a non-convex optimization problem – more complex than the logistic regression.

Various forms of regularization are used to prevent overfitting. One of the most important is dropout: randomly dropping some units and their connections from the network during training

Gradient descent itself also has many architectural variants.

Finally, most modern neural networks are built using computation graph formalisms that make it easy and natural to do gradient computation and parallelization onto vector-based GPUs (Graphic Processing Units).

-

Recap

- Vector Semantics & Embeddings
  - Lexical and Vector Semantics
  - Words as Vectors
  - Measuring similarity & tf-idf
  - Word2Vec
- Neural Networks
  - Perceptron, units, activation functions
  - Feed forward
  - Training
- Neural Language Models

Informatics