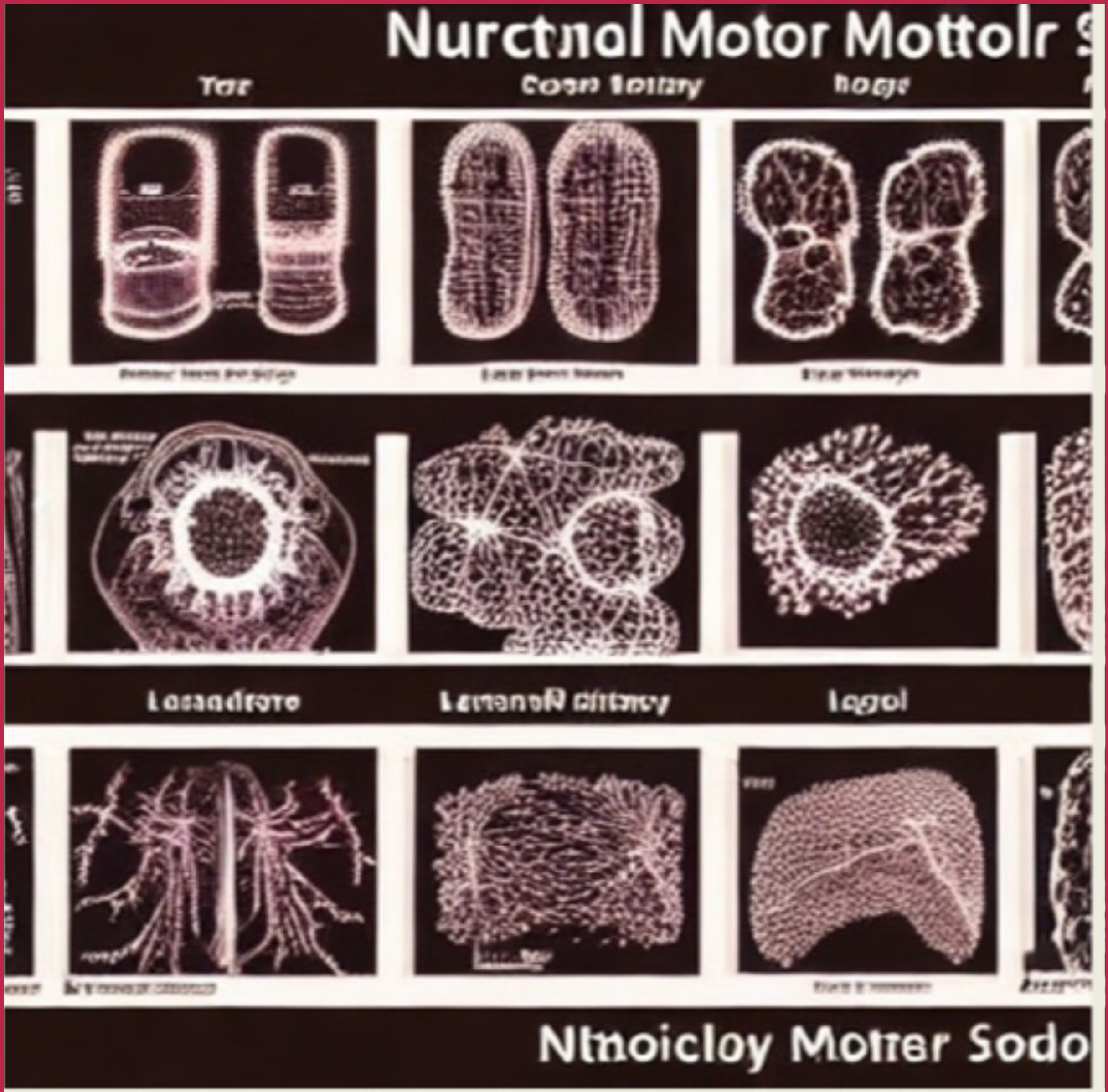


Image: AI text2img generated from lecture's title. Background: Viva Magenta (Pantone 18-1750) 2024 Color of the year!

Neural models of language

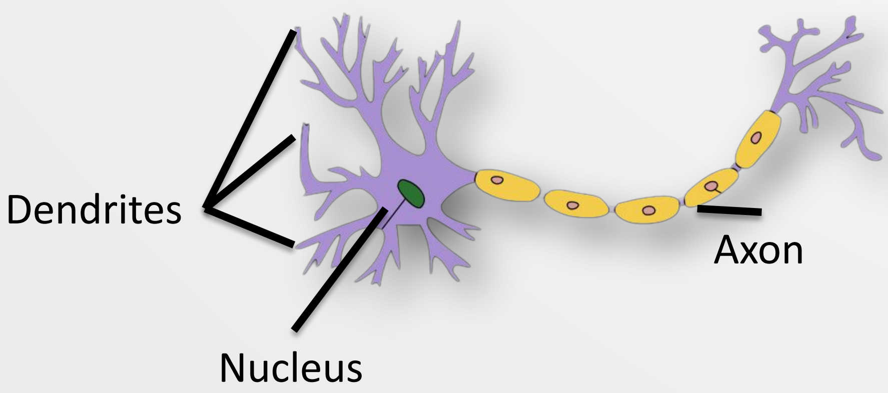
CSC401/2511 – Natural Language Computing – Spring 2026

Lecture 5

University of Toronto

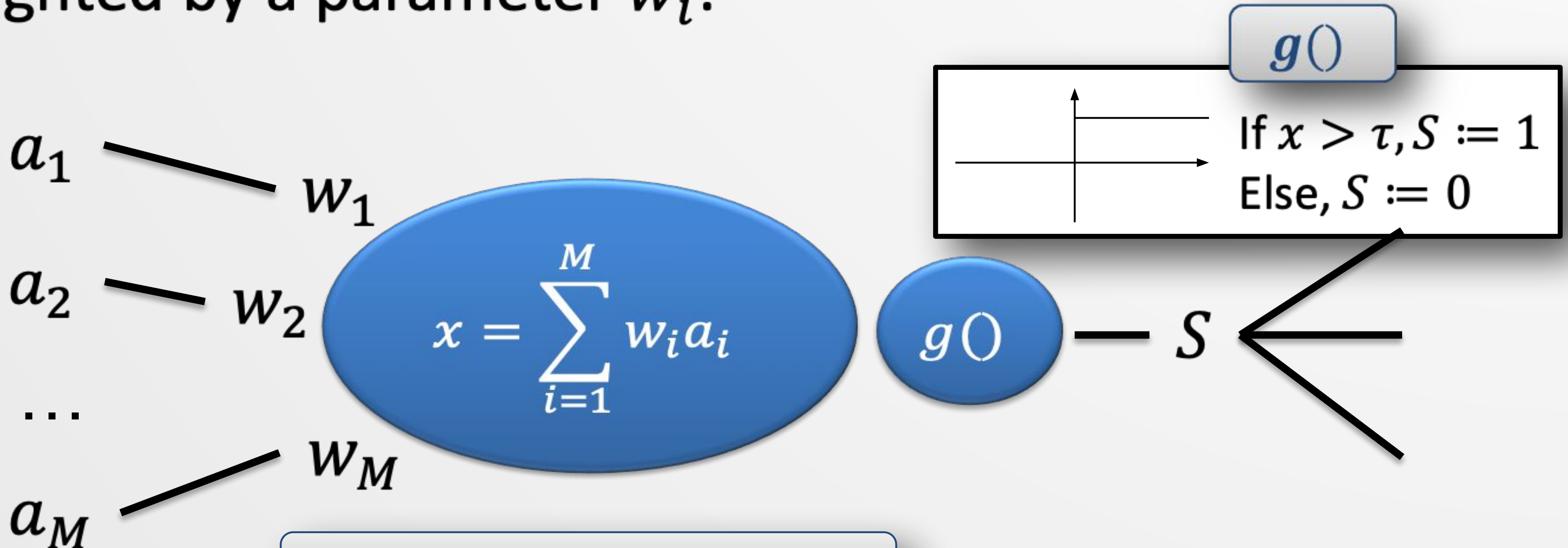
Artificial neural networks

- **Artificial neural networks (ANNs)** were loosely inspired by networks of cytoplasmic protrusions in the brain.
  - Each unit has many inputs (~**dendrites**), one output (~**axon**).
  - The **nucleus** fires (sending an electric signal along the axon) given input from other neurons.
  - 'Learning' was formerly thought to occur at the **synapses** that connect neurons, either by amplifying or attenuating signals.



Perceptron: an artificial neuron

- Each neuron calculates a **weighted sum** of its inputs and compares this to a threshold, τ. If the sum exceeds the threshold, the neuron fires.
  - Inputs a_i are activations from adjacent neurons, each weighted by a parameter w_i.



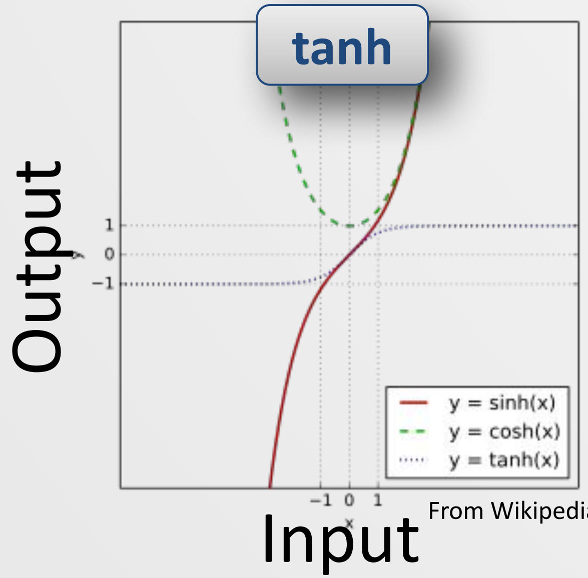
McCullogh-Pitts model

Feed-forward output

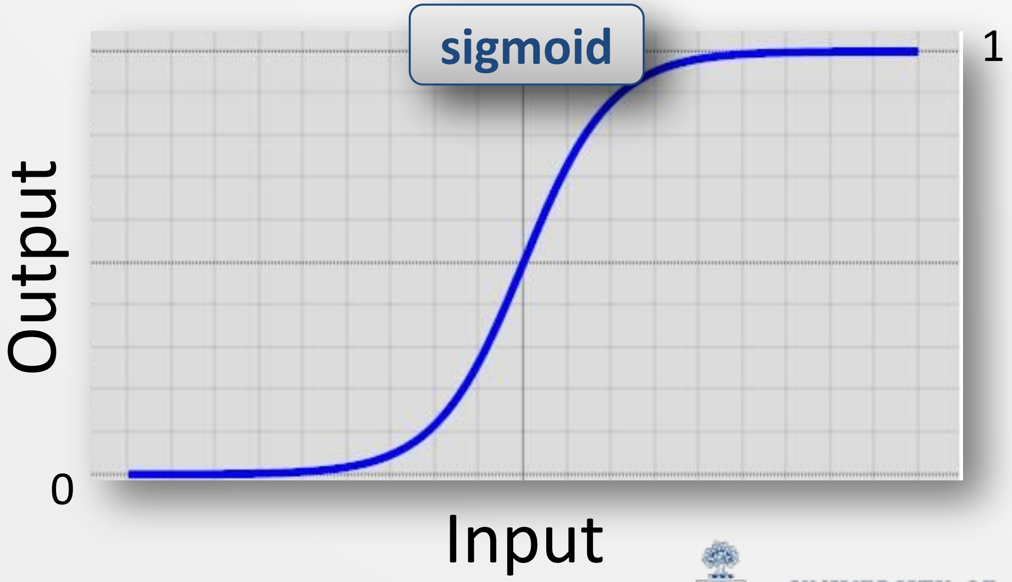
- Output is determined by an **activation function**, $g()$, which **can be non-linear (**of weighted input). Activation is empirically determined, but not learned as a parameter.
- Popular activation functions include **tanh** and the **sigmoid**:

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

  - The sigmoid's derivative is the easily computable $\sigma' = \sigma \cdot (1 - \sigma)$

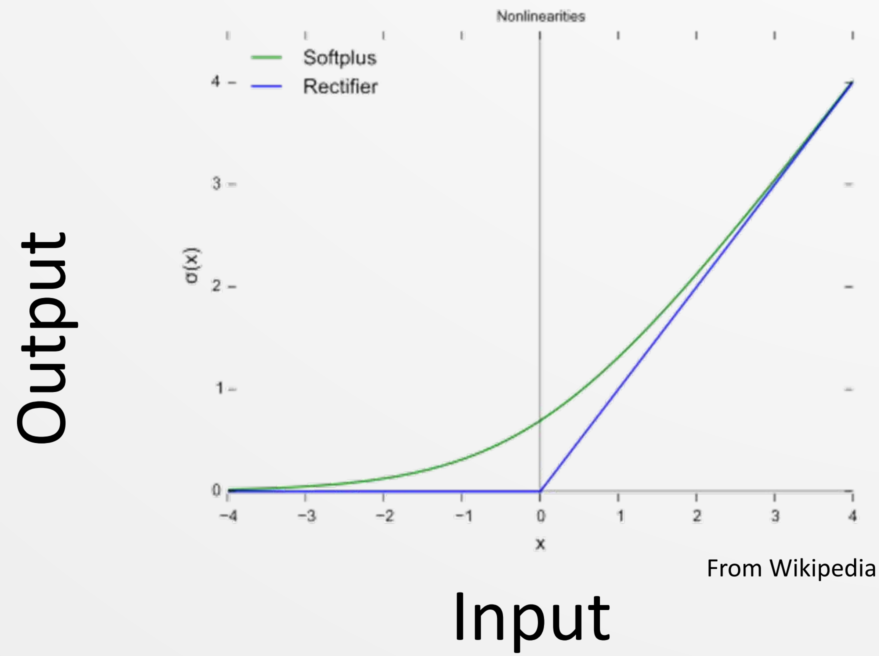


From Wikipedia



Rectified Linear Units (ReLUs)

- Since 2011, the **ReLU** $S = g(x) = \max(0, x)$ has become popular.
  - More appeals to biological plausibility, but sparse activation, and reduced likelihood of vanishing gradients are very practical reasons.
- A smooth approximation is the **softplus** $\log(1 + e^x)$, which has a simple derivative $1/(1 + e^{-x})$



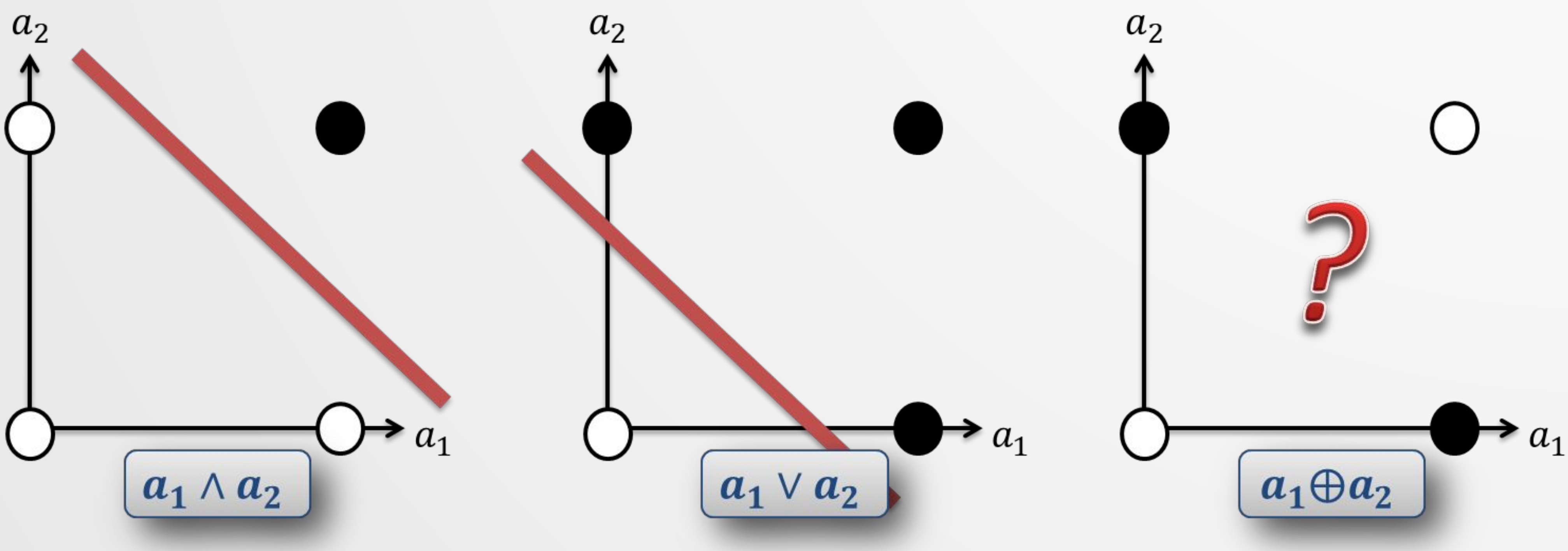
From Wikipedia

- ***Why do we care about the derivatives?***

X Glorot, A Bordes, Y Bengio (2011). Deep sparse rectifier neural networks. AISTATS.

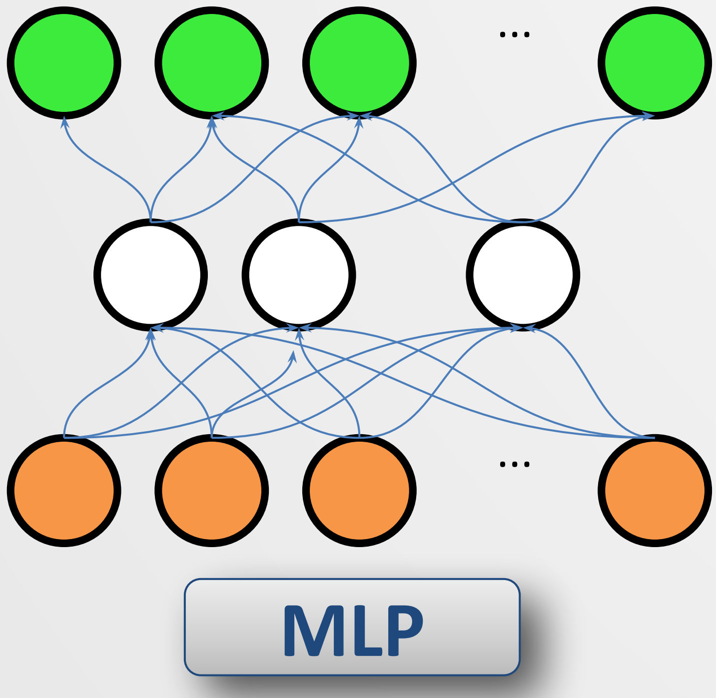
Threshold perceptra and XOR

- Some relatively simple logical functions cannot be learned by threshold perceptra (since they are not linearly separable).



Multi-layer neural networks

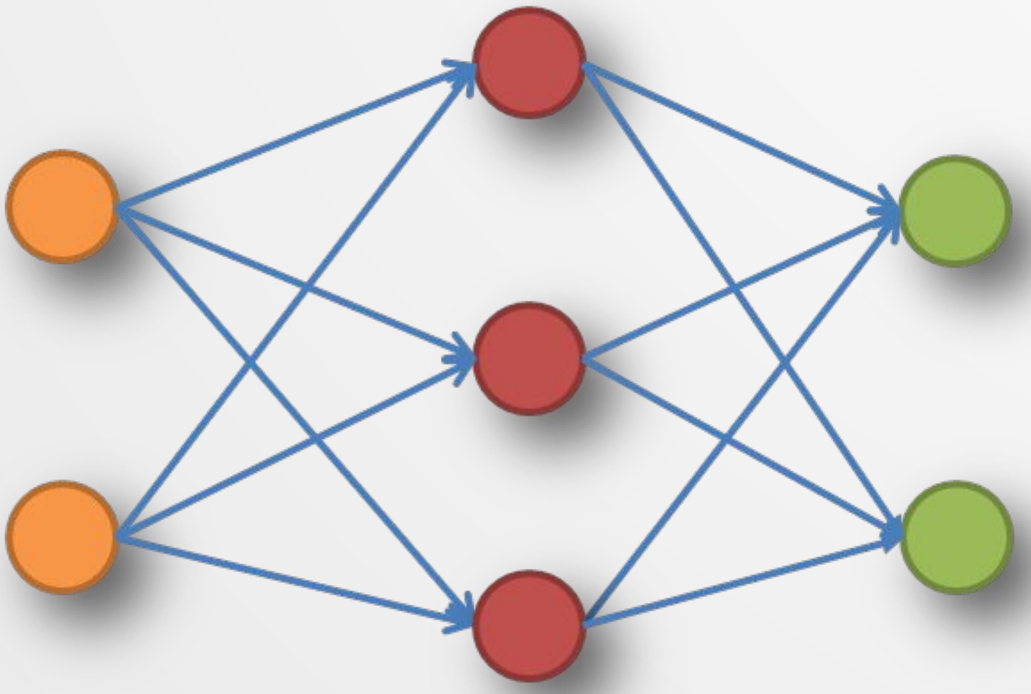
- Complex functions can be represented by layers of perceptron (**multi-layer perceptron, MLPs**).



- Inputs are passed to the **input layer.**
- **Activations** are propagated through **hidden layers** to the **output layer**.
- MLPs are quite **robust to noise**. Sometimes, we even add noise.

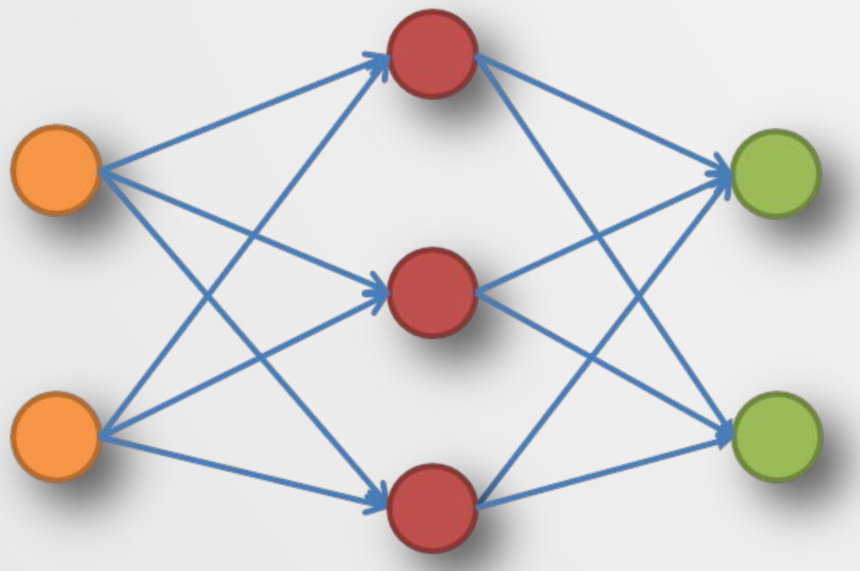
MLP Example

- Consider this simple **fully-connected** MLP below:
- How do we use it given a piece of input?



Gradient Descent

- Now that we know how NN works, how can we get one? (i.e. how to "learn" one so that it is useful?)
- Answer: Update the parameters (θ) via Gradient Descent!
- Idea: adjust the parameters **in proportion to the error**

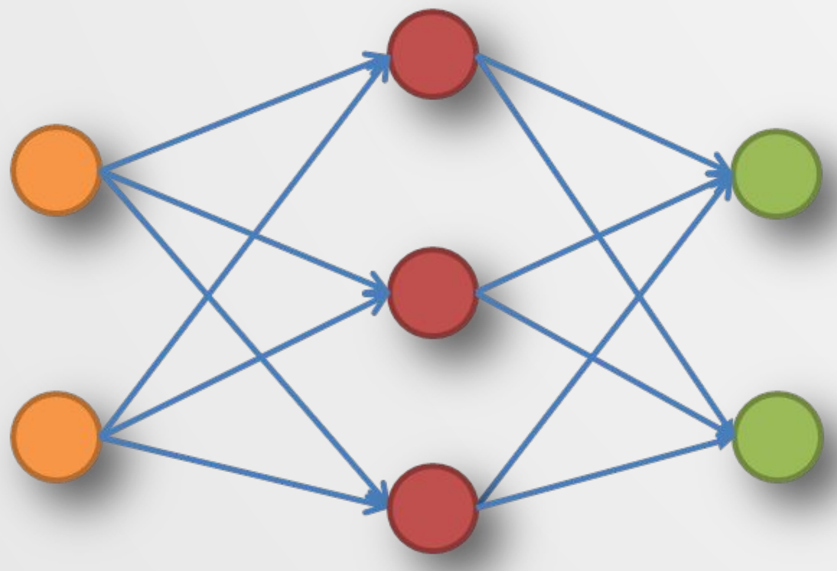


$$\theta^{(new)} = \theta^{(old)} - \alpha \nabla_{\theta} L$$

- α: Learning Rate
- ∇_{θ}L: **Gradient** of Loss

Backpropagation

- How do we compute the gradients?
- Answer: Compute the gradients ($\nabla_\theta L$) via Backpropagation!
- As it turns out, the computation is not that bad



- Hint: Easier with **backprop signals** and carefully-chosen activation function!

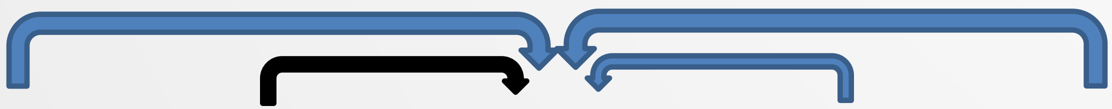
UNIVERSITY OF TORONTO

Learning word semantics

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

— J.R. Firth (1957)

$$P(w_t = lugubrious \mid w_{t-1} = feeling, w_{t-2} = been, \ldots)$$



been	feeling	lugubrious	all	day
felt	a	lugubrious	sadness	in

…

Here, we're predicting the *center* word given the context.
This is called the '**continuous bag of words**' (**CBOW**) model[1].

[1] Mikolov T, Corrado G, Chen K, *et al.* Efficient Estimation of Word Representations in Vector Space. *Proc (ICLR 2013)* 2013;:1–12.
https://code.google.com/p/word2vec/

Words

- Given a corpus with D (e.g., $= 100K$) unique words, the **one-hot approach** uniquely assigns **each word** an index in D-dimensional vectors ('one-hot' representation).

lugubrious

0	0	0	0	..	0	1	0	…	0

D

- In psychology, **word-feature representations** assign **features** to each index in a much denser vector.
  - E.g., concept-based features 'cheerful', 'emotional-tone'.

1	0.8	2.5	0.81	…	99

$d \ll D$

- Neither of these is learned.

Using word representations

Without a latent space,

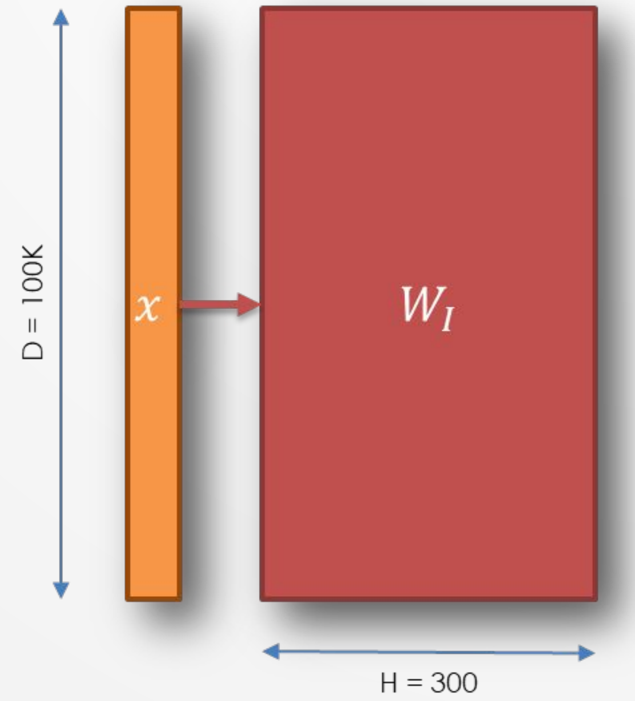
$\text{lugubrious} = [0,0,0, \ldots, 0,1,0, \ldots, 0]$, &

$\text{sad} = [0,0,0, \ldots, 0,0,1, \ldots, 0]$ so

Similarity = $\cos(x, y) = 0.0$

EMBEDDING

$$v_w = x^\intercal W_I$$



In latent space,

$\text{lugubrious} = [0.8, 0.69, 0.4, \ldots, 0.05]_H$, &

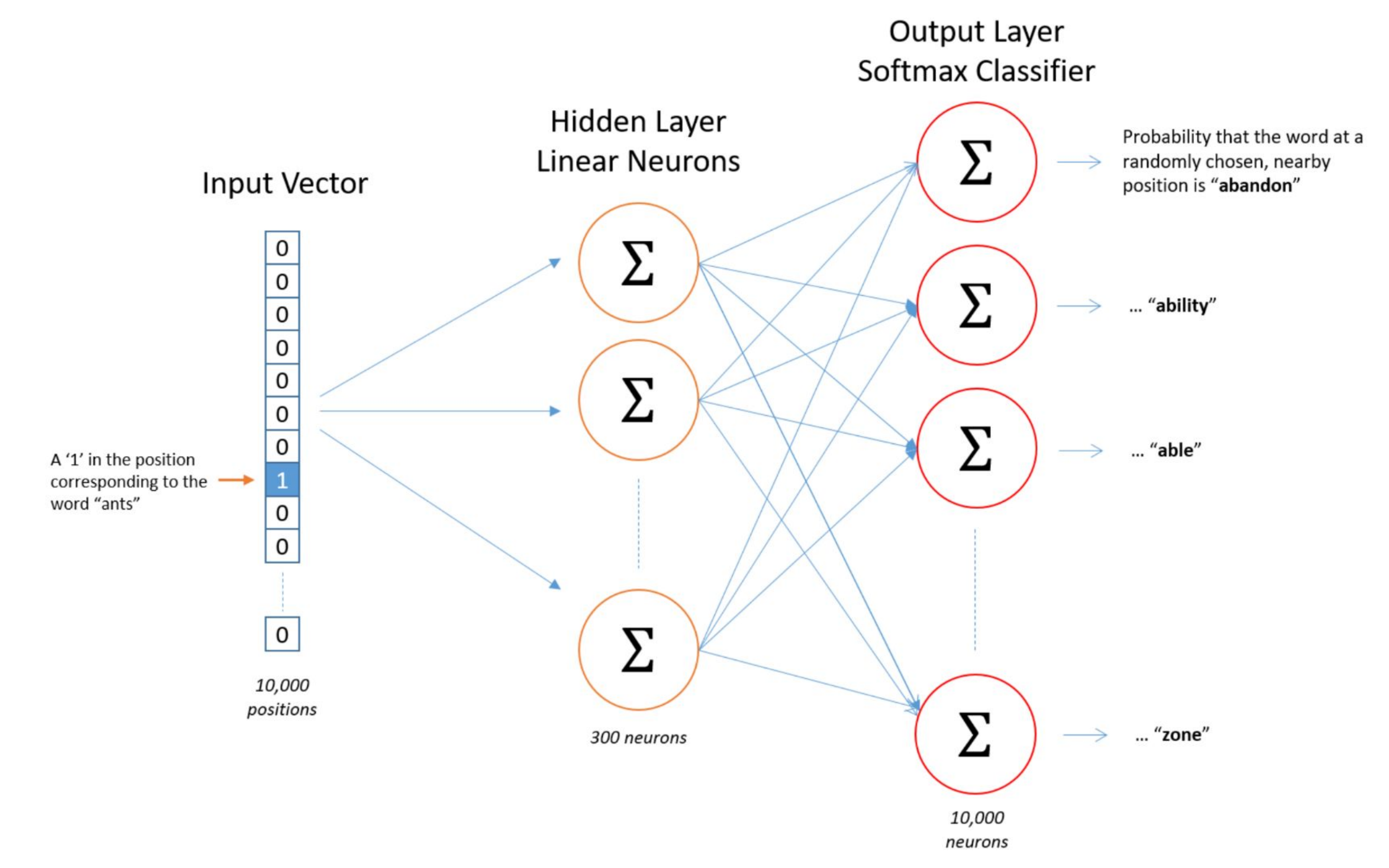
$\text{sad} = [0.9, 0.7, 0.43, \ldots, 0.05]_H$ so

Similarity = $\cos(x, y) = 0.9$

Reminder:

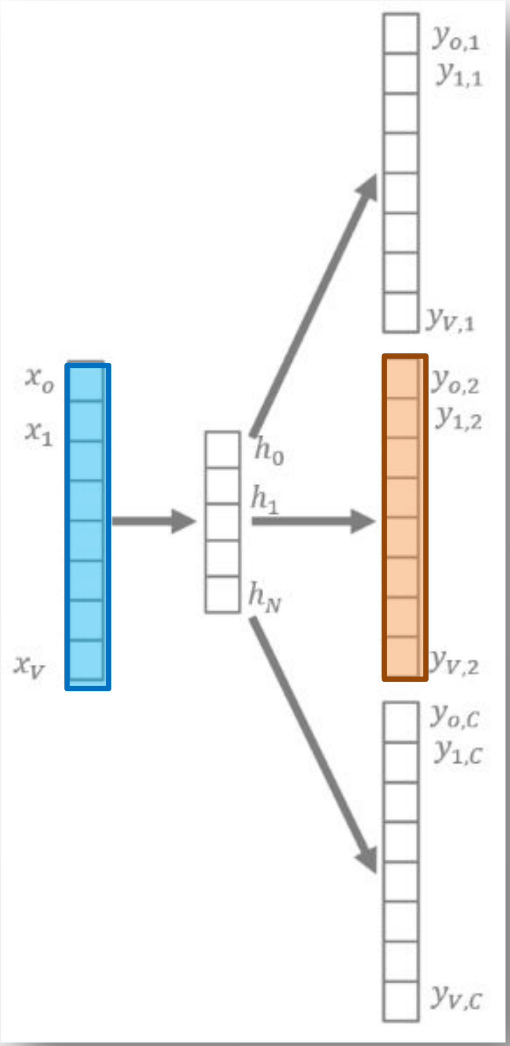
$$\cos(u, v) = \frac{u \cdot v}{||u|| \times ||v||}$$

word2vec training regimen



Skip-grams with negative sampling

- Most word types do not appear together within a small window. The default process does not know this.
  - Also, not all that efficient – would be nice not to update $H \times D$ weights
  - **Contrastive learning:** push away from negative examples as well as towards positives.
- For the observed (true) pair (*lugubrious*, *sadness*), only the output neuron for *sadness* should be 1, and all $D - 1$ others should be 0.



- Mathematical Intuition:
  - $P(w_o|w_c) = \dfrac{\exp(v_o^T V_c)}{\sum_{w=1}^{D} \exp(v_w^T V_c)}$

Computationally infeasible

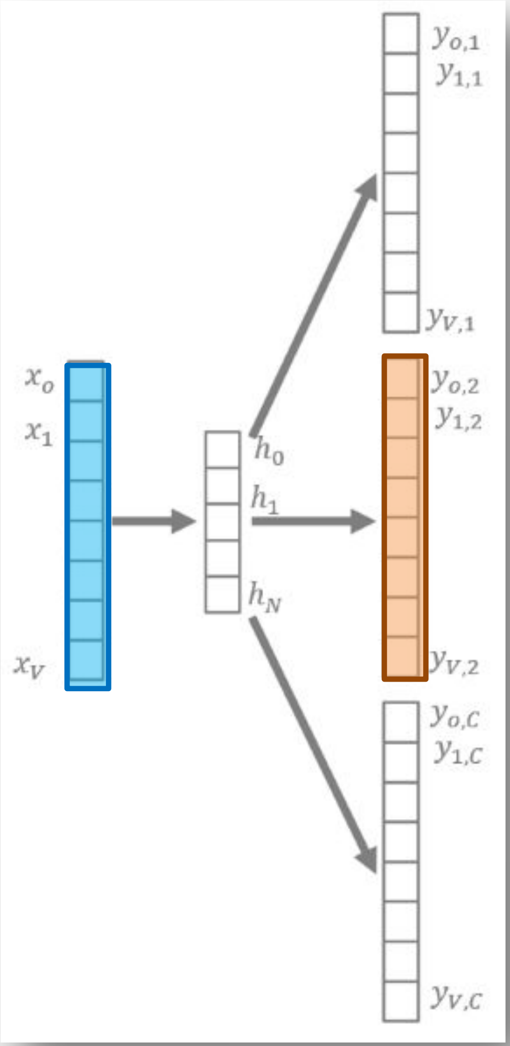
Skip-grams with negative sampling

- We want to **maximize** the association of ***observed*** (positive) contexts:

  lugubrious sad
  lugubrious feeling
  lugubrious tired

- We want to **minimize** the association of '***hallucinated***' contexts:

  lugubrious happy
  lugubrious roof
  lugubrious truth



UNIVERSITY OF TORONTO

Skip-grams with negative sampling

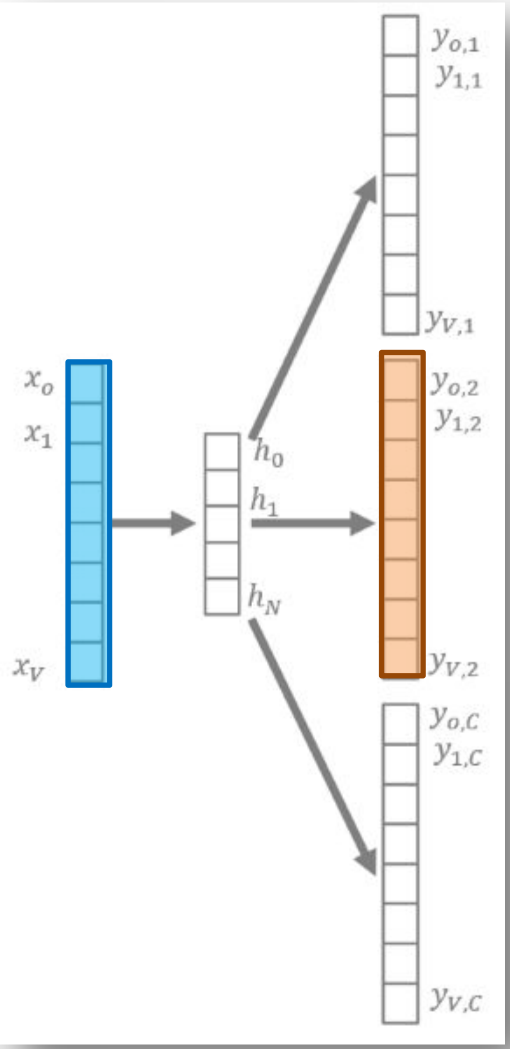
- Choose a small number k of 'negative' words, then update the weights only for all the positive and the k negative words.
  - $5 \leq k \leq 20$ can work in practice for fewer data.
  - For $D = 100K$, we only update 0.006% of the weights in the output layer.

$$J(\theta) = \log \sigma(v_o^T v_c) + \sum_{i=1}^{k} \mathbb{E}_{i \sim \underline{P(w)}}[\log \sigma(-v_i^T v_c)]$$

Unigram dist.

- Mimno and Thompson (2017) choose the top k words by modified unigram probability:

$$\underline{P^*(w_{t+1})} = \frac{C(w_{t+1})^{\frac{3}{4}}}{\sum_w C(w)^{\frac{3}{4}}}$$



Mimno, D., & Thompson, L. (2017). The strange geometry of skip-gram with negative sampling. *EMNLP 2017*. [link]

RECURRENT NEURAL NETWORKS

Statistical language models

- Probability is conditioned on (window of) n previous words*
- A necessary (but incorrect) Markov assumption: each observation only factors through **a short linear history** of length *L*.

$$P(w_n|w_{1:(n-1)}) \approx P(w_n|w_{(n-L+1):(n-1)})$$

- Probabilities are estimated by computing unigrams and bigrams

$$P(s) = \prod_{i=1}^{t} P(w_i|w_{i-1})$$

bigram

$$P(s) = \prod_{i=2}^{t} P(w_i|w_{i-2}w_{i-1})$$

trigram

*From Lecture 2

Statistical language models

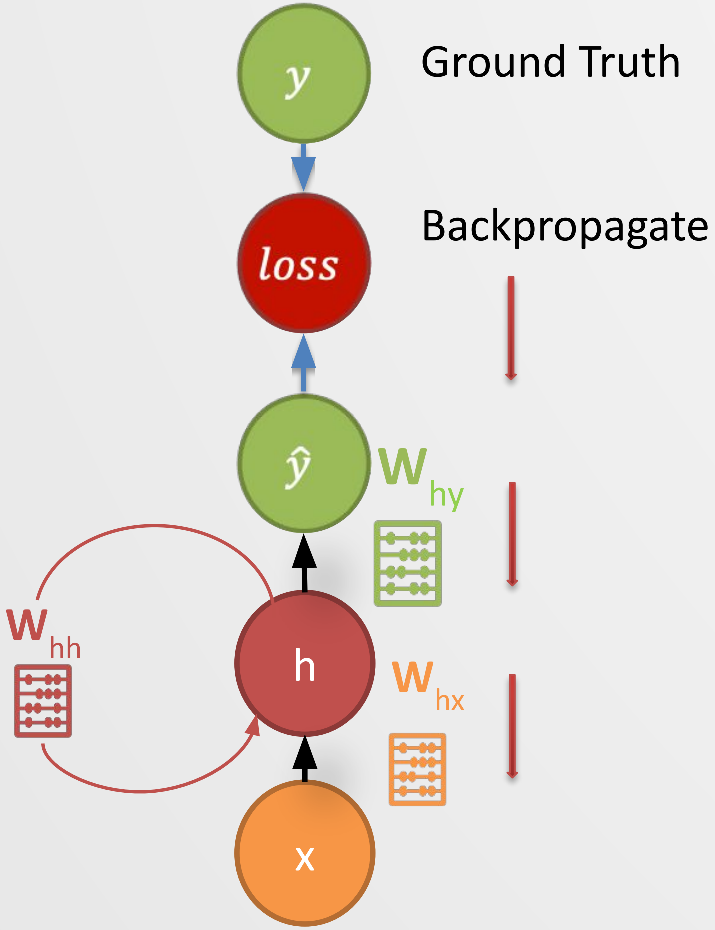
- Using higher n-gram counts (with smoothing) improves performance*

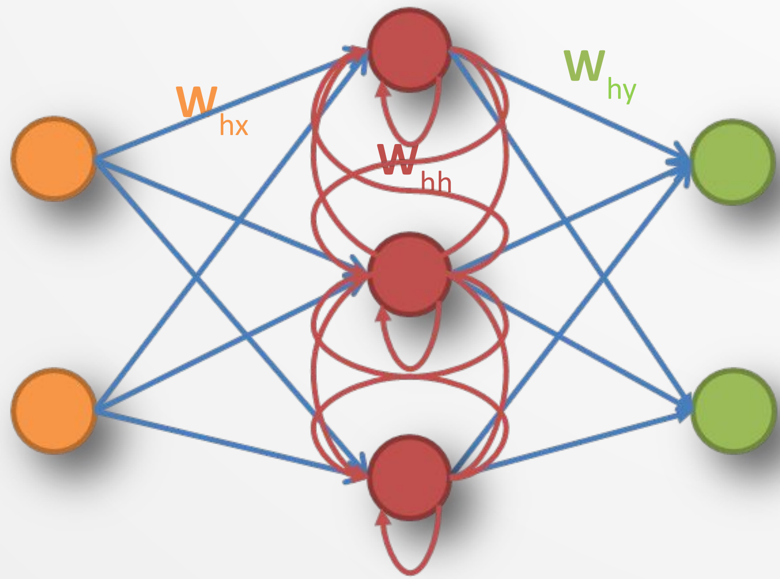
- *RNN intuition:*
  - Use as much history as we need to use
  - Use the **same set of weight** parameters for each word (or across all time steps) to keep the size of the network down
  - Memory requirement now scales with number of words

*From Lecture 2

Recurrent neural networks (RNNs)

- An RNN has **feedback** connections in its structure so that it ‘*remembers*’ previous states, when reading a sequence.





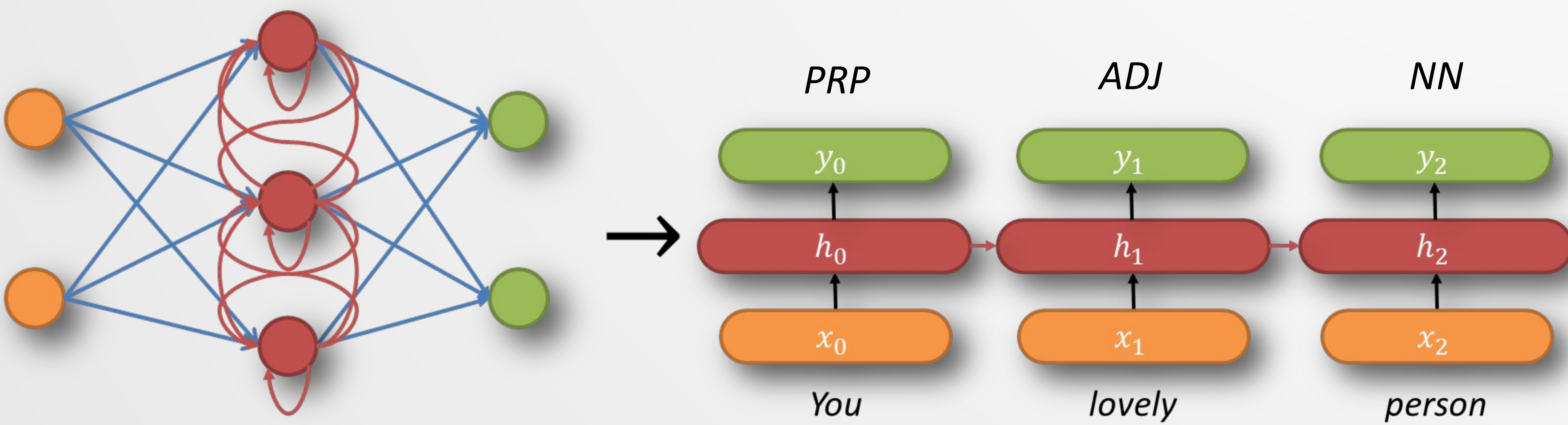
Elman network feed hidden units back

Jordan network (not shown) feed output units back

UNIVERSITY OF TORONTO

RNNs: Unrolling the h_i

- Copies of the same network can be applied (i.e., **unrolled**) at each point in a time series.
- Now we can use an approximation: backpropagation through time (BPTT).

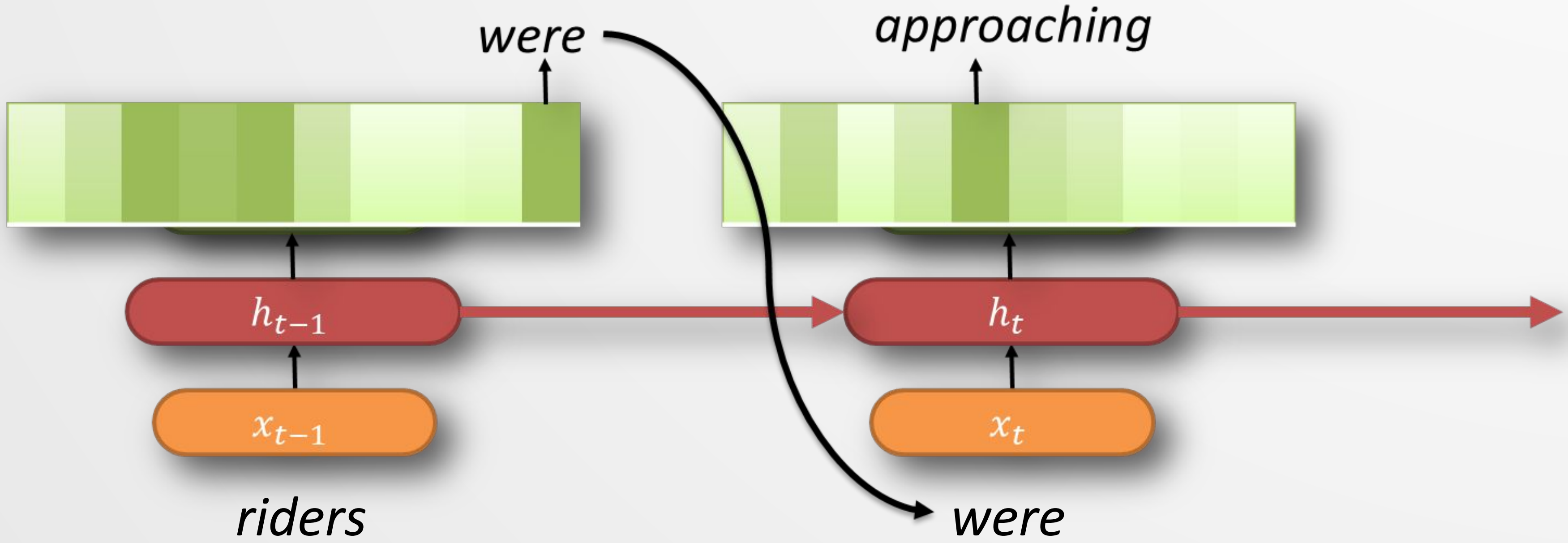


$$h_t = g(W_I[\boldsymbol{h_{t-1}}; \boldsymbol{x}] + c)$$
$$\boldsymbol{y_t} = W_O\boldsymbol{h_t} + b$$

UNIVERSITY OF TORONTO

Sampling from a RNN LM

- If $|h_i| < |V|$, we've already reduced the number of parameters relative to trigrams.
- Good news: NN encodings tend to be very compact.

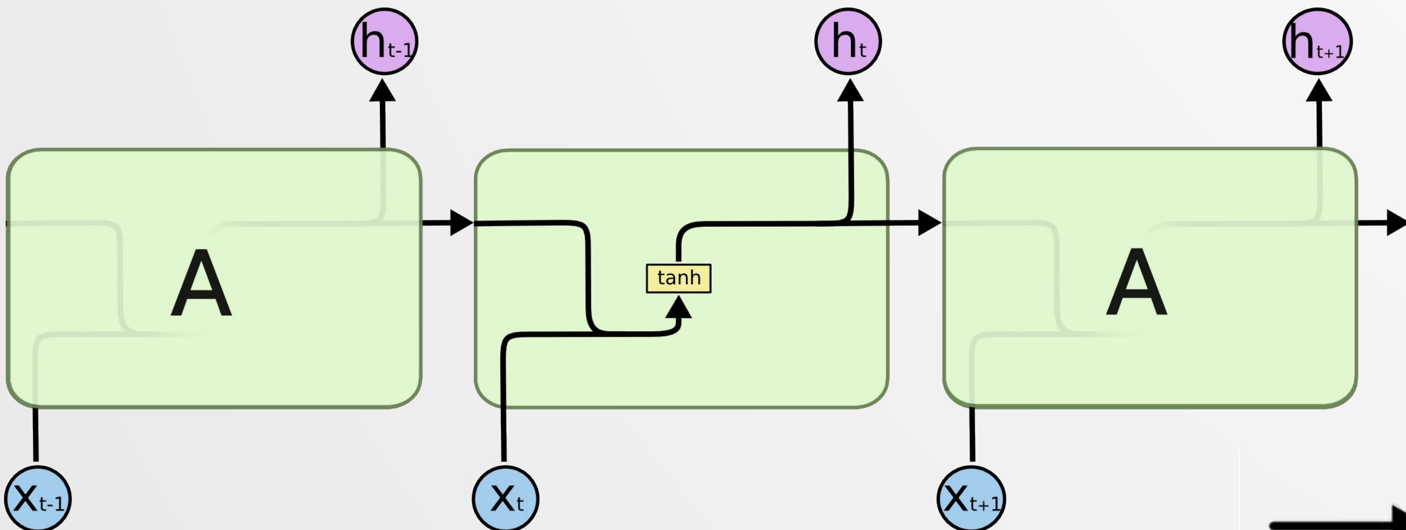


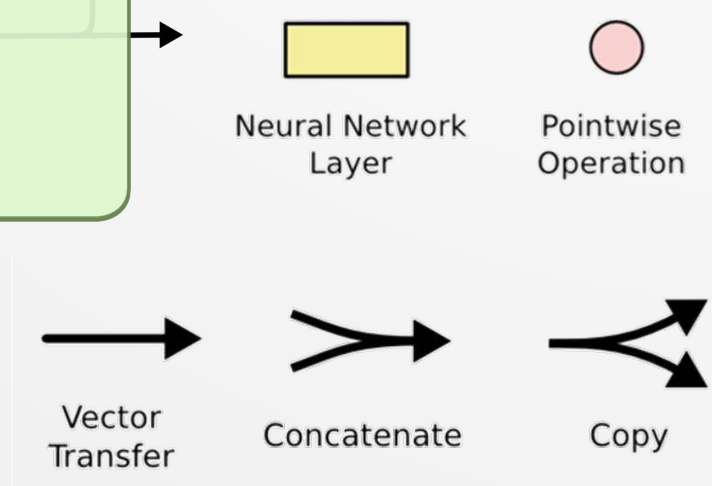
$$h_t = g([W_{hh}\boldsymbol{h_{t-1}}; W_{hx}\boldsymbol{x_t}] + c)$$
$$\hat{y}_t = softmax\,(W_{hy}\boldsymbol{h_t} + b)$$

Karpathy (2015),
The Unreasonable Effectiveness of Recurrent Neural Networks

RNNs and retrograde amnesia

- Bad news: gradients don't multiply out well over long distances (**gradient decay).**
- Can we spend some parameters to store extra information?





Here, 'A' represents identical recurrent cell blocks.

Imagery and sequence from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

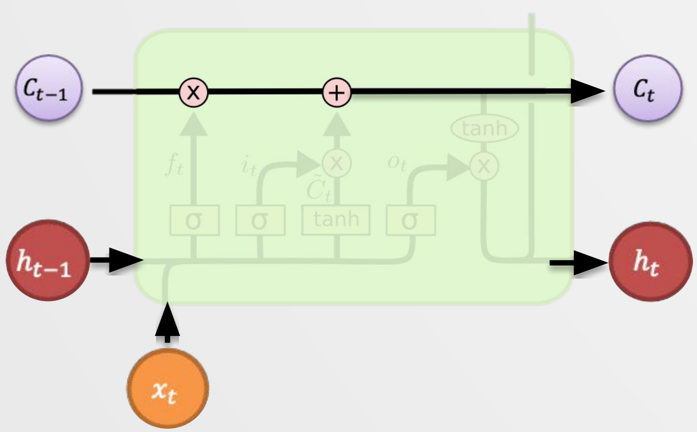
RNNs and retrograde amnesia

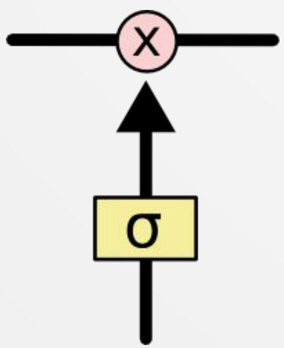
- **Catastrophic forgetting** is common.
  - E.g., the **relevant** context in "*The teacher taught transformers terribly telling tiring, tortuous theories …*" has likely been **overwritten** by the time h_{13} is produced.

Bengio Y, Simard P, Frasconi P. (1994) Learning Long-Term Dependencies with Gradient Descent is Difficult. IEEE Trans. Neural Networks.;**5**:157–66. doi:10.1109/72.279181

Long short-term memory (LSTM)

- Within each *recurrent unit or cell*:
  - Self-looping recurrence for ***cell state*** *using vector C*
  - Information flow regulating structures called ***gates***



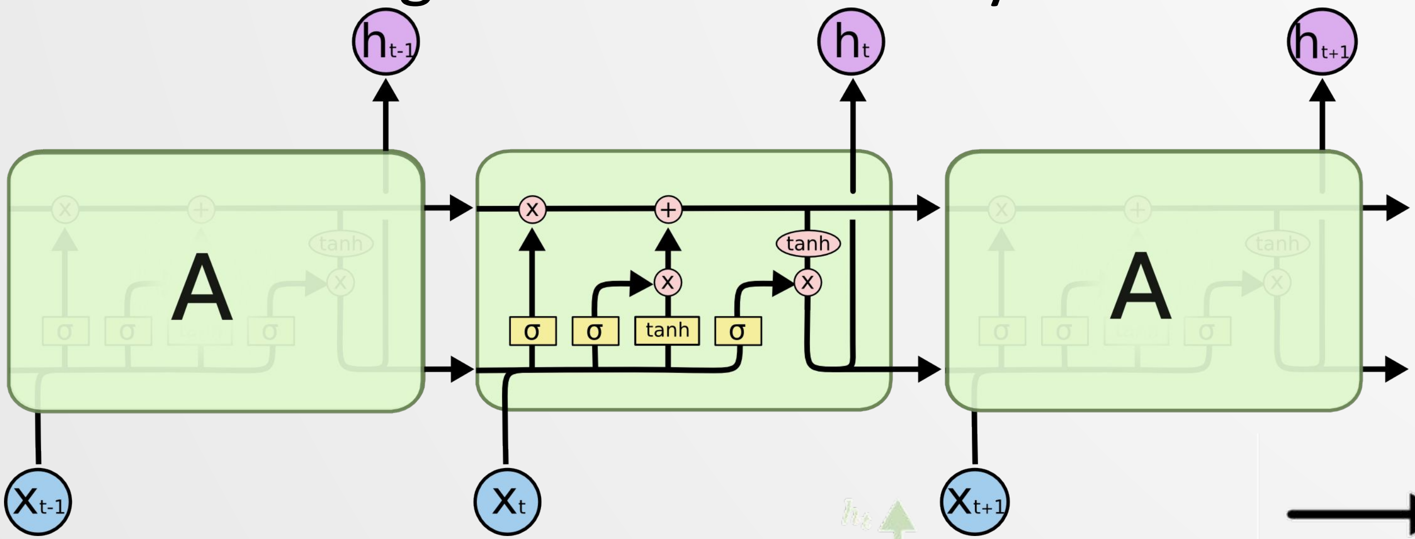


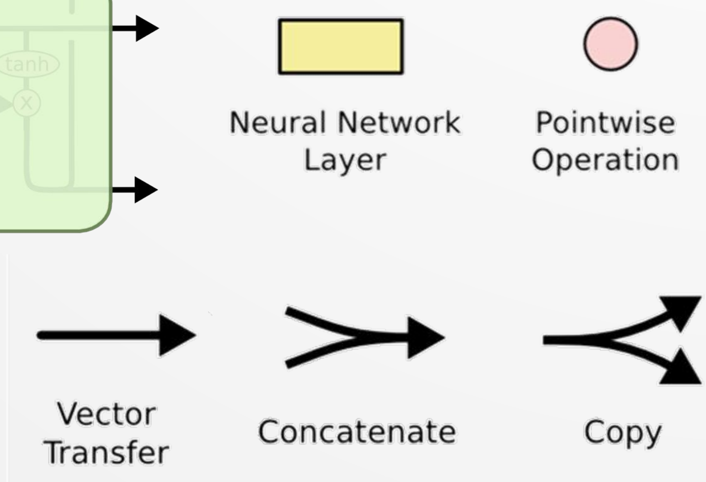
Pointwise multiplication

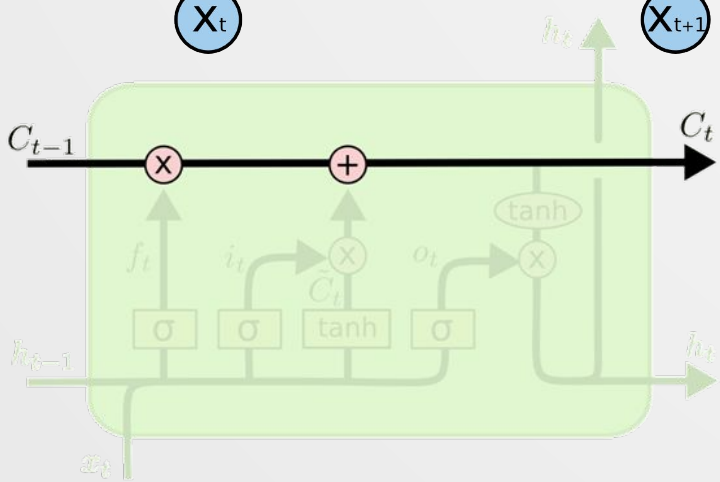
Sigmoid neural net layer

LSTM – core ideas

- In each **cell** (i.e. recurrent unit) in an LSTM, there are four interacting neural network layers.



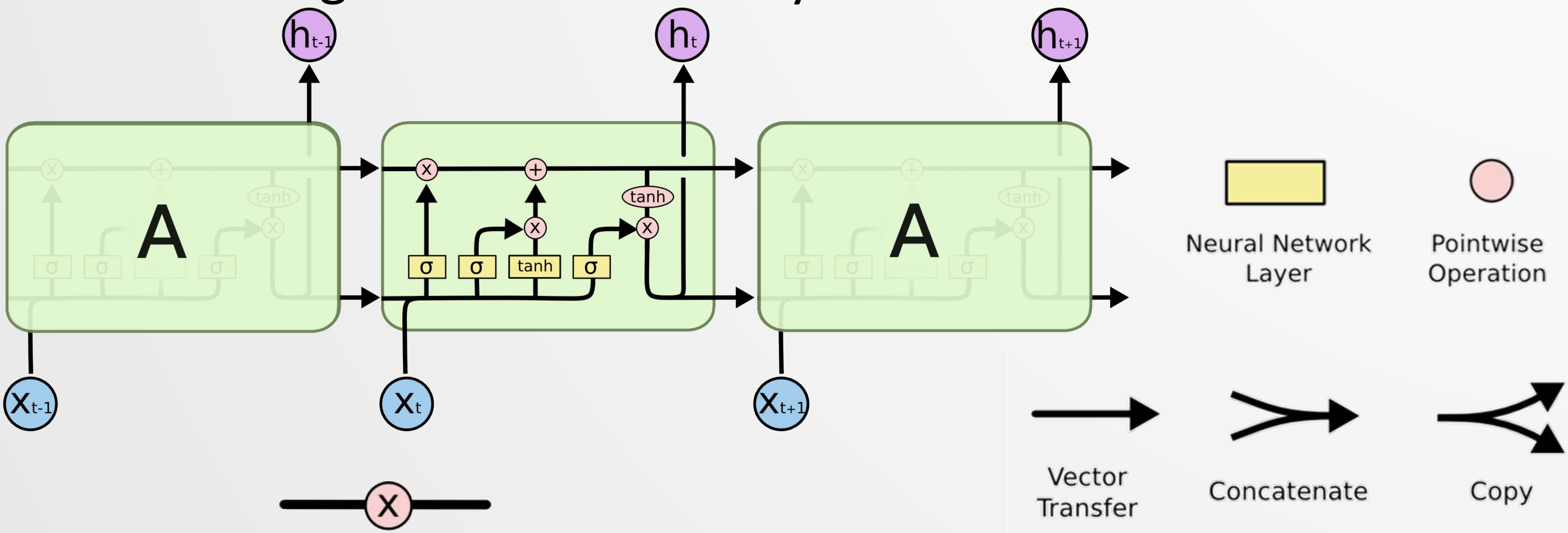


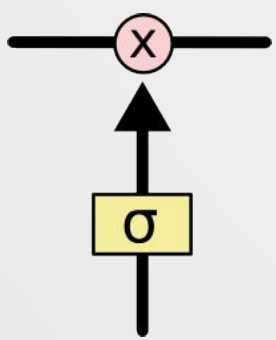


The **cell state** is a special vector stream that runs through the entire chain and **stores long-term information.**

LSTM – core ideas

- In each **cell** (i.e. recurrent unit) in an LSTM, there are four interacting neural network layers.

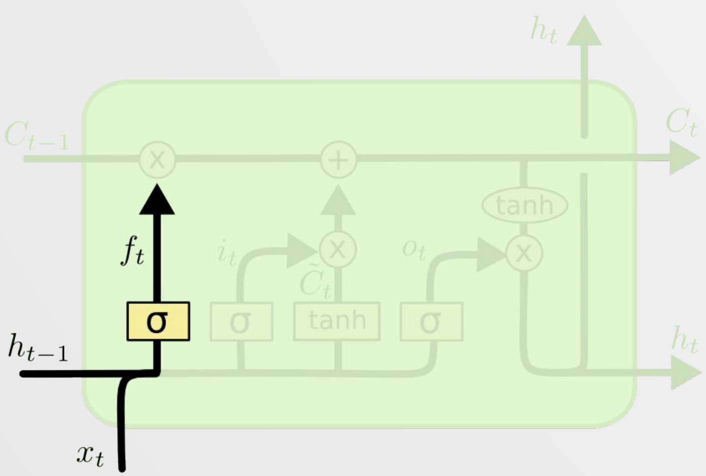




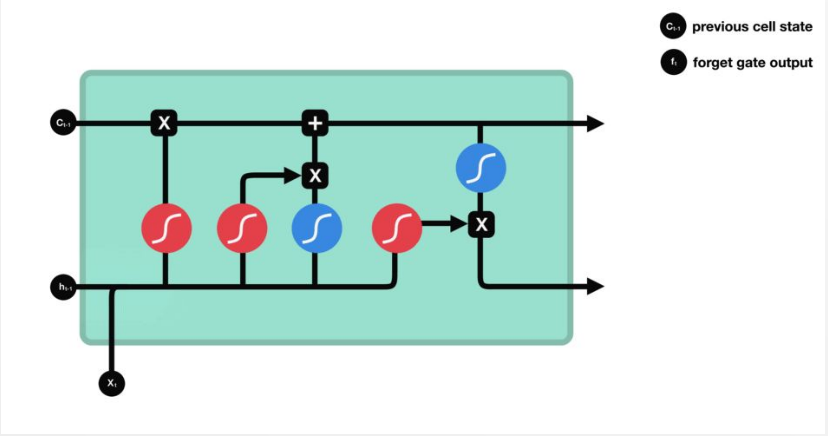
Gates decide what information should be withheld in the cell state.
They are a **sigmoid** followed by a pointwise ×.
Values near 0 block information; values near 1 pass information.

LSTM step 1: decide what to forget

- The **forget gate layer** compares h_{t-1} and the current input x_t to decide which elements in cell state C_{t-1} to keep and which to turn off.
  - E.g., the cell state might 'remember' the number (sing./plural) of the current subject, in order to predict appropriately conjugated verbs, but decide to forget it when a new subject is mentioned at x_t.
    - (There's scant evidence that such information is so readily interpretable.)

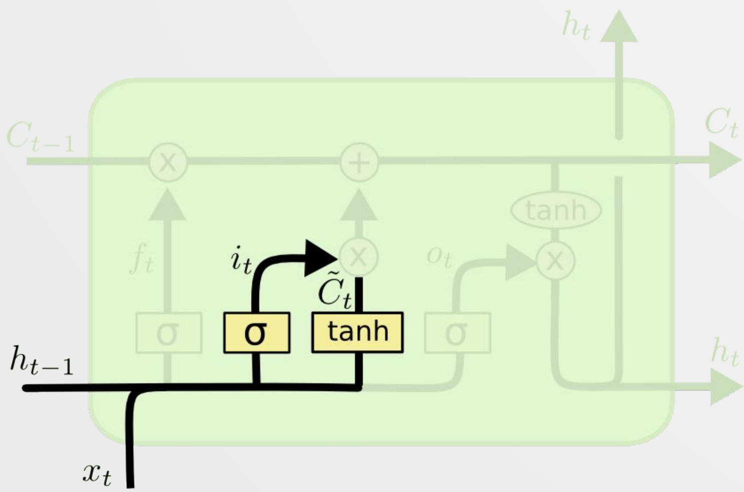


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$



LSTM step 2: decide what to store

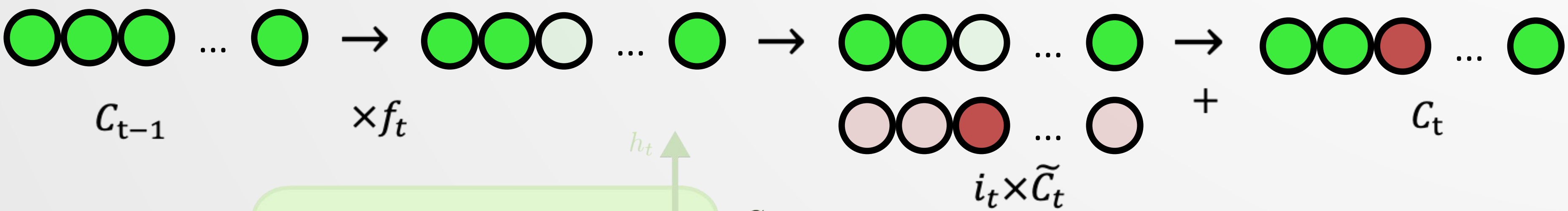
- The **input gate layer** has two steps.
  - First, a sigmoid layer σ decides which cell units to update.
  - Next, a tanh layer creates new candidate values $\tilde{C}_t$.
  - E.g., the σ can turn on the 'number' units, and the tanh can push information on the current subject.
  - The σ layer is important – we don't want to push information on units (i.e., latent dimensions) for which we have no information.

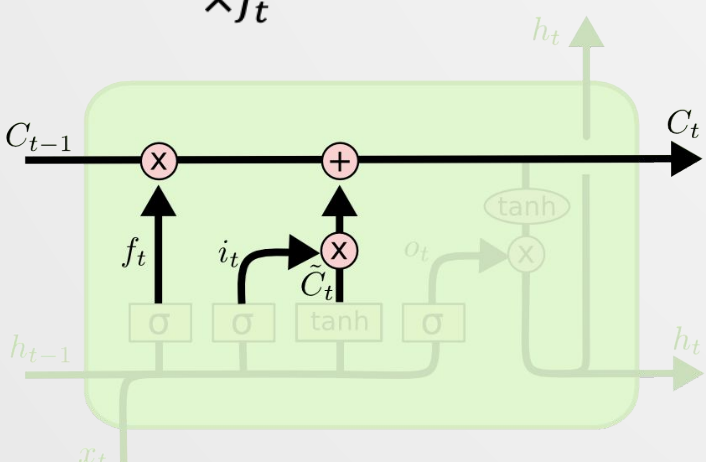


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

LSTM step 3: update the cell state

- Update C_{t-1} to C_t.
  - First, forget what we want to forget: multiply C_{t-1} by f_t.
  - Then, create a 'mask vector' of information we want to store, $i_t \times \tilde{C}_t$.
  - Finally, write this information to the new cell state C_t.





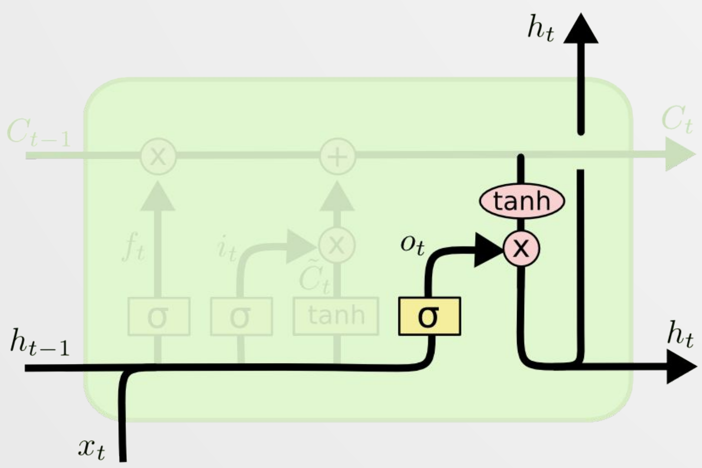
$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$$

UNIVERSITY OF TORONTO

LSTM step 4: output and feedback

- Output something, o_t, based on the current x_t and h_{t-1}.
- Combine the output with the cell to give your h_t.
  - Normalize cell C_t on [-1,1] using tanh and combine with o_t

- In some sense, C_t is **long-term** memory and h_t is the **short-term memory** (hence the name).

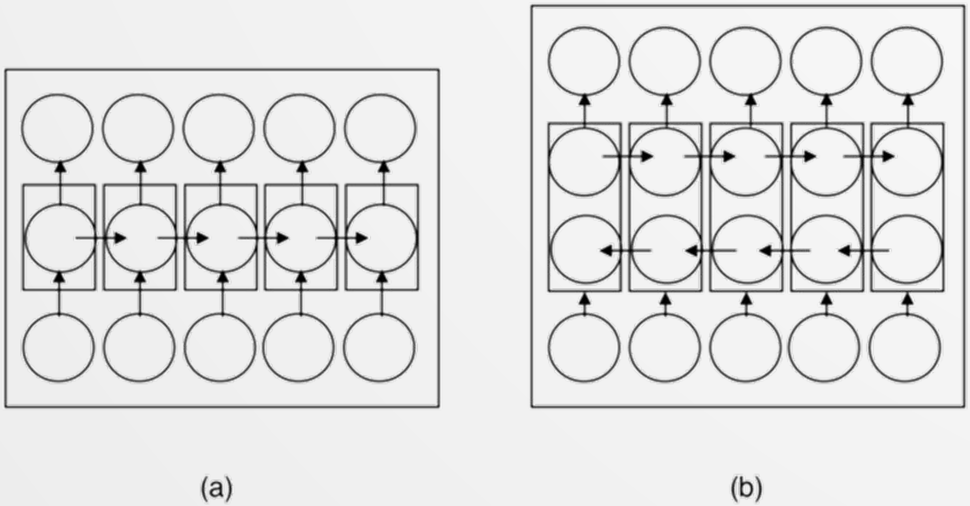


$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \times \tanh(C_t)$$

Variants of LSTMs

- There are many variations on LSTMs.
  - ‘*Bidirectional LSTMs*’ (and bidirectional RNNs generally), learn. (Similar: *Multi-stack RNNs)*

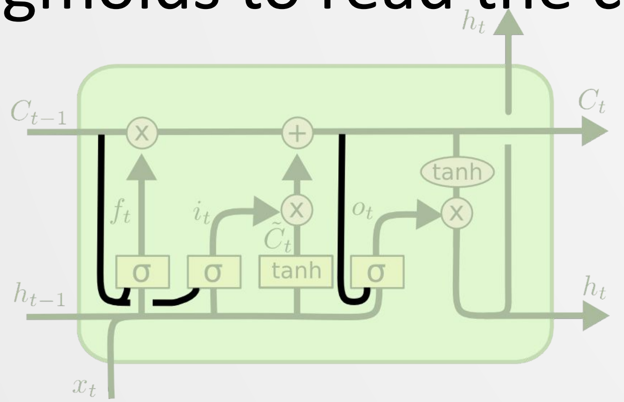


Structure overview
(a) unidirectional RNN
(b) bidirectional RNN

Schuster, Mike, and Kuldip K. Paliwal. (1997) Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on* **45(**11) (1997): 2673-2681.2.

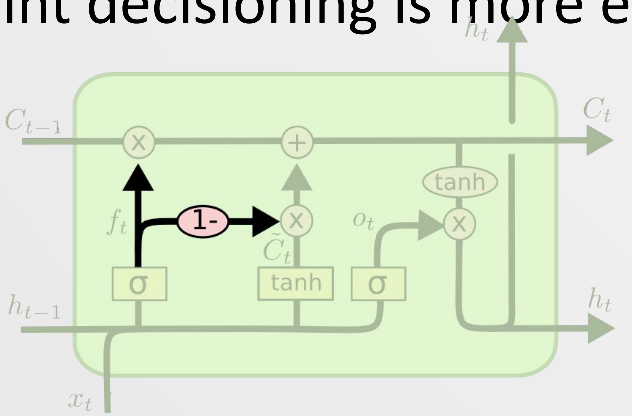
Variants of LSTMs

- Gers & Schmidhuber (2000) add ‘**peepholes**’ that allow all sigmoids to read the cell state.



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \;+\; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \;+\; b_o\right)$$

- We can **couple** the ‘*forget*’ and ‘*input*’ gates.
  - Joint decisioning is more efficient.

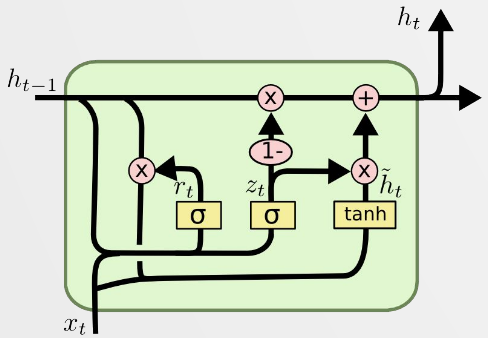


$$C_t = f_t * C_{t-1} + \boldsymbol{(1 - f_t)} * \tilde{C}_t$$

UNIVERSITY OF TORONTO

Aside - Variants of LSTMs

- **Gated Recurrent units** (GRUs; Cho *et al* (2014)) go a step further and also merge the cell and hidden states.



$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$ **Update** gate

$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$ **Reset** gate (0: replace units in h_{t-1} with those in x_t)

$\tilde{h}_t = \tanh(W \cdot [r_t * h_{t-1}, x_t])$

$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$

- Which of these variants is best? Do the differences matter?
  - Greff, et al. (2015) do a nice comparison of popular variants, finding that **they're all about the same**
  - Jozefowicz, et al. (2015) tested more than ten thousand RNN architectures, finding some that worked better than LSTMs on certain tasks.

CONTEXTUAL WORD EMBEDDINGS

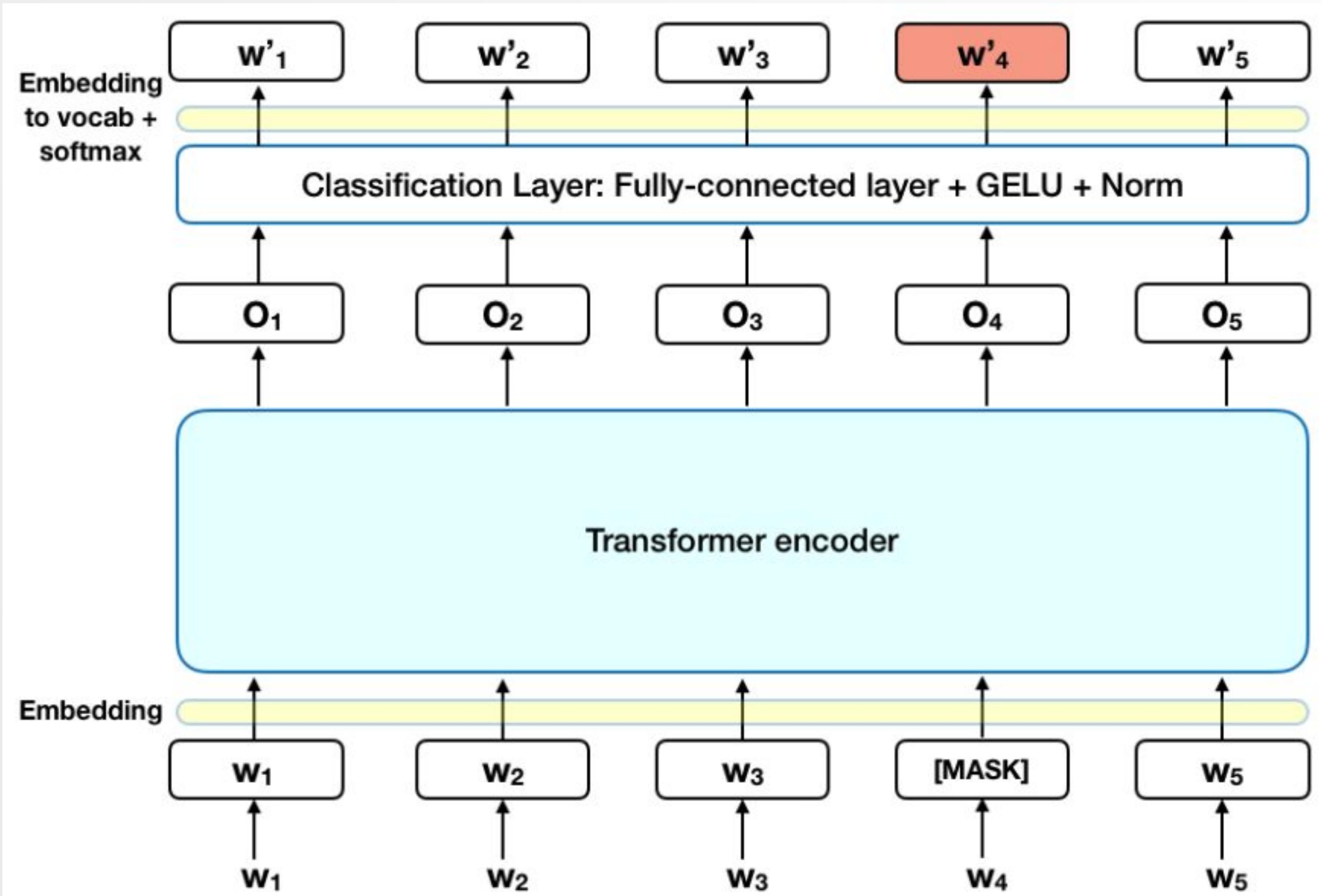
Deep contextualized representations

- What does the word *play* mean?

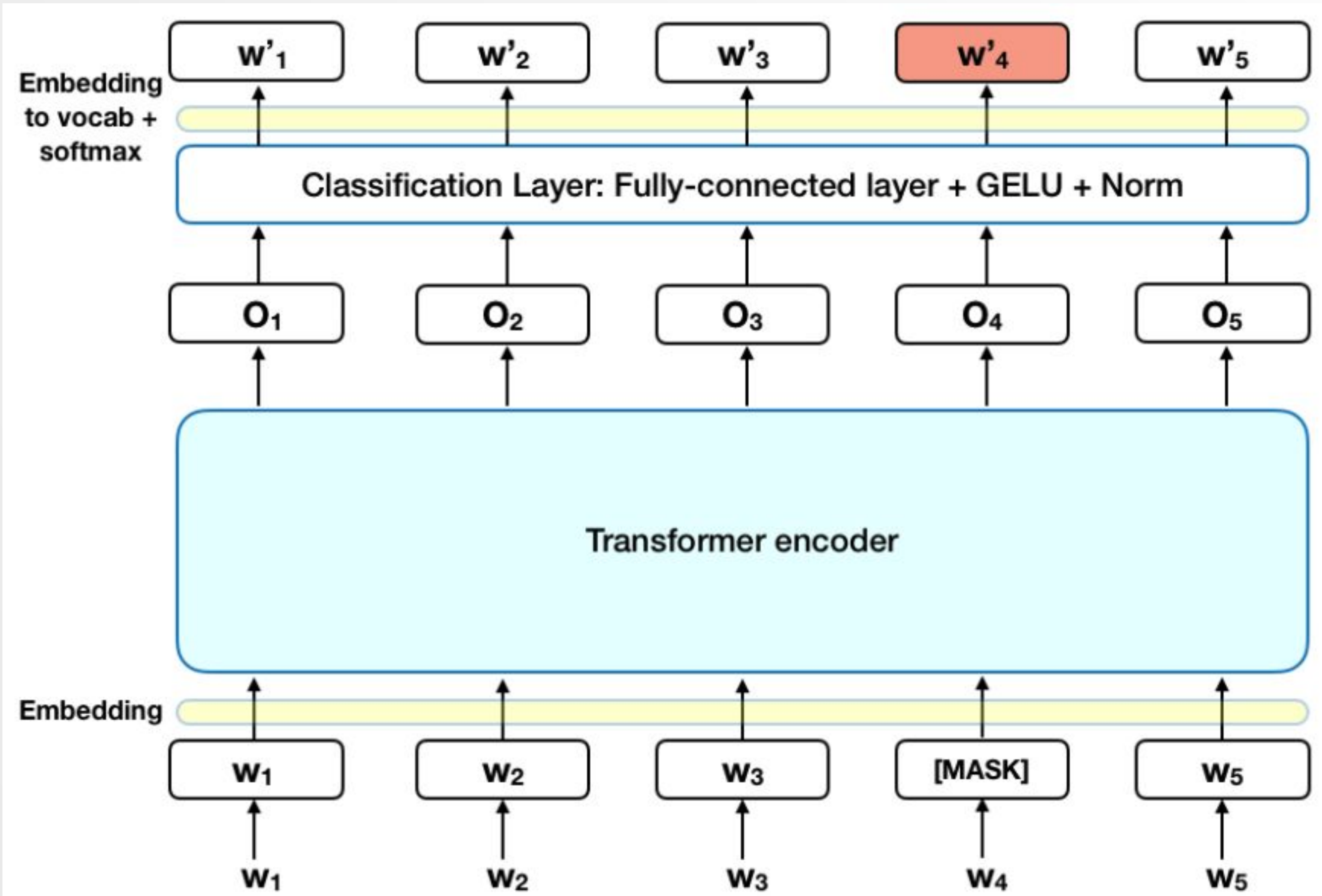
AllenNLP

Peters ME, Neumann M, Iyyer M, *et al.* (2018) Deep contextualized word representations. Published Online First: 2018. doi:10.18653/v1/N18-1202; http://arxiv.org/abs/1802.05365

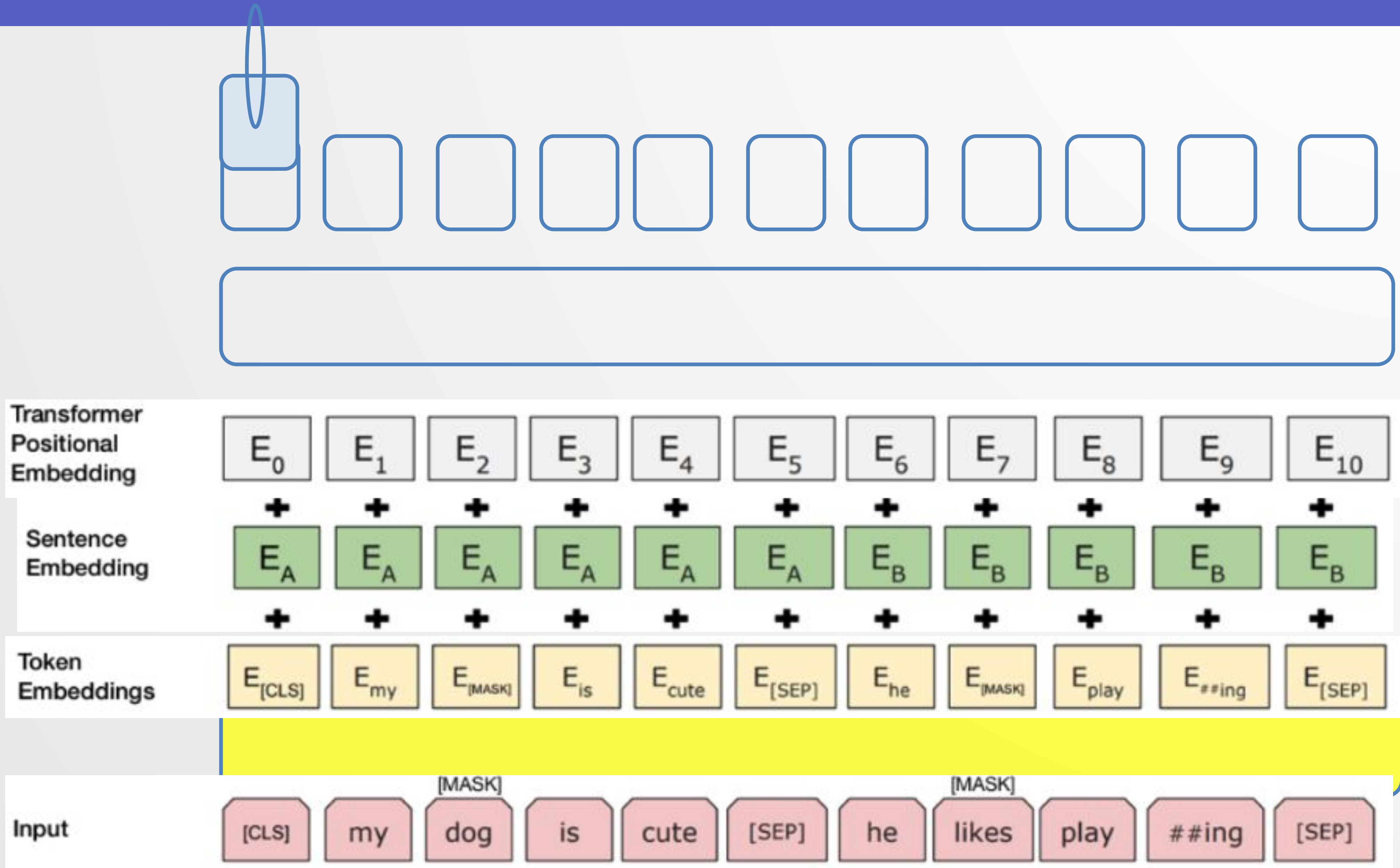
• BERT


w'_1
w'_2
w'_3
w'_4
w'_5
Embedding to vocab + softmax
Classification Layer: Fully-connected layer + GELU + Norm
O_1
O_2
O_3
O_4
O_5
Transformer encoder
Embedding
w_1
w_2
w_3
[MASK]
w_5
w_1
w_2
w_3
w_4
w_5

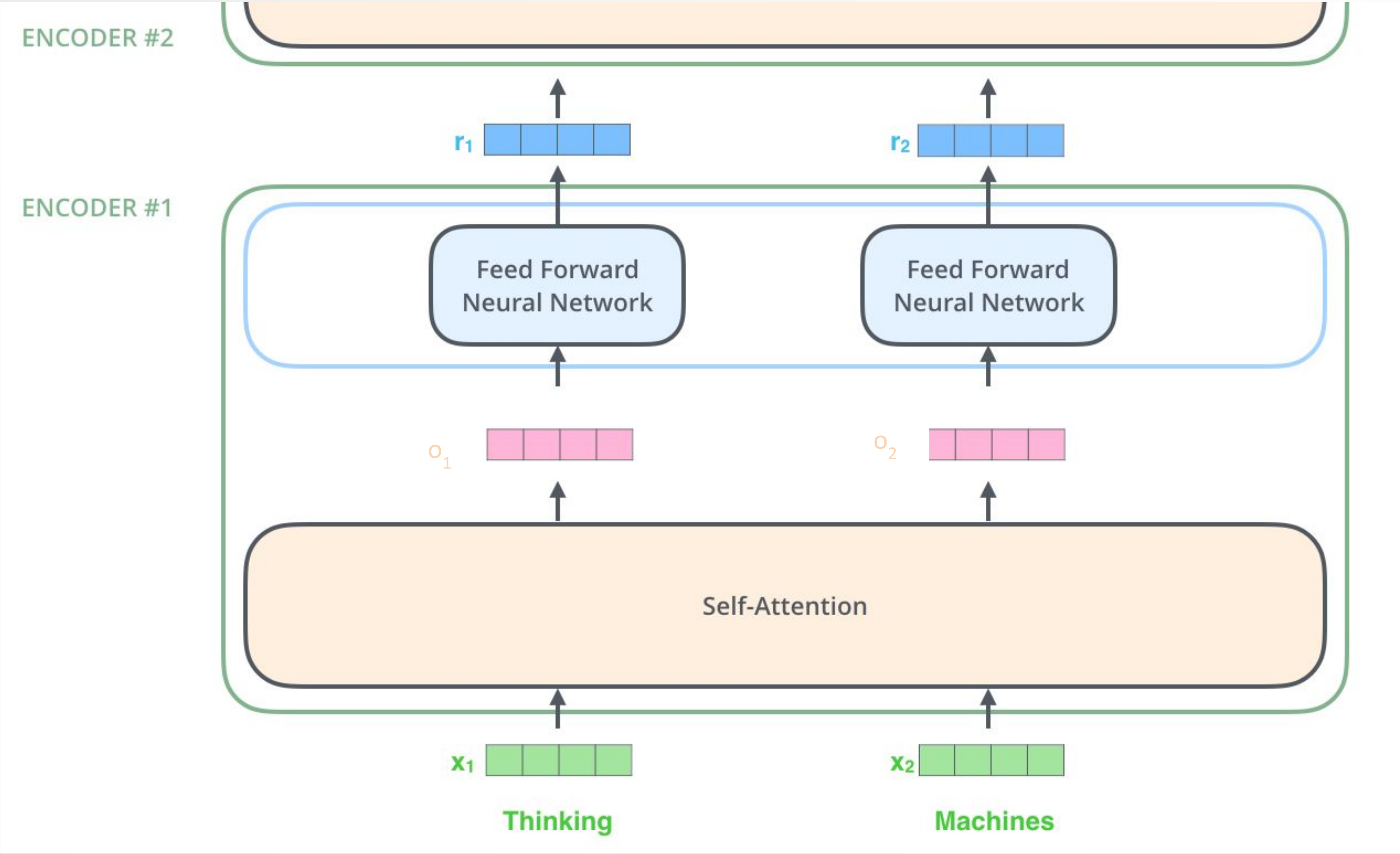
•Training task 1: Masking



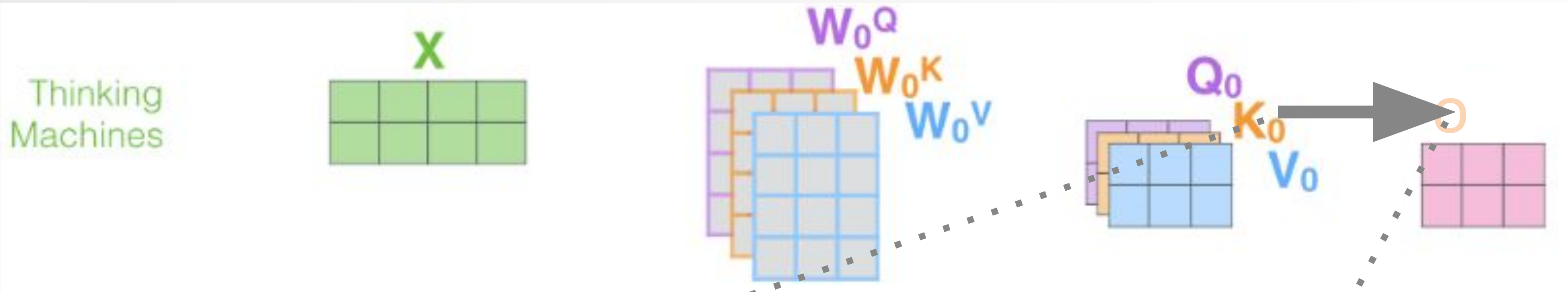
.Training task 2: Next Sent.



•Transformers

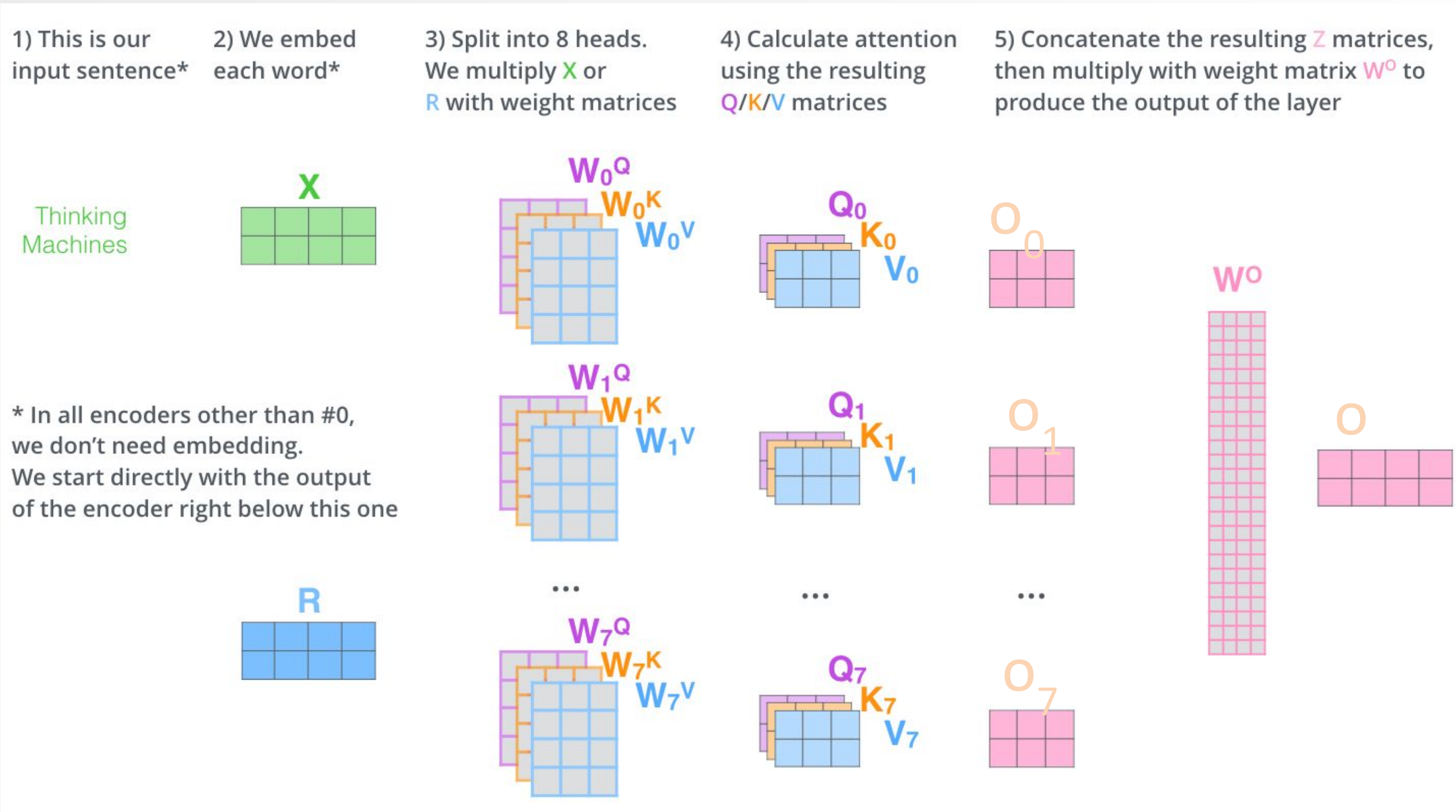


.Self-attention



$$\alpha_{ij} = \frac{\exp\left(q_i^T k_j\right)}{\sum_{l=1}^{n} \exp\left(q_i^T k_l\right)} \qquad o_i = \sum_{j=1}^{n} \alpha_{ij} v_j$$

.Multiheaded Self attention



.Positional encodings

$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

where

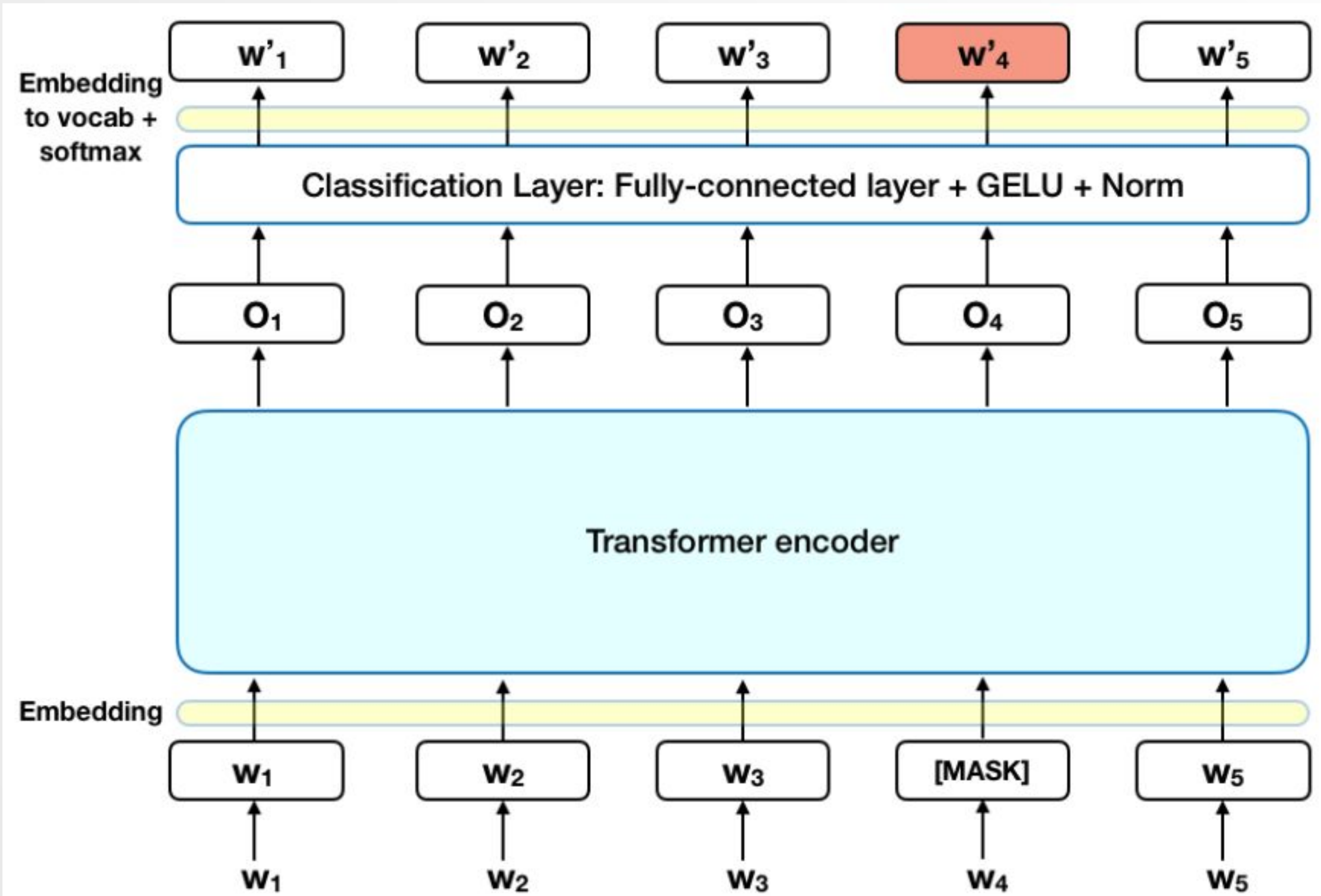
$$\vec{p_t}^{(i)} = f(t)^{(i)} := \begin{cases} \sin(\omega_k . t), & \text{if } i = 2k \\ \cos(\omega_k . t), & \text{if } i = 2k + 1 \end{cases}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

.Huh?

- Encodings of any two distinct positions are distinct
- Each position maps to only one encoding
- Test sentences may be longer than training
- Distance between two positions should be constant across sentences (of varying lengths).

•Training task 1: Masking



.The truth about masking

- Real easy to do well on MASKed position and nothing else
- Real easy to learn to copy the context-independent embedding
- So…
  - 80% of the time: MASK
  - 10% of the time: correct word
  - 10% of the time: another random word