Deep Learning for NLP

The lecture starts at 14:15

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

Contents

Vector Semantics & Embeddings

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)

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 …”

Definition

> “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. “

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)

Observations:

- 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.

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)

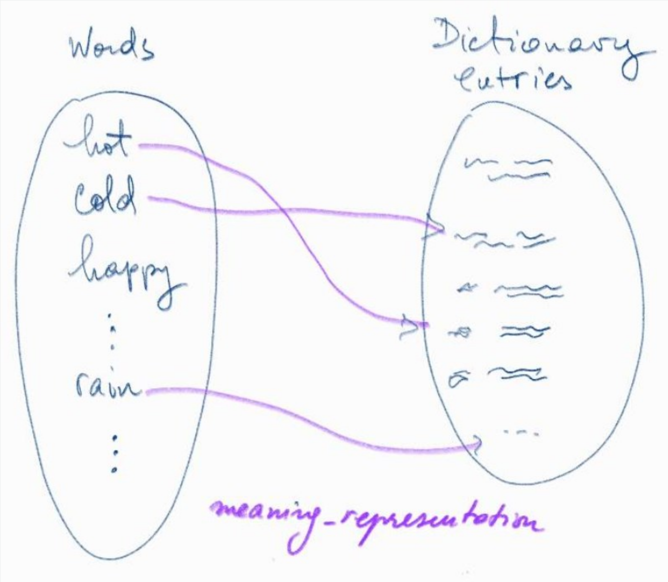
Lexical Semantics

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**

Lexical Semantics

-> Model of the **meaning**

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 - 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**

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: <u>difference in form associated with difference in meaning</u>

Lexical Semantics – Antonymity

- Opposite senses:
  - up / down   hot / cold   in / out

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

Opposite senses wrt. ONE feature of meaning.

Lexical Semantics – Similarity

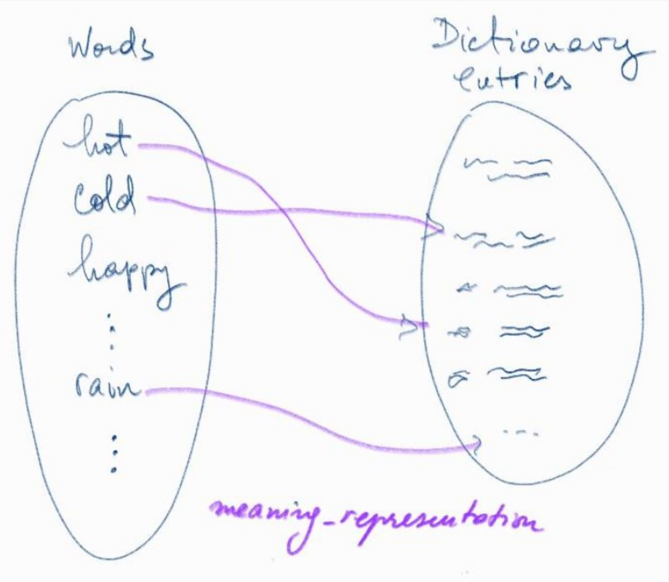
Similar meaning, but not synonyms

```
car, bicycle
cow, horse
```

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

Sense vs. sense
Word vs. word



Words Senses

Lexical Semantics - Relatedness

Words are related by

- Semantic fields
- …

`car, bicycle`: **similar**

`coffee, cup` What's the relationship?

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

Lexical Semantics - Relatedness

Words are related by

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

`car, bicycle`: **similar**

`coffee, cup` What's the relationship?

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

-> topic models (LDA)

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`.

→ Relationships are usually modelled with ontologies

Lexical Semantics – Connotations

Words have affective meanings

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

positive evaluation (great, love)

negative evaluation (terrible, hate).

But: “terribly good!”

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)

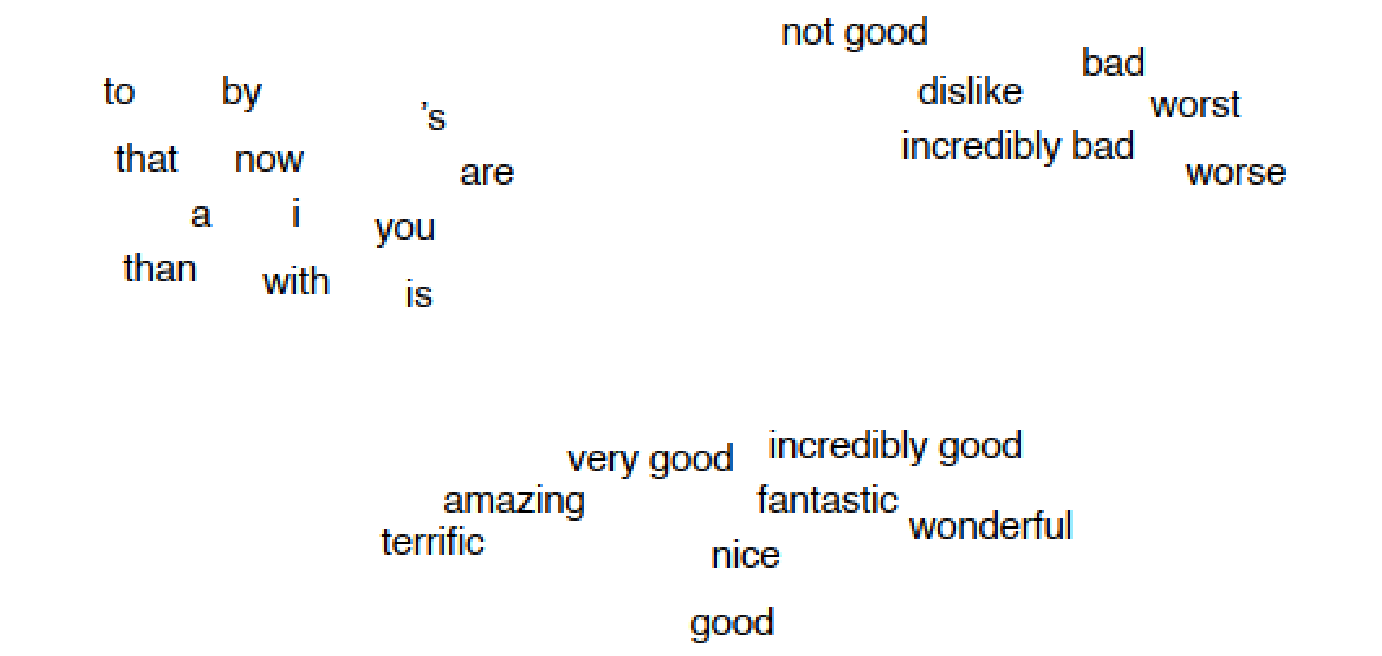
	Valence	Arousal	Dominance
courageous	8.05	5.5	7.38
music	7.67	5.57	6.5
heartbreak	2.45	5.65	3.58
cub	6.71	3.95	4.24
life	6.68	5.59	5.89

Osgood et al. (1957)

Vector Semantics

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

- Standard in NLP



Types of Embeddings

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

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

From Words to Vectors

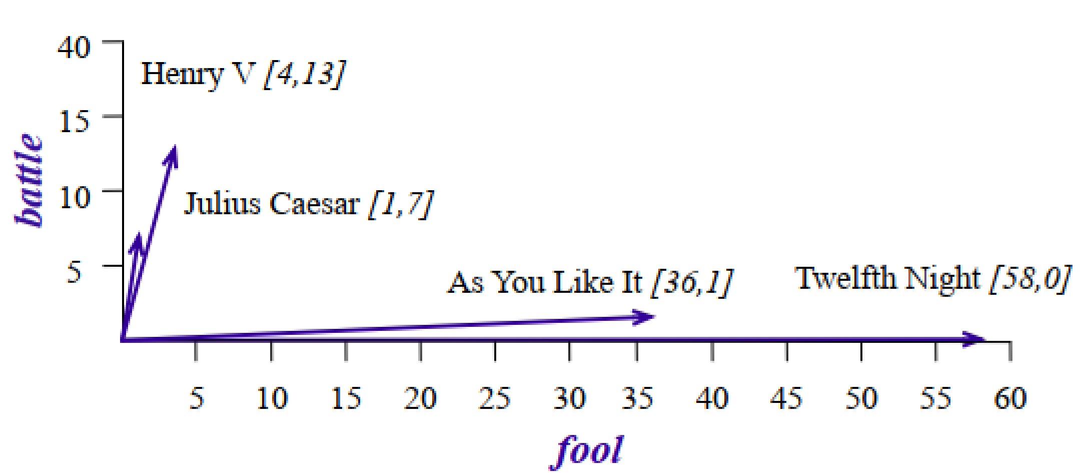
Based on co-occurrence counts

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

Documents as vectors

- Vectors similar for the two comedies



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

Documents as vectors

Word–word matrix (term–context matrix) ← more common!

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

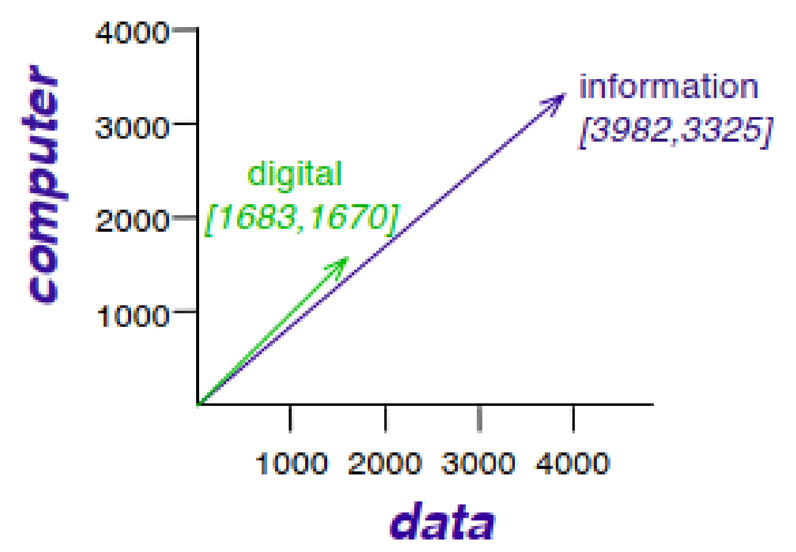
	aardvark	...	computer	data	result	pie	sugar	...
cherry	0	...	2	8	9	442	25	
strawberry	0	...	0	0	1	60	19	
digital	0	...	1670	1683	85	5	4	
information	0	...	3325	3982	378	5	13	

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

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	

Cosine example

However ….

raw-frequencies are

- skewed
- non-discriminative

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

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

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

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:

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

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

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:

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

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

Recap

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

  - Word2Vec

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)

Dense Vectors – Word2Vec

Why dense vectors?

- easier to use as features in machine learning (short → fewer 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

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/

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

Brilliant Insight!

Use running text as implicitly supervised training data!

Take a word *s* 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)

Word2Vec: Skip-Gram

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

1. Treat the target word and a neighbouring 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 of the trained model 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:

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

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

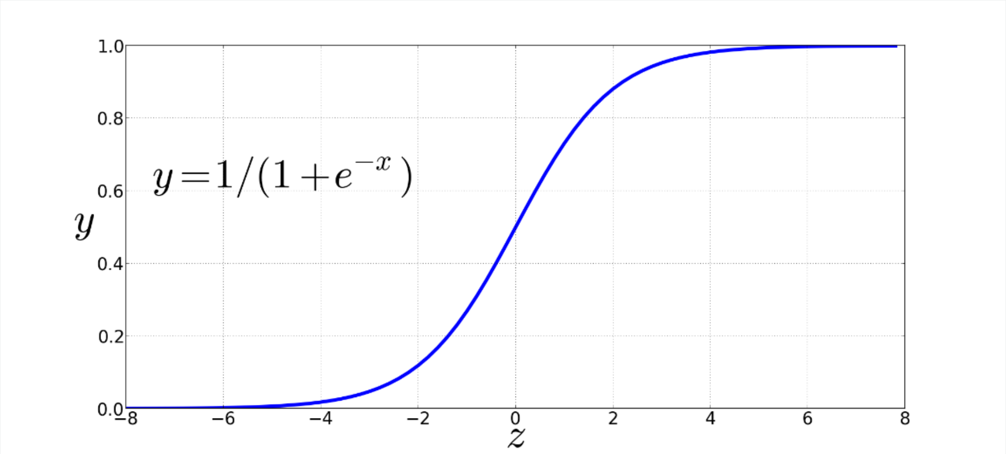
How to compute P(+ | t,c)?

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

Problem:

Dot product is not a probability!

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



Turning dot product into a probability

Apply the sigmoid function to the output of the dot product => probabilities

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

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

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: words are independent of each other!

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

Training Phase

Given:

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

Goal: Adjust embeddings such that

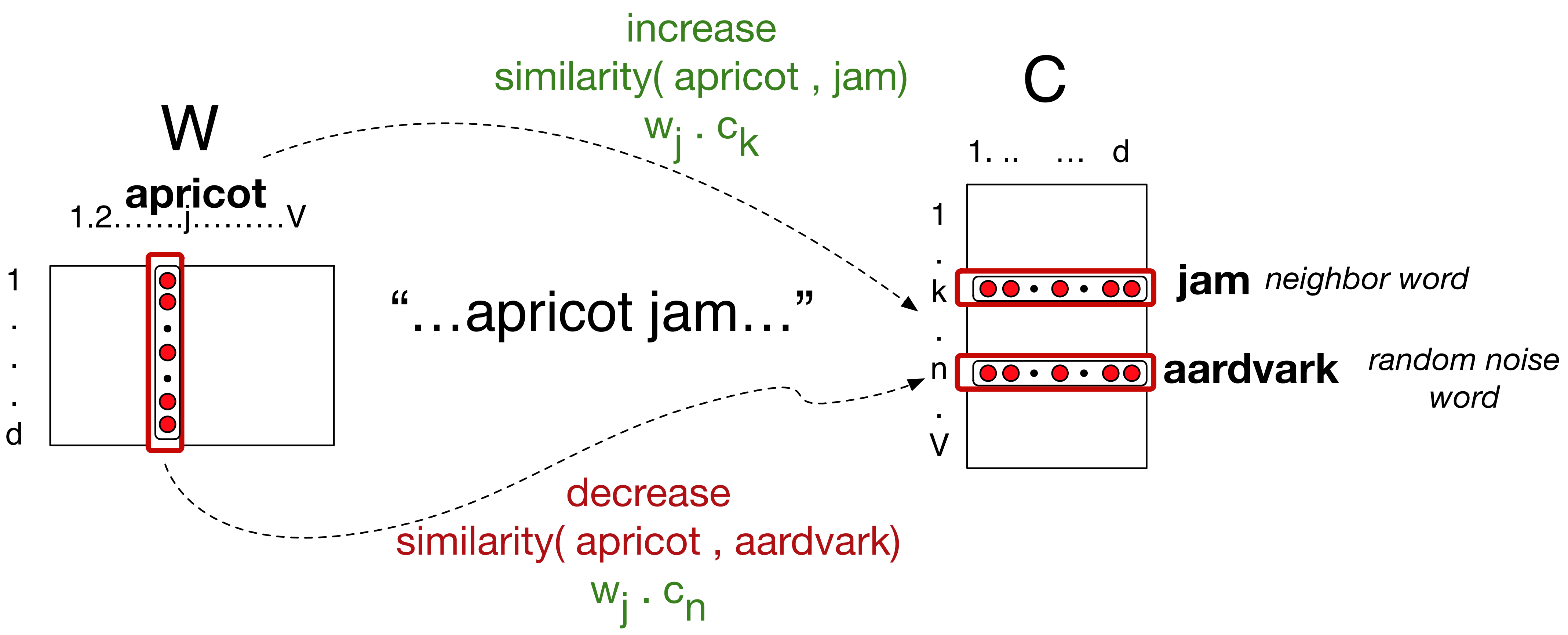
- Positive (target, context) instance similarity is maximised
- Necative (target, context) instance similarity is minimized

Formally:

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

Use Gradient Descent

Training Phase



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)

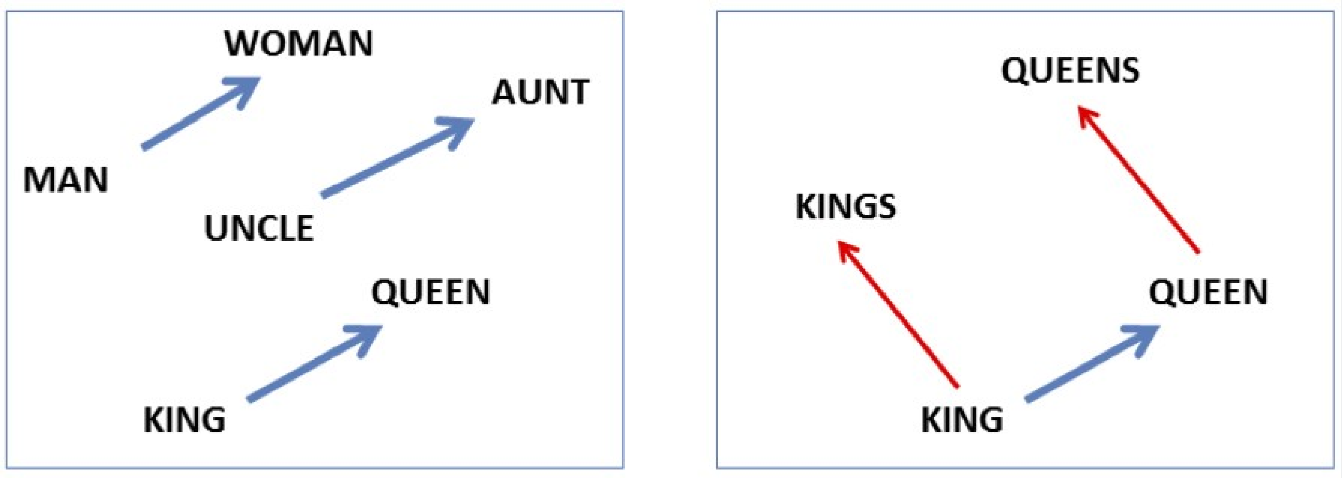
- Throw away the classifier code and keep the embeddings (regression weights!)

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’)

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

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.

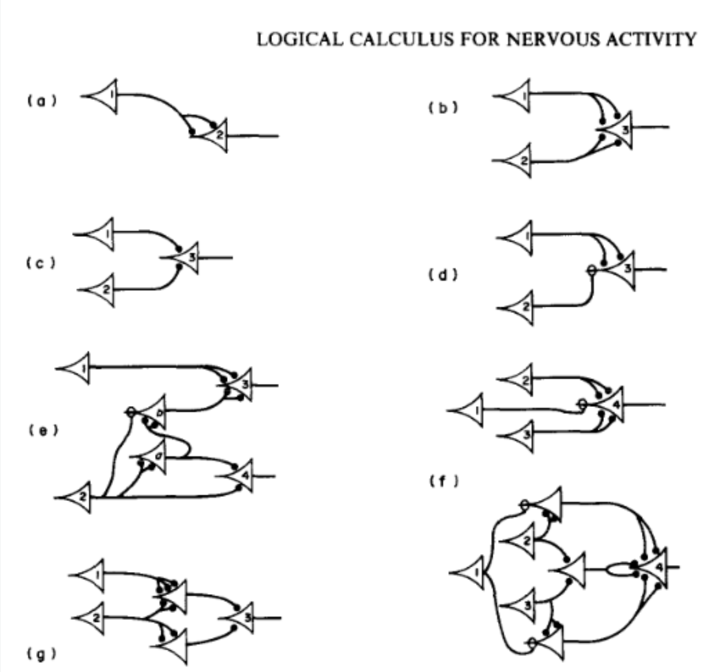
Recap

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

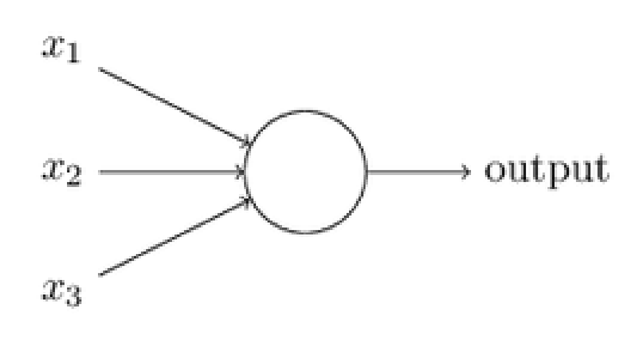
Neural Networks

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



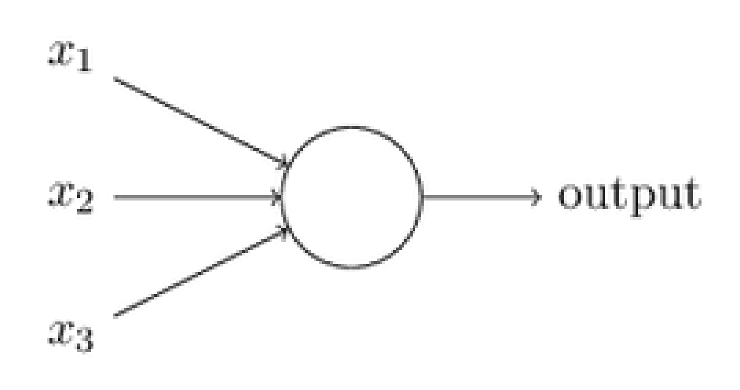
- 1950s and '60s
  perceptron



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

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*

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

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 three (four) factors:

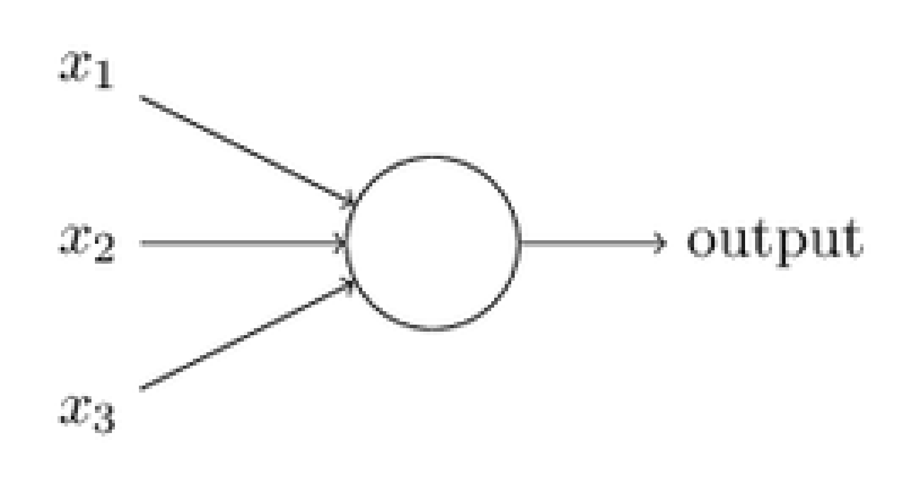
1. Is the weather good?
2. Does your friend/partner want to accompany you?
3. Is the festival near public transit? (You don't own a car)

Perceptron – a Device that Takes a Decision

1. Is the weather good?

2. Friend/partner Joining?

3. Public transportation



output = {1/go, 0/no_go}

Perceptron – a Device that Takes a Decision

1. Is the weather good?

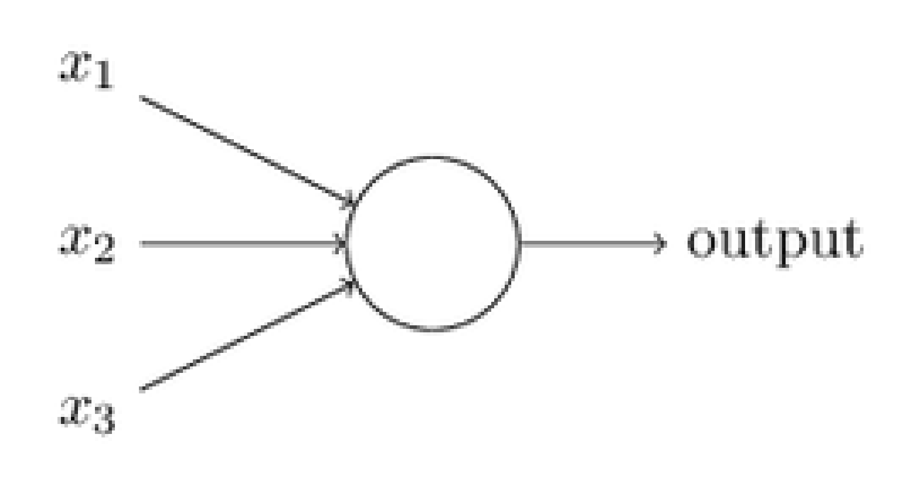
    0 – bad, 1 – good

2. Friend/partner Joining?

    0 – no, 1 – yes

3. Public transportation

    0 – no, 1 – yes



output = {1/go, 0/no_go}

Perceptron – a Device that Takes a Decision

1. Is the weather good?

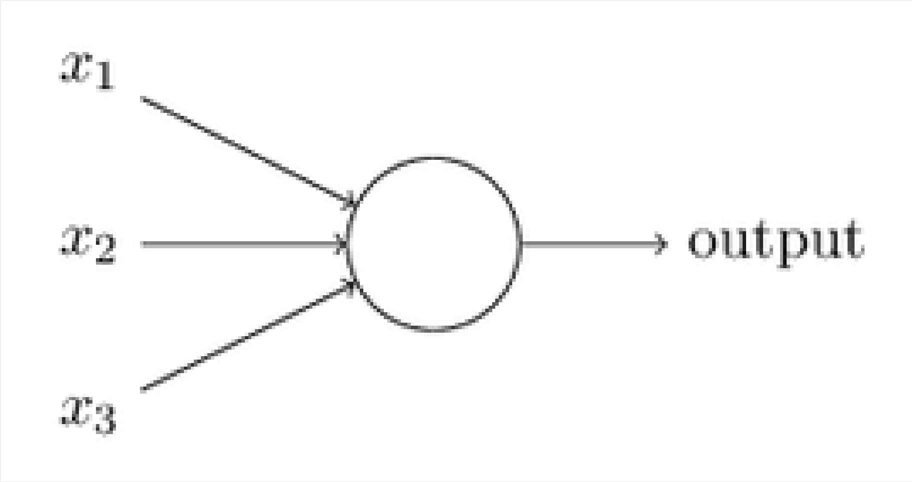
    0 – bad, 1 – good, *w_1 = 6*

2. Friend/partner Joining?

    0 – no, 1 – yes *w_2 = 2*

3. Public transportation

    0 – no, 1 – yes *w_3= 2*



output = {1/go, 0/no_go}

threshold = 7

compute $\sum_j w_j x_j$

Perceptron – a Device that Takes a Decision

1. Is the weather good?

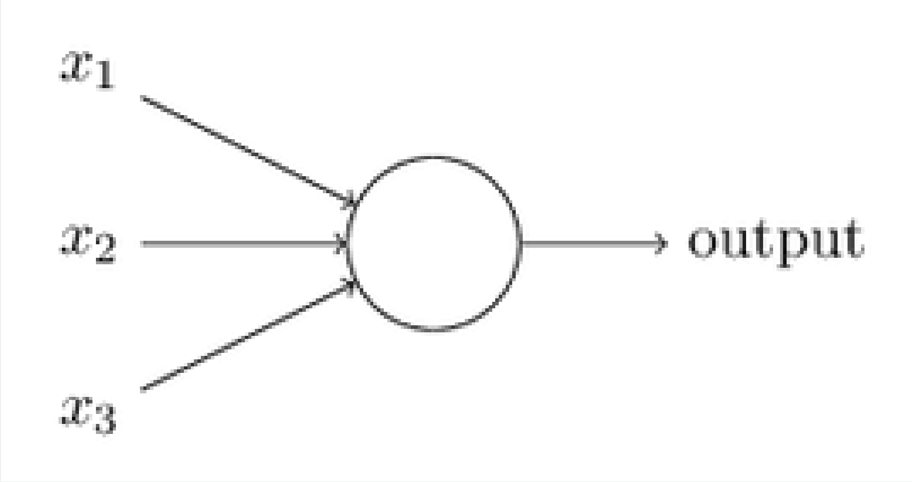
   0 – bad, **1 – good**, *w_1 = 6*

2. Friend/partner Joining?

   0 – no, **1 – yes** *w_2 = 2*

3. Public transportation

   0 – no, 1 – yes *w_3= 2*



output = {1/go, 0/no_go}

threshold = 7

compute $\sum_j w_j x_j$ = 8 (go)

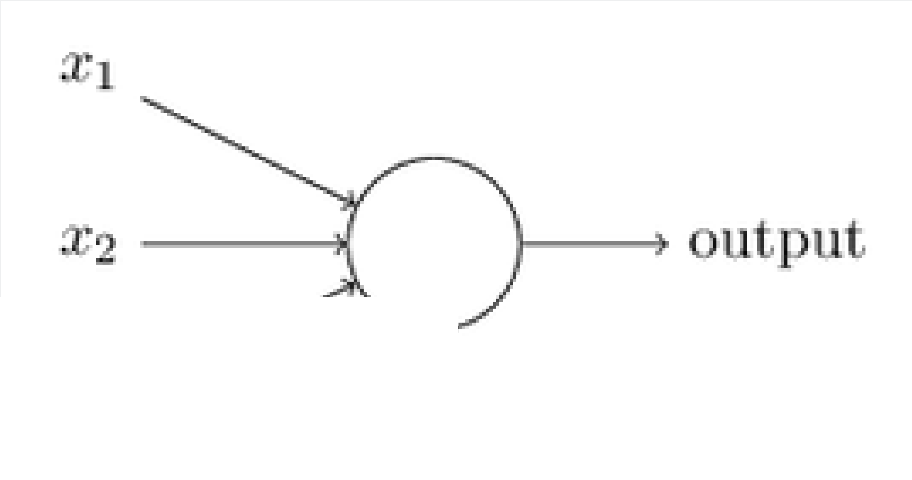
Perceptron – a Device that Takes a Decision

1. Is the weather good?

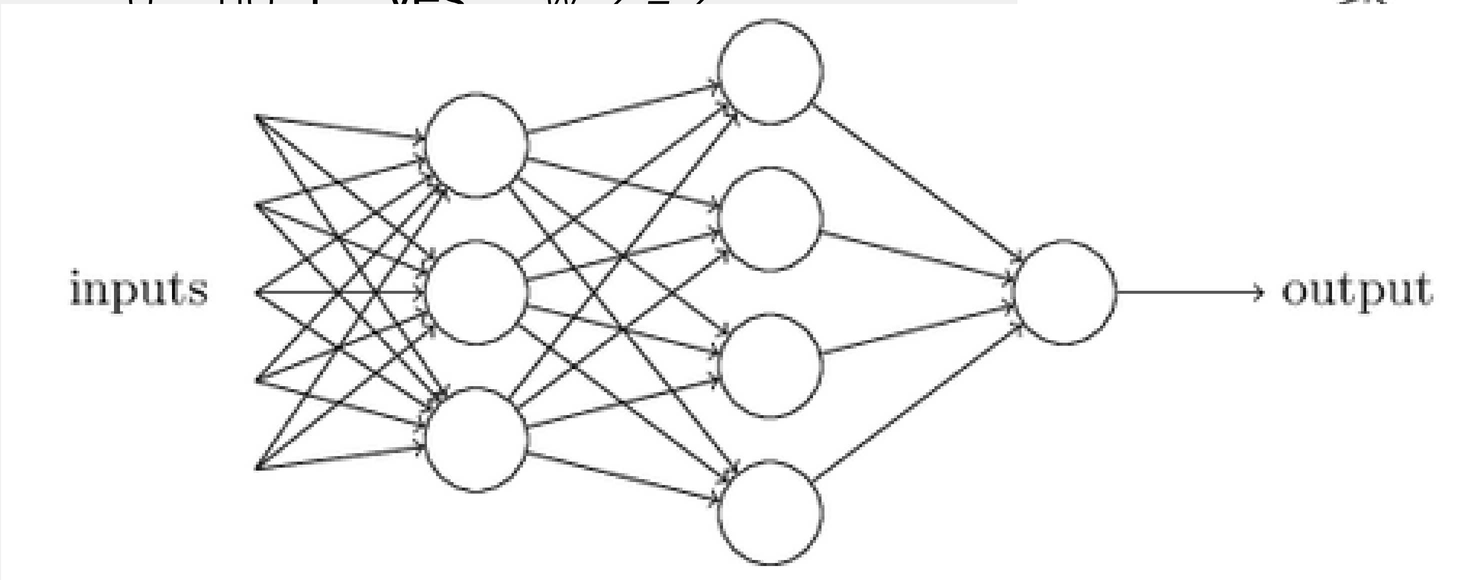
    0 – bad, **1 – good**, *w_1 = 6*

2. Friend/partner Joining?

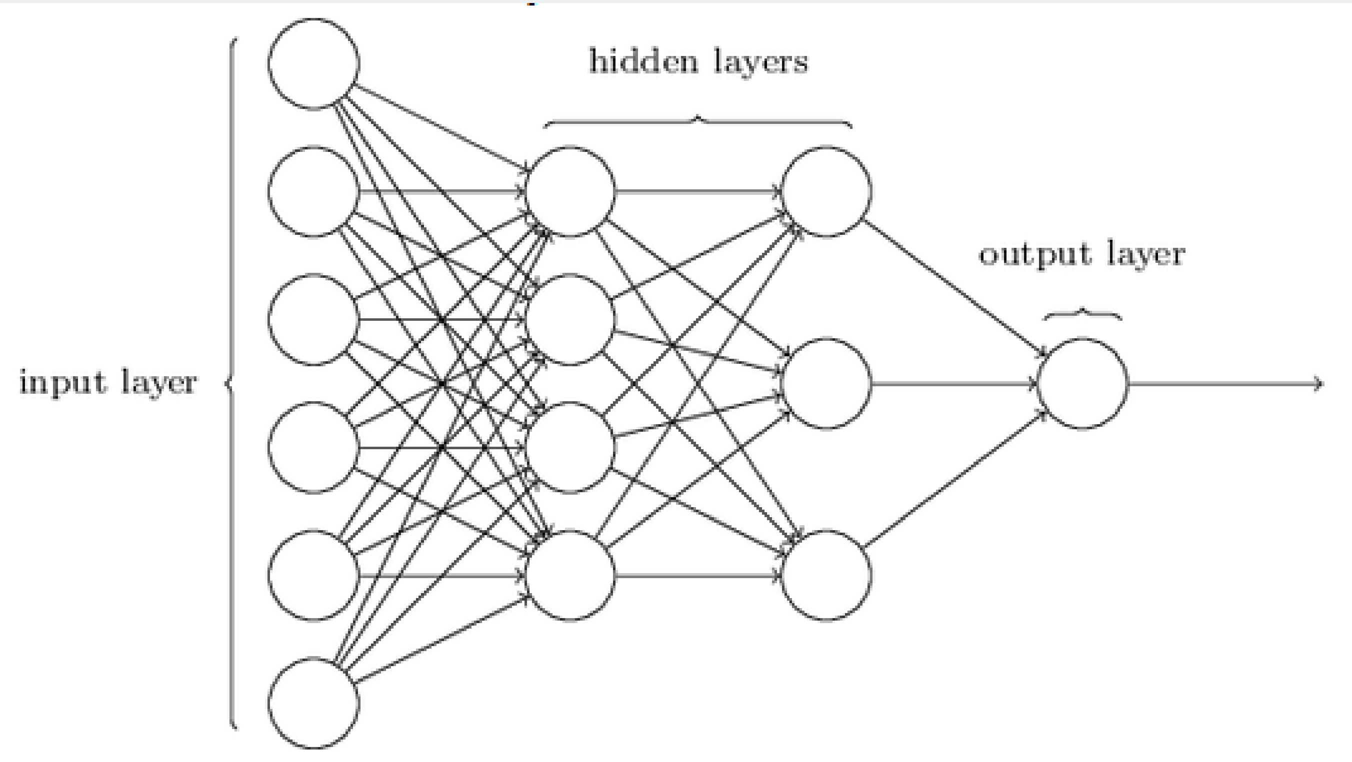
    0 – no, **1 – yes**, *w_2 = 2*



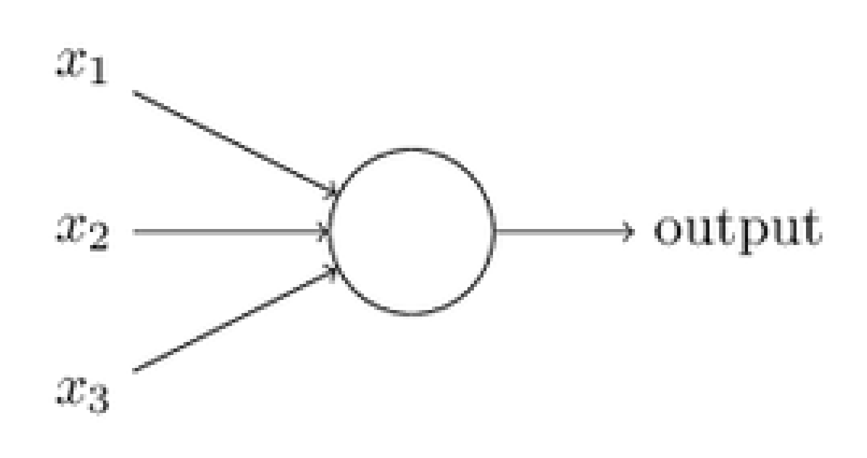
no_go}



Architecture of a (feedforward) Neural Network



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"

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

Sigmoid Neuron

Perceptrons: small changes in intput cause large changes in output

Reason: Nodes (neurons) have only two states: 0 or 1

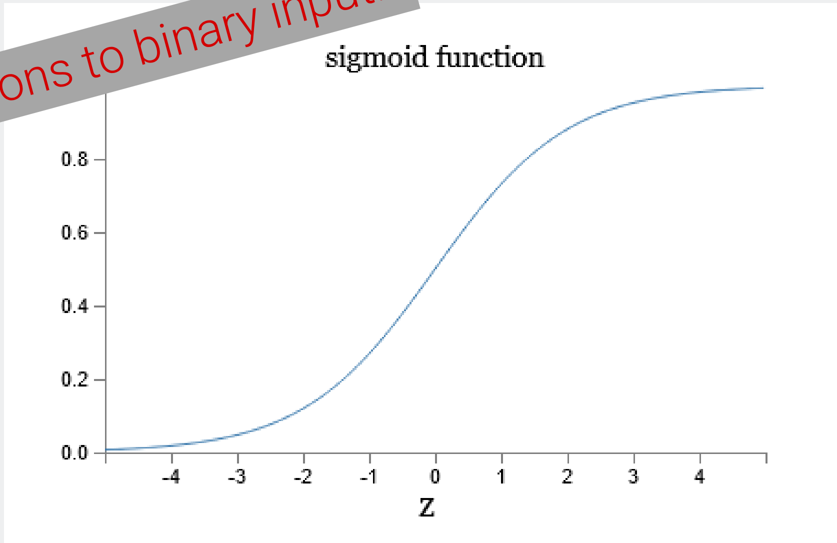
Can we output a continuum of values?

Between 0 and 1?

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

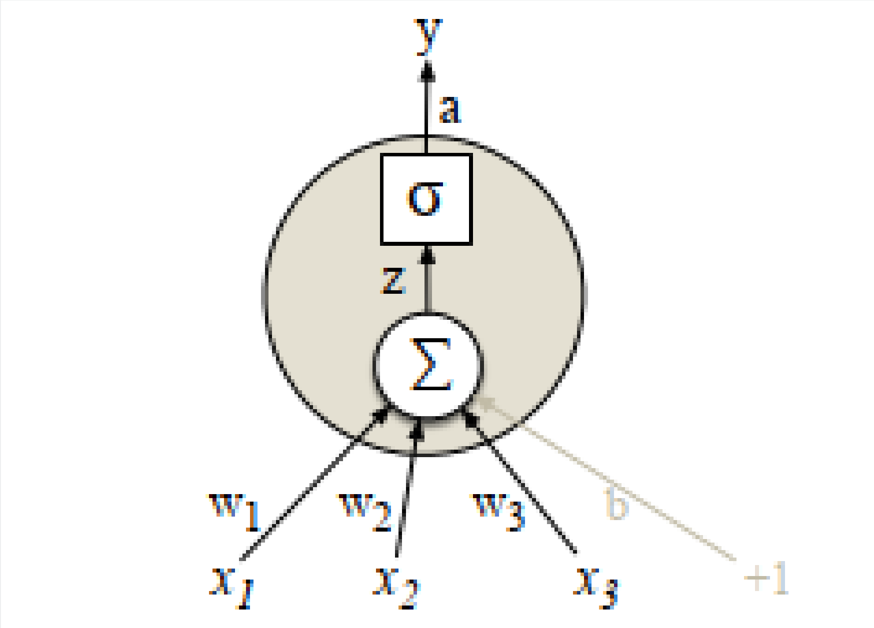
No restrictions to binary input!

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



Activation Functions

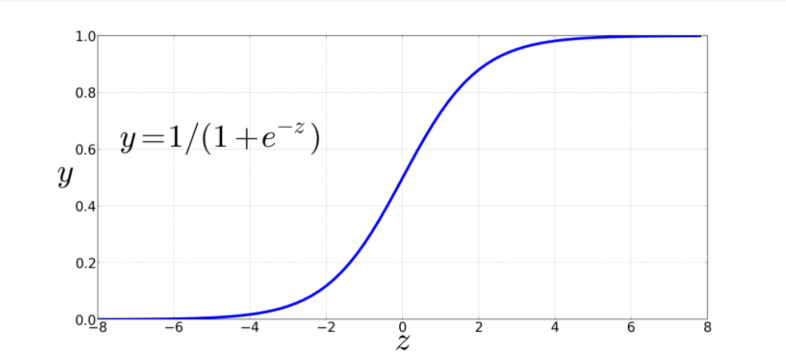
1. Sigmoid function
2. Hyperbolic tan
3. Rectified Linear Unit (ReLU)
4. Leaky Rectified Linear Unit
5. Maxout
6. …



Activation Functions

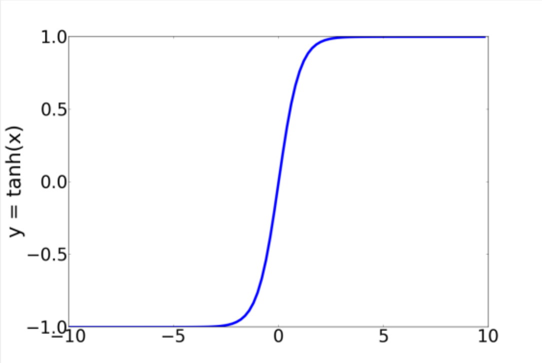
1. Sigmoid function

$$\frac{1}{1+e^{-z}}$$



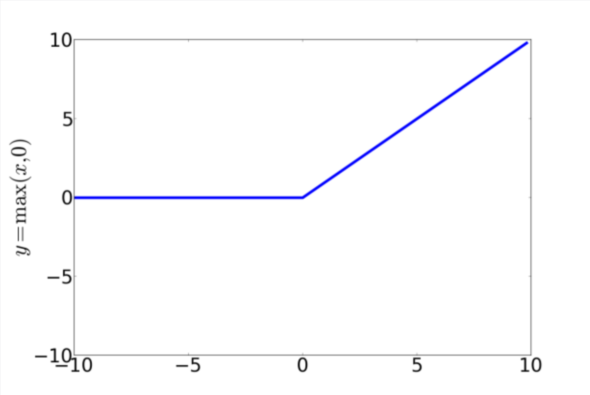
2. Hyperbolic tan

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



3. Rectified Linear Unit (ReLU)

$$max(x,0)$$



Feed-Forward Neural Networks

- Multilayer network
- Units connected without cycles

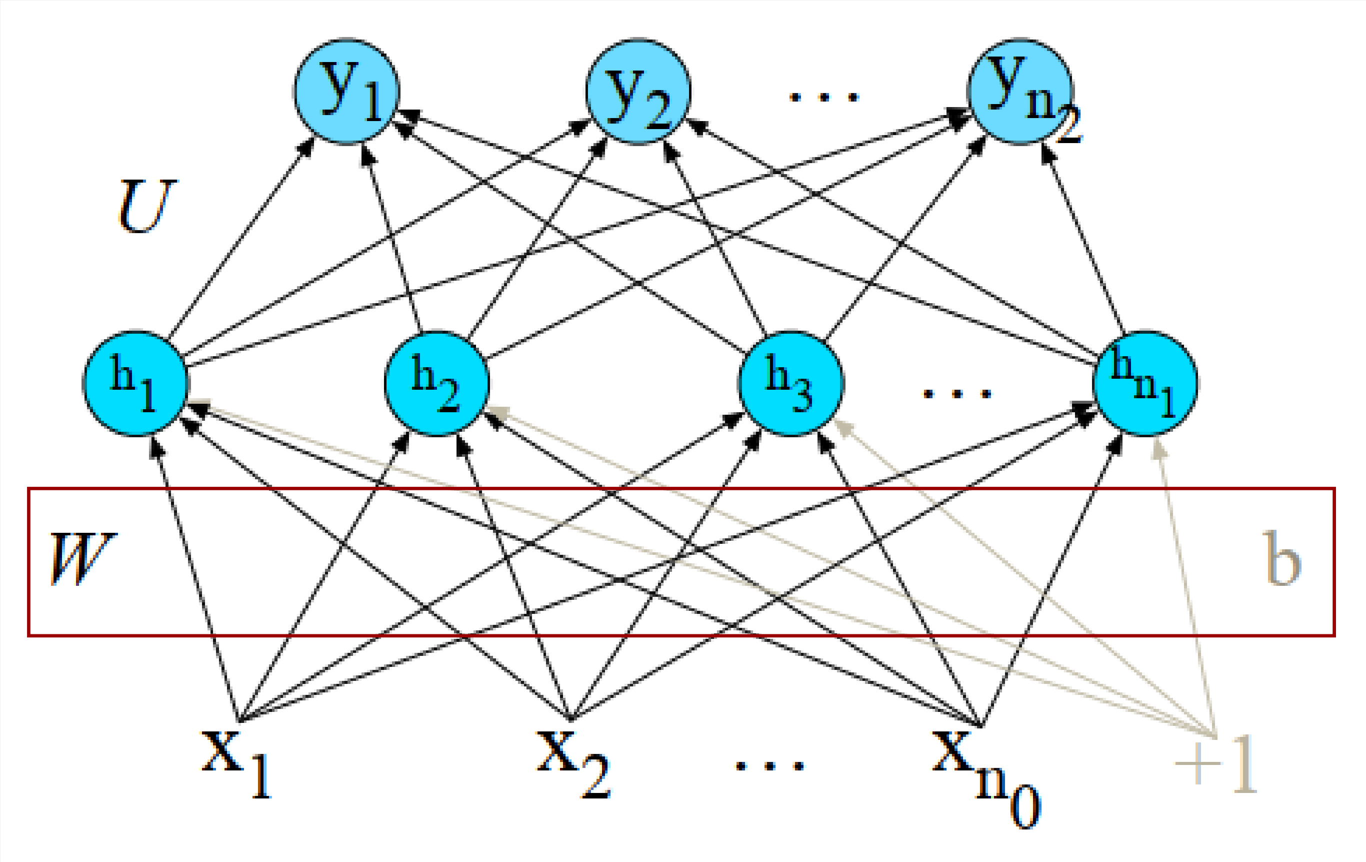
- Node types:
  - Input units
  - Hidden units
  - Output units

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)



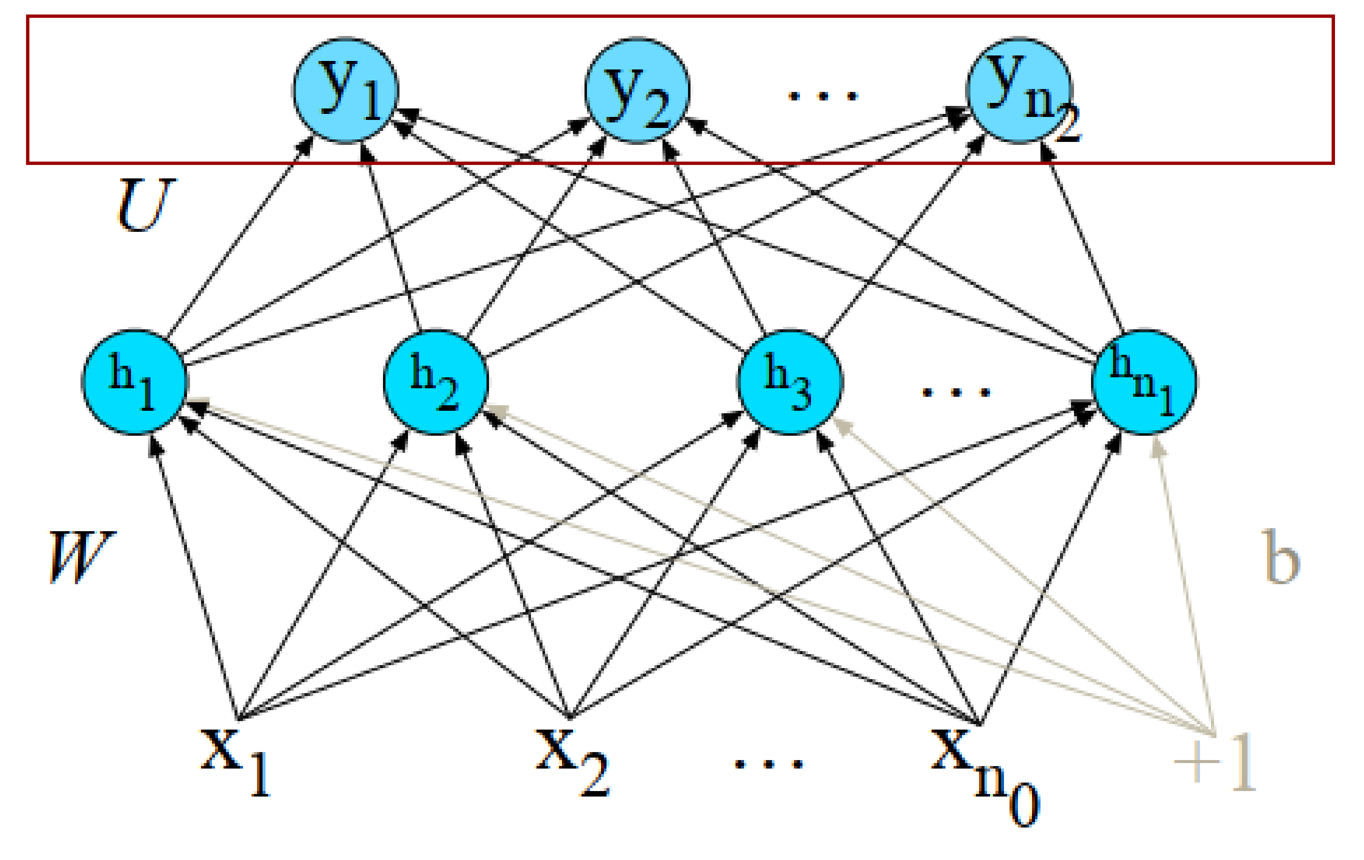
Feed-forward Neural Network

Output layer

probability distribution

Hidden layer (hypothesis)

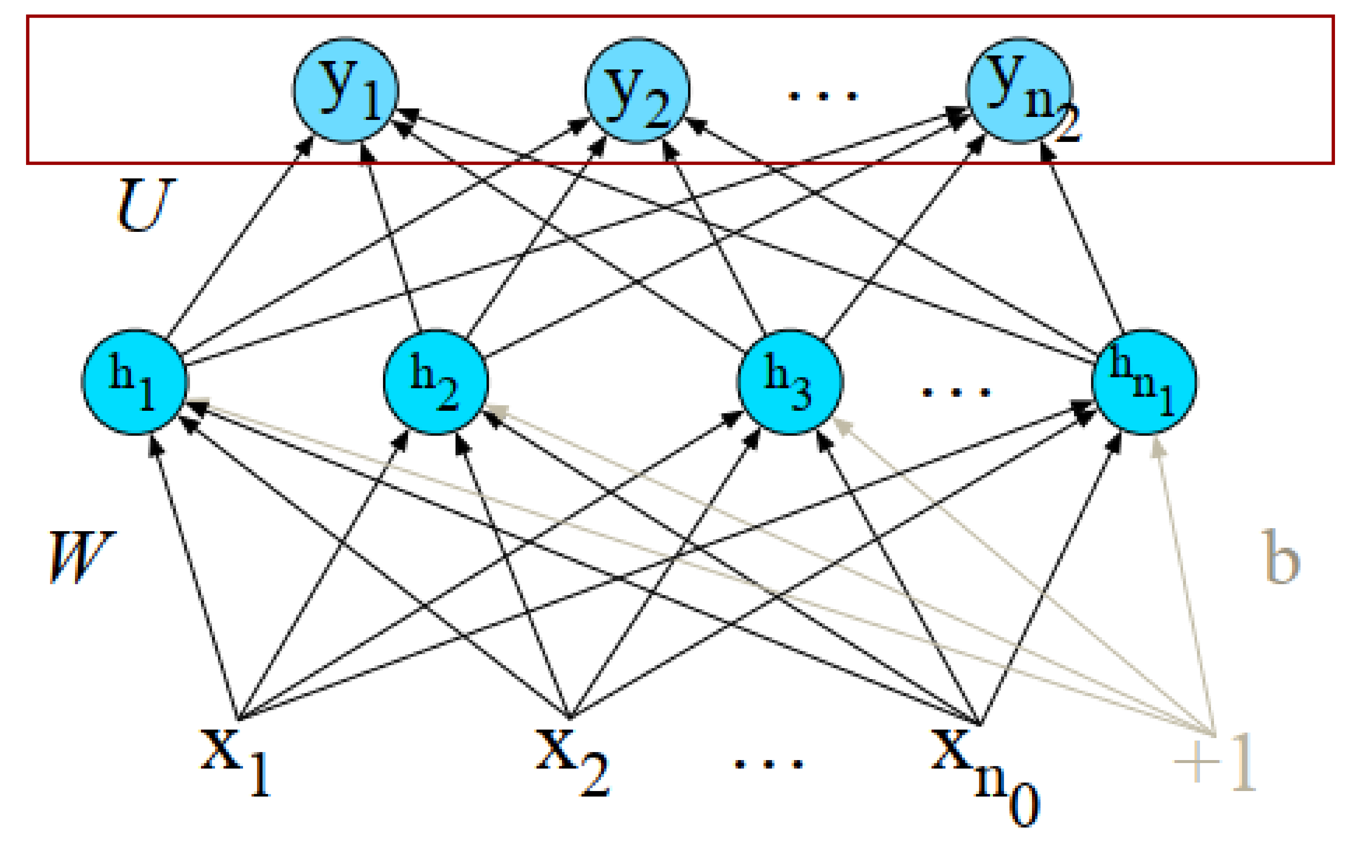
Input layer



Feed-forward Neural Network

Output layer

probability distribution



U output layer weight matrix

$z = U\,h$ – no output

Normalizing

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

Feed-forward Neural Network

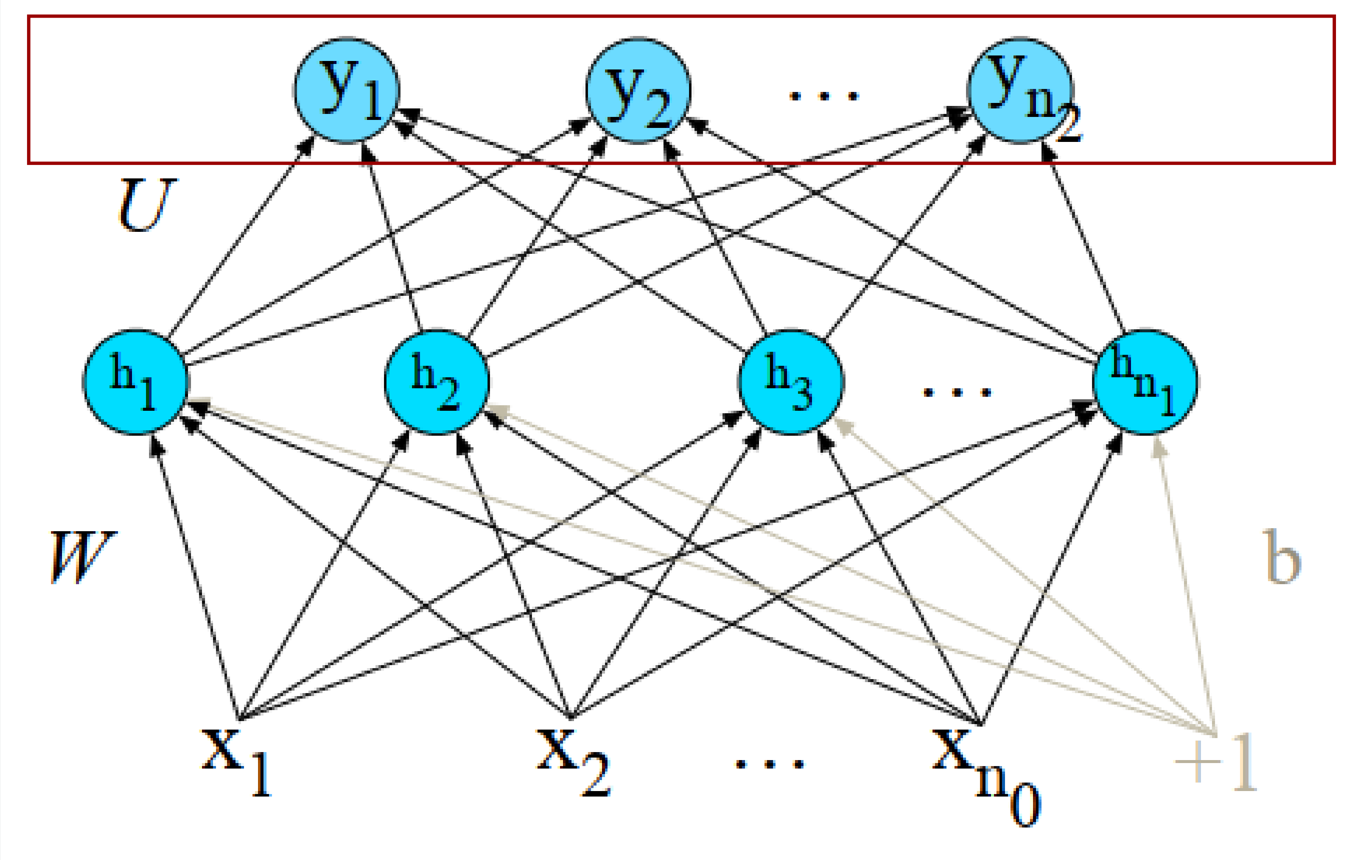
Output layer

probability distribution

Hidden layer (hypothesis)

representation of the input

Input layer



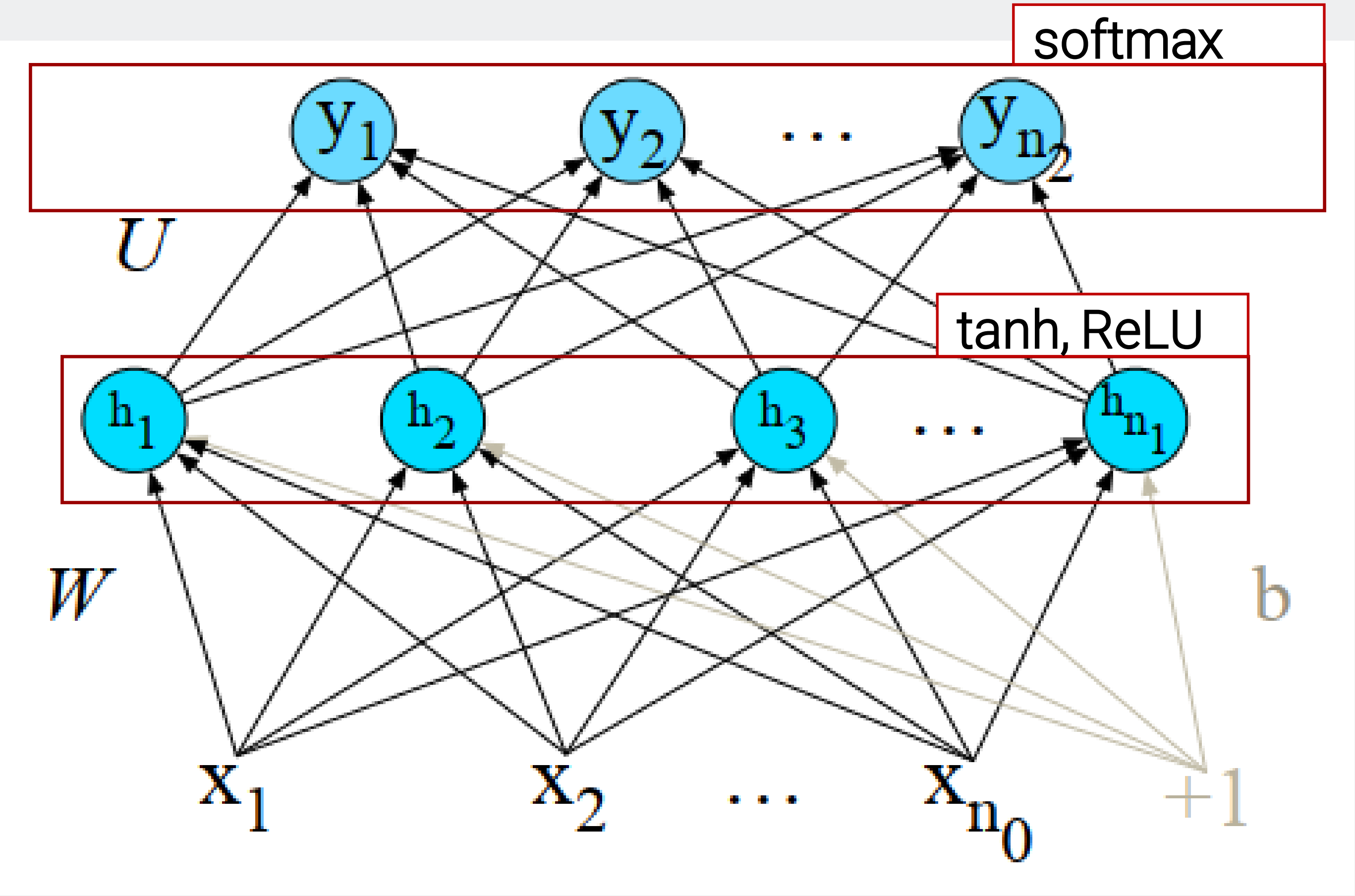
Feed-forward Neural Network

~ logistic regression:

(a) with many layers,

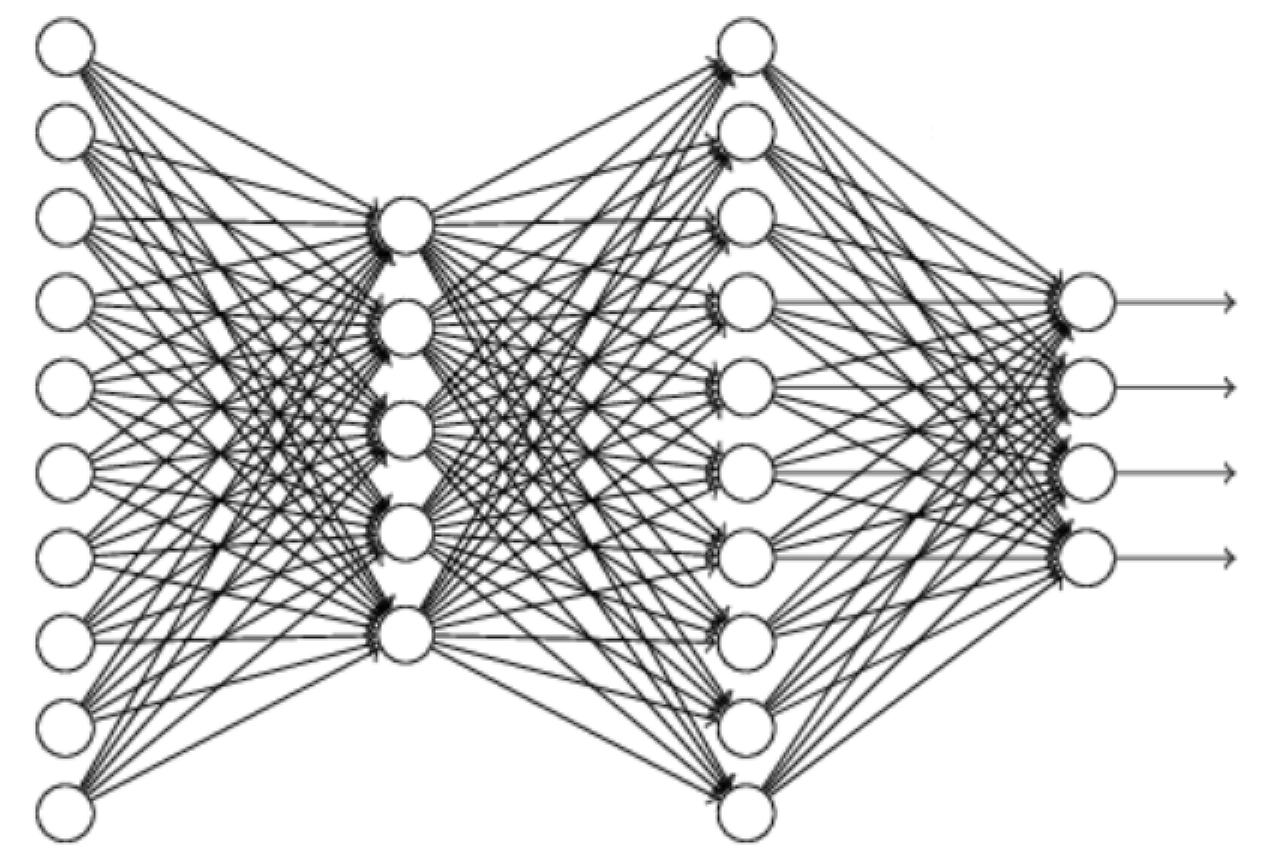
(b) induces the feature representations themselves (not "by hand").

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



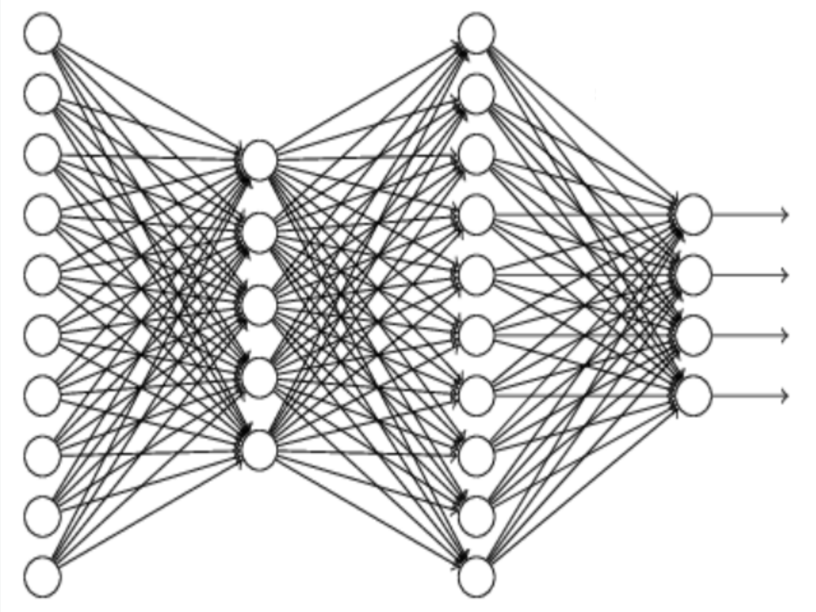
Training (Forward) Neural Networks

- 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)



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
  - Problem: loss computed in last layer, where to distribute the error along the many layers?



- (1986) Error backpropagation

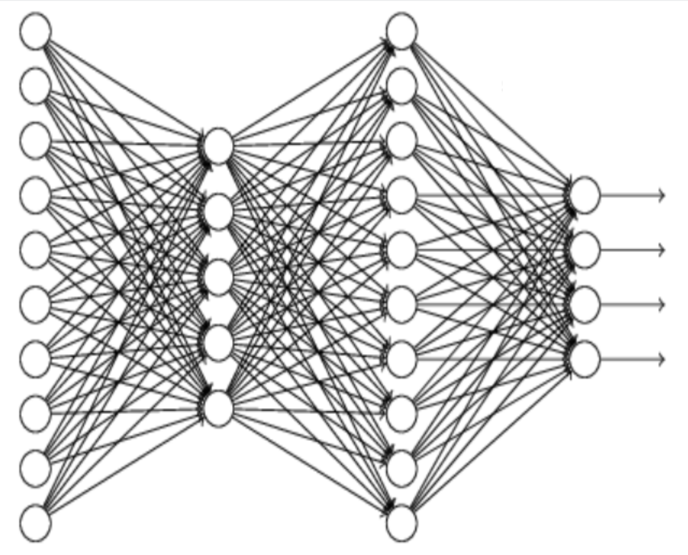
  a.k.a. reverse differentiation

Training (Forward) Neural Networks

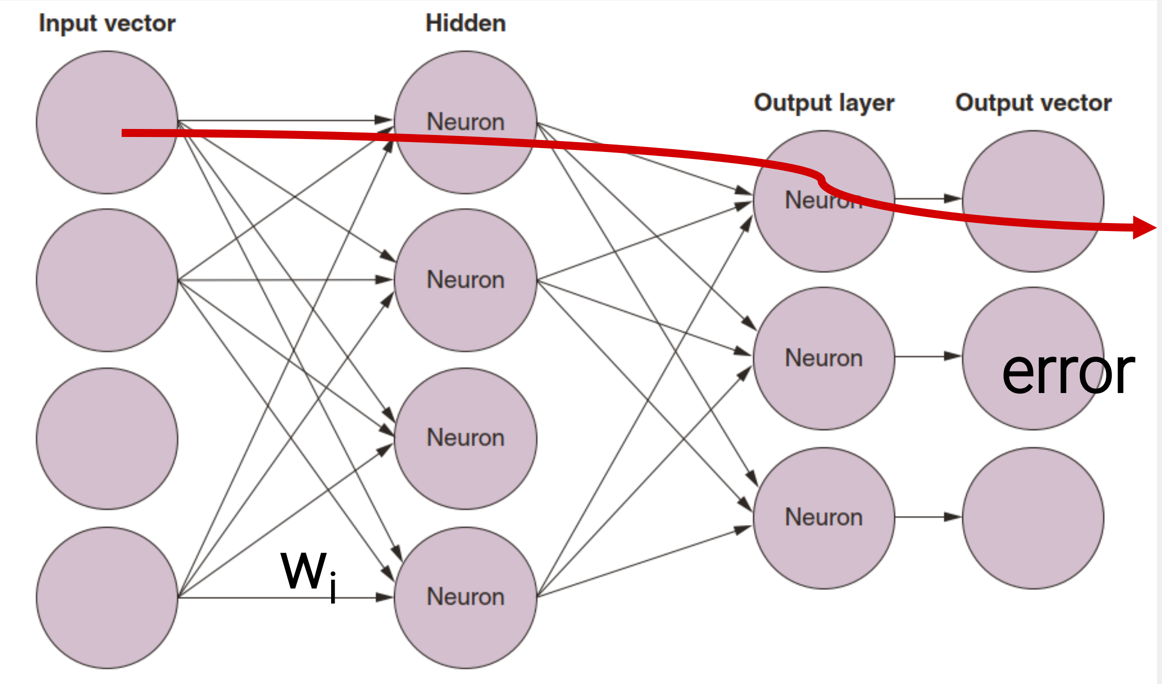
- Define a **loss function** (for ŷ and y)
  - Cross-entropy loss function
- Error backpropagation (originating from computation graphs)

- Requires activation functions that are continuously differentiable

- Derivative –> partial derivatives wrt. variables



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

http://neuralnetworksanddeeplearning.com/index.html

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
  - …

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!

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)

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