Natural Language Processing

Neural Networks and Neural LMs

Yulia Tsvetkov

yuliats@cs.washington.edu

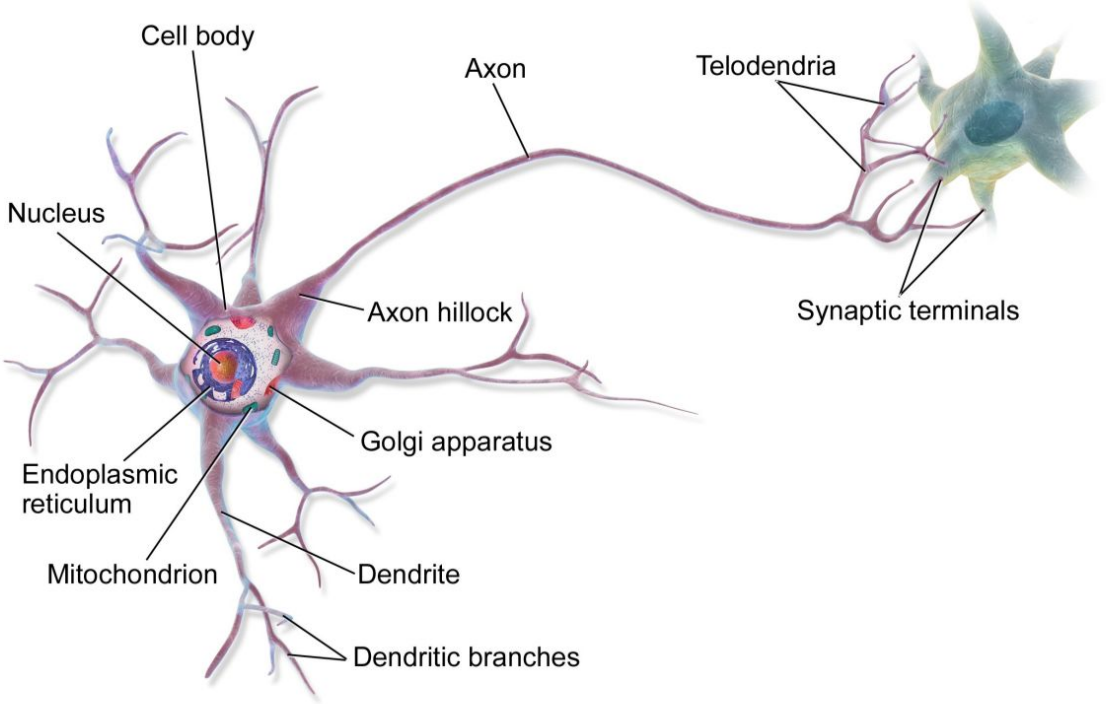
PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Readings

- Neutral networks chapters in J&M 6:
  - https://web.stanford.edu/~jurafsky/slp3/6.pdf

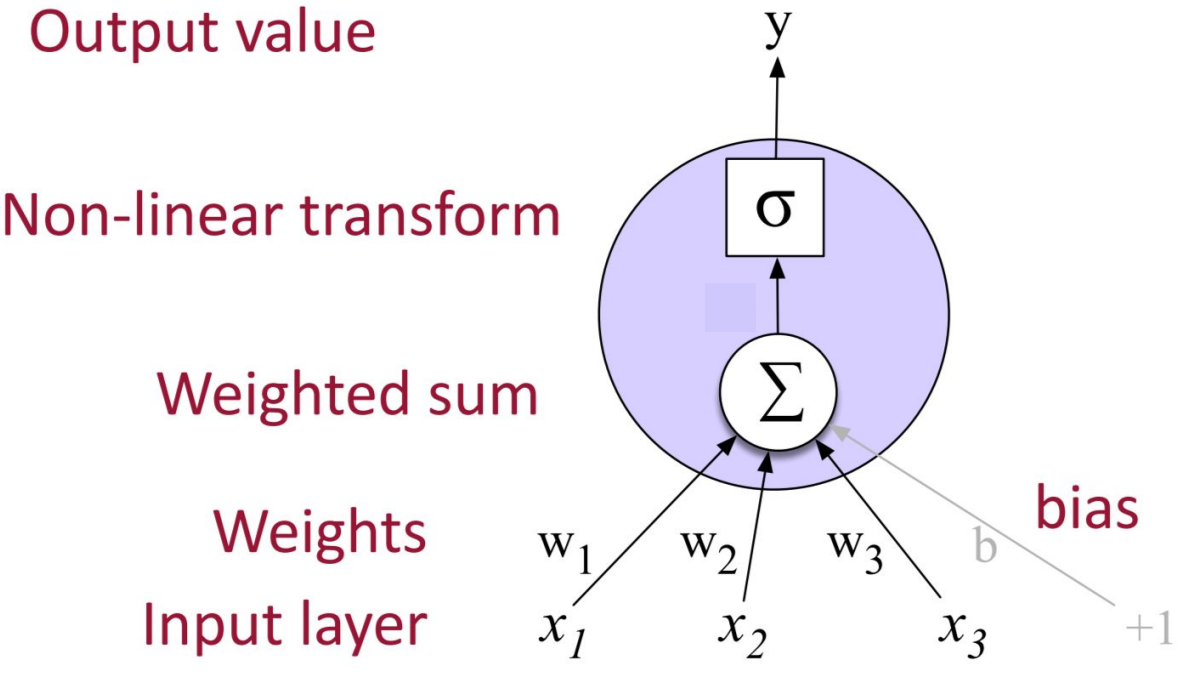
- Hundreds of blog posts and tutorials

This is in your brain



By BruceBlaus - Own work, CC BY 3.0,
https://commons.wikimedia.org/w/index.php?curid=28761830

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:

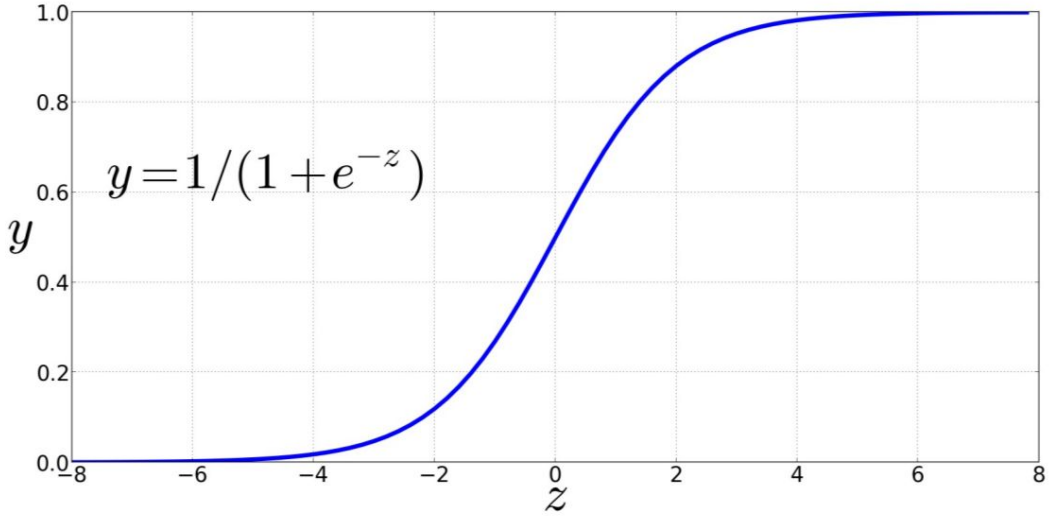
$$y = a = f(z)$$

Non-Linear Activation Functions

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

Sigmoid

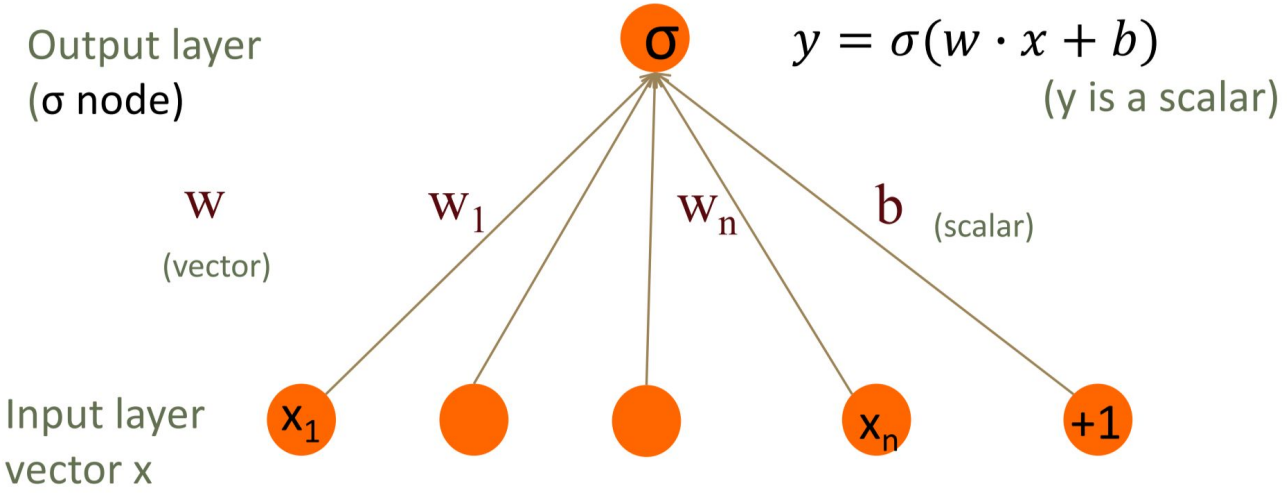
$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$



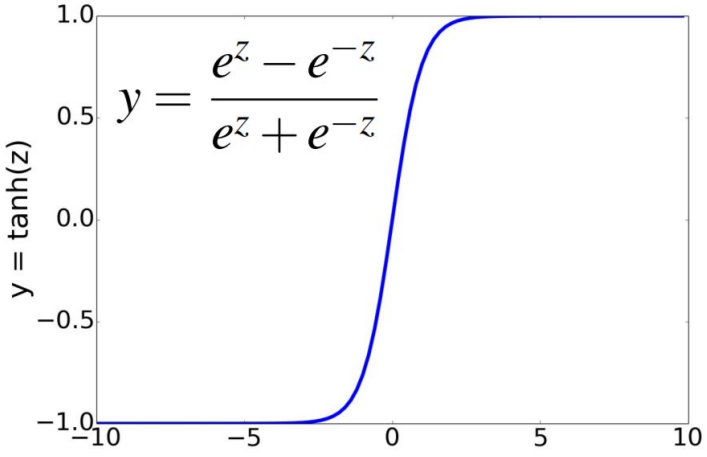
Final function the unit is computing

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

Binary Logistic Regression as a 1-layer network

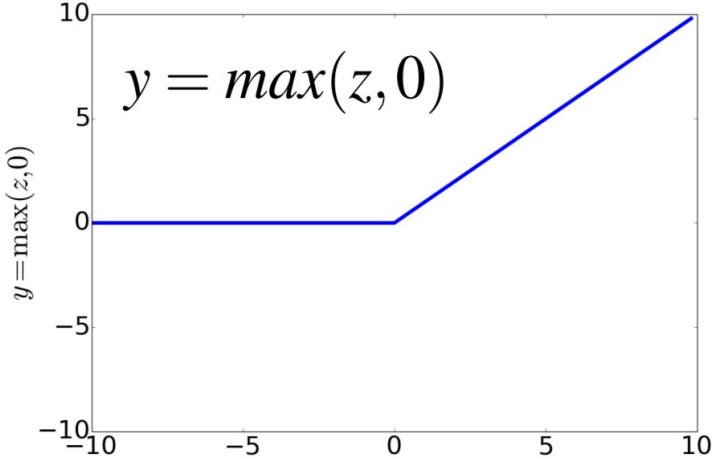


Non-Linear Activation Functions besides sigmoid



tanh

Most Common:



ReLU
Rectified Linear Unit

Final unit again

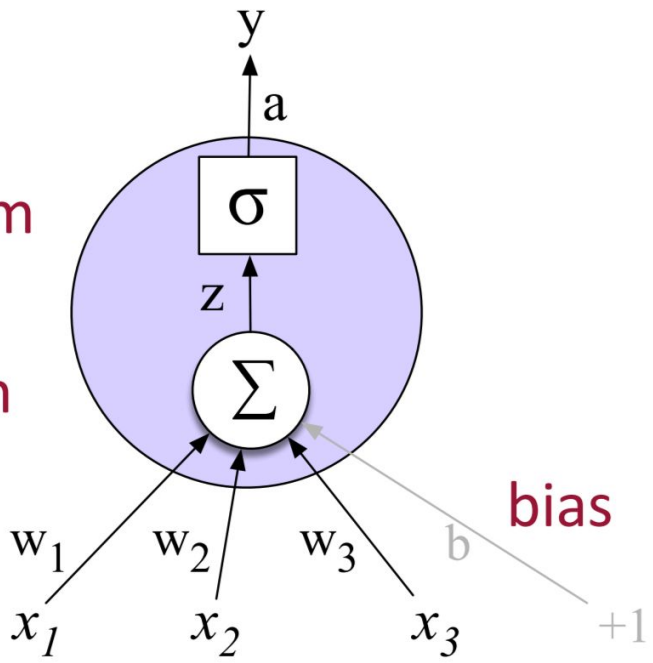
Output value

Non-linear transform

Weighted sum

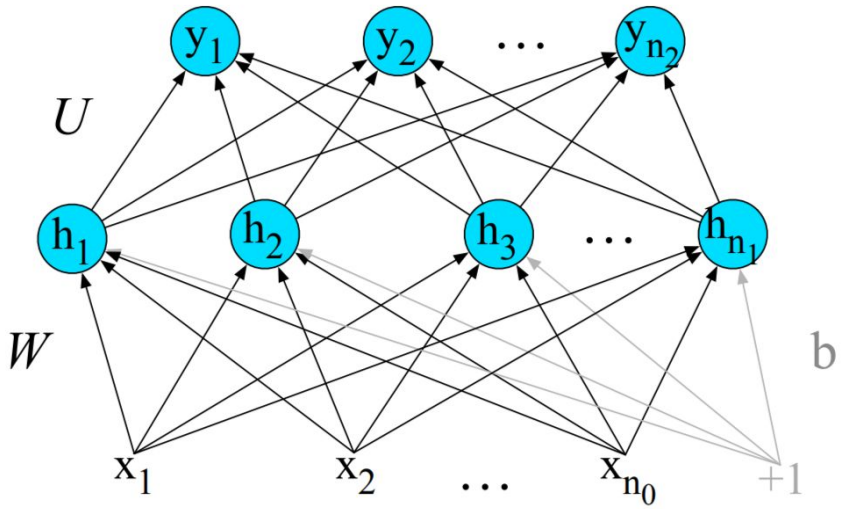
Weights

Input layer

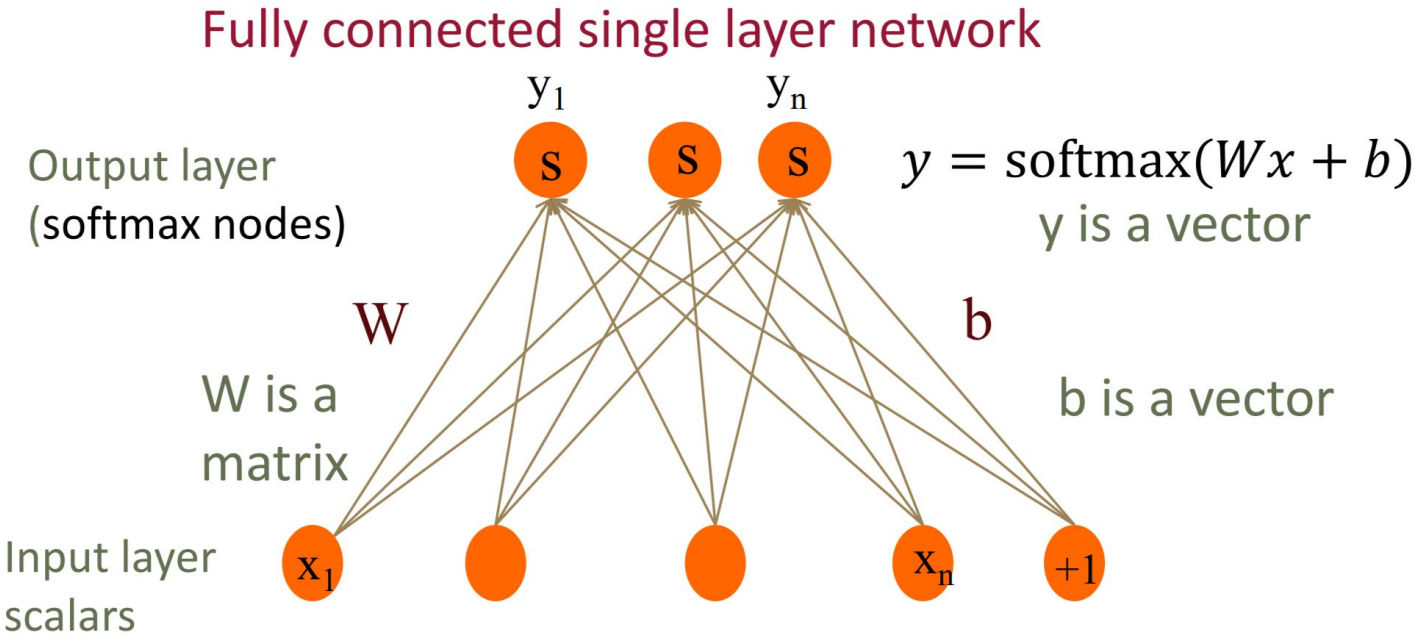


Feedforward Neural Networks

- 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:

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

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

Example:

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

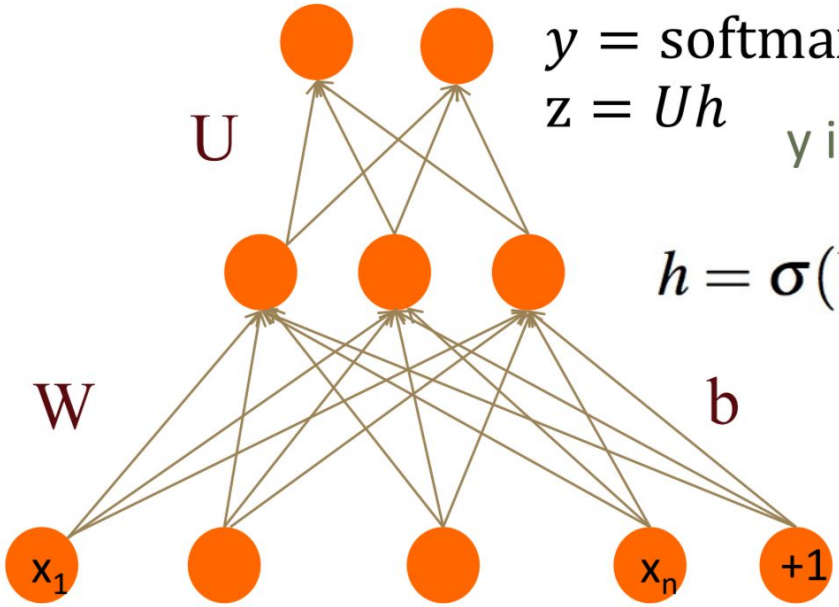
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Two-Layer Network with softmax output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)



$y = \text{softmax}(z)$

$z = Uh$

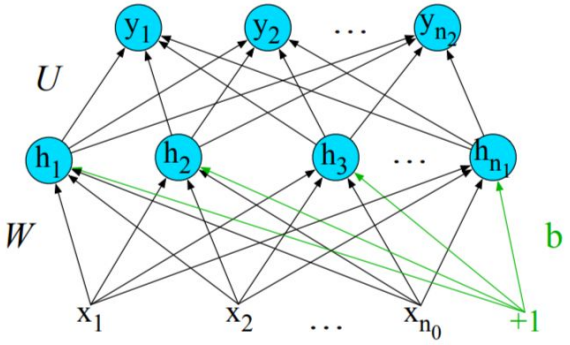
y is a vector

$h = \sigma(Wx + b)$

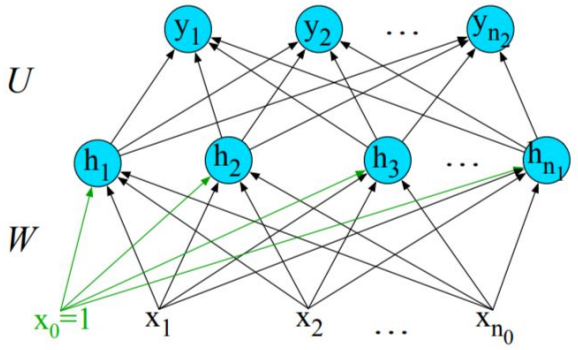
Could be ReLU
Or tanh

Replacing the bias unit

Instead of:



We'll do this:

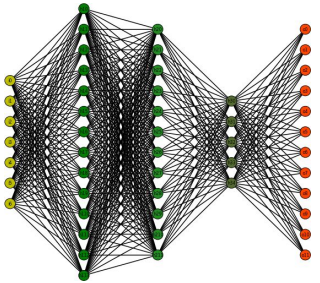


Learning the weights

- 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 Friday), but simpler models are useful to consider!

Common methods for getting short dense vectors

- In count-based models - Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- “Neural Language Model”-inspired models
  - The weight matrix in the input layer is often used as “word embeddings”
  - Compute one embeddings for each word (word types)
- Alternative to these "static embeddings":
  - Contextual Embeddings (ELMo, BERT)
  - Compute distinct embeddings for a word in its context
  - Separate embeddings for each token of a word

Simple static embeddings you can download!

word2vec (Mikolov et al)

https://code.google.com/archive/p/word2vec/

GloVe (Pennington, Socher, Manning)

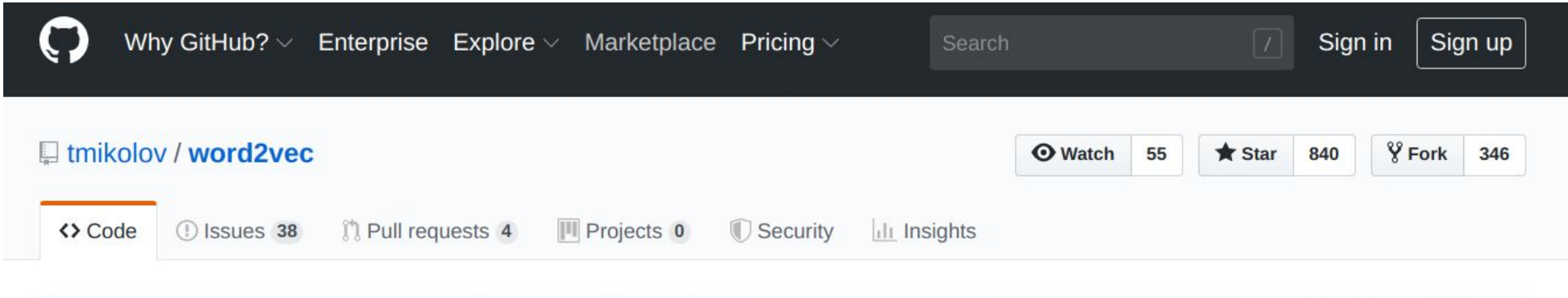
http://nlp.stanford.edu/projects/glove/

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

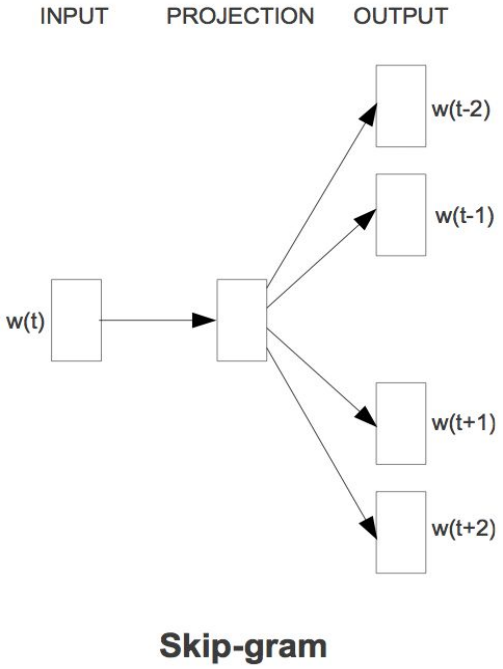
- Popular embedding method
- Very fast to train
- Code available on the web
- Idea: predict rather than count



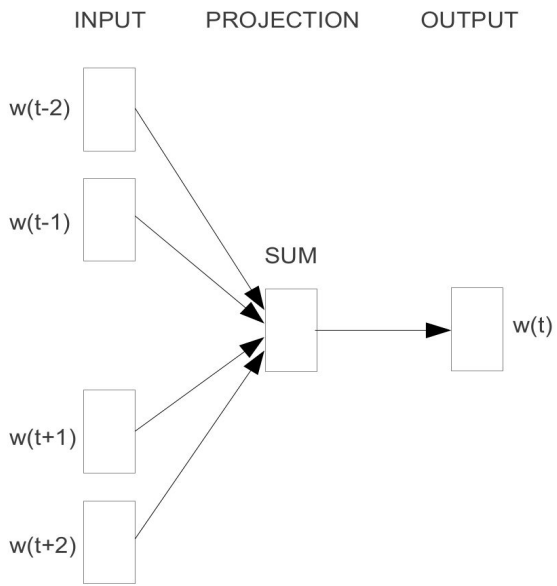
word2vec

- 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
- Big idea: **self-supervision**:
  - A word c that occurs near apricot in the corpus cats as the gold "correct
  - answer" for supervised learning
  - No need for human labels
  - Bengio et al. (2003); Collobert et al. (2011)

word2Vec



Skip-gram



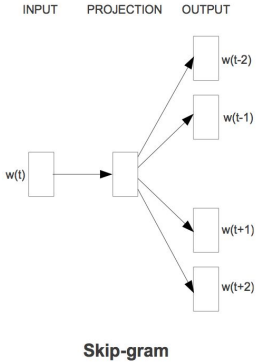
CBOW

- [Mikolov et al.' 13]

Skip-gram Prediction

- Predict vs Count

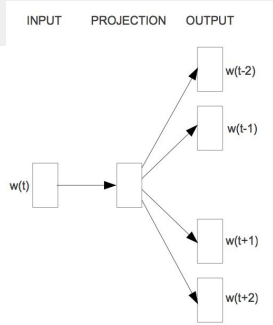
the cat sat on the mat



Skip-gram

Skip-gram Prediction

- Predict vs Count



Skip-gram

the cat sat on the mat

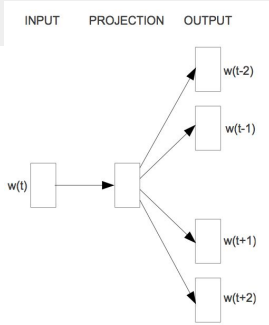
w_t = the → **CLASSIFIER** →
w_{t-2} = <start$_{-2}$>
w_{t-1} = <start$_{-1}$>
w_{t+1} = cat
w_{t+2} = sat

context size = 2

Skip-gram Prediction

- Predict vs Count

the cat sat on the mat



Skip-gram

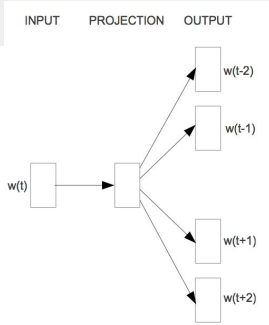
w_t = cat → **CLASSIFIER** →
w_{t-2} = <start$_{-1}$>
w_{t-1} = the
w_{t+1} = sat
w_{t+2} = on

context size = 2

Skip-gram Prediction

- Predict vs Count

the cat sat on the mat



Skip-gram

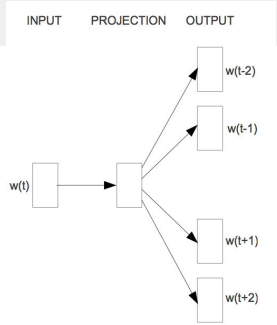
w_t = sat → **CLASSIFIER** →
w_{t-2} = the
w_{t-1} = cat
w_{t+1} = on
w_{t+2} = the

context size = 2

Skip-gram Prediction

- Predict vs Count

the cat sat on the mat



Skip-gram

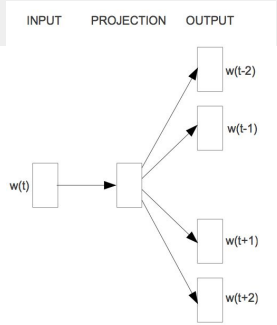
w_t = on → **CLASSIFIER** →
w_{t-2} = cat
w_{t-1} = sat
w_{t+1} = the
w_{t+2} = mat

context size = 2

Skip-gram Prediction

- Predict vs Count

the cat sat on the mat



Skip-gram

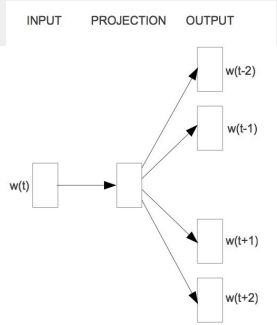
w_t = the → **CLASSIFIER** →
w_{t-2} = sat
w_{t-1} = on
w_{t+1} = mat
w_{t+2} = <end$_{+1}$>

context size = 2

Skip-gram Prediction

- Predict vs Count

the cat sat on the mat



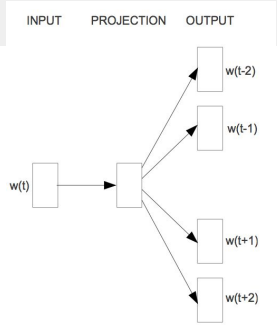
Skip-gram

w_t = mat → **CLASSIFIER** →
w_{t-2} = on
w_{t-1} = the
w_{t+1} = <end$_{+1}$>
w_{t+2} = <end$_{+2}$>

context size = 2

Skip-gram Prediction

- Predict vs Count



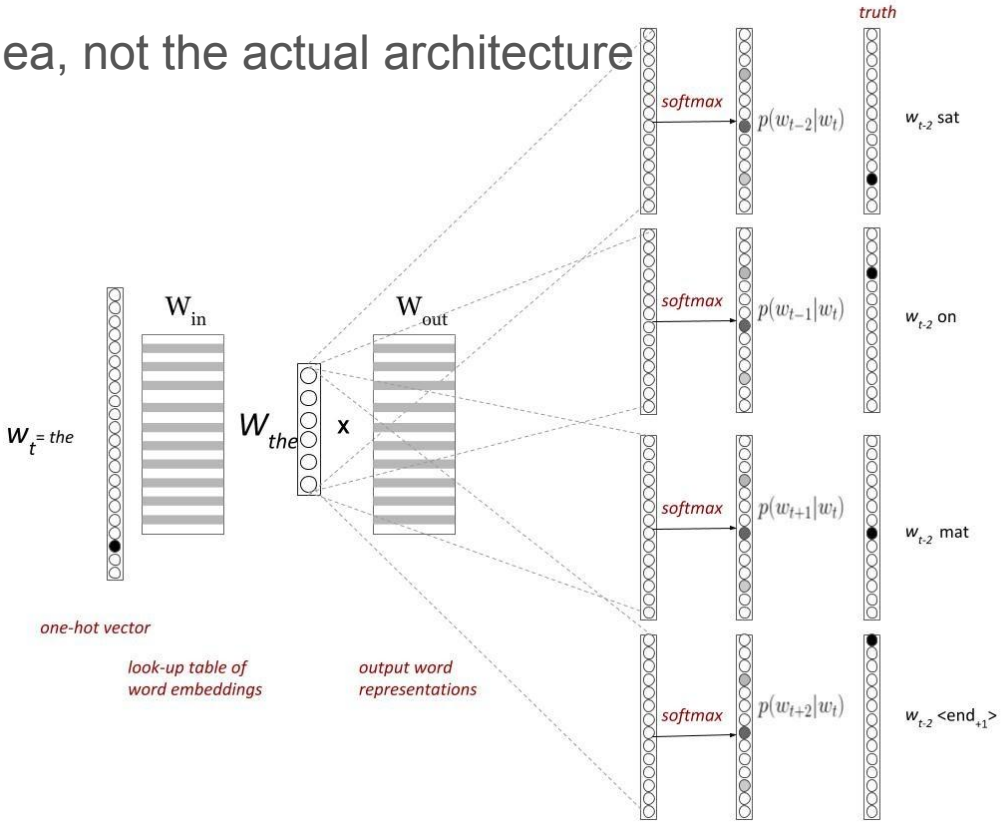
Skip-gram

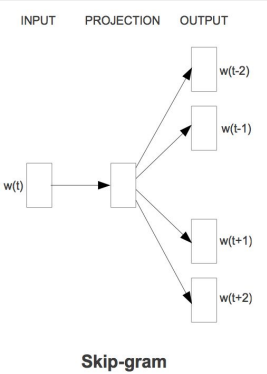
w_t = the → **CLASSIFIER** → w_{t-2} = sat, w_{t-1} = on, w_{t+1} = mat, w_{t+2} = <end$_{+1}$>

w_t = the → **CLASSIFIER** → w_{t-2} = <start$_{-2}$>, w_{t-1} = <start$_{-1}$>, w_{t+1} = cat, w_{t+2} = sat

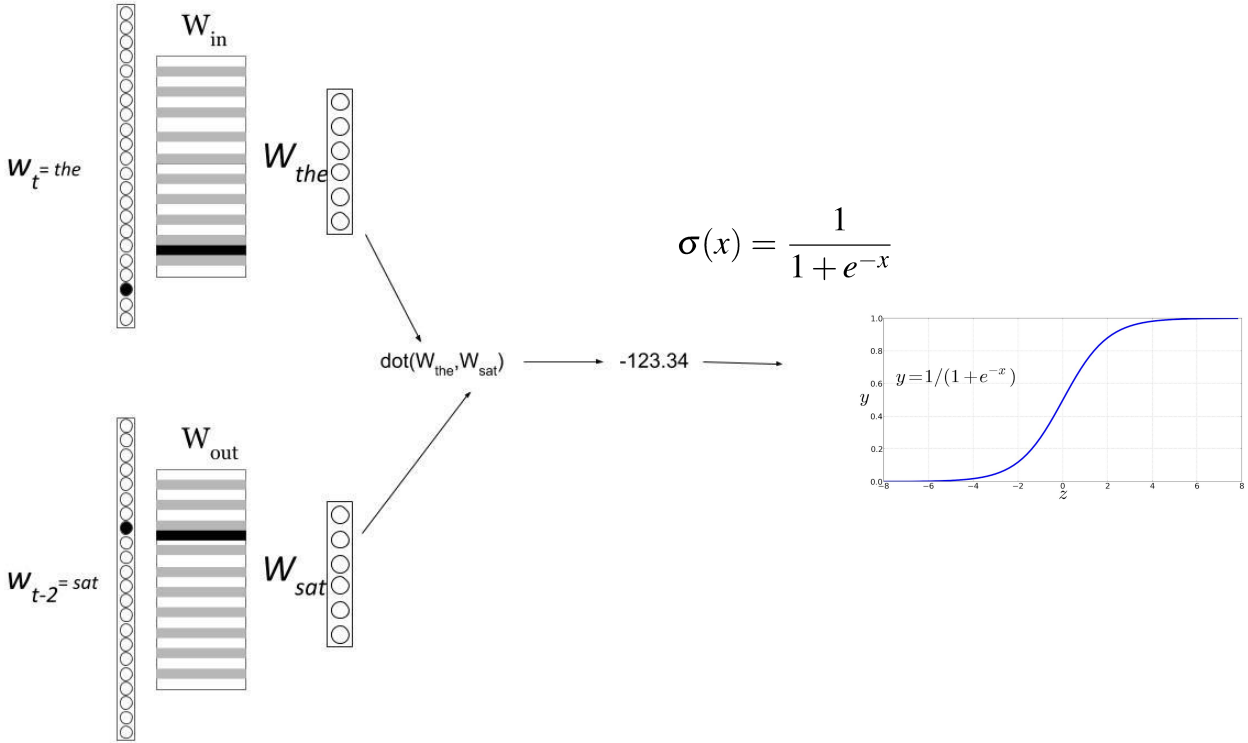
Skip-gram Prediction

- Conceptual idea, not the actual architecture





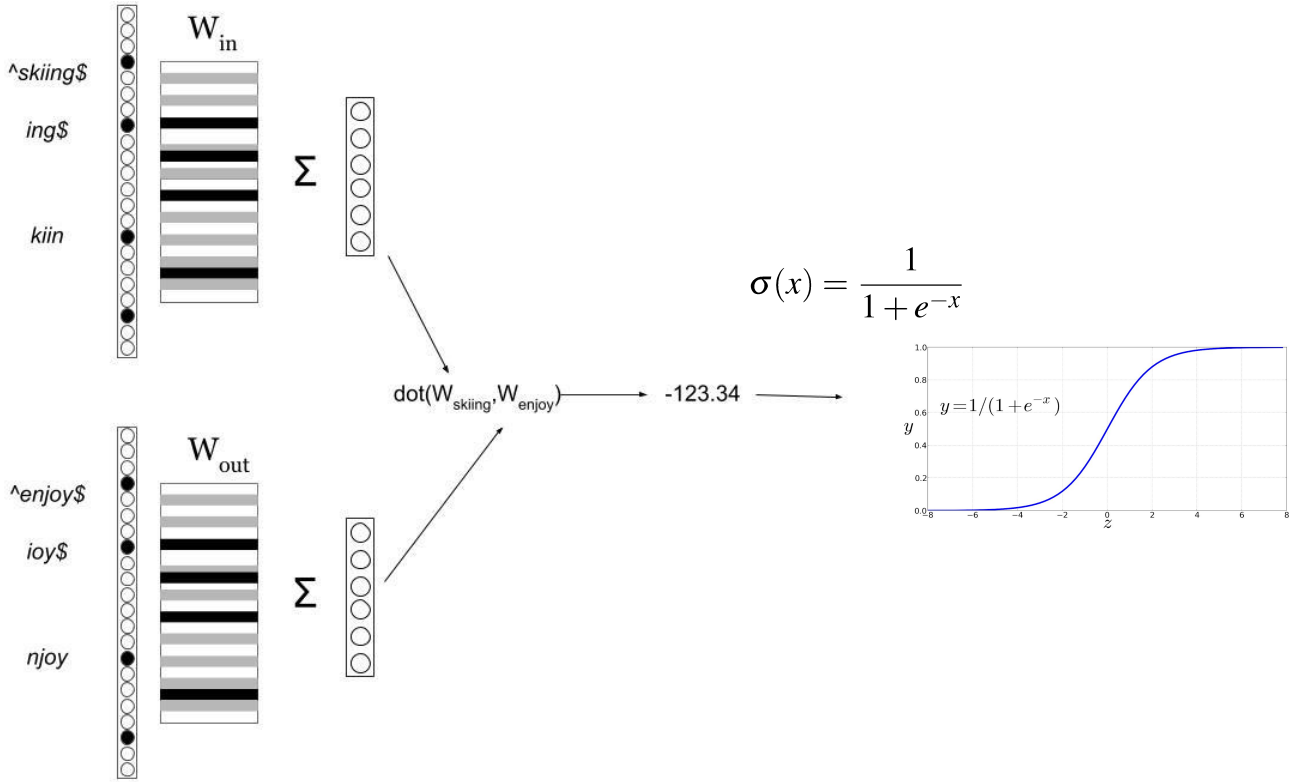
How to compute p(+|t,c)?



Approach: predict if candidate word c is a "neighbor"

1. Treat the target word *t* and a neighboring context word *c* as **positive examples**
2. Randomly sample other words in the lexicon to get negative examples
3. Use logistic regression to train a classifier to distinguish those two cases
4. Use the learned weights as the embeddings

FastText


W_in
^skiing$
ing$
kiin
Σ
dot(W_skiing,W_enjoy)
-123.34
W_out
^enjoy$
ioy$
njoy
Σ
σ(x) = 1 / (1 + e^−x)
y=1/(1+e^−x)
y
z

SGNS

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

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

Return probability that c is a real context word:

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

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

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

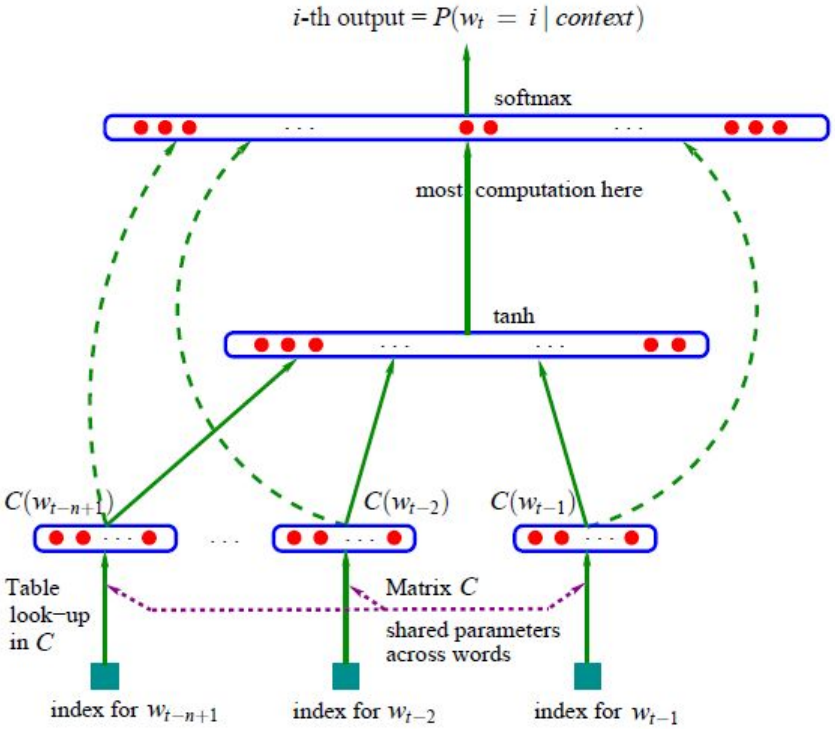


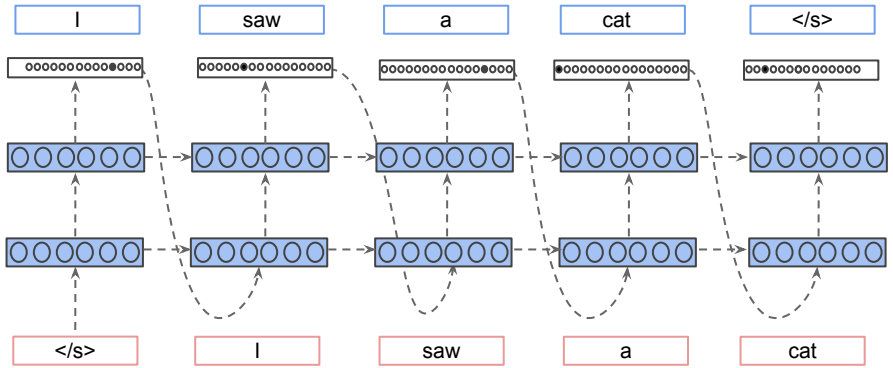
Image: (Bengio et al, 03)

Neural LMs

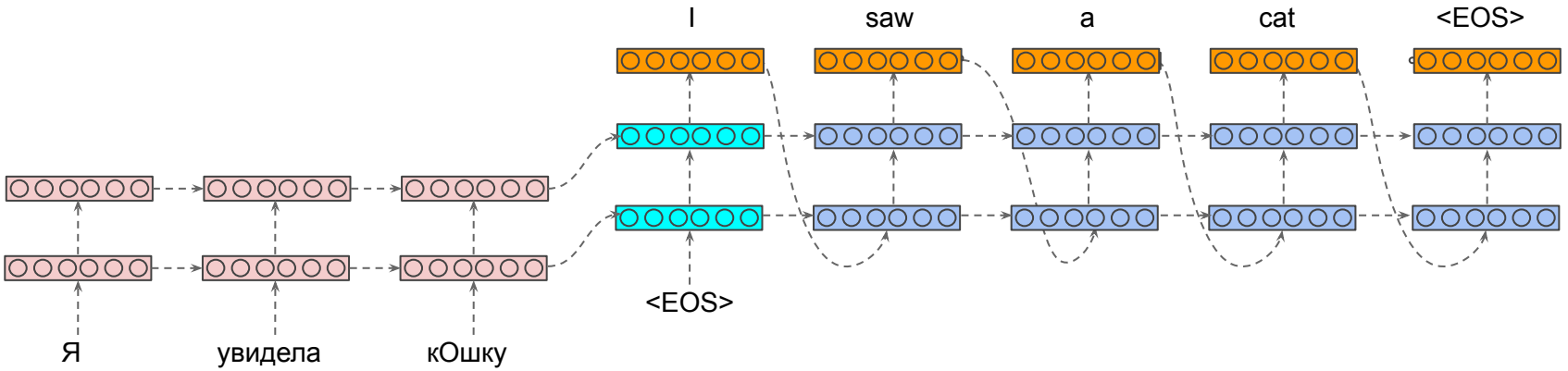
	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		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

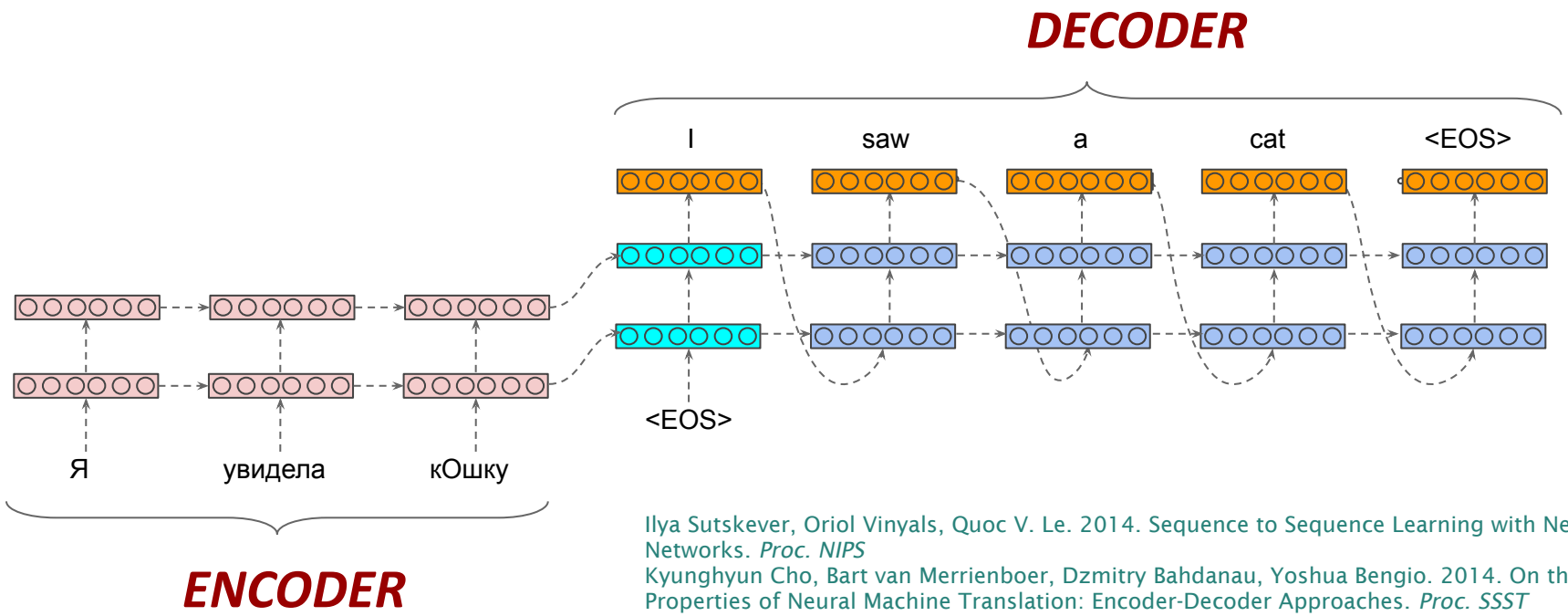


Sequence-to-Sequence Models



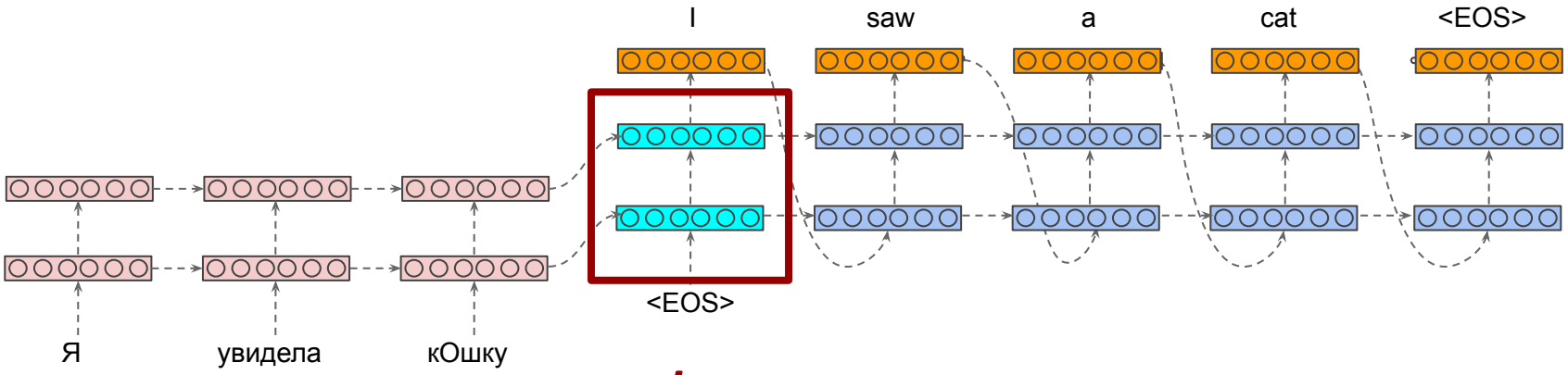
Ilya Sutskever, Oriol Vinyals, Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. Proc. NIPS

Sequence-to-Sequence Models for Neural Machine Translation



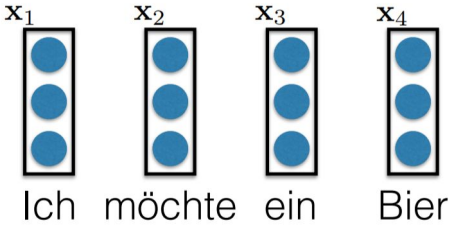
Ilya Sutskever, Oriol Vinyals, Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. *Proc. NIPS*
Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *Proc. SSST*

Sequence-to-Sequence Models for NMT

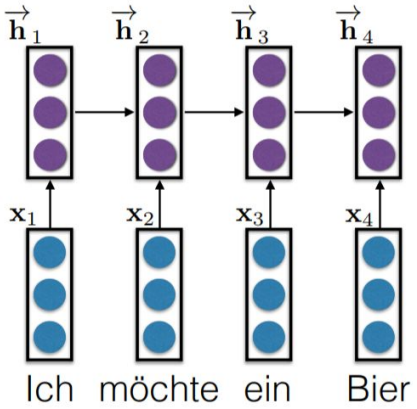


sentence representation

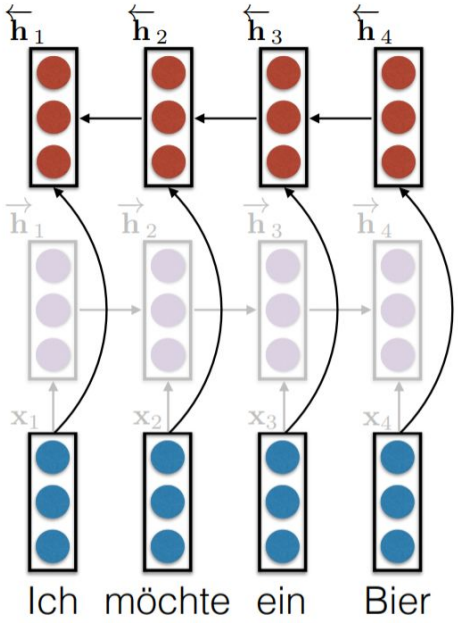
Encoder: Bidirectional RNN



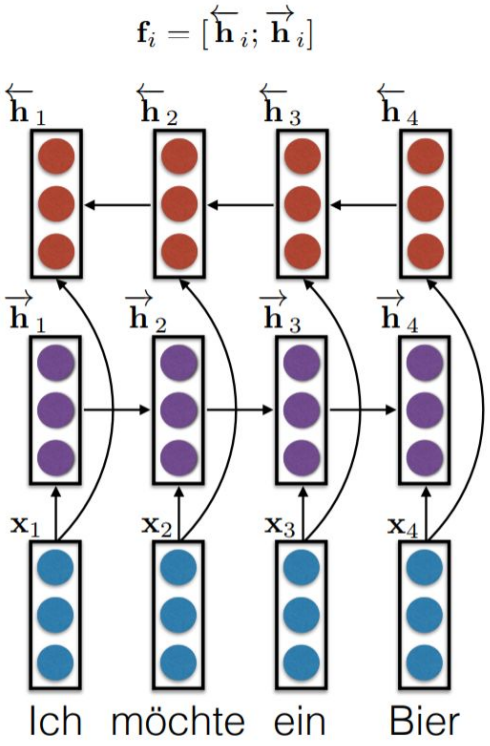
Encoder: Bidirectional RNN



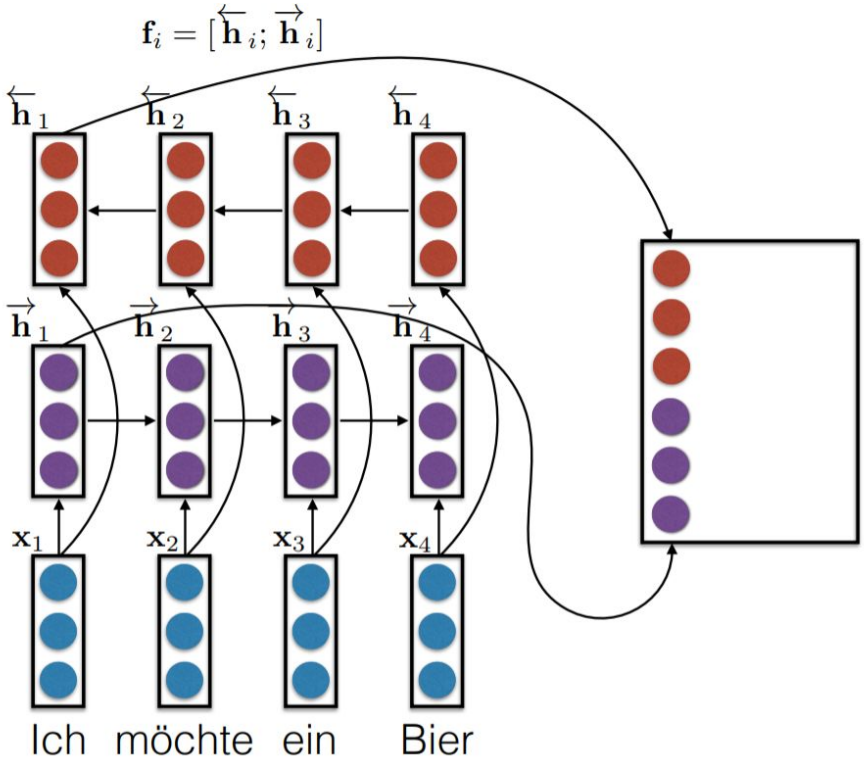
Encoder: Bidirectional RNN



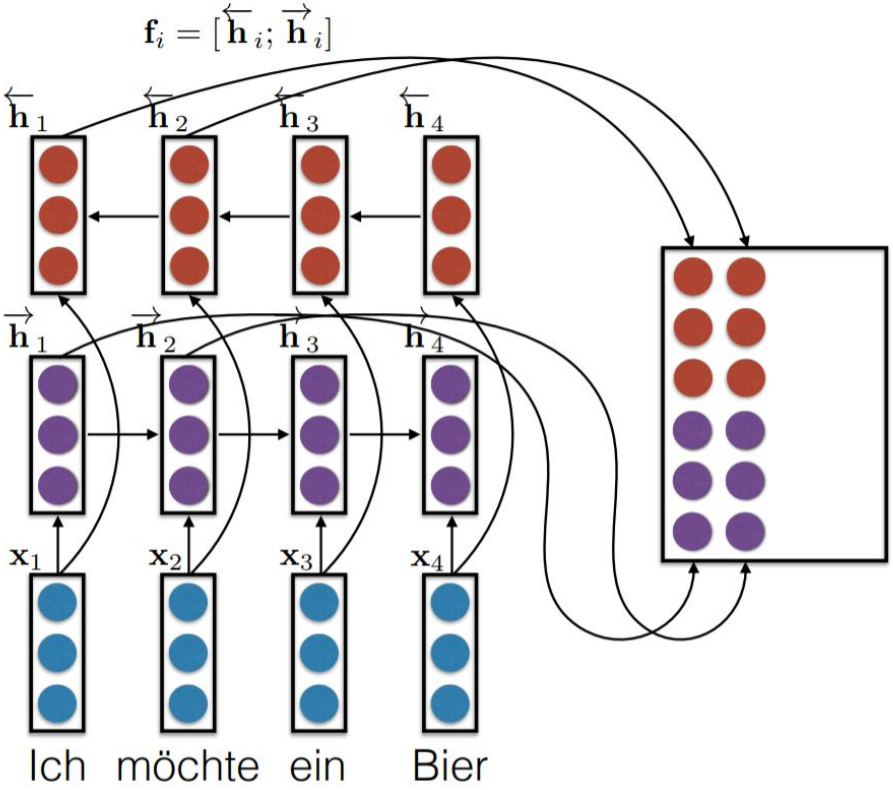
Encoder: Bidirectional RNN



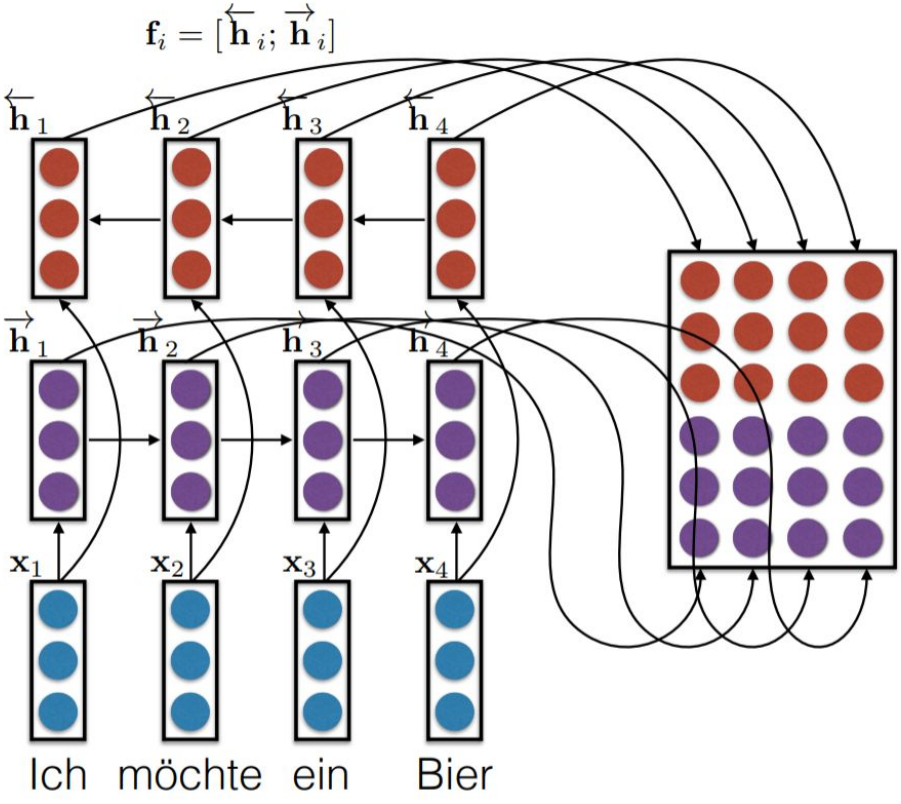
Encoder: Bidirectional RNN



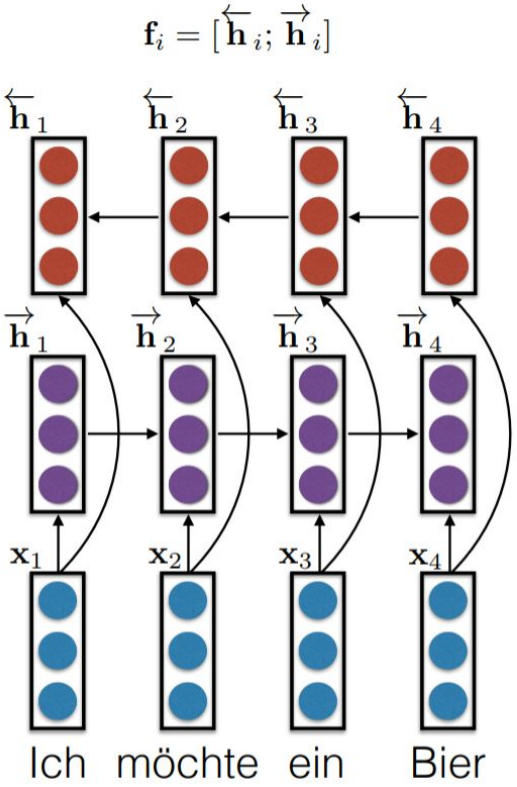
Encoder: Bidirectional RNN

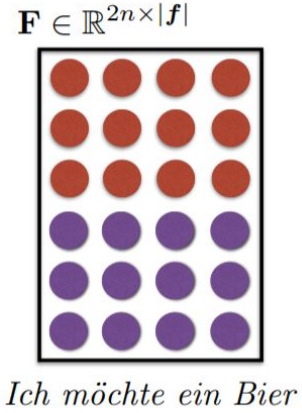


Encoder: Bidirectional RNN



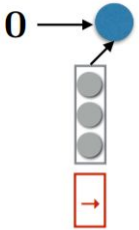
Matrix Sentence Encoding

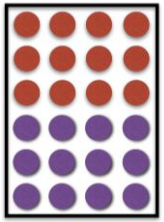




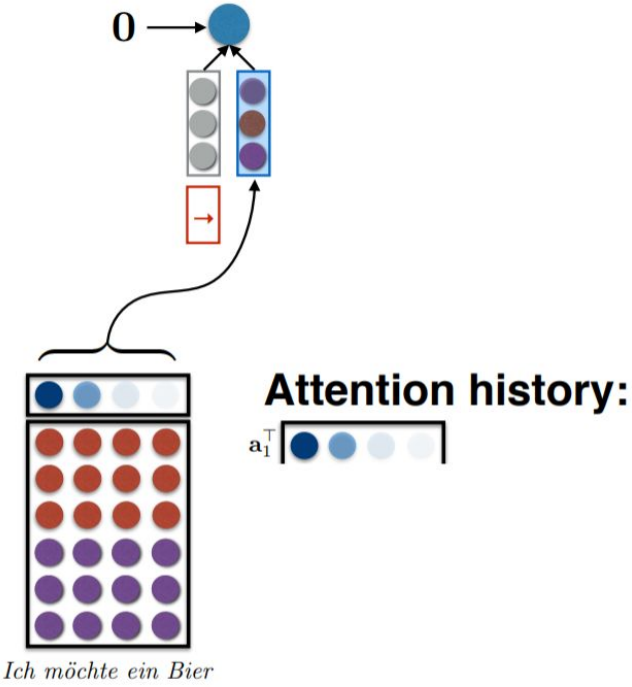
- matrix-encoded sentence

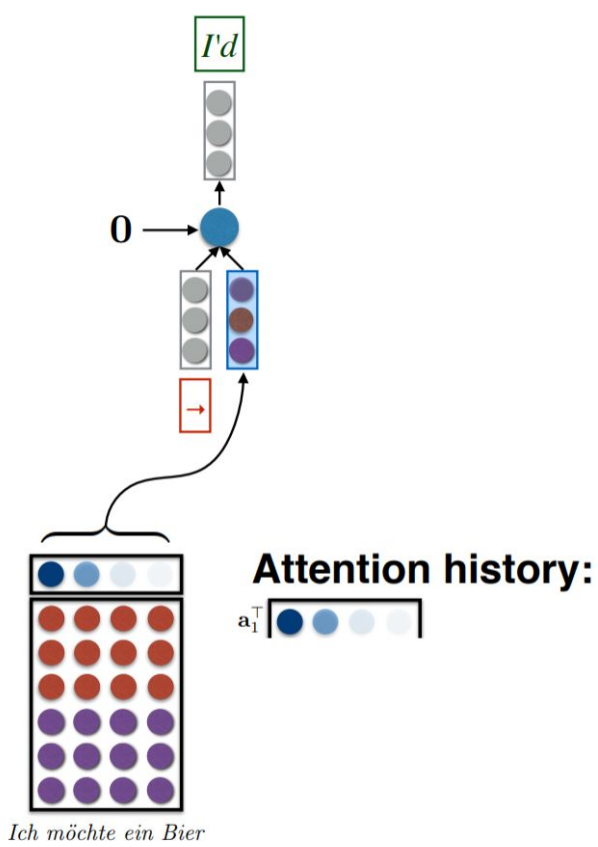
Decoder: RNN + Attention

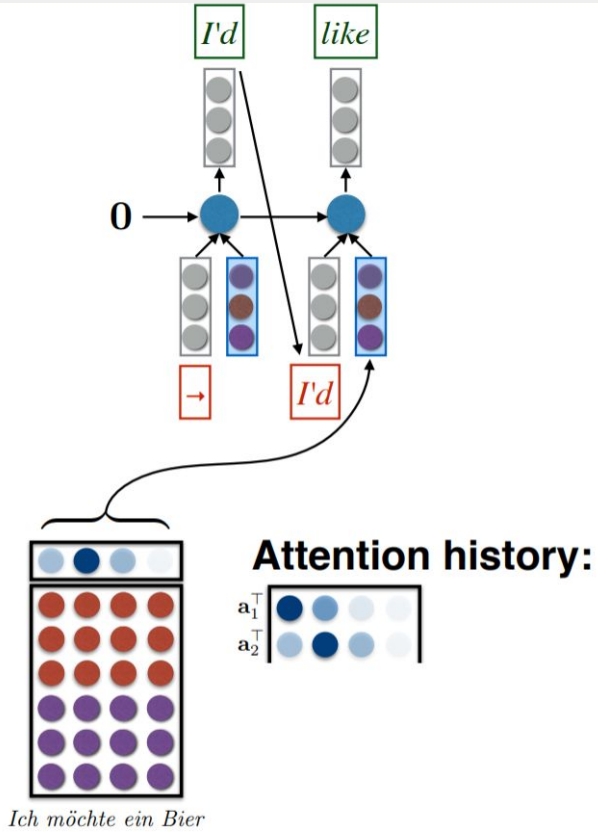


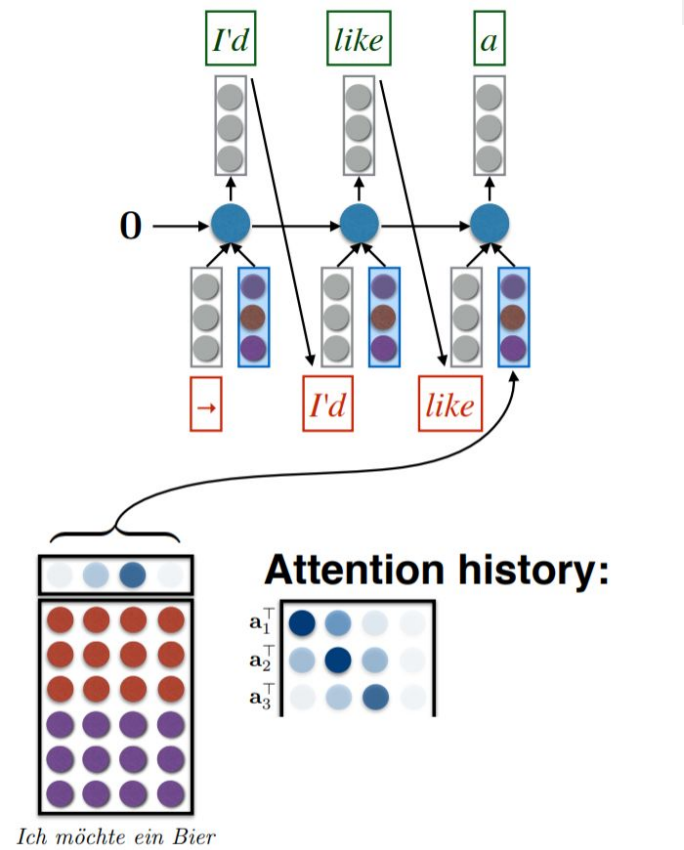


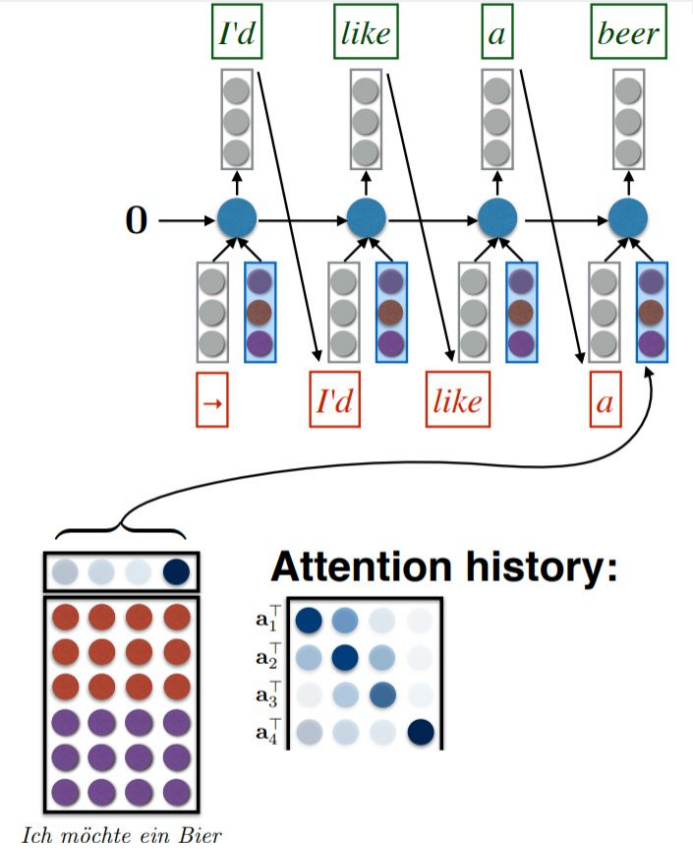
Ich möchte ein Bier

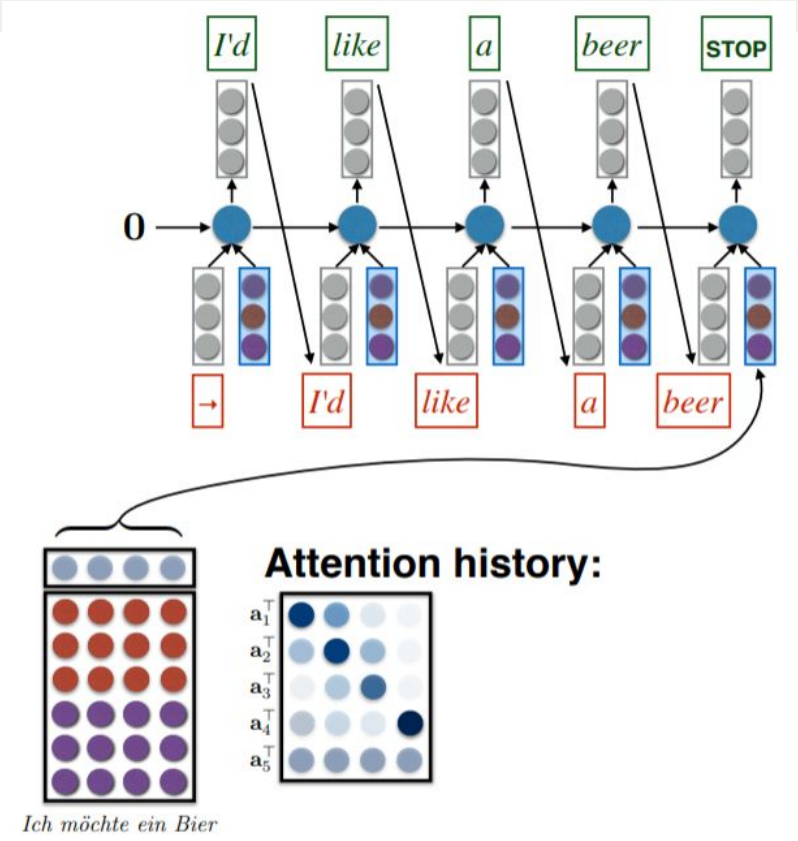

0
→
Attention history:
$\mathbf{a}_1^\top$
Ich möchte ein Bier


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


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


I'd
like
a
0
→
I'd
like
Attention history:
$\mathbf{a}_1^\top$
$\mathbf{a}_2^\top$
$\mathbf{a}_3^\top$
Ich möchte ein Bier


I'd
like
a
beer
0
→
I'd
like
a
Attention history:
Ich möchte ein Bier


I'd
like
a
beer
STOP
0
→
I'd
like
a
beer
Attention history:
Ich möchte ein Bier