Natural Language Processing

NN review and Neural LMs

Luke Zettlemoyer

lsz@cs.washington.edu

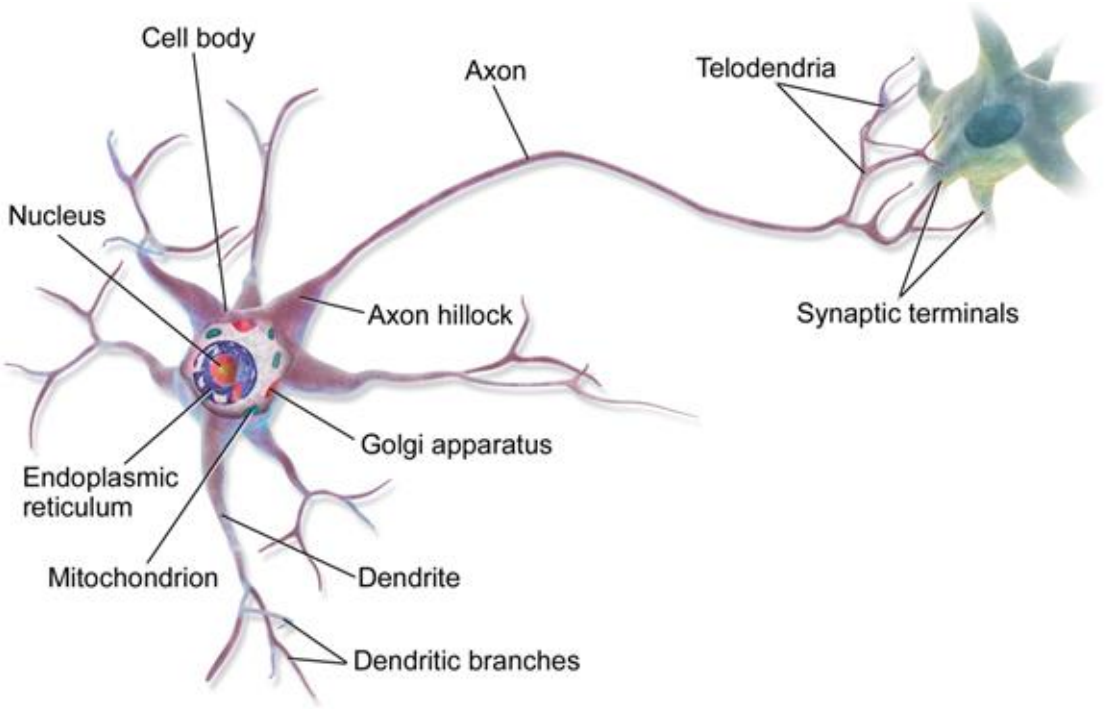
PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Readings

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

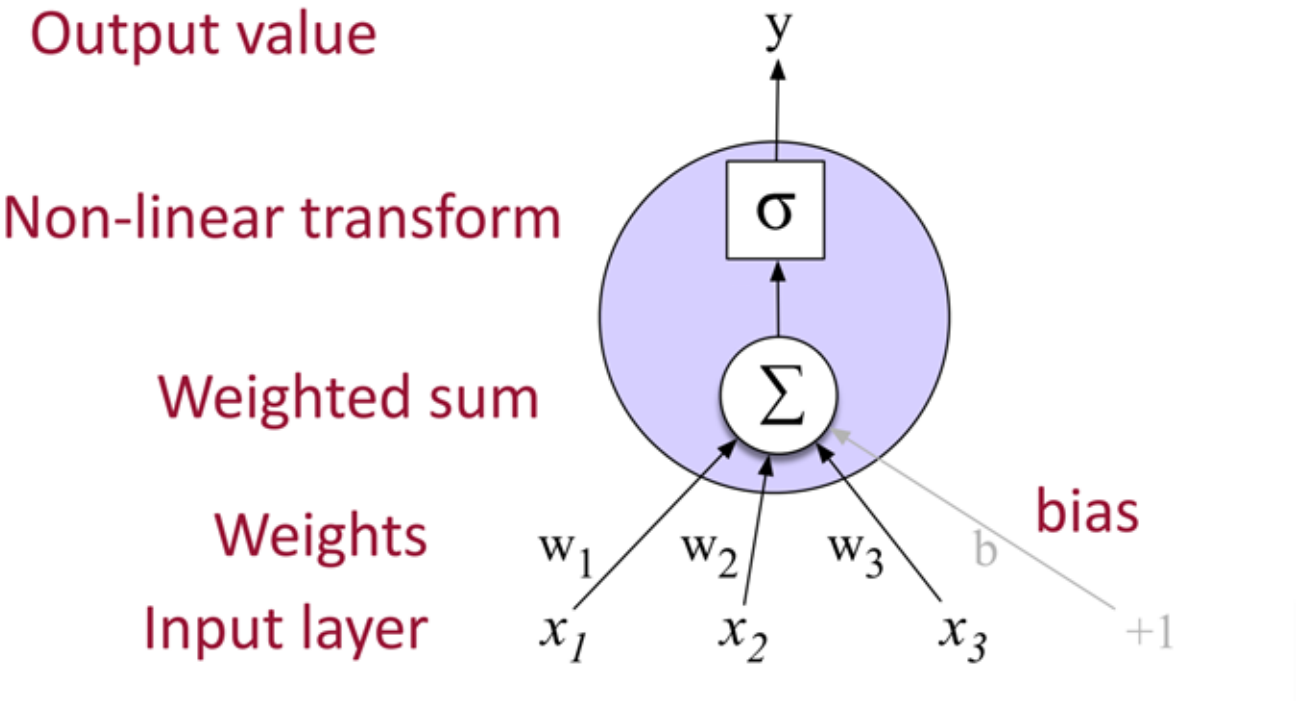
- Hundreds of blog posts and tutorials

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:

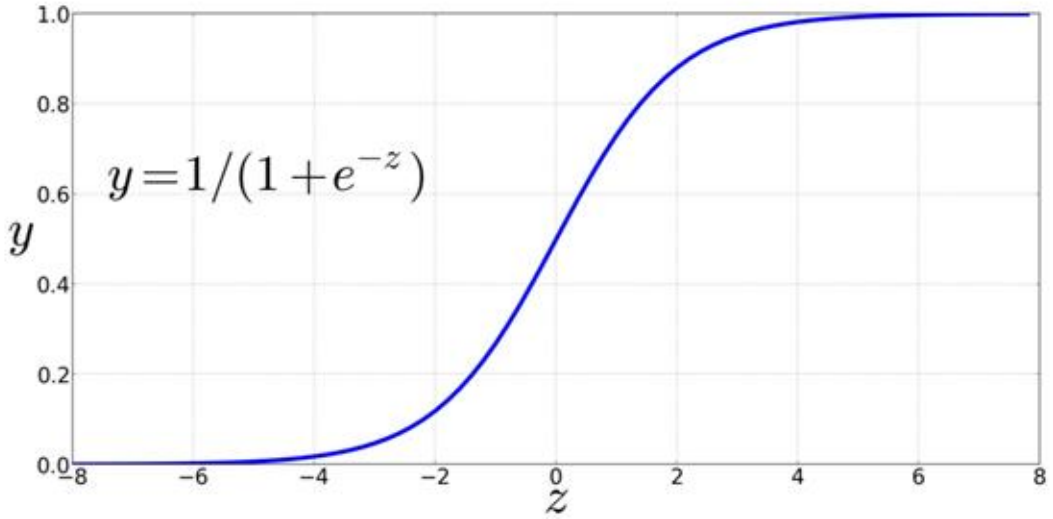
$$y = a = f(z)$$

Non-Linear Activation Functions

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

Sigmoid

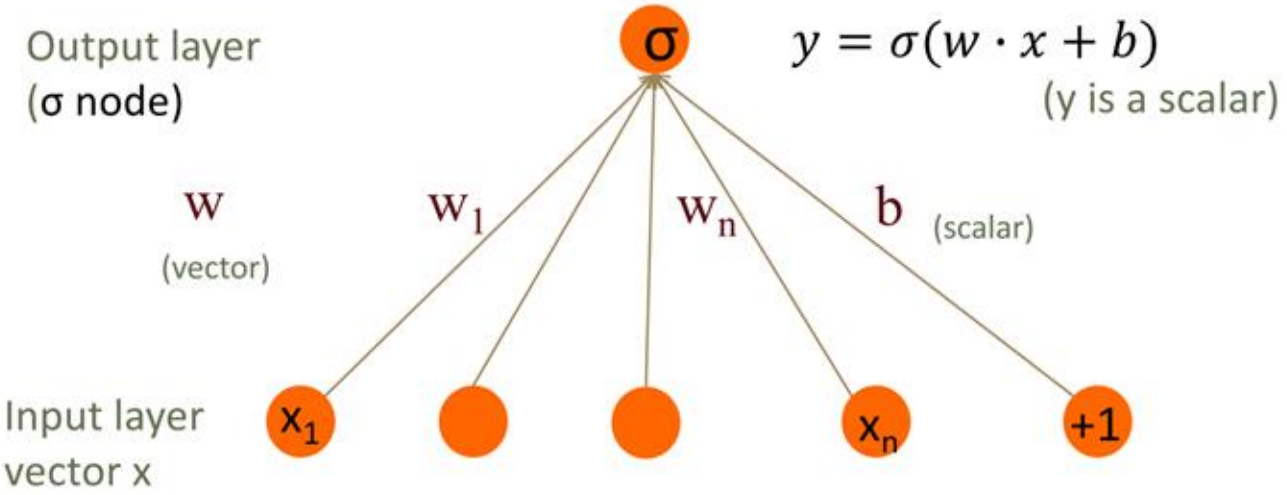
$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$



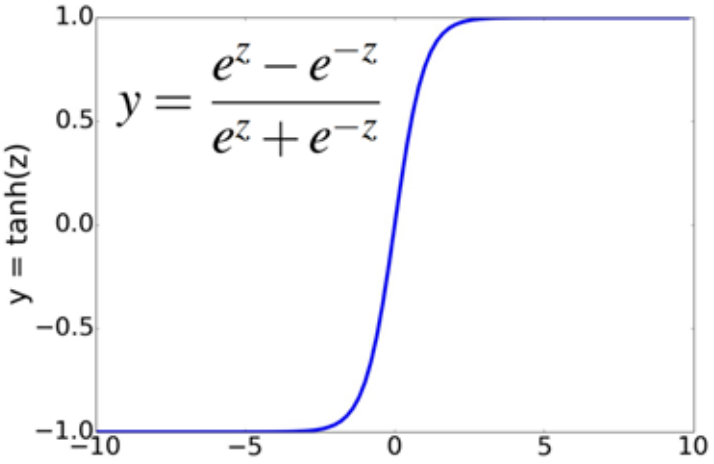
Final function the unit is computing

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

Binary Logistic Regression as a 1-layer network

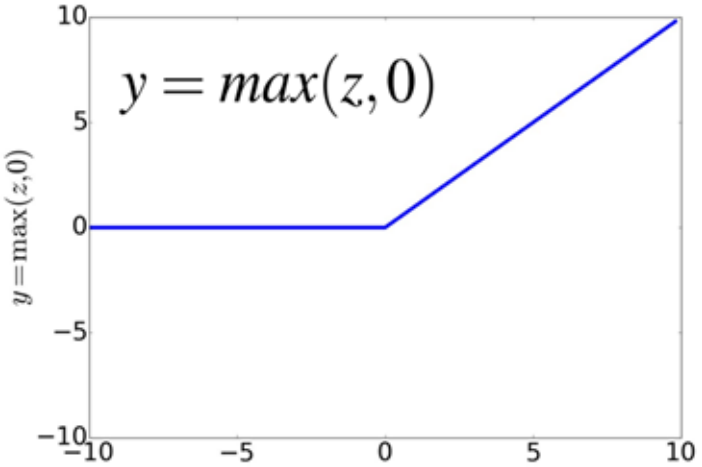


Non-Linear Activation Functions besides sigmoid



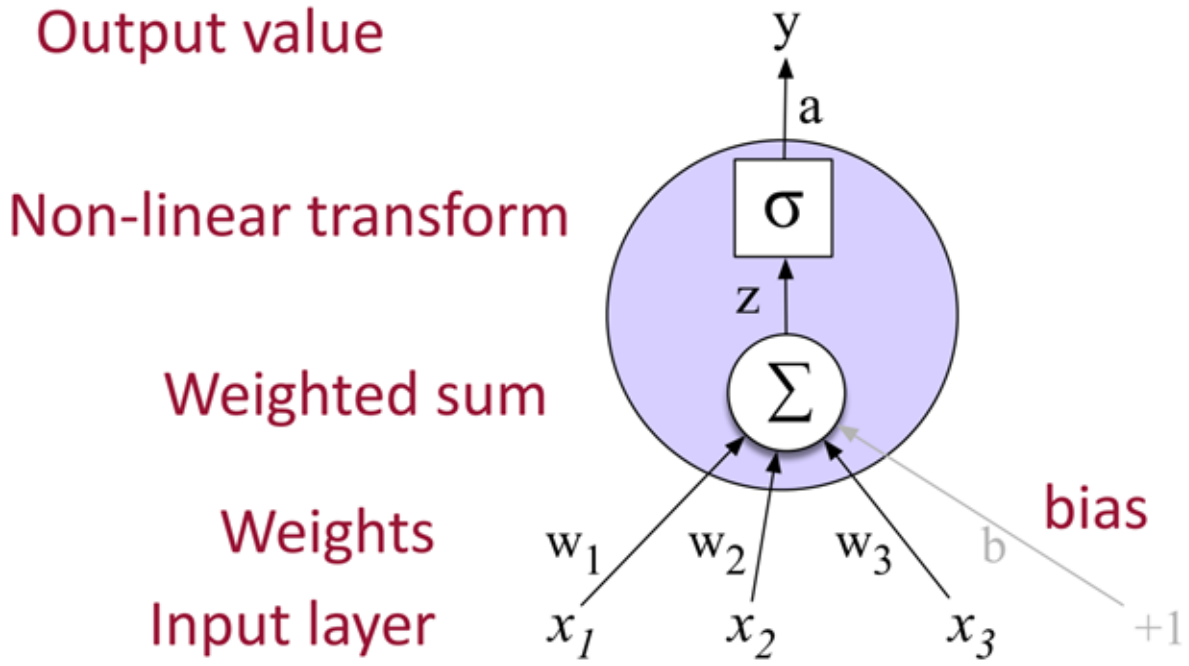
tanh

Most Common:



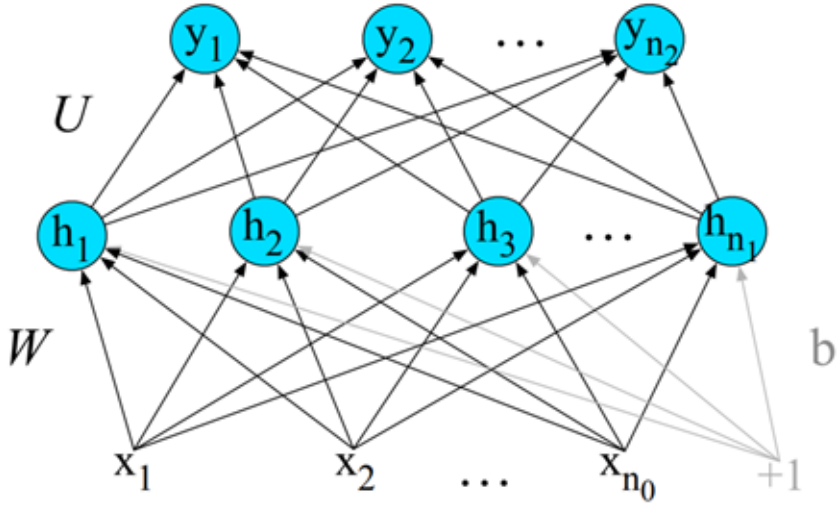
ReLU
Rectified Linear Unit

Final unit again

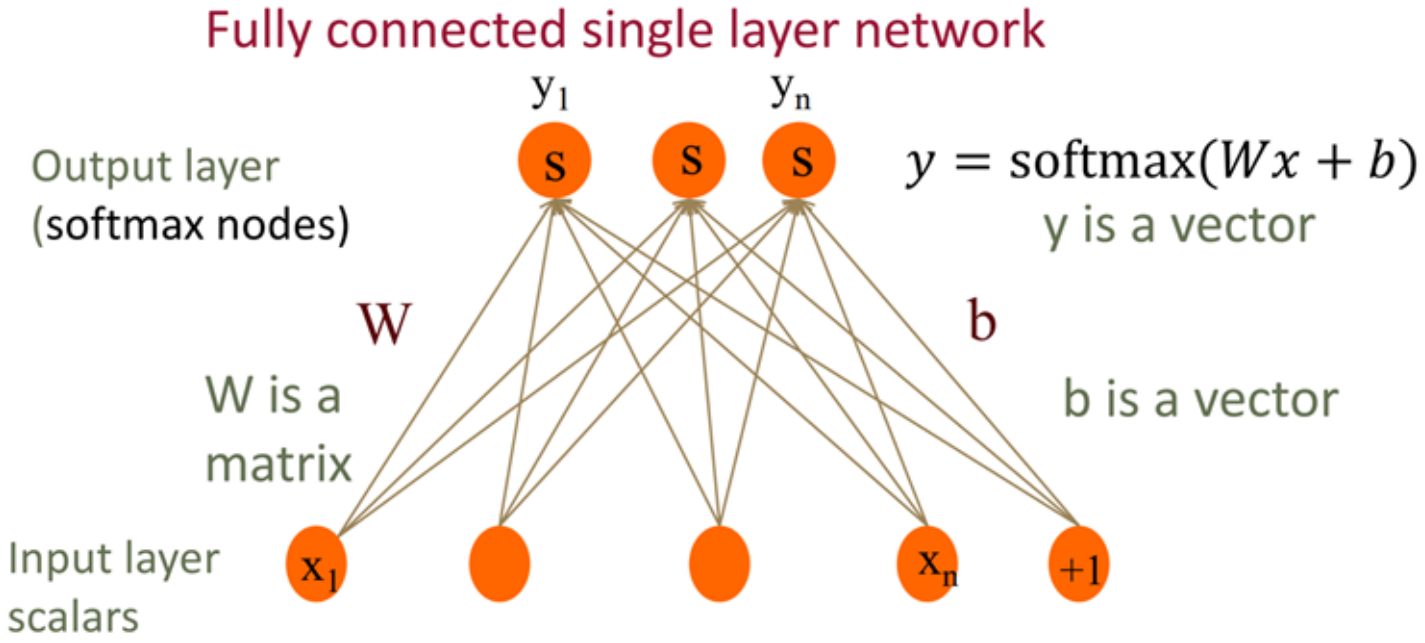


Feedforward Neural Networks

- 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:

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

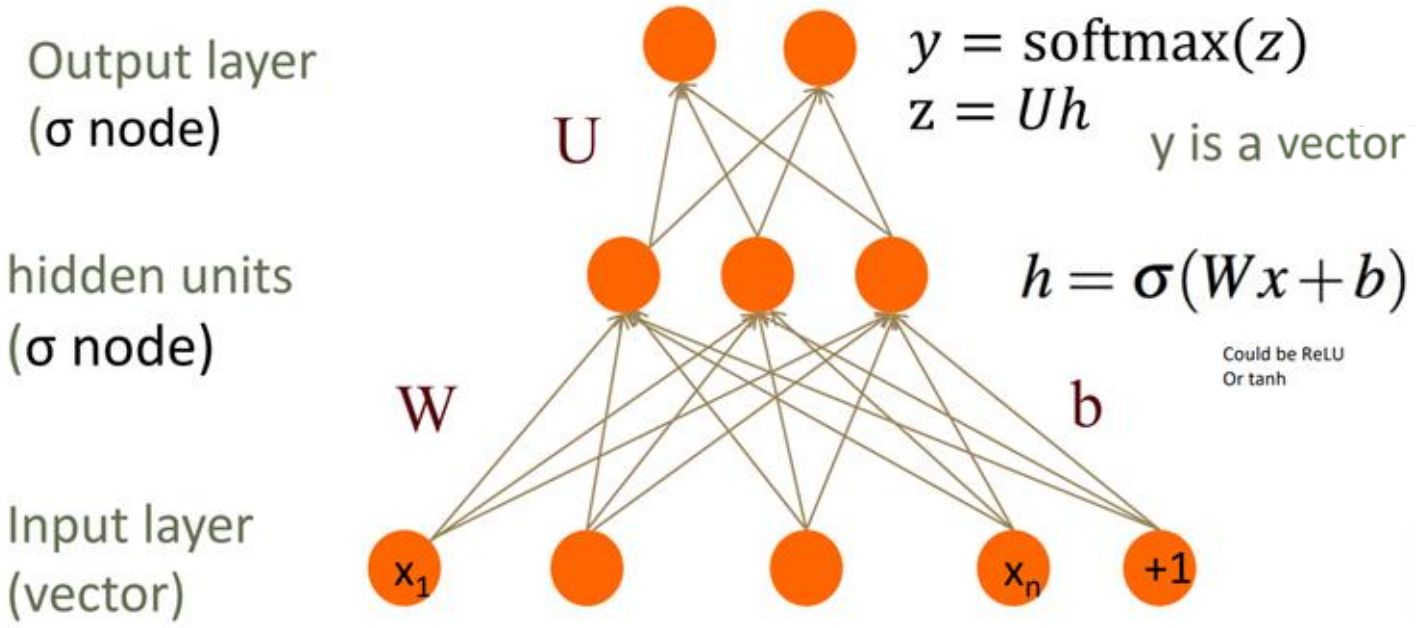
$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \leq i \leq k$$

Example:

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

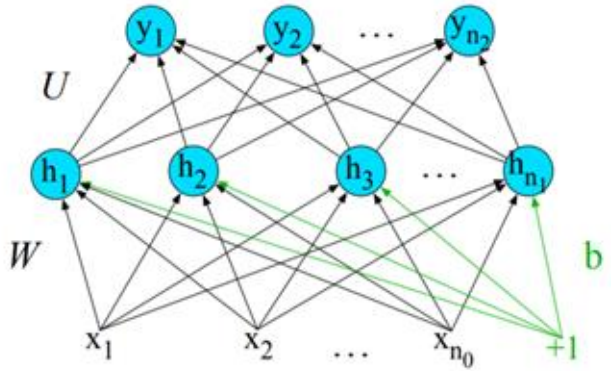
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Two-Layer Network with softmax output

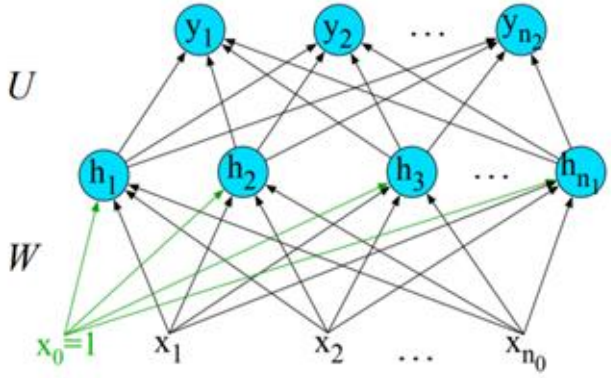


Replacing the bias unit

Instead of:



We'll do this:

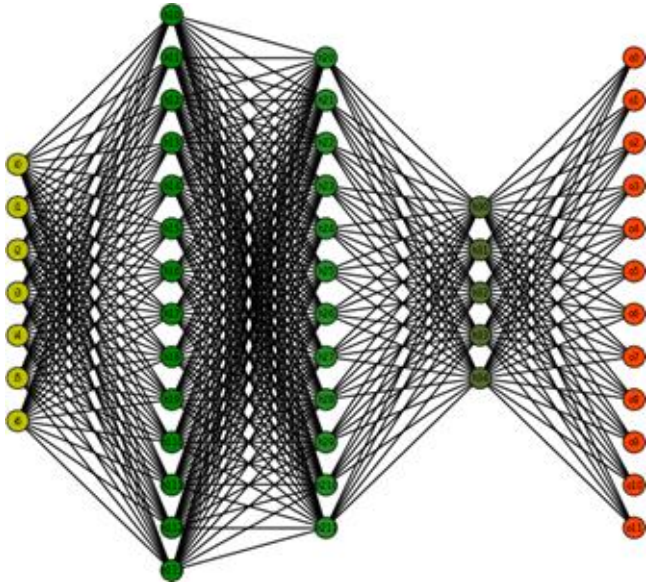


Learning the weights

- 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

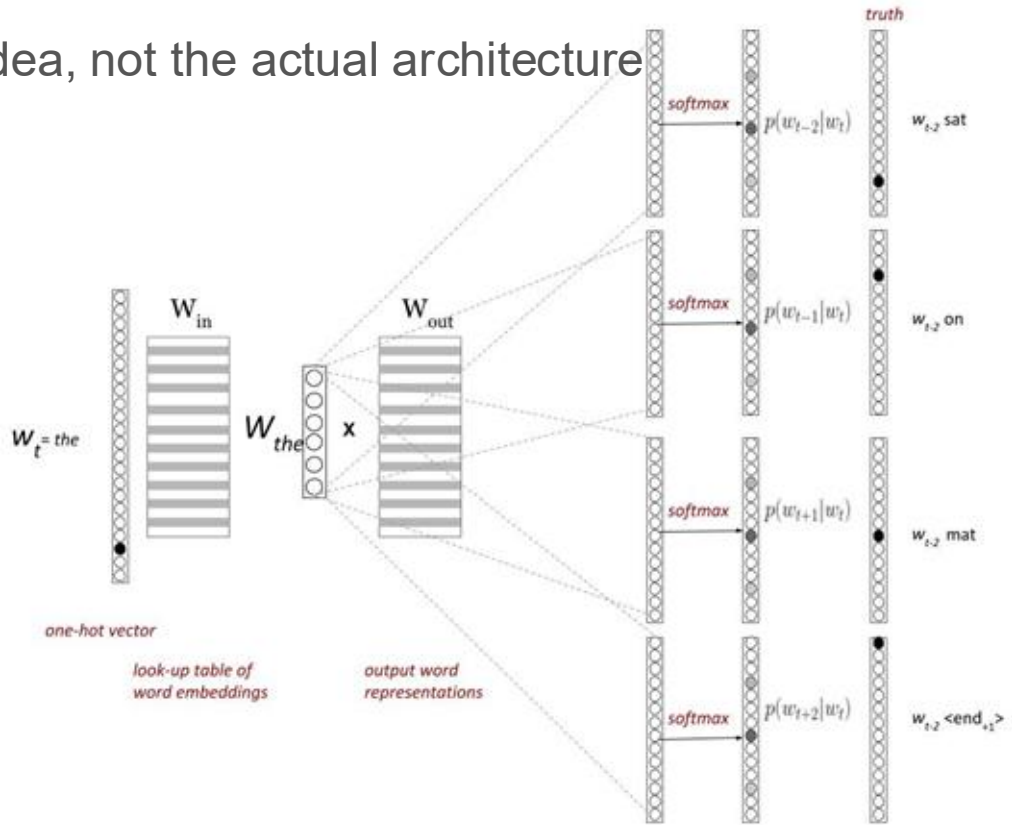
State of the art systems use more powerful neural architectures (we will learn transformers architectures on Friday), but simpler models are useful to consider!

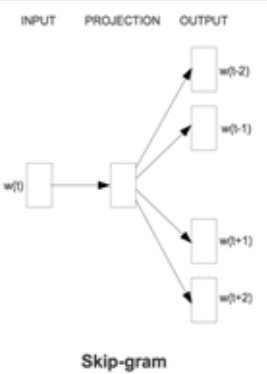
word2vec

- 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)

Skip-gram Prediction

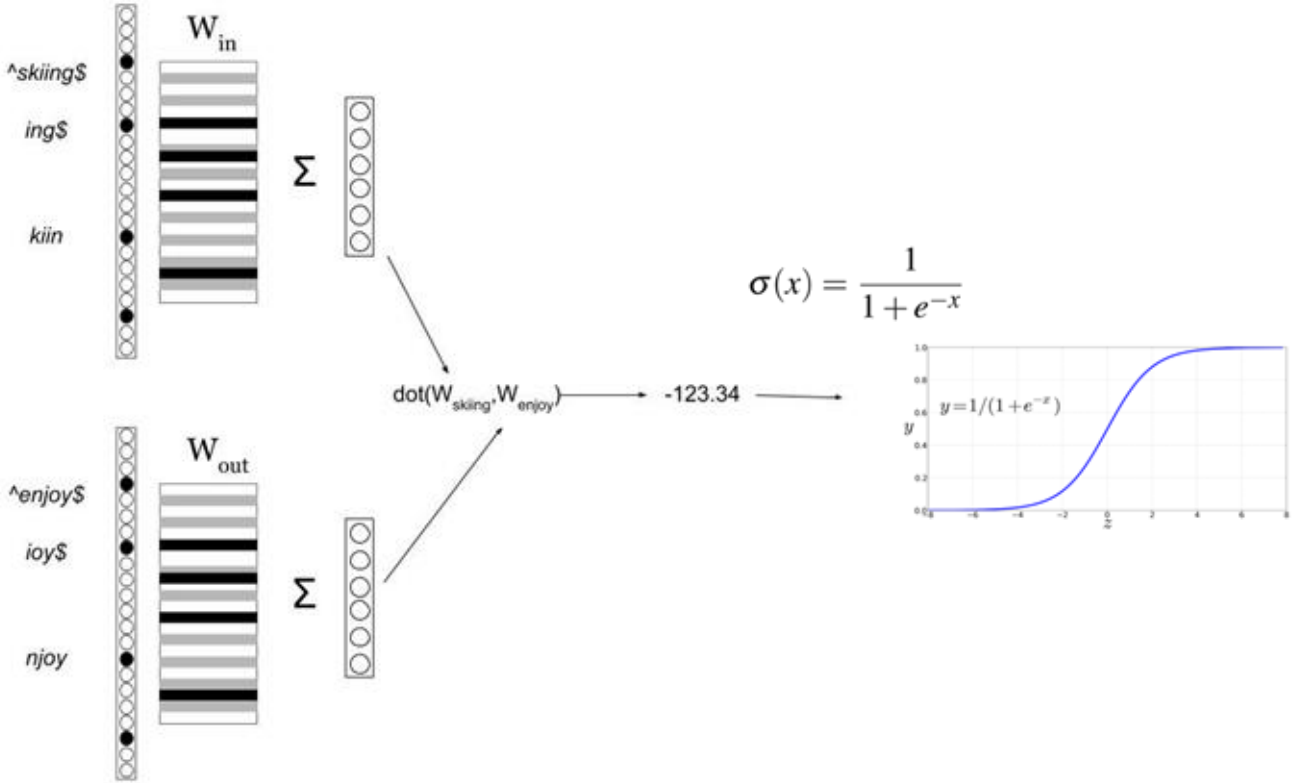
- Conceptual idea, not the actual architecture





Skip-gram

FastText


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

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

Preview: BERT as a text classification problem

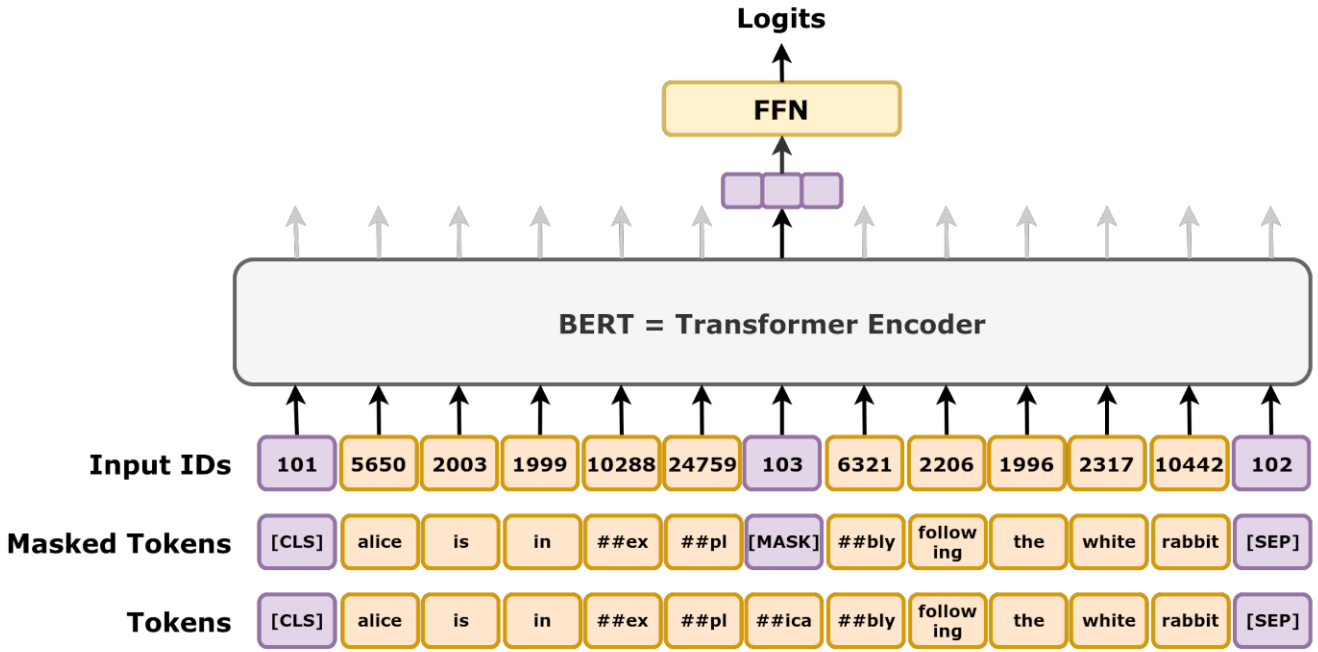


Image: https://en.wikipedia.org/wiki/BERT_(language_model)

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

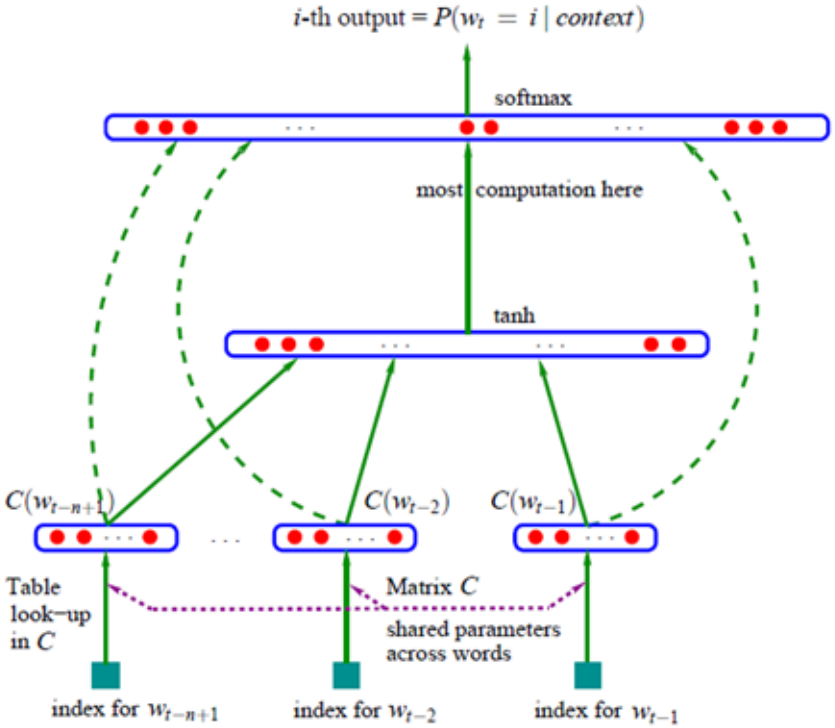


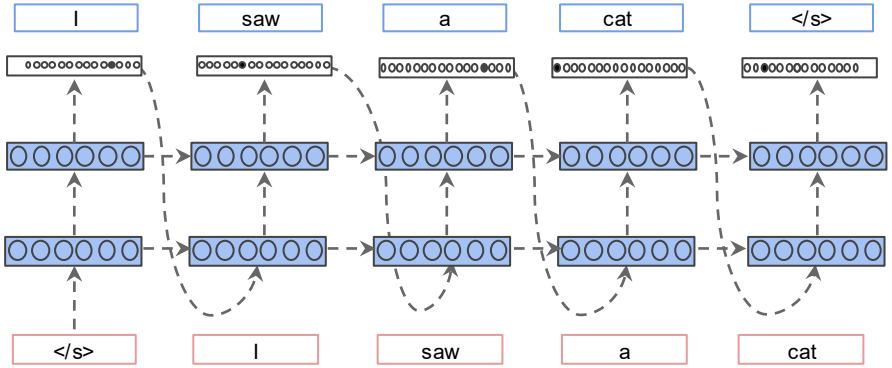
Image: (Bengio et al, 03)

Neural LMs

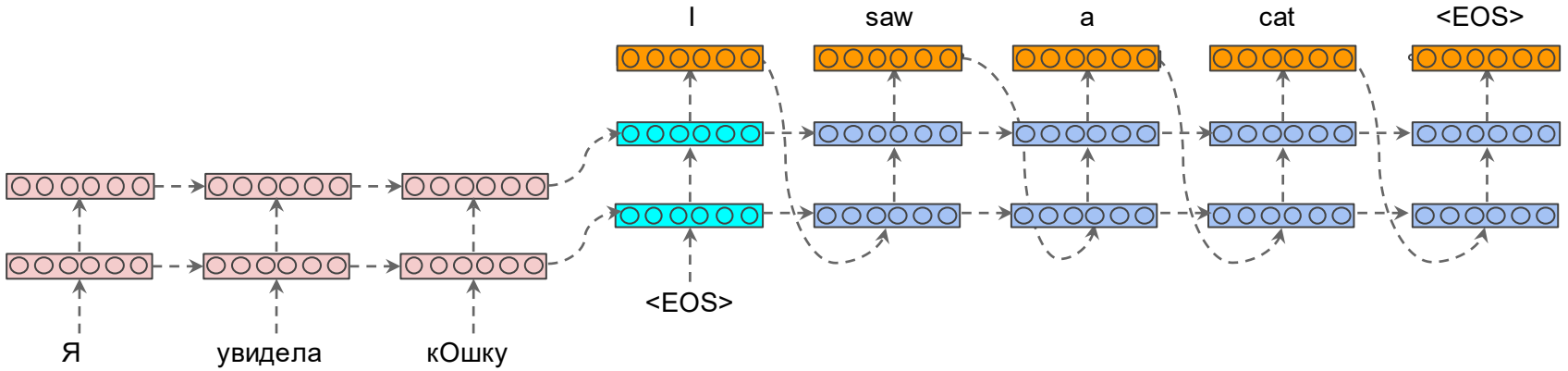
	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5	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

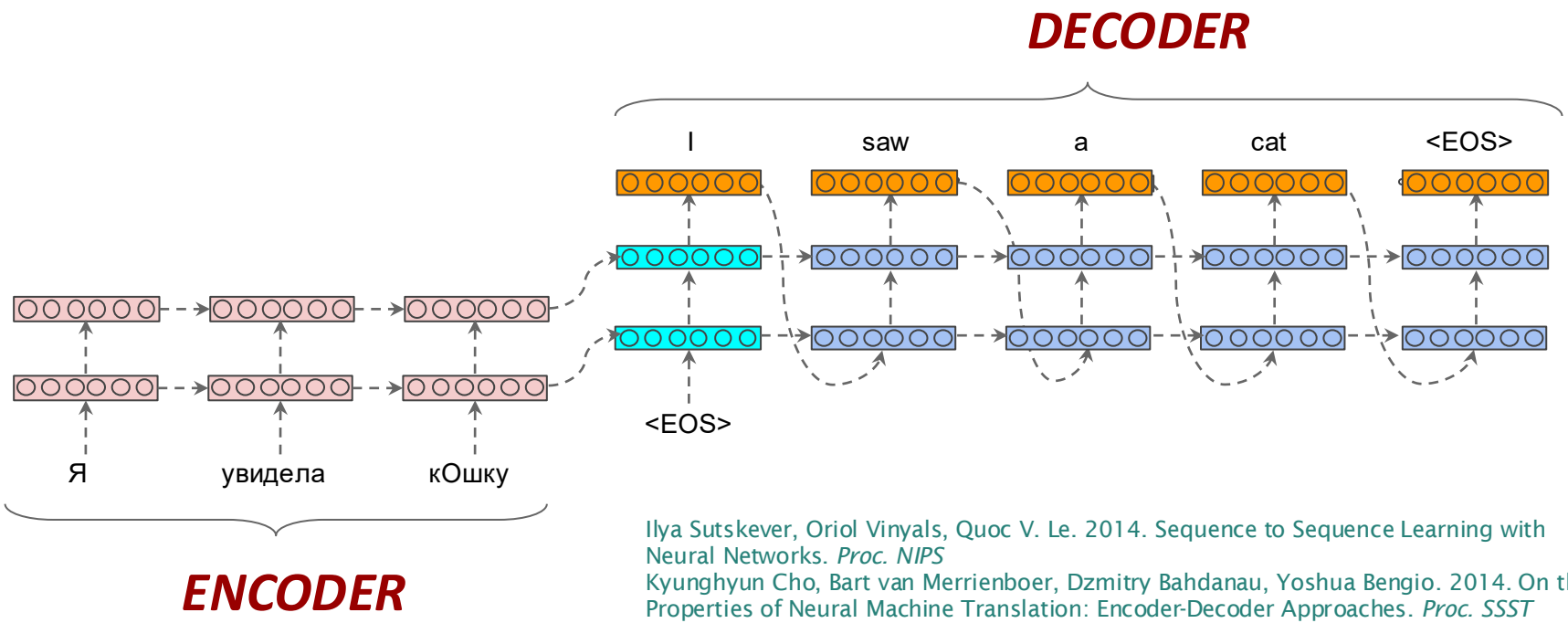


Sequence-to-Sequence Models



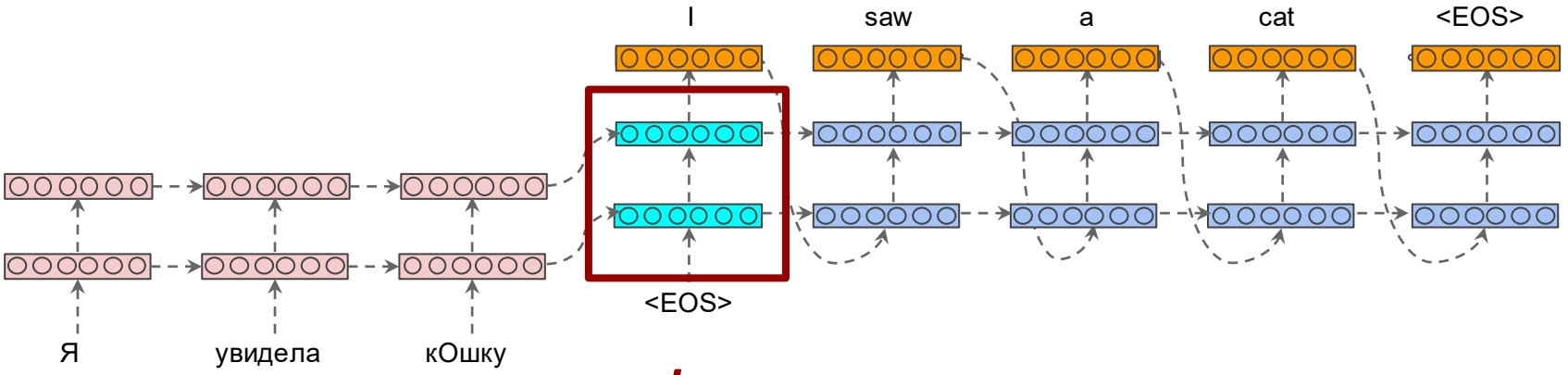
Ilya Sutskever, Oriol Vinyals, Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. Proc. NIPS

Sequence-to-Sequence Models for Neural Machine Translation



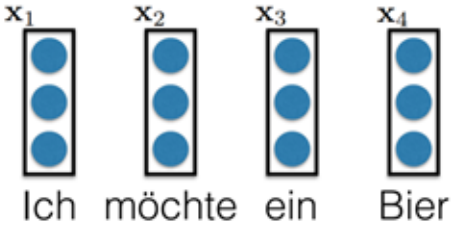
Ilya Sutskever, Oriol Vinyals, Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. *Proc. NIPS*
Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, Yoshua Bengio. 2014. On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. *Proc. SSST*

Sequence-to-Sequence Models for NMT

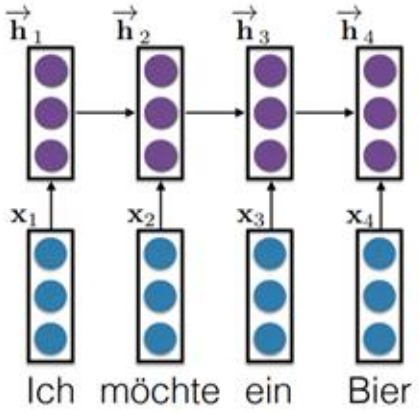


sentence representation

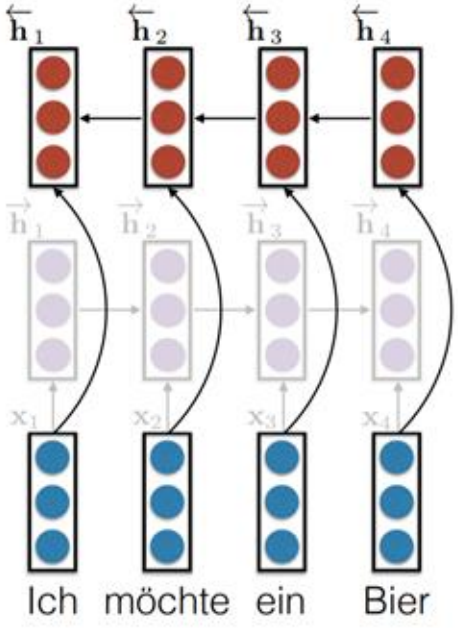
Encoder: Bidirectional RNN



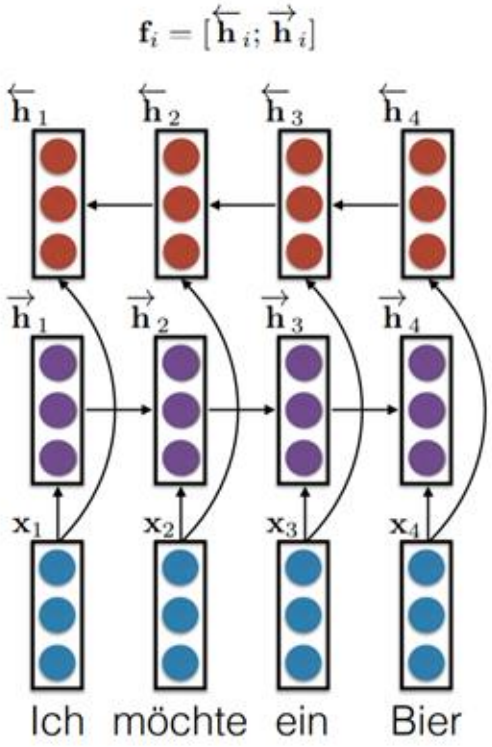
Encoder: Bidirectional RNN



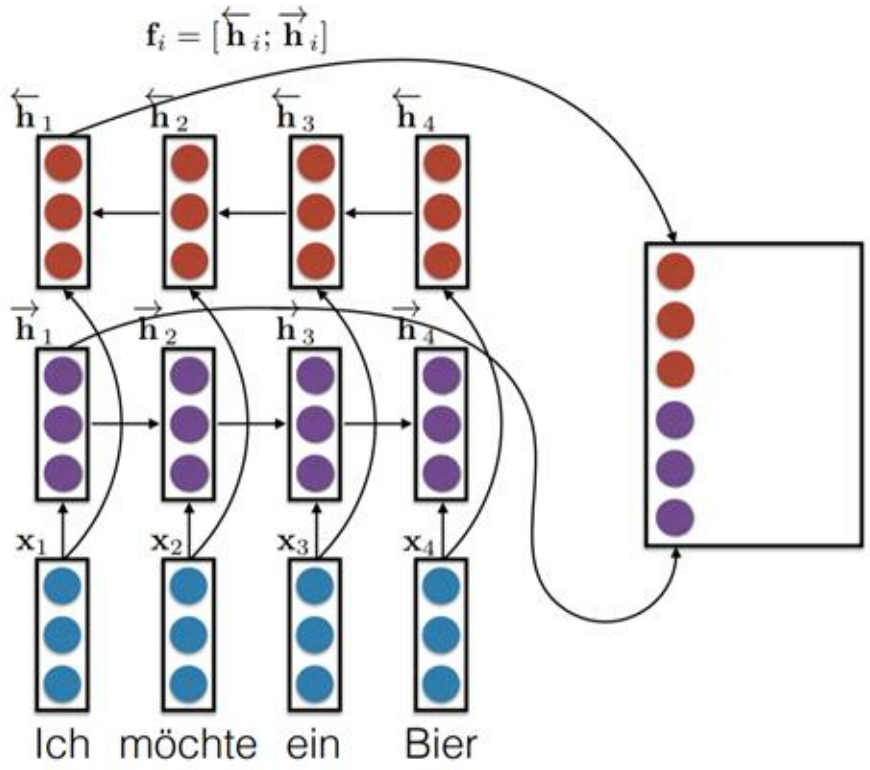
Encoder: Bidirectional RNN



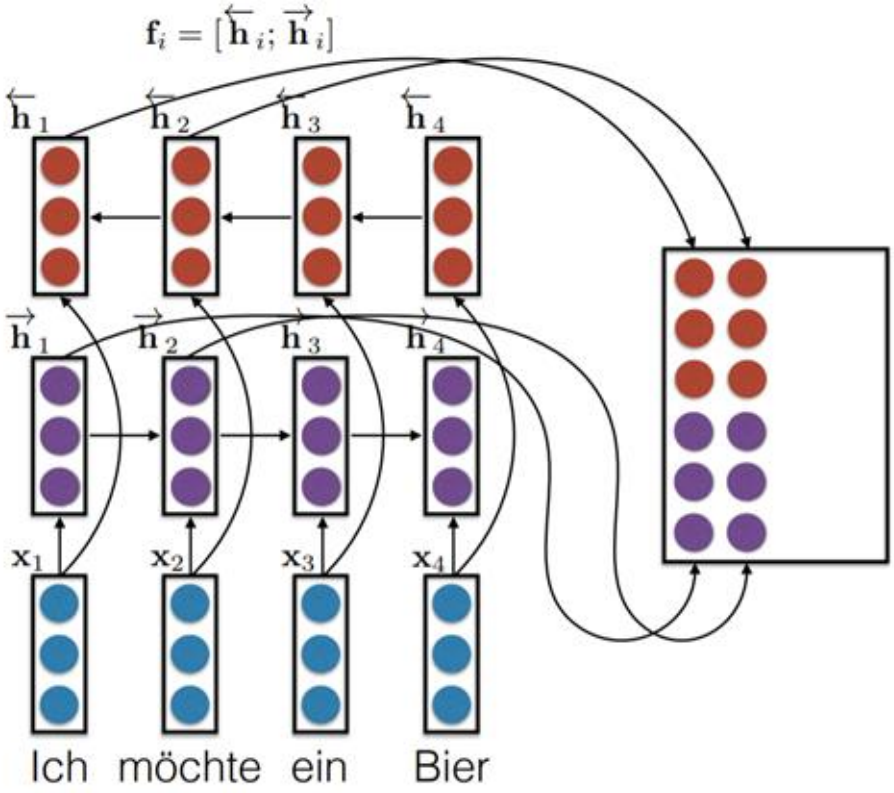
Encoder: Bidirectional RNN



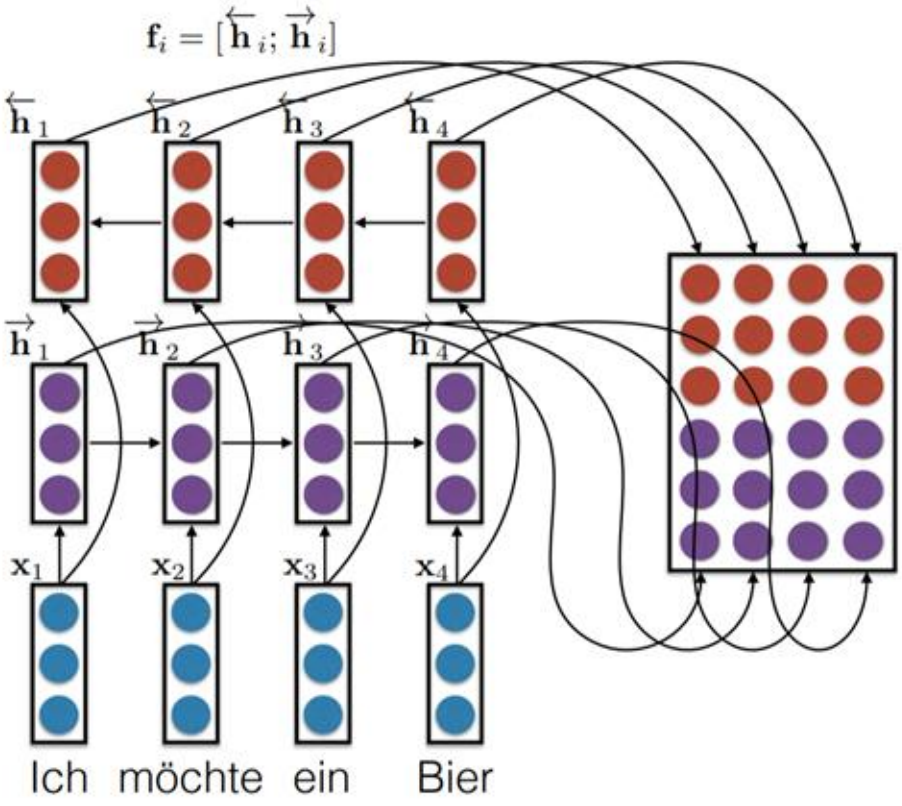
Encoder: Bidirectional RNN



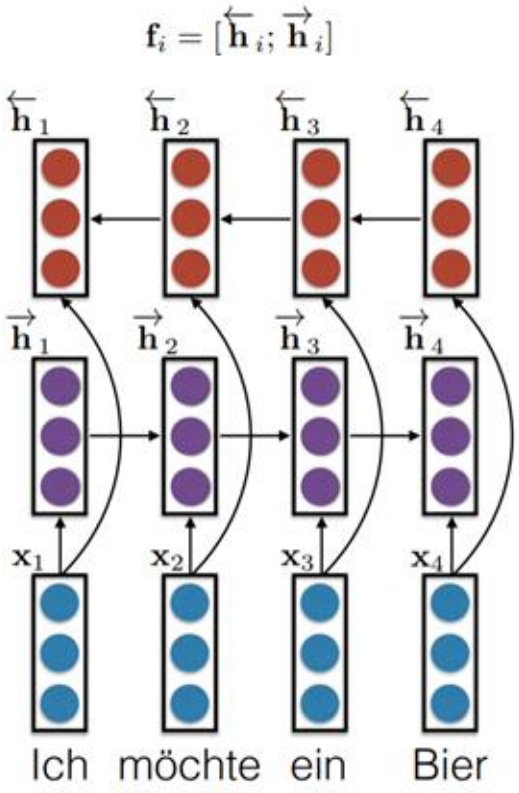
Encoder: Bidirectional RNN

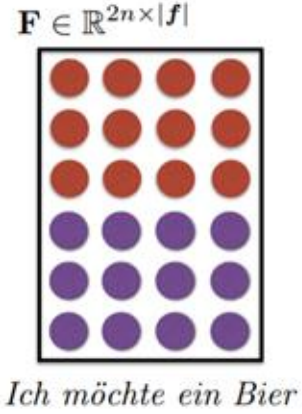


Encoder: Bidirectional RNN



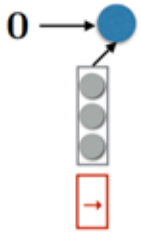
Matrix Sentence Encoding

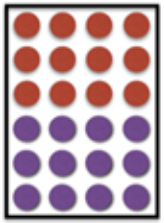




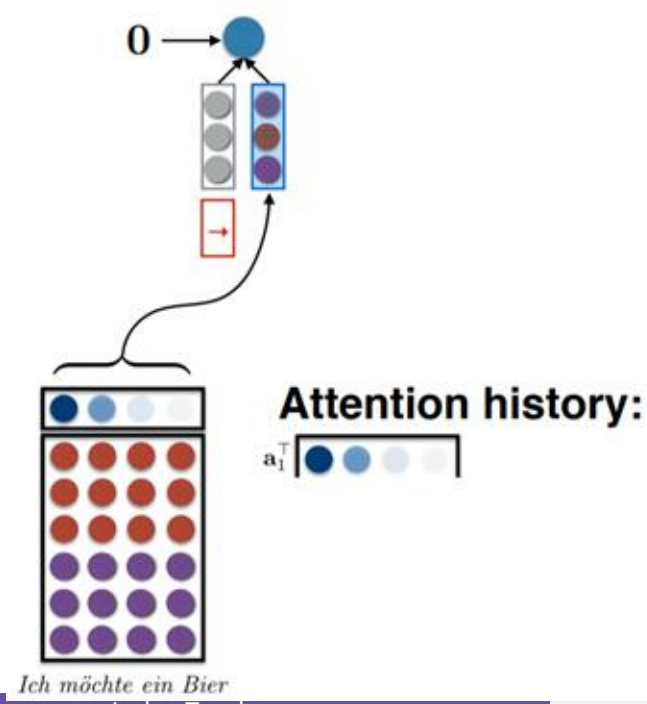
- matrix-encoded sentence

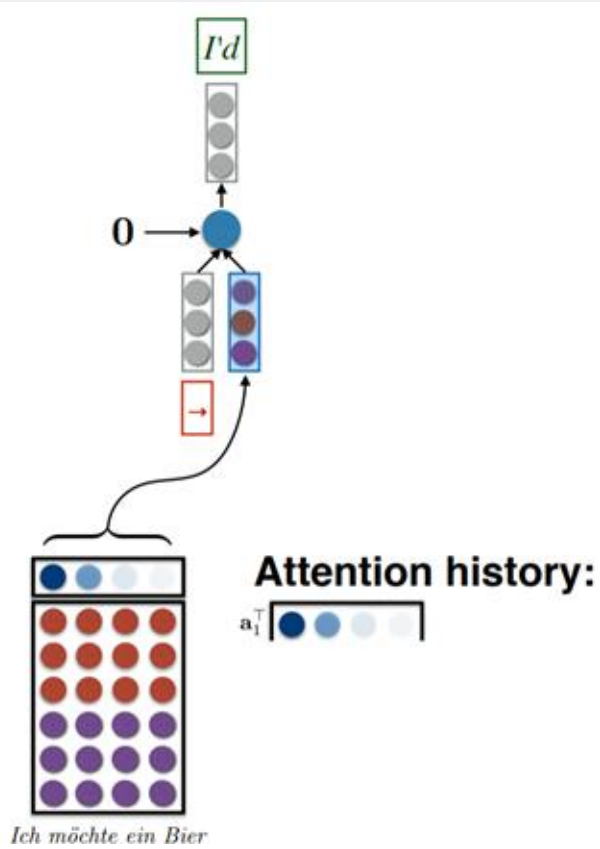
Decoder: RNN + Attention

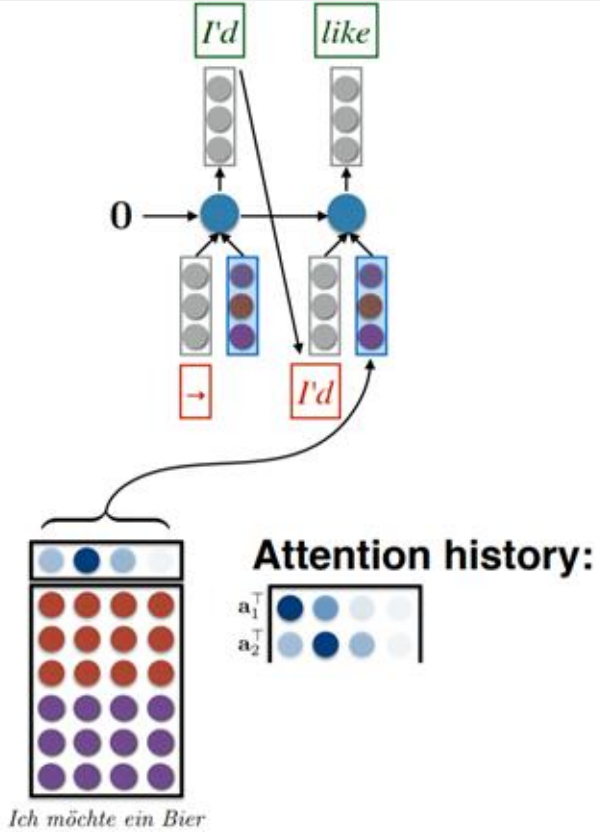


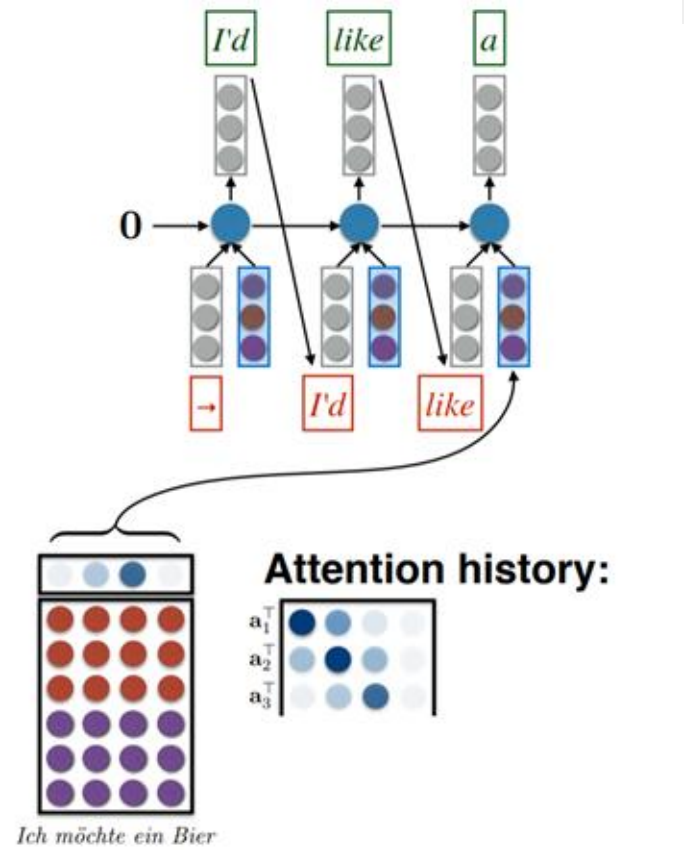


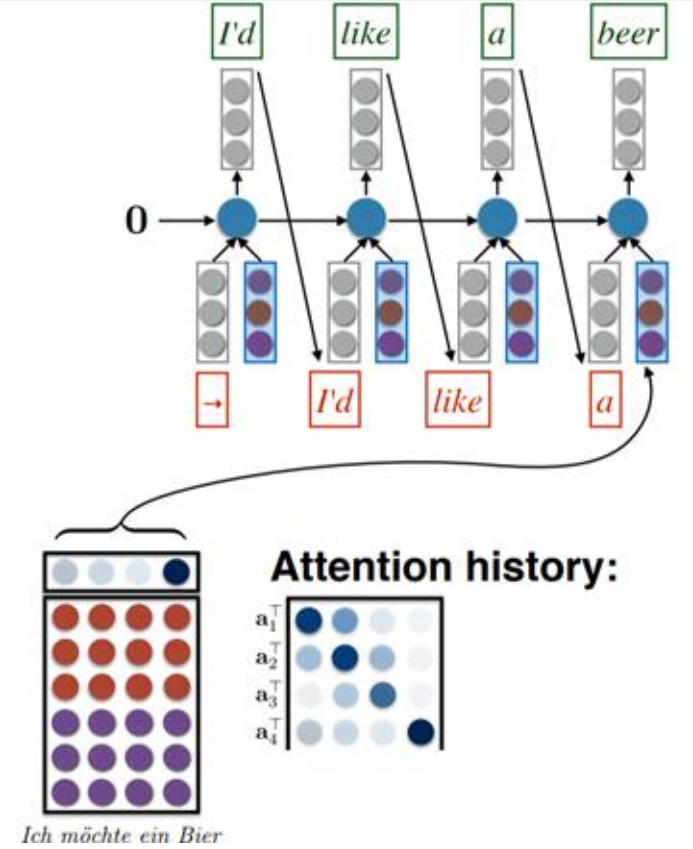
Ich möchte ein Bier

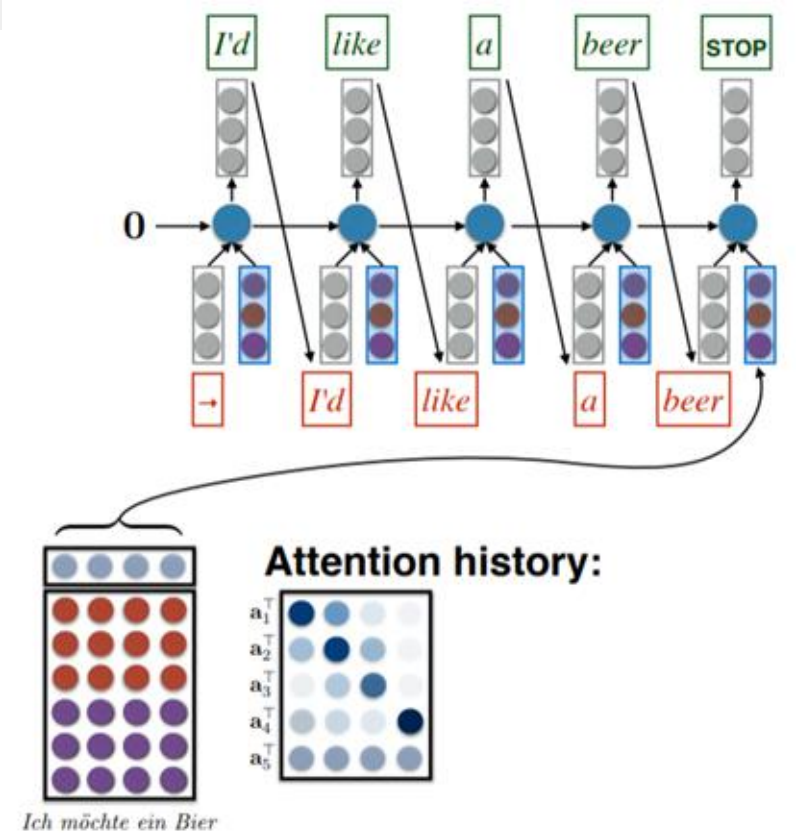

0
Attention history:
$\mathbf{a}_1^\top$
Ich möchte ein Bier


Id
0
→
Attention history:
$a_1^\top$
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:
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