Review

ML & DL

Introduction to machine/deep learning

Word

Word vectors, language modeling

Sequence

Sequence modeling, seq2seq learning

Transformer

Attention mechanism, encoder/decoder

Pretraining

Masking, natural language generation

Prompting

Prompts, in-context learning

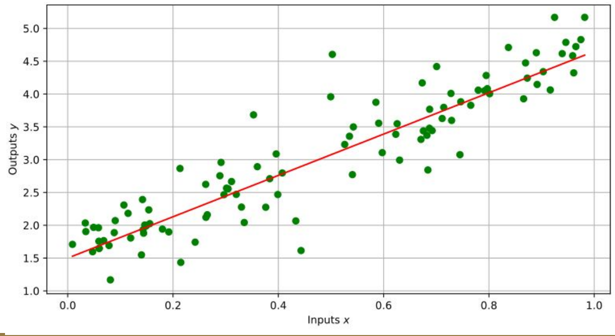
Linear Regression

What is it?

The basic idea behind a regression is that, you want to model the relationship between a real valued outcome variable y, and a vector of explanatory variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$.

A linear regression relates y to a linear predictor function of $\mathbf{x}$.
For a given data point i, the linear function is of the form:

$$\hat{y}_i = w_0 + w_1 x_{i1} + \ldots + w_d x_{id}$$



Linear Regression

Least Square Formulation of the Loss

In this formulation, the least squares fit is the line that **minimizes** the sum of the squared distances between observed data and predicted values, i.e. it minimizes the **Residual Sum of Squares** (*RSS*):

$$\arg\min_{\mathbf{w}} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

where $\hat{y}_i = w_0 + w_1 x_i$ is the predicted outcome for the i^{th} observation.

Logistic Regression

Combine sigmoid function with Bernoulli distribution

$$\mathcal{P}(y|\mathbf{x},\mathbf{w}) = Bernoulli(y|sigm(\mathbf{w}^\top \mathbf{x}))$$

$$p(y_i = 1|\mathbf{x}_i, \mathbf{w}) = sigm(\mathbf{w}^\top \mathbf{x}_i)$$

$$p(y_i = 0|\mathbf{x}_i, \mathbf{w}) = 1 - sigm(\mathbf{w}^\top \mathbf{x}_i)$$

$$\mathbf{w}^* = \mathrm{argmax}_{\mathbf{w}} \log p(\mathcal{D}|\mathbf{w})$$

Negative Log Likelihood (NLL)

$$= \mathrm{argmin}_{\mathbf{w}} - \textstyle\sum_{i=1}^{N} (y_i \log \mu_i + (1 - y_i) \log(1 - \mu_i))$$

This is also called the Cross Entropy Error Function.

Multiclass Logistic Regression

Different from binary logistic regression, we model the probability using **Softmax** as

$$\mathcal{P}(y_i = c|\mathbf{x}_i, \mathbf{W}) = \mu_{ic} = \frac{\exp(\mathbf{w}_c^\top \mathbf{x}_i)}{\sum_{c'=1}^{C} \exp(\mathbf{w}_{c'}^\top \mathbf{x}_i)}$$

$$\mathcal{P}(y_i|\mathbf{x}_i, \mathbf{W}) = \prod_{c=1}^{C} \mu_{ic}^{y_{ic}}$$

$$\boxed{\mathbf{W} = [\mathbf{w}_1; \ldots; \mathbf{w}_C]}$$

$$\boxed{y_{ic} = \mathbb{I}(y_i = c)}$$

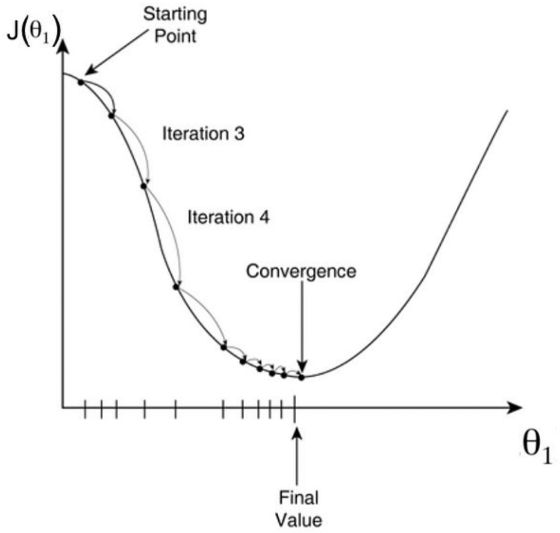
$$\begin{aligned}\mathbf{W}^* &= \operatorname{argmax}_{\mathbf{W}} \log p(\mathcal{D}|\mathbf{W}) \\ &= \operatorname{argmax}_{\mathbf{W}} \sum_{i=1}^{N} \log \prod_{c=1}^{C} \mu_{ic}^{y_{ic}} \\ &= \operatorname{argmin}_{\mathbf{W}} \sum_{i=1}^{N} \sum_{c=1}^{C} -y_{ic} \log \mu_{ic}\end{aligned}$$

Gradient Descent

The most commonly used method for unconstrained optimization

Goal: minimize $\sum_{i=1}^{N} J_i(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} J(\theta)$$

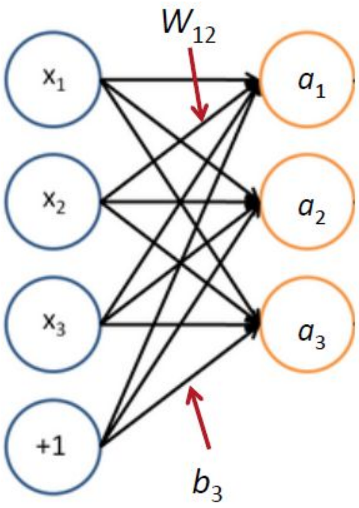


Neural Network

$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$

$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

....



In Matrix Notation

$$z = Wx + b$$

$$a = f(z)$$

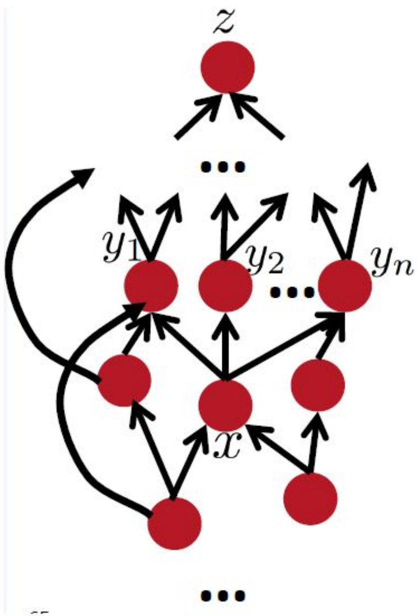
Activation function

Where f() is applied element-wise

$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$

Chain Rule of Derivatives

- Chain rule in computational graph



Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

$\{y_1, y_2, \ldots y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

Review

ML & DL

Introduction to machine/deep learning

Word

Word vectors, language modeling

Sequence

Sequence modeling, seq2seq learning

Transformer

Attention mechanism, encoder/decoder

Pretraining

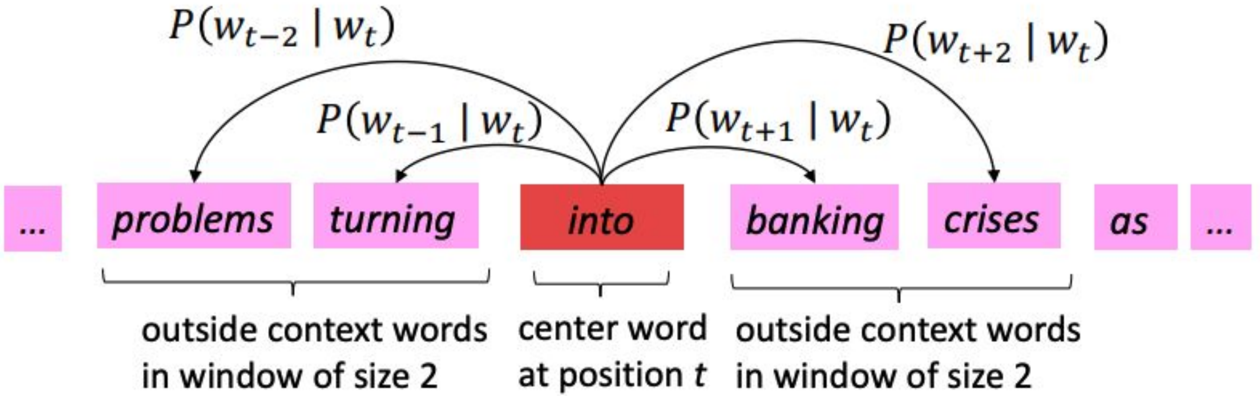
Masking, natural language generation

Prompting

Prompts, in-context learning

Word2vec – Skipgram

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2vec – Skipgram

- We want to minimize the objective function:

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

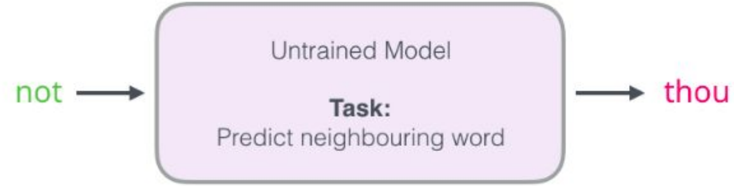
- **Question:** How to calculate $P(w_{t+j} \mid w_t;\theta)$?
- **Answer:** We will *use two* vectors per word *w*:
  - v_w when *w* is a center word
  - u_w when *w* is a context word
- Then for a center word *c* and a context word *o*:

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

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

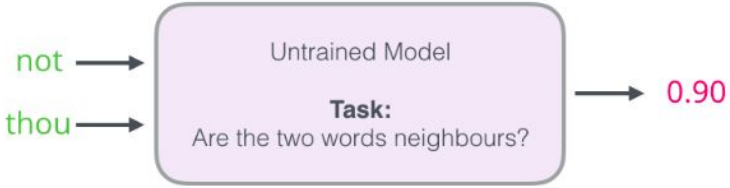
Skipgram with Negative Sampling

Change Task from



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

into



$$\log p(t = 1|c, o) = \log \sigma(u_o^T v_c)$$

$$\boxed{J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^{k} \log \sigma(-u_{o_i}^T v_c)}$$

Word Vectors for NER

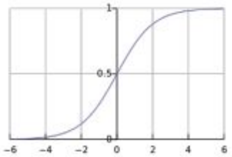
$$J_t(\theta) = \sigma(s) = \frac{1}{1+e^{-s}}$$

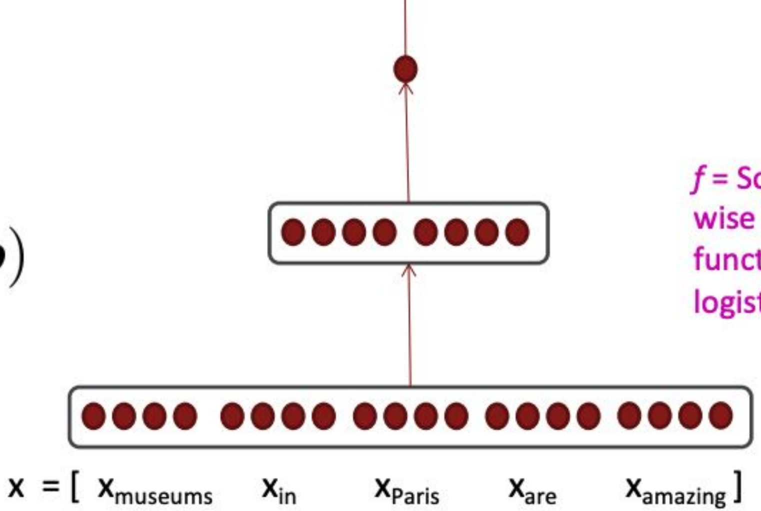
predicted model probability of class

$$s = \boldsymbol{u}^T \boldsymbol{h}$$

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$$\boldsymbol{x} \quad \text{(input)}$$





Binary prediction

f = Some element-wise non-linear function, e.g., logistic, tanh, ReLU

Word Vectors for NER

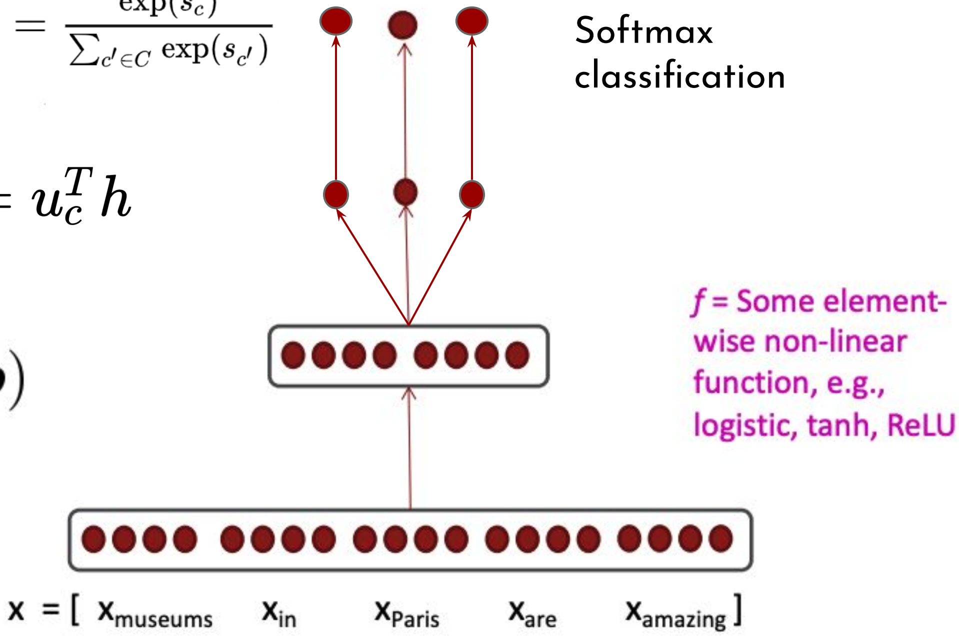
$\cancel{J_t(\theta) = \sigma(s)}$ $J_t(\theta) = softmax(s_c) = \frac{\exp(s_c)}{\sum_{c' \in C} \exp(s_{c'})}$

predicted model probability of class

$\cancel{s = \boldsymbol{u}^T \boldsymbol{h}}$ $s_c = u_c^T h$

$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$

$\boldsymbol{x}$ (input)



Language Modeling

- A language model takes a list of words (history/context), and attempts to predict the word that follows them

More formally: given a sequence of words $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$, compute the probability distribution of the next word $\boldsymbol{x}^{(t+1)}$:

$$P(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})$$

where $\boldsymbol{x}^{(t+1)}$ can be any word in the vocabulary $V = \{\boldsymbol{w}_1, ..., \boldsymbol{w}_{|V|}\}$

$$\begin{aligned} P(\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(T)}) &= P(\boldsymbol{x}^{(1)}) \times P(\boldsymbol{x}^{(2)} | \boldsymbol{x}^{(1)}) \times \cdots \times P(\boldsymbol{x}^{(T)} | \boldsymbol{x}^{(T-1)}, \ldots, \boldsymbol{x}^{(1)}) \\ &= \prod_{t=1}^{T} P(\boldsymbol{x}^{(t)} | \boldsymbol{x}^{(t-1)}, \ldots, \boldsymbol{x}^{(1)}) \end{aligned}$$

Neural Language Modeling

$$\boxed{\hat{y}_i = \frac{\exp(u_i h + b_i)}{\sum_j^{|V|} \exp(u_j h + b_j)}}$$

output distribution

$$\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1)$$

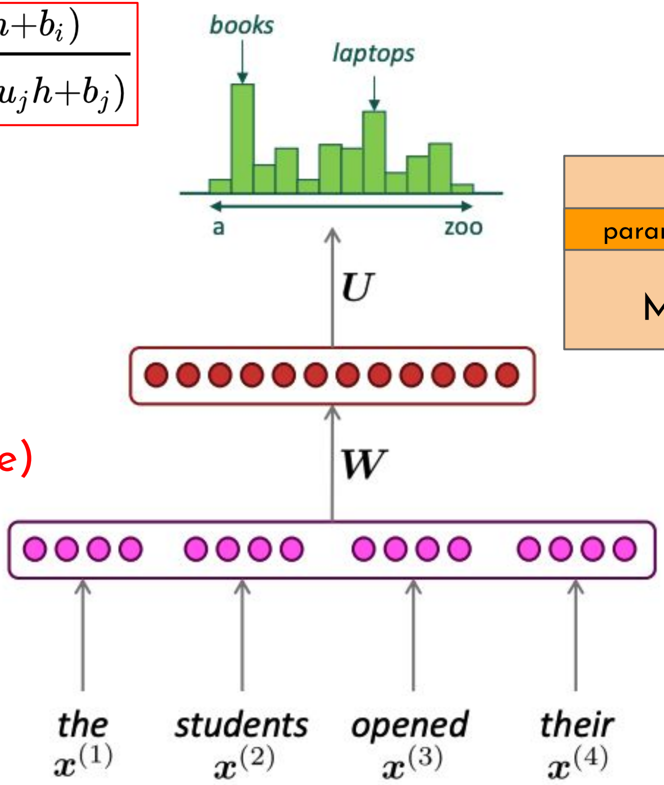
(Or average)

concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



param for ith word

Matrix U

Neural Language Modeling

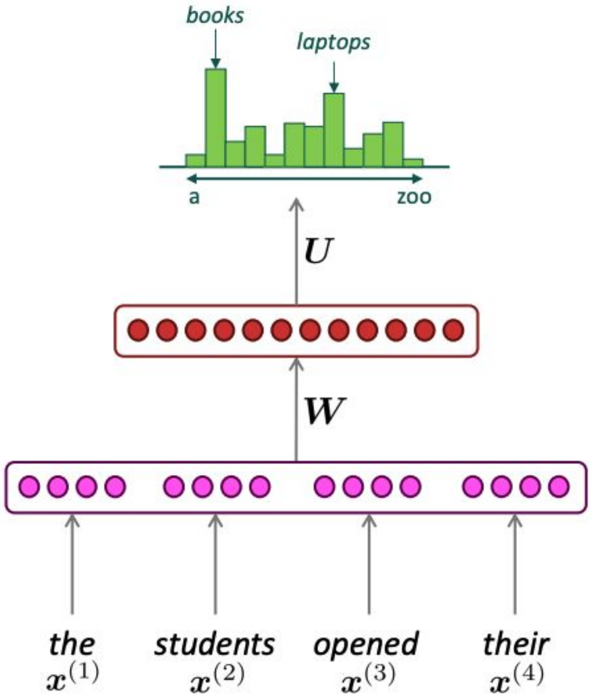
Improvements over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.

We need a neural architecture that can process *any length input*



Review

ML & DL

Introduction to machine/deep learning

Word

Word vectors, language modeling

Sequence

Sequence modeling, seq2seq learning

Transformer

Attention mechanism, encoder/decoder

Pretraining

Masking, natural language generation

Prompting

Prompts, in-context learning

Recurrent Neural Networks

hidden states

$h^{(t)} = \sigma\left(W_h h^{(t-1)} + W_e e^{(t)} + b_1\right)$

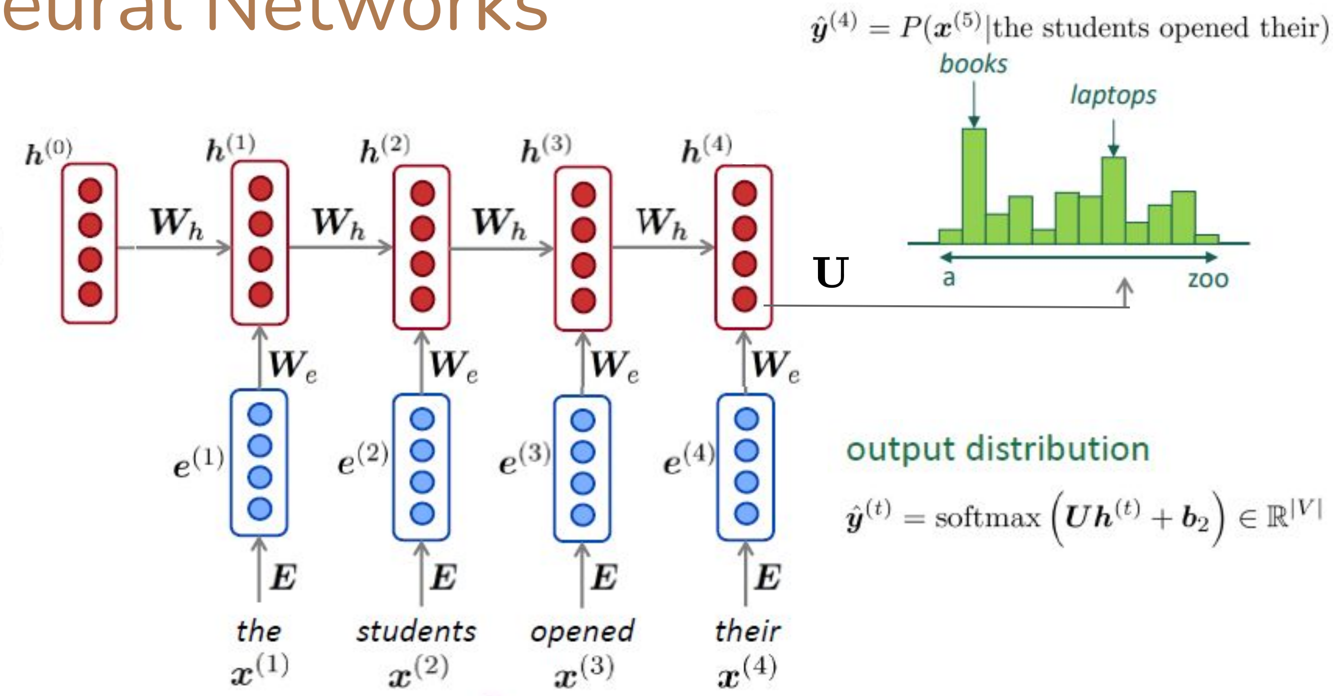
$h^{(0)}$ is the initial hidden state

word embeddings

$e^{(t)} = E x^{(t)}$

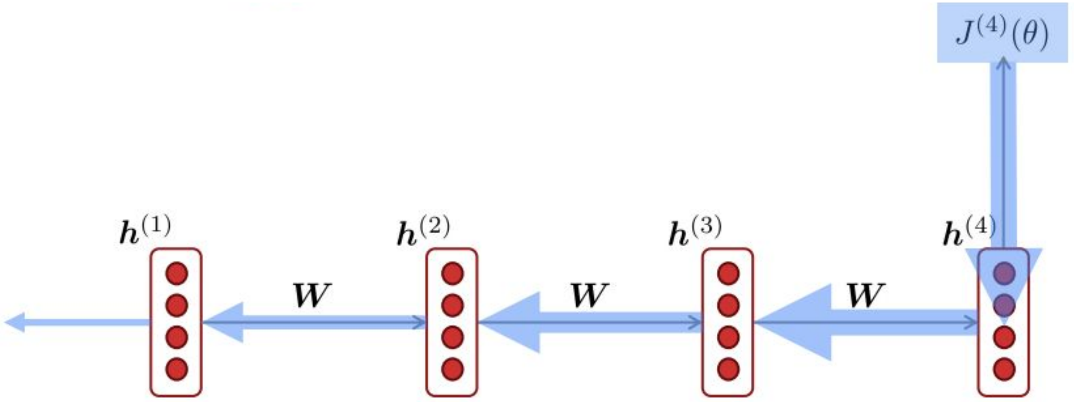
words / one-hot vectors

$x^{(t)} \in \mathbb{R}^{|V|}$



The input sequence can be of arbitrary length.

Vanishing Gradient Problem for RNNs



$$\frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(1)}} = \boxed{\frac{\partial \boldsymbol{h}^{(2)}}{\partial \boldsymbol{h}^{(1)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(3)}}{\partial \boldsymbol{h}^{(2)}}} \times \boxed{\frac{\partial \boldsymbol{h}^{(4)}}{\partial \boldsymbol{h}^{(3)}}} \times \frac{\partial J^{(4)}}{\partial \boldsymbol{h}^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:
When these are small, the gradient signal gets smaller and smaller as it backpropagates further

RNNs

Advantages of RNNs

- Can process any length input. Computation for step t can (in theory) use information from many steps back.
- Model size doesn't increase for longer input context.
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

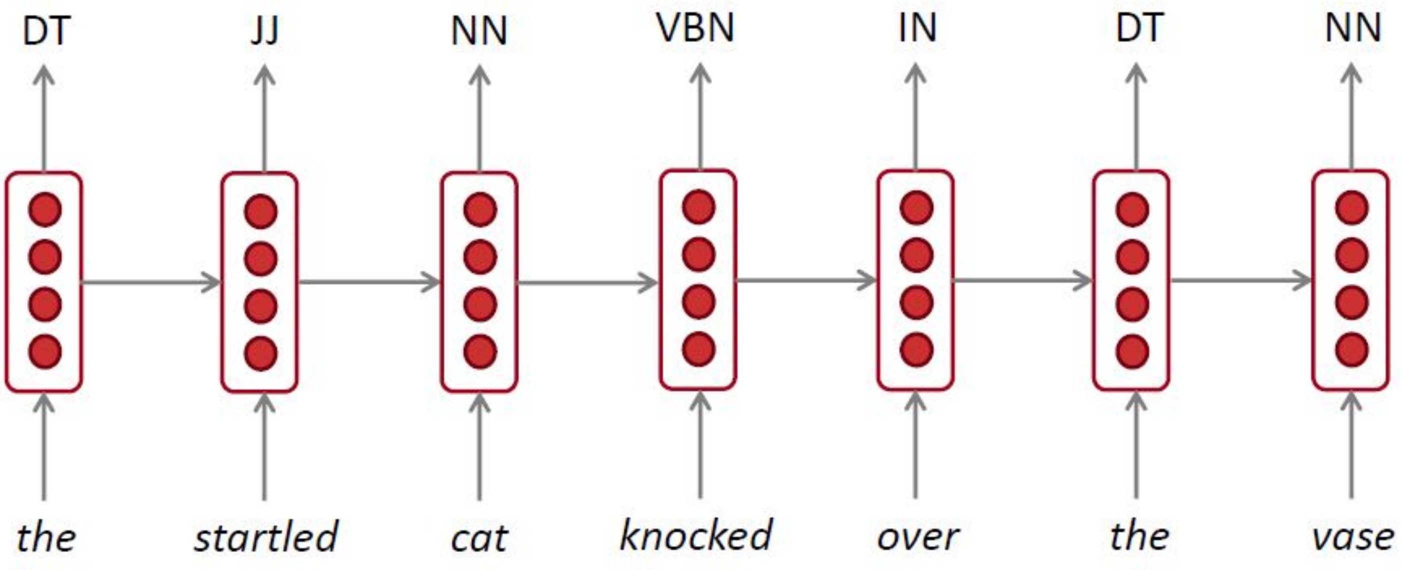
Disadvantages of RNNs

LSTM to the rescue!

- Recurrent computation is slow.
- In practice, difficult to access information from many steps back.

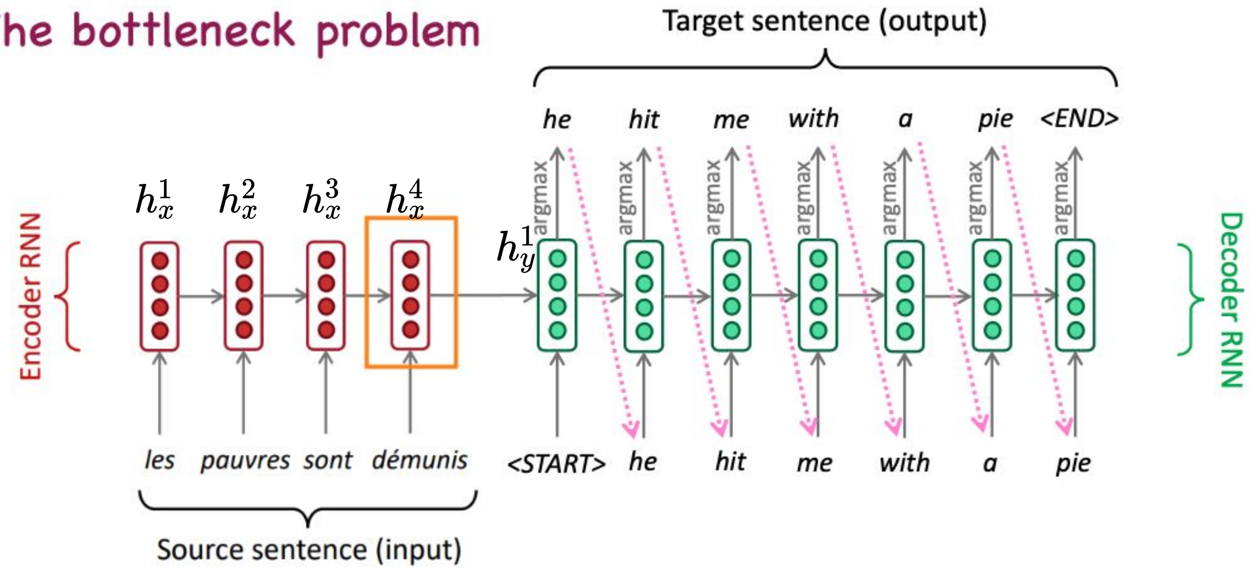
Sequence Tagging with RNNs

- `POS Tagging`



Seq2Seq Modeling with RNNs

- The bottleneck problem



Encoding of the source sentence. This needs to capture all information about the source sentence. Information bottleneck!

$$h_y^1 = f(W_{ye} \cdot e_{<START>} + W_{yh} \cdot \boxed{h_x^4} + b_y)$$

$$h_x^1 = f(W_{xe} \cdot e_{les} + W_{xh} \cdot h_x^0 + b_x)$$

Review

ML & DL
Introduction to machine/deep learning

Word
Word vectors, language modeling

Sequence
Sequence modeling, seq2seq learning

Transformer
Attention mechanism, encoder/decoder

Pretraining
Masking, natural language generation

Prompting
Prompts, in-context learning

Attentions

- We have encoder hidden states $h_1, \ldots, h_N \in \mathbb{R}^h$
- On timestep *t*, we have decoder hidden state $s_t \in \mathbb{R}^h$
- We get the attention scores $\boldsymbol{e}^t$ for this step:

$$\boldsymbol{e}^t = [\boldsymbol{s}_t^T \boldsymbol{h}_1, \ldots, \boldsymbol{s}_t^T \boldsymbol{h}_N] \in \mathbb{R}^N$$

- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

$$\alpha^t = \mathrm{softmax}(\boldsymbol{e}^t) \in \mathbb{R}^N$$

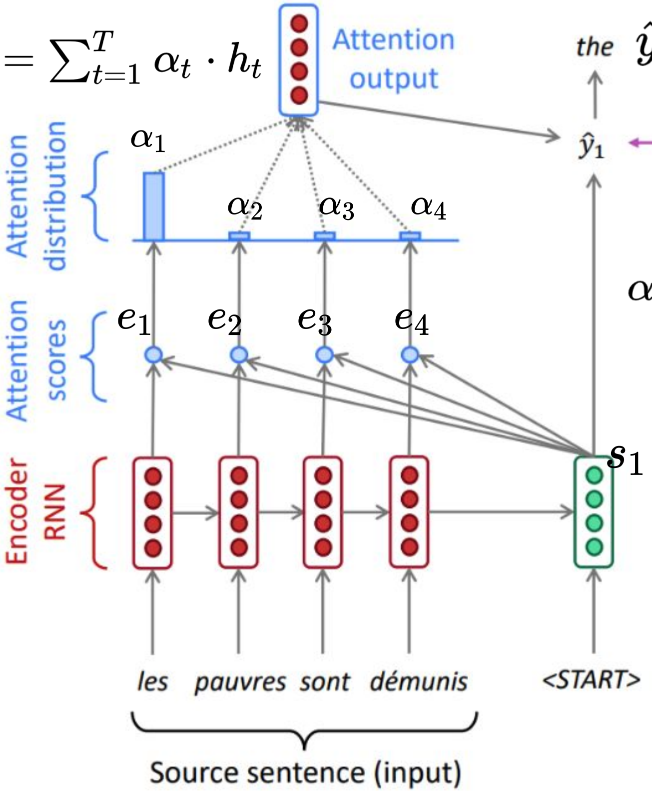
- We use α^t to take a weighted sum of the encoder hidden states to get the attention output $\boldsymbol{a}_t$

$$\boldsymbol{a}_t = \sum_{i=1}^{N} \alpha_i^t \boldsymbol{h}_i \in \mathbb{R}^h$$

Attentions

$$a_1 = \sum_{t=1}^{T} \alpha_t \cdot h_t$$

$$e_t = s_1^T h_t$$



$$\hat{y}_t = softmax(W_y \cdot [a_t; s_t] + b_y)$$

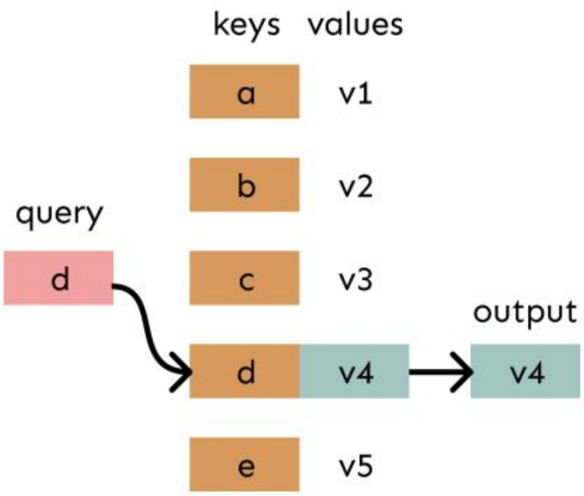
Concatenate attention output with decoder hidden state, then use to compute $\hat{y}_1$ as before

$$\alpha_t = softmax(e_t) = \frac{\exp(e_t)}{\sum_{i=1}^{T} \exp(e_i)}$$

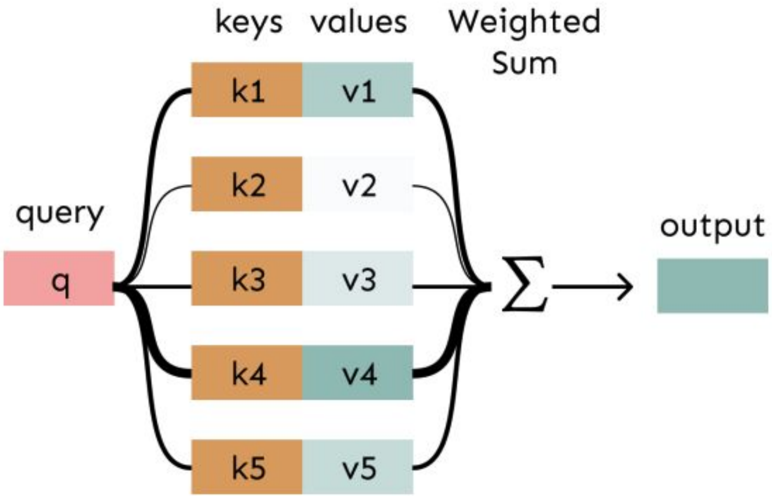
Attentions as QKV

We can think of **attention** as performing fuzzy lookup in a key-value store.

In a **lookup table**, we have a table of **keys** that map to **values**. The **query** matches one of the keys, returning its value.



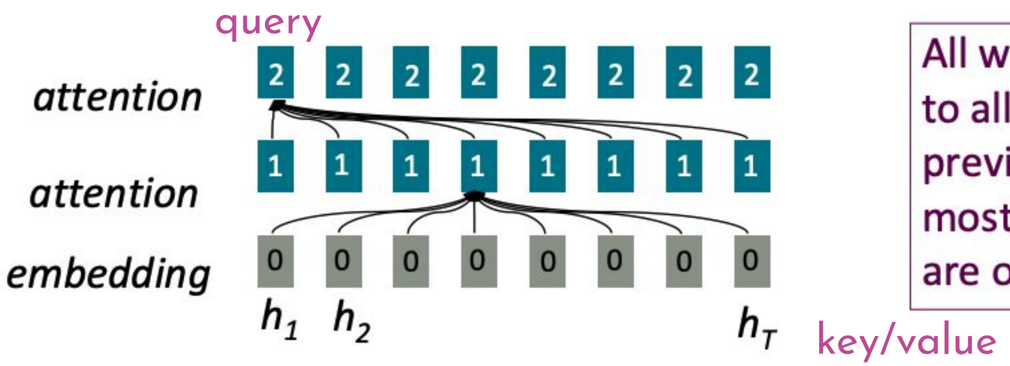
In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



Self Attentions

- Treats each word's representation as a query to access and incorporate information from a set of values.
- Easy to parallelize (per layer).
- Maximum interaction distance: O(1), since all words interact at every layer!

Each word can be query, key, value



All words attend to all words in previous layer; most arrows here are omitted

Self Attentions

Let $\boldsymbol{w}_{1:n}$ be a sequence of words in vocabulary V, like *Zuko made his uncle tea*.

For each $\boldsymbol{w}_i$, let $\boldsymbol{x}_i = E\boldsymbol{w}_i$, where $E \in \mathbb{R}^{d\times|V|}$ is an embedding matrix.

1. Transform each word embedding with weight matrices $\mathrm{Q}, \mathrm{K}, \mathrm{V}$, each in $\mathbb{R}^{d\times d}$

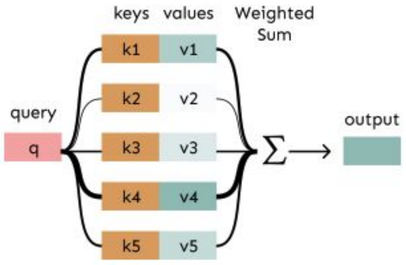
$$\boldsymbol{q}_i = Q\boldsymbol{x}_i \text{ (queries)} \qquad \boldsymbol{k}_i = K\boldsymbol{x}_i \text{ (keys)} \qquad \boldsymbol{v}_i = V\boldsymbol{x}_i \text{ (values)}$$

2. Compute pairwise similarities between keys and queries; normalize with softmax

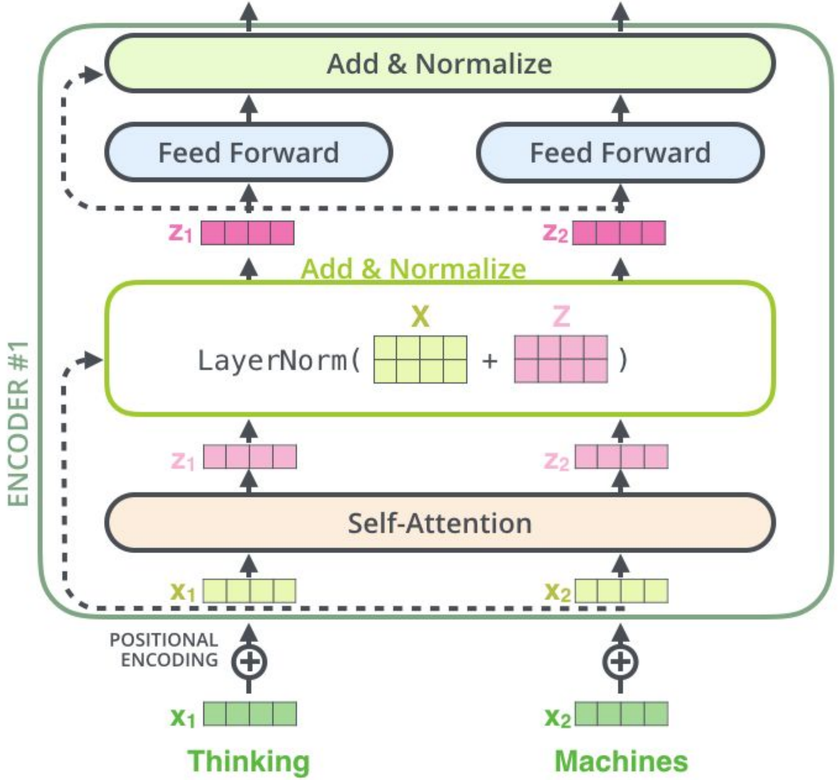
$$\boldsymbol{e}_{ij} = \boldsymbol{q}_i^{\top}\boldsymbol{k}_j \qquad \boldsymbol{\alpha}_{ij} = \frac{\exp(\boldsymbol{e}_{ij})}{\sum_{j'}\exp(\boldsymbol{e}_{ij'})}$$

3. Compute output for each word as weighted sum of values

$$\boldsymbol{o}_i = \sum_j \boldsymbol{\alpha}_{ij}\,\boldsymbol{v}_i$$

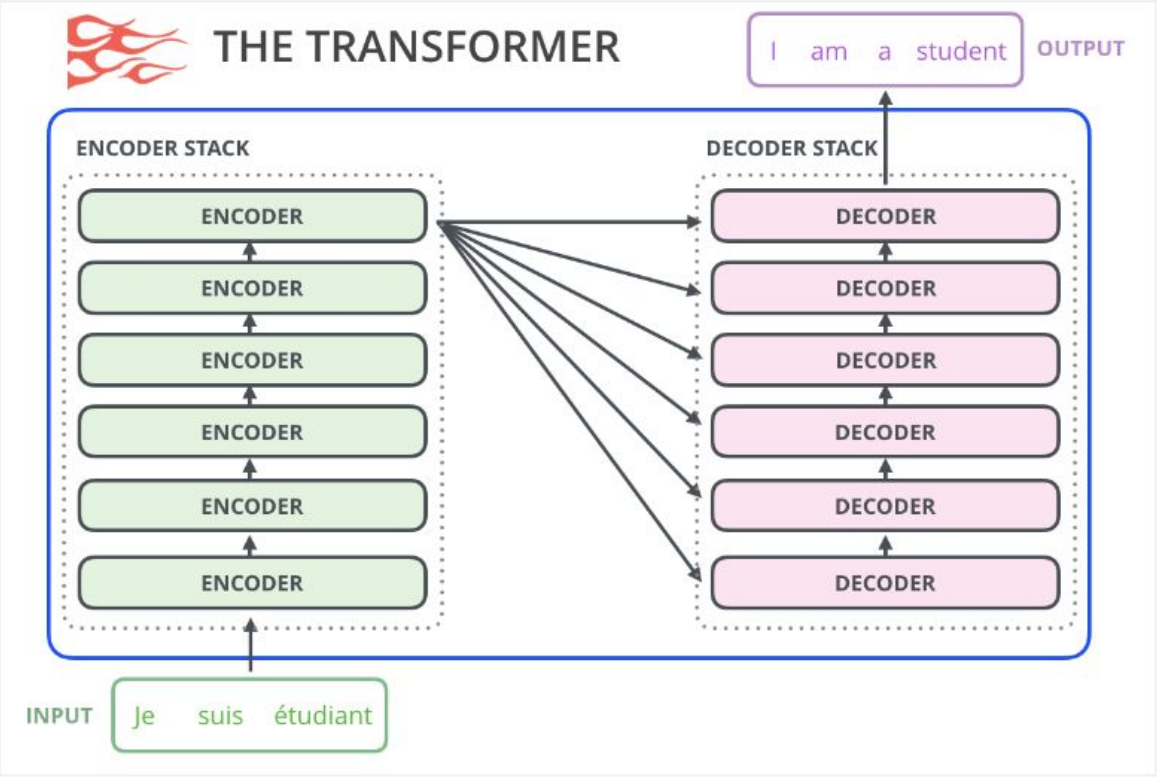


From Attentions to Transformers



multihead

Transformer Encoder-Decoder



Review

ML & DL
Introduction to machine/deep learning

Word
Word vectors, language modeling

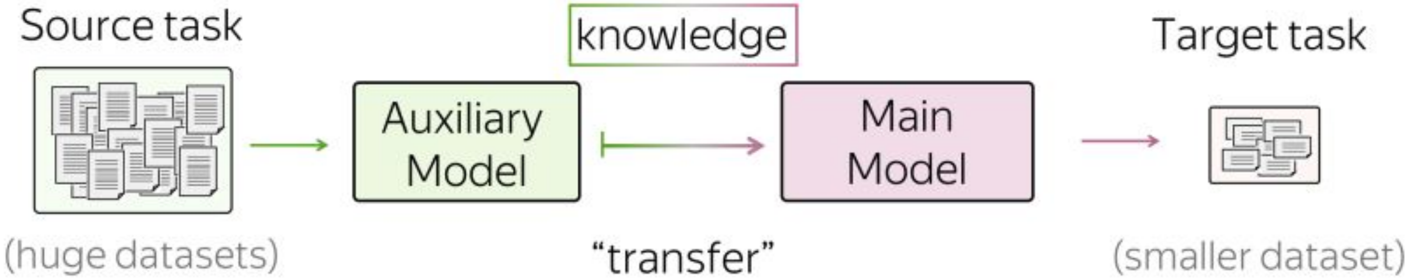
Sequence
Sequence modeling, seq2seq learning

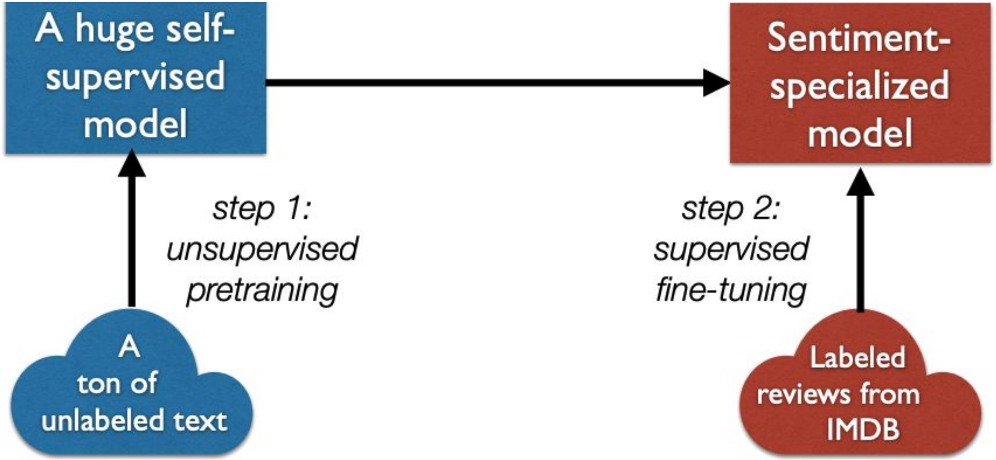
Transformer
Attention mechanism, encoder/decoder

Pretraining
Masking, natural language generation

Prompting
Prompts, in-context learning

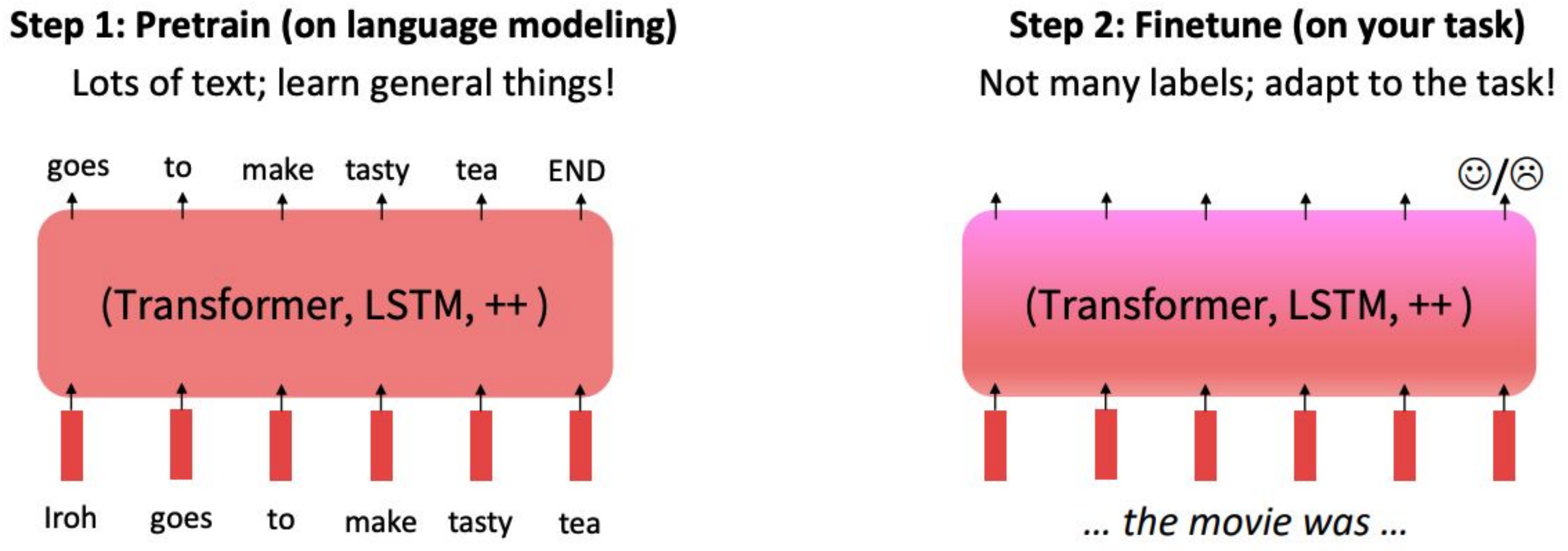
Pre-training and Fine-tuning



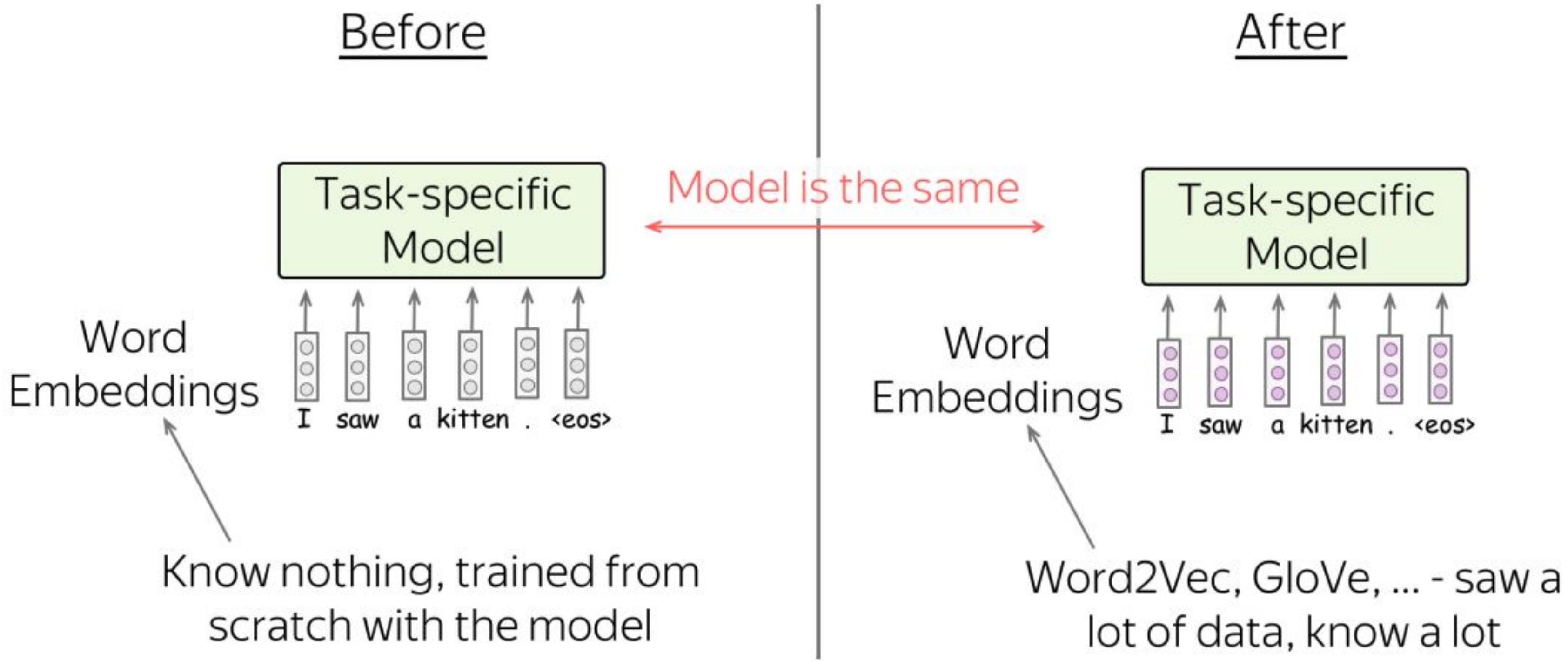


Pre-training to Fine-tuning

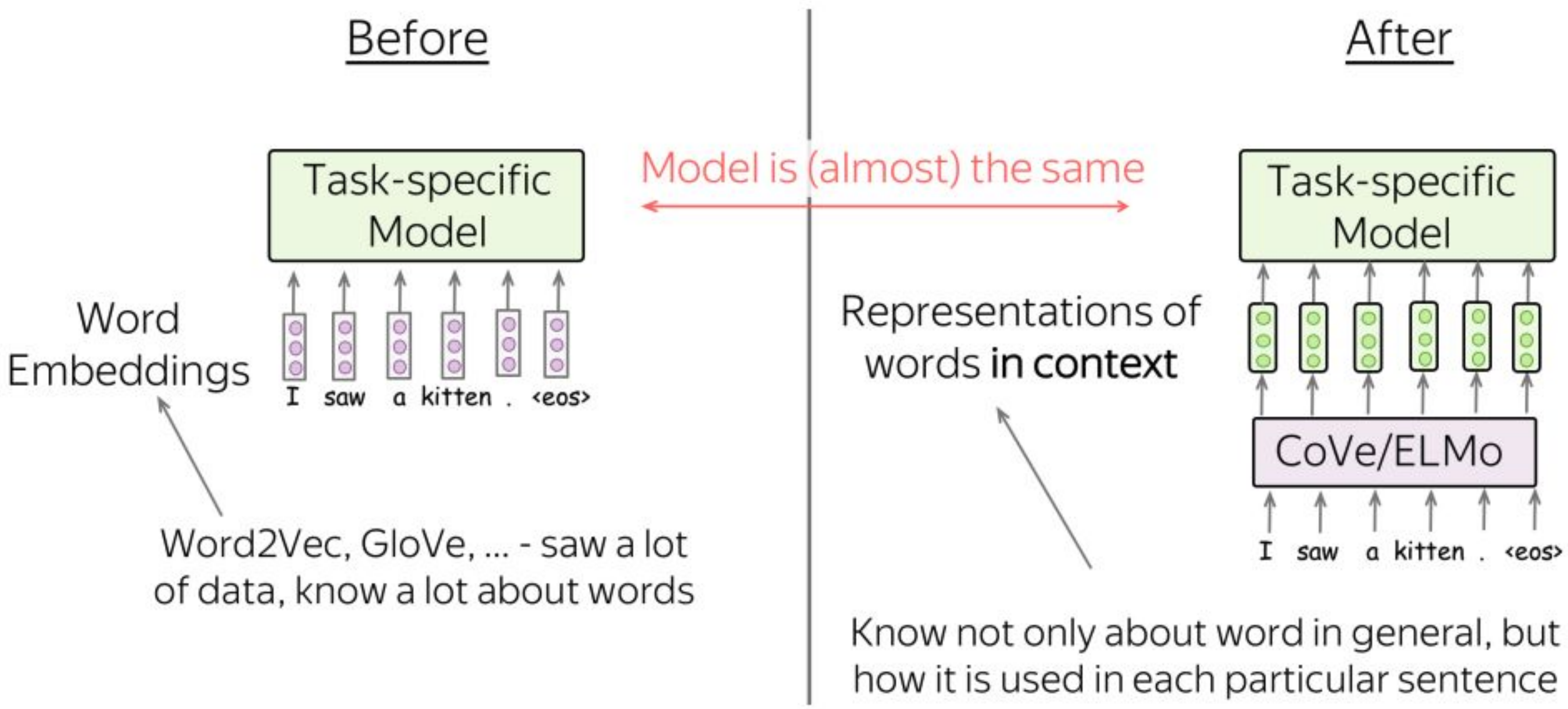
Pretraining can improve NLP applications by serving as parameter initialization.



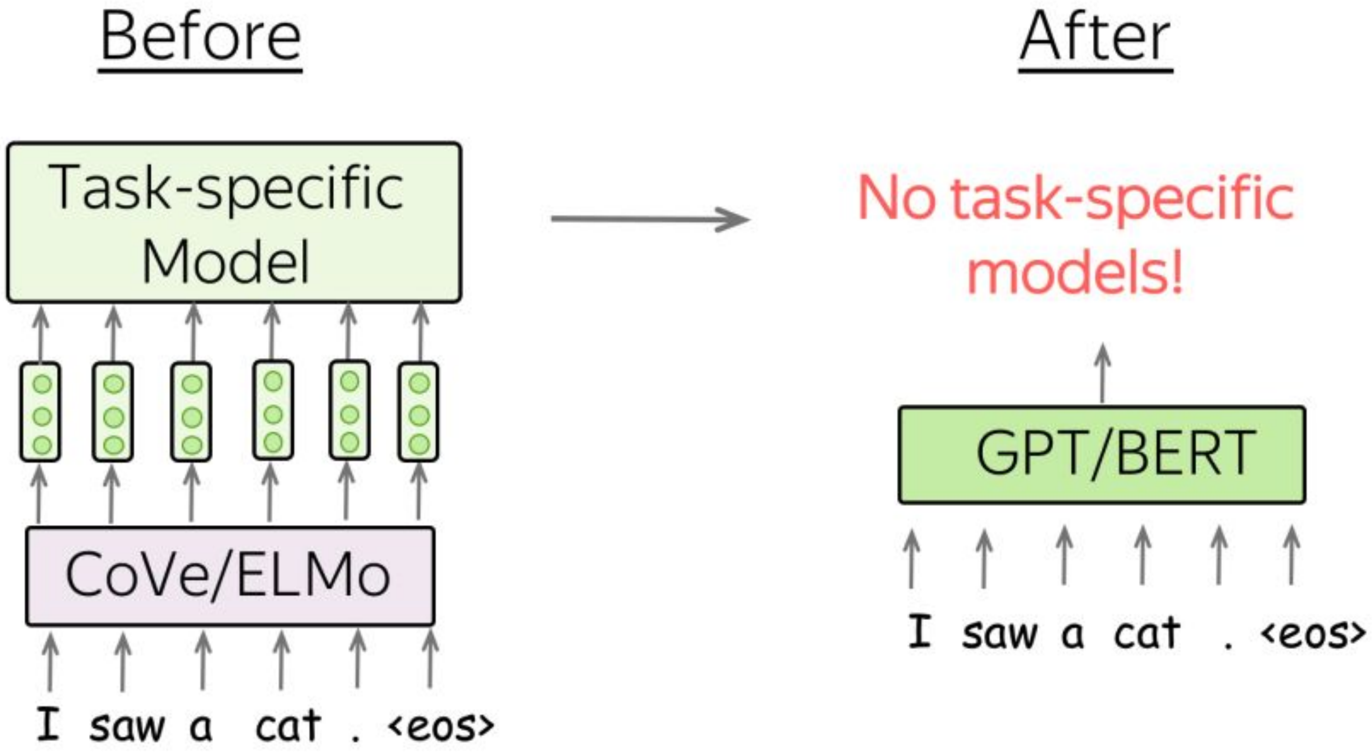
Transfer Through Word Embeddings



Transferring Words-in-Context

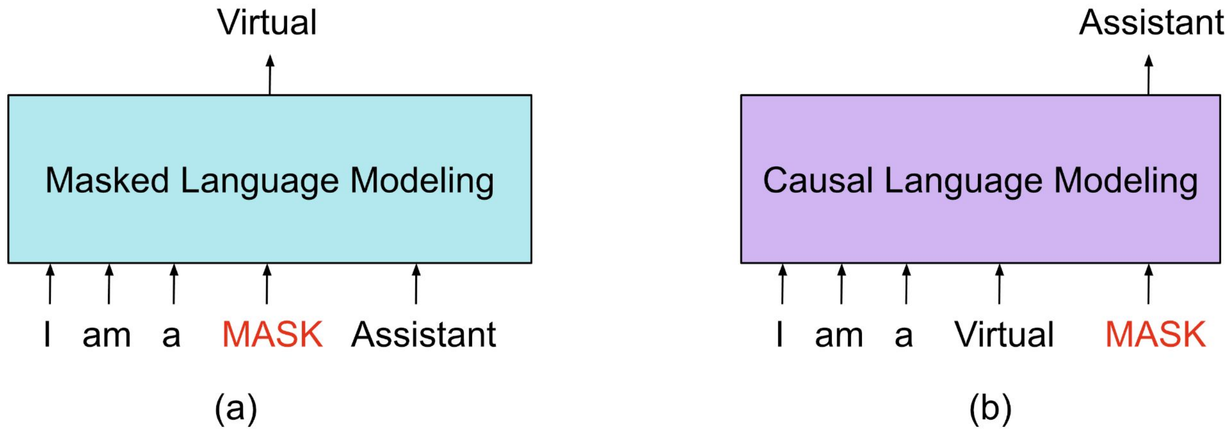


Transferring Entire Models



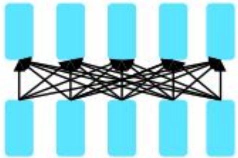
Language Model Pretraining

- Masked Language Modeling (MLM)
- (Causal) Language Modeling (LM)



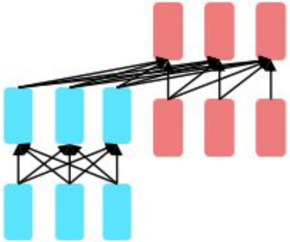
3 Training Paradigms

The neural architecture influences the type of pretraining, and natural use cases.



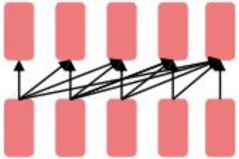
Encoders

- Gets bidirectional context – can condition on future!
- How do we train them to build strong representations?



Encoder-Decoders

- Good parts of decoders and encoders?
- What's the best way to pretrain them?



Decoders

- Language models! What we've seen so far.
- Nice to generate from; can't condition on future words

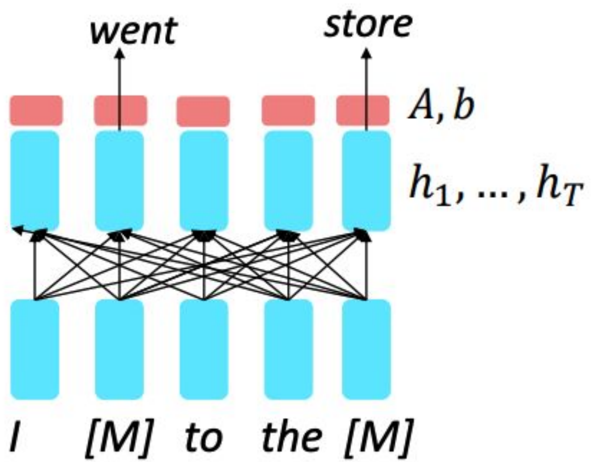
Encoder

So far, we've looked at language model pretraining. But **encoders get bidirectional context,** so we can't do language modeling!

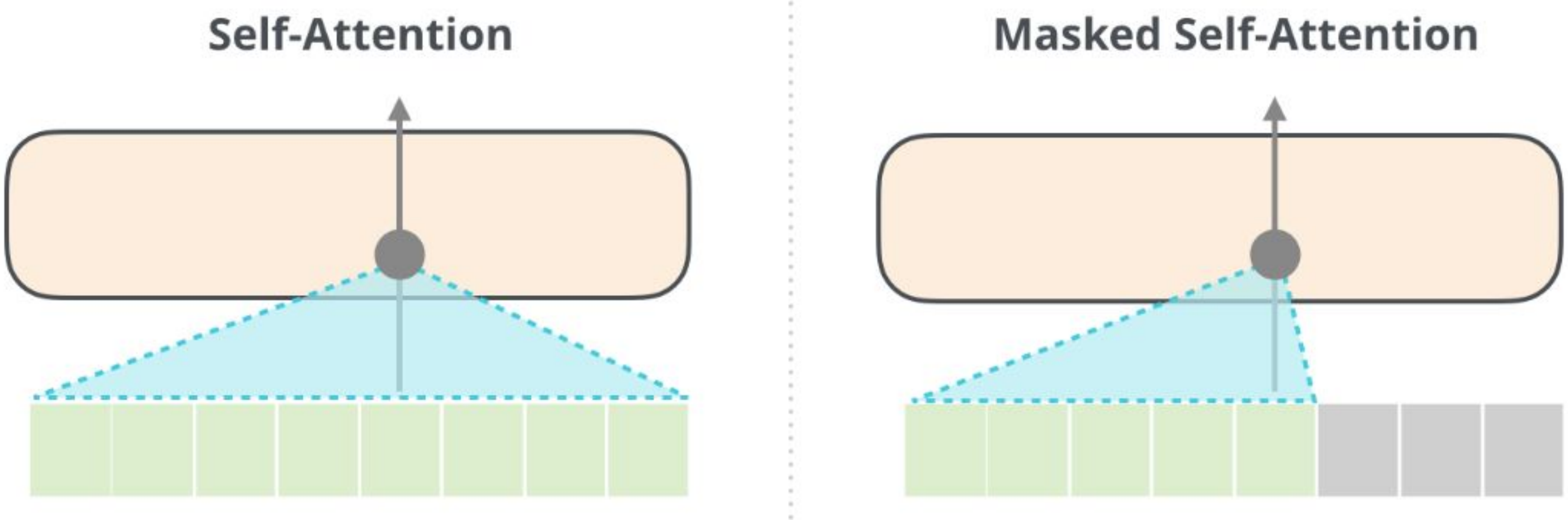
Idea: replace some fraction of words in the input with a special [MASK] token; predict these words.

$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$y_i \sim Aw_i + b$$

Only add loss terms from words that are "masked out." If $\tilde{x}$ is the masked version of x, we're learning $p_\theta(x|\tilde{x})$. Called **Masked LM**.



Masked Attentions in Decoder



Review

ML & DL

Introduction to machine/deep learning

Word

Word vectors, language modeling

Sequence

Sequence modeling, seq2seq learning

Transformer

Attention mechanism, encoder/decoder

Pretraining

Masking, natural language generation

Prompting

Prompts, in-context learning

Zero-Shot / Few-Shot Prompting

The End

Good luck to everyone!!!