

Review

ML & DL

Introduction to
machine/deep learning

Transformer

Attention mechanism,
encoder/decoder

Pretraining

Masking, natural
language generation

Word

Word vectors,
language modeling

Sequence

Sequence modeling,
seq2seq learning

Prompting

Prompts, in-context
learning

Linear Regression

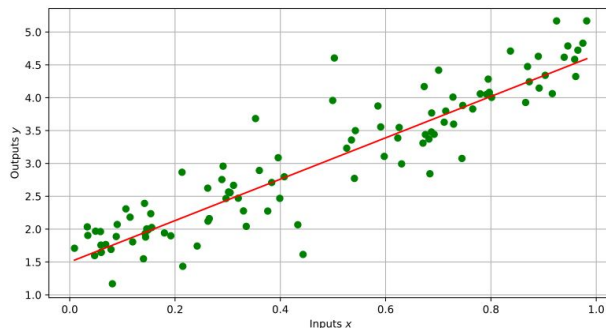
What is it?

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

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:

$$\hat{y}_i = w_0 + w_1x_{i1} + \dots + w_dx_{id}$$



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


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

$$p(y_i = 1|\mathbf{x}_i, \mathbf{w}) = \text{sigm}(\mathbf{w}^\top \mathbf{x}_i) \quad \text{squash value to (0,1)}$$

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

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

Negative Log
Likelihood
(NLL)


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

This is also called the **Cross Entropy Error Function**.

Multiclass Logistic Regression

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

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

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

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

$$= \operatorname{argmax}_{\mathbf{W}} \sum_{i=1}^N \log \prod_{c=1}^C \mu_{ic}^{y_{ic}}$$

$$= \operatorname{argmin}_{\mathbf{W}} \sum_{i=1}^N \sum_{c=1}^C -y_{ic} \log \mu_{ic}$$

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

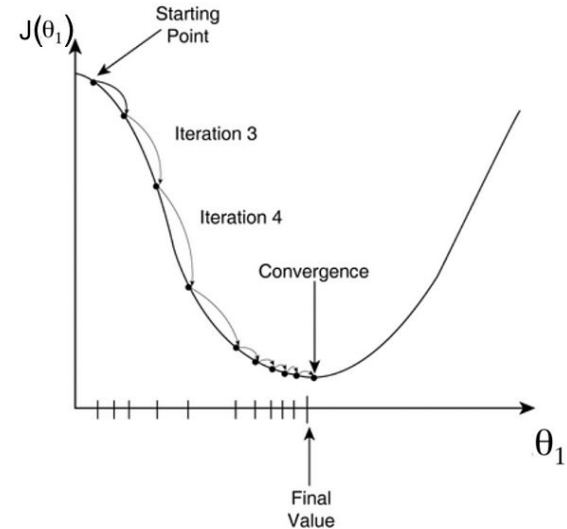
$$y_{ic} = \mathbb{I}(y_i = c)$$

Gradient Descent

The most commonly used method for unconstrained optimization

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

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



Neural Network

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

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

.....

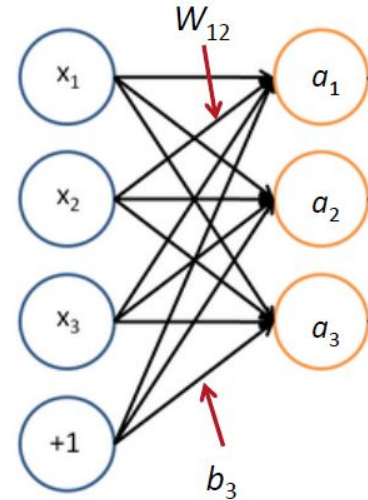
In Matrix Notation

$$z = Wx + b$$

$$a = f(z)$$

Where $f()$ is applied element-wise

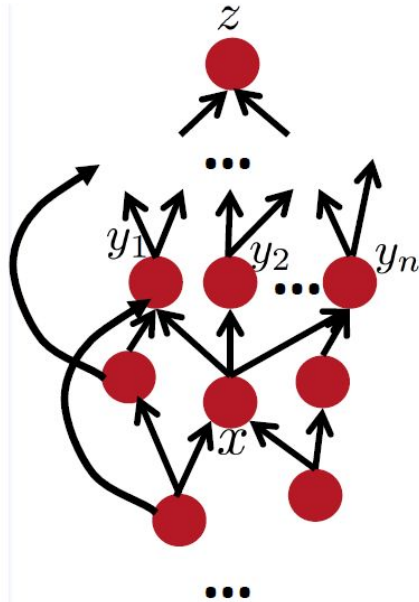
$$f([z_1, z_2, z_3]) = [f(z_1), f(z_2), f(z_3)]$$



Activation function

Chain Rule of Derivatives

- Chain rule in computational graph



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

$\{y_1, y_2, \dots, y_n\}$ = 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

Transformer

Attention mechanism,
encoder/decoder



Pretraining

Masking, natural
language generation

Word

Word vectors,
language modeling



Sequence

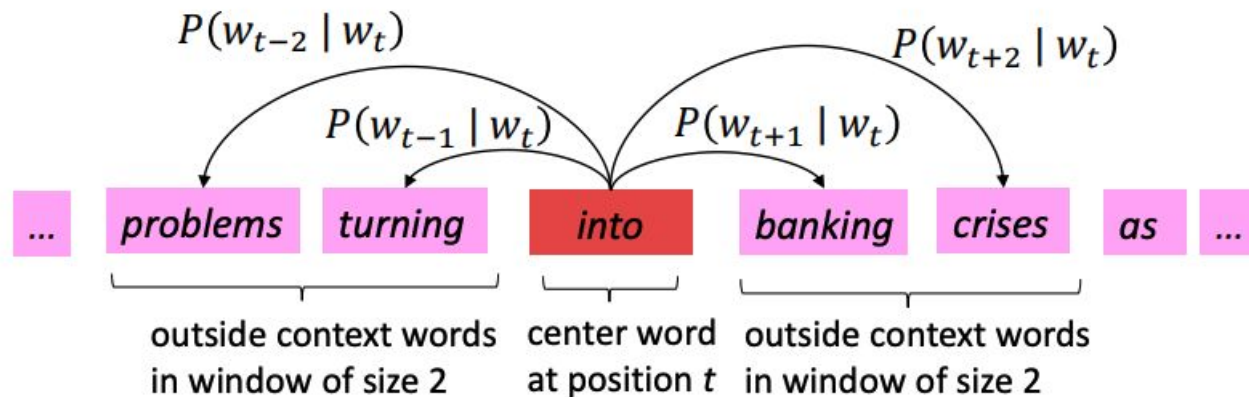
Sequence modeling,
seq2seq learning

Prompting

Prompts, in-context
learning

Word2vec – Skipgram

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



Word2vec – Skipgram

- We want to minimize the objective function:

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

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

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

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

Skipgram with Negative Sampling

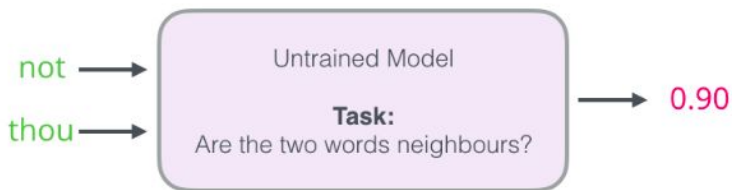
Change Task from



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

negative sampling to make the problem less costly,

into



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

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

Word Vectors for NER

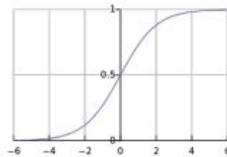
$$J_t(\theta) = \sigma(s) = \frac{1}{1 + e^{-s}}$$

predicted model
probability of class

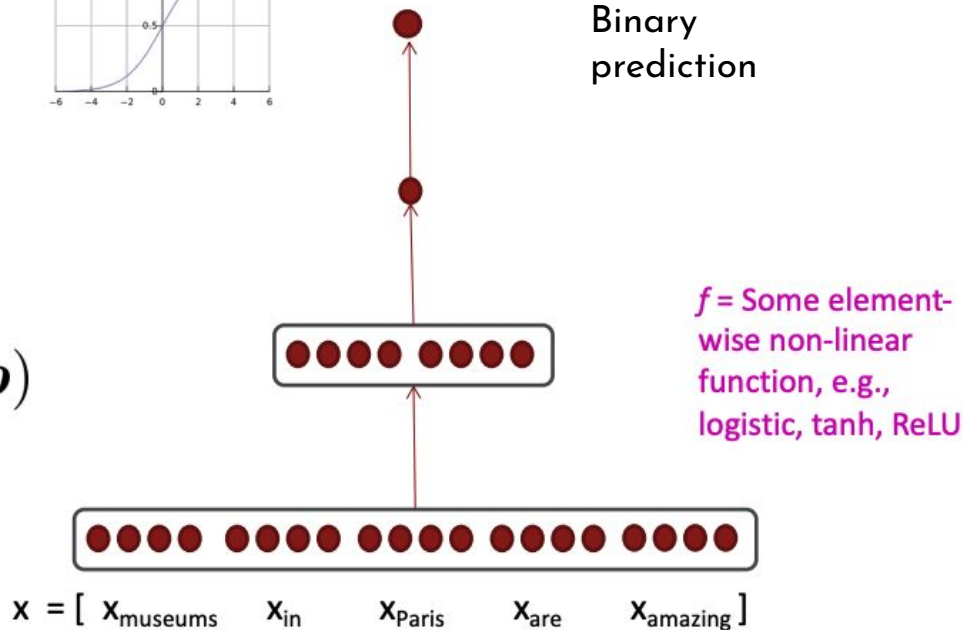
$$s = u^T h$$

$$h = f(Wx + b)$$

x (input)



Binary
prediction



Word Vectors for NER

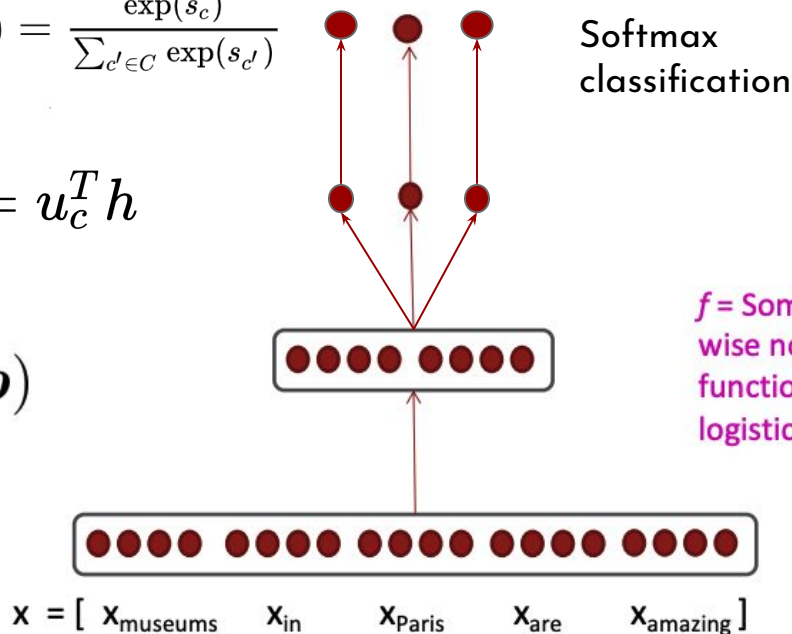
~~$J_t(\theta) = \sigma(s)$~~ $J_t(\theta) = softmax(s_c) = \frac{\exp(s_c)}{\sum_{c' \in C} \exp(s_{c'})}$

predicted model
probability of class

~~$s = u^T h$~~ $s_c = u_c^T h$

$$h = f(Wx + b)$$

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 $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}$, compute the probability distribution of the next word $\mathbf{x}^{(t+1)}$:

$$P(\mathbf{x}^{(t+1)} | \mathbf{x}^{(t)}, \dots, \mathbf{x}^{(1)})$$

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

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

Neural Language Modeling

this is the first LM created, a fixed window and then generate

output distribution

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

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

hidden layer

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

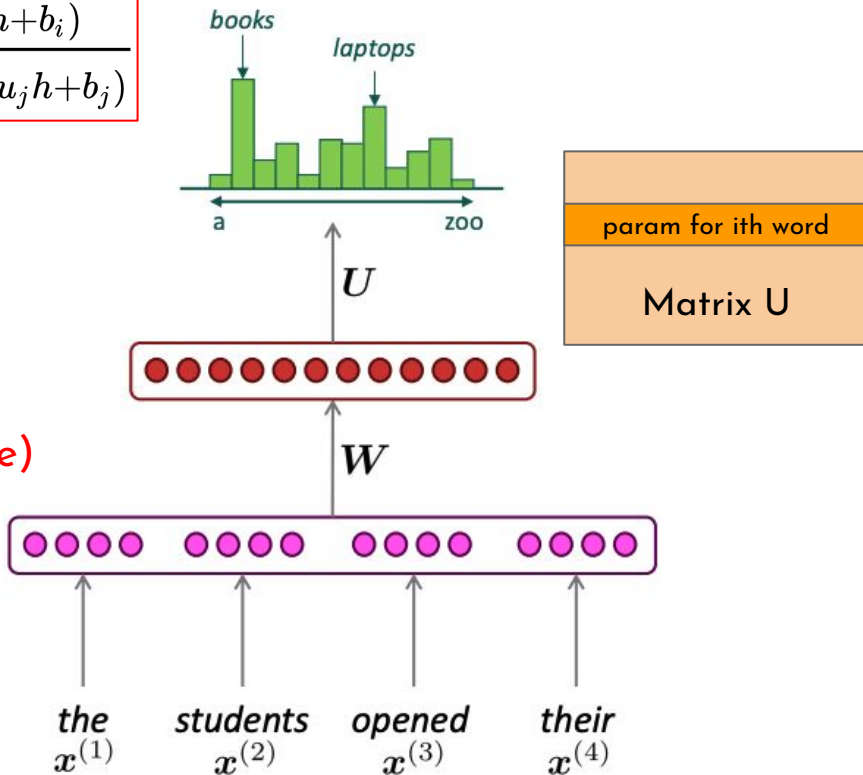
(Or average)

concatenated word embeddings

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

words / one-hot vectors

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



Neural Language Modeling

Improvements over n -gram LM:

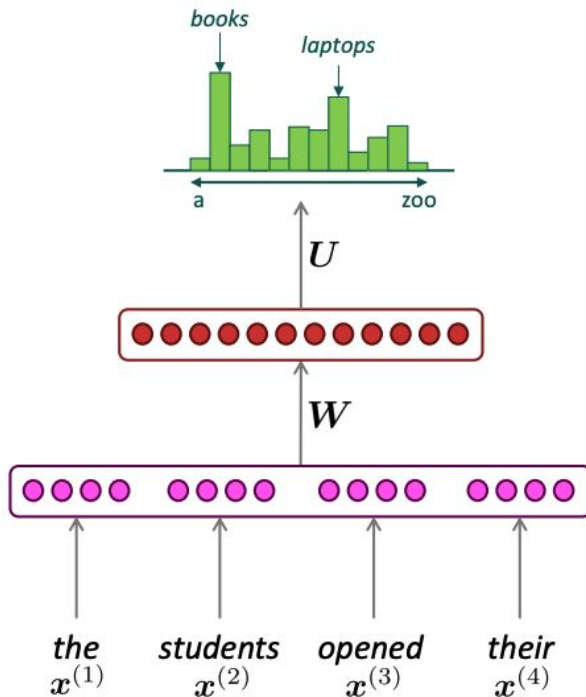
- No sparsity problem
- Don't need to store all observed n -grams

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

Transformer

Attention mechanism,
encoder/decoder

Pretraining

Masking, natural
language generation

Word

Word vectors,
language modeling

Sequence

Sequence modeling,
seq2seq learning

Prompting

Prompts, in-context
learning

Recurrent Neural Networks

hidden states

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

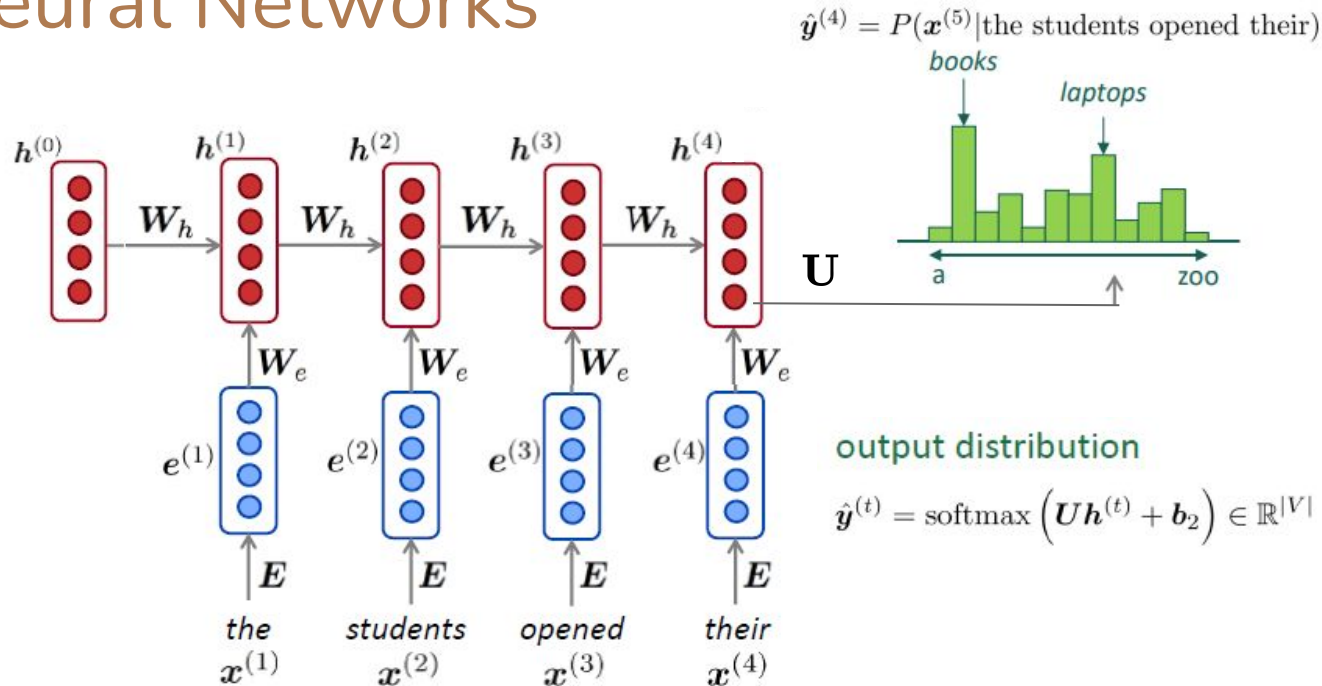
$h^{(0)}$ is the initial hidden state

word embeddings

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

words / one-hot vectors

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

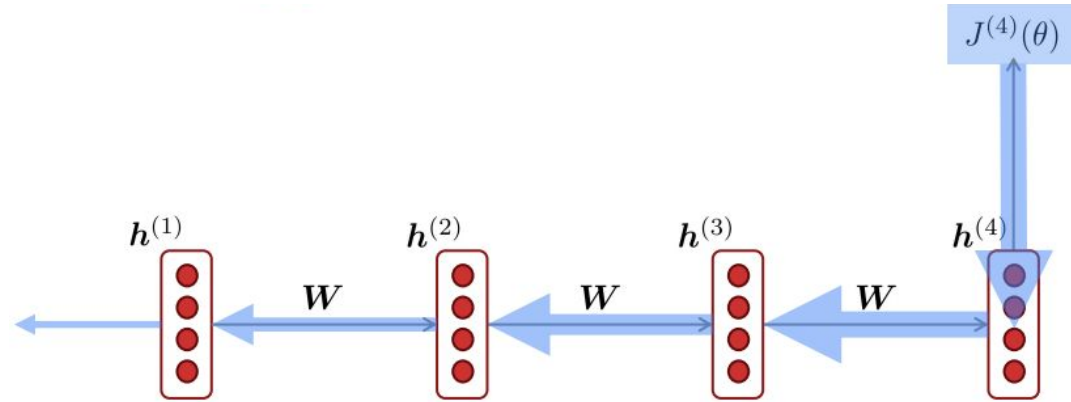


output distribution

$$\hat{y}^{(t)} = \text{softmax}(U h^{(t)} + b_2) \in \mathbb{R}^{|V|}$$

The input sequence can be of arbitrary length.

Vanishing Gradient Problem for RNNs



$$\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(1)}} = \left[\frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}} \right] \times \left[\frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}} \right] \times \left[\frac{\partial \mathbf{h}^{(4)}}{\partial \mathbf{h}^{(3)}} \right] \times \frac{\partial J^{(4)}}{\partial \mathbf{h}^{(4)}}$$

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.

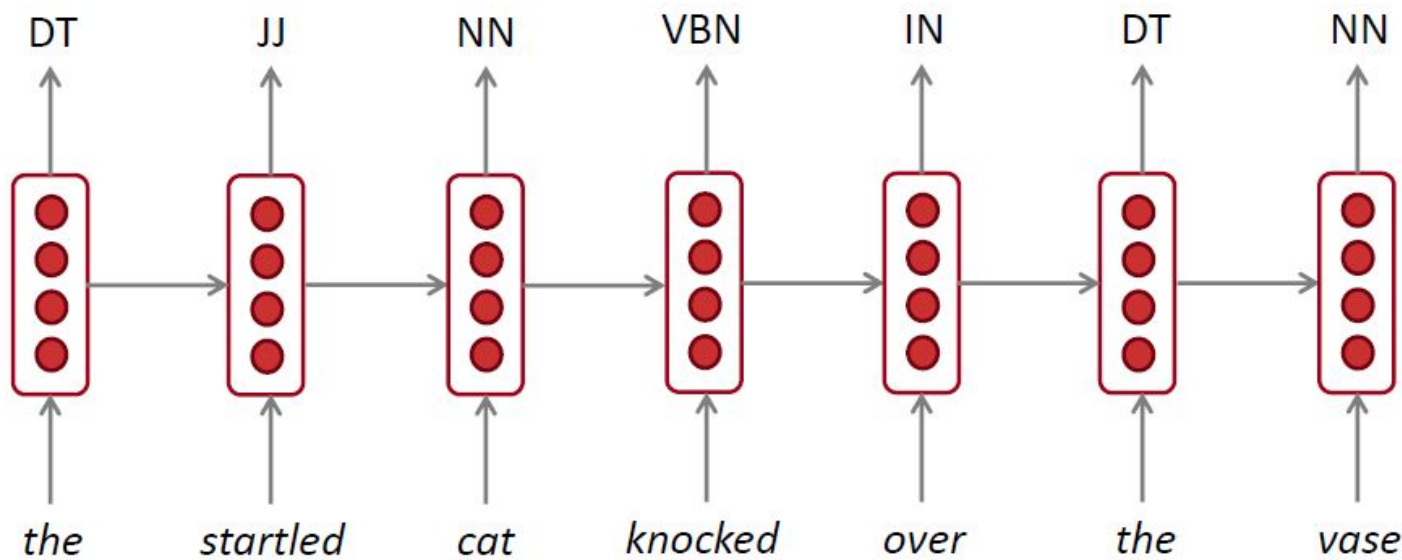
Disadvantages of RNNs

LSTM to the rescue!

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

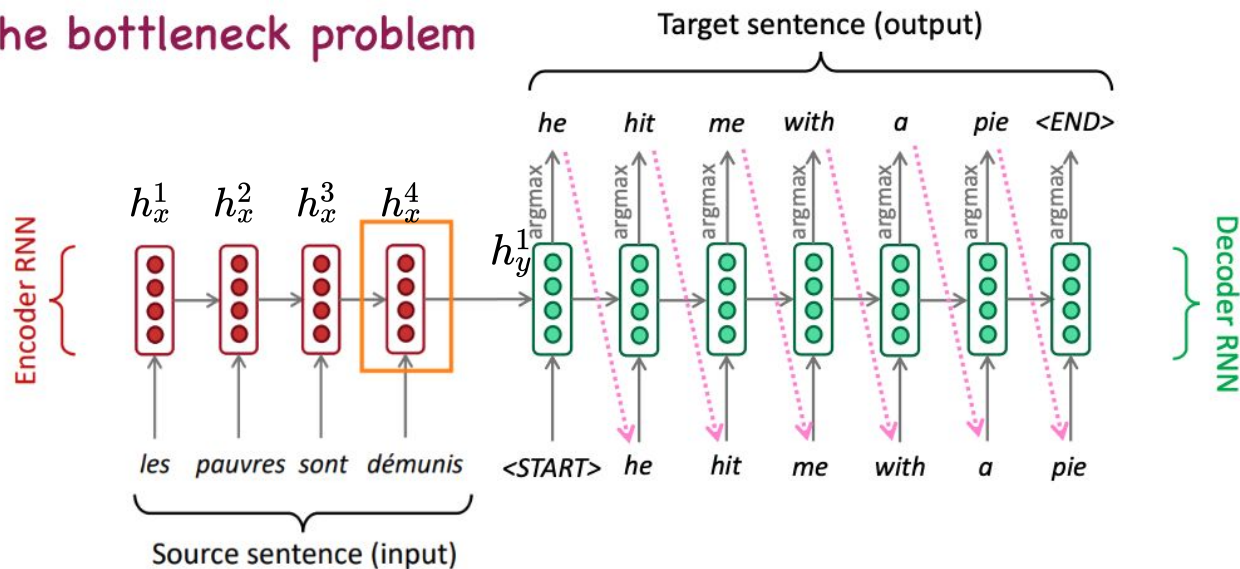
Sequence Tagging with RNNs

- 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!

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

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

Review

ML & DL

Introduction to
machine/deep learning

Transformer

Attention mechanism,
encoder/decoder

Pretraining

Masking, natural
language generation

Word

Word vectors,
language modeling

Sequence

Sequence modeling,
seq2seq learning

Prompting

Prompts, in-context
learning

Attentions

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

$$e^t = [s_t^T h_1, \dots, s_t^T h_N] \in \mathbb{R}^N$$

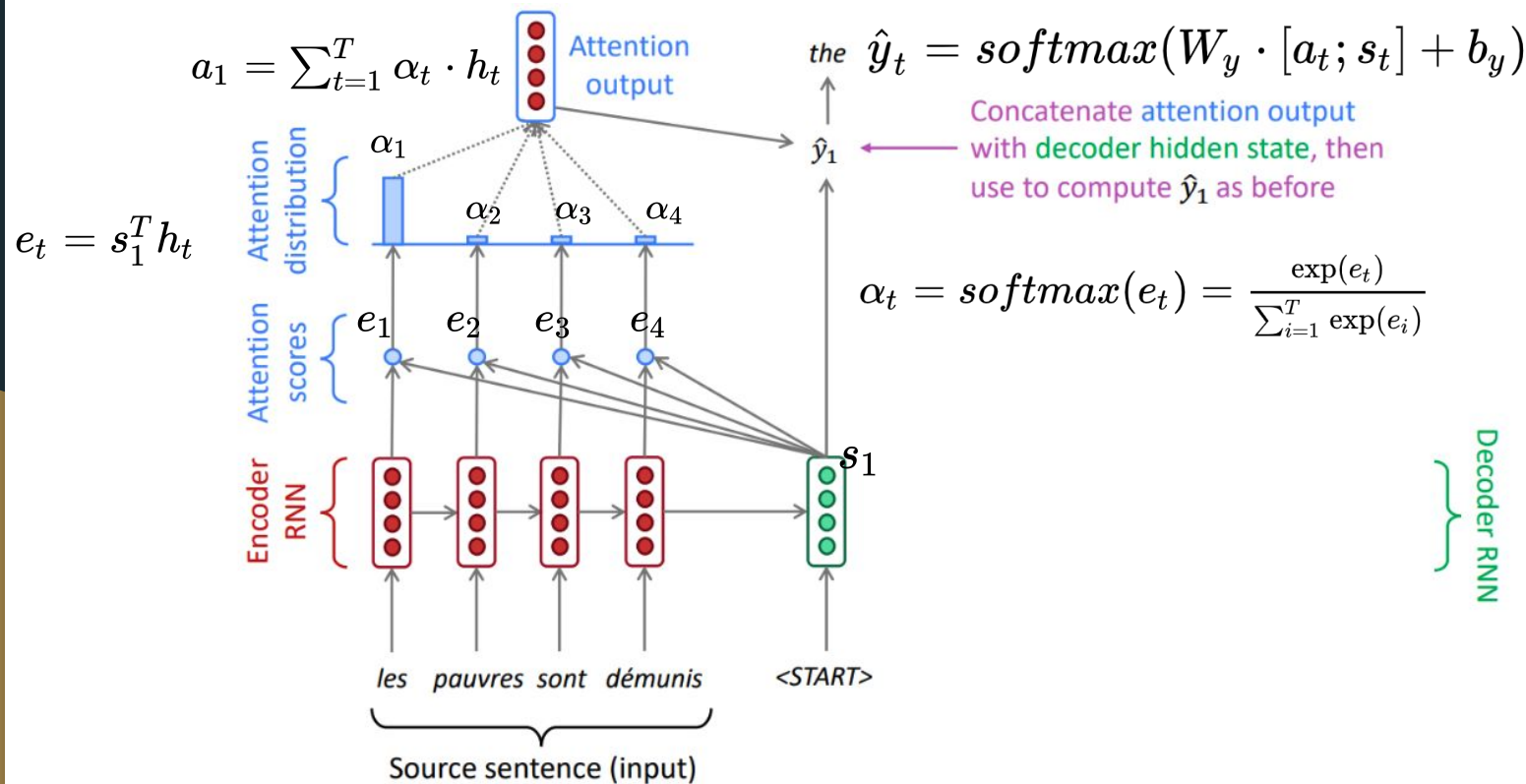
- We take softmax to get the attention distribution α^t for this step (this is a probability distribution and sums to 1)

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

- We use α^t to take a weighted sum of the encoder hidden states to get the attention output a_t

$$a_t = \sum_{i=1}^N \alpha_i^t h_i \in \mathbb{R}^h$$

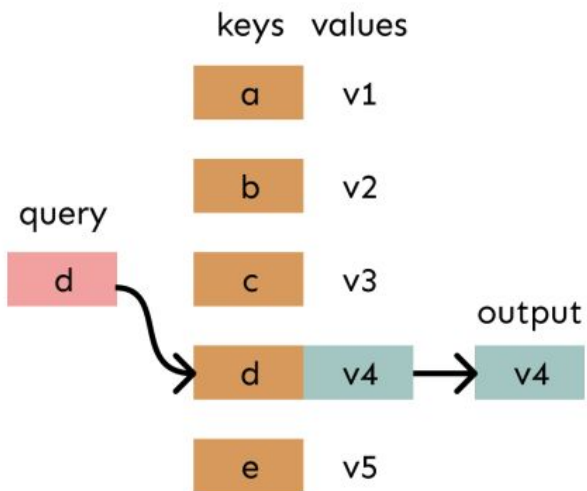
Attentions



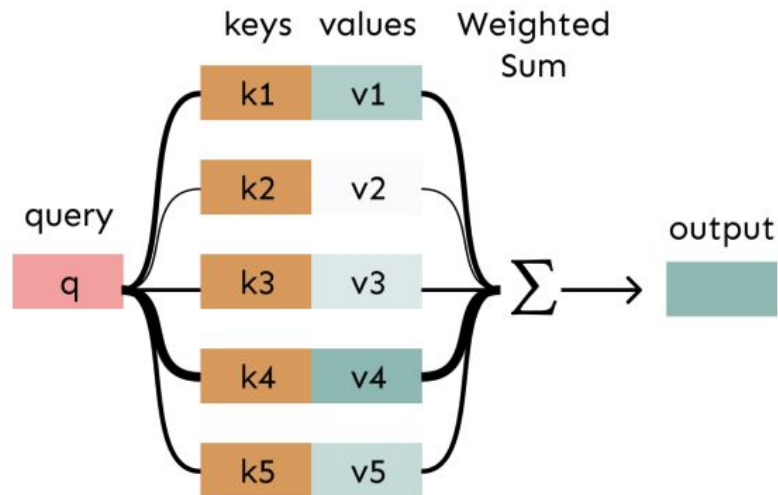
Attentions as QKV

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.

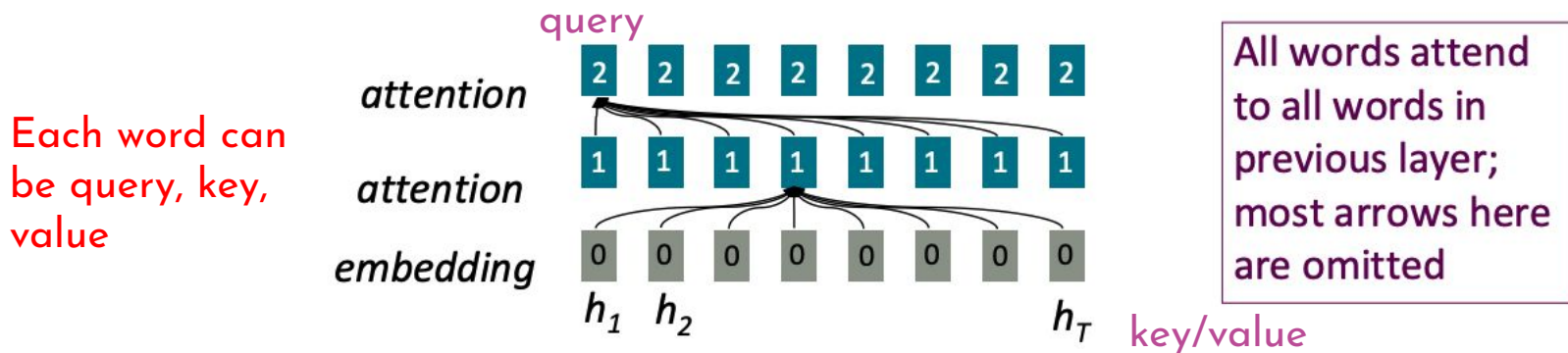


In **attention**, the **query** matches all **keys** *softly*, to a weight between 0 and 1. The keys' **values** are multiplied by the weights and summed.



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!



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

Self Attentions

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

For each w_i , let $x_i = Ew_i$, where $E \in \mathbb{R}^{d \times |V|}$ is an embedding matrix.

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

$$q_i = Qx_i \text{ (queries)} \quad k_i = Kx_i \text{ (keys)} \quad v_i = Vx_i \text{ (values)}$$

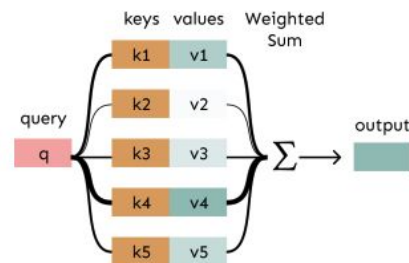
2. Compute pairwise similarities between keys and queries; normalize with softmax

query is fixed

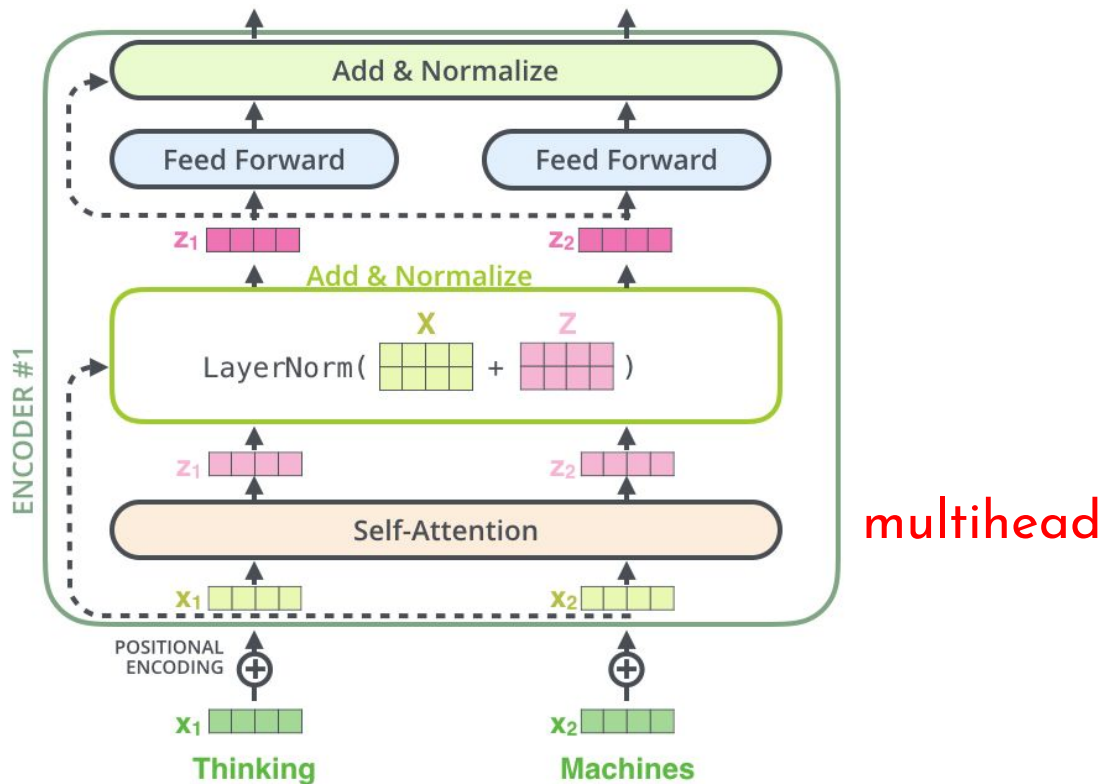
$$e_{ij} = q_i^\top k_j \quad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'} \exp(e_{ij'})}$$

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

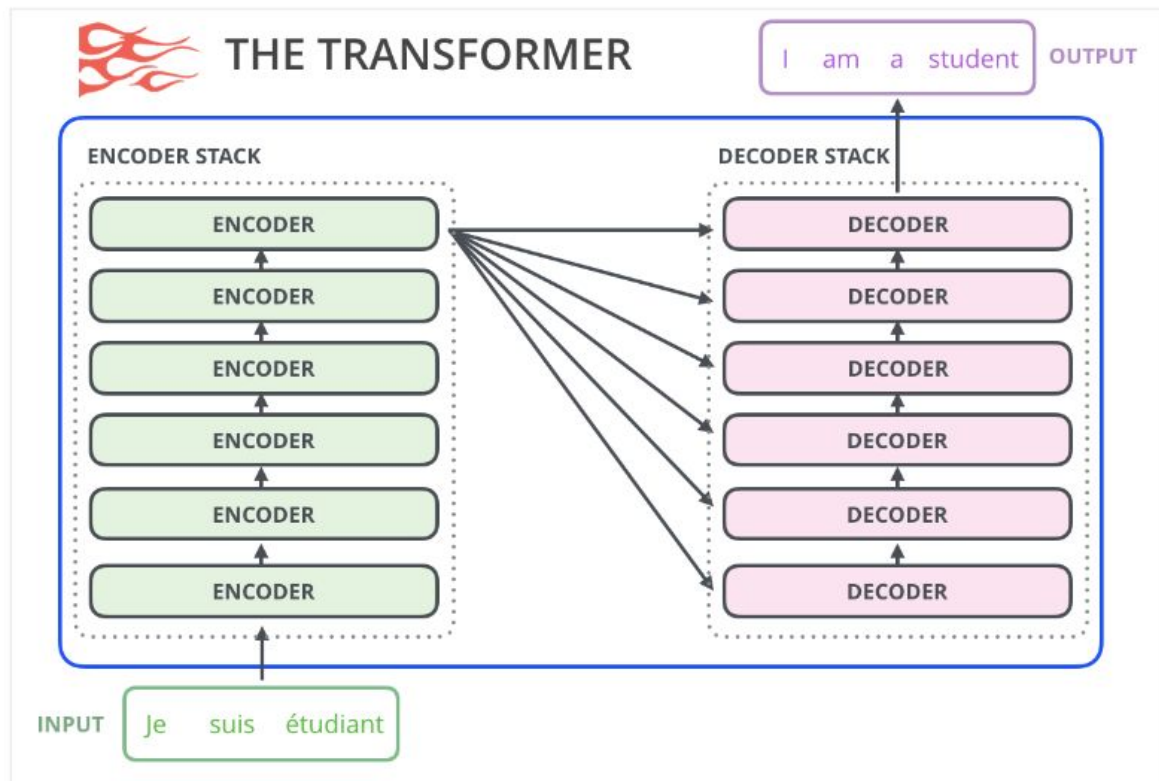
$$o_i = \sum_j \alpha_{ij} v_j$$



From Attentions to Transformers



Transformer Encoder-Decoder



Review

ML & DL

Introduction to
machine/deep learning

Transformer

Attention mechanism,
encoder/decoder

Pretraining

Masking, natural
language generation

Word

Word vectors,
language modeling

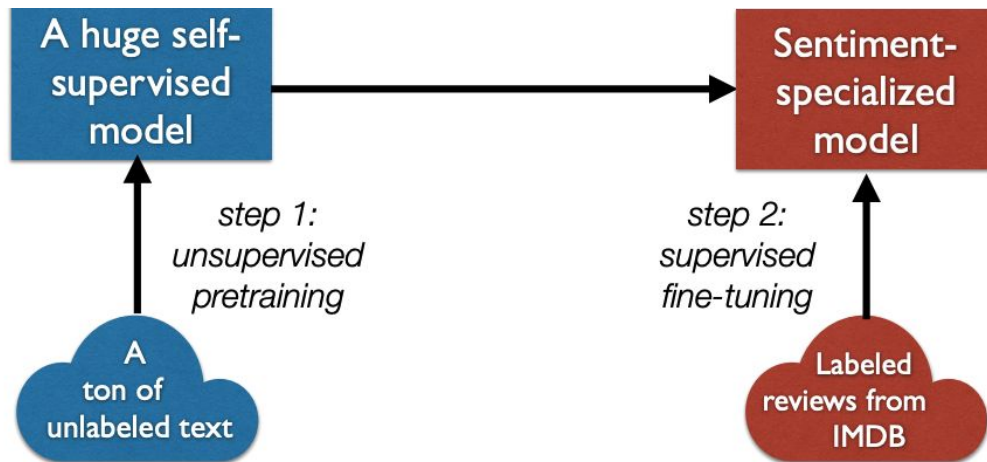
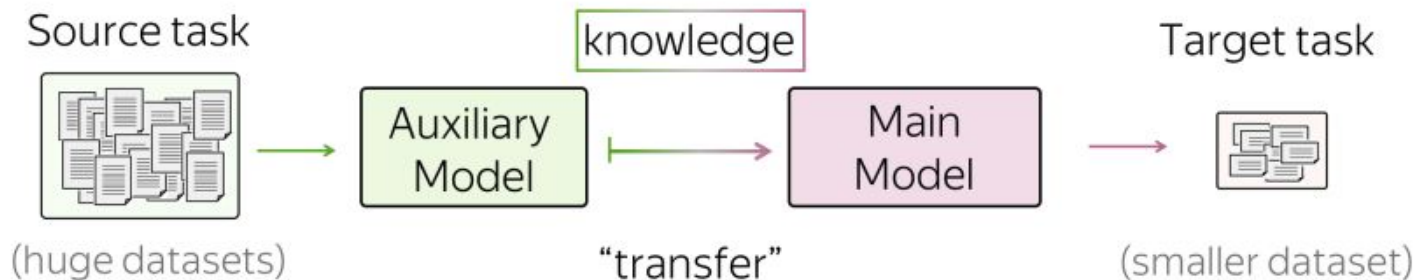
Sequence

Sequence modeling,
seq2seq learning

Prompting

Prompts, in-context
learning

Pre-training and Fine-tuning

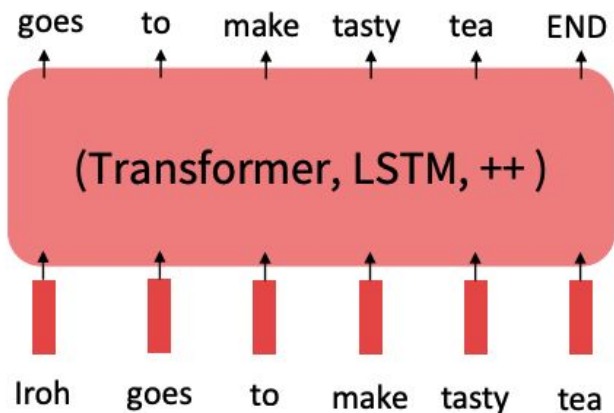


Pre-training to Fine-tuning

Pretraining can improve NLP applications by serving as parameter initialization.

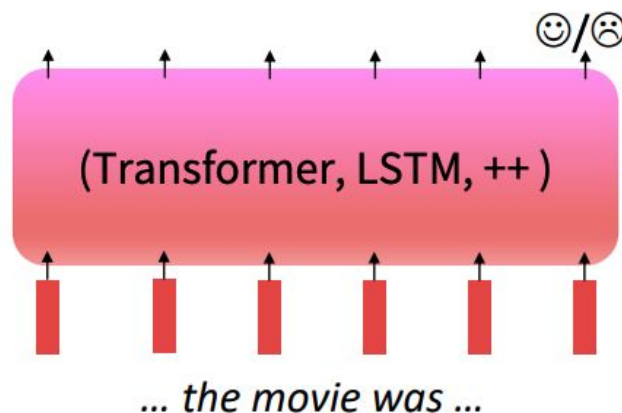
Step 1: Pretrain (on language modeling)

Lots of text; learn general things!

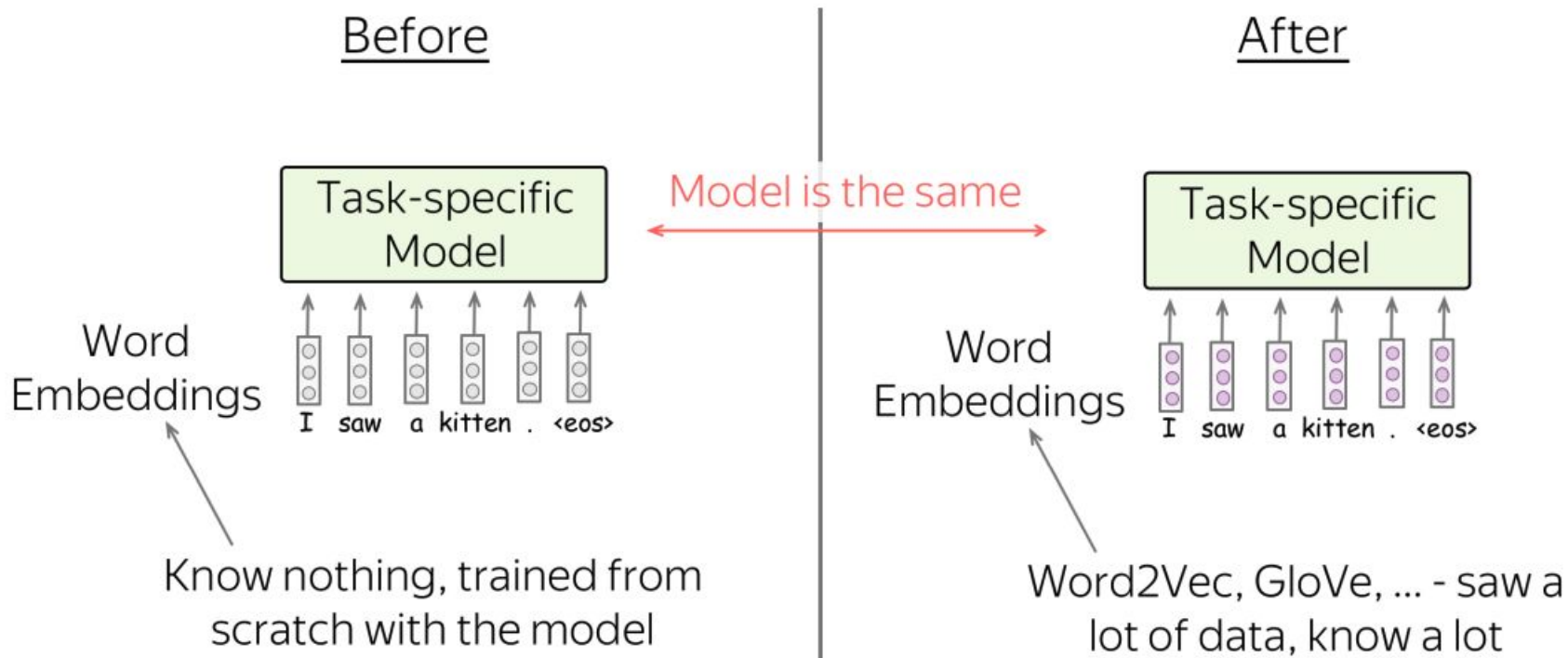


Step 2: Finetune (on your task)

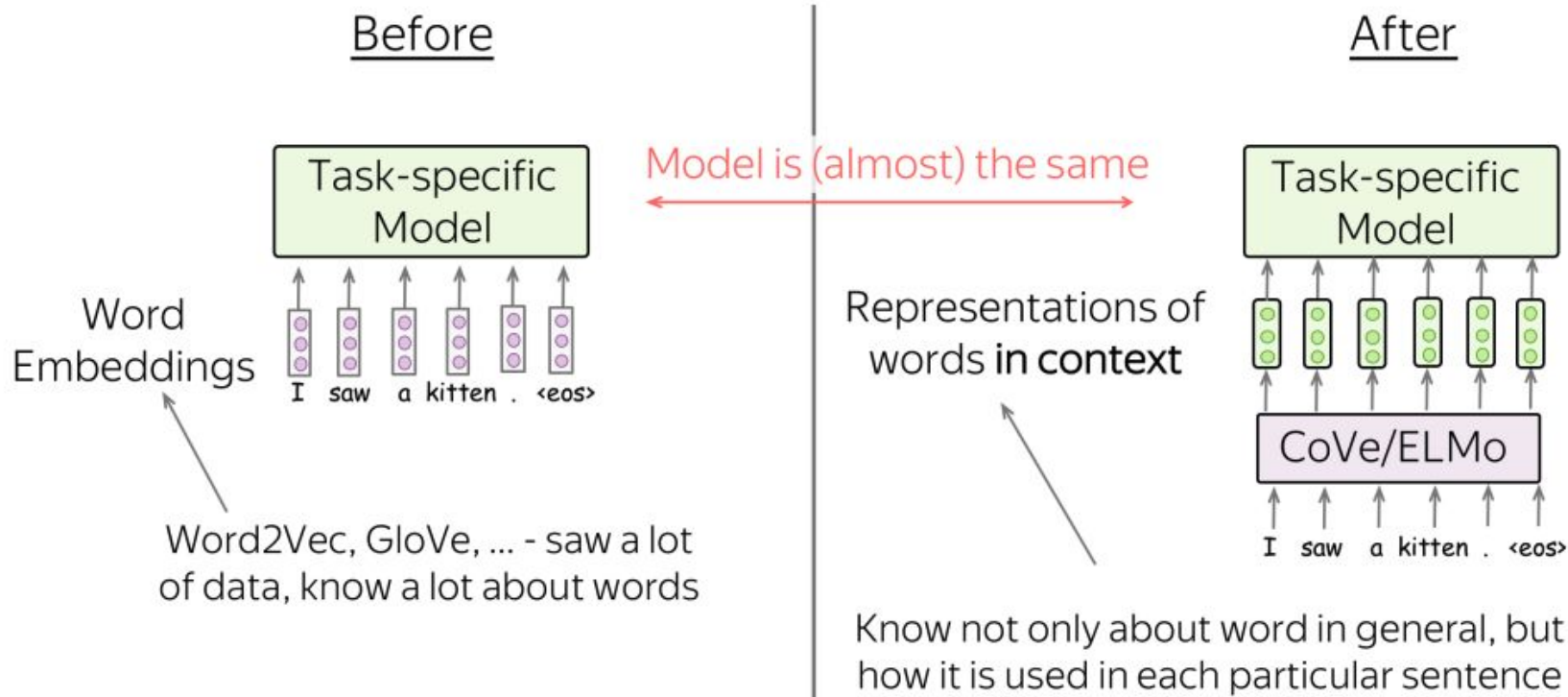
Not many labels; adapt to the task!



Transfer Through Word Embeddings

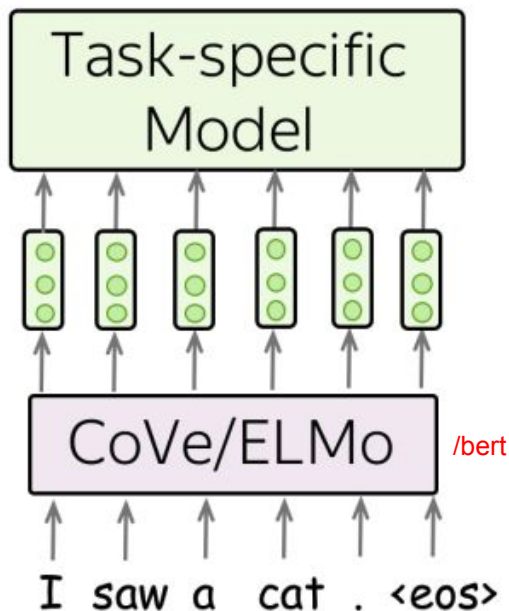


Transferring Words-in-Context



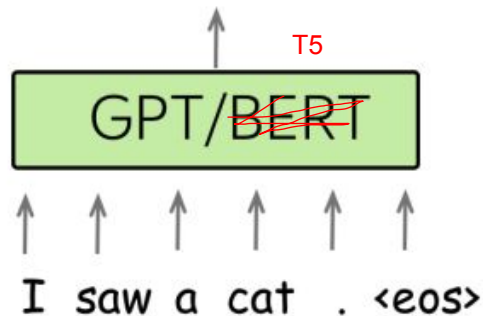
Transferring Entire Models

Before



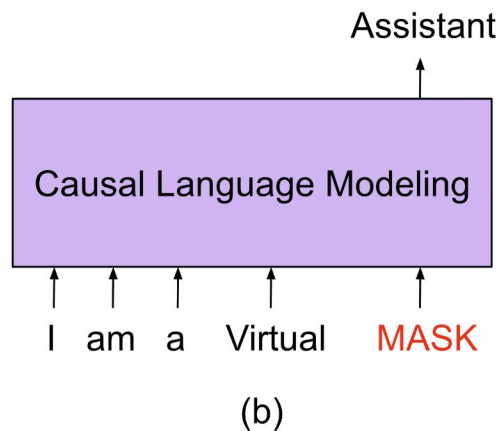
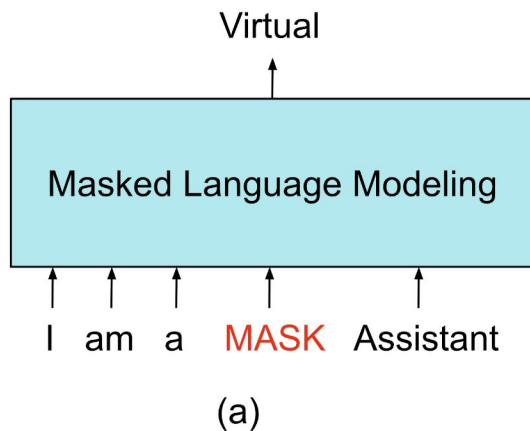
After

No task-specific
models!



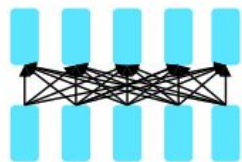
Language Model Pretraining

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



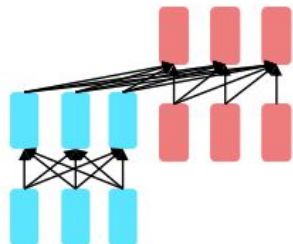
3 Training Paradigms

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



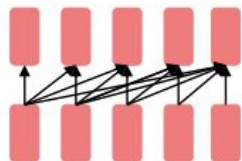
Encoders

- 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

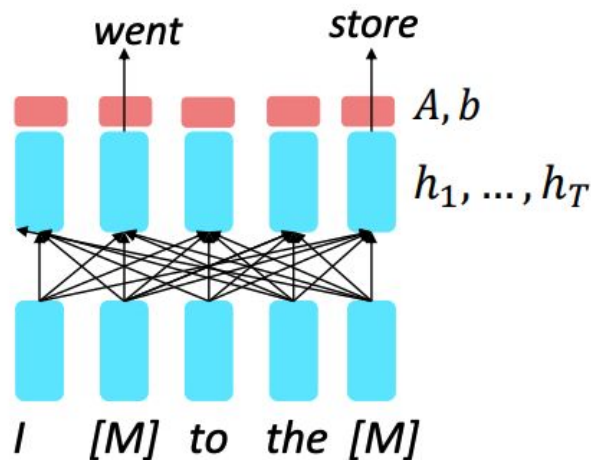
Encoder

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.

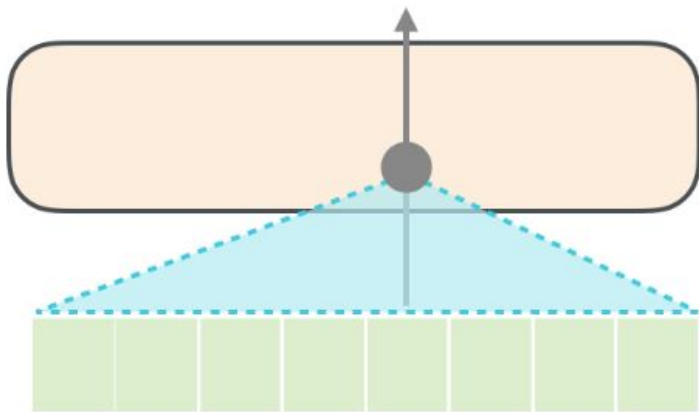
$$h_1, \dots, h_T = \text{Encoder}(w_1, \dots, w_T)$$
$$y_i \sim Aw_i + b$$

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**.

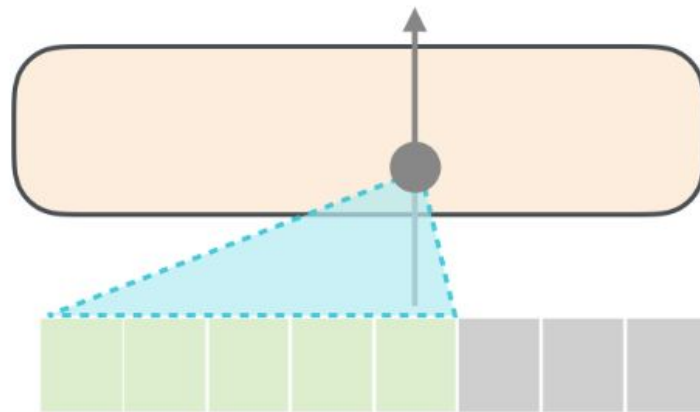


Masked Attentions in Decoder

Self-Attention



Masked Self-Attention



Review

ML & DL

Introduction to
machine/deep learning

Transformer

Attention mechanism,
encoder/decoder

Pretraining

Masking, natural
language generation

Word

Word vectors,
language modeling

Sequence

Sequence modeling,
seq2seq learning

Prompting

Prompts, in-context
learning

Zero-Shot / Few-Shot Prompting

Zero/few-shot prompting

1 Translate English to French: ←
2 sea otter => loutre de mer ←
3 peppermint => menthe poivrée ←
4 plush girafe => girafe peluche ←
5 cheese => ←

Traditional fine-tuning

1 sea otter => loutre de mer ←



gradient update



1 peppermint => menthe poivrée ←



gradient update



1 cheese => ←

The End

Good luck to everyone!!!