Natural Language Processing

Neural networks

Yulia Tsvetkov

yuliats@cs.washington.edu

With many slides by Dan Jurafsky

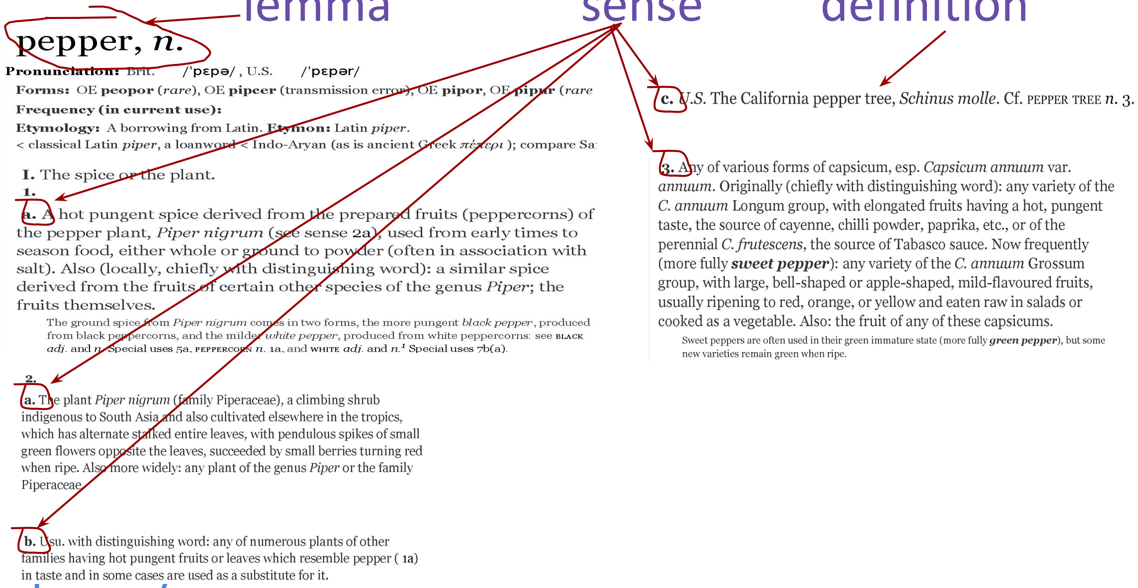
PAUL G. ALLEN SCHOOL OF COMPUTER SCIENCE & ENGINEERING

Readings

- Neutral networks chapter in J&M 3
- Advanced tutorial
- Hundreds of blog posts and tutorials

Lexical semantics

- How should we represent the meaning of the word?
  - Words, lemmas, senses, definitions



pepper, *n.*

Pronunciation: Brit. /ˈpɛpə/ , U.S. /ˈpɛpər/

Forms: OE **peopor** (*rare*), OE **pipcer** (transmission error), OE **pipor**, OE **pipur** (*rare*

Frequency (in current use):

Etymology: A borrowing from Latin. **Etymon:** Latin *piper*.

< classical Latin *piper*, a loanword < Indo-Aryan (as is ancient Greek *πέπερι*); compare Sa

I. The spice or the plant.

1.

a. A hot pungent spice derived from the prepared fruits (peppercorns) of the pepper plant, *Piper nigrum* (see sense 2a), used from early times to season food, either whole or ground to powder (often in association with salt). Also (locally, chiefly with distinguishing word): a similar spice derived from the fruits of certain other species of the genus *Piper*; the fruits themselves.

The ground spice from *Piper nigrum* comes in two forms, the more pungent *black pepper*, produced from black peppercorns, and the milder *white pepper*, produced from white peppercorns: see BLACK *adj.* and *n.* Special uses 5a, PEPPERCORN *n.* 1a, and WHITE *adj.* and *n.*[1] Special uses 7b(a).

2.

a. The plant *Piper nigrum* (family Piperaceae), a climbing shrub indigenous to South Asia and also cultivated elsewhere in the tropics, which has alternate stalked entire leaves, with pendulous spikes of small green flowers opposite the leaves, succeeded by small berries turning red when ripe. Also more widely: any plant of the genus *Piper* or the family Piperaceae.

b. Usu. with distinguishing word: any of numerous plants of other families having hot pungent fruits or leaves which resemble pepper (1a) in taste and in some cases are used as a substitute for it.

c. *U.S.* The California pepper tree, *Schinus molle*. Cf. PEPPER TREE *n.* 3.

3. Any of various forms of capsicum, esp. *Capsicum annuum* var. *annuum*. Originally (chiefly with distinguishing word): any variety of the *C. annuum* Longum group, with elongated fruits having a hot, pungent taste, the source of cayenne, chilli powder, paprika, etc., or of the perennial *C. frutescens*, the source of Tabasco sauce. Now frequently (more fully ***sweet pepper***): any variety of the *C. annuum* Grossum group, with large, bell-shaped or apple-shaped, mild-flavoured fruits, usually ripening to red, orange, or yellow and eaten raw in salads or cooked as a vegetable. Also: the fruit of any of these capsicums.

Sweet peppers are often used in their green immature state (more fully ***green pepper***), but some new varieties remain green when ripe.

http://www.oed.com/

Problems with discrete representations

- Too coarse
  - *expert ↔ skillful*
- Sparse
  - *wicked, badass, ninja*
- Subjective
- Expensive
- Hard to compute word relationships

S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced, proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe
...
S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good

expert [0 0 0 **1** 0 0 0 0 0 0 0 0 0 0 0]

skillful [0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0]

- dimensionality: PTB: 50K, Google1T 13M

Distributional hypothesis

"The meaning of a word is its use in the language"

[Wittgenstein PI 43]

"You shall know a word by the company it keeps"

[Firth 1957]

If A and B have almost identical environments we say that they are synonyms.

[Harris 1954]

Example

- Suppose you see these sentences:
  - Ongchoi is delicious sautéed with garlic.
  - Ongchoi is superb over rice
  - Ongchoi leaves with salty sauces

- And you've also seen these:
  - …spinach sautéed with garlic over rice
  - Chard stems and leaves are delicious
  - Collard greens and other salty leafy greens

Ongchoi: Ipomoea aquatica "Water Spinach"

Ongchoi is a leafy green like spinach, chard, or collard greens

空心菜
kangkong
rau muống
...

Yamaguchi, Wikimedia Commons, public domain

Model of meaning focusing on similarity

- Each word = a vector
  - not just “word” or word45.
  - similar words are “nearby in space”
  - We build this space automatically by seeing which words are nearby in text

Word embeddings or word vectors

WORD	d1	d2	d3	d4	d5	...	d50
summer	0.12	0.21	0.07	0.25	0.33	...	0.51
spring	0.19	0.57	0.99	0.30	0.02	...	0.73
fall	0.53	0.77	0.43	0.20	0.29	...	0.85
light	0.00	0.68	0.84	0.45	0.11	...	0.03
clear	0.27	0.50	0.21	0.56	0.25	...	0.32
blizzard	0.15	0.05	0.64	0.17	0.99	...	0.23

We'll discuss 2 kinds of embeddings

- **tf-idf**
  - Information Retrieval workhorse!
  - A common baseline model
  - Sparse vectors
  - Words are represented by (a simple function of) the counts of nearby words

- **Word2vec**
  - Dense vectors
  - Representation is created by training a classifier to predict whether a word is likely to appear nearby
  - https://fasttext.cc/docs/en/crawl-vectors.html
  - Later we'll discuss extensions called contextual embeddings

Word-word matrix ("term-context matrix")

	knife	dog	sword	love	like
knife	0	1	6	5	5
dog	1	0	5	5	5
sword	6	5	0	5	5
love	5	5	5	0	5
like	5	5	5	5	2

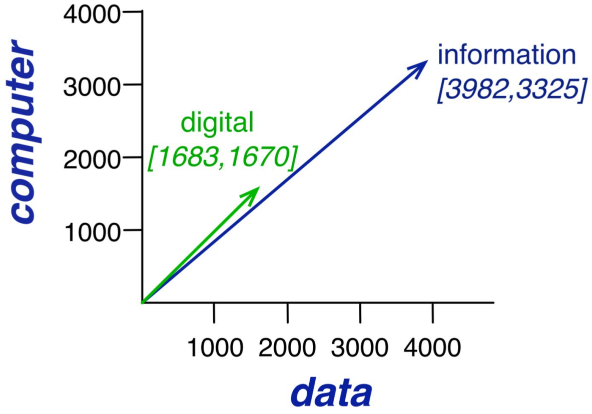
- Two words are "similar" in meaning if their context vectors are similar
  - Similarity == relatedness

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	...



Cosine for computing word similarity

The dot product between two vectors is a scalar:

$$\text{dot product}(\mathbf{v}, \mathbf{w}) = \mathbf{v} \cdot \mathbf{w} = \sum_{i=1}^{N} v_i w_i = v_1 w_1 + v_2 w_2 + \ldots + v_N w_N$$

- The dot product tends to be high when the two vectors have large values in the same dimensions
- Dot product can thus be a useful similarity metric between vectors

Problem with raw dot-product

- Dot product favors long vectors
  - Dot product is higher if a vector is longer (has higher values in many dimension)
    Vector length:

$$|\mathbf{v}| = \sqrt{\sum_{i=1}^{N} v_i^2}$$

- Frequent words (of, the, you) have long vectors (since they occur many times with other words).
  - So dot product overly favors frequent words

Alternative: cosine for computing word similarity

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

Based on the definition of the dot product between two vectors a and b

$$\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\theta$$

$$\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}||\mathbf{b}|} = \cos\theta$$

Cosine examples

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

	pie	data	computer
cherry	442	8	2
digital	114	80	62
information	36	58	1

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

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

Count-based representations

	knife	dog	sword	love	like
knife	0	1	6	5	5
dog	1	0	5	5	5
sword	6	5	0	5	5
love	5	5	5	0	5
like	5	5	5	5	2

- Counts: term-frequency
  - remove stop words
  - use $\log_{10}(tf)$

But raw frequency is a bad representation

- The co-occurrence matrices we have seen represent each cell by word frequencies
- Frequency is clearly useful; if sugar appears a lot near apricot, that's useful information
- But overly frequent words like the, it, or they are not very informative about the context
- It's a paradox! How can we balance these two conflicting constraints?

Two common solutions for word weighting

tf-idf: tf-idf value for word t in document d:

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

Words like “the” or “it” have very low idf

PMI: Pointwise mutual information

$$\mathrm{PMI}(w_1, w_2) = log \frac{p(w_1,w_2)}{p(w_1)p(w_2)}$$

See if words like “good” appear more often with “great” than we would expect by chance

TF-IDF

- What to do with words that are evenly distributed across many documents?

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d)+1)$$

$$\text{idf}_i = \log\left(\frac{N}{\text{df}_i}\right)$$

Total # of docs in collection

# of docs that have word i

Words like "the" or "good" have very low idf

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

Positive Pointwise Mutual Information (PPMI)

- In word--context matrix
- Do words w and c co-occur more than if they were independent?

$$\text{PMI}(w,c) = \log_2 \frac{P(w,c)}{P(w)P(c)}$$

$$\text{PPMI}(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P(c)}, 0)$$

- PMI is biased toward infrequent events
  - Very rare words have very high PMI values
  - Give rare words slightly higher probabilities α=0.75

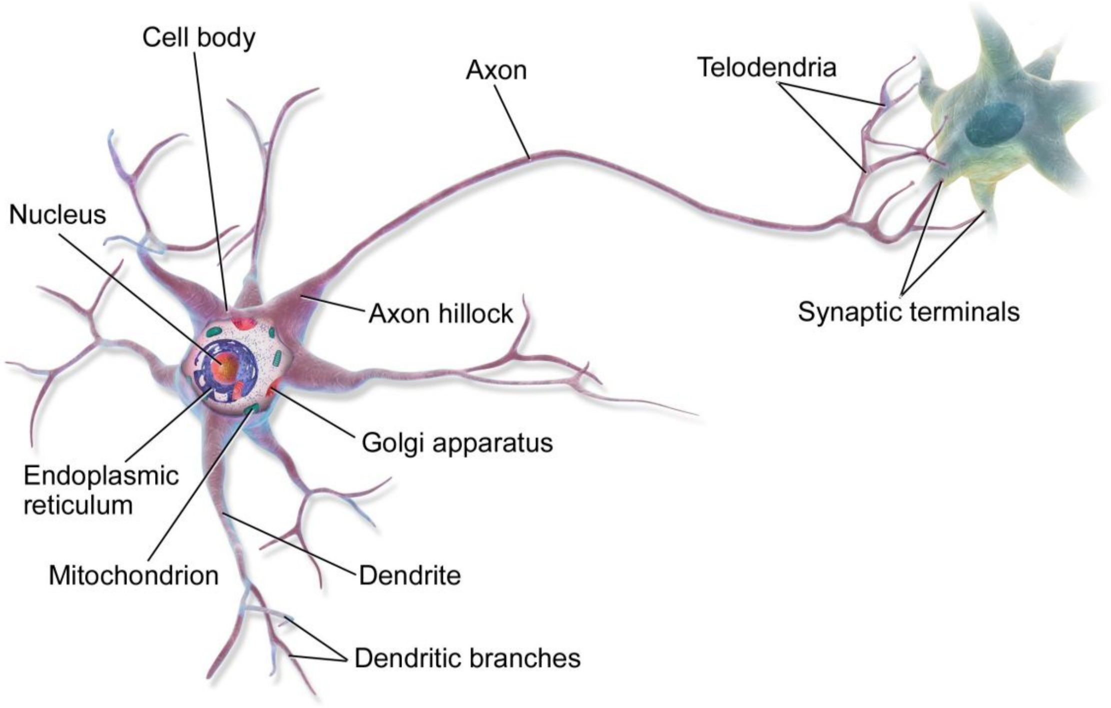
$$\text{PPMI}_\alpha(w,c) = \max(\log_2 \frac{P(w,c)}{P(w)P_\alpha(c)}, 0) \qquad P_\alpha(c) = \frac{count(c)^\alpha}{\sum_c count(c)^\alpha}$$

We'll discuss 2 kinds of embeddings

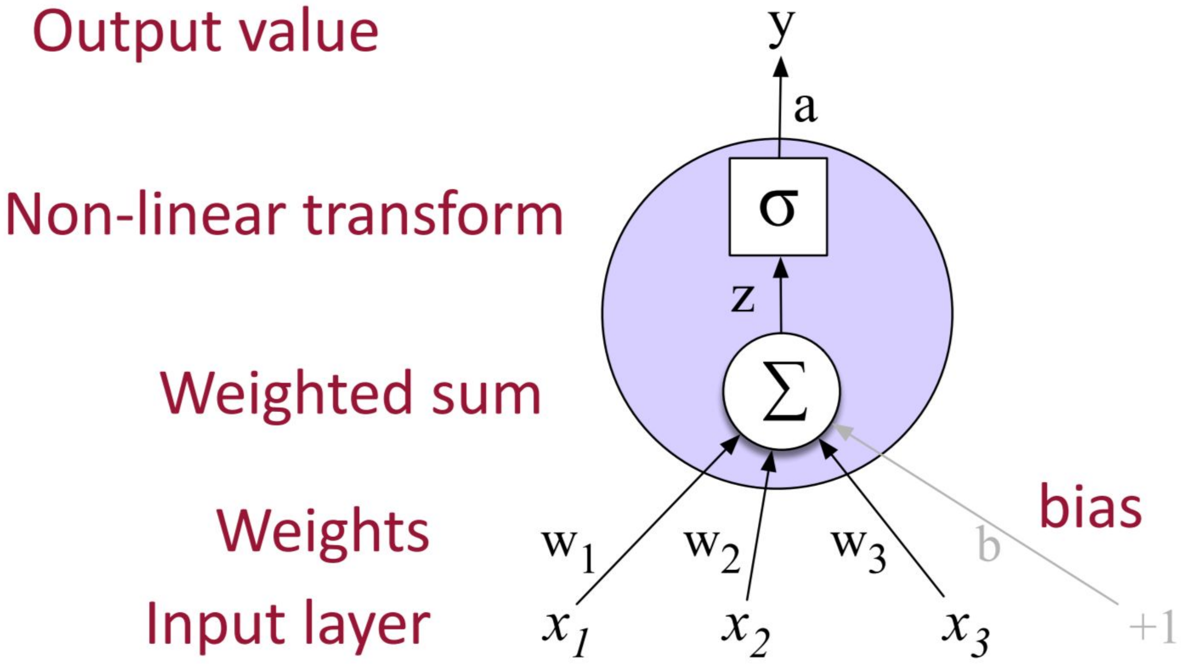
- **tf-idf**
  - Information Retrieval workhorse!
  - A common baseline model
  - Sparse vectors
  - Words are represented by (a simple function of) the counts of nearby words

- **Word2vec**
  - Dense vectors
  - Representation is created by training a classifier to predict whether a word is likely to appear nearby
  - https://fasttext.cc/docs/en/crawl-vectors.html
  - Later we'll discuss extensions called contextual embeddings

This is in your brain



Neural Network Unit (this is not in your brain)



Neural unit

- Take weighted sum of inputs, plus a bias

$$z = b + \sum_i w_i x_i$$

$$z = w \cdot x + b$$

- Instead of just using z, we'll apply a nonlinear activation function f:

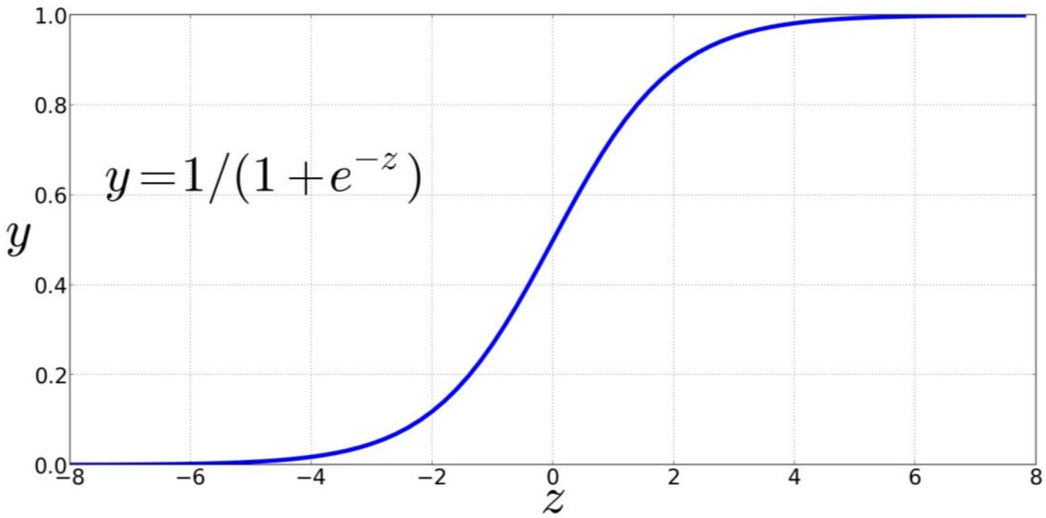
$$y = a = f(z)$$

Non-Linear Activation Functions

- We've already seen the sigmoid for logistic regression:

Sigmoid

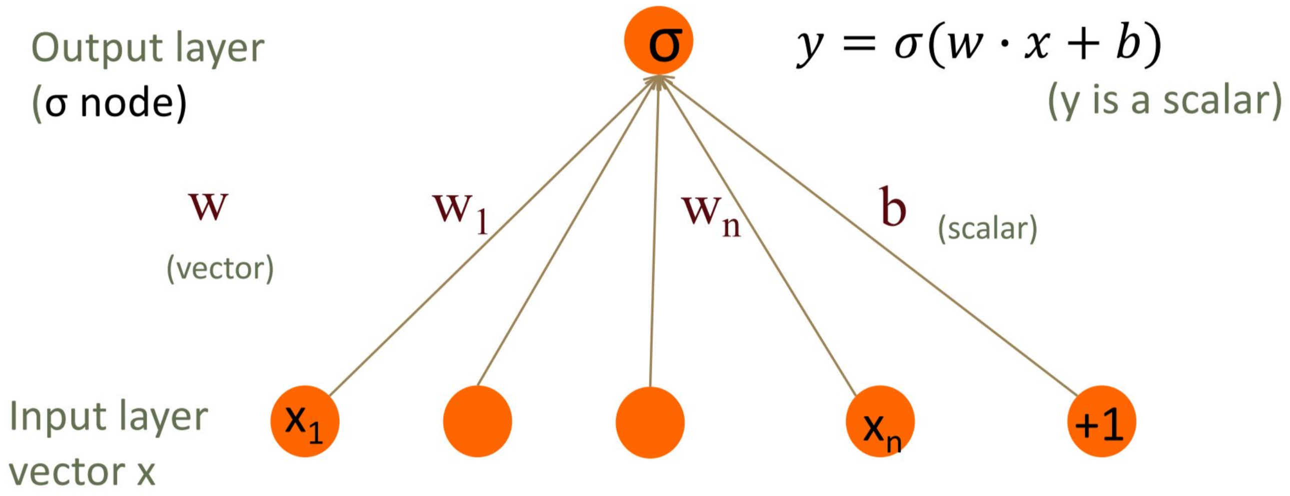
$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$



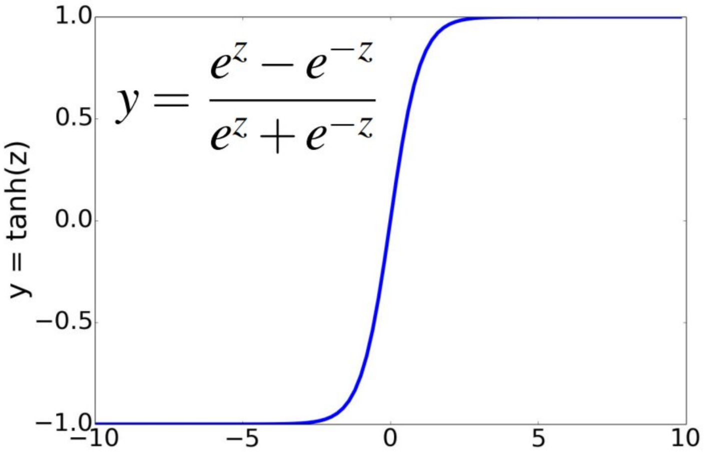
Final function the unit is computing

$$y = \sigma(w \cdot x + b) = \frac{1}{1 + \exp(-(w \cdot x + b))}$$

Binary Logistic Regression as a 1-layer network

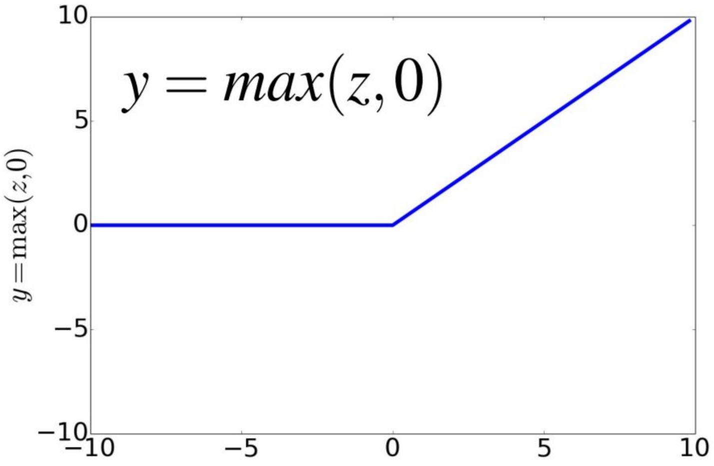


Non-Linear Activation Functions besides sigmoid



tanh

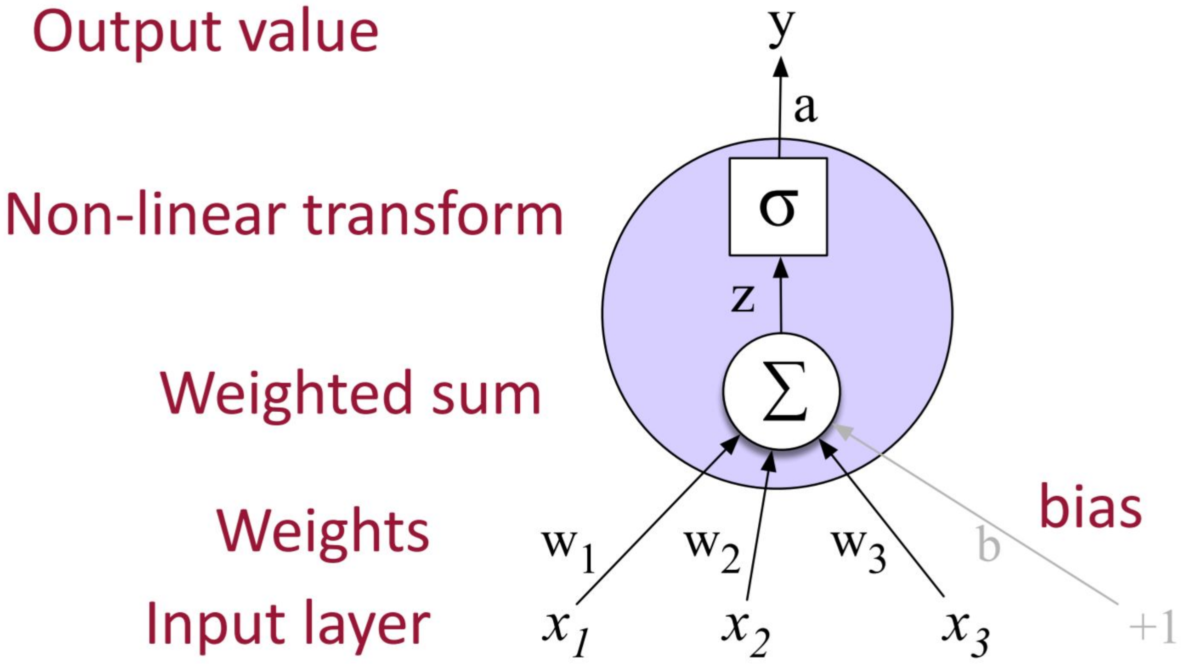
Most Common:



ReLU

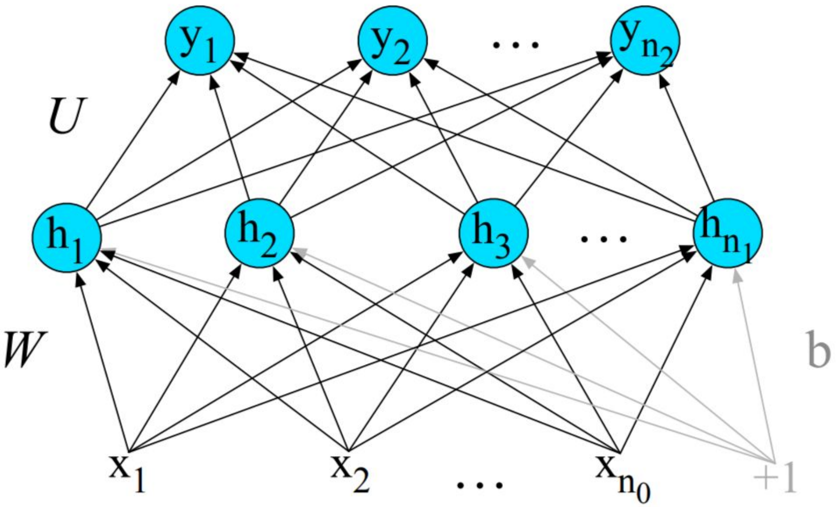
Rectified Linear Unit

Final unit again

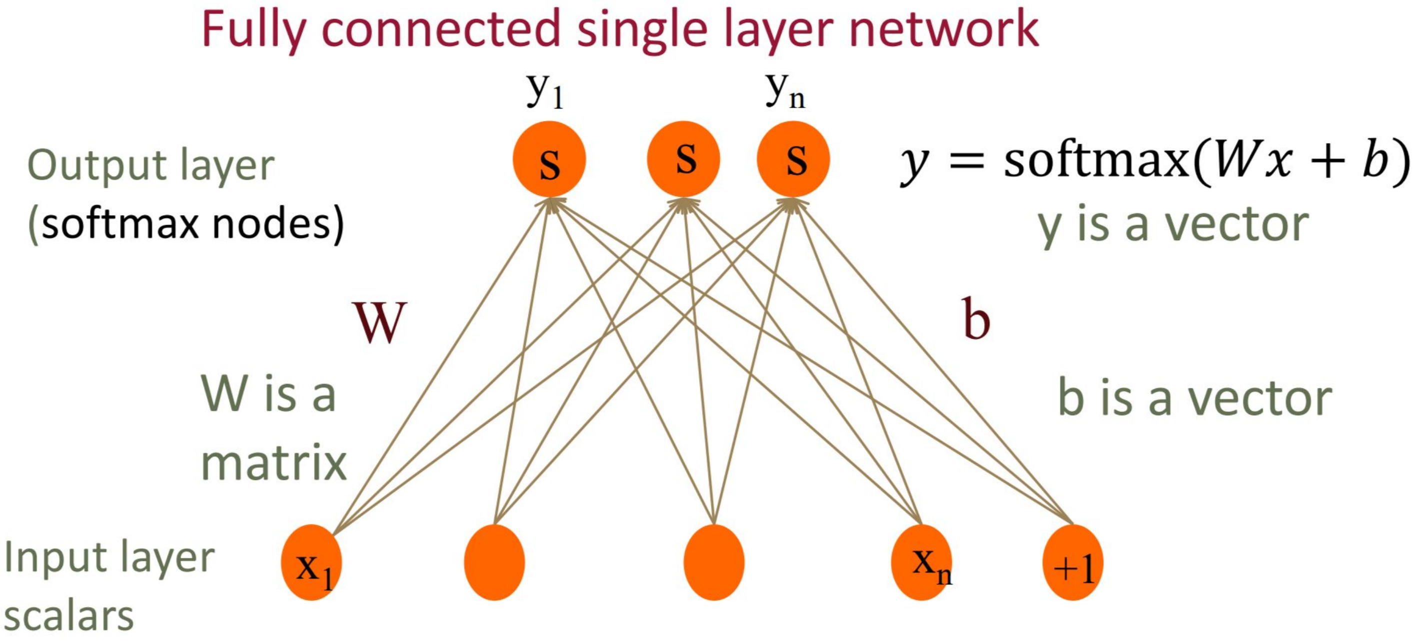


Feedforward Neural Networks

- Can also be called multi-layer perceptrons (or MLPs) for historical reasons
  - (we don't count the input layer in counting layers!)



Multinomial Logistic Regression as a 1-layer Network



softmax: a generalization of sigmoid

- For a vector z of dimensionality k, the softmax is:

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k}\exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k}\exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k}\exp(z_i)}\right]$$

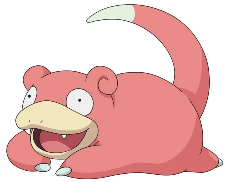
$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k}\exp(z_j)} \quad 1 \le i \le k$$

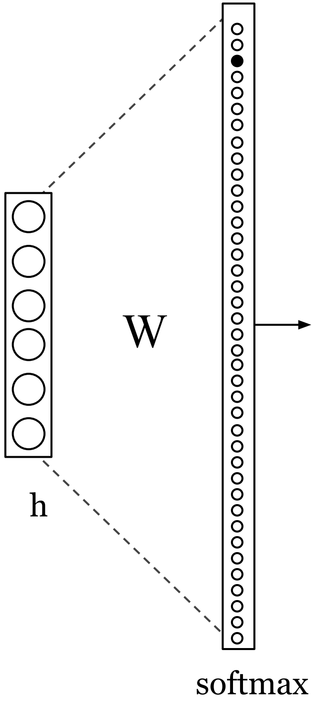
Example:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

softmax





$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

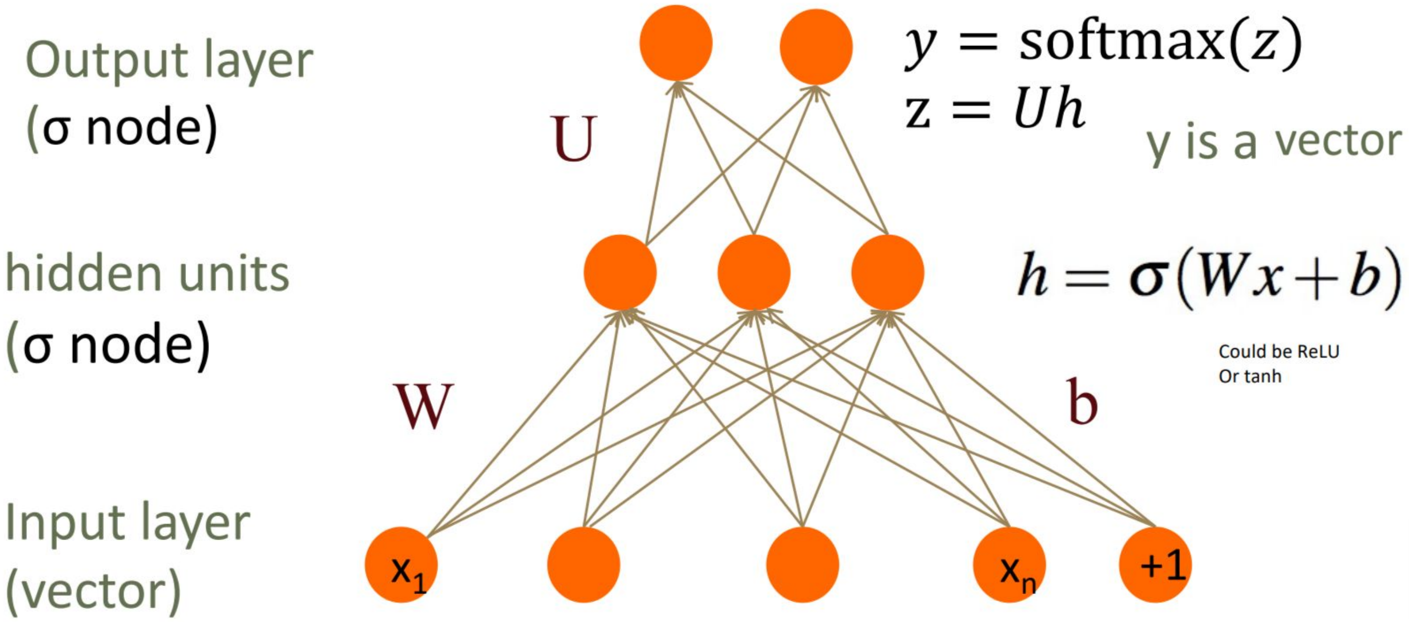
$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \leq i \leq k$$

Example:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

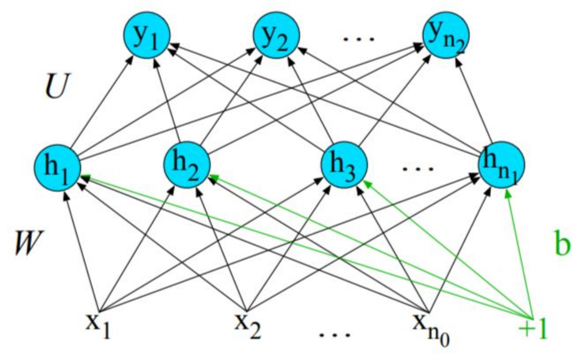
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Two-Layer Network with softmax output

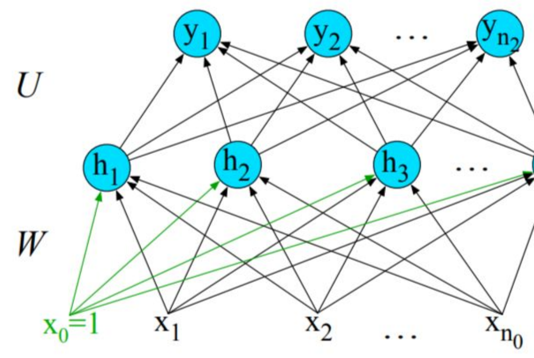


Replacing the bias unit

Instead of:



We'll do this:

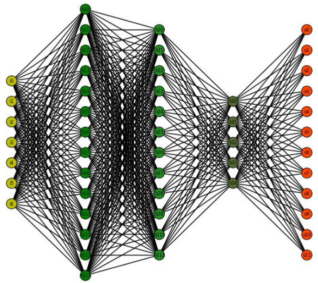


Learning the weights

- Cross-entropy loss
- Backpropagation algorithm

Algorithm 1 Backpropagation Algorithm

procedure TRAIN
    $X \leftarrow$ Training Data Set of size mxn
    $y \leftarrow$ Labels for records in X
    $w \leftarrow$ The weights for respective layers
    $l \leftarrow$ The number of layers in the neural network, 1...L
    $D_{ij}^{(l)} \leftarrow$ The error for all l,i,j
    $t_{ij}^{(l)} \leftarrow 0$. For all l,i,j
    For $i = 1$ to m
        $a^l \leftarrow feedforward(x^{(i)}, w)$
        $d^l \leftarrow a(L) - y(i)$
        $t_{ij}^{(l)} \leftarrow t_{ij}^{(l)} + a_j^{(l)} \cdot t_i^{l+1}$
    if $j \neq 0$ **then**
        $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)} + \lambda w_{ij}^{(l)}$
    else
        $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)}$
        where $\frac{\partial}{\partial w_{ij}^{(l)}} J(w) = D_{ij}^{(l)}$



Applying neural networks to NLP tasks

Use cases for feedforward networks

- Word representations
- Text classification
- Language modeling

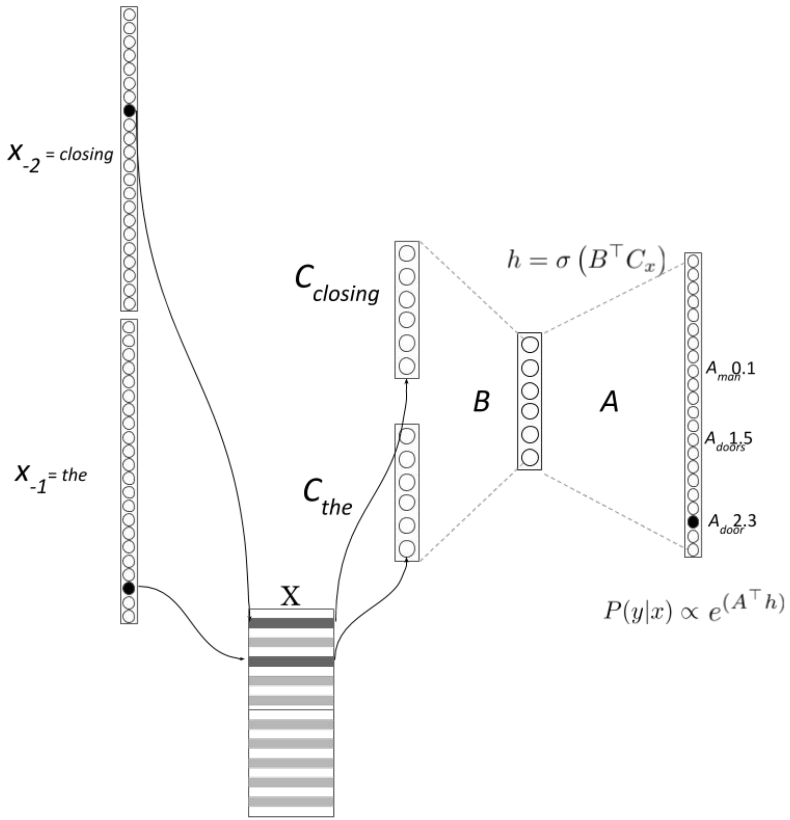
State of the art systems use more powerful neural architectures (we will learn transformers architectures on Monday), but simple models are useful to consider!

Distributed representations

Word Vectors

WORD	d1	d2	d3	d4	d5	...	d50
summer	0.12	0.21	0.07	0.25	0.33	...	0.51
spring	0.19	0.57	0.99	0.30	0.02	...	0.73
fall	0.53	0.77	0.43	0.20	0.29	...	0.85
light	0.00	0.68	0.84	0.45	0.11	...	0.03
clear	0.27	0.50	0.21	0.56	0.25	...	0.32
blizzard	0.15	0.05	0.64	0.17	0.99	...	0.23

“One hot” vectors and dense word vectors (embeddings)

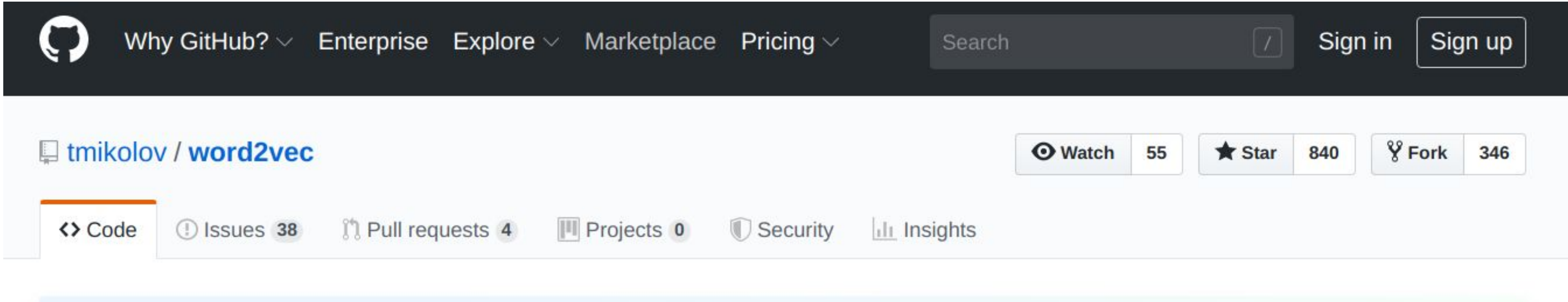


Low-dimensional word representations

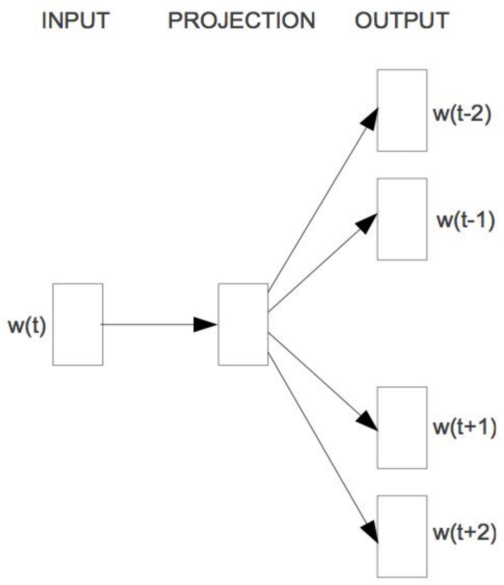
- Learning representations by back-propagating errors
  - Rumelhart, Hinton & Williams, 1986
- A neural probabilistic language model
  - Bengio et al., 2003
- Natural Language Processing (almost) from scratch
  - Collobert & Weston, 2008
- Word representations: A simple and general method for semi-supervised learning
  - Turian et al., 2010
- Distributed Representations of Words and Phrases and their Compositionality
  - Word2Vec; Mikolov et al., 2013

Word2Vec

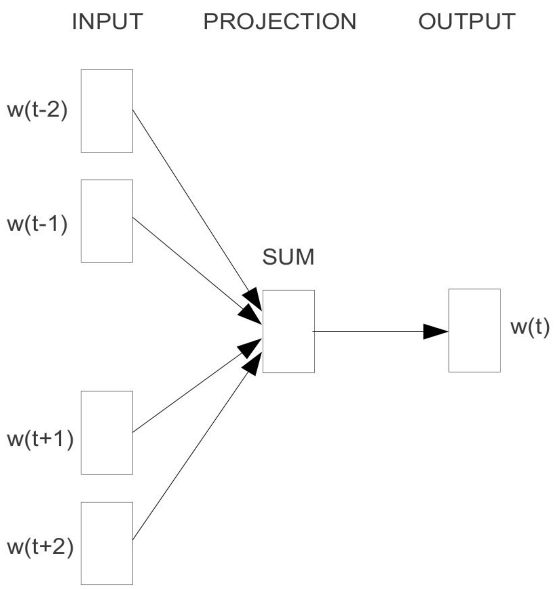
- Popular embedding method
- Very fast to train
- Code available on the web
- Idea: predict rather than count



Word2Vec



Skip-gram



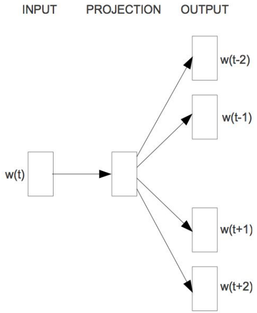
CBOW

- [Mikolov et al.' 13]

Skip-gram Prediction

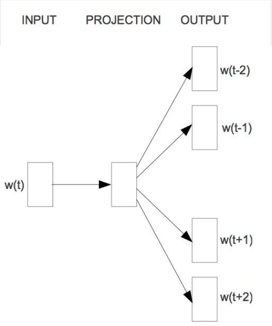
- Predict vs Count

the cat sat on the mat



Skip-gram

Skip-gram Prediction



Skip-gram

- Predict vs Count

the cat sat on the mat

w_t = the → **CLASSIFIER** →

$w_{t-2} = <\text{start}_{-2}>$
$w_{t-1} = <\text{start}_{-1}>$
w_{t+1} = cat
w_{t+2} = sat

context size = 2

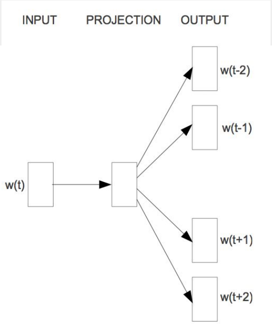
Skip-gram Prediction

- Predict vs Count

the cat sat on the mat

w_t = cat → **CLASSIFIER** →

w_{t-2} = <start$_{-1}$>
w_{t-1} = the
w_{t+1} = sat
w_{t+2} = on

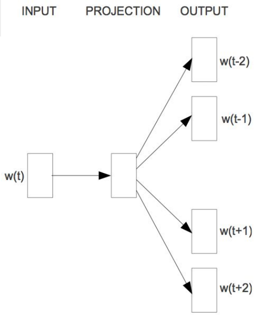


Skip-gram

context size = 2

Skip-gram Prediction

- Predict vs Count



Skip-gram

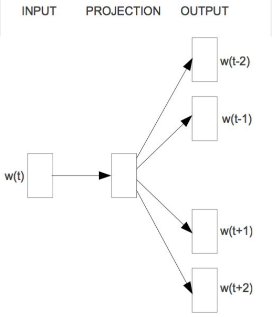
the cat sat on the mat

w_t = sat → **CLASSIFIER** →
w_{t-2} = the
w_{t-1} = cat
w_{t+1} = on
w_{t+2} = the

context size = 2

Skip-gram Prediction

- Predict vs Count



Skip-gram

the cat sat on the mat

w_t = on → CLASSIFIER → w_{t-2} = cat, w_{t-1} = sat, w_{t+1} = the, w_{t+2} = mat

context size = 2

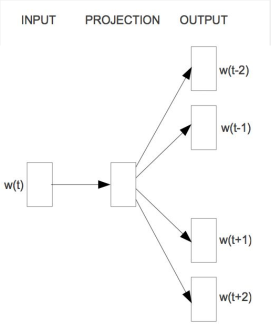
Skip-gram Prediction

- Predict vs Count

the cat sat on the mat

w_t = the → **CLASSIFIER** →

w_{t-2} = sat
w_{t-1} = on
w_{t+1} = mat
w_{t+2} = <end$_{+1}$>



Skip-gram

context size = 2

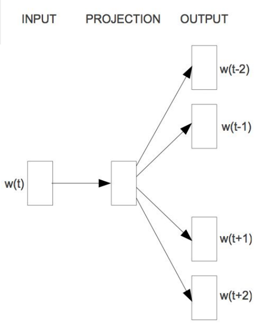
Skip-gram Prediction

- Predict vs Count

the cat sat $\boxed{\text{on the } \underline{\text{mat}}}$

w_t = mat → **CLASSIFIER** →

w_{t-2} = on
w_{t-1} = the
w_{t+1} = <end$_{+1}$>
w_{t+2} = <end$_{+2}$>

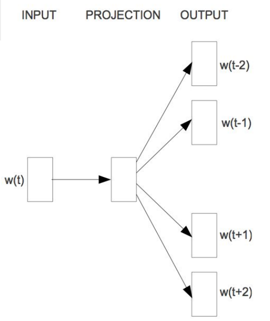


Skip-gram

context size = 2

Skip-gram Prediction

- Predict vs Count



Skip-gram

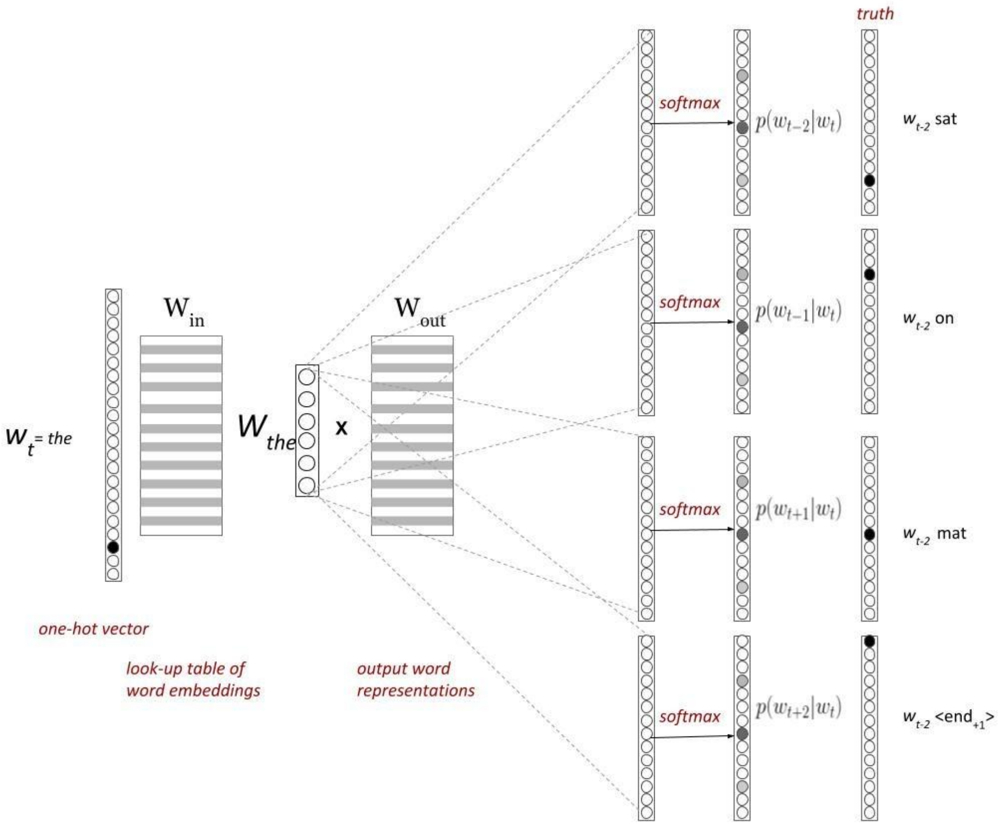
w_t = the → **CLASSIFIER** →

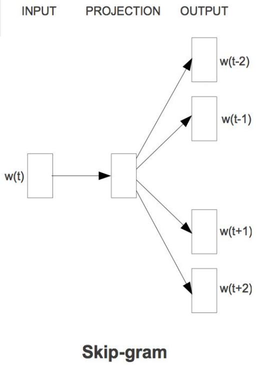
w_{t-2} = sat
w_{t-1} = on
w_{t+1} = mat
$w_{t+2} = <\text{end}_{+1}>$

w_t = the → **CLASSIFIER** →

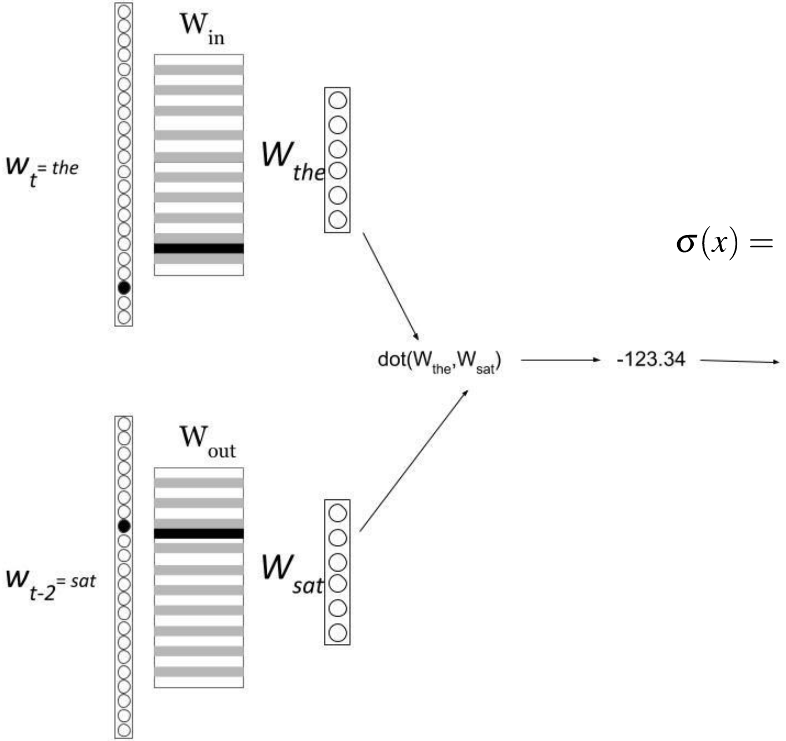
$w_{t-2} = <\text{start}_{-2}>$
$w_{t-1} = <\text{start}_{-1}>$
w_{t+1} = cat
w_{t+2} = sat

Skip-gram Prediction

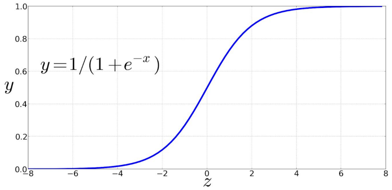




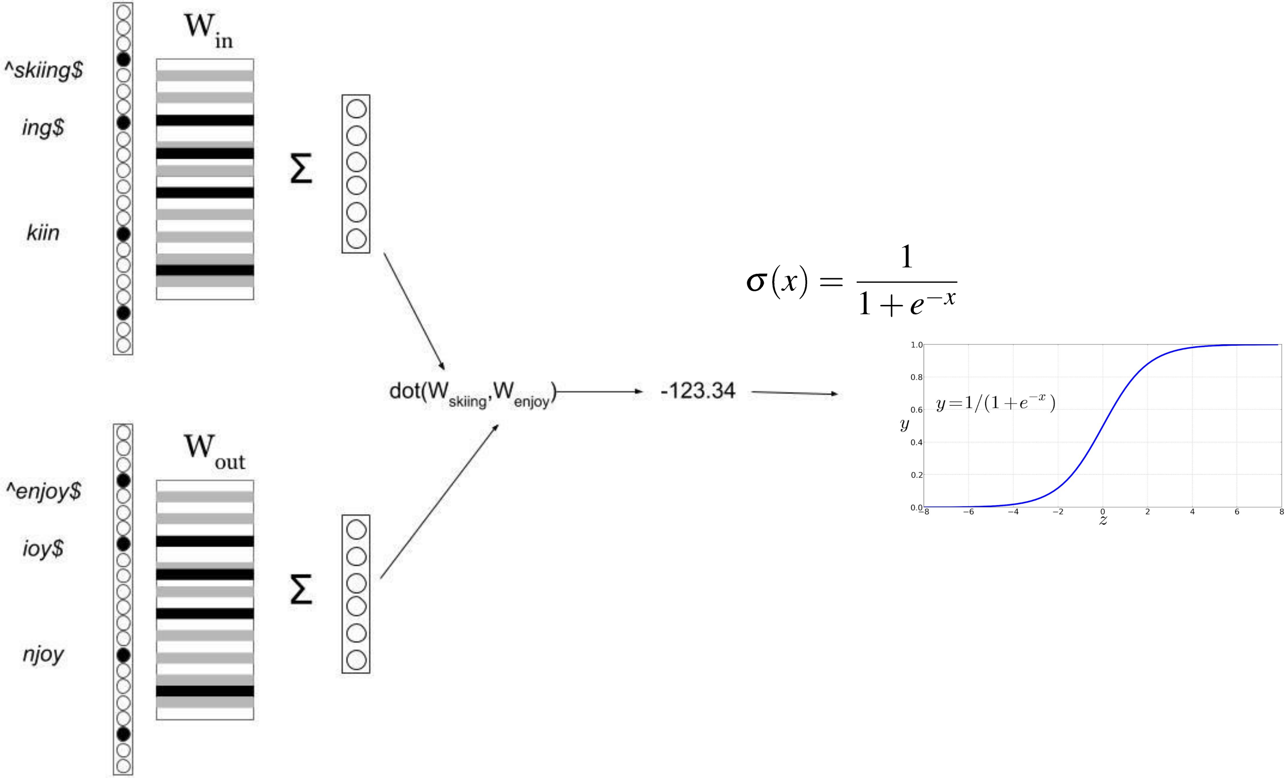
How to compute p(+|t,c)?



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



FastText



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

SGNS

Given a tuple (t,c) = target, context

- (cat, sat)
- (cat, aardvark)

Return probability that c is a real context word:

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

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

Learning the classifier

- Iterative process
  - We'll start with 0 or random weights
  - Then adjust the word weights to
    - make the positive pairs more likely
    - and the negative pairs less likely
  - over the entire training set:

$$\sum_{(t,c)\in +} logP(+|t,c) + \sum_{(t,c)\in -} logP(-|t,c)$$

- Train using gradient descent

BERT

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin Ming-Wei Chang Kenton Lee Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html

Use cases for feedforward networks

- Word representations
- Text classification
- Language modeling

State of the art systems use more powerful neural architectures (we will learn transformers architectures on Monday), but simple models are useful to consider!

Neural LMs

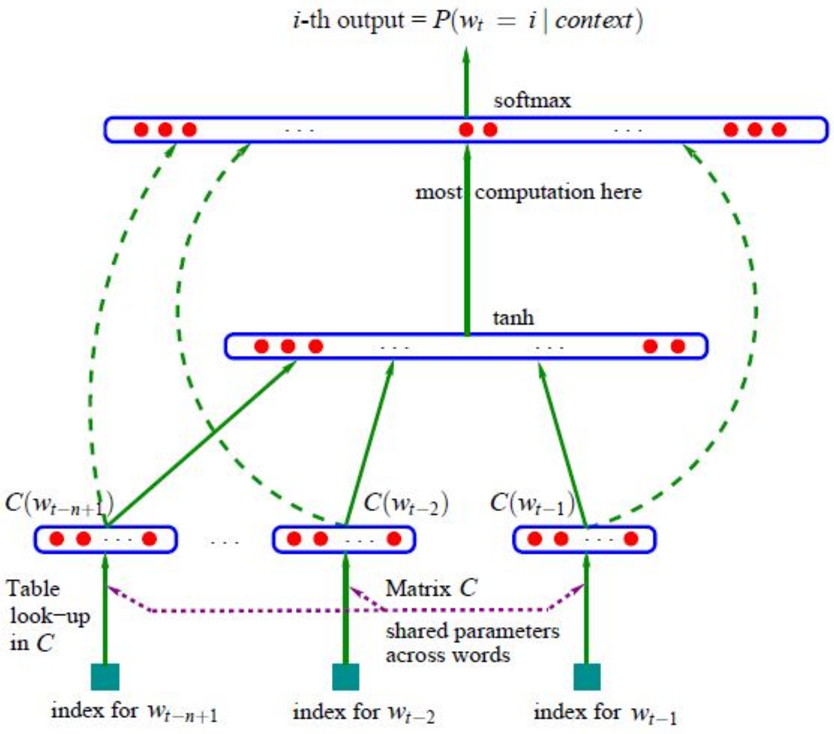


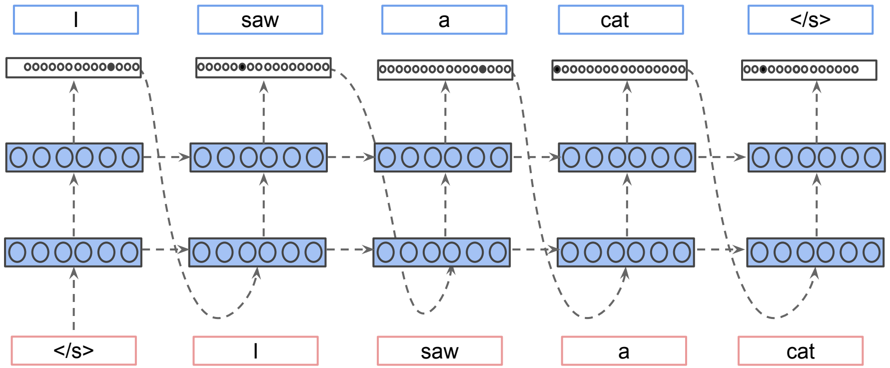
Image: (Bengio et al, 03)

Neural LMs

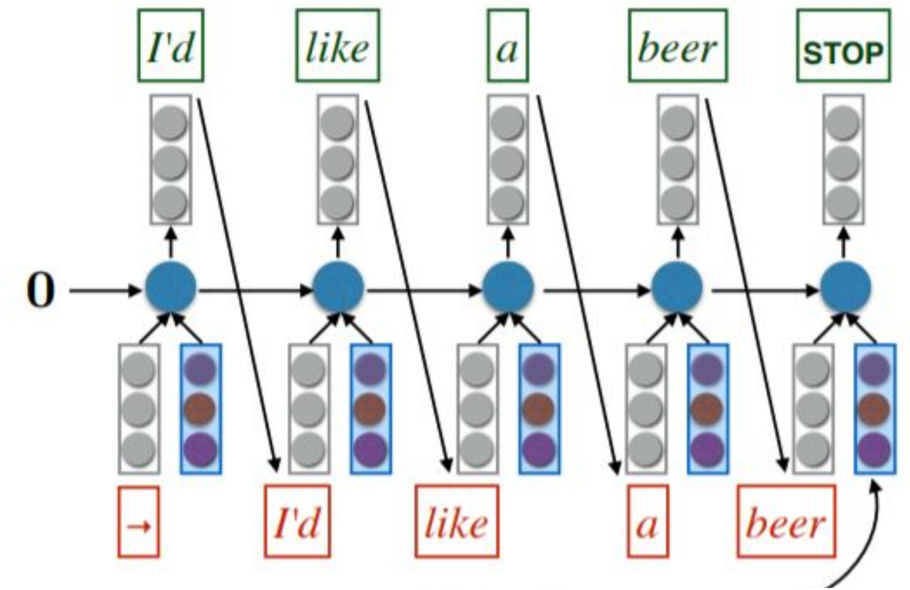
	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321

(Bengio et al, 03)

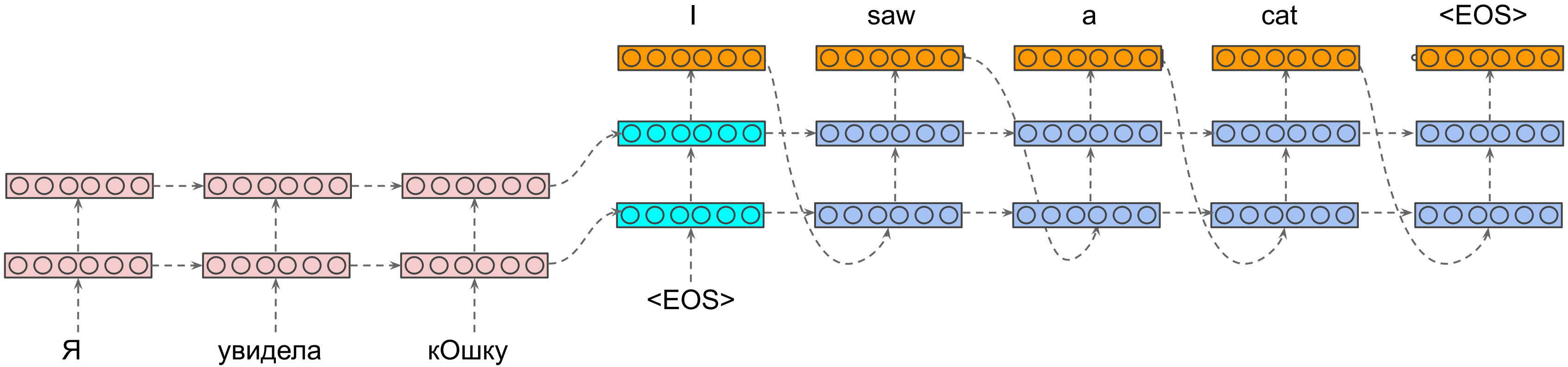
Recurrent LMs



Recurrent LMs

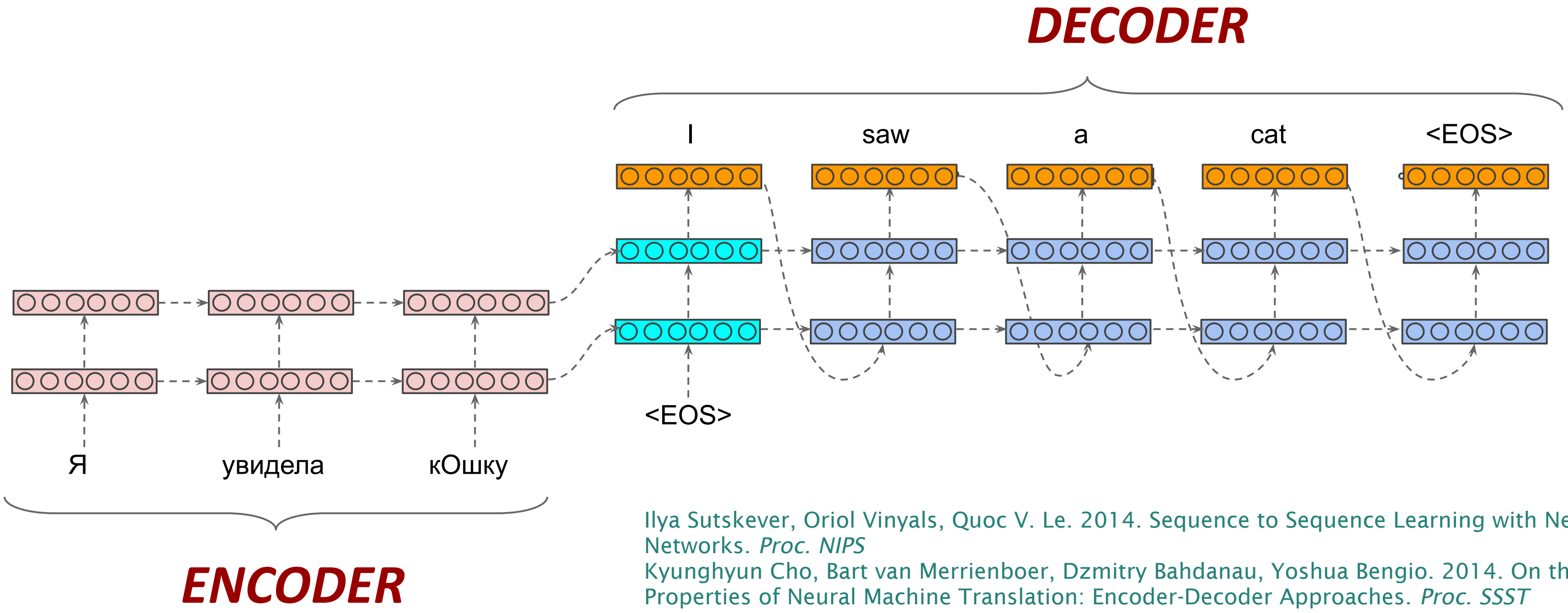


Sequence-to-Sequence Models



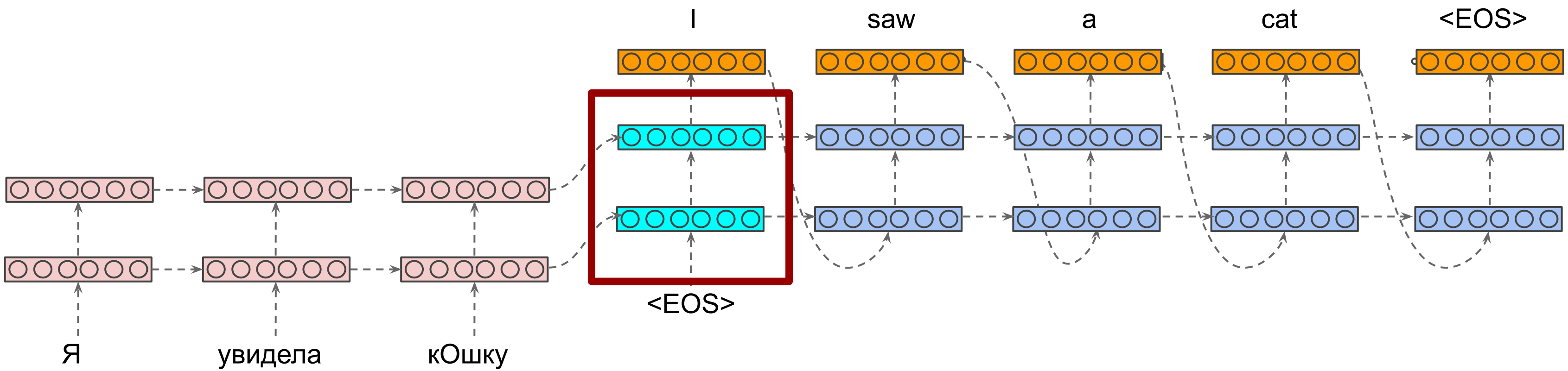
Ilya Sutskever, Oriol Vinyals, Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. Proc. NIPS

Sequence-to-Sequence Models for Neural Machine Translation



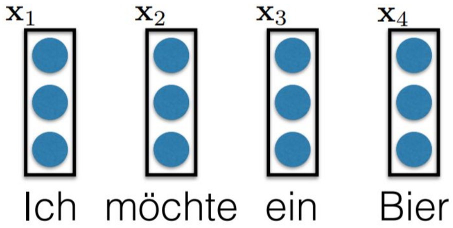
Ilya Sutskever, Oriol Vinyals, Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. *Proc. NIPS*
Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *Proc. SSST*

Sequence-to-Sequence Models for NMT

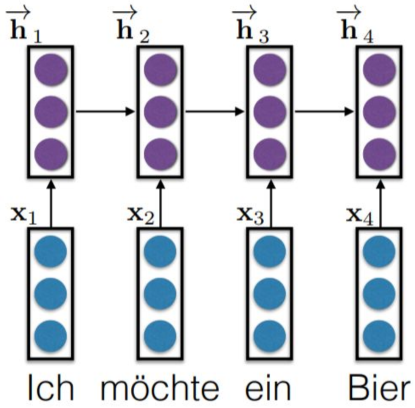


sentence representation

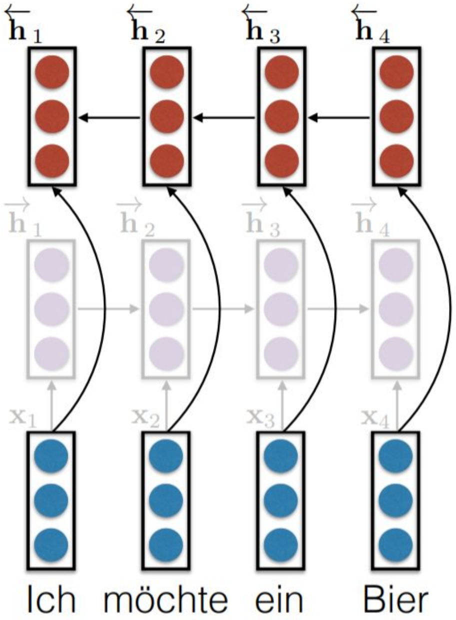
Encoder: Bidirectional RNN



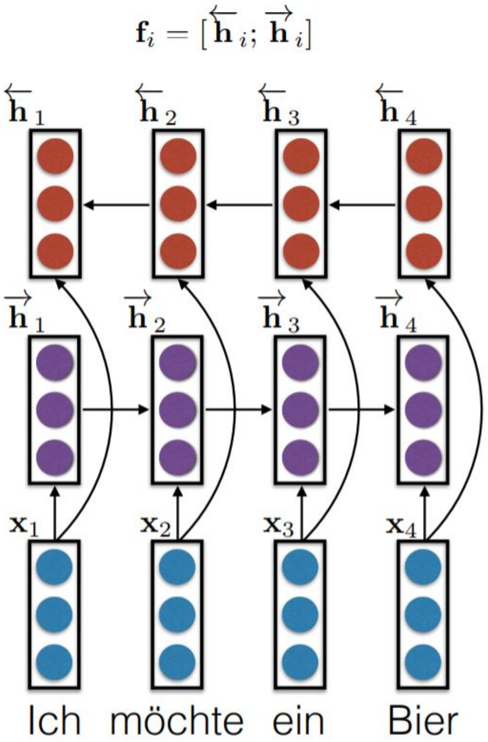
Encoder: Bidirectional RNN



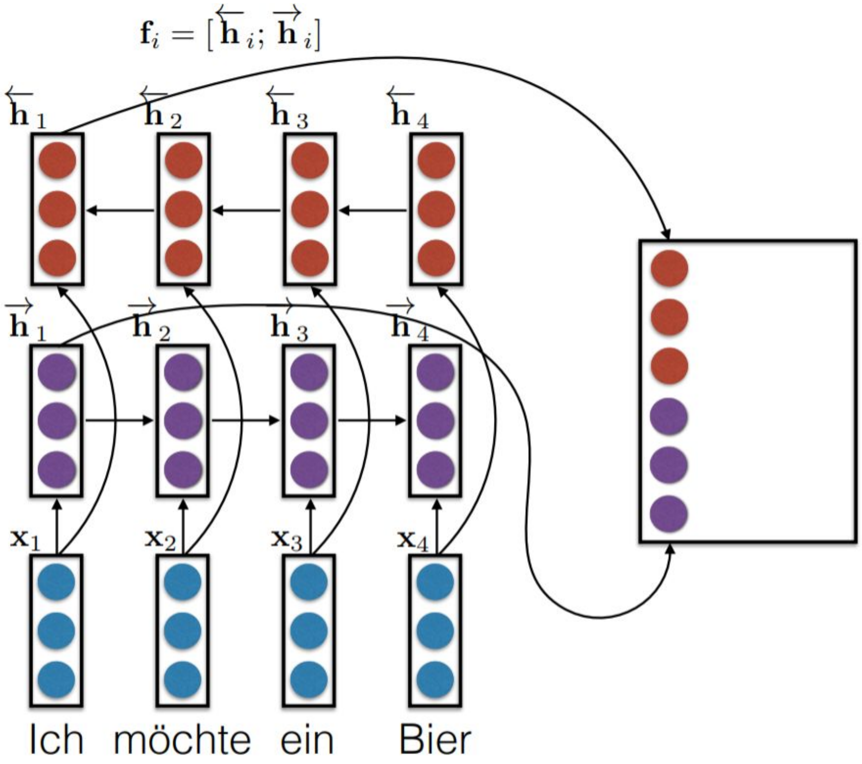
Encoder: Bidirectional RNN



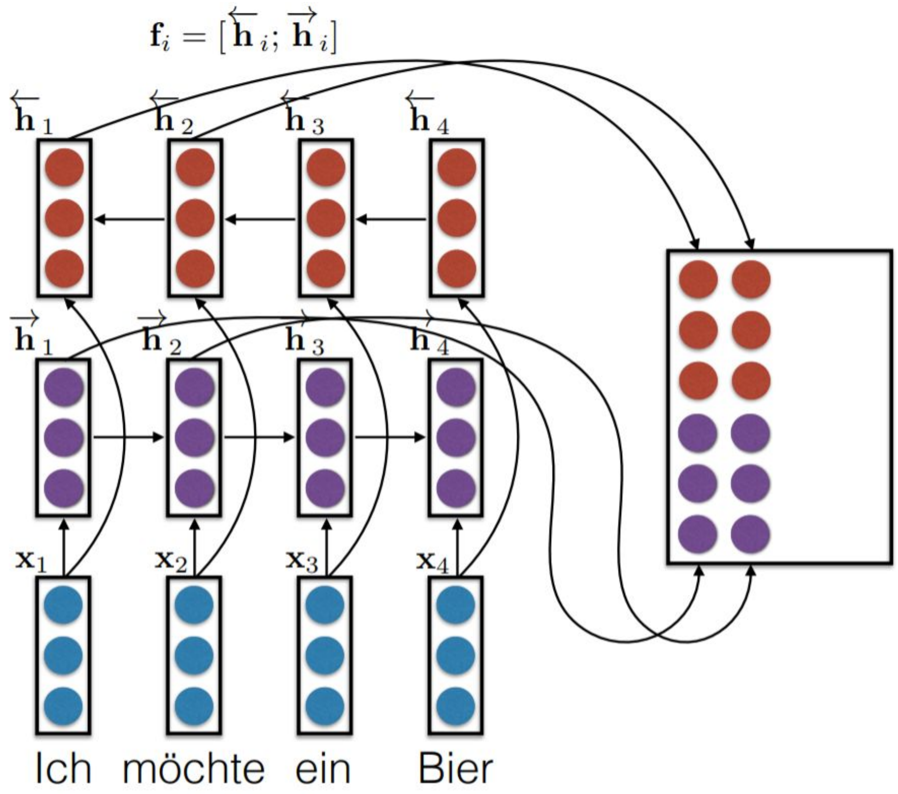
Encoder: Bidirectional RNN



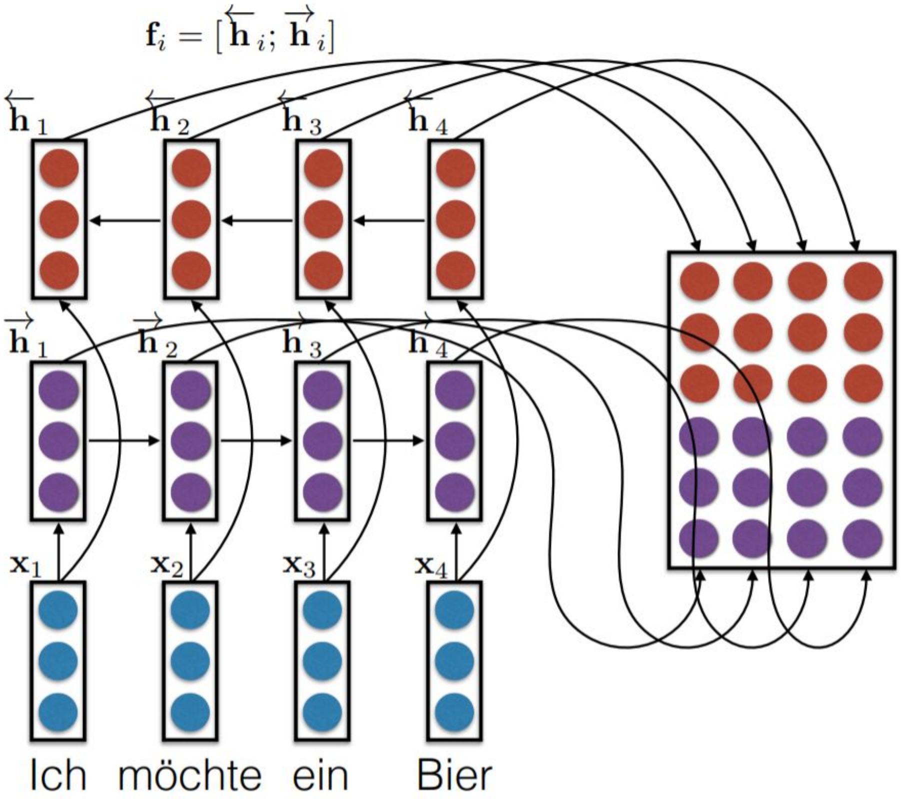
Encoder: Bidirectional RNN



Encoder: Bidirectional RNN

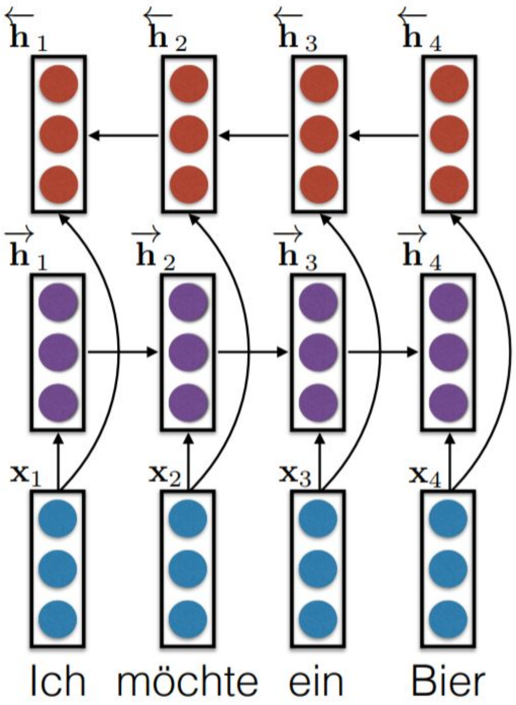


Encoder: Bidirectional RNN

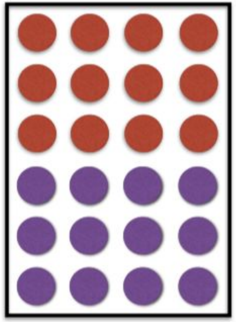


Matrix Sentence Encoding

$$\mathbf{f}_i = [\overleftarrow{\mathbf{h}}_i; \overrightarrow{\mathbf{h}}_i]$$



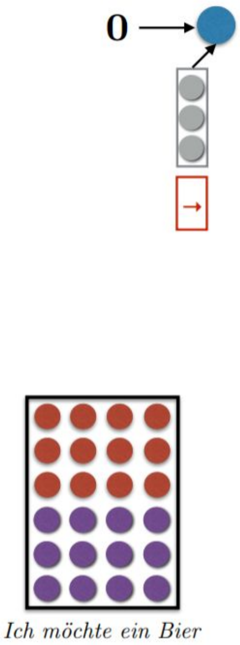
$\mathbf{F} \in \mathbb{R}^{2n \times |\boldsymbol{f}|}$

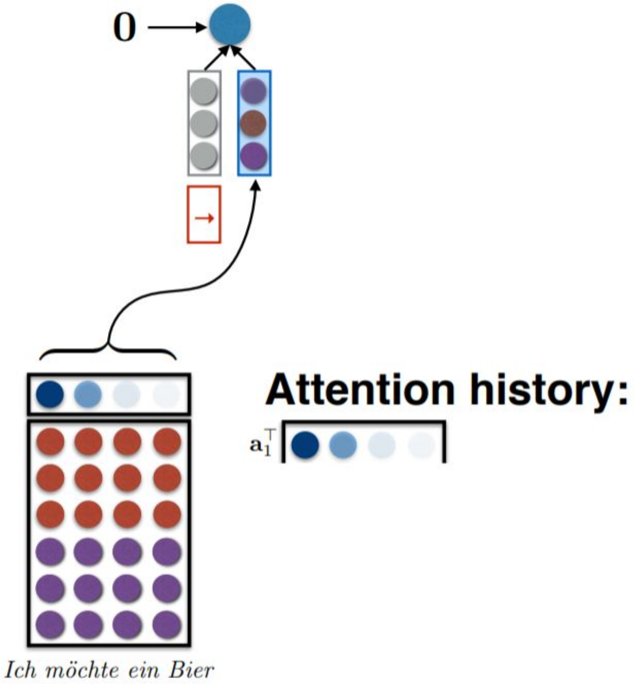


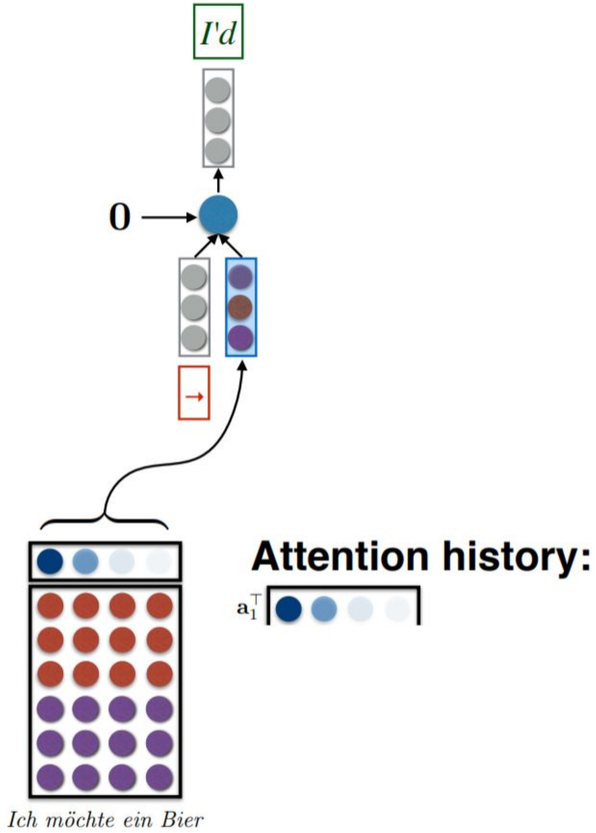
Ich möchte ein Bier

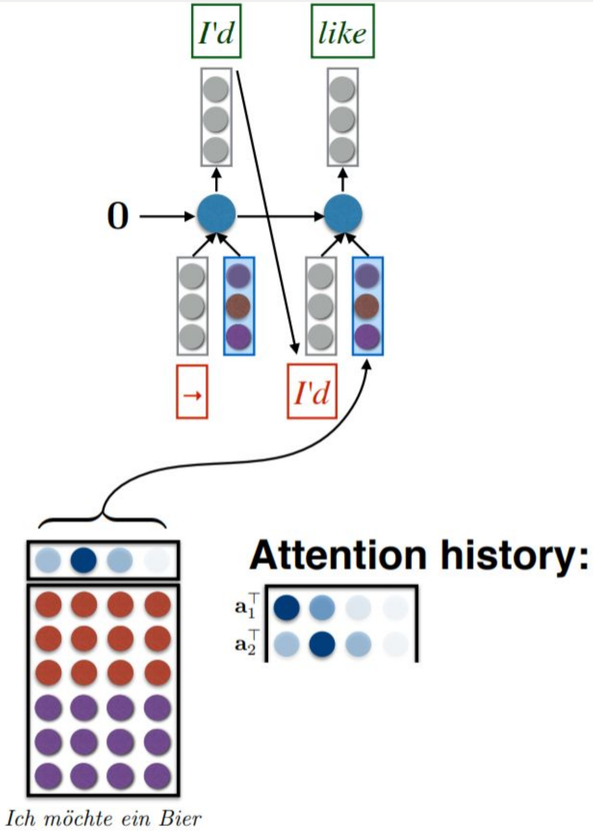
- matrix-encoded sentence

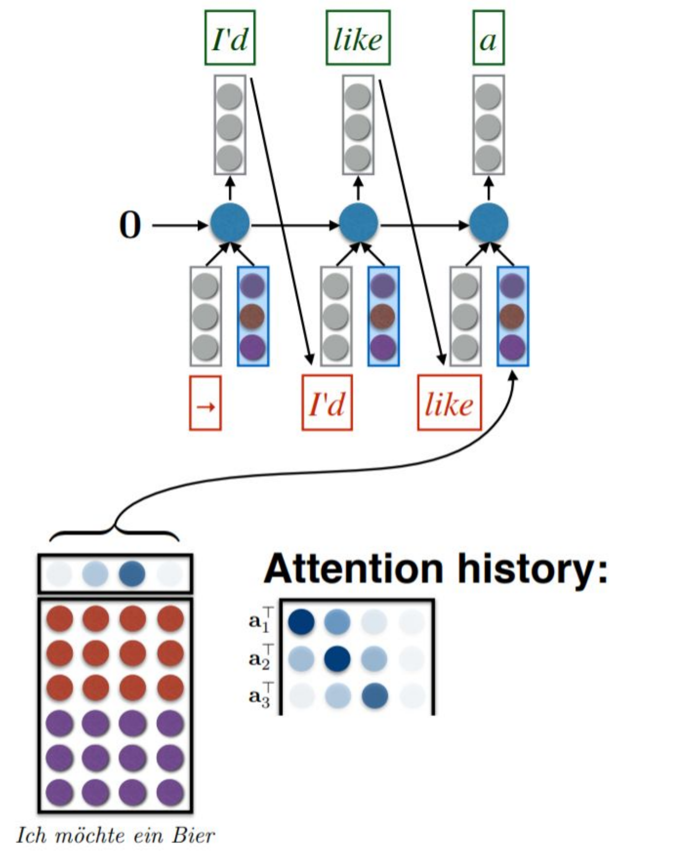
Decoder: RNN + Attention

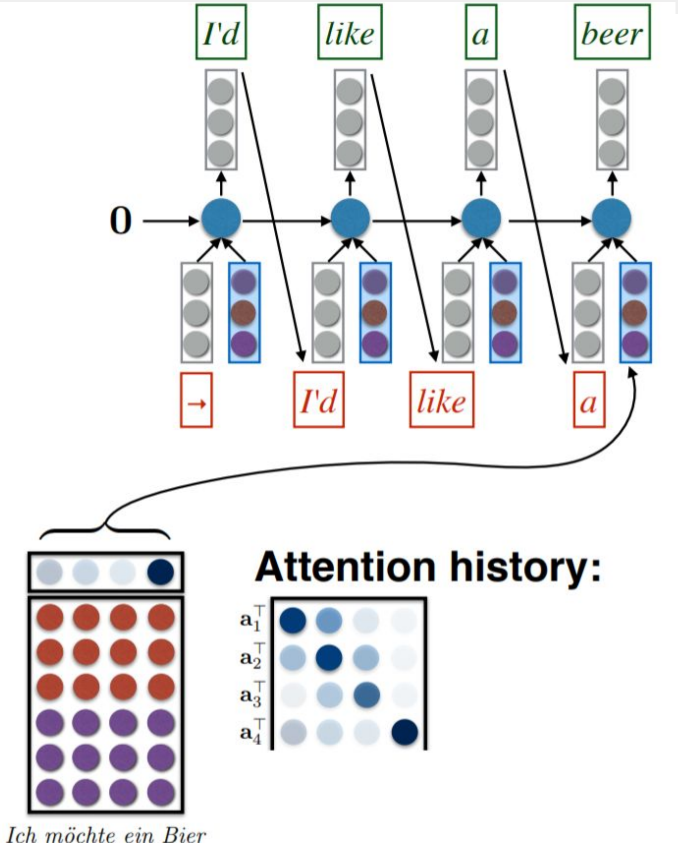


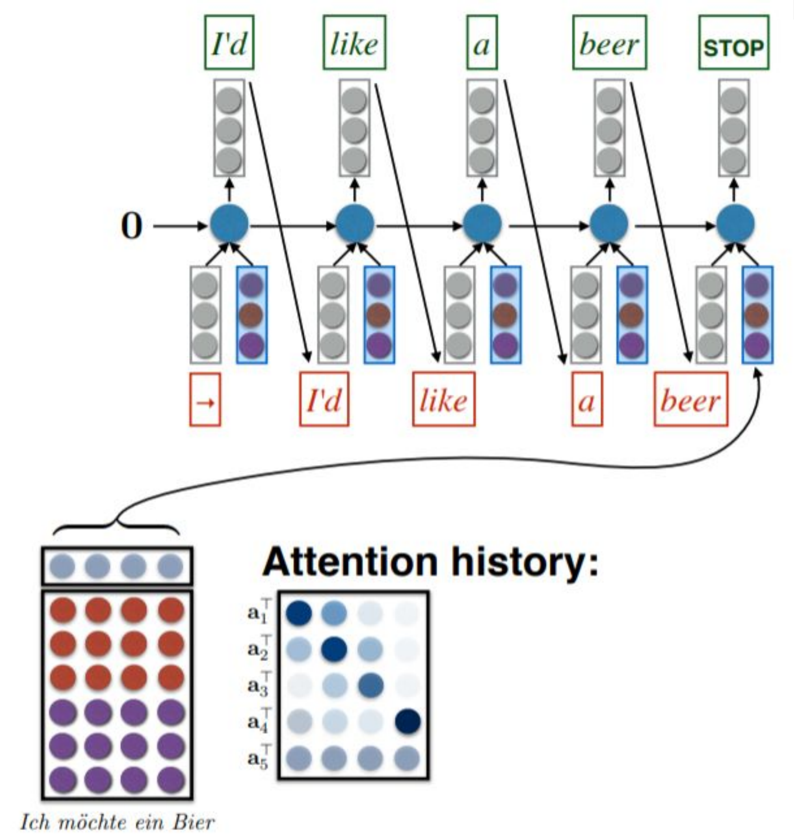

0
Attention history:
$\mathbf{a}_1^\top$
Ich möchte ein Bier


I'd
0
Attention history:
$\mathbf{a}_1^\top$
Ich möchte ein Bier


I'd
like
0
→
I'd
Ich möchte ein Bier
Attention history:
$\mathbf{a}_1^\top$
$\mathbf{a}_2^\top$


I'd
like
a
0
→
I'd
like
Attention history:
$a_1^\top$
$a_2^\top$
$a_3^\top$
Ich möchte ein Bier


I'd
like
a
beer
0
→
I'd
like
a
Attention history:
$a_1^\top$
$a_2^\top$
$a_3^\top$
$a_4^\top$
Ich möchte ein Bier


I'd
like
a
beer
STOP
0
→
I'd
like
a
beer
Attention history:
$a_1^\top$
$a_2^\top$
$a_3^\top$
$a_4^\top$
$a_5^\top$
Ich möchte ein Bier