# Word Embedding Models

Zhijun Yin

Assistant Professor of Biomedical Informatics and Computer Science

Wednesday, June 14, 2023

# Word Representation


https://xkcd.com/1838/

# One-Hot Vector (Traditional NLP)

**Vocabulary** : {machine, learning, system, pour, data, …, algebra}

*|V| unique words*

machine => (1, 0, 0, 0, 0, …, 0); learning => (0, 1, 0, 0, 0, …, 0)

*vector of size |V|* *vector of size |V|*

"machine learning system … " => (1, 1, 0, 0, 1, …, 0)

$\mathbf{x}=(x_1, x_2, x_3, x_4, x_5, \ldots, x_{|v|})$

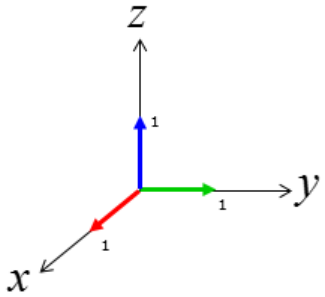
# One-Hot Vector (Traditional NLP)

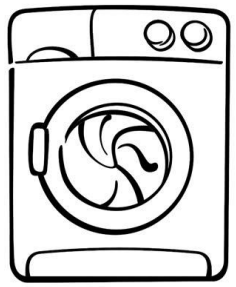
But how does (1, 0, 0, 0, 0, …, 0) carry the meaning of machine?

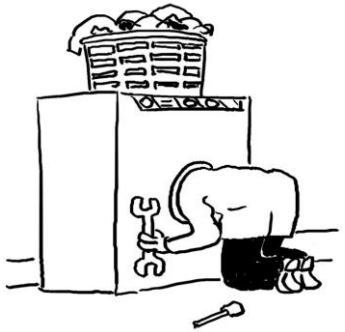
What if another word, say, device, is denoted by (0, 0, 1, 0, 0, …, 0)?

What if a misspelling, say, machin, is denoted by (0, 0, 0, 0, 0, …, 1)?

Can you tell their similarity based on these one-hot vectors?

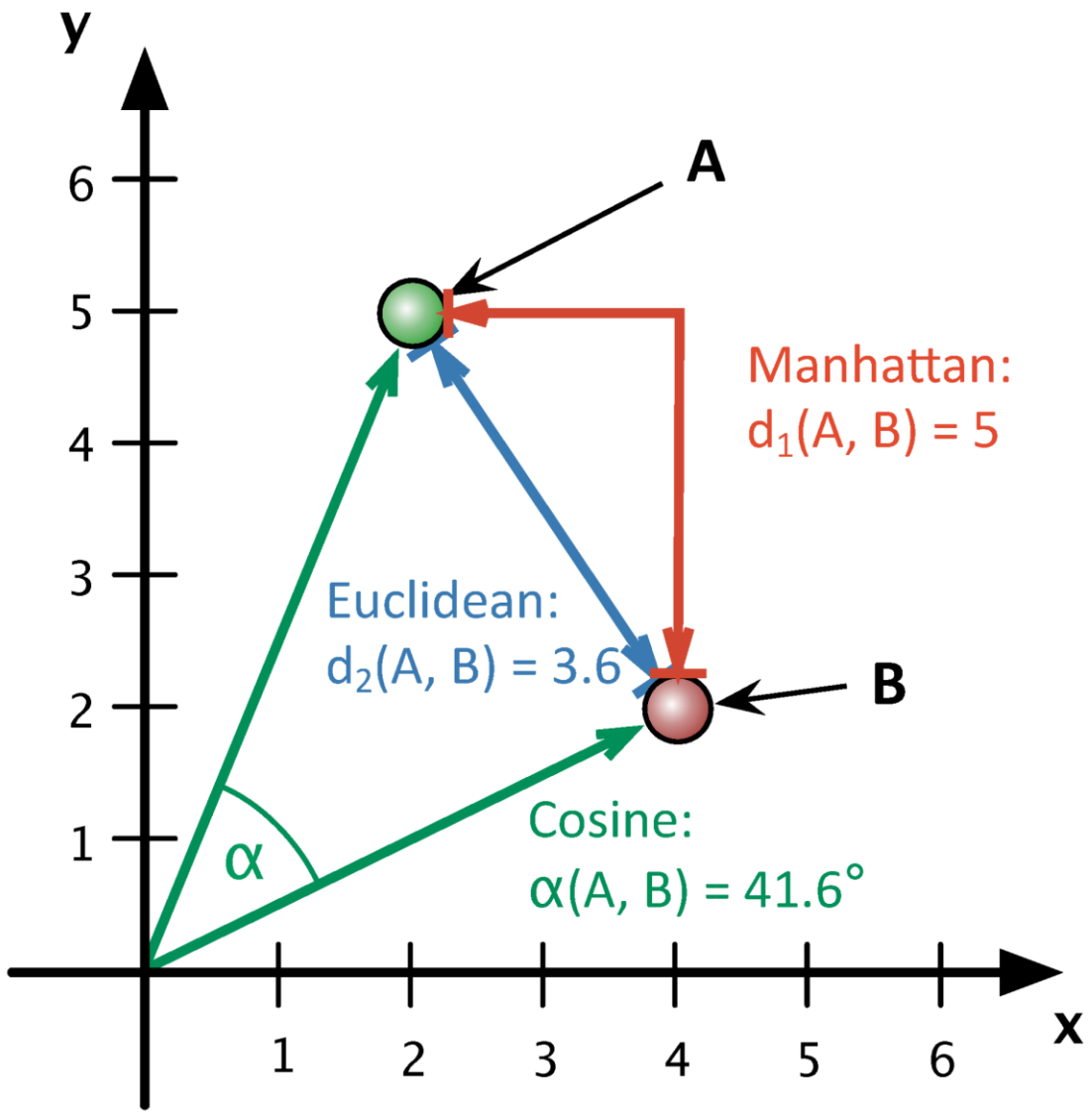




# Encoding Similarity in Vectors

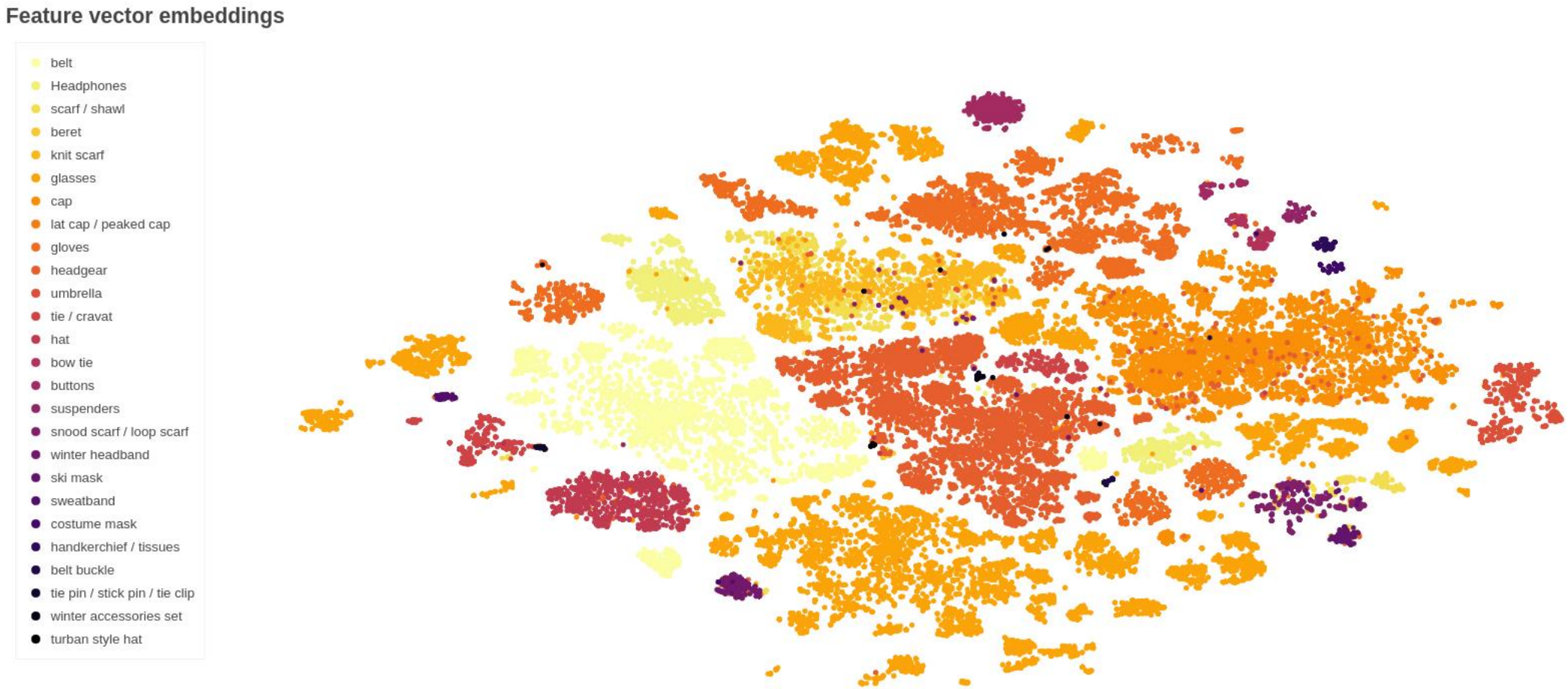
Model



# Word Vectors


- Espresso? But I ordered a cappuccino!
- Don't worry, the cosine distance between them is so small that they are almost the same thing.

# Visualization of Word Vectors




https://visual-meta.com/2018/07/20/product-embeddings/

# Context Defines Meanings

No matter how convincing the **machine** is, once I know it is a **machine**, I won't care about it anymore.

The **machine** should start looking for correlations we would not expect.

Algorithm

Yes, our body is just a **machine** for living, that is all.

The implication is always that some people are simply unable to do any job that a **machine** cannot do.

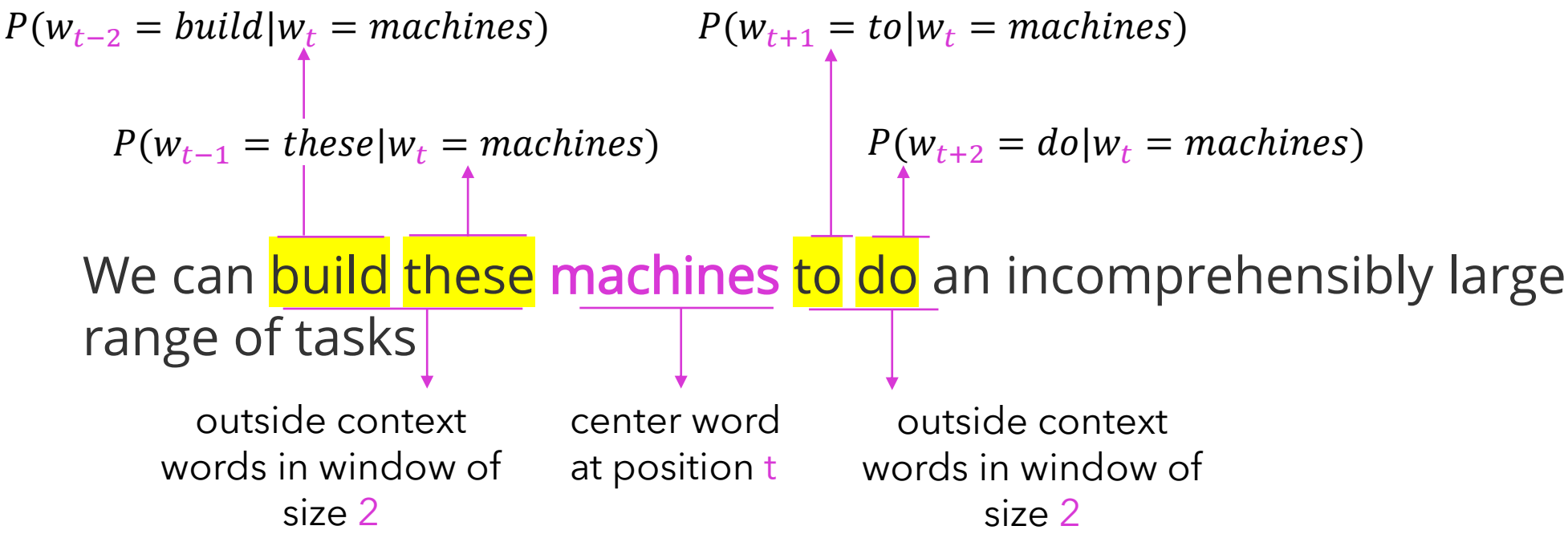
The shells and kernels are then separated in a winnowing **machine**.

# Word2vec General Idea

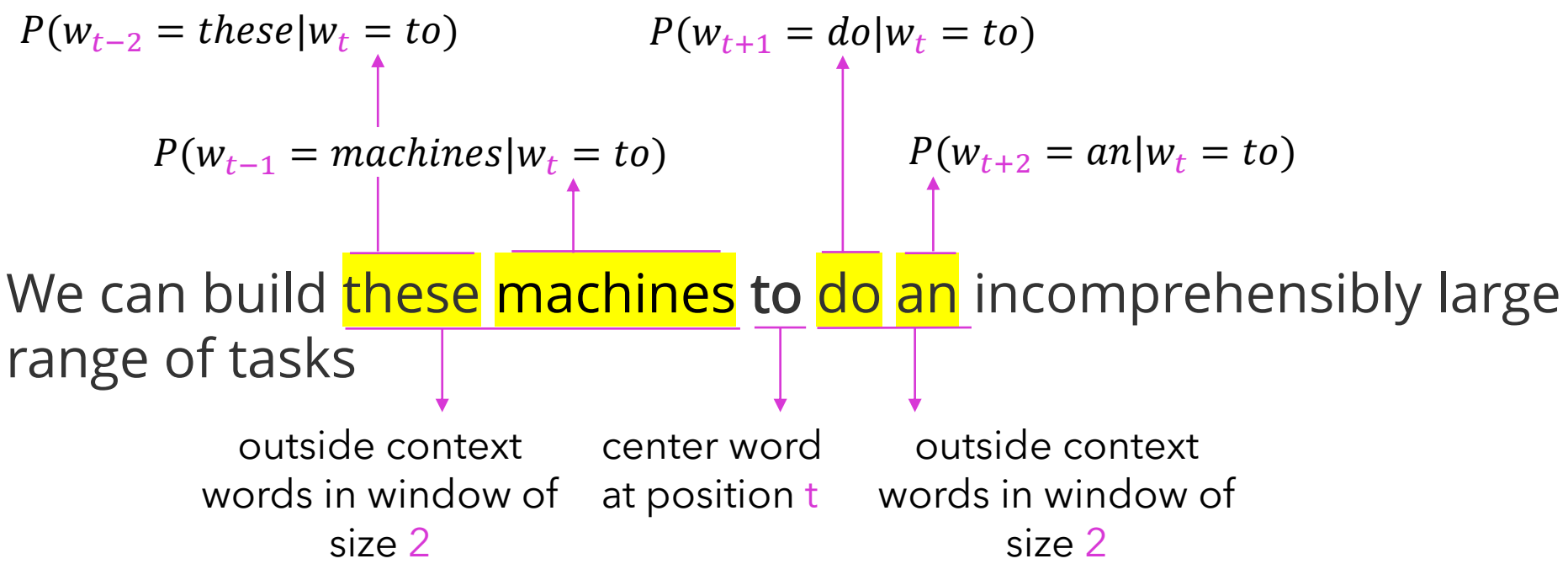
- Given a large corpus of text, the goal is to represent each word in the vocabulary by a vector
- Go through each word in the text, use the similarity of word vectors to calculate the probability of the neighbor words given the word (or vice versa)
- Keep adjusting the word vectors in order to maximize this probability

Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." *Advances in neural information processing systems*. 2013.

# Word2vec Example



# Word2vec Example



# Word2vec Objective Function

*Likelihood* *All parameters to be optimized*

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

*window size*

minimize

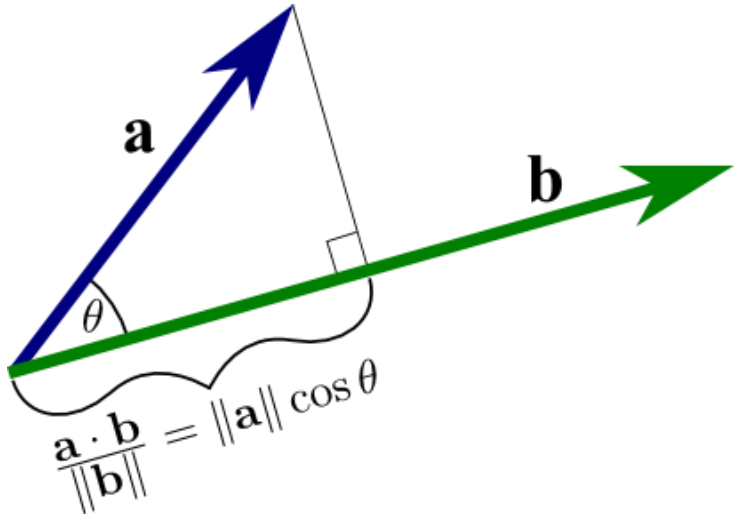
$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

*(average) negative log likelihood*

# Word2vec Objective Function

How to calculate $P(w_{t+j}|w_t;\theta)$?

- Use two vectors for each word $w$
  - $v_w$ when $w$ is a center word
  - $u_w$ when $w$ is a context word
- For a center word $c$ and a context word $o$:



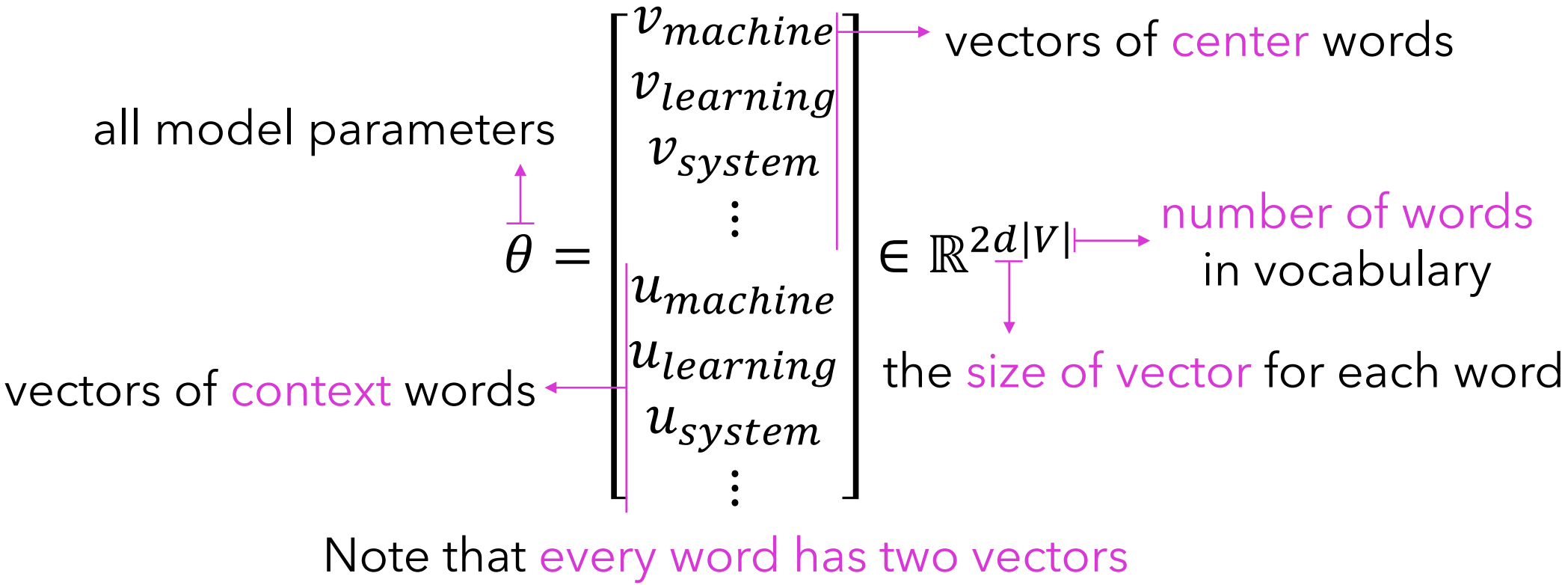

*positive*

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

*Dot product measures similarity of $c$ and $o$*

→ *generate probability distribution* → *Softmax($u_o^T v_c$)*

# Word2vec Optimization



# Gradient Descent

Derivative, a real number, masures the sensitivity (rate) of change of $\mathrm{J}(\theta)$ with respect to $\theta$

How do we change $\theta$ to minimize $\mathrm{J}(\theta)$ ?

Answer: Change $\theta$ against the derivative

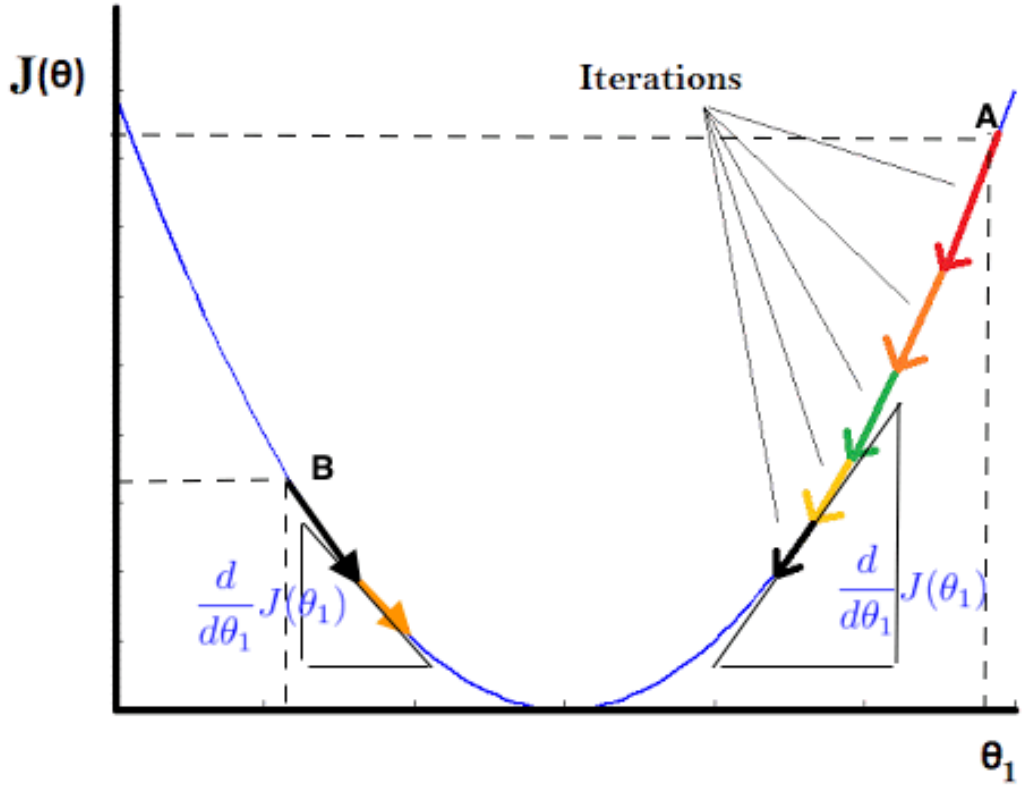
*learning rate (step size)*

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

*each iteration*

*each parameter*

$$\theta_j^{new} = \theta_j^{new} - \alpha \frac{\partial}{\partial \theta_j} \mathrm{J}(\theta)$$



# Inefficiency of Minimizing Loss Function with Gradient Descent

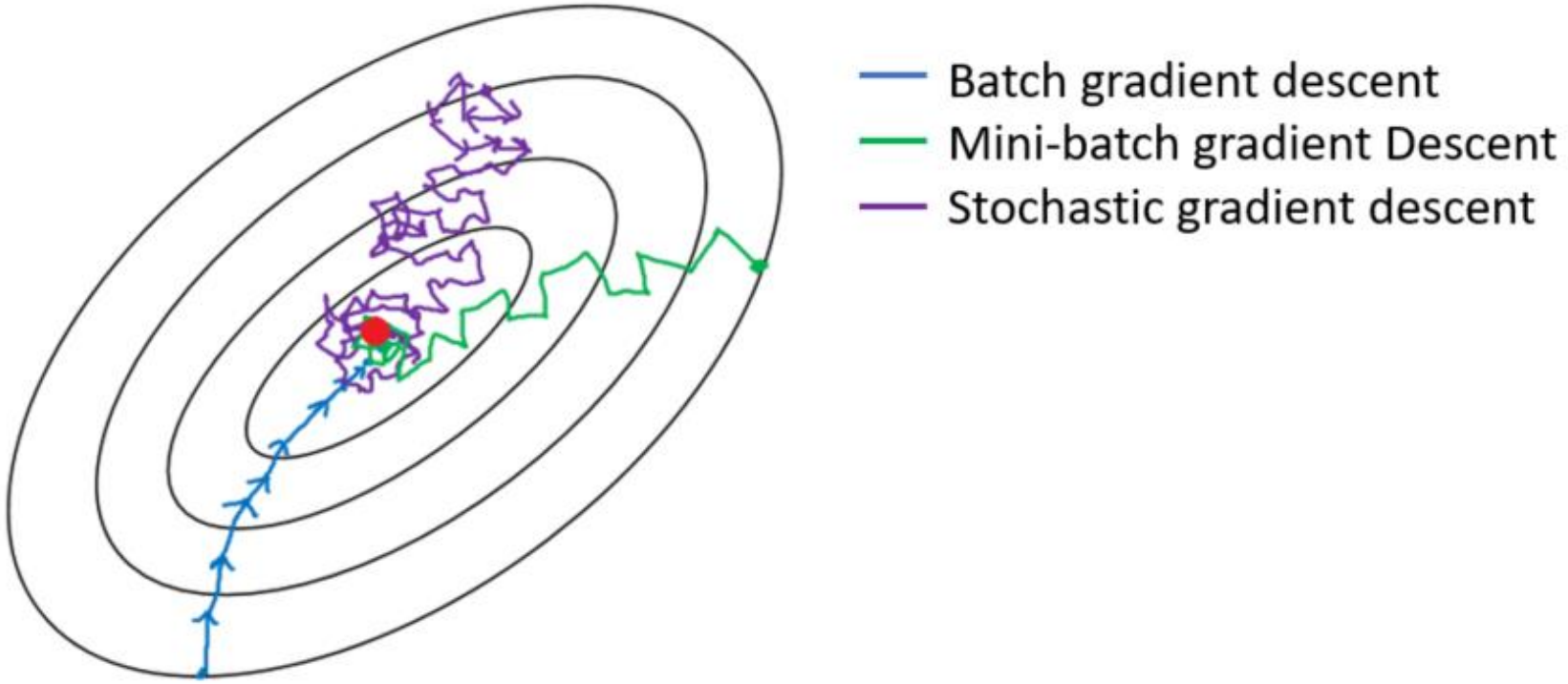
Go through the entire Documents

$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \log P(w_{t+j}|w_t;\theta)$$

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Go through the entire Vocabulary

# Variants of Gradient Descent




https://suniljangirblog.wordpress.com/2018/12/13/variants-of-gradient-descent/

# Negative Sampling

Maximize the similarity of the words in the same context and minimize it when they occur in different contexts

$$L(\theta) = \log \sigma(u_o^T v_c) + \sum_{k=1}^{K} E_{w_k \sim P_{n(w)}} [\log \sigma(-u_k^T v_c)]$$

$$P_n(w) = \frac{U(w)^{3/4}}{Z}$$

# Deriving the Objective for Negative Sampling

$$\arg\max_{\theta} \prod_{(a,b)\in D} p(D=1|a,b;\theta) \prod_{(c,o)\in D'} p(D=0|c,o;\theta)$$

*Probability that a and b are neighbors*

*Probability that c and o are **NOT** neighbors*

$$\arg\max_{\theta} \sum_{(a,b)\in D} \log p(D=1|a,b;\theta) + \sum_{(c,o)\in D'} \log(1-p(D=1|c,o;\theta))$$

$$\sigma(u_b^T v_a) = \frac{1}{1+e^{-u_b^T v_a}}$$

# Subsampling of Frequent Words

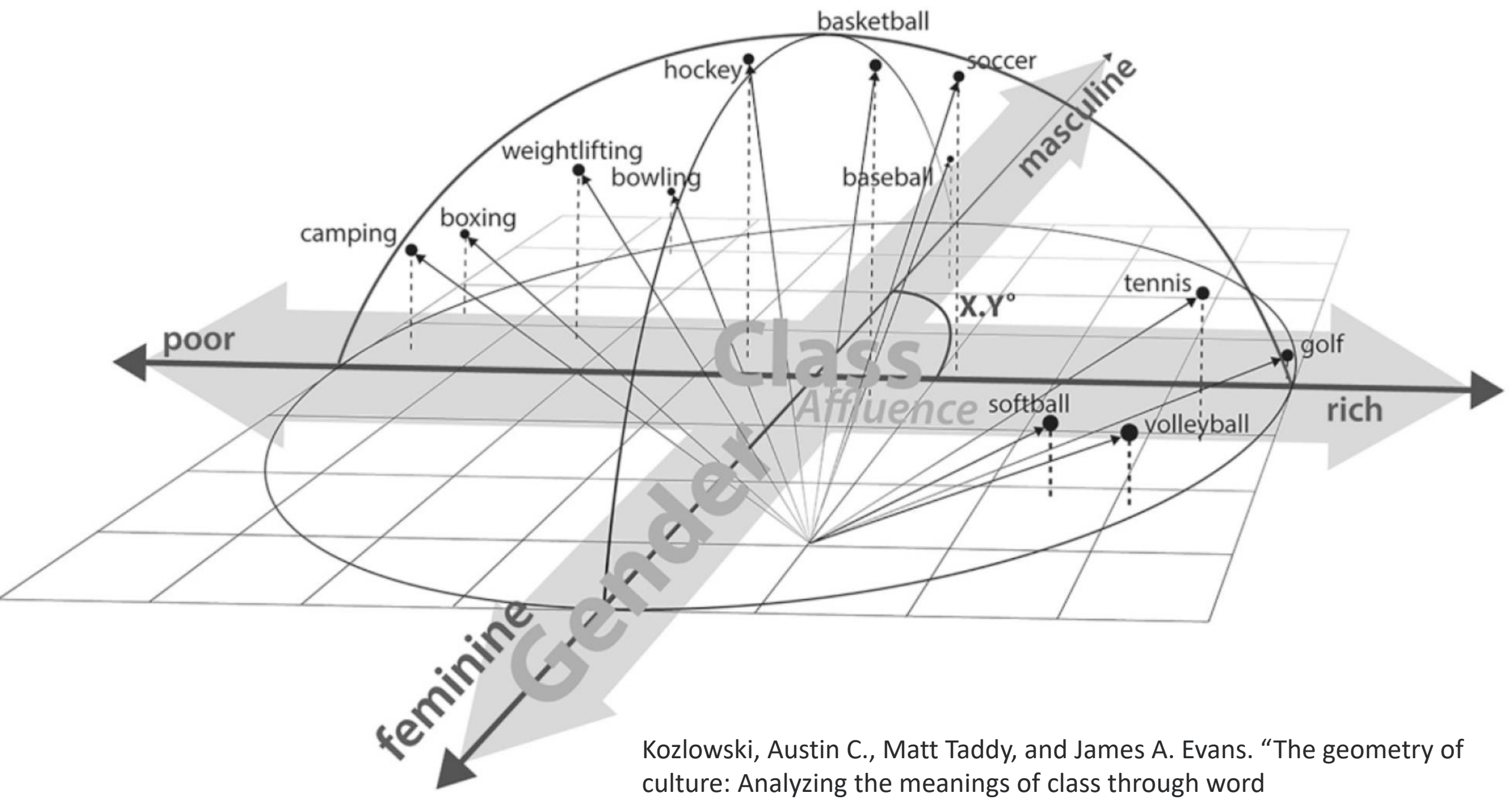
In very large corpora, the most frequent words can easily occur hundreds of millions of times (e.g., “in”, “the”, and “a”). Such words usually provide less information value than the rare words. (“France”, “Paris”) vs (“France”, ”the”)

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

*A chosen threshold, typically around* $10^{-5}$

*Frequency of word* $w_i$

It aggressively subsamples words whose frequency is greater than t while preserving the ranking of the frequencies




Kozlowski, Austin C., Matt Taddy, and James A. Evans. “The geometry of culture: Analyzing the meanings of class through word embeddings.” *American Sociological Review* 84, no. 5 (2019): 905–949.

# Debiasing Word Embeddings?

## Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

**Tolga Bolukbasi**[1], **Kai-Wei Chang**[2], **James Zou**[2], **Venkatesh Saligrama**[1,2], **Adam Kalai**[2]
[1]Boston University, 8 Saint Mary's Street, Boston, MA
[2]Microsoft Research New England, 1 Memorial Drive, Cambridge, MA
tolgab@bu.edu, kw@kwchang.net, jamesyzou@gmail.com, srv@bu.edu, adam.kalai@microsoft.com

# About Bias


Yann LeCun
@ylecun
People are biased.
Data is biased, in part because people are biased.
Algorithms trained on biased data are biased.
But learning algorithms themselves are not biased.
Bias in data can be fixed.
Bias in people is harder to fix.
Published 2019
nytimes.com
Biased Algorithms Are Easier to Fix Than Biased People (Published 2019)
Racial discrimination by algorithms or by people is harmful — but that's where the similarities end.
1:17 AM · Dec 7, 2019 · Facebook


Elements of
Deep Neural
Networks

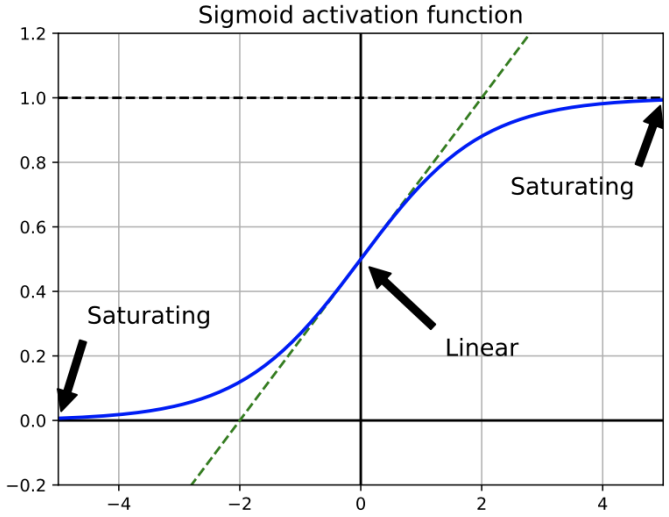
# Multi-Layer Neural Network

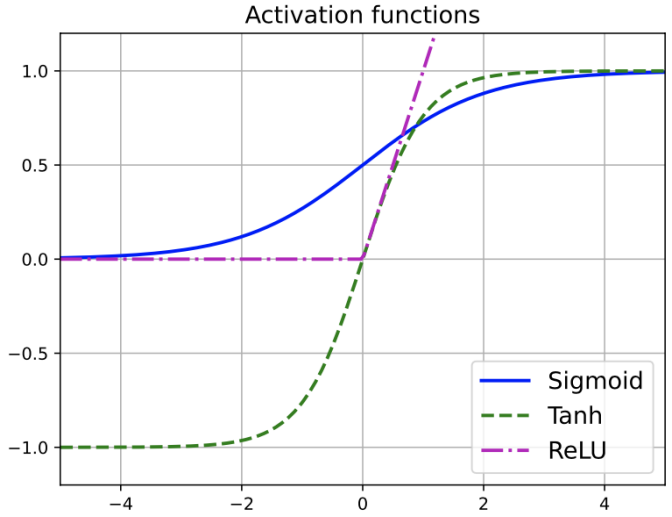
- The assumption of the linear input-output mapping is too strong, leading to a limited model capacity: $f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$
- One extension is to perform a feature transformation $f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b}$, but still linear and limiting
- A natural extension is to allow feature transformation with its own parameters, $\boldsymbol{\theta_2}$, $f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{x};\boldsymbol{\theta_2}) + \boldsymbol{b}$
- If we compose $L$ functions, we have $f(\boldsymbol{x};\boldsymbol{\theta}) = f_L(f_{L-1}(\cdots(f_1(\boldsymbol{x}))\cdots))$, where $f(\boldsymbol{x};\boldsymbol{\theta}_\ell))$ is the function at layer $\ell$

# Differentiable $f(\boldsymbol{x}; \boldsymbol{\theta}_\ell)$

- Differential activation function $\varphi: \mathbb{R} \to \mathbb{R}$
- Hidden units at each layer: $z_l = f(z_{l-1}) = \varphi_l(\boldsymbol{b}_l + \boldsymbol{W}_l z_{l-1})$

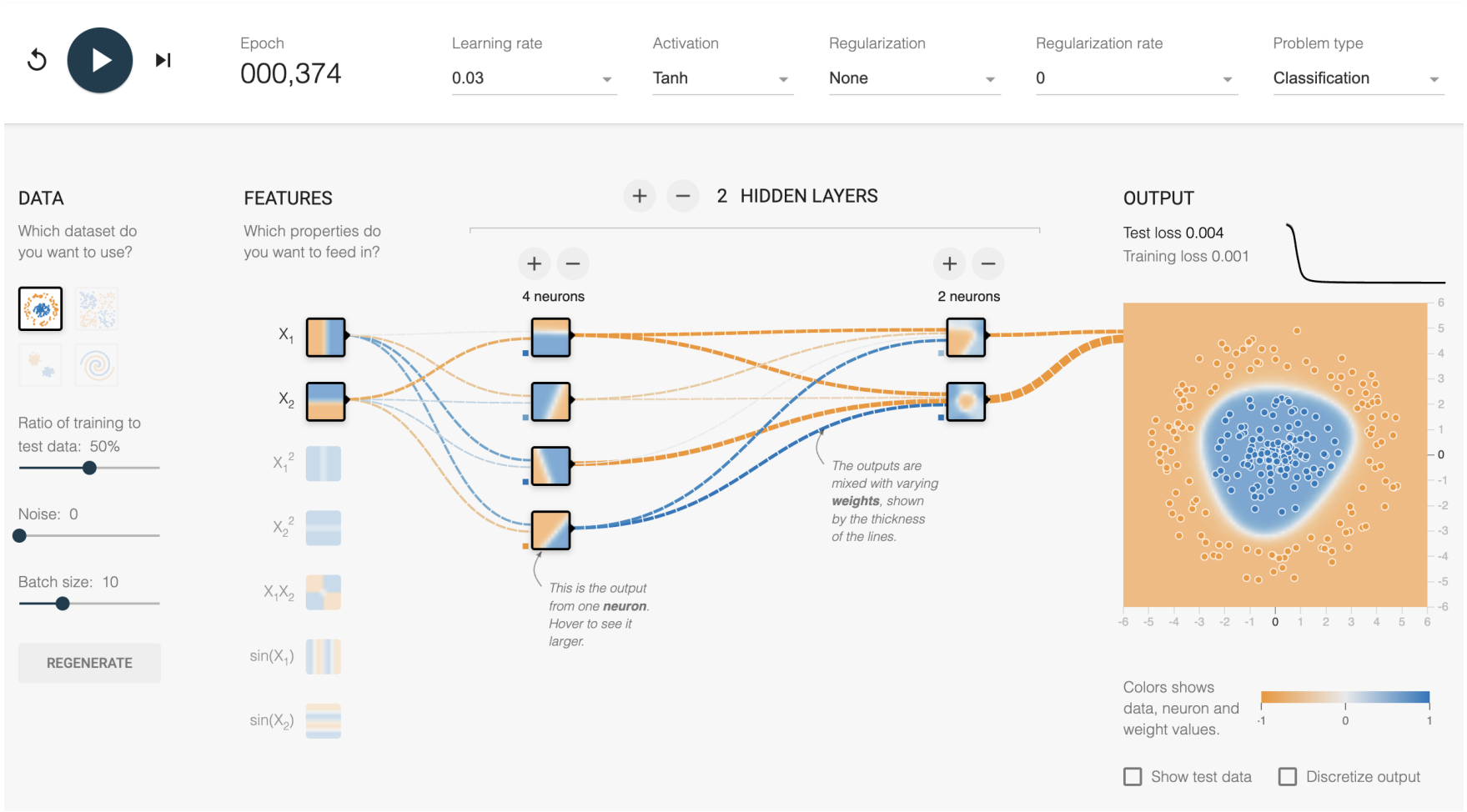
Pre-activations: $a_l$



# Illustration of NN with 2 Hidden Layers

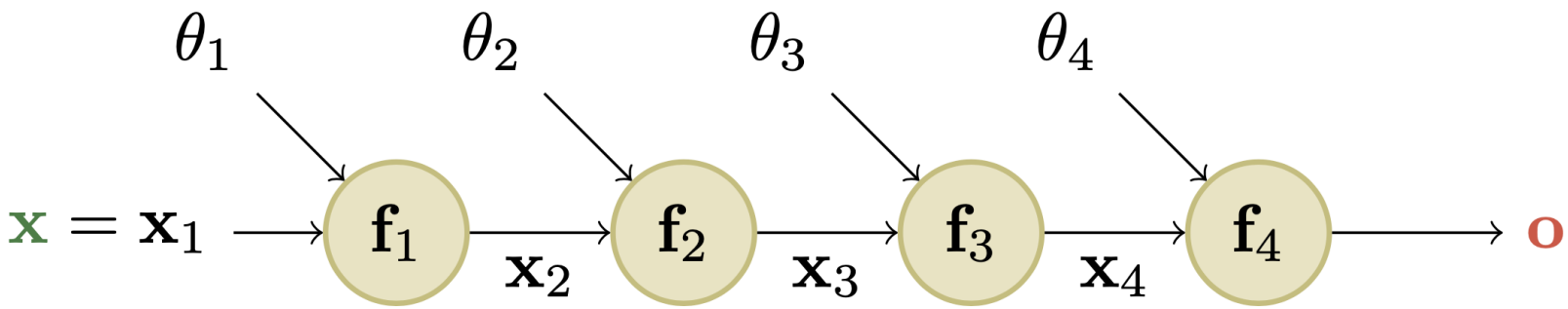
http://playground.tensorflow.org



# Backpropagation

$$o = f(x) = f_4 \circ f_3 \circ f_2 \circ f_1$$

$$f_1 \colon \mathbb{R}^n \longrightarrow \mathbb{R}^{m_1}, f_2 \colon \mathbb{R}^{m_1} \longrightarrow \mathbb{R}^{m_2}, f_3 \colon \mathbb{R}^{m_2} \longrightarrow \mathbb{R}^{m_3}, f_4 \colon \mathbb{R}^{m_3} \longrightarrow \mathbb{R}^{m}$$



# Jacobian Using Chain Rules

- $\boldsymbol{J}_f(\boldsymbol{x}) = \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{x}} \in \mathbb{R}^{m \times n}$
- $\frac{\partial \boldsymbol{o}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{x_4}} \frac{\partial \boldsymbol{x_3}}{\partial \boldsymbol{x_2}} \frac{\partial \boldsymbol{x_2}}{\partial \boldsymbol{x}} = \frac{\partial f_4(\boldsymbol{x_4})}{\partial \boldsymbol{x_4}} \frac{\partial f_3(\boldsymbol{x_3})}{\partial \boldsymbol{x_3}} \frac{\partial f_2(\boldsymbol{x_2})}{\partial \boldsymbol{x_2}} \frac{\partial f_1(\boldsymbol{x_1})}{\partial \boldsymbol{x}}$
- $\boldsymbol{J}_f(\boldsymbol{x}) = \frac{\partial \boldsymbol{o}}{\partial \boldsymbol{x}} = \boldsymbol{J}_{\boldsymbol{f}_4}(\boldsymbol{x}_4)\boldsymbol{J}_{\boldsymbol{f_3}}(\boldsymbol{x}_3)\, \boldsymbol{J}_{\boldsymbol{f}_2}(\boldsymbol{x}_2)\, \boldsymbol{J}_{\boldsymbol{f}_1}(\boldsymbol{x})$
- $\boldsymbol{J}_f(\boldsymbol{x}) = \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla f_1(\boldsymbol{x})^T \\ \vdots \\ \nabla f_m(\boldsymbol{x})^T \end{pmatrix} = \left( \frac{\partial \boldsymbol{f}}{\partial x_1}, \cdots, \frac{\partial \boldsymbol{f}}{\partial x_n} \right) \in \mathbb{R}^{m \times n}$

# Jacobian Using Chain Rules

- Extract the i[th] row from $\boldsymbol{J}_f(\boldsymbol{x})$ by using a vector Jacobian product (VJP) of the form $\boldsymbol{e}_i^T \boldsymbol{J}_f(\boldsymbol{x})$
- Extract the j[th] column from $\boldsymbol{J}_f(\boldsymbol{x})$ by using a Jacobian vector product (JVP) of the form $\boldsymbol{J}_f(\boldsymbol{x})\boldsymbol{e}_j$
- If $n < m$, it is more efficient to compute $\boldsymbol{J}_f(\boldsymbol{x})$ for each column $j = 1{:}n$ by using JVPs in a right-to-left manner. The right multiplication with a column vector $\boldsymbol{v}$ is

$$\boldsymbol{J}_f(\boldsymbol{x})\,\boldsymbol{v} = \boldsymbol{J}_{\boldsymbol{f}_4}(\boldsymbol{x}_4)\,\boldsymbol{J}_{\boldsymbol{f}_3}(\boldsymbol{x}_3)\,\boldsymbol{J}_{\boldsymbol{f}_2}(\boldsymbol{x}_2)\,\boldsymbol{J}_{\boldsymbol{f}_1}(\boldsymbol{x})\,\boldsymbol{v}$$

- If $n > m$, it is more efficient to compute $\boldsymbol{J}_f(\boldsymbol{x})$ for each row i $= 1{:}m$ by using VJPs in a left-to-right manner. The left multiplication with a row vector $u^T$ is

$$u^T\boldsymbol{J}_f(\boldsymbol{x}) = u^T\boldsymbol{J}_{\boldsymbol{f}_4}(\boldsymbol{x}_4)\,\boldsymbol{J}_{\boldsymbol{f}_3}(\boldsymbol{x}_3)\,\boldsymbol{J}_{\boldsymbol{f}_2}(\boldsymbol{x}_2)\,\boldsymbol{J}_{\boldsymbol{f}_1}(\boldsymbol{x})$$

# Backpropagation

$$\mathcal{L} = f_4 \circ f_3 \circ f_2 \circ f_1$$
$$x_2 = f_1(x, \theta_1) = W_1 x$$
$$x_3 = f_2(x_2, \emptyset) = \varphi(x_2)$$
$$x_4 = f_3(x_3, \theta_3) = W_3 x_3$$
$$\mathcal{L} = f_4(x_4, y) = \frac{1}{2}\|x_4 - y\|^2$$

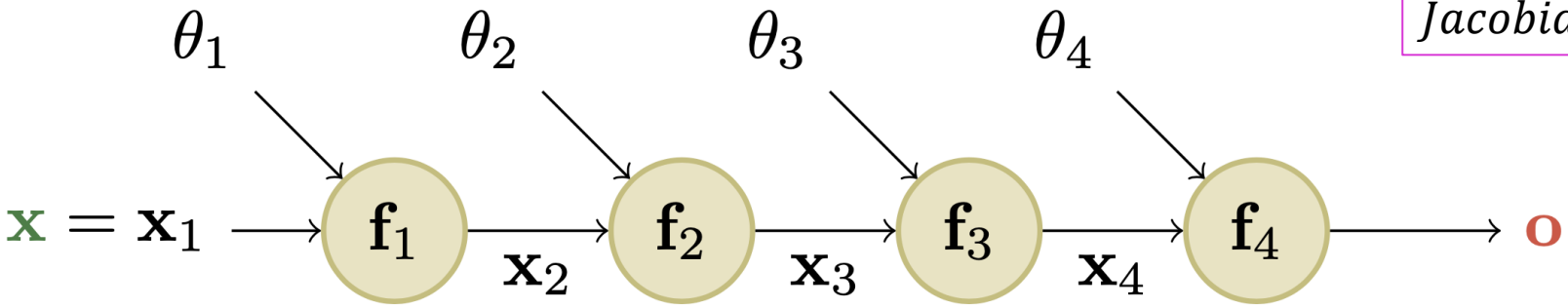
**Gradient Row Vector (GRV)** with dimension equal to the number of parameters in layer 3

**Jacobian,** which is an $n_4 \times n_3$ *matrix*

$$\frac{\partial \mathcal{L}}{\partial \theta_3} = \frac{\partial \mathcal{L}}{\partial x_4}\frac{\partial x_4}{\partial \theta_3}$$
$$\frac{\partial \mathcal{L}}{\partial \theta_2} = \frac{\partial \mathcal{L}}{\partial x_4}\frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial \theta_2}$$
$$\frac{\partial \mathcal{L}}{\partial \theta_1} = \frac{\partial \mathcal{L}}{\partial x_4}\frac{\partial x_4}{\partial x_3}\frac{\partial x_3}{\partial x_2}\frac{\partial x_2}{\partial \theta_1}$$

**Algorithm**:

1. Forward propagation to obtain all the $x_l$ and $\mathcal{L}$, or $f_l(x_l, \theta_l)$
2. Backward propagation. In each layer $l$, calculate:
   1. $\frac{\partial \mathcal{L}}{\partial \theta_l} = GRVX_{l+1}\frac{\partial f_l(x_l, \theta_l)}{\partial \theta_l}$ ($x_l$)
   2. $GRV_l = GRVX_{l+1}\frac{\partial f_l(x_l, \theta_l)}{\partial x_l}$ (*Jacobian*!)



# Backpropagation

- Objective function layer
  - E.g., cross entropy layer $\boldsymbol{J} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{x}} = (\boldsymbol{p} - \boldsymbol{y})^T \in \mathbb{R}^{1 \times C}$, $\boldsymbol{p} = softmax(\boldsymbol{x})$
- **Elementwise** nonlinearity
  - $\boldsymbol{J} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = diag(\varphi'(\boldsymbol{x}))$
- Linear layer
  - $\boldsymbol{J} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}} = W \in \mathbb{R}^{m \times n}$, where $\boldsymbol{x} \in \mathbb{R}^n$ , $\boldsymbol{z} \in \mathbb{R}^m$
  - $\boldsymbol{J} = \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} \in \mathbb{R}^{m \times (m \times n)}$, where $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}_{ij}} = (0 \quad \cdots \quad x_j \quad \cdots \quad 0)^T$ or $\left[u^T \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}}\right]_{1,:} = ux^T$

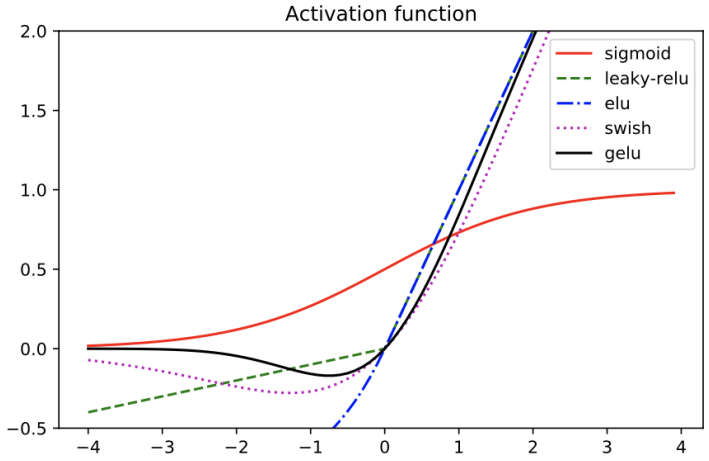
Dimension: $m \times n$

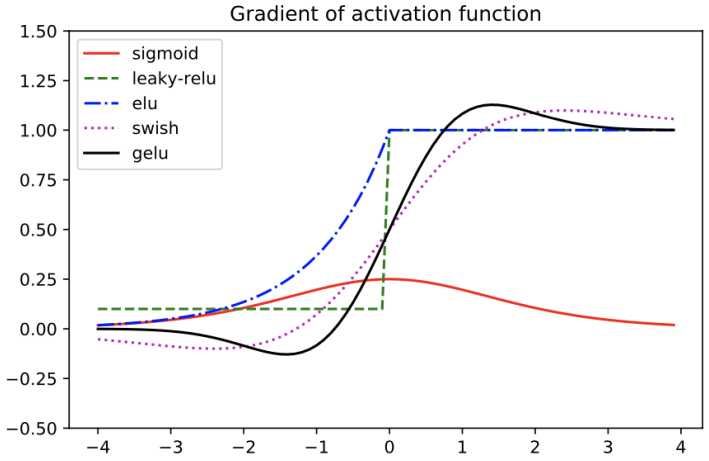
With index of $i$

# Training Neural Networks

- Tuning the learning rate → Gradient clipping: $\boldsymbol{g}' = \min\left(1, \frac{c}{\|\boldsymbol{g}\|}\right)\boldsymbol{g}$
- Vanishing and exploding gradients
  - Change activation functions to prevent the gradient from becoming too large or too small → Non-saturating activation functions
  - Modify architecture so that the updates are additive rather than multiplicative → Residual network
  - Modify architecture to standardize the activations at each layer → layer normalization
  - Carefully choose the initial values of the parameters → Xavier initialization

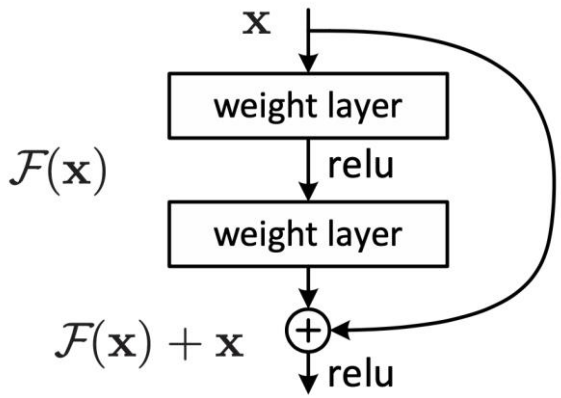
# Activation Functions

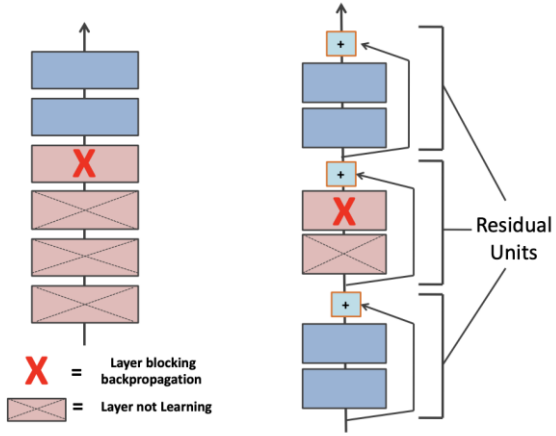



| Name | Definition | Range |
|---|---|---|
| Sigmoid | $\sigma(a) = \frac{1}{1+e^{-a}}$ | $[0, 1]$ |
| Hyperbolic tangent | $\tanh(a) = 2\sigma(2a) - 1$ | $[-1, 1]$ |
| Softplus | $\sigma_+(a) = \log(1 + e^a)$ | $[0, \infty]$ |
| Rectified linear unit | $\mathrm{ReLU}(a) = \max(a, 0)$ | $[0, \infty]$ |
| Leaky ReLU | $\max(a, 0) + \alpha \min(a, 0)$ | $[-\infty, \infty]$ |
| Exponential linear unit | $\max(a, 0) + \min(\alpha(e^a - 1), 0)$ | $[-\infty, \infty]$ |
| Swish | $a\sigma(a)$ | $[-\infty, \infty]$ |
| GELU | $a\Phi(a)$ | $[-\infty, \infty]$ |

# Residual Networks $\mathcal{F}_l'(x) = \mathcal{F}_l(x) + x$



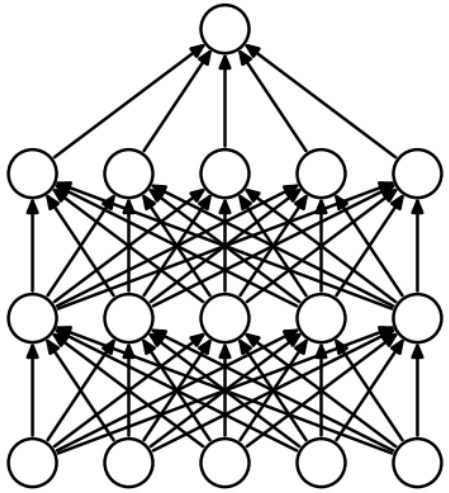

$$z_L = z_l + \sum_{i=l}^{L-1} \mathcal{F}_i(z_i; \theta_i)$$

$$\frac{\partial \mathcal{L}}{\partial \theta_l} = \frac{\partial \mathcal{L}}{\partial z_l}\frac{\partial z_l}{\partial \theta_l} = \frac{\partial \mathcal{L}}{\partial z_L}\frac{\partial z_L}{\partial z_l}\frac{\partial z_l}{\partial \theta_l} = \frac{\partial \mathcal{L}}{\partial z_L}\left(1 + \sum_{i=l}^{L-1} \frac{\partial \mathcal{F}_i(z_i; \theta_i)}{\partial z_l}\right)\frac{\partial z_l}{\partial \theta_l} = \frac{\partial \mathcal{L}}{\partial z_L}\frac{\partial z_l}{\partial \theta_l} + other\_terms$$

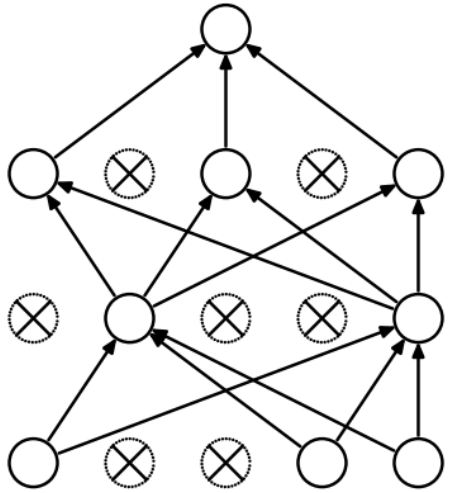
# Normalizations

- **Batch normalization** ensures the distribution of the activations within **a layer** has **zero mean** and **unit variance**, when **averaged across the samples in a minibatch**
- **Layer Normalization** directly estimates the normalization statistics from the **summed inputs to the neurons within a hidden layer** so the normalization does not introduce any new dependencies between training cases

# Dropout: An implicit Ensemble of Neural Networks



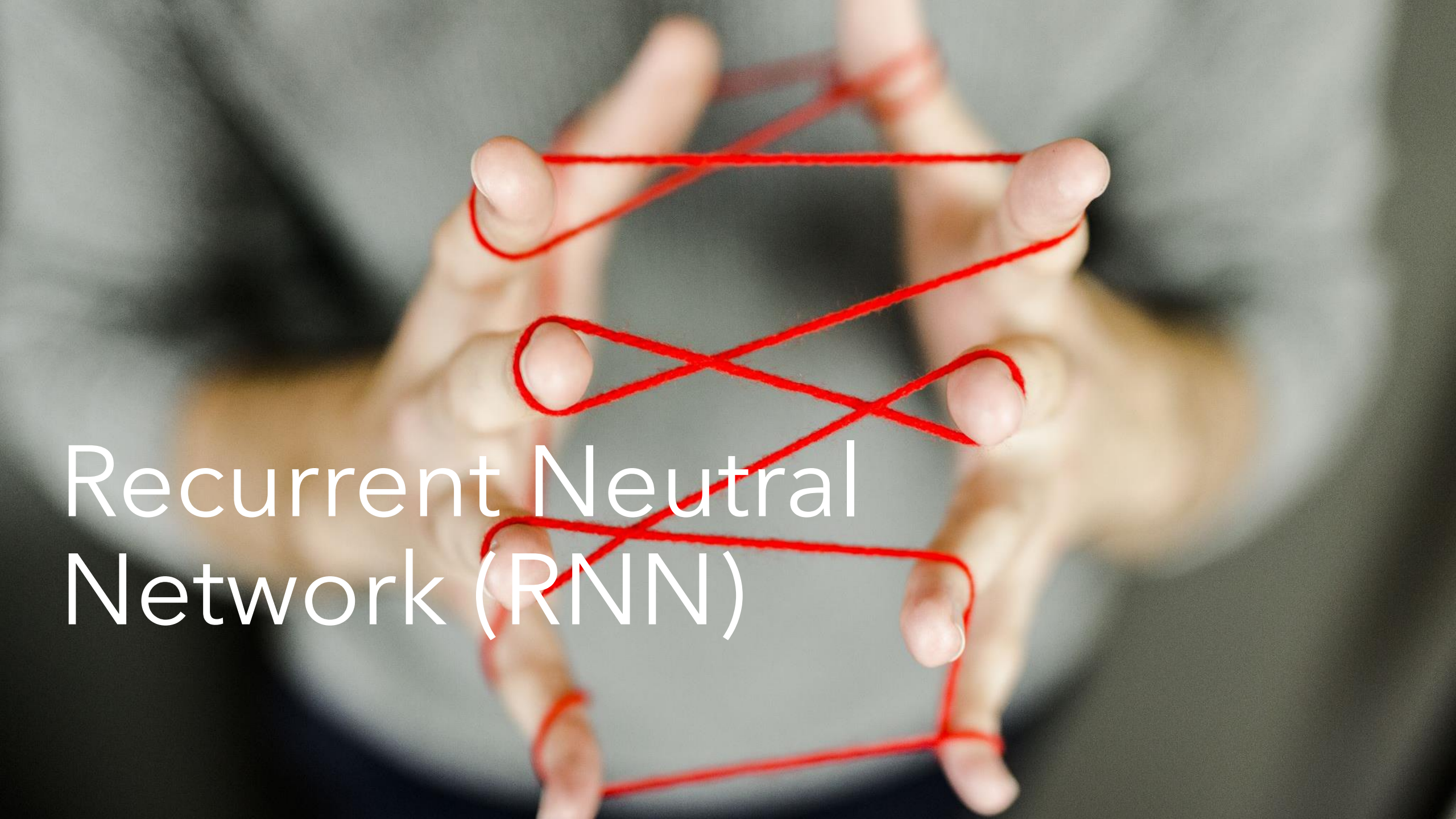
(a) Standard Neural Net



(b) After applying dropout.

- During training, randomly turning on some unit in each layer with a $Ber(1-p)$, where $p$ is the **dropping probability**
- At test time, we turn on all the units, but the weights should be reweighted by multiplying the $1-p$, the variance of the Bernoulli distribution, also the **keep probability**

It prevents complex **co-adaptation** of the hidden units, such that each unit must learn to perform well even if some of the other units are missing at random.



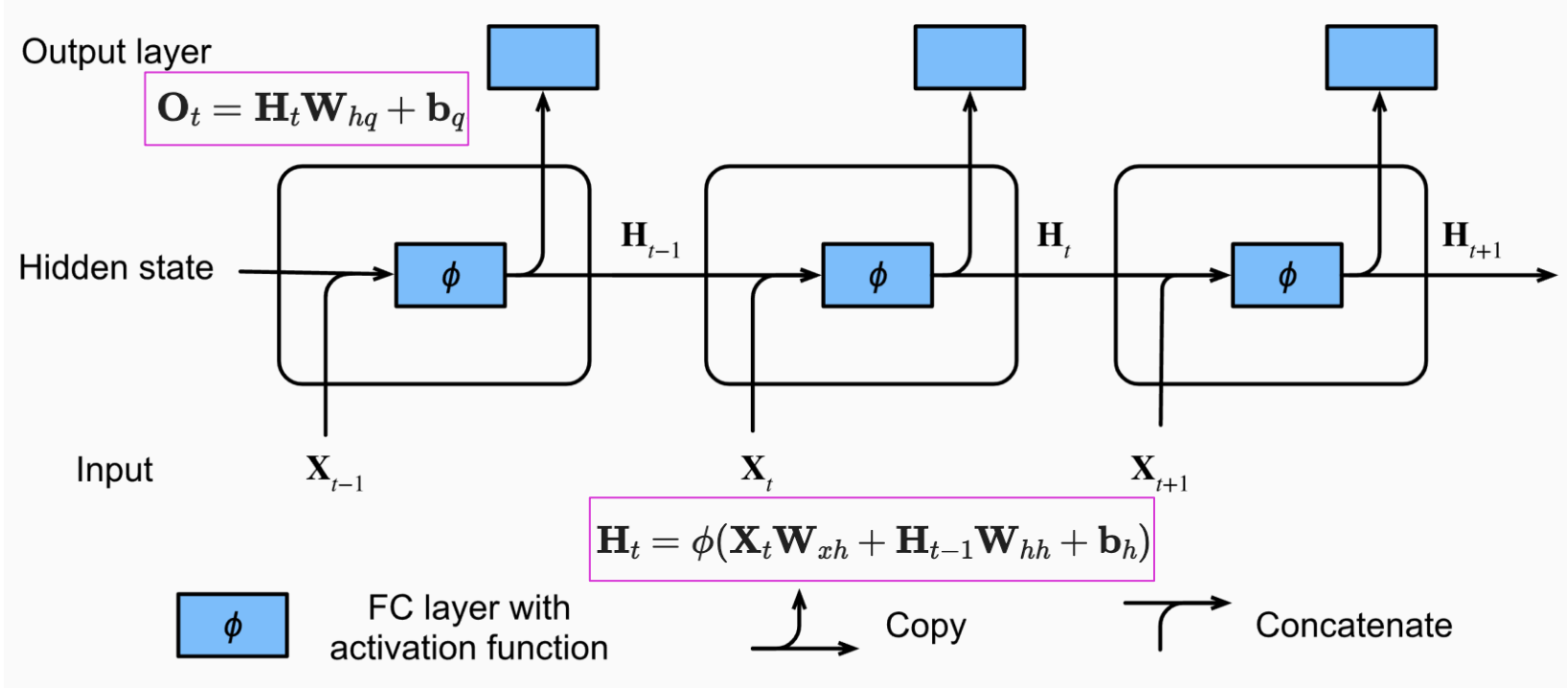
# Recurrent Neutral Network (RNN)

# When do we need an RNN?

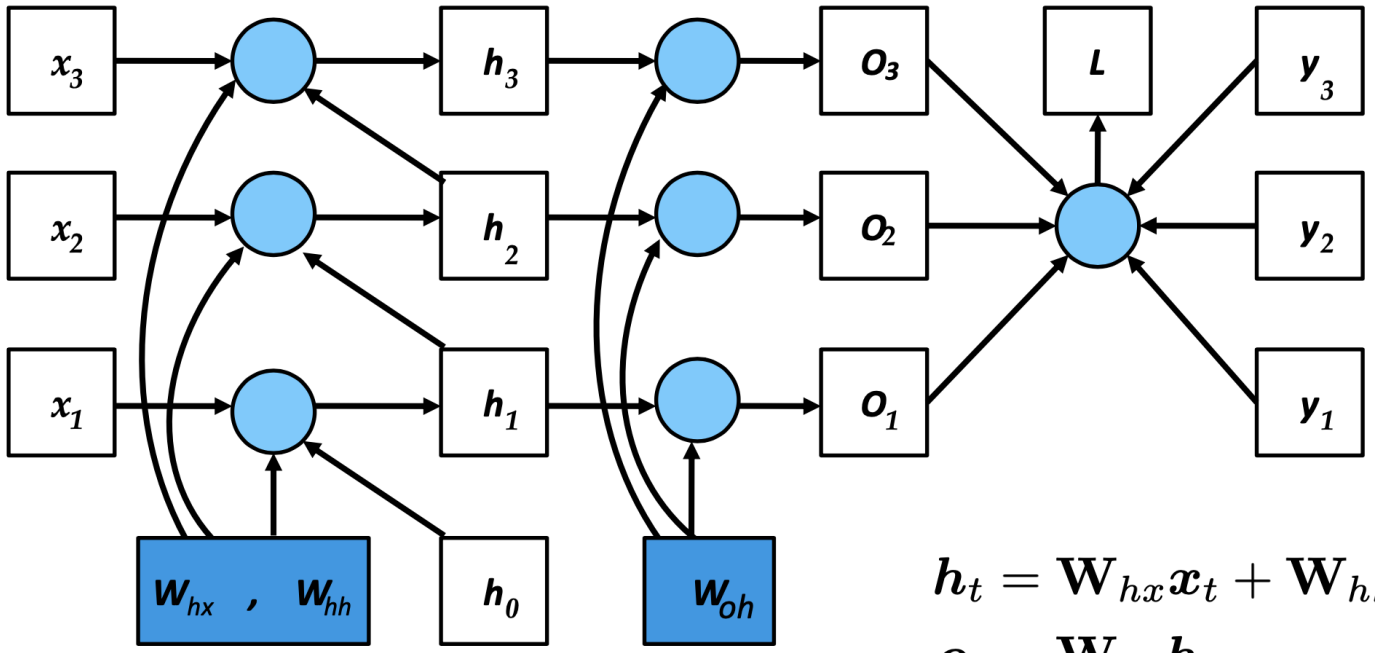
"Whenever there is a **sequence** of data and that **temporal dynamics** that connects the data is **more important than the spatial content** of each individual frame."

– Lex Fridman (MIT)

# An RNN with Hidden States



# An RNN Unrolled (vertically) for 3 Time Steps

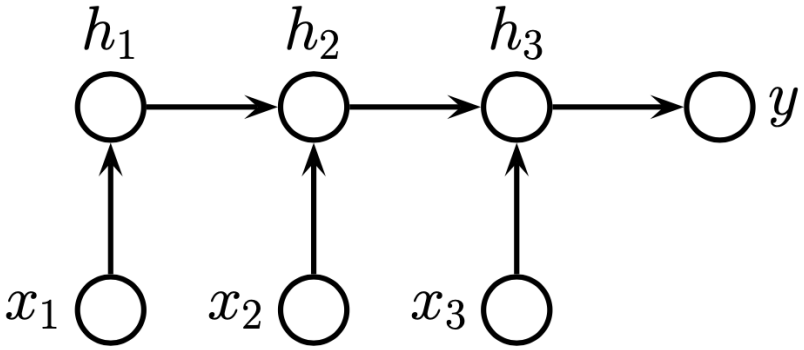



$$\boldsymbol{h}_t = \mathbf{W}_{hx}\boldsymbol{x}_t + \mathbf{W}_{hh}\boldsymbol{h}_{t-1}$$

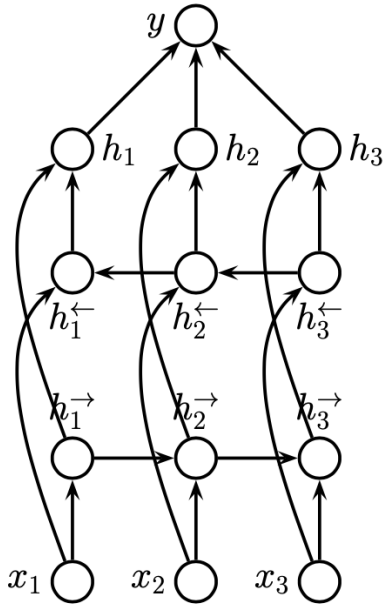
$$\boldsymbol{o}_t = \mathbf{W}_{ho}\boldsymbol{h}_t$$

$$L = \frac{1}{T}\sum_{t=1}^{T} \ell(y_t, \boldsymbol{o}_t)$$

# RNN for Sequence Classification

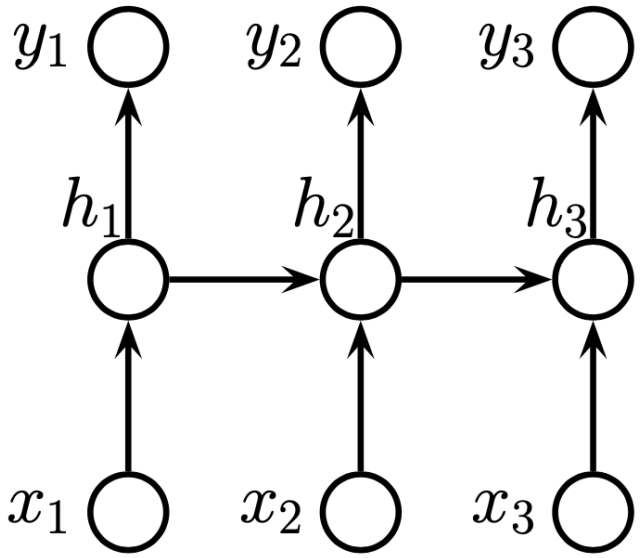



RNN for sequence classification

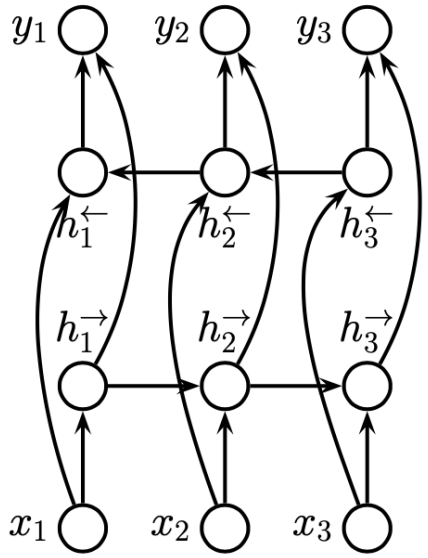



Bi-directional RNN for sequence classification

# Seq2seq (Aligned Case)

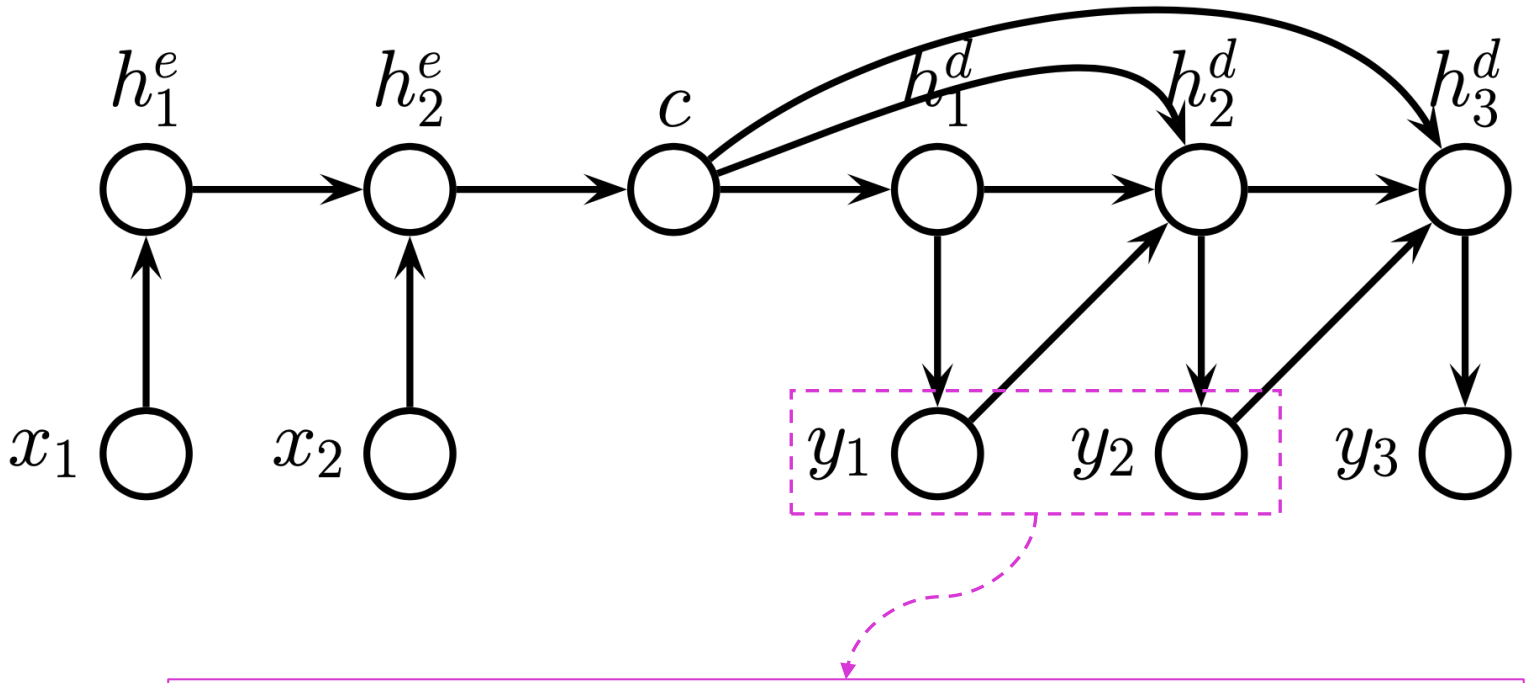



RNN for transforming a sequence to another



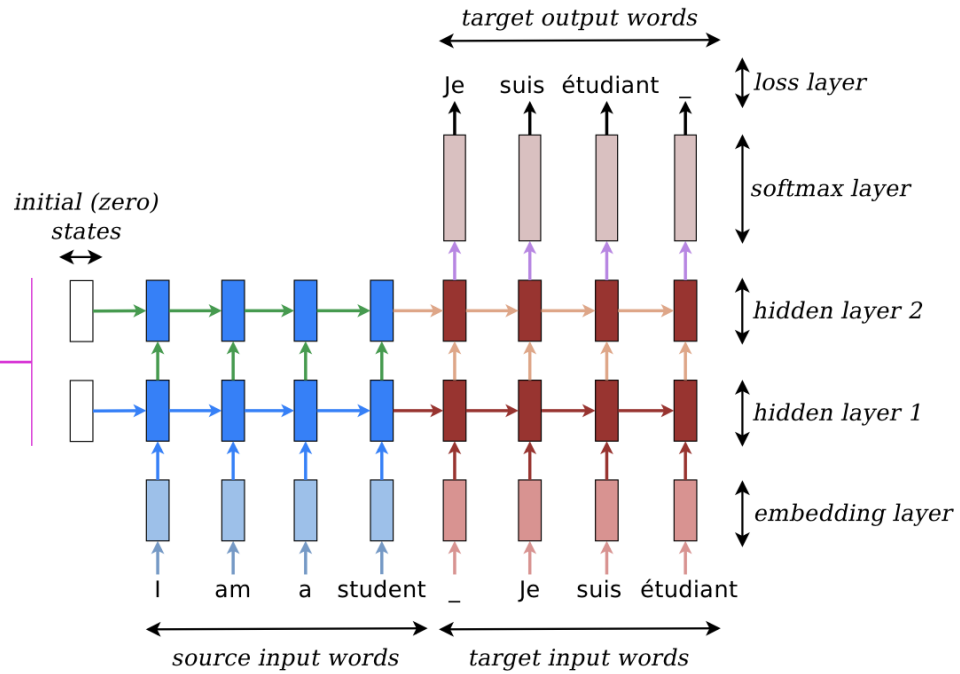

Bi-directional RNN for the same task

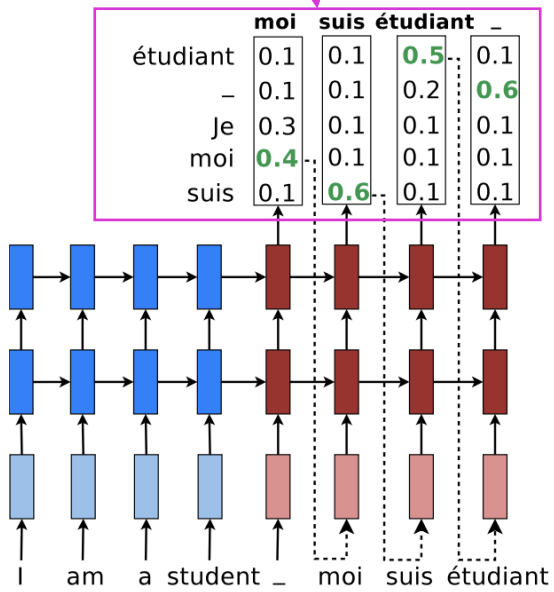
# Seq2seq (Unaligned Case)



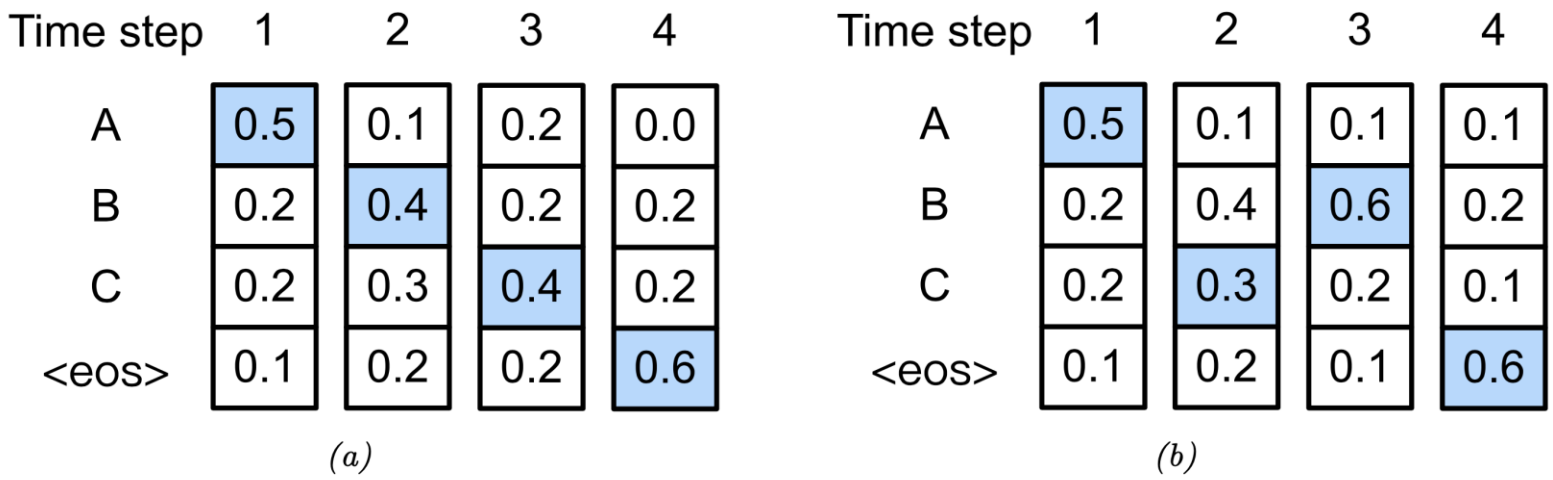
# Seq2seq With Greedy Decoding

Deep RNN



# Issue of Greedy Decoding

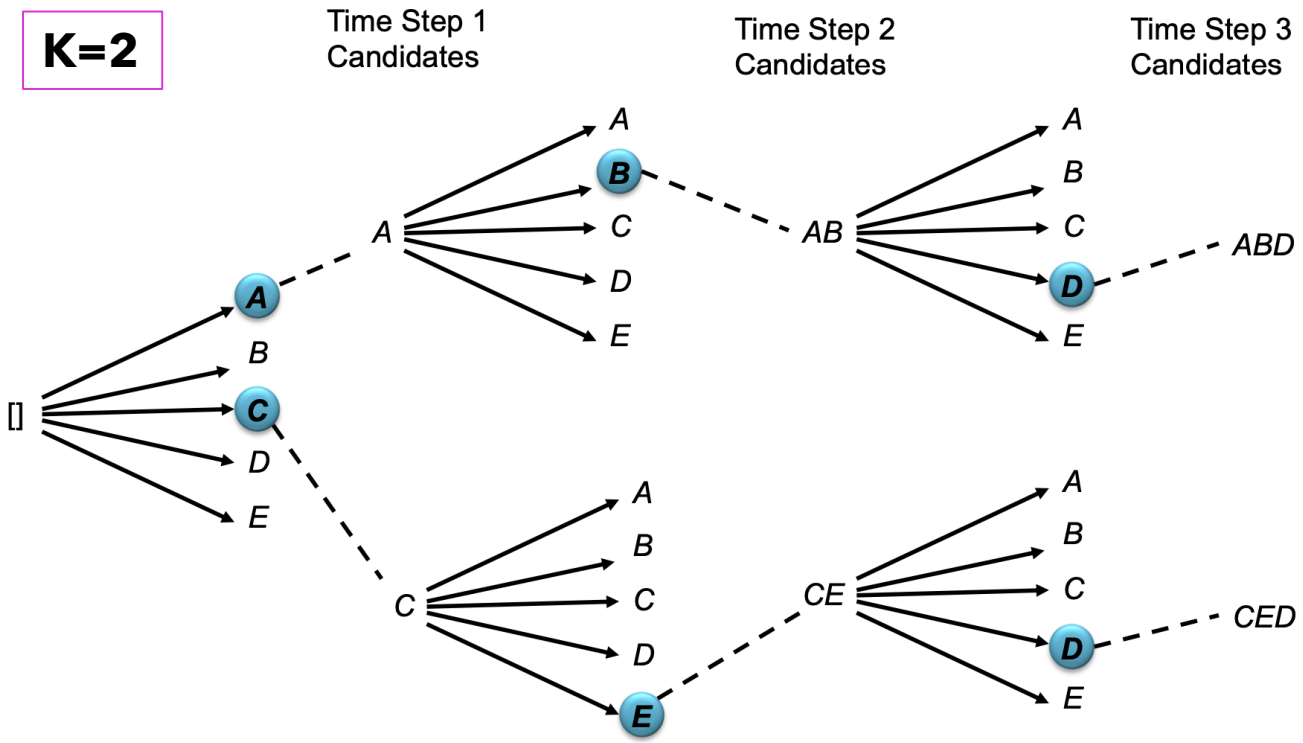



Greedy search: P(seq)=0.048

Not greedy decoding in step 2: P(seq)=0.054

The reason is that the locally optimal symbol at step **t** might not be on the **globally optimal** path

# Beam Search




1. Compute the **top K** candidate outputs at each step
2. Expand each one in **all V possible ways**, to generate **V** × **K** candidates, from which we pick the **top K** again
3. Repeat step 1, 2 until the end

# Backpropagation Through Time (BPTT)

- Consider the following model

$$\boldsymbol{h}_t = \mathbf{W}_{hx}\boldsymbol{x}_t + \mathbf{W}_{hh}\boldsymbol{h}_{t-1} \quad \rightarrow \quad \boldsymbol{h}_t = f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{w}_h)$$

$$\boldsymbol{o}_t = \mathbf{W}_{ho}\boldsymbol{h}_t \qquad\qquad \boldsymbol{o}_t = g(\boldsymbol{h}_t, \boldsymbol{w}_o)$$

$$L = \frac{1}{T}\sum_{t=1}^{T} \ell(y_t, \boldsymbol{o}_t)$$

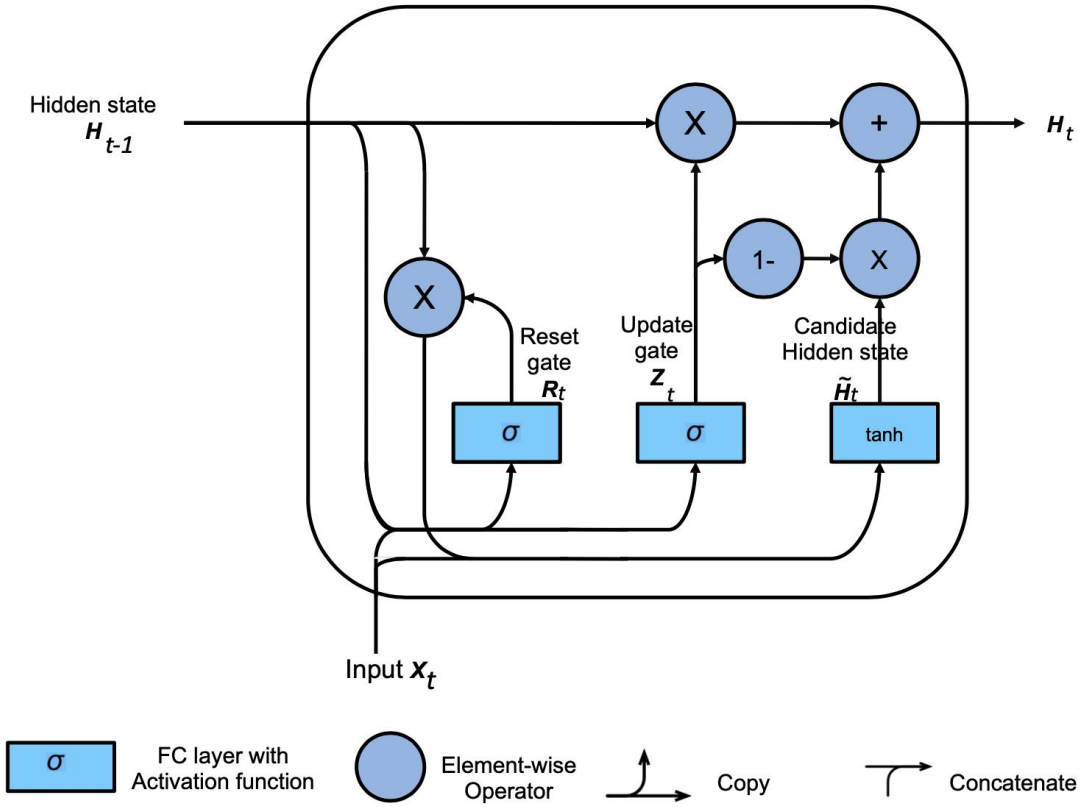
Vanishing & exploding gradients

$$\frac{\partial L}{\partial \boldsymbol{w}_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial \ell(y_t, \boldsymbol{o}_t)}{\partial \boldsymbol{w}_h} = \frac{1}{T}\sum_{t=1}^{T} \frac{\partial \ell(y_t, \boldsymbol{o}_t)}{\partial \boldsymbol{o}_t} \frac{\partial g(\boldsymbol{h}_t, \boldsymbol{w}_o)}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{w}_h}$$

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{w}_h} = \frac{\partial f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{w}_h)}{\partial \boldsymbol{w}_h} + \frac{\partial f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{w}_h)}{\partial \boldsymbol{h}_{t-1}} \frac{\partial \boldsymbol{h}_{t-1}}{\partial \boldsymbol{w}_h}$$

$$\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{w}_h} = \frac{\partial f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}, \boldsymbol{w}_h)}{\partial \boldsymbol{w}_h} + \sum_{i=1}^{t-1} \left( \prod_{j=i+1}^{t} \frac{\partial f(\boldsymbol{x}_j, \boldsymbol{h}_{j-1}, \boldsymbol{w}_h)}{\partial \boldsymbol{h}_{j-1}} \right) \frac{\partial f(\boldsymbol{x}_i, \boldsymbol{h}_{i-1}, \boldsymbol{w}_h)}{\partial \boldsymbol{w}_h}$$

# Gated Recurrent Units (GRU)

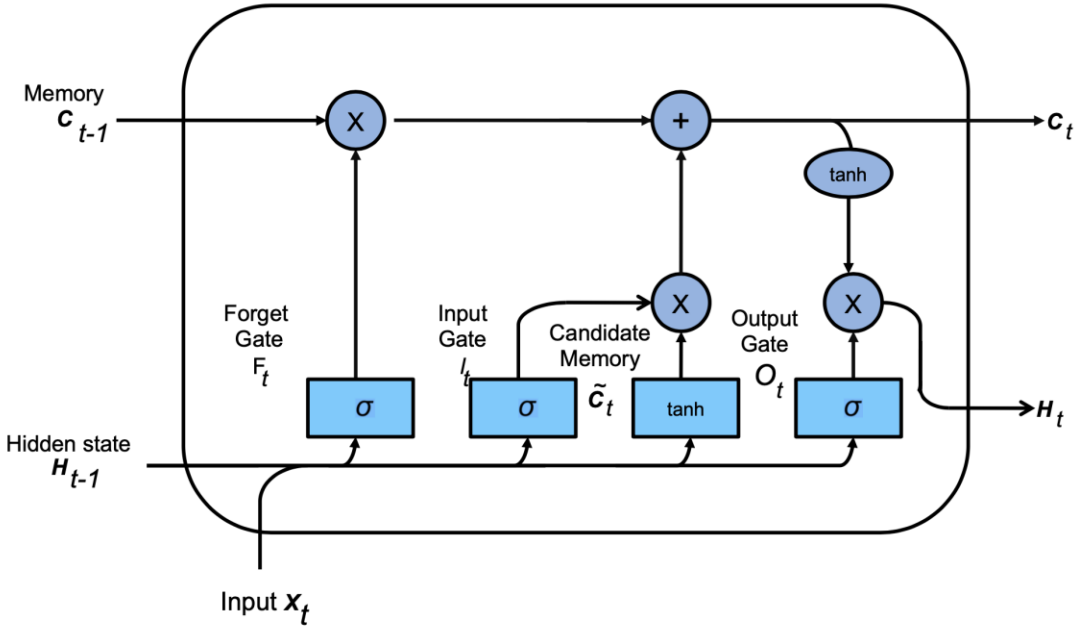


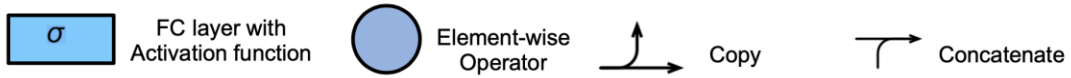

$$\mathbf{R}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xr} + \mathbf{H}_{t-1} \mathbf{W}_{hr} + \boldsymbol{b}_r)$$
$$\mathbf{Z}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xz} + \mathbf{H}_{t-1} \mathbf{W}_{hz} + \boldsymbol{b}_z)$$

$$\tilde{\mathbf{H}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xh} + (\mathbf{R}_t \odot \mathbf{H}_{t-1}) \mathbf{W}_{hh} + \boldsymbol{b}_h)$$

$$\mathbf{H}_t = \mathbf{Z}_t \odot \mathbf{H}_{t-1} + (1 - \mathbf{Z}_t) \odot \tilde{\mathbf{H}}_t$$

# Long Short Term Memory (LSTM) Model




$$\mathbf{O}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xo} + \mathbf{H}_{t-1} \mathbf{W}_{ho} + \boldsymbol{b}_o)$$
$$\mathbf{I}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xi} + \mathbf{H}_{t-1} \mathbf{W}_{hi} + \boldsymbol{b}_i)$$
$$\mathbf{F}_t = \sigma(\mathbf{X}_t \mathbf{W}_{xf} + \mathbf{H}_{t-1} \mathbf{W}_{hf} + \boldsymbol{b}_f)$$

$$\tilde{\mathbf{C}}_t = \tanh(\mathbf{X}_t \mathbf{W}_{xc} + \mathbf{H}_{t-1} \mathbf{W}_{hc} + \boldsymbol{b}_c)$$

$$\mathbf{C}_t = \mathbf{F}_t \odot \mathbf{C}_{t-1} + \mathbf{I}_t \odot \tilde{\mathbf{C}}_t$$

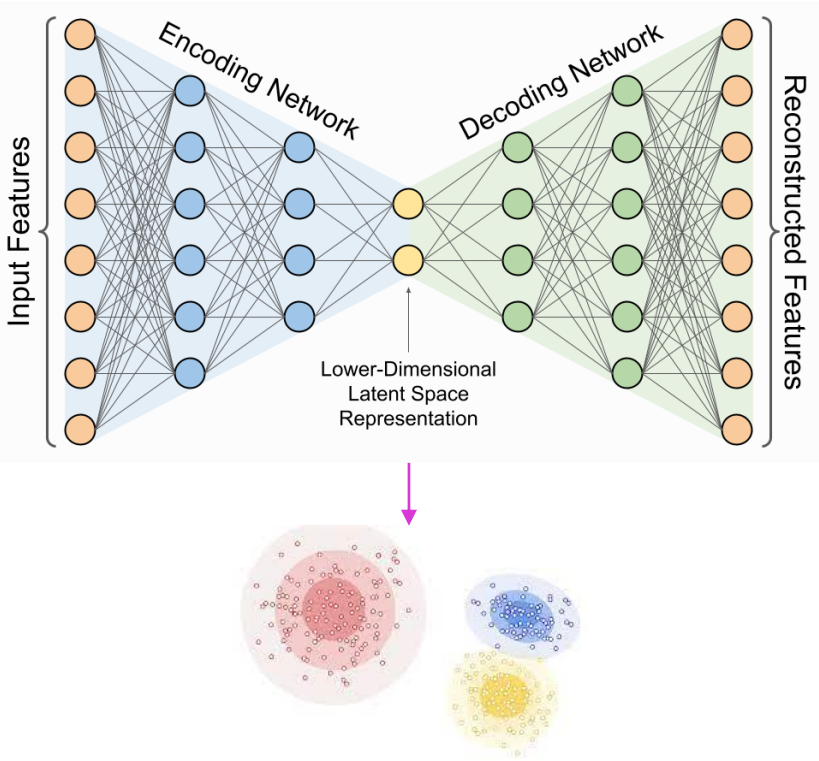
$$\mathbf{H}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}_t)$$

# Patient Subtyping via Time-Aware LSTM Networks (T-LSTM)

- In the study of various diseases, **heterogeneity** among patients usually leads to different progression patterns and may require different types of therapeutic intervention
- It is important to study patient subtyping, which is grouping of patients into disease characterizing subtypes

Baytas, Inci M., et al. "Patient subtyping via time-aware LSTM networks." *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*. 2017.

# Idea: AE+Clustering



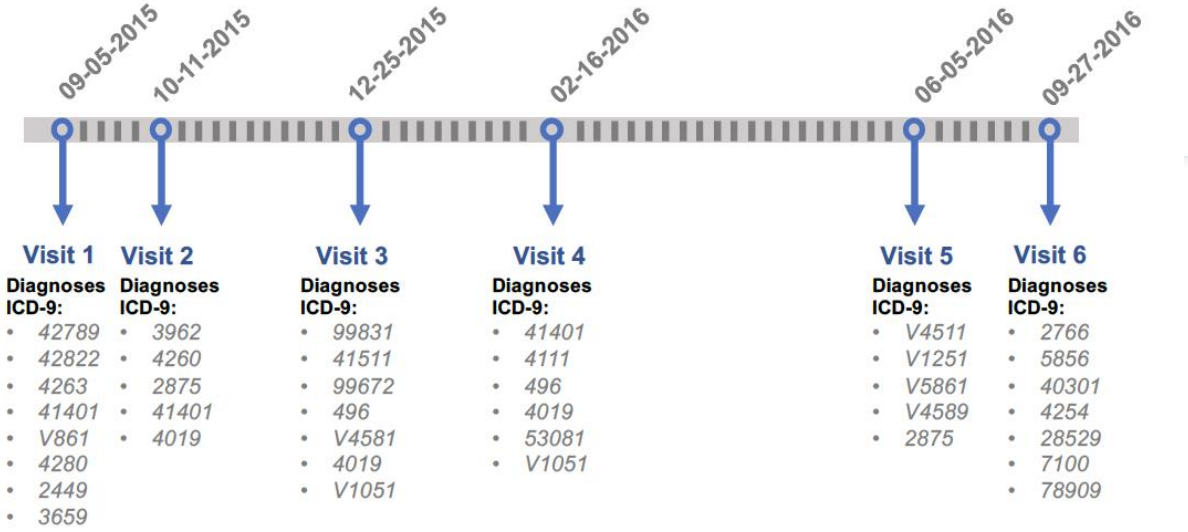




**Figure 1: An example segment of longitudinal patient records. The patient had 6 office visits between Sept 5, 2015 and Sept 27, 2016. In each visit, diagnosis of the patient was given by a set of ICD-9 codes. Time spans between two successive visits can vary, and may be months apart. Such time irregularity results in a significant challenge in patient subtyping.**

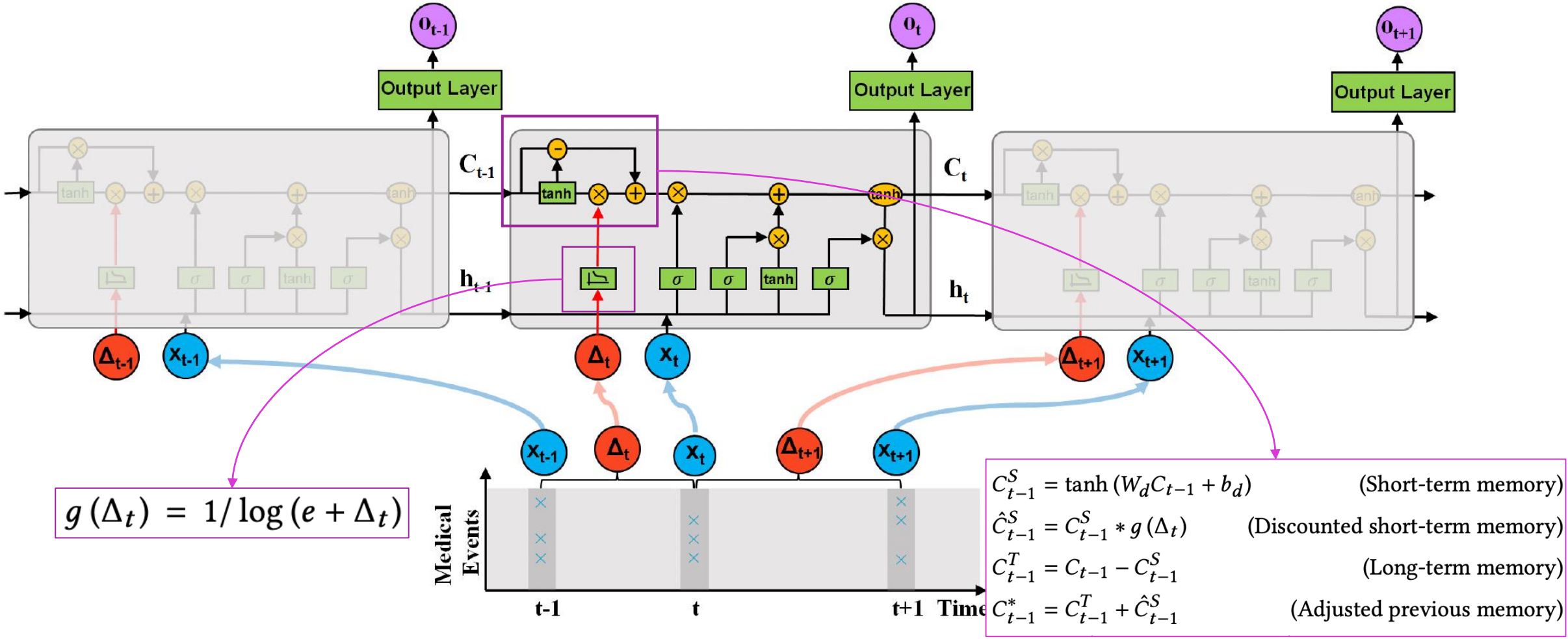



**Figure 2: Illustration of the proposed time-aware long-short term memory (T-LSTM) unit, and its application on analyzing healthcare records. Green boxes indicate networks and yellow circles denote point-wise operators. T-LSTM takes two inputs, input record and the elapsed time at the current time step. The time lapse between the records at time $t-1$, $t$ and $t+1$ can vary from days to years in healthcare domain. T-LSTM decomposes the previous memory into long and short term components and utilizes the elapsed time ($\Delta_t$) to discount the short term effects.**

# T-LSTM Auto-Encoder

$$E_r = \sum_{i=1}^{L} \left\| X_i - \hat{X}_i \right\|_2^2$$

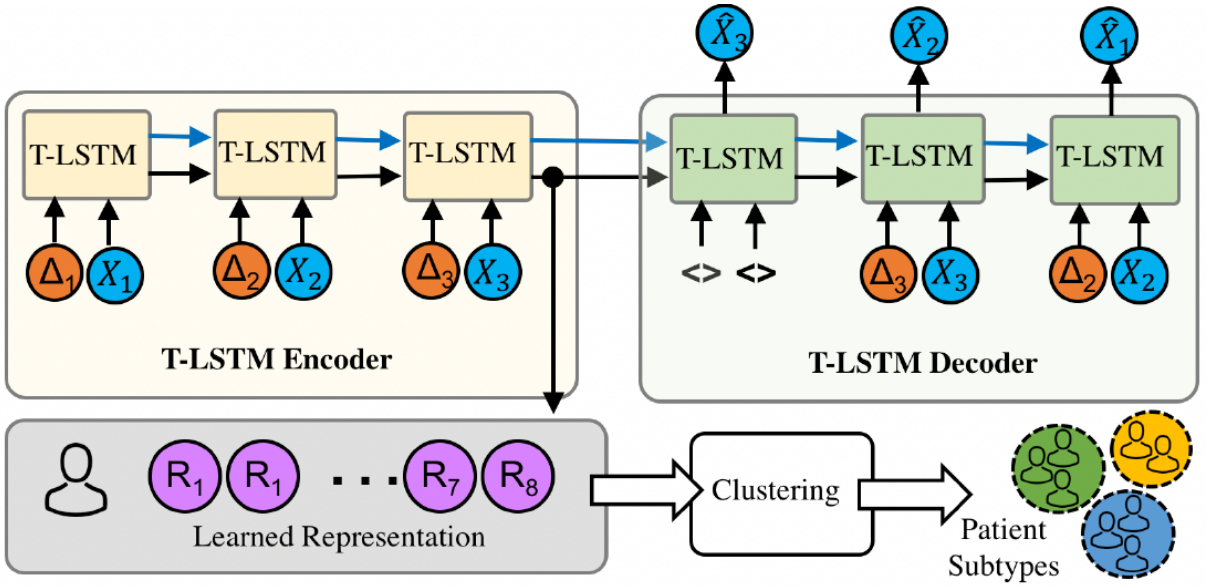


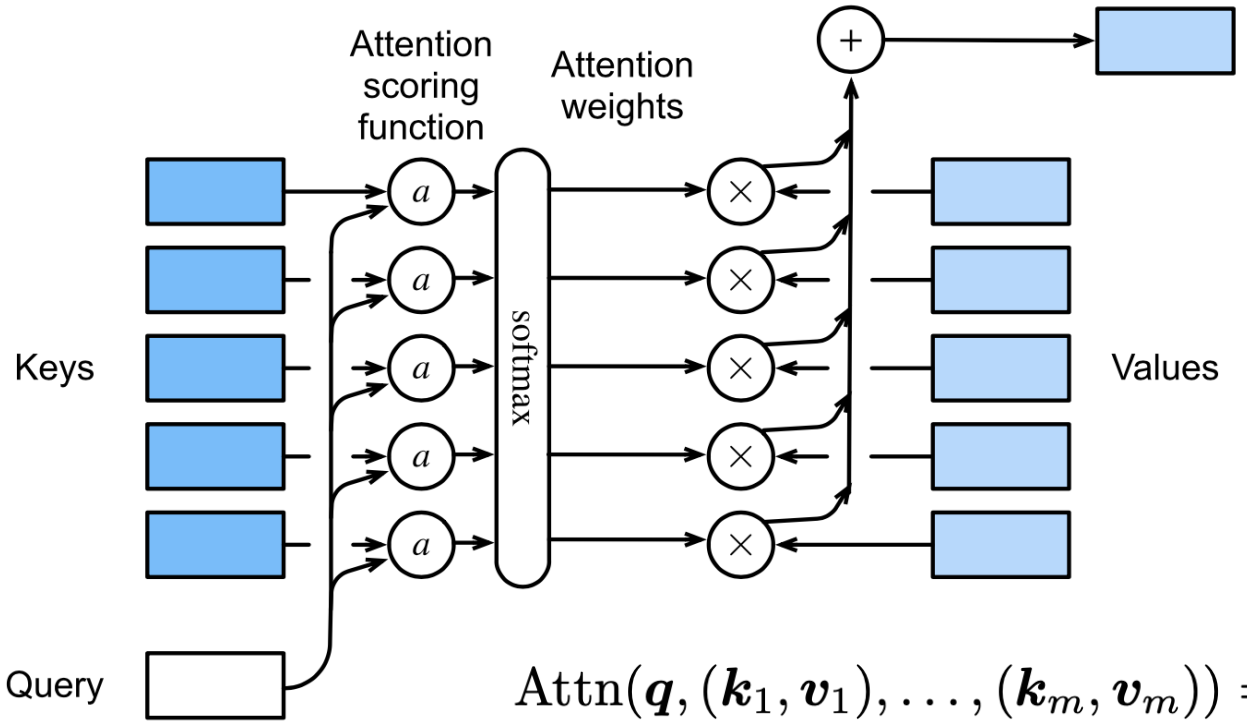

**Figure 3: Clustering patients with a single-layer T-LSTM Auto-Encoder. Blue arrows denote the cell memory and the black arrows denote the hidden states. After the representations ($R_i$, $i = 1, 2, \cdots, 8$) are learned for the population, we can cluster the patients and obtain subtypes for each group as the prominent common medical features of the group. Number of layers should be increased in case of dimensionality reduction to be able to capture more complex structure with fewer iterations compared to single layer.**

# Attention


- Bahdanau, Dzmitry, Kyung… d Yoshua Bengio. "Neural machine translati… learning to align and translate." *arXiv preprint a… 3 (2014).*

- Luong, Minh-Thang, Hieu … topher D. Manning. "Effective approaches to a… based machine translation." *arXiv preprint … 25 (2015).*

# Illustration of Attention




Masked Attention: restrict attention to valid inputs by setting their attention scores as a large negative number.

Attention computes a **weighted average** of a set of **values**, where the weights are derived by **comparing** the **query** vector to a set of **keys**.

$$\text{Attn}(\boldsymbol{q}, (\boldsymbol{k}_1, \boldsymbol{v}_1), \ldots, (\boldsymbol{k}_m, \boldsymbol{v}_m)) = \text{Attn}(\boldsymbol{q}, (\boldsymbol{k}_{1:m}, \boldsymbol{v}_{1:m})) = \sum_{i=1}^{m} \alpha_i(\boldsymbol{q}, \boldsymbol{k}_{1:m}) \boldsymbol{v}_i \in \mathbb{R}^v$$

Attention Score

$$\alpha_i(\boldsymbol{q}, \boldsymbol{k}_{1:m}) = \text{softmax}_i([a(\boldsymbol{q}, \boldsymbol{k}_1), \ldots, a(\boldsymbol{q}, \boldsymbol{k}_m)]) = \frac{\exp(a(\boldsymbol{q}, \boldsymbol{k}_i))}{\sum_{j=1}^{m} \exp(a(\boldsymbol{q}, \boldsymbol{k}_j))}$$

# Kernel Regression (Non-Parametric Attn)

$$f(x) = \sum_{i=1}^{n} \alpha_i(x, x_{1:n}) y_i$$

normalized similarity score of test input $x$ to training input $x_i$

Attn score function

$$\mathcal{K}_\sigma(u) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}u^2}$$

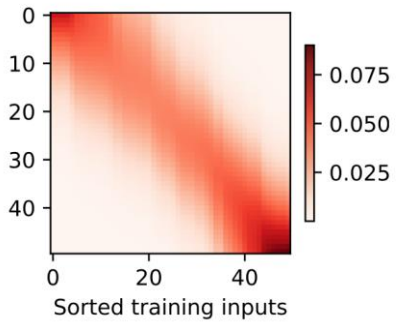
Bandwidth

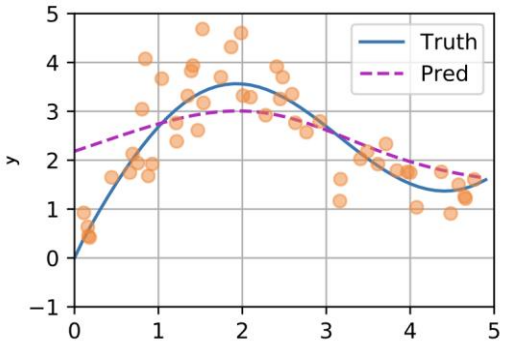
$$\mathcal{K}(u; w) = \exp(-\frac{w^2}{2}u^2)$$

$$a(x, x_i) = \mathcal{K}_\sigma(x - x_i)$$

$$\begin{aligned} f(x) &= \sum_{i=1}^{n} \alpha_i(x, x_{1:n}) y_i \\ &= \sum_{i=1}^{n} \frac{\exp[-\frac{1}{2}((x - x_i)w)^2]}{\sum_{j=1}^{n} \exp[-\frac{1}{2}((x - x_j)w)^2]} y_i \\ &= \sum_{i=1}^{n} \text{softmax}_i \left[-\frac{1}{2}((x - x_1)w)^2, \cdots, -\frac{1}{2}((x - x_n)w)^2\right] y_i \end{aligned}$$

query keys values



# Parametric Attention

$\boldsymbol{k} \in \mathbb{R}^k \quad \mathbf{W}_k \in \mathbb{R}^{h \times k}$

$\boldsymbol{q} \in \mathbb{R}^q \quad \mathbf{W}_q \in \mathbb{R}^{h \times q}$

Additive Attn

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{w}_v^\top \tanh(\mathbf{W}_q \boldsymbol{q} + \mathbf{W}_k \boldsymbol{k}) \in \mathbb{R}$$

set $k = q = d$

$$a(\boldsymbol{q}, \boldsymbol{k}) = \boldsymbol{q}^\top \boldsymbol{k} / \sqrt{d} \in \mathbb{R}$$

Scaled Dot-product Attn

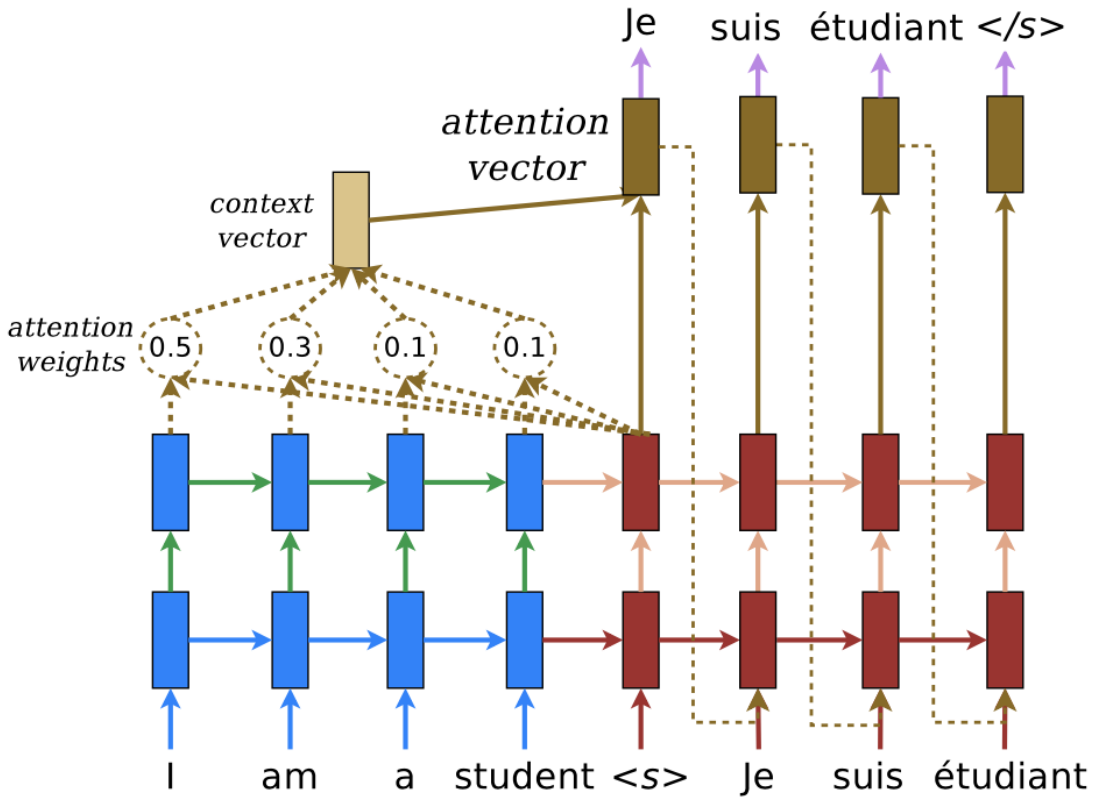
minibatches

$$\mathbf{Q} \in \mathbb{R}^{n \times d}, \mathbf{K} \in \mathbb{R}^{m \times d}, \mathbf{V} \in \mathbb{R}^{m \times v}$$

$$\mathrm{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}})\mathbf{V} \in \mathbb{R}^{n \times v}$$

Row-wise

# Seq2seq with Attention




In a standard Seq2seq, the context vector $\boldsymbol{C}$ in decoder RNN represents the encoding of the input. It is usually the final state of the encoder RNN → $\boldsymbol{C}$ is the bottleneck

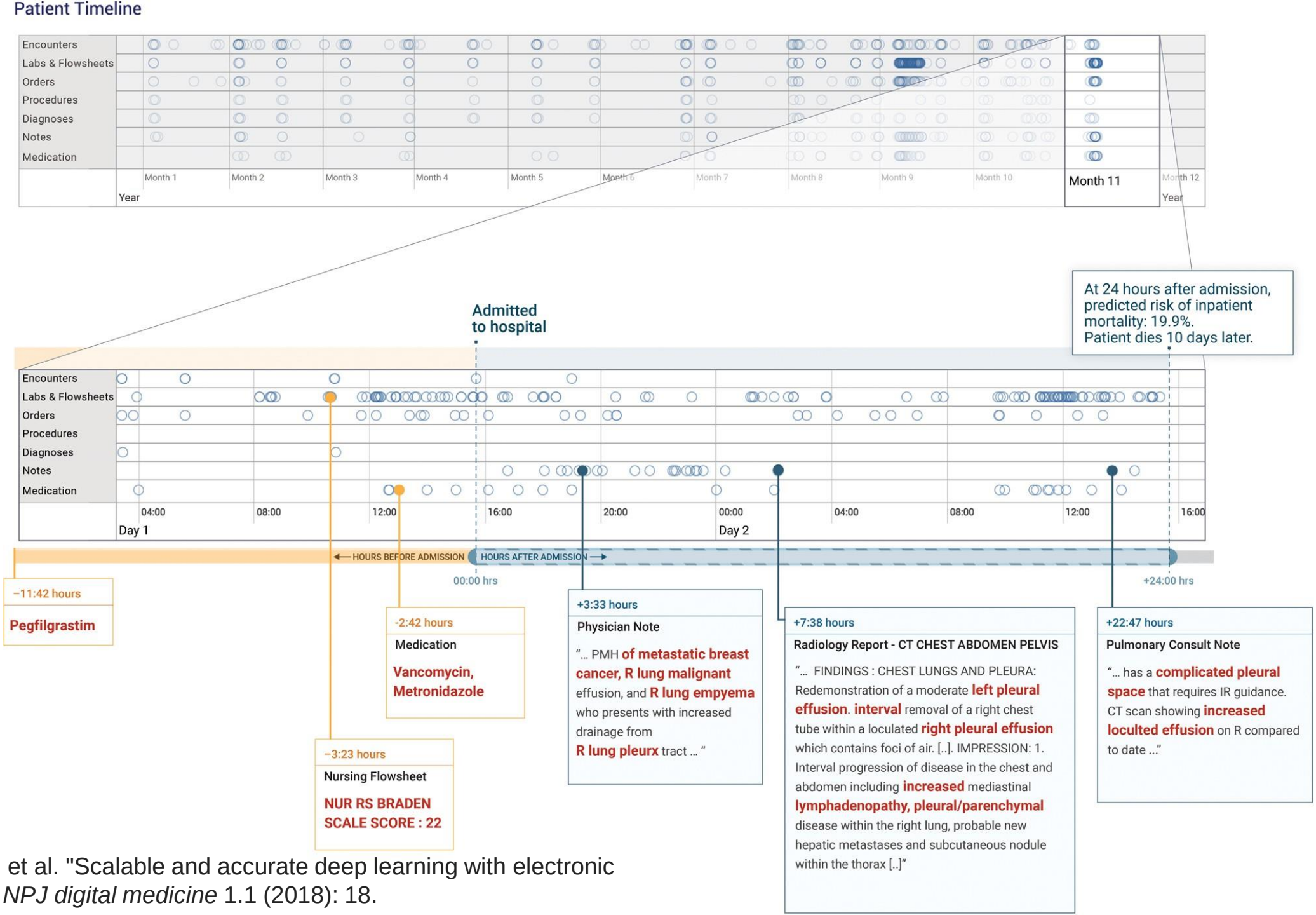
with attn

$$\boldsymbol{c}_t = \sum_{i=1}^{T} \alpha_i(\boldsymbol{h}_{t-1}^d, \boldsymbol{h}_{1:T}^e)\boldsymbol{h}_i^e$$

***Note***: *the figure only illustrates the attention at step $t = 1$ in the decoder RNN. The attention should be applied at each step $t$.*

# Soft vs Hard Attention

- Soft Attention: each output attends to a weighted combination of all the input locations.
- Hard attention: each input can only attend to one input location




Rajkomar, Alvin, et al. "Scalable and accurate deep learning with electronic health records." *NPJ digital medicine* 1.1 (2018): 18.

# Transformers

# Self-Attention

- The RNN decoder in Seq2seq uses attention to the input sequence in order to capture contextual embeddings of each input
- We can modify the model so the encoder attends to itself, leading to **self attention**.

$$\boldsymbol{y}_i = \text{Attn}(\boldsymbol{x}_i, (\boldsymbol{x}_1, \boldsymbol{x}_1), \ldots, (\boldsymbol{x}_n, \boldsymbol{x}_n))$$

The query is $x_i$, and the keys and values are all the (valid) inputs $x_1, x_2, \cdots, x_n$

In a decoder, we set $x_i = y_{i-1}$ and $n = i - 1$ so all the previously generated outputs are available

# Self-Attention

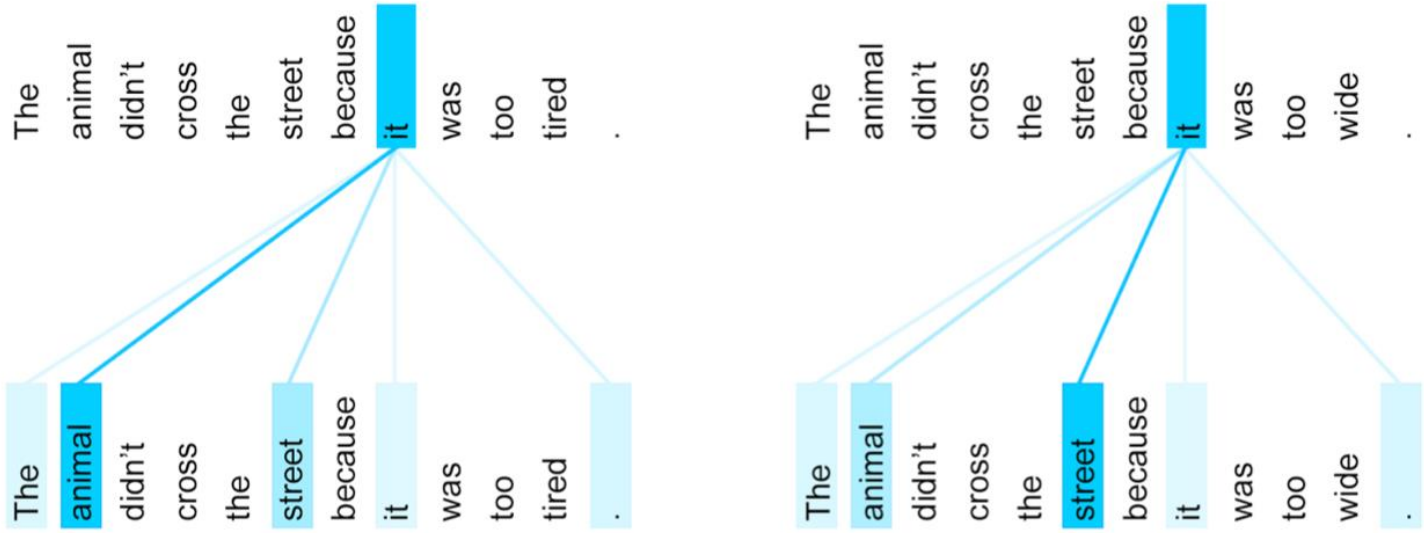



Illustration of how encoder self-attention for the word "it" differs depending on the input context.

coreference resolution

# Multi-Headed Attention

- Use multiple attention matrices to capture different notions of similarity

$$\boldsymbol{q} \in \mathbb{R}^{d_q} \quad \boldsymbol{k}_j \in \mathbb{R}^{d_k} \quad \boldsymbol{v}_j \in \mathbb{R}^{d_v}$$

$$\mathbf{W}_i^{(q)} \in \mathbb{R}^{p_q \times d_q}, \mathbf{W}_i^{(k)} \in \mathbb{R}^{p_k \times d_k}, \mathbf{W}_i^{(v)} \in \mathbb{R}^{p_v \times d_v}$$

$$\boldsymbol{h}_i = \text{Attn}(\mathbf{W}_i^{(q)}\boldsymbol{q}, \{\mathbf{W}_i^{(k)}\boldsymbol{k}_j, \mathbf{W}_i^{(v)}\boldsymbol{v}_j\}) \in \mathbb{R}^{p_v}$$

The $i^{th}$ attention head

$$\mathbf{W}_o \in \mathbb{R}^{p_o \times hp_v}$$

$$\boldsymbol{h} = \text{MHA}(\boldsymbol{q}, \{\boldsymbol{k}_j, \boldsymbol{v}_j\}) = \mathbf{W}_o \begin{pmatrix} \boldsymbol{h}_1 \\ \vdots \\ \boldsymbol{h}_h \end{pmatrix} \in \mathbb{R}^{p_o}$$

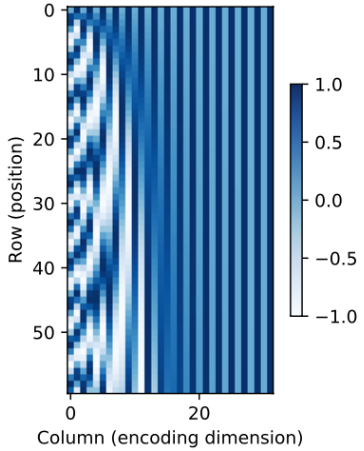
If we set $p_q h = p_k h = p_v h = p_o$, we can compute all the output heads in parallel.

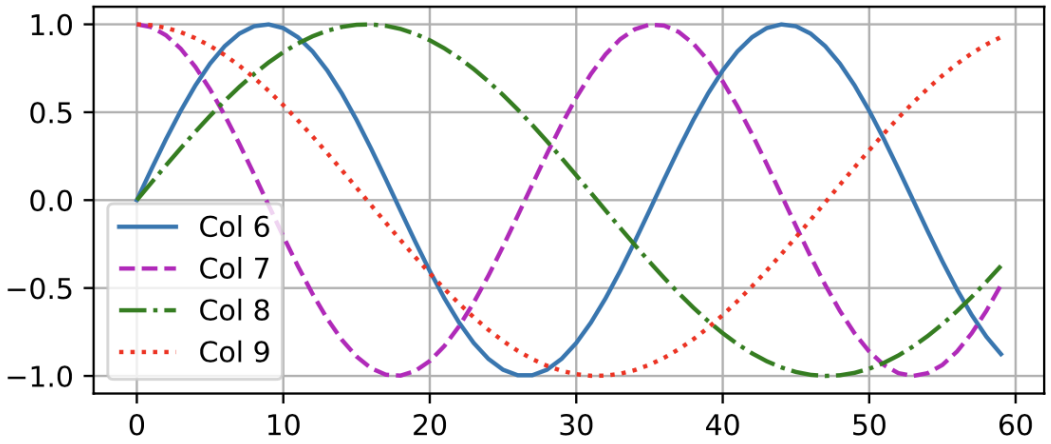
# Positional Encoding

$$p_{i,2j} = \sin\left(\frac{i}{C^{2j/d}}\right), \; p_{i,2j+1} = \cos\left(\frac{i}{C^{2j/d}}\right)$$

sinusoidal basis

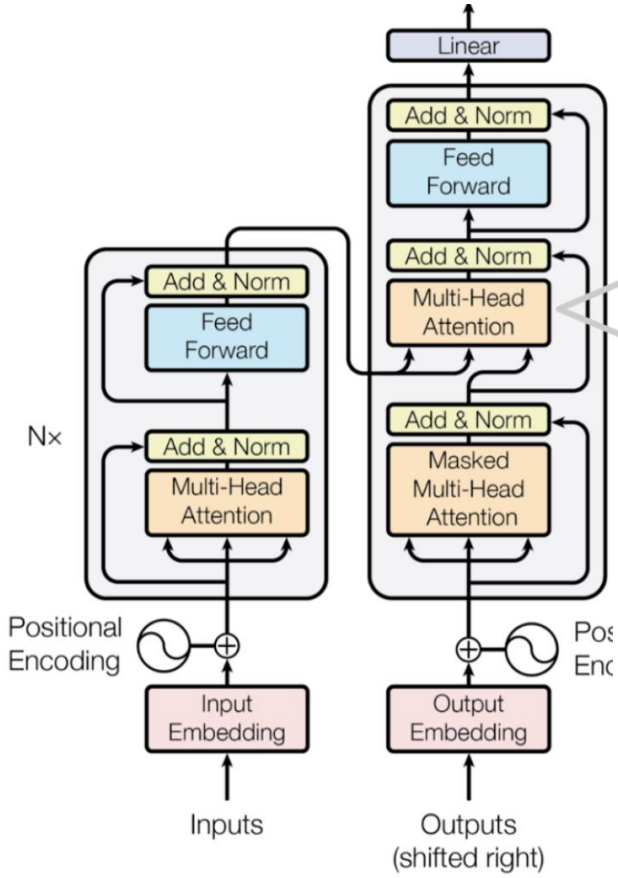
- The performance of "vanilla" self-attention can be low, since attention is **permutation invariant**, and hence ignores the input word ordering
- To overcome this, we can concatenate the word embeddings with a **positional embedding**, so that the model knows what order the words occur in.

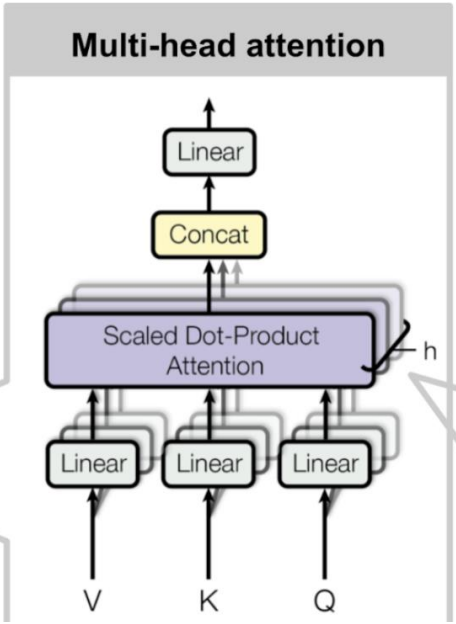


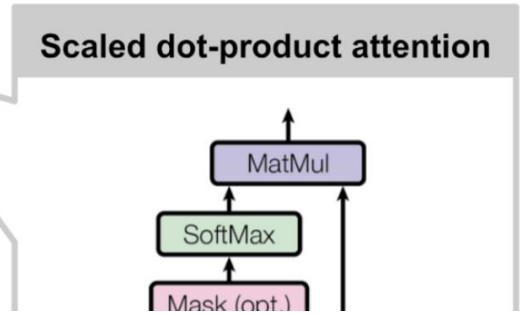



# Putting It All Together

```
def EncoderBlock(X):
  Z = LayerNorm(MultiHeadAttn(Q=X, K=X, V=X) + X)
  E = LayerNorm(FeedForward(Z) + Z)
  return E
def Encoder(X, N):
  E = POS(Embed(X))
  for n in range(N):
     E = EncoderBlock(E)
  return E
```



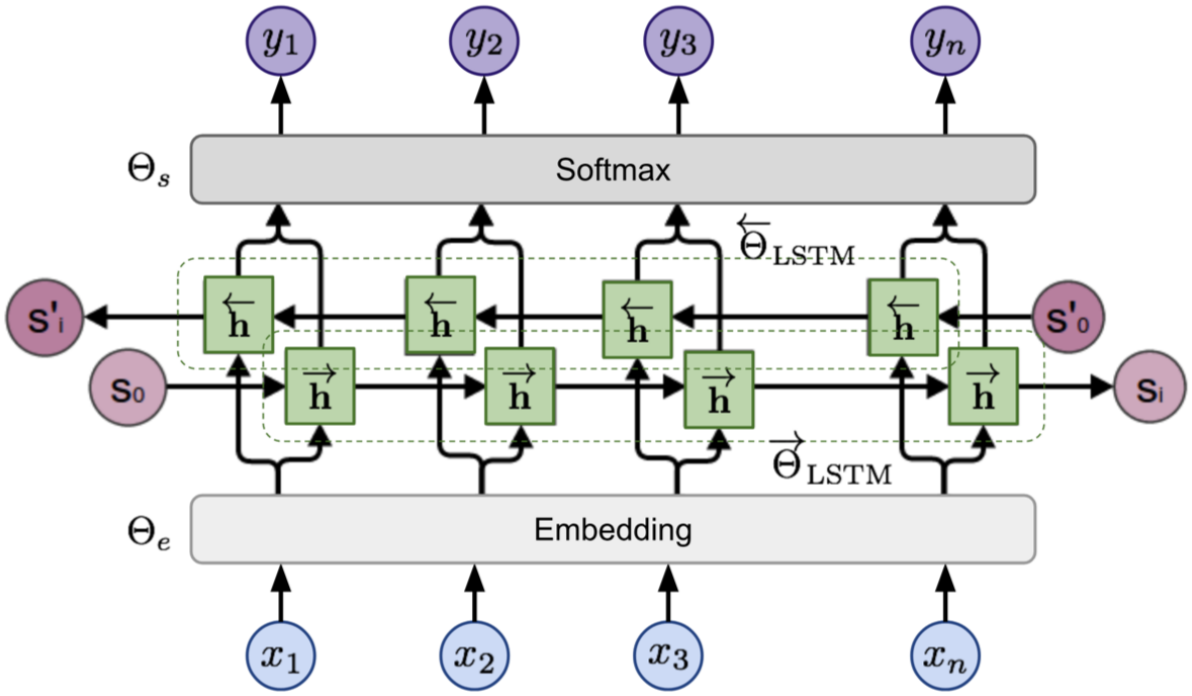

```
def DecoderBlock(Y, E):
  Z = LayerNorm(MultiHeadAttn(Q=Y, K=Y, V=Y) + Y)
  Z' = LayerNorm(MultiHeadAttn(Q=Z, K=E, V=E) + Z)
  D = LayerNorm(FeedForward(Z') + Z')
  return D
def Decoder(Y, E, N):
  D = POS(Embed(Y))
  for n in range(N):
     D = DecoderBlock(D,E)
  return D
```

Zoom-In!


Language Models and
Unsupervised Representation
Learning
$p(x_1, \ldots, x_T) = \prod_{t=1}^{T} p(x_t | x_{1:t-1})$

# Embeddings from Language Model (ELMo)

$$\mathcal{L}(\boldsymbol{\theta}) = -\sum_{t=1}^{T} [\log p(x_t | \boldsymbol{x}_{1:t-1}; \boldsymbol{\theta}_e, \boldsymbol{\theta}^{\rightarrow}, \boldsymbol{\theta}_s) + \log p(x_t | \boldsymbol{x}_{t+1:T}; \boldsymbol{\theta}_e, \boldsymbol{\theta}^{\leftarrow}, \boldsymbol{\theta}_s)]$$




After LM training, each token $x_k$ has

$$\begin{aligned} R_k &= \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} \mid j = 1, \ldots, L\} \\ &= \{\mathbf{h}_{k,j}^{LM} \mid j = 0, \ldots, L\}, \end{aligned}$$

For inclusion in a downstream model, ELMo collapses all layers in R into a single vector,

$$\mathbf{ELMo}_k = E(R_k; \boldsymbol{\Theta}_e)$$

For a specific downstream task,

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM}$$

# Bidirectional Encoder Representations from Transformers (BERT) → Two Tasks

A. Masked Language Model Task

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\mathbb{E}_{\boldsymbol{m}} \sum_{i\in\boldsymbol{m}} -\log p(x_i|\boldsymbol{x}_{-\boldsymbol{m}})$$

A random binary mask

Given a randomly chosen token (15%),
- 80% of the time, replace it with [mask] token
- 10% of the time, replace it with a random word
- 10% of the time, keep the word unchanged

`Let's make [MASK] chicken! [SEP] It [MASK] great with orange sauce.`

"some"

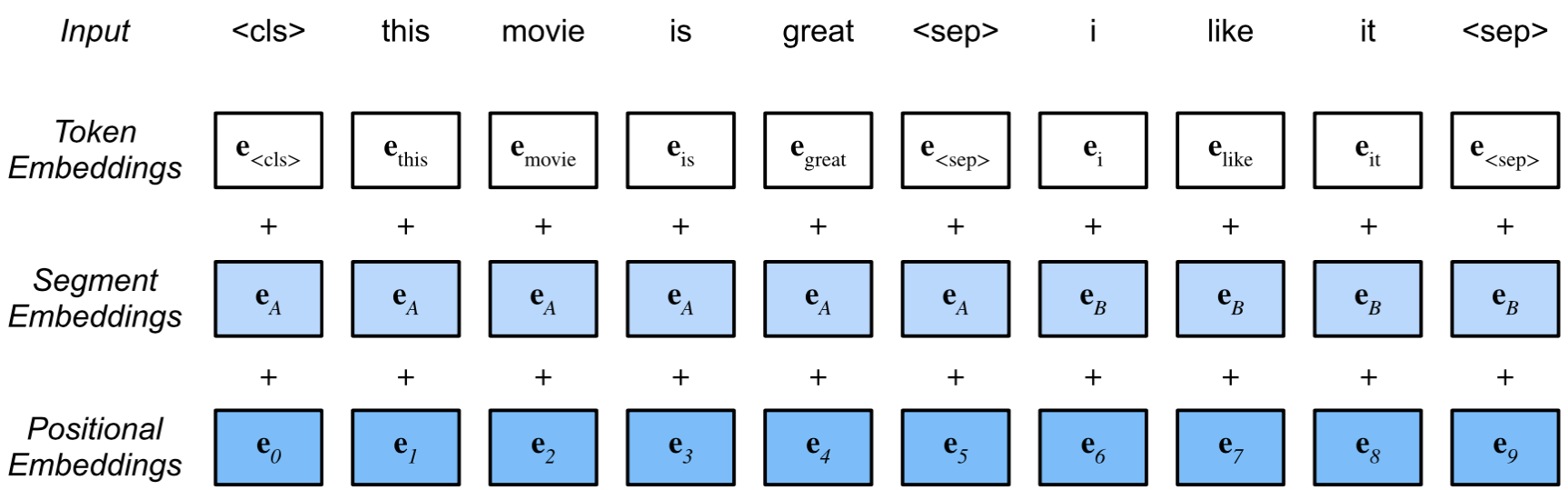
"tastes"

# Bidirectional Encoder Representations from Transformers (BERT) → Two Tasks

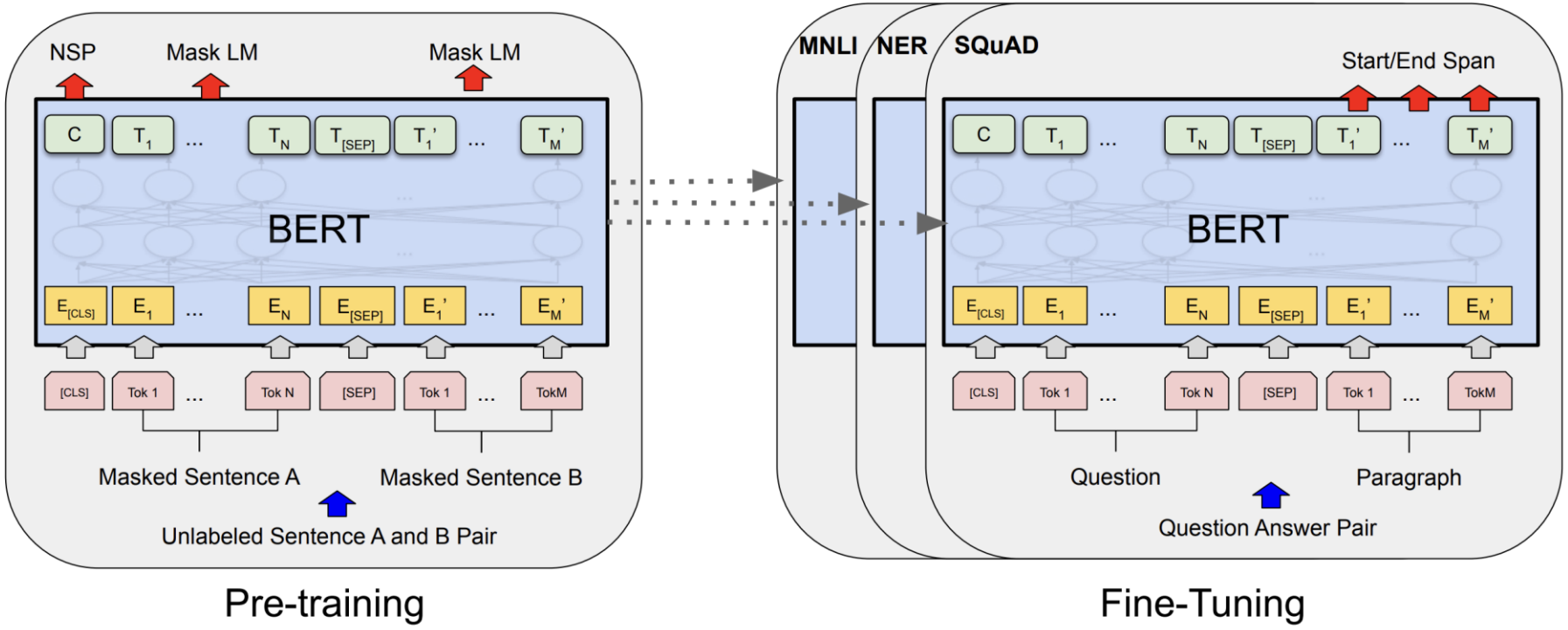
B. Next Sentence Prediction

If B follows A, then $y = 1$; if B is a randomly chosen sentence, then $y = 1$

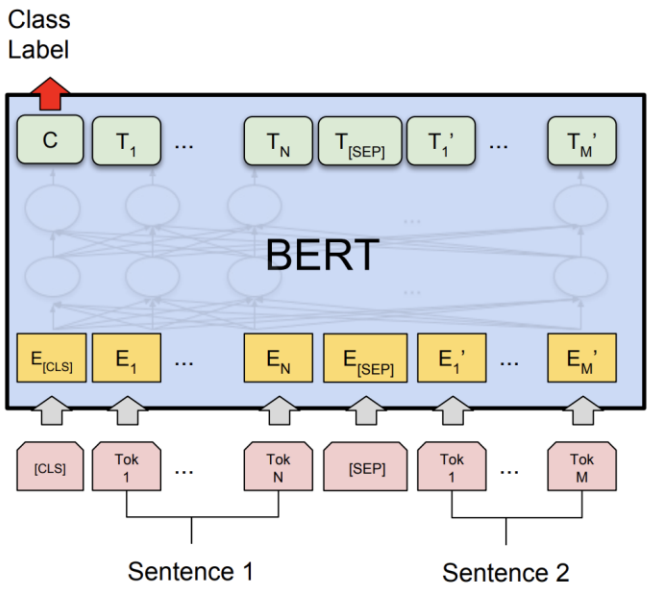
$$\text{CLS}\ A_1\ A_2; \ldots\ A_m;\ \text{SEP}\ B_1\ B_2; \ldots; B_n\ \text{SEP}$$



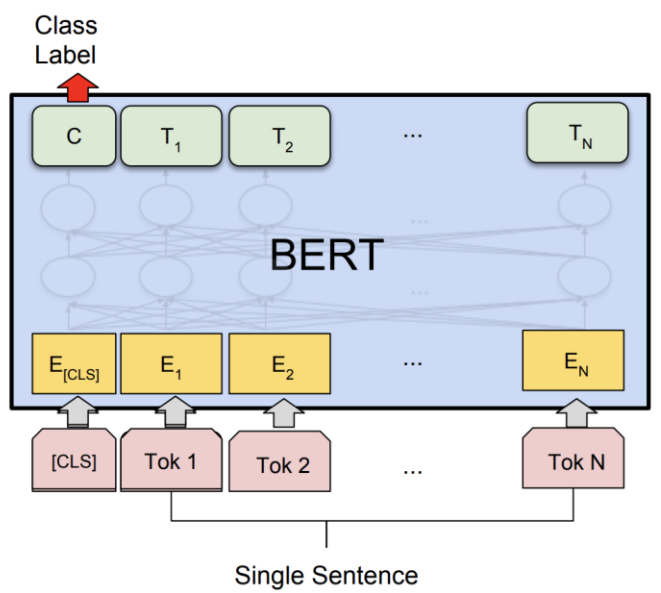
# BERT Fine-Tuning



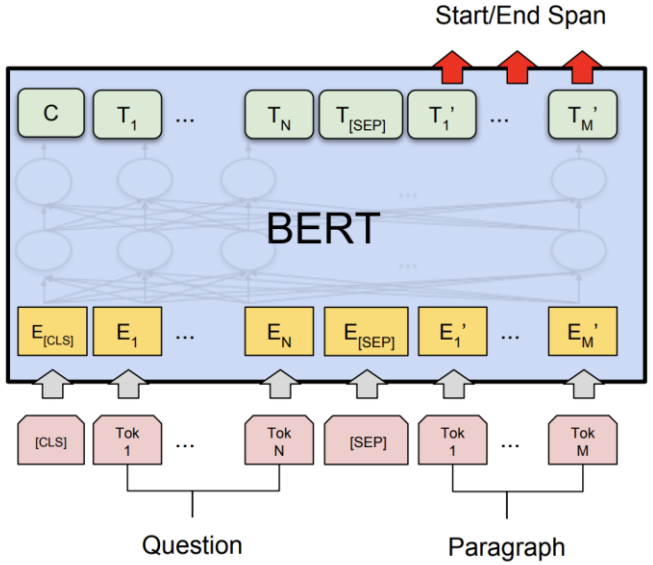
# BERT Fine-Tuning



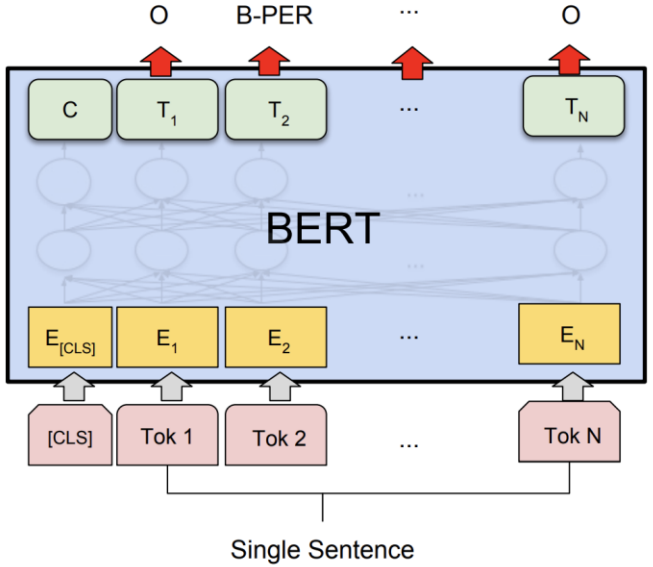

(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG




(b) Single Sentence Classification Tasks: SST-2, CoLA

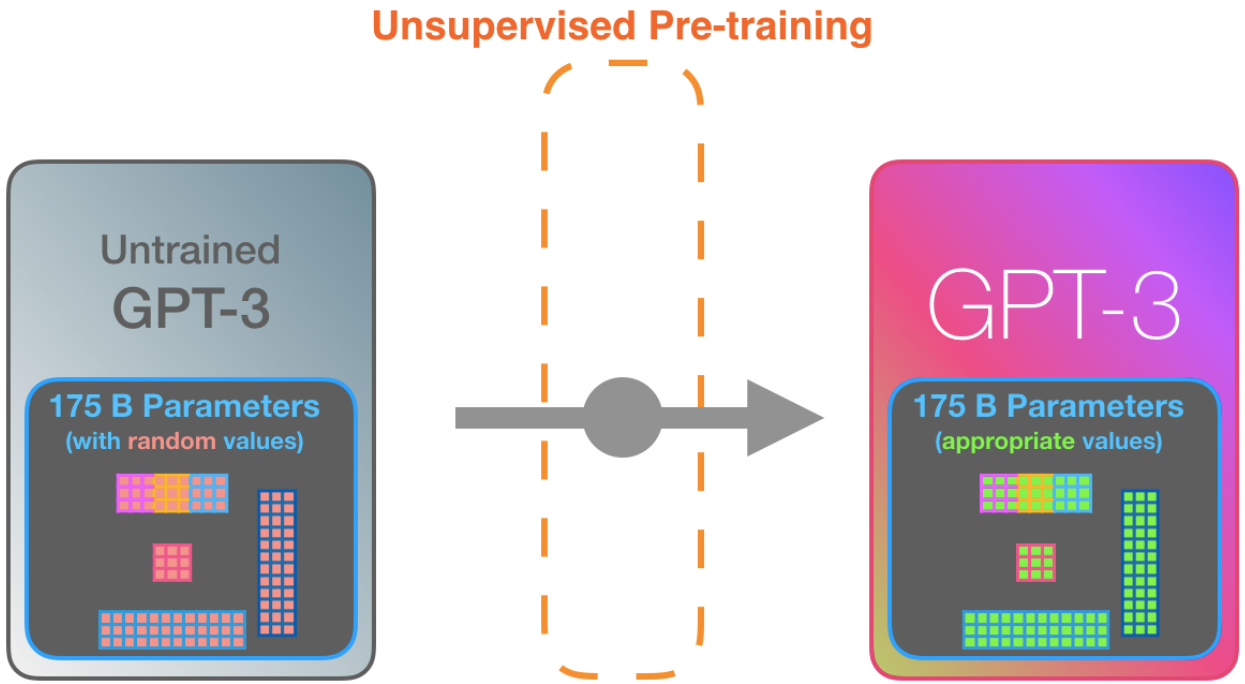



(c) Question Answering Tasks: SQuAD v1.1

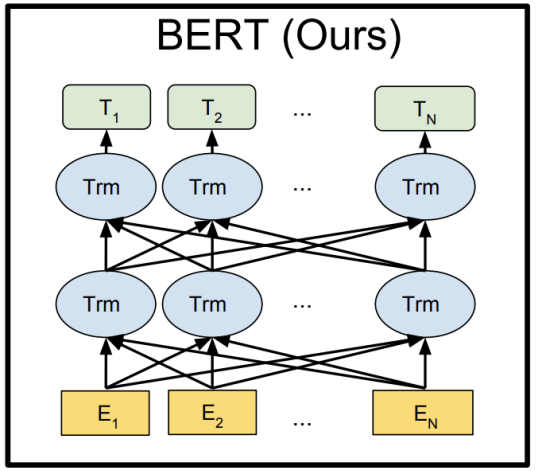


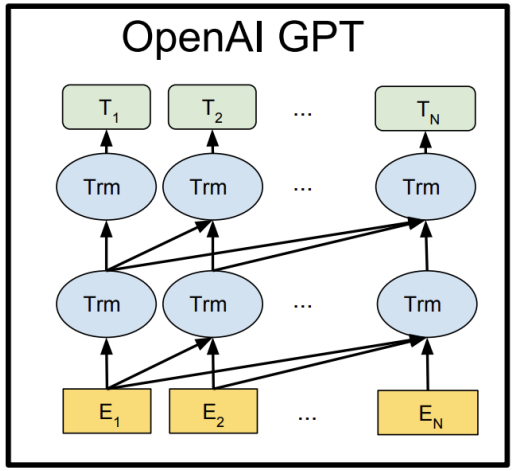

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

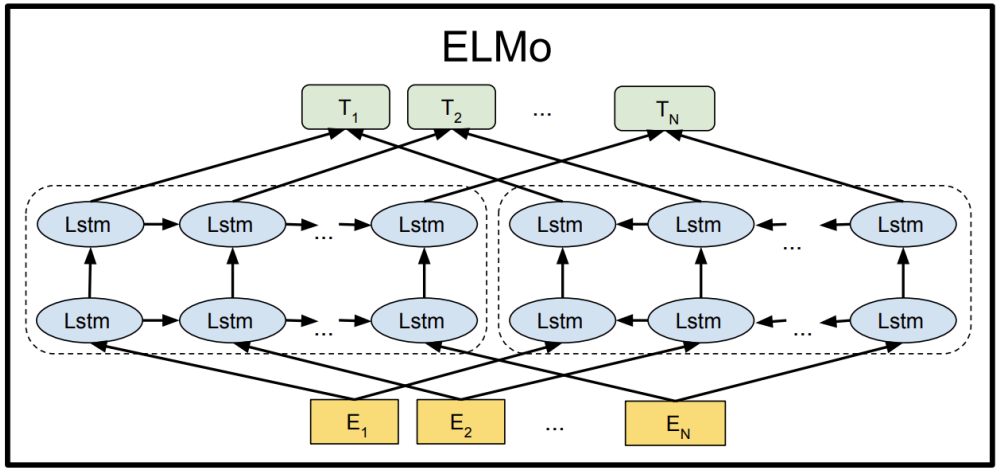
# GPT


Unsupervised Pre-training
Untrained
GPT-3
175 B Parameters
(with random values)
GPT-3
175 B Parameters
(appropriate values)

# Difference in BERT, GPT, ELMo




Transformer encoder

Transformer decoder

The concatenation of independently trained left-to-right and right-to left LSTMs

Fine-tuning approach

Feature-based approach

Jointly Conditioned on both left and right context in all layers

# ChatGPT

Ouyang, Long, et al. "Training language models to follow instructions with human feedback." *arXiv preprint arXiv:2203.02155* (2022).