

Large Language Models (LLMs)

Dr. Sunil Saumya
Assistant Professor, DSAI
IIIT Dharwad



Course Information

- **Class Timing:** Monday and Thursday 9 to 10:30 AM
- **Class Venue:** Hybrid
 - C004: for B.Tech 6th Sem and Generative AI Minor students
 - Online: <https://meet.google.com/cjs-kxec-gwq>
- **Evaluation:**
 - Mid Sem exam: 20%
 - End Sem exam: 30%
 - Quiz (2): 10%
 - Assignment: 10%
 - Course Project: 30% (group wise)



Course Project

- Some problem statements and datasets will be floated by the end of 2nd week.
- You will have to form the group consists of 3-4 members.
- You need to
 - Develop models, evaluate models, prepare presentation and write technical reports
- You are encouraged to publish your work in a good conferences/journals.
- **Key Deliverables:**
 - Final project report in 8 Pages ACL Latex format (15%)
 - Repo of dataset and source code (5%)
 - Final project presentation (10%)
- **Do not plagiarize!**



Course Content

- In this course we will focus of concept and implementation.
- we will discuss state of art papers about Large Language Models time to time.
- As part of self study you will be asked to read papers and present it in the class.



Course Content

Unit 1: Foundations of Large Language Models

- **Introduction to Language Models**
 - Fundamentals of language models: History, evolution, and significance in NLP
 - Word Embeddings, Neural LMs
- **Transformer Architecture**
 - Transformer basics: Self-attention mechanism, Encoder-Decoder structure
 - Encoder layers and encoder stack
- **Core Components of Transformers**
 - Position encoding, batch normalization, and layer normalization
 - Teacher forcing and masked attention

Course Content

Unit 2: Architectures of Large Language Models

- **Decoder-only Large Language Models**
 - Detailed study of GPT architecture (Causal Language Model)
 - Pre-training vs. fine-tuning for downstream applications
 - Decoding strategies: Greedy, Beam Search, Top-k, and Top-p
- **Encoder-only Large Language Models**
 - BERT architecture (Masked Language Model) and training objectives
 - Applications and adaptation for various NLP tasks
- **Tokenization Techniques**
 - Sub-word tokenization, Byte Pair Encoding, WordPiece, and SentencePiece
- **Introduction to Small Language Models**
 - Overview and applications of Small Language Models in resource-constrained settings

Course Content

Unit 3: Advanced Architectures and Adaptation of LLMs

- **Encoder-Decoder Models**
 - Introduction to models like BART and T5
 - Understanding the Text-to-Text framework and zero-shot learning
- **Advanced Model Components**
 - Pre-training strategies, scaling laws, and instruction fine-tuning
 - Advanced attention mechanisms and the Mixture of Experts approach
- **Parameter-Efficient Fine-Tuning (PEFT)**
 - Techniques for efficient adaptation and inference
 - Advantages of PEFT in optimizing LLMs for specific tasks
- **Efficient Model Adaptation**
 - Retrieval and tool augmentation for enhanced model capabilities
 - Prompt Engineering, Chains, Memory and Agents

Course Content

Unit 4: LLMs in Practice and Future Directions

- **Addressing Challenges in LLMs**
 - Bias, toxicity, hallucination, and alignment in LLMs
 - Interpreting LLMs: Understanding the inner workings and output reasoning
- **Specialized Models and Emerging Trends**
 - Multimodal LLMs
 - Vision-Language models and long-context LLMs
 - Model editing, self-evolving LLMs, and efficient inference
- **Future of LLMs**
 - Ethical implications, evolving capabilities, and emerging applications in LLMs

Pre-requisites

- Excitement about language
- Willingness to learn
- **Desirables**
 - Machine Learning
 - Deep Learning
- **This course will not cover**
 - Coding practice
 - Generative models for modalities other than text

Reading and Reference Materials

Text books:

- Introduction to Large Language Models (Generative AI for Text), Tonmoy Chakraborty, Wiley, 2024.
- Hands-On Large Language Models, Jay Alammar and Maarten Grootendorst, 1st edition, O'Reilly Media, 2024.

Other resources:

- <https://www.cse.iitm.ac.in/~miteshk/llm-course.html>
- https://onlinecourses.nptel.ac.in/noc25_cs45/preview
- <https://stanford-cs324.github.io/winter2022/lectures/introduction/>

Many more online resources.

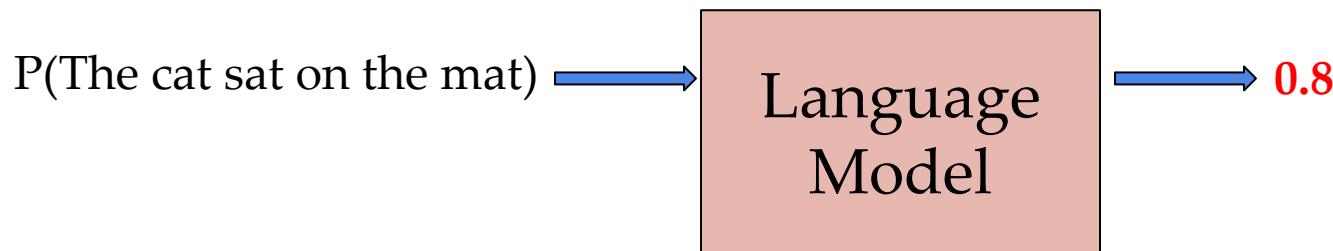


What is Language Model?

- Language model is a probability distribution over sequences of tokens.
 - Language model assigns each sequence of tokens a probability.
 - Probability intuitively tells us how “**good**” a sequence of tokens is.

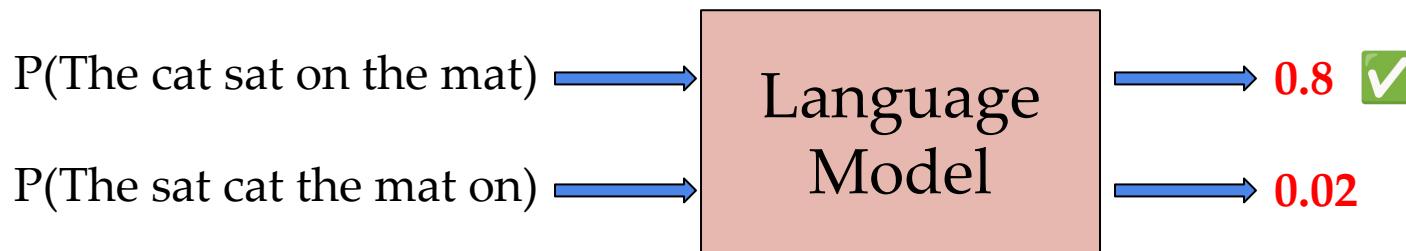
What is Language Model?

- Language model is a probability distribution over sequences of tokens.
 - Language model assigns each sequence of tokens a probability.
 - Probability intuitively tells us how “**good**” a sequence of tokens is.



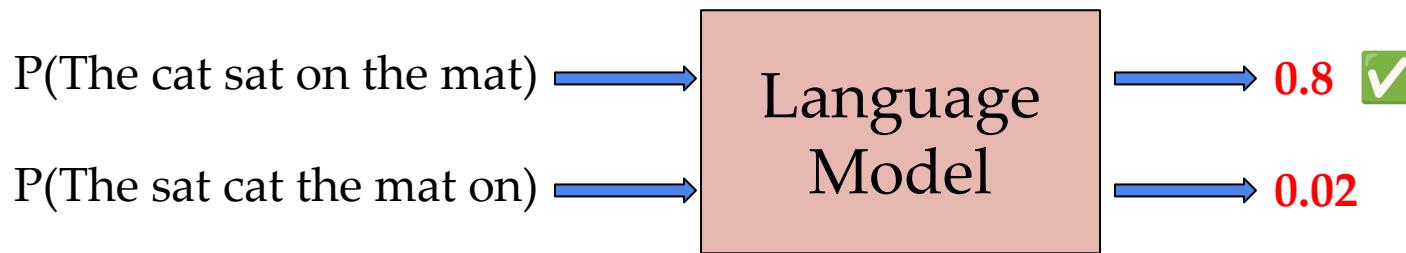
What is Language Model?

- Language model is a probability distribution over sequences of tokens.
 - Language model assigns each sequence of tokens a probability.
 - Probability intuitively tells us how “**good**” a sequence of tokens is.



What is Language Model?

- Language model is a probability distribution over sequences of tokens.
 - Language model assigns each sequence of tokens a probability.
 - Probability intuitively tells us how “**good**” a sequence of tokens is.



- Language Model understands **Syntax, Semantics and word knowledge**.

What is Language Model?

- Suppose we have a vocabulary V of a set of tokens.

$$V = \{\text{ate}, \text{ball}, \text{cheese}, \text{mouse}, \text{the}\}$$

the language model might assign:

$p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) = 0.02$, → **Word Knowledge**

$p(\text{the}, \text{cheese}, \text{ate}, \text{the}, \text{mouse}) = 0.01$, → **No word knowledge**

$p(\text{mouse}, \text{the}, \text{the}, \text{cheese}, \text{ate}) = 0.0001$. → **Syntactically wrong**

Language Model: Generation

- As defined, a language model p takes a sequence and returns a probability to assess its goodness.
- We can also **generate** a sequence given a language model.

$$p(\text{the}, \text{mouse}, \text{ate}, \text{the}, \text{cheese}) = p(\text{the}) \\ p(\text{mouse} | \text{the}) \\ p(\text{ate} | \text{the}, \text{mouse}) \\ p(\text{the} | \text{the}, \text{mouse}, \text{ate}) \\ p(\text{cheese} | \text{the}, \text{mouse}, \text{ate}, \text{the})$$

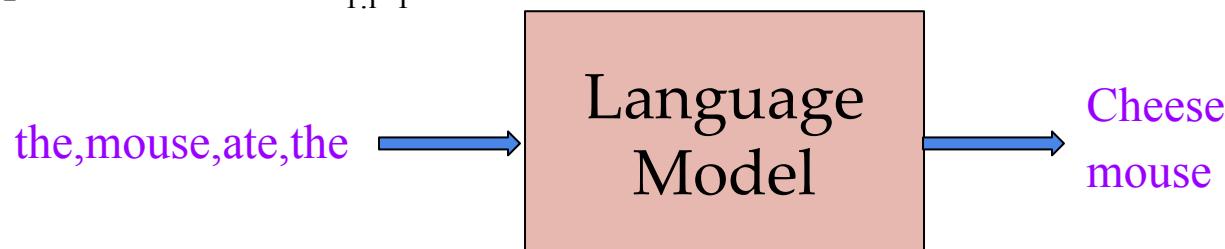
- This is “**Autoregressive language models**”.

Autoregressive Language Model

- Mathematically, autoregressive language model can be represented as:

$$p(x_{1:L}) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \cdots p(x_L | x_{1:L-1}) = \prod_{i=1}^L p(x_i | x_{1:i-1})$$

- In particular, $p(x_i | x_{1:i-1})$ is a conditional probability distribution of the next token x_i given the previous tokens $x_{1:i-1}$.

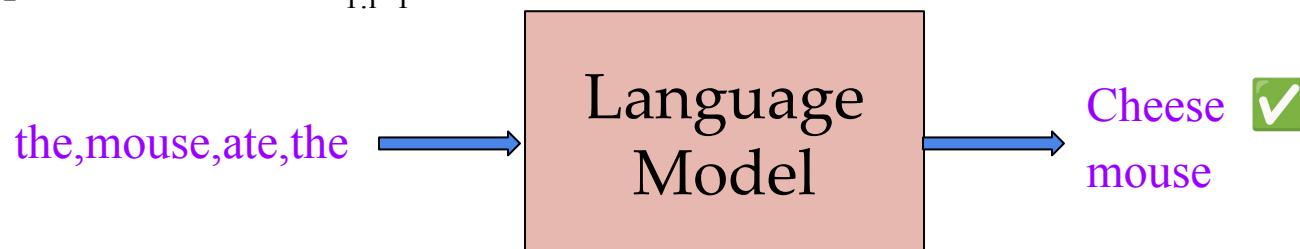


Autoregressive Language Model

- Mathematically, autoregressive language model can be represented as:

$$p(x_{1:L}) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \cdots p(x_L | x_{1:L-1}) = \prod_{i=1}^L p(x_i | x_{1:i-1})$$

- In particular, $p(x_i | x_{1:i-1})$ is a conditional probability distribution of the next token x_i given the previous tokens $x_{1:i-1}$.



Language Model Applications

- Text Generation
- Machine Translation
- Chatbots and Virtual Assistants
- Text Summarization
- Search and Information Retrieval
- Question- Answering

And so on..

Markov Language Model

Idea: Next state depends only on the current state.

- Consider the dataset sample:

“The cat sat on the mat.”

“The dog chased the ball.”

“The bird sang a beautiful song.”

Markov Language Model

Idea: Next state depends only on the current state.

- Consider the dataset sample: **Step 1: States**

"The cat sat on the mat."

S: Start (beginning of sentence)

"The dog chased the ball."

N: Noun

"The bird sang a beautiful song."

V: Verb

Adj: Adjective

E: Sentence End

Preprocessing: Remove stop words *the, on, a*

Markov Language Model

Idea: Next state depends only on the current state.

- Consider the dataset sample:

“ **S** cat sat mat **E** ”

“ **S** dog chased ball **E** ”

“ **S** bird sang beautiful song **E** ”

Step 2: Transition Probabilities

	N	V	Adj	E
S	3	0	0	0
N	0	3	0	3
V	2	0	1	0
Adj	1	0	0	0

Frequency table

Markov Language Model

Idea: Next state depends only on the current state.

- Consider the dataset sample:

“ *s cat sat mat e* ”

“ *s dog chased ball e* ”

“ *s bird sang beautiful song e* ”

Step 2: Transition Probabilities

	N	V	Adj	E
S	3/3	0	0	0
N	0	3/6	0	3/6
V	2/3	0	1/3	0
Adj	1/1	0	0	0

Probability table

Markov Language Model

- Consider the dataset sample:

“ **S** cat sat mat **E** ”

“ **S** dog chased ball **E** ”

“ **S** bird sang beautiful song **E** ”

Frequency table

	N	V	Adj
cat	1	0	0
sat	0	1	0
mat	1	0	0
dog	1	0	0
chased	0	1	0
ball	1	0	0
bird	1	0	0
sang	0	1	0
beautiful	0	0	1
song	1	0	0

Step 3: Emission Probabilities

Markov Language Model

- Consider the dataset sample:

“ **S** cat sat mat **E** ”

“ **S** dog chased ball **E** ”

“ **S** bird sang beautiful song **E** ”

Step 3: Emission Probabilities

Probability table

	N	V	Adj
cat	1/6	0	0
sat	0	1/3	0
mat	1/6	0	0
dog	1/6	0	0
chased	0	1/3	0
ball	1/6	0	0
bird	1/6	0	0
sang	0	1/3	0
beautiful	0	0	1/1
song	1/6	0	0

Emission Probabilities

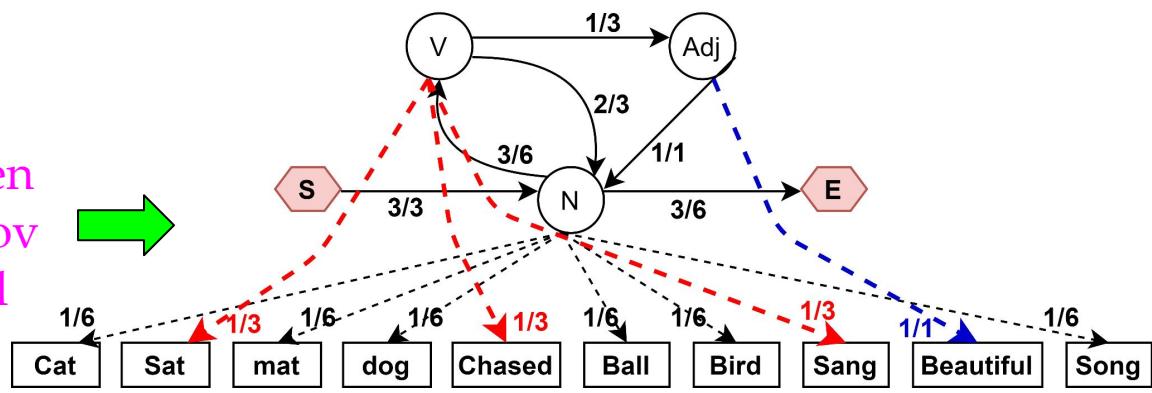
	N	V	Adj
cat	1/6	0	0
sat	0	1/3	0
mat	1/6	0	0
dog	1/6	0	0
chased	0	1/3	0
ball	1/6	0	0
bird	1/6	0	0
sang	0	1/3	0
beautiful	0	0	1/1
song	1/6	0	0

Markov Language Model

Transition Probabilities

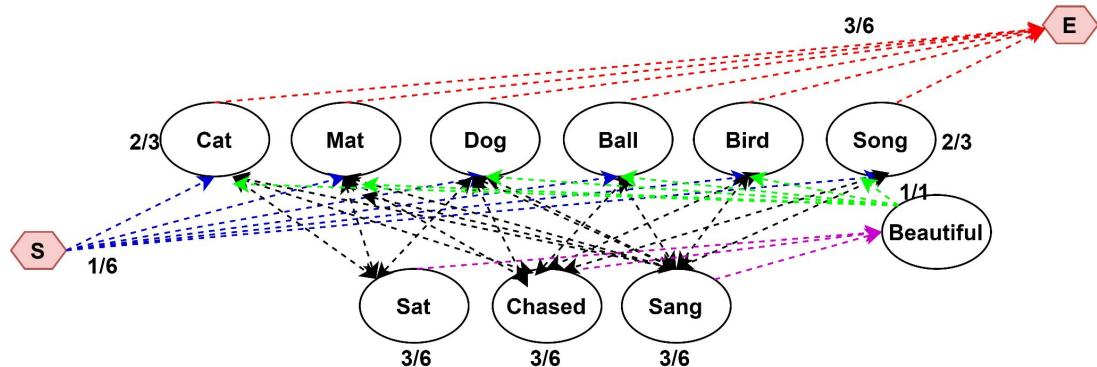
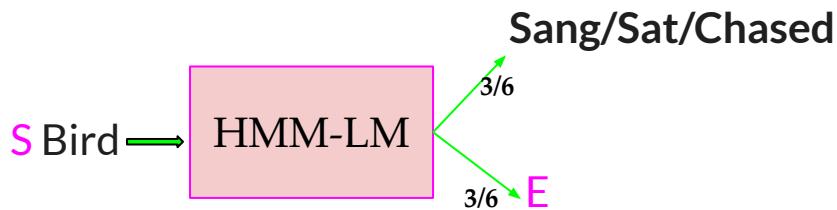
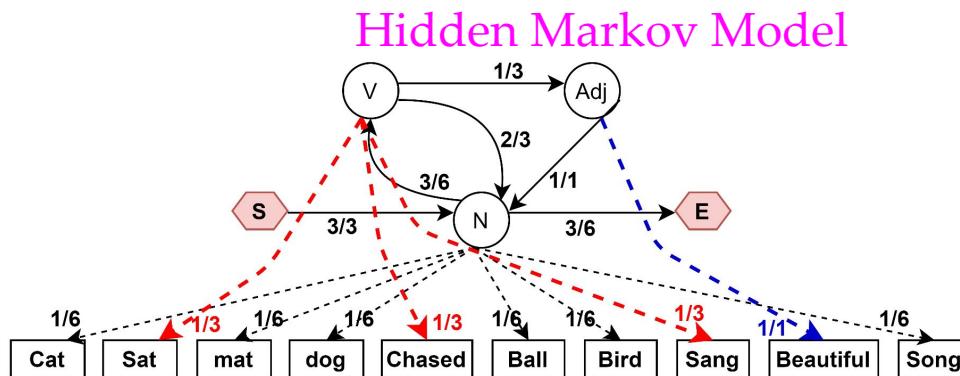
	N	V	Adj	E
S	3/3	0	0	0
N	0	3/6	0	3/6
V	2/3	0	1/3	0
Adj	1/1	0	0	0

Hidden
Markov
Model



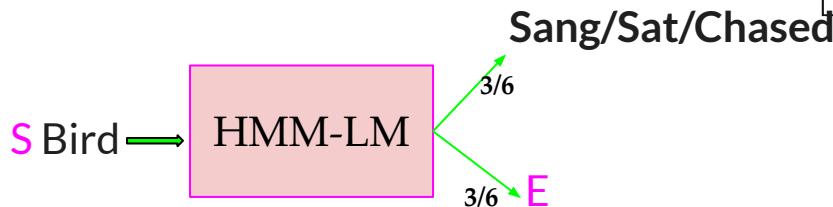
Markov Language Model

Test sample: S Bird ----?



Markov Language Model

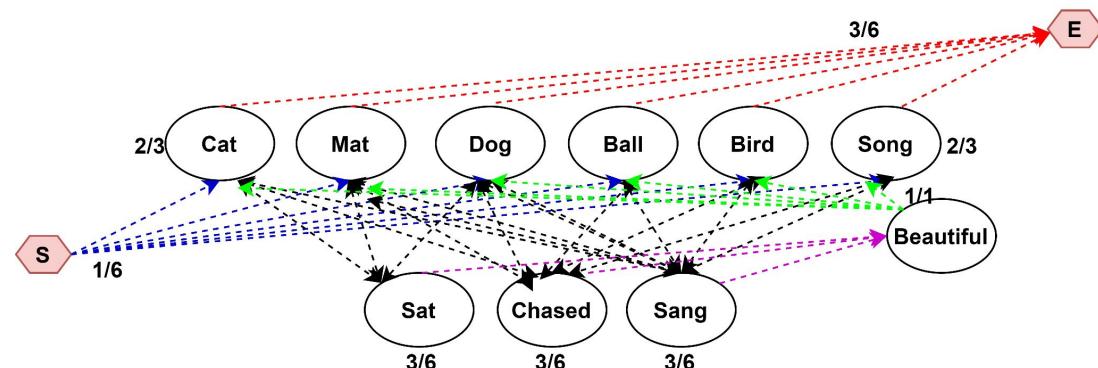
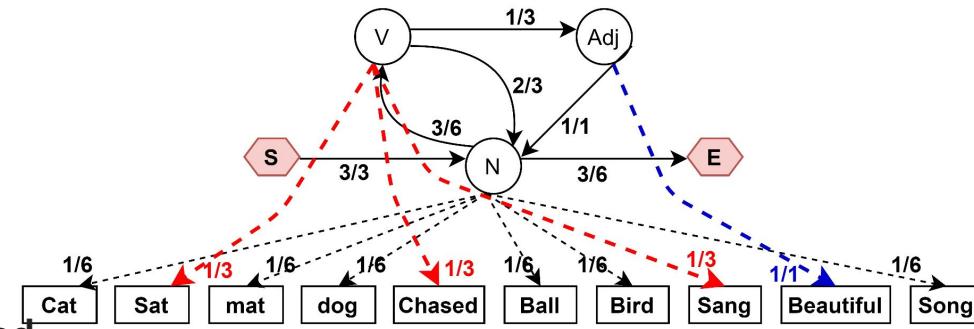
Test sample: S Bird ----?



S Bird Sang/Sat/Chased
Or
S Bird E

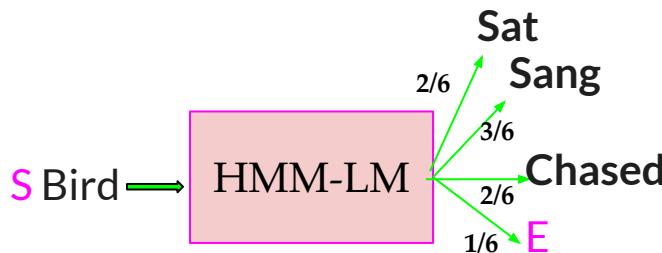


Hidden Markov Model



Markov Language Model

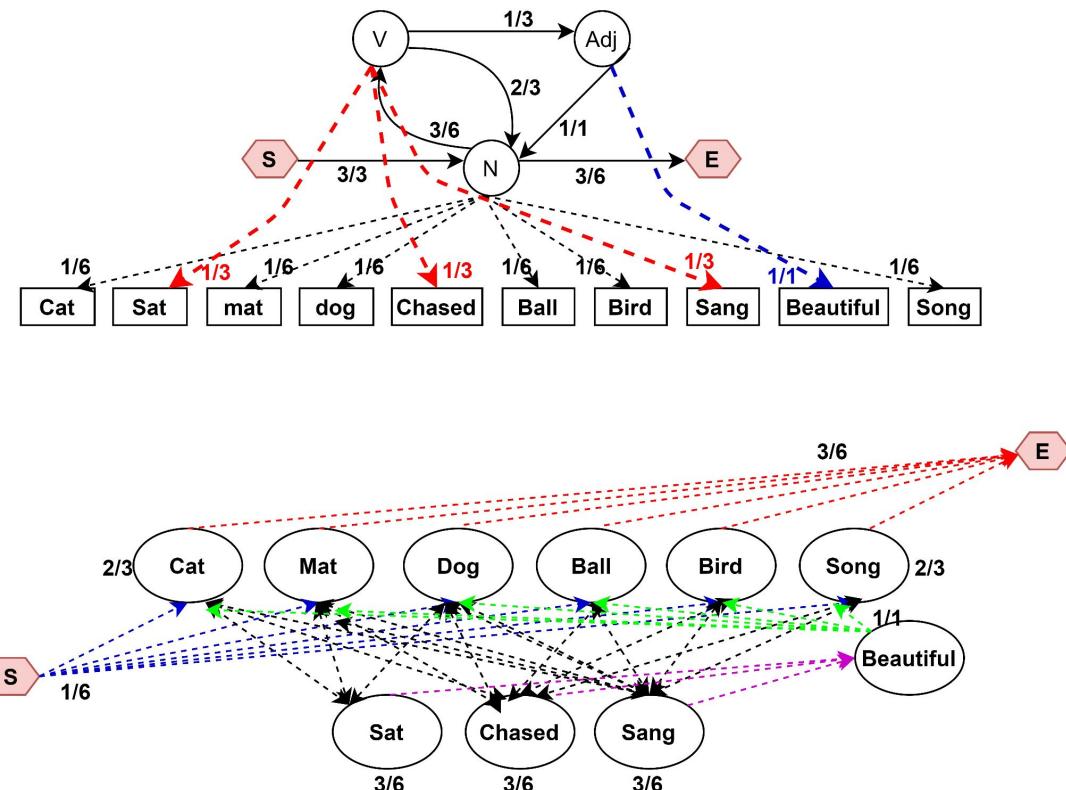
Test sample: S Bird ----?



The prediction will be:

S Bird Sang

Hidden Markov Model



N-gram Language Model

Probability of a word depends only on the previous word(s) is called Markov assumption.

- Bigram: a first-order Markov model
- Trigram: a second-order Markov model;
- N-Gram: a N-1 order Markov model.

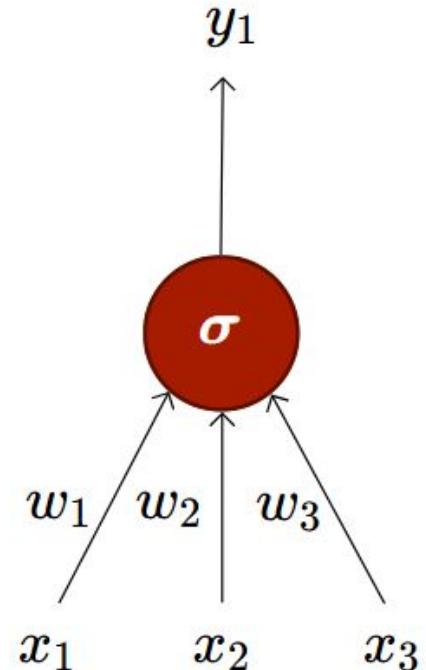
Demo code for Next Word Prediction using HMM

Neural Language Model

The most fundamental unit of a neural language model is **artificial** neuron.

- Why it is called as neuron? Where does the inspiration come from?
- The inspiration comes from the biology (more specifically from the brain).

Biological neurons = neural cells = neural processing units



<https://web.stanford.edu/%7Ejurafsky/slp3/7.pdf>

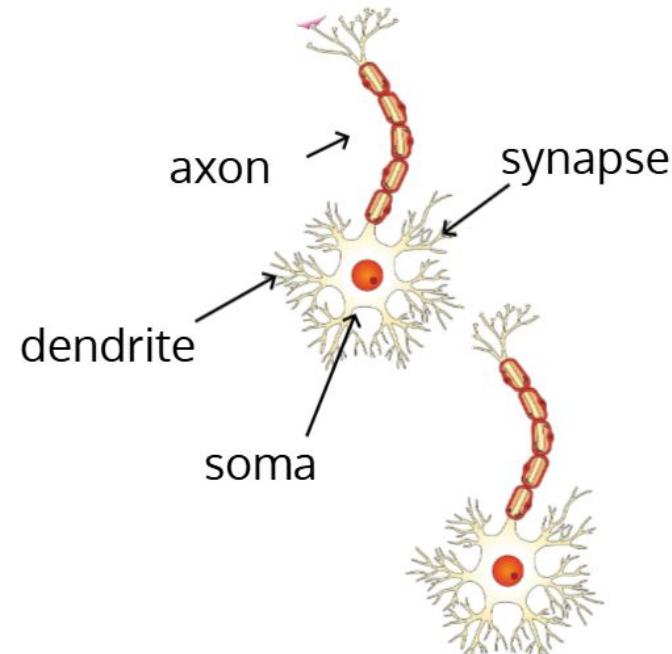
Biological Neurons

Dendrite: receives signals from other neurons

Synapse: point of connection to other neurons

Soma: process the information

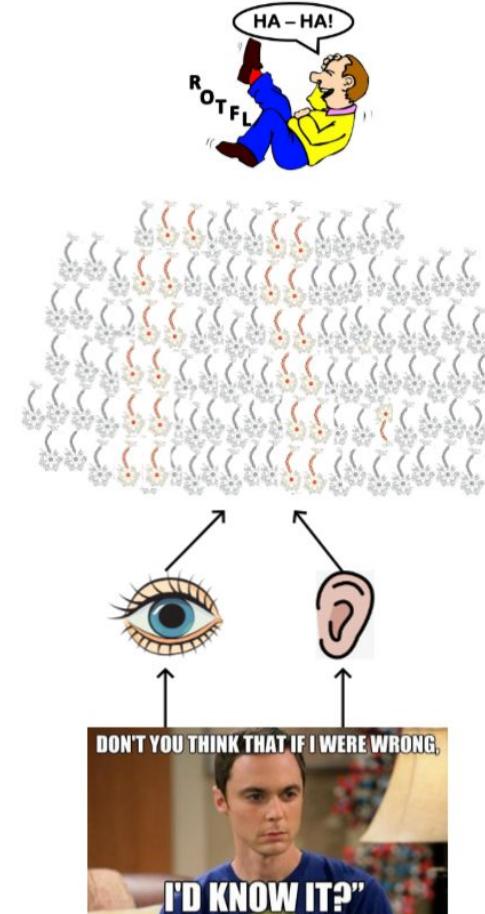
Axon: Transmits the output of this neuron



<https://cdn.vectorstock.com/i/composite/12,25/neuron-cell-vector-81225.jpg>

Biological Neurons

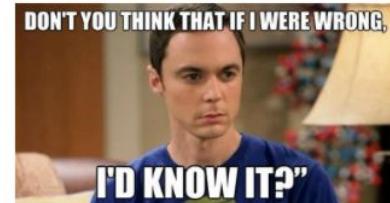
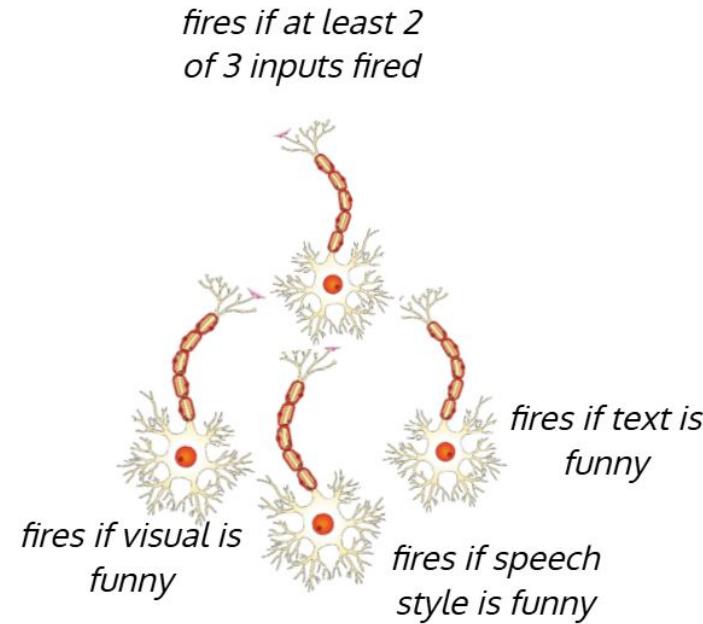
- Of course, in reality, there is not a single neuron which does all this.
- There is a massively parallel interconnected network of neurons.
- These neurons may fire (**in red**) and the process continues eventually resulting in a response (laughter in this case).
- An average human brain has around 10^{11} (100 billion) neurons!



https://iitm-pod.slides.com/arunprakash_ai/cs6910-lecture-2/fullscreen#/0/26/1

Biological Neurons

- This massively parallel network also ensures that there is division of work.
- Each neuron may perform a certain role or respond to a certain stimulus.
- The neurons in the brain are arranged in a hierarchy



Perceptron working

Suppose inputs,
 $x_1 = 0.2, x_2 = 0.4, x_3 = 0.6$

And weights
 $w_1 = -0.3, w_2 = 0.4, w_3 = -0.8$

Activation function:

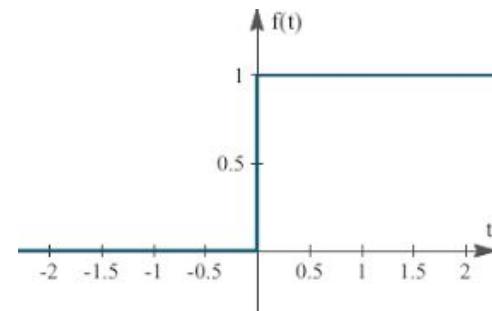
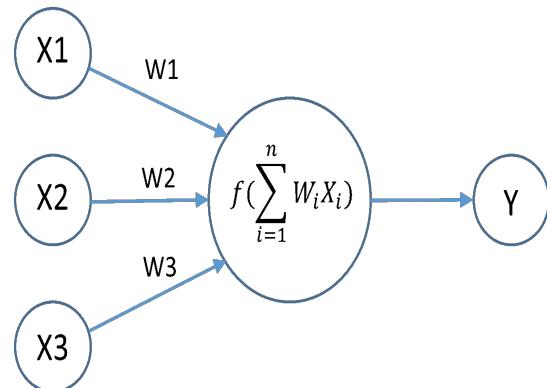
Step Function (threshold)

1 if $w_i x_i > 0$
and
0 if $w_i x_i \leq 0$

Weighted sum:

$$\begin{aligned} &x_1 w_1 + x_2 w_2 + x_3 w_3 \\ &= 0.2 * -0.3 + 0.4 * 0.4 + 0.6 * -0.8 \\ &= -0.06 + 0.16 - 0.48 \\ &= -0.38 \end{aligned}$$

$$\begin{aligned} Y &= f(x_1 w_1 + x_2 w_2 + x_3 w_3) \\ &= f(-0.38) = 0 \end{aligned}$$



Step Function

Perceptron working

Suppose inputs,
 $x_1 = 0.2, x_2 = 0.4, x_3 = 0.6,$
 $b = 0.5$

And weights
 $w_1 = -0.3, w_2 = 0.4, w_3 = -0.8$

Activation function:

Step Function (threshold)

1 if $w_i x_i > 0$
and
0 if $w_i x_i \leq 0$

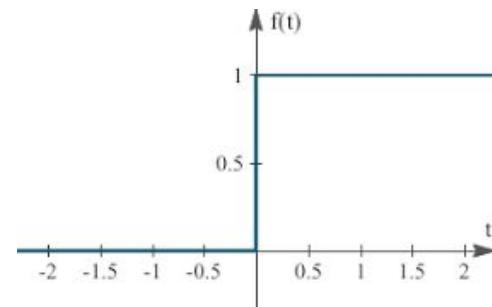
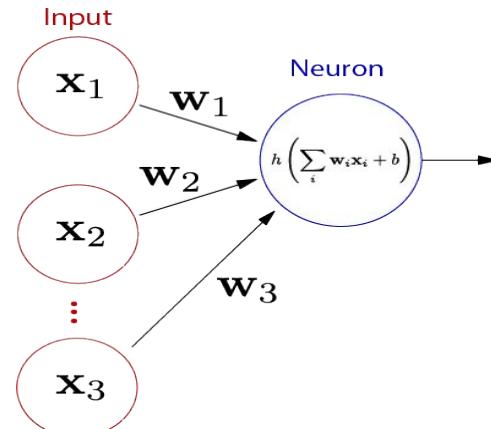
Weighted sum:

$$\begin{aligned} &x_1 w_1 + x_2 w_2 + x_3 w_3 \\ &= 0.2 * -0.3 + 0.4 * 0.4 + 0.6 * -0.8 \\ &= -0.06 + 0.16 - 0.48 \\ &= \mathbf{-0.38} \end{aligned}$$

$$\begin{aligned} Y &= f(x_1 w_1 + x_2 w_2 + x_3 w_3) \\ &= f(-0.38) = 0 \end{aligned}$$

New output

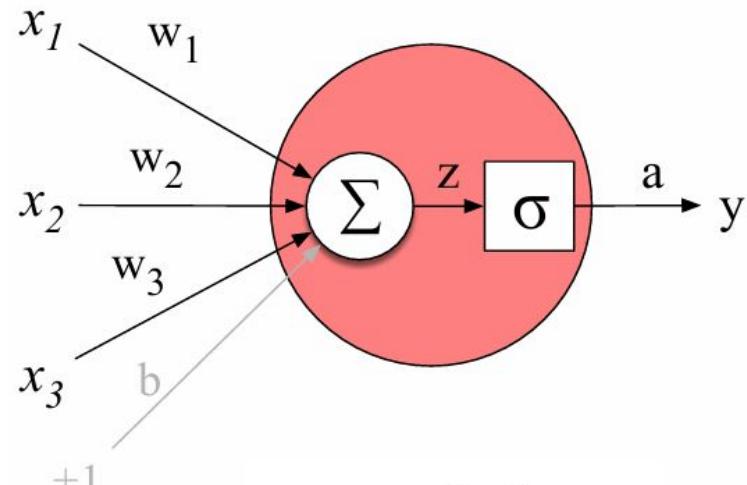
$$\begin{aligned} Y &= f(x_1 w_1 + x_2 w_2 + x_3 w_3 + b) \\ Y &= f(-0.38 + 0.5) \\ Y &= f(0.12) = 1 \end{aligned}$$



Step Function

Artificial Neural Units

- The building block of a neural network is a single computational unit.
- A unit takes a set of real valued numbers as input, performs some computation on them, and produces an output.
- A neural unit is taking a **weighted sum of its inputs**, with one additional term in the sum called a **bias** term.



$$z = b + \sum_i w_i x_i$$

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

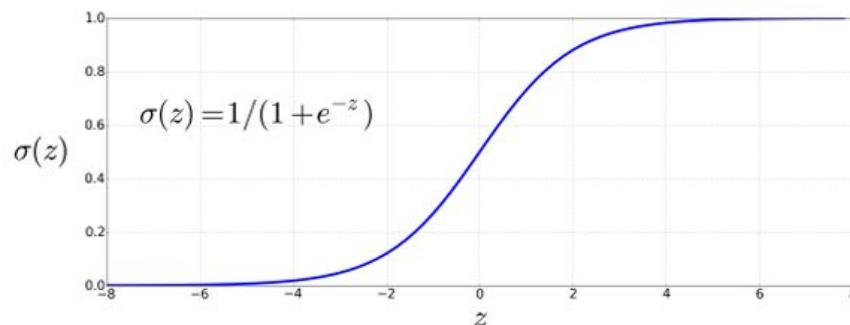
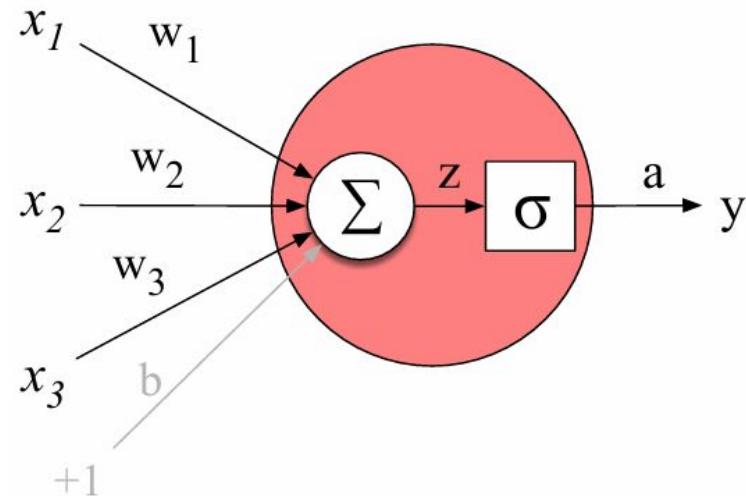
Artificial Neural Units

- Let's suppose we have a unit with the following weight vector and bias:

$$\mathbf{x} = [0.5, 0.6, 0.1], \mathbf{w} = [0.2, 0.3, 0.9], \text{ and } b = 0.5$$

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

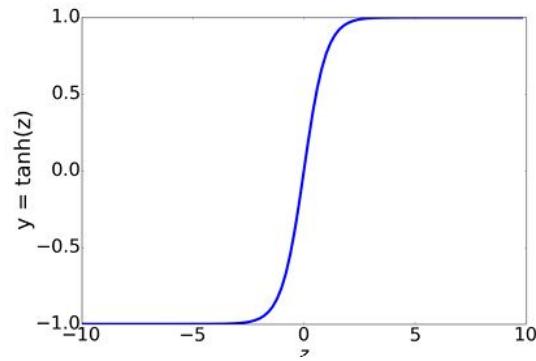
$$= \frac{1}{1 + e^{-(.5*.2+.6*.3+.1*.9+.5)}} = \frac{1}{1 + e^{-0.87}} = .70$$



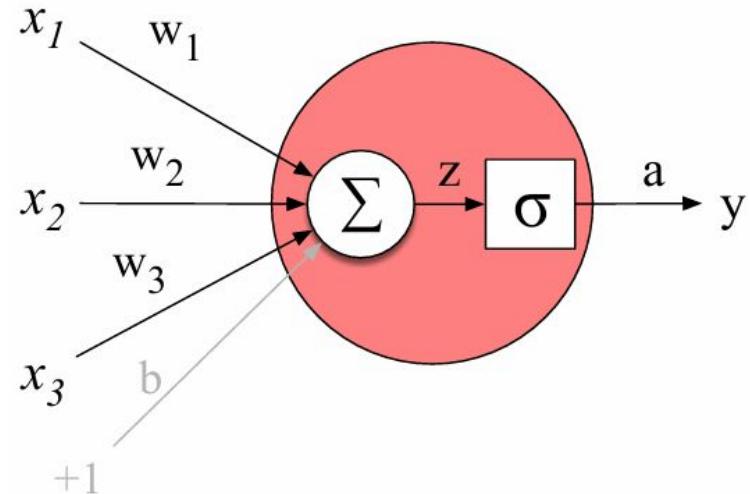
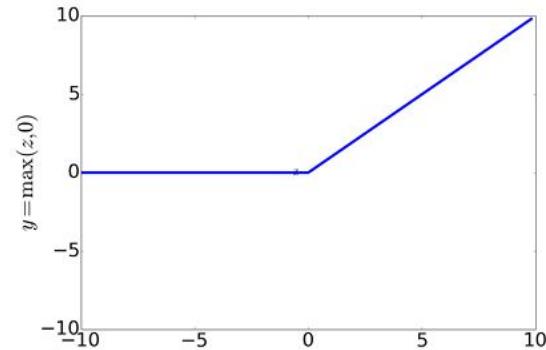
Sigmoid Activation

Other activation function

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

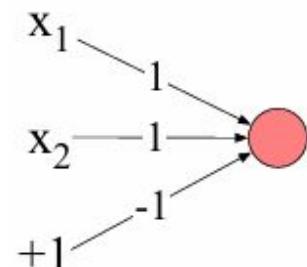


$$y = \text{ReLU}(z) = \max(z, 0)$$

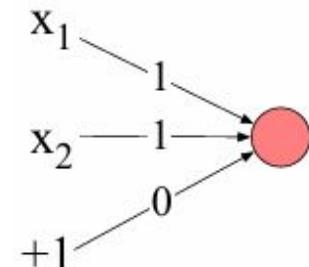


The XOR Problem

AND		OR		XOR				
x1	x2	y	x1	x2	y	x1	x2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0



Logical AND



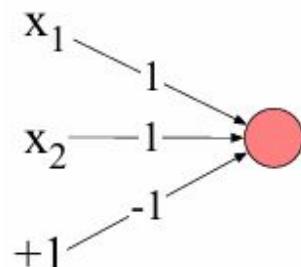
Logical OR

$$y = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

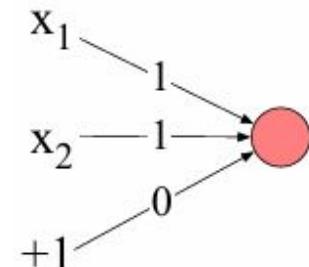
The XOR Problem

it's not possible to build a perceptron to compute logical XOR!

AND		OR		XOR	
x1	x2	y	x1	x2	y
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	0



Logical AND

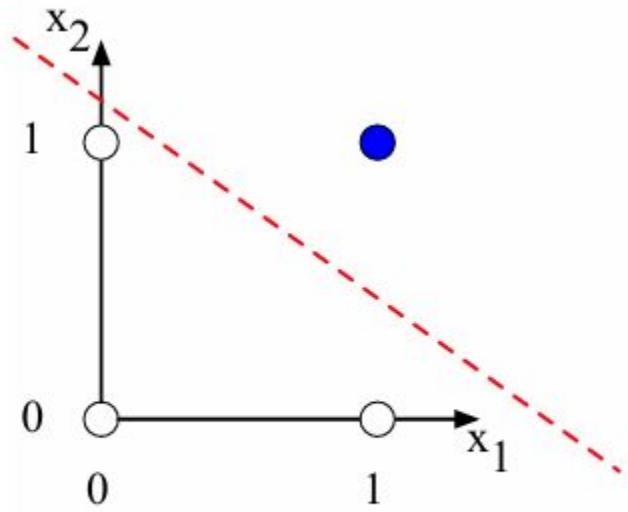


Logical OR

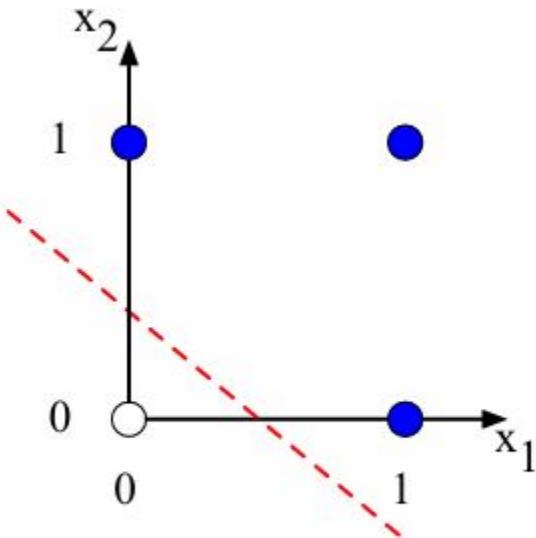
$$y = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

The intuition behind this important result relies on understanding that a perceptron is a linear classifier.

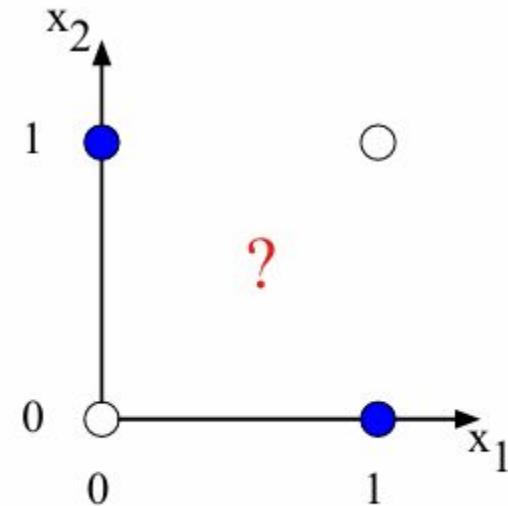
The XOR Problem



a) x_1 AND x_2



b) x_1 OR x_2



c) x_1 XOR x_2

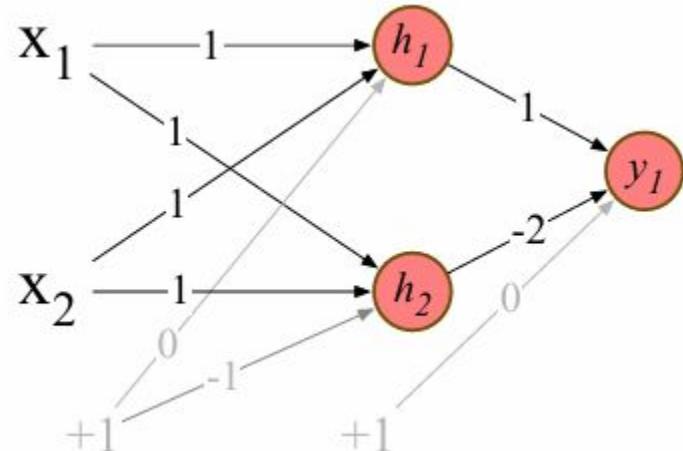
The Solution: neural networks

$x = [0, 0]$, $h = \text{ReLU}[0, -1] = [0, 0]$, $y = 0$

$x = [0, 1]$, $h = \text{ReLU}[1, 0] = [1, 0]$, $y = 1$

$x = [1, 0]$, $h = \text{ReLU}[1, 0] = [1, 0]$, $y = 1$

$x = [1, 1]$, $h = \text{ReLU}[2, 1] = [2, 1]$, $y = 0$



XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

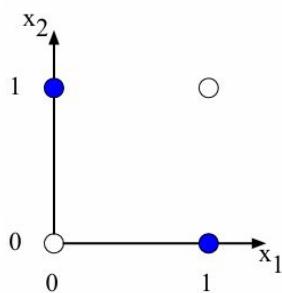
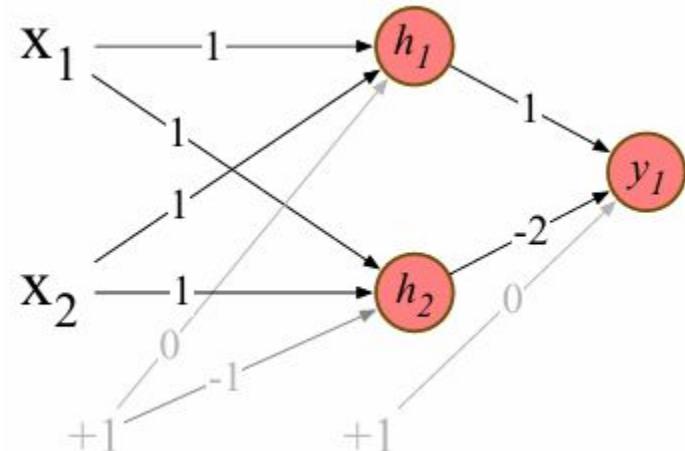
The Solution: neural networks

$x = [0, 0]$, $h = \text{ReLU}[0, -1] = [0, 0]$, $y = 0$

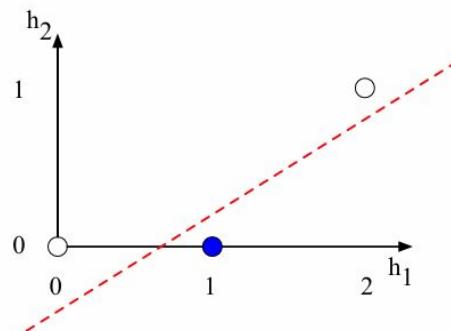
$x = [0, 1]$, $h = \text{ReLU}[1, 0] = [1, 0]$, $y = 1$

$x = [1, 0]$, $h = \text{ReLU}[1, 0] = [1, 0]$, $y = 1$

$x = [1, 1]$, $h = \text{ReLU}[2, 1] = [2, 1]$, $y = 0$



a) The original x space



b) The new (linearly separable) h space

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

Feedforward Neural Network

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{z} = \mathbf{U}\mathbf{h}$$

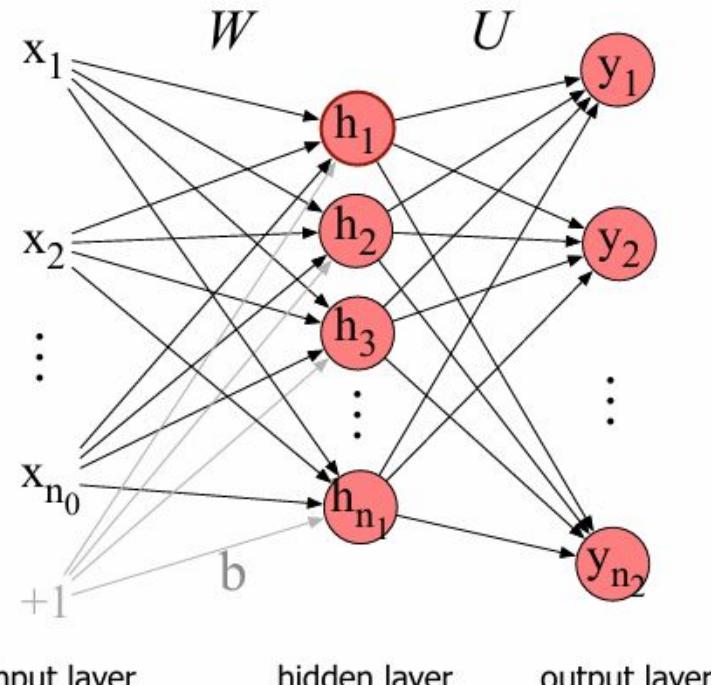
$$\mathbf{y} = \text{softmax}(\mathbf{z})$$

$$\text{softmax}(\mathbf{z}_i) = \frac{\exp(\mathbf{z}_i)}{\sum_{j=1}^d \exp(\mathbf{z}_j)} \quad 1 \leq i \leq d$$

Thus for example given a vector

$$\mathbf{z} = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1],$$

$$\text{softmax}(\mathbf{z}) = [0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$



2-layer Feed Forward Network

Replacing Bias unit

Instead of an equation like

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

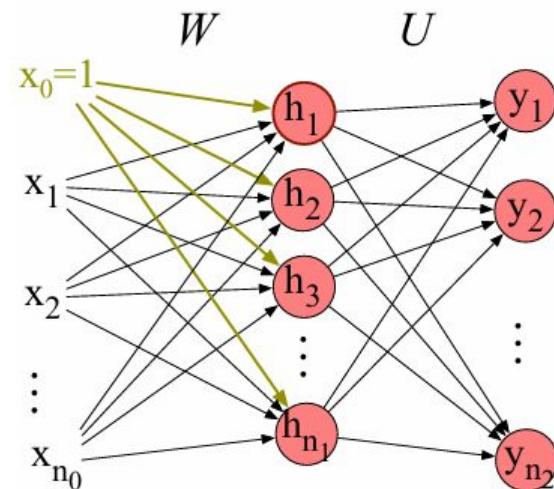
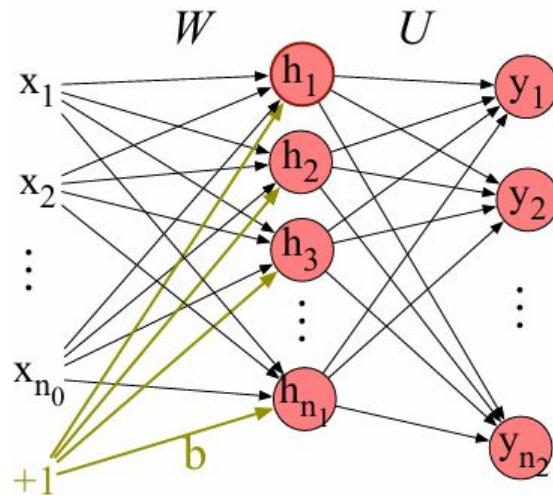
We use

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x})$$

Where $\mathbf{x} = \mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_{n_0}$

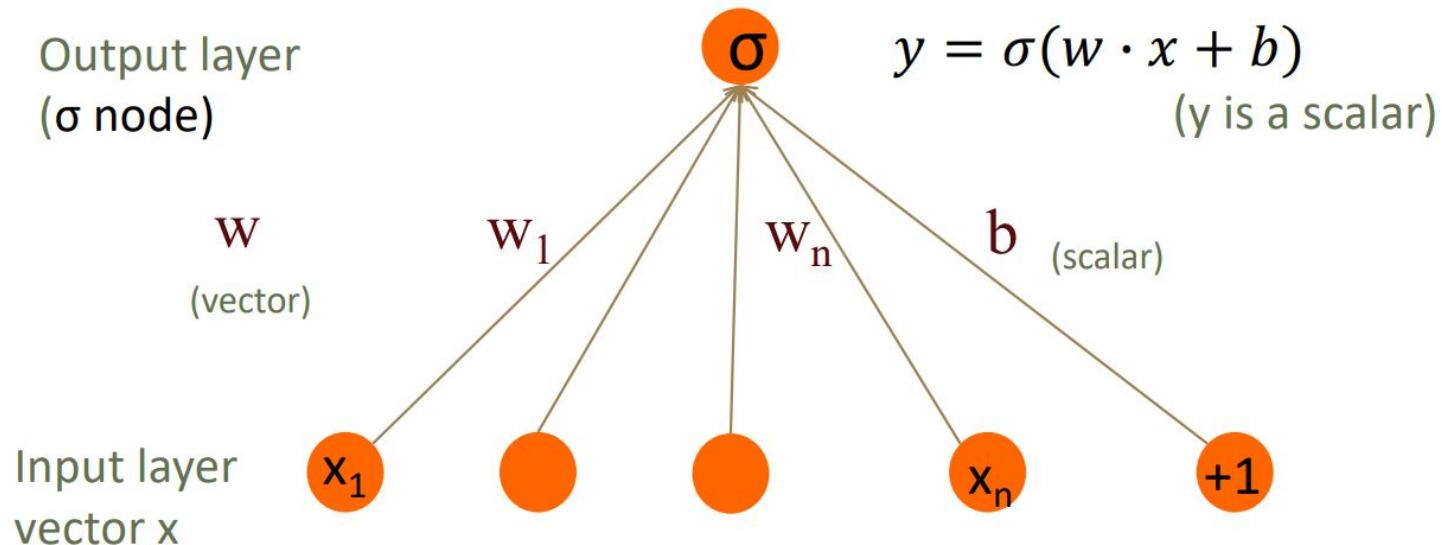
Updated equation is:

$$\mathbf{h}_j = \sigma \left(\sum_{i=0}^{n_0} \mathbf{w}_{ji} \mathbf{x}_i \right)$$



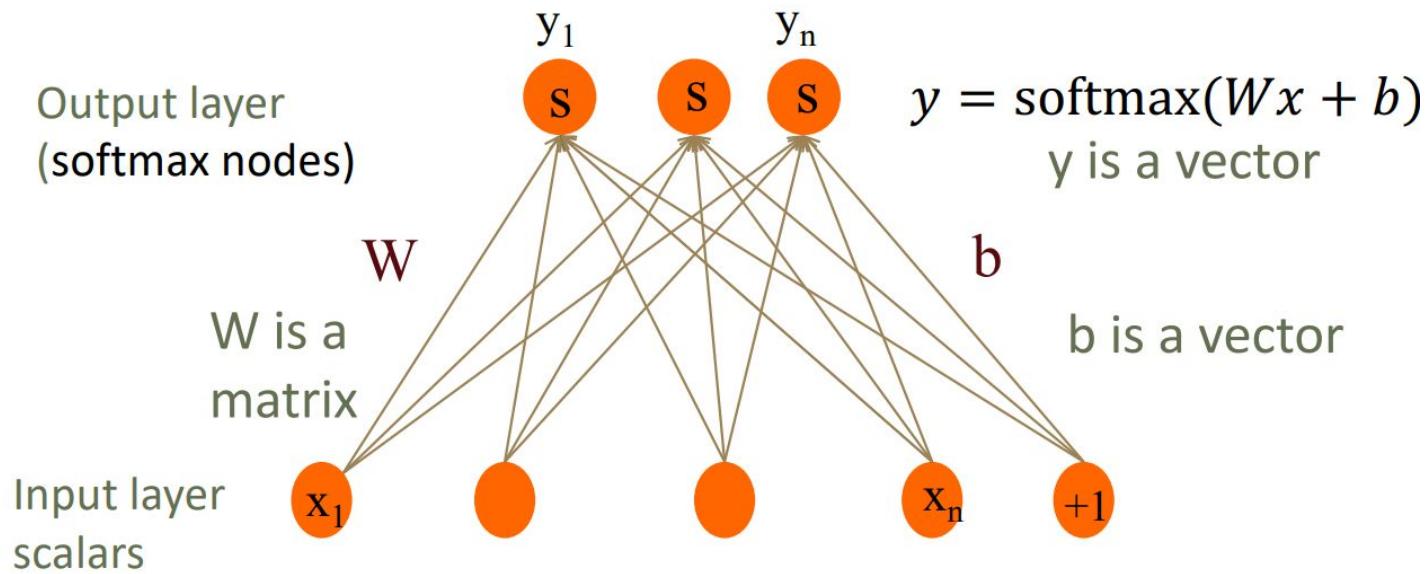
Binary Logistic Regression as a 1-layer Network

We don't count input layer in counting layers.



Multinomial Logistic Regression as a 1-layer Network

Fully connected single layer network



Softmax Activation Function

For a vector z of dimensionality k , the softmax is

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

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

Example:

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

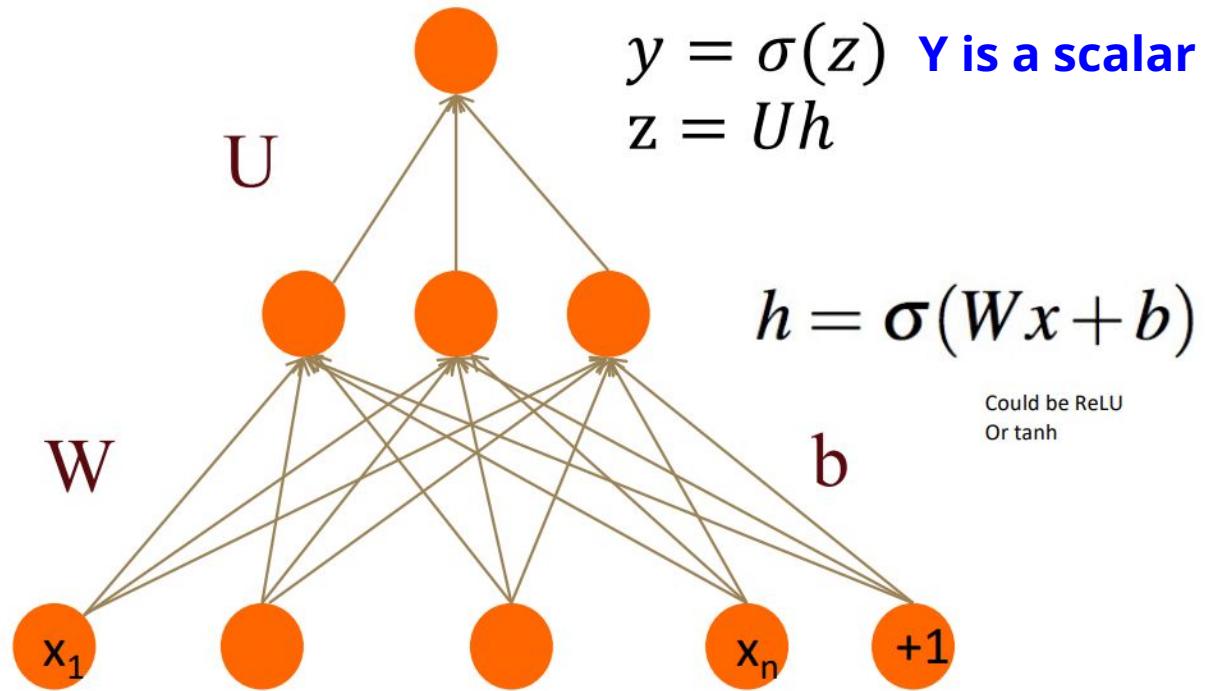
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Two-Layer Network with scalar output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)

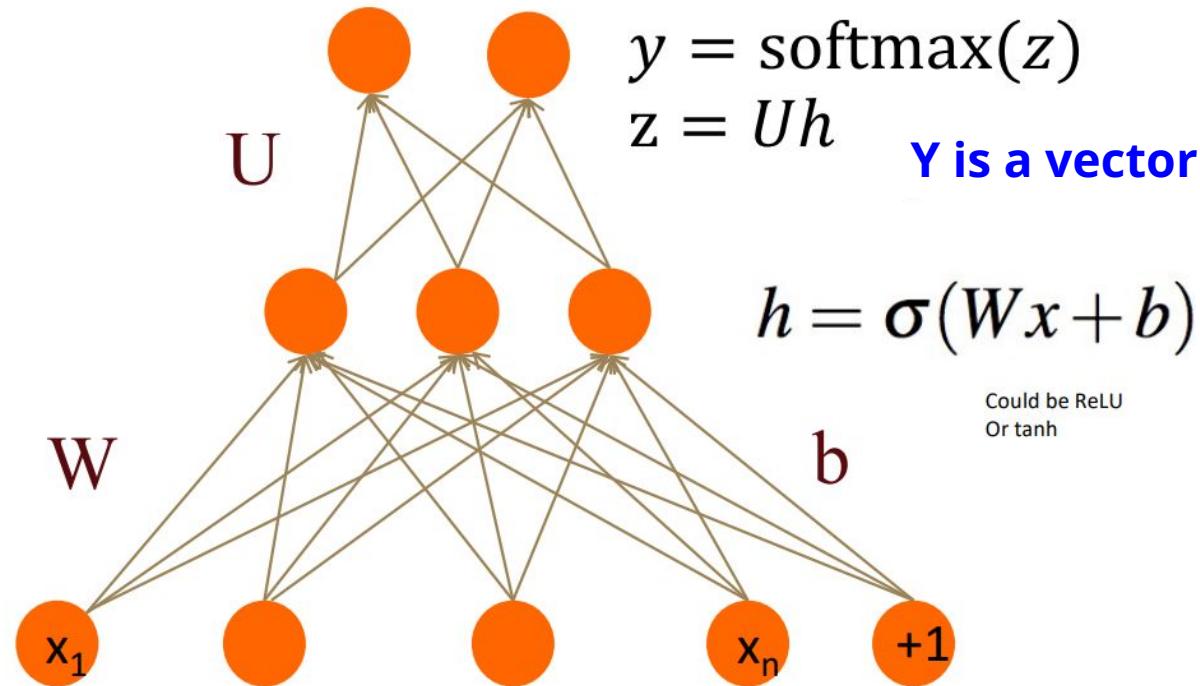


Two-Layer Network with softmax output

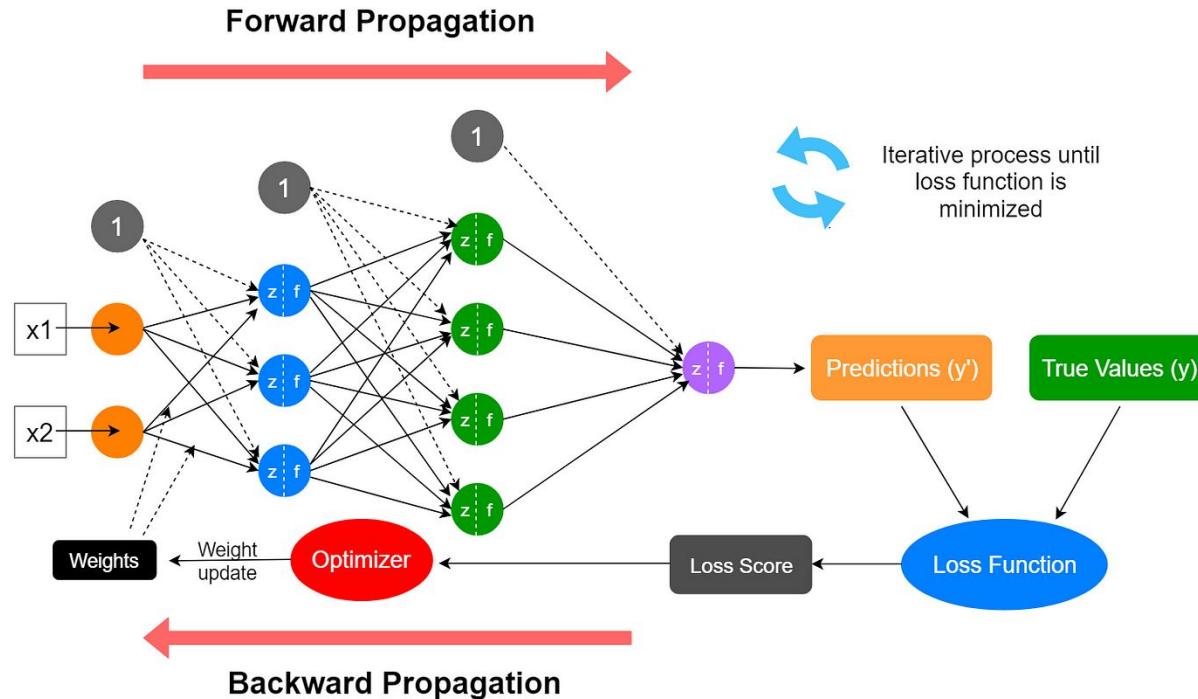
Output layer
(σ node)

hidden units
(σ node)

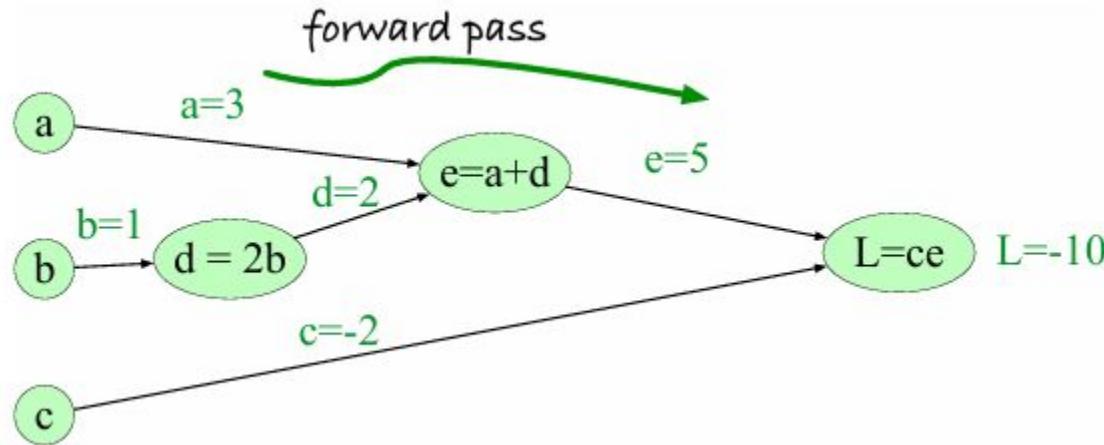
Input layer
(vector)



Training Neural Network



Training NN: Computational graph Forward pass

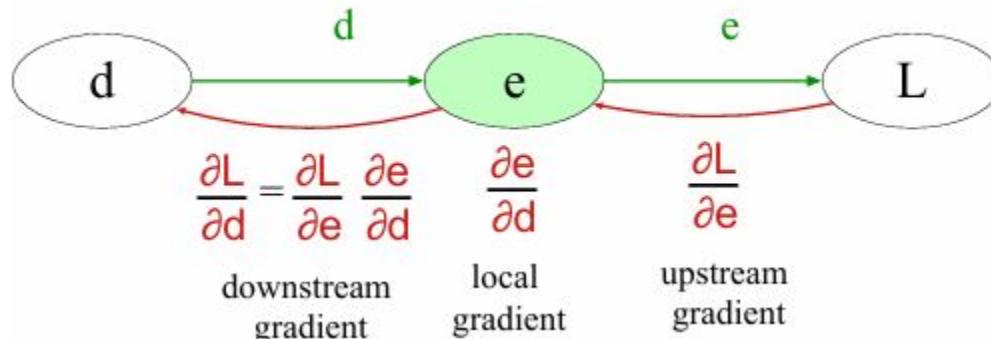


Computation graph for the function $L(a,b,c)=c(a+2b)$, with values for input nodes

$a = 3, b = 1, c = 2,$

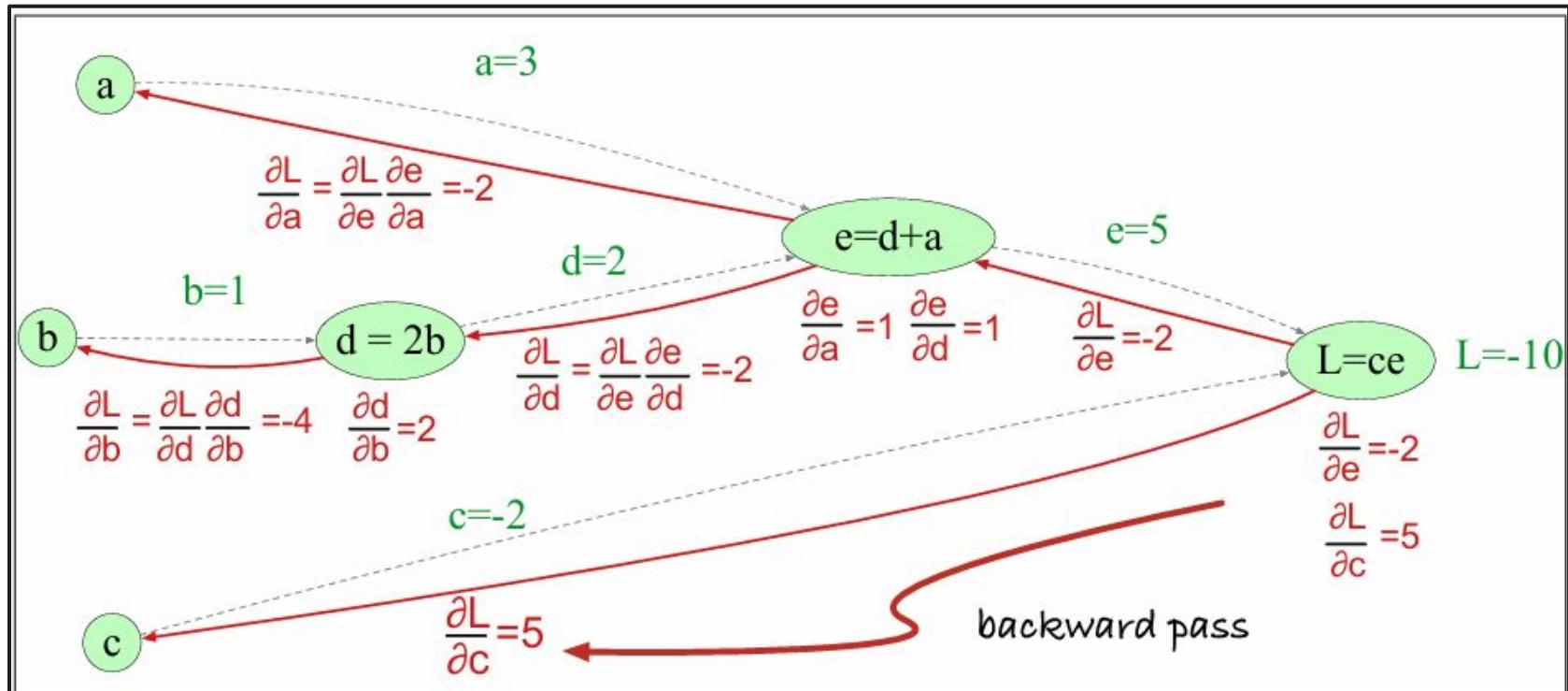
showing the forward pass computation of L .

Training NN: Computational graph Backward pass



- Each node (like **e** here) takes an upstream gradient, multiplies it by the local gradient (the gradient of its output with respect to its input), and uses the chain rule to compute a downstream gradient to be passed on to a prior node.
- A node may have multiple local gradients if it has multiple inputs.

Training NN: Computational graph Backward pass



Neural Language Modeling

- **Language modeling:** predicting upcoming words from prior words.
- Compared to n-gram models, neural language models can handle much longer histories, can generalize better over contexts of similar words, and are more accurate at word-prediction.
- On the other hand, neural net language models are much more complex, are slower and need more energy to train, and are less interpretable than n-gram models, so for some smaller tasks an n-gram language model is still the right tool.

Neural Language Modeling

- The feedforward neural LM approximates the probability of a word given the entire prior context:

$$P(w_t | w_1, \dots, w_{t-1}) \approx P(w_t | w_{t-N+1}, \dots, w_{t-1})$$

Train Example: I have to make sure that the cat gets fed.

Our test set has the prefix “I forgot to make sure that the dog gets”.
What’s the next word?

Neural Language Modeling

- An **n-gram** language model will predict “fed” after “that the cat gets”, but not after “that the dog gets”.
- But a neural LM, knowing that “cat” and “dog” have similar **embeddings**, will be able to generalize from the “cat” context to assign a high enough probability to “fed” even after seeing “dog”.

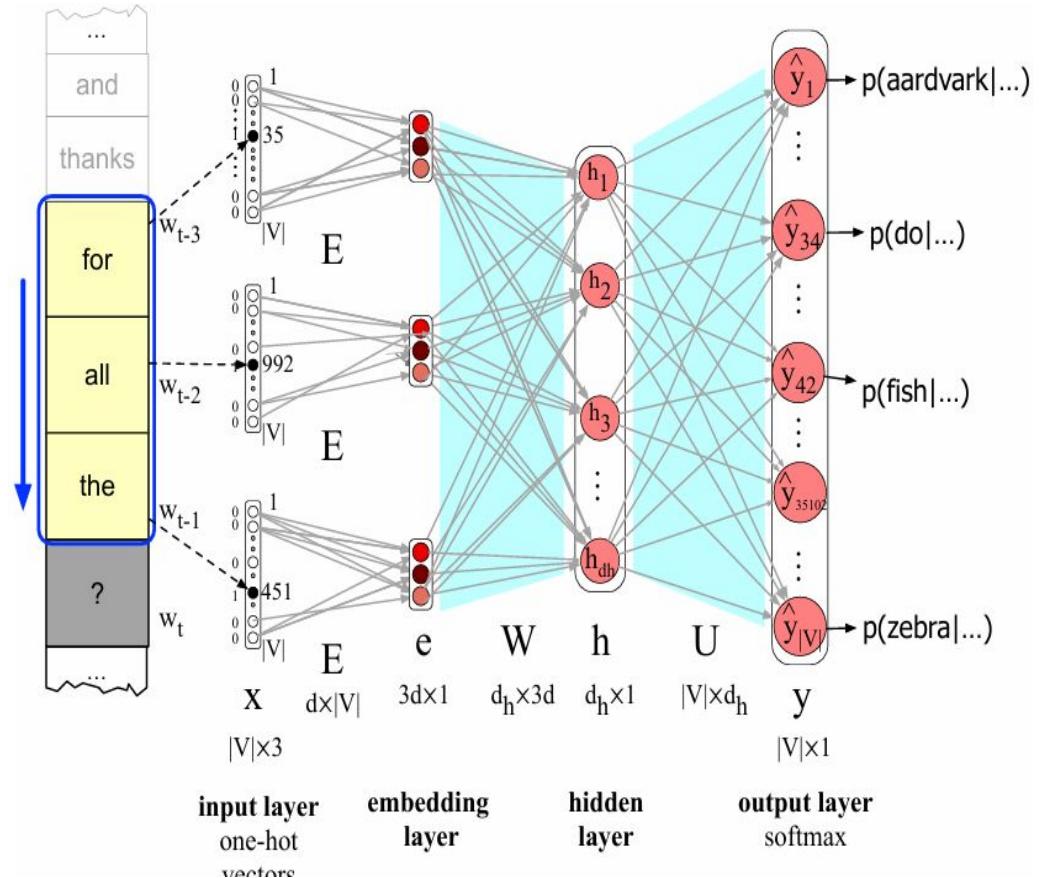
Neural Language Modeling

$$\mathbf{e} = [\mathbf{E}\mathbf{x}_{t-3}; \mathbf{E}\mathbf{x}_{t-2}; \mathbf{E}\mathbf{x}_{t-1}]$$

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

$$\mathbf{z} = \mathbf{U}\mathbf{h}$$

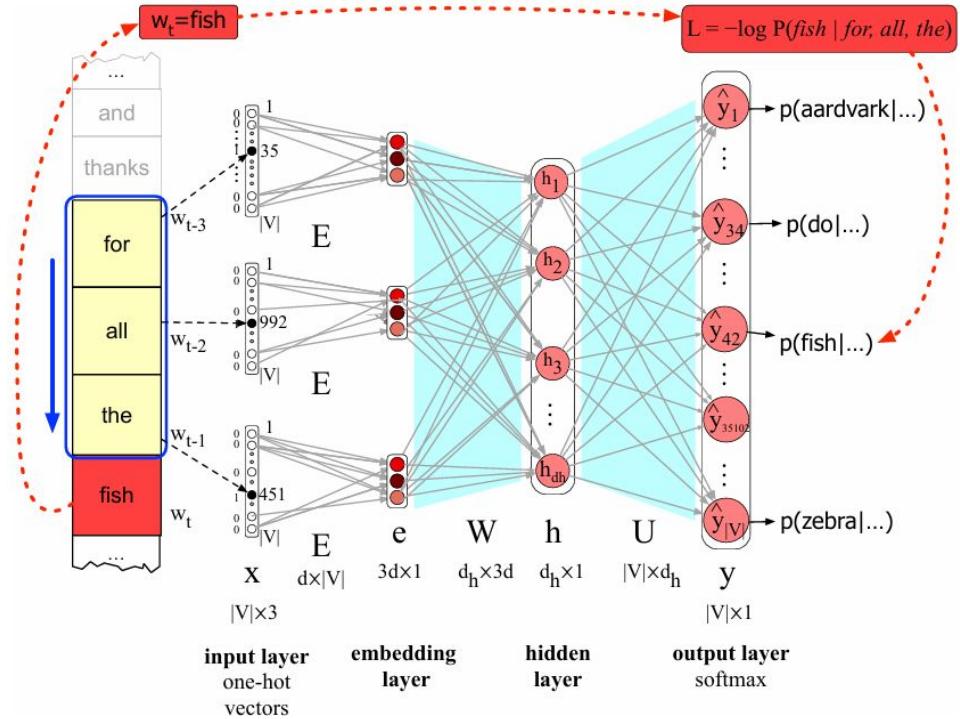
$$\hat{\mathbf{y}} = \text{softmax}(\mathbf{z})$$



Training Neural Language Modeling

It uses the self-supervision for training.

Demo



Recurrent Neural Network

Sequential Data

cgpa	iq	place?	Pred
7.1	71	1-yes	0.73
8.5	85	0-no	0.63
9.5	95	0-no	

Non-sequential Data

Sequential Data Example

Text Data:

We are in NLM class

Speech Data:

Time Series Data:

DNA Sequence:

Stock Price prediction Data:

Date	Open Price	High Price	Low Price	Close Price	Volume
2024-02-20	100.5	102.3	98.75	101.25	125,000
2024-02-21	101.25	103.75	99.5	103	150,000
2024-02-22	103	104.5	101	102.5	130,000

Sequential Data

**RNN is designed
for sequential data**

Sequential Data Example

Text Data:

We are in LLM class

Speech Data:

Time Series Data:

DNA Sequence:

Stock Price prediction Data:

Date	Open Price	High Price	Low Price	Close Price	Volume
2024-02-20	100.5	102.3	98.75	101.25	125,000
2024-02-21	101.25	103.75	99.5	103	150,000
2024-02-22	103	104.5	101	102.5	130,000

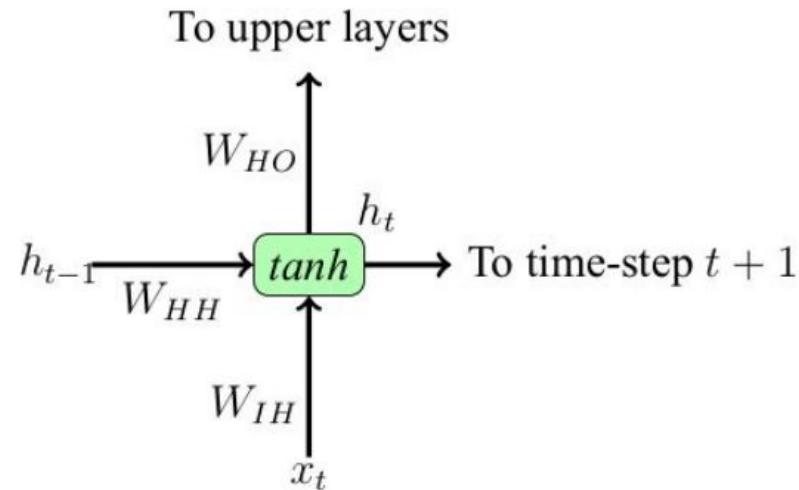
Why RNN?

- Text inputs are of variable sizes
- Zero padding makes the unnecessary computation
- Totally disregarding the sequential information

RNN-The what?

- A neural network with feedback connections
- Enable networks to do temporal processing
- Good at learning sequences
- Acts as memory unit

$$\begin{aligned} h_t &= \tanh(W_{IH} \cdot x_t + W_{HH} \cdot h_{t-1} + b) \\ &= \tanh([W_{IH}, W_{HH}] \cdot [x_t, h_{t-1}] + b) \end{aligned}$$

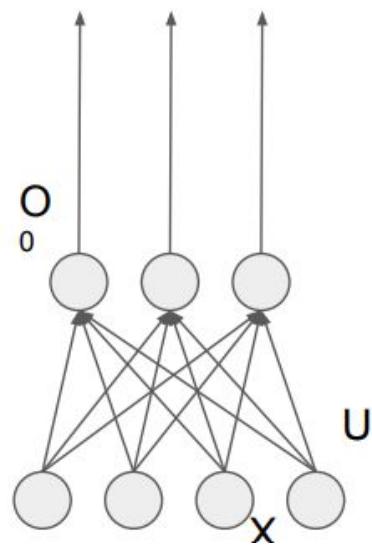


RNN-visualization

- Problem:
 - I/p sequence (X) : X^0, X^1, \dots, X^T
 - O/p sequence (O) : O^0, O^1, \dots, O^T
- Representation of data:
 - I/p dimension : 4
 - $X^0 \rightarrow 0\ 1\ 1\ 0$
 - O/p dimension : 3
 - $O^0 \rightarrow 0\ 0\ 1$
- Network Architecture
 - Number of neurons at I/p layer : 4
 - Number of neurons at O/p layer : 3
 - Do we need hidden layers?
 - If yes, number of neurons at each hidden layers

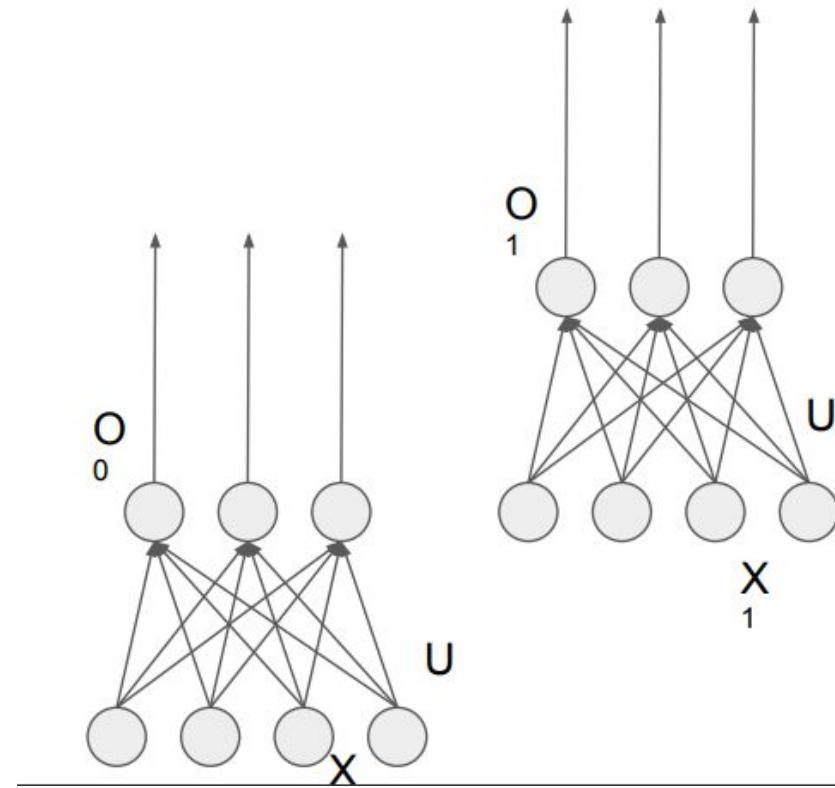
RNN-visualization

Network at Timestamp: 0



RNN-visualization

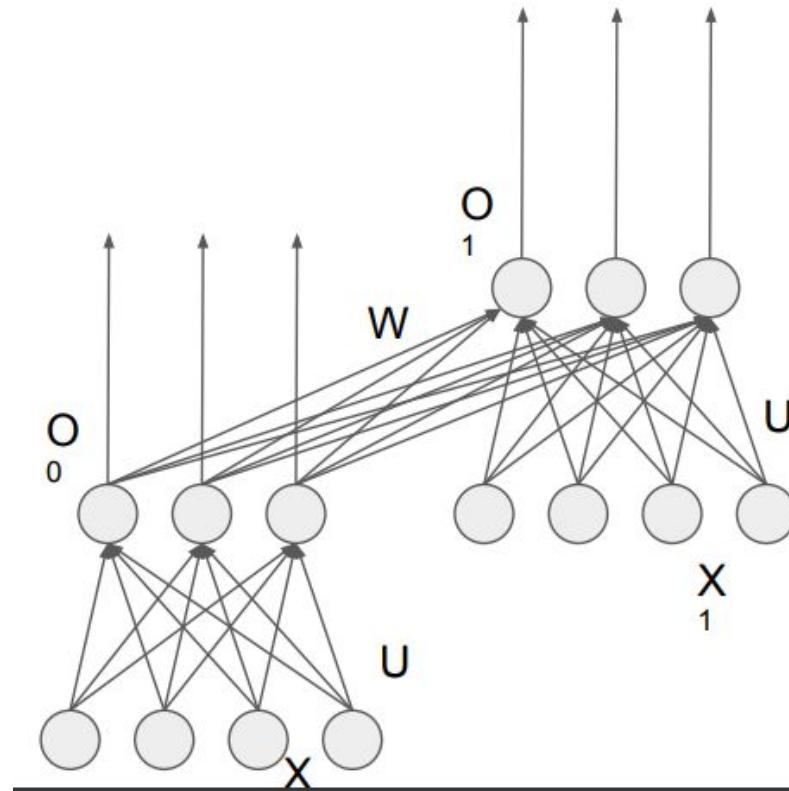
Network at Timestamp: 1



RNN-visualization

Network at Timestamp: 1

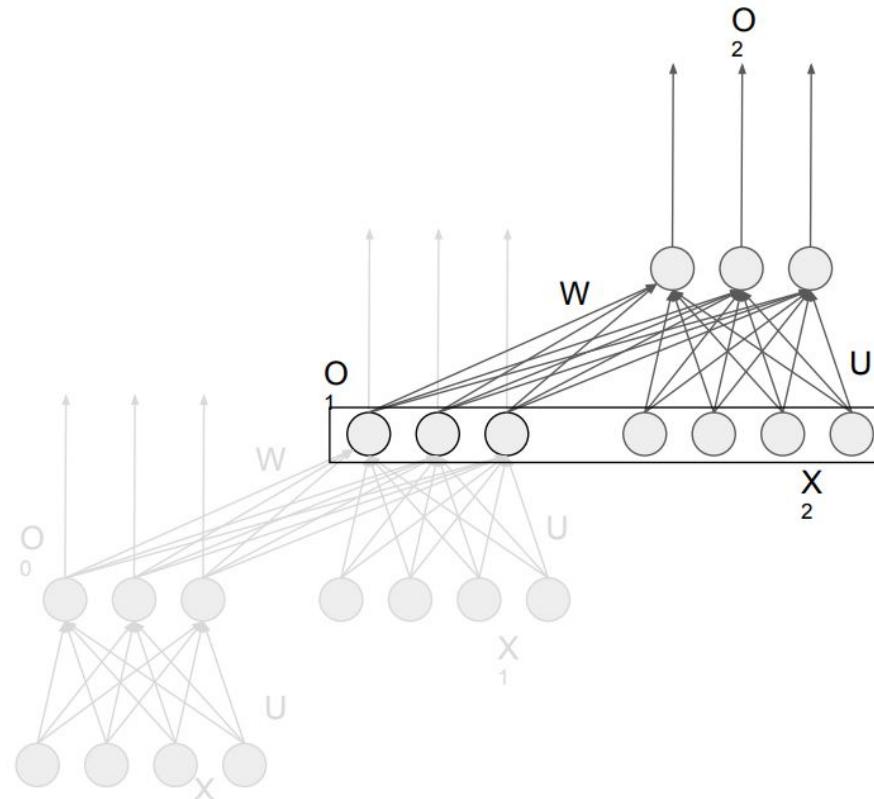
$$\begin{aligned} O^1 &= f(W \cdot O^0 + U \cdot X^1) \\ &= f([W, U] \cdot [O^0, x^1]) \end{aligned}$$



RNN-visualization

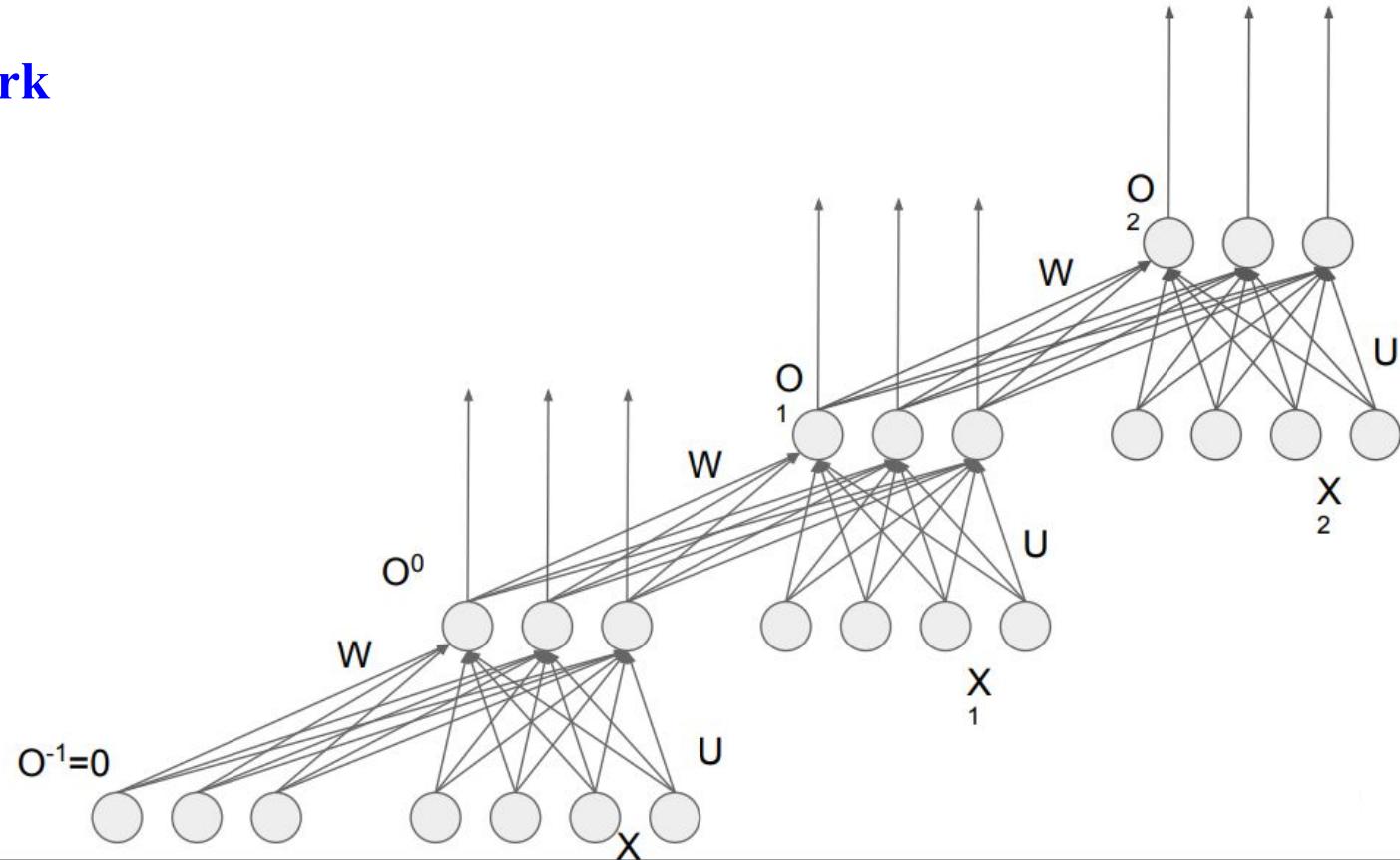
Network at Timestamp: 1

$$\begin{aligned} O^2 &= f(W \cdot O^1 + U \cdot X^2) \\ &= f([W, U] \cdot [O^1, x^2]) \end{aligned}$$

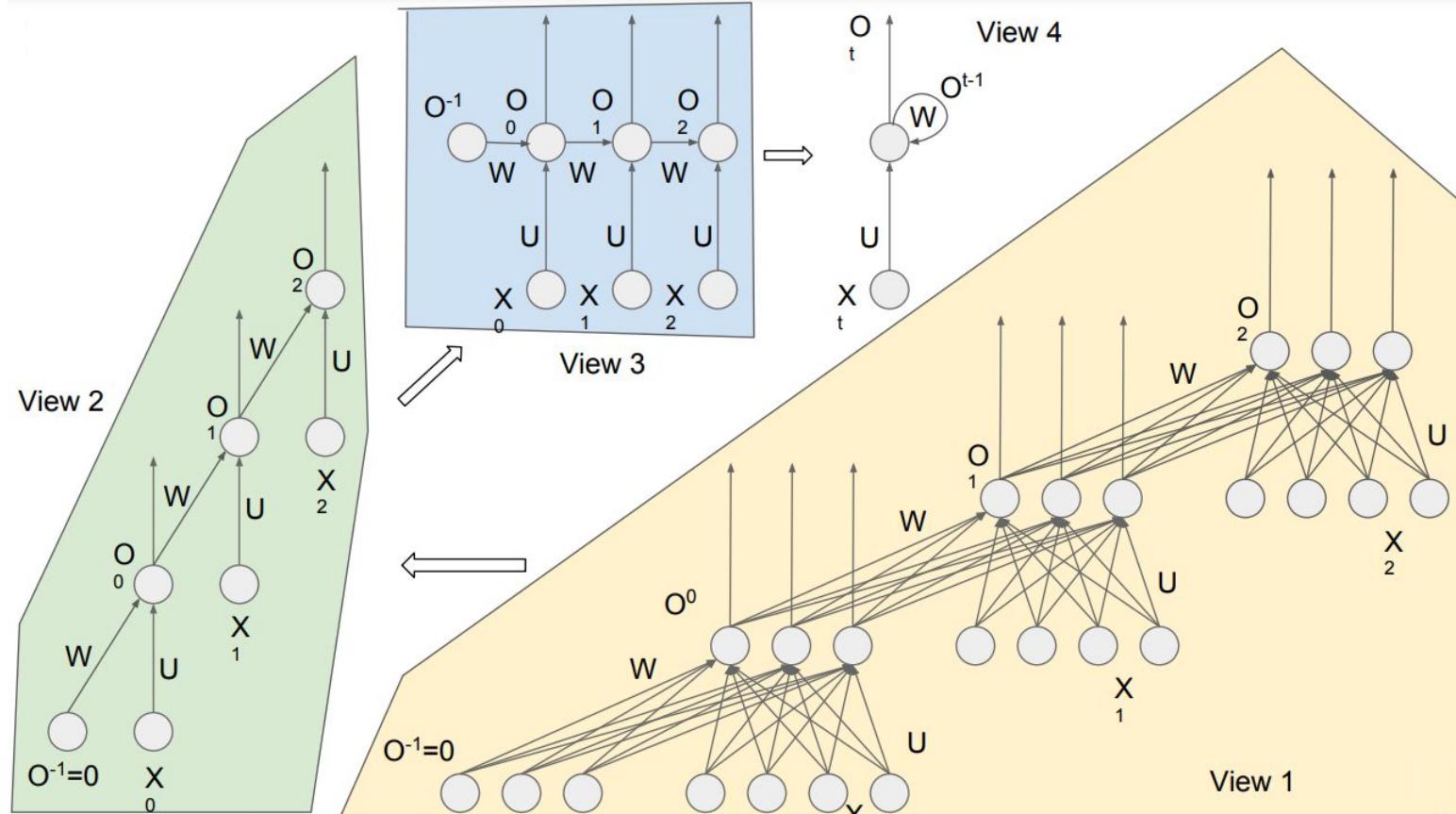


RNN-visualization

Complete Network

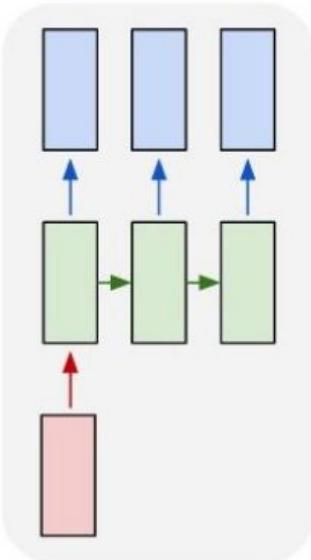


RNN-visualization

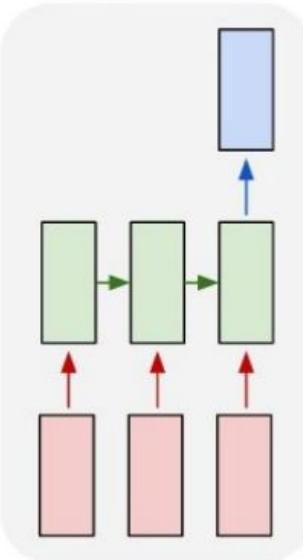


RNN-Types

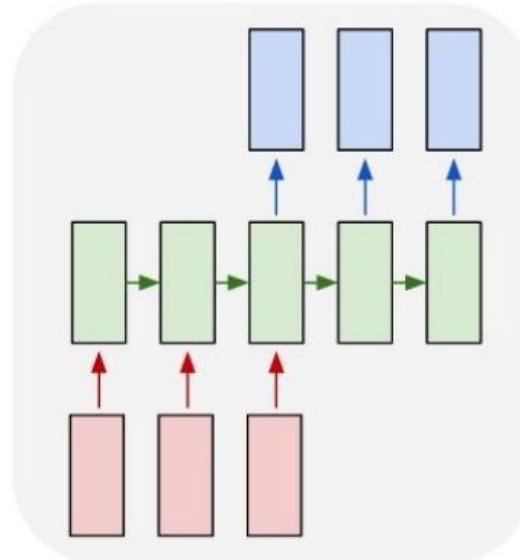
one to many



many to one



many to many



many to many

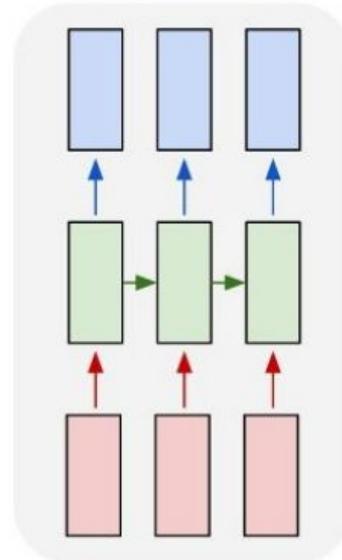


Image
Captioning

Sentiment
Analysis

Machine
Translation

Language
modelling

RNN-Backpropagation

$$S_t = g_1(W_x x_t + W_s S_{t-1})$$

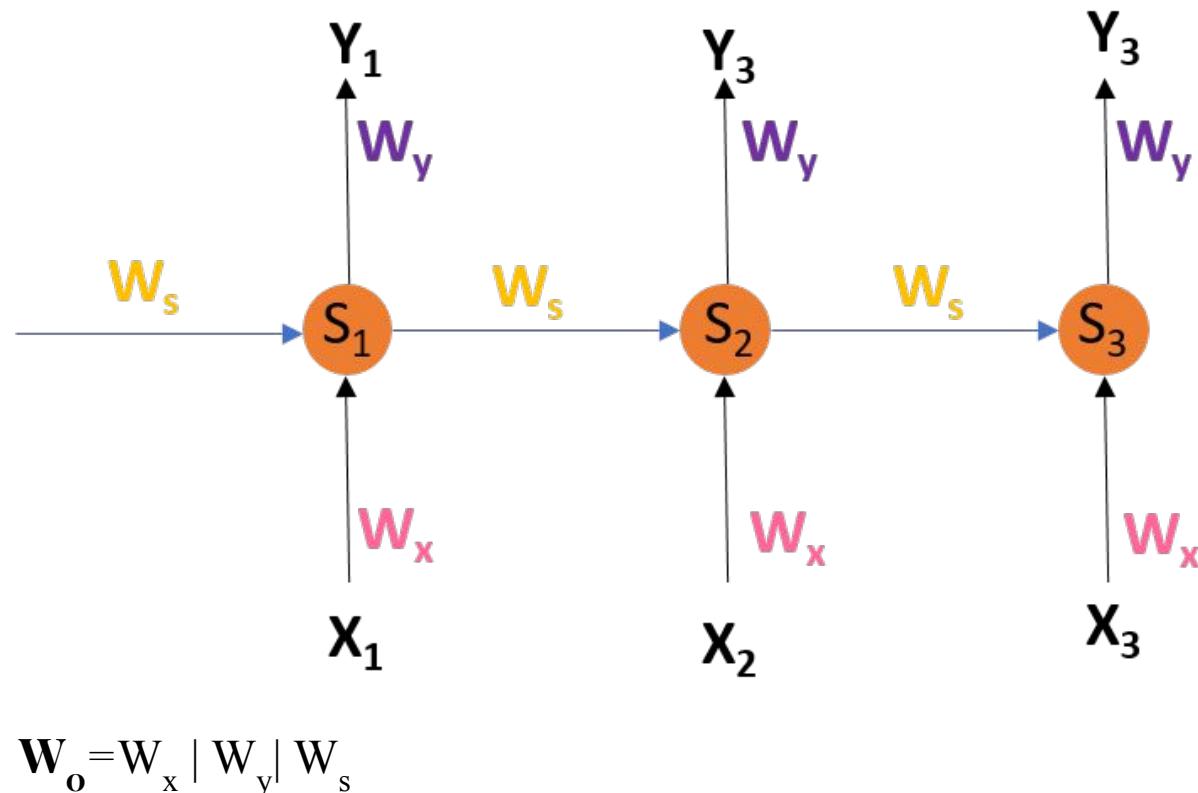
$$Y_t = g_2(W_Y S_t)$$

$$E_t = (d_t - Y_t)^2$$

Gradient Descent:

W_x , W_y , W_s would be adjusted

$$w_0 = w_0 - \alpha \frac{\partial E}{\partial w_0}$$



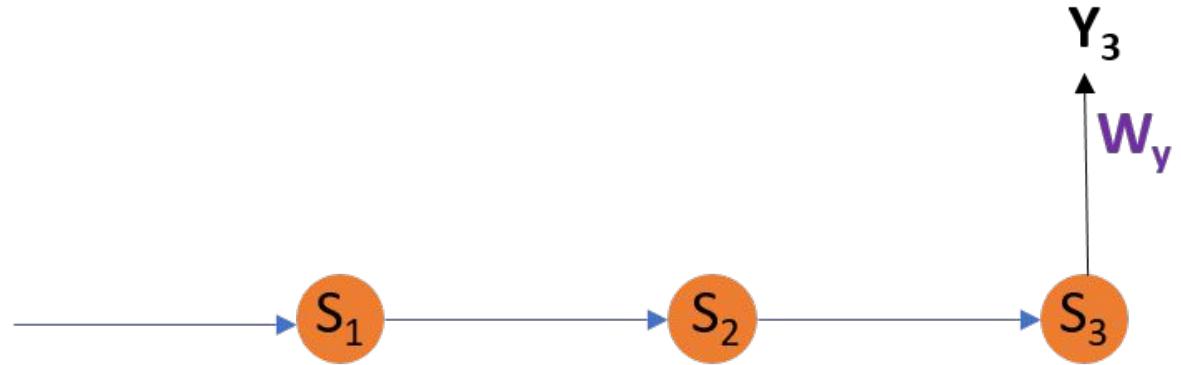
RNN-Backpropagation

$$S_t = g_1(W_x x_t + W_s S_{t-1})$$

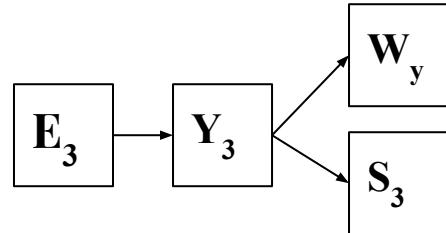
$$Y_t = g_2(W_Y S_t)$$

$$E_t = (d_t - Y_t)^2$$

Backpropagation at t=3



$$E_3 = (d_3 - Y_3)^2$$



$$\frac{\partial E_3}{\partial W_y} = \frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial W_Y}$$

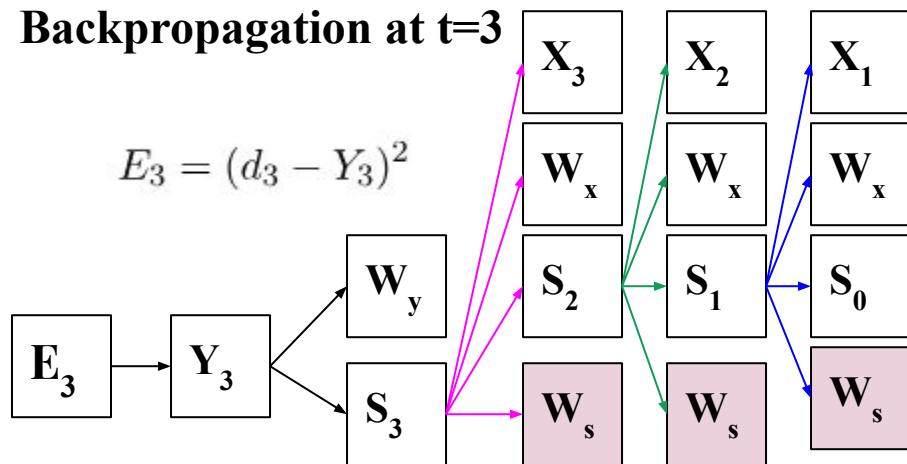
RNN-Backpropagation

$$S_t = g_1(W_x x_t + W_s S_{t-1})$$

$$Y_t = g_2(W_Y S_t)$$

$$E_t = (d_t - Y_t)^2$$

Backpropagation at t=3



$$E_3 = (d_3 - Y_3)^2$$



$$\frac{\partial E_3}{\partial W_S} = \left(\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial W_S} \right) +$$

$$\left(\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial W_S} \right) +$$

$$\left(\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial S_1} \cdot \frac{\partial S_1}{\partial W_S} \right)$$

RNN-Backpropagation

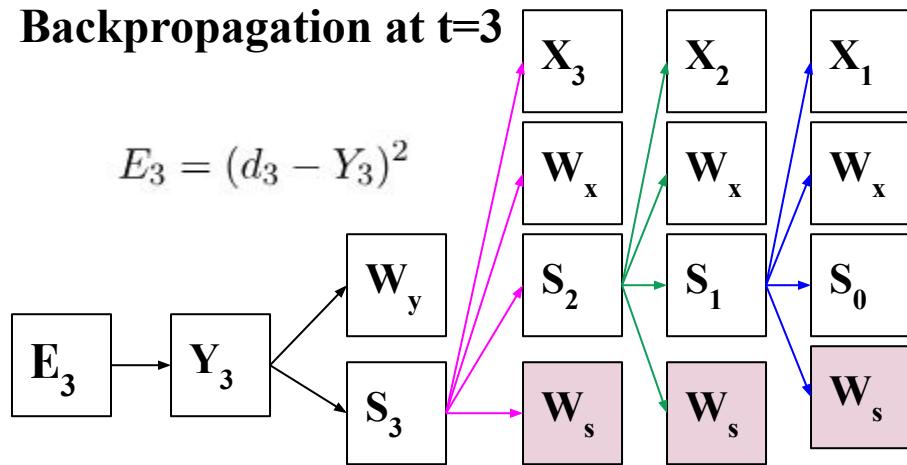
$$S_t = g_1(W_x x_t + W_s S_{t-1})$$

$$Y_t = g_2(W_Y S_t)$$

$$E_t = (d_t - Y_t)^2$$

Backpropagation at t=3

$$E_3 = (d_3 - Y_3)^2$$



$$\frac{\partial E_N}{\partial W_S} = \sum_{i=1}^N \frac{\partial E_N}{\partial Y_N} \cdot \frac{\partial Y_N}{\partial S_i} \cdot \frac{\partial S_i}{\partial W_S}$$

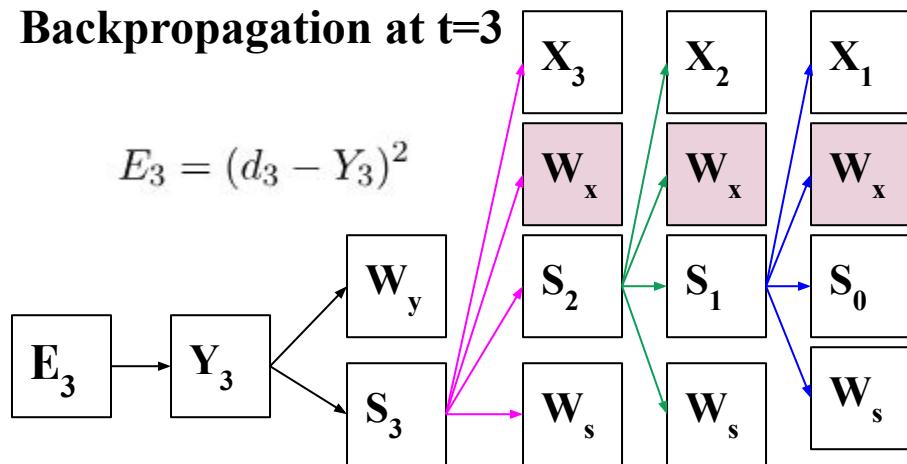
RNN-Backpropagation

$$S_t = g_1(W_x x_t + W_s S_{t-1})$$

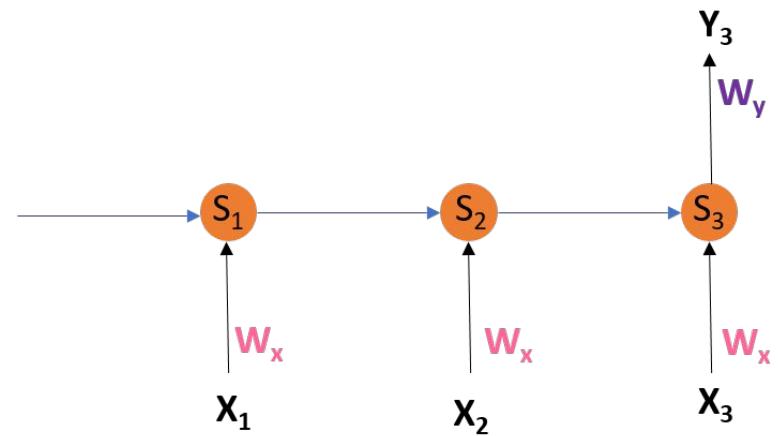
$$Y_t = g_2(W_Y S_t)$$

$$E_t = (d_t - Y_t)^2$$

Backpropagation at t=3



$$E_3 = (d_3 - Y_3)^2$$



$$\frac{\partial E_3}{\partial W_X} = \left(\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial W_X} \right) +$$

$$\left(\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial W_X} \right) +$$

$$\left(\frac{\partial E_3}{\partial Y_3} \cdot \frac{\partial Y_3}{\partial S_3} \cdot \frac{\partial S_3}{\partial S_2} \cdot \frac{\partial S_2}{\partial S_1} \cdot \frac{\partial S_1}{\partial W_X} \right)$$

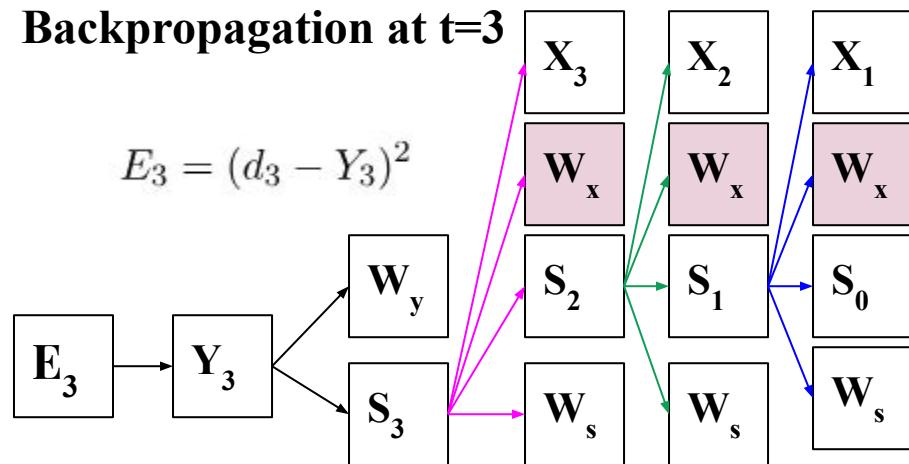
RNN-Backpropagation

$$S_t = g_1(W_x x_t + W_s S_{t-1})$$

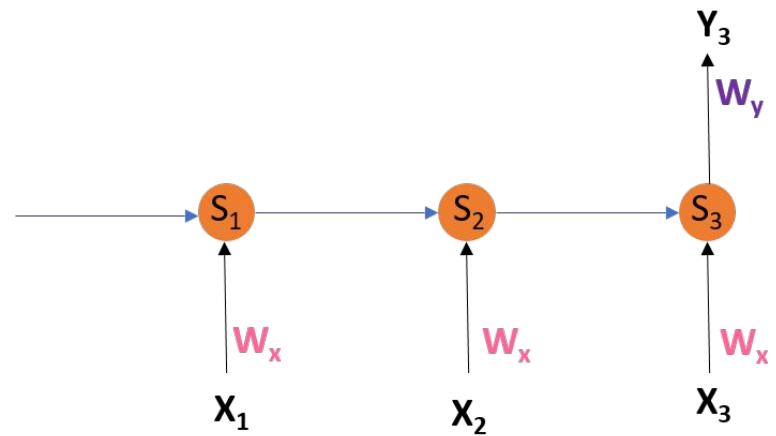
$$Y_t = g_2(W_Y S_t)$$

$$E_t = (d_t - Y_t)^2$$

Backpropagation at t=3



$$E_3 = (d_3 - Y_3)^2$$



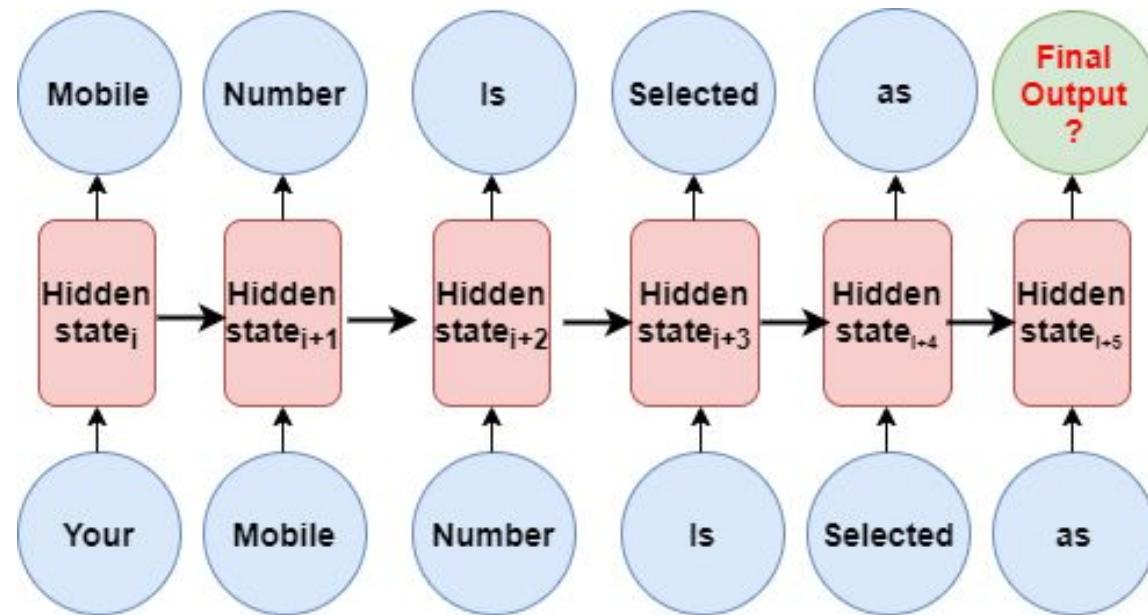
$$\frac{\partial E_N}{\partial W_S} = \sum_{i=1}^N \frac{\partial E_N}{\partial Y_N} \cdot \frac{\partial Y_N}{\partial S_i} \cdot \frac{\partial S_i}{\partial W_X}$$

RNN-Problem

- Vanishing Gradient
- Exploding gradient

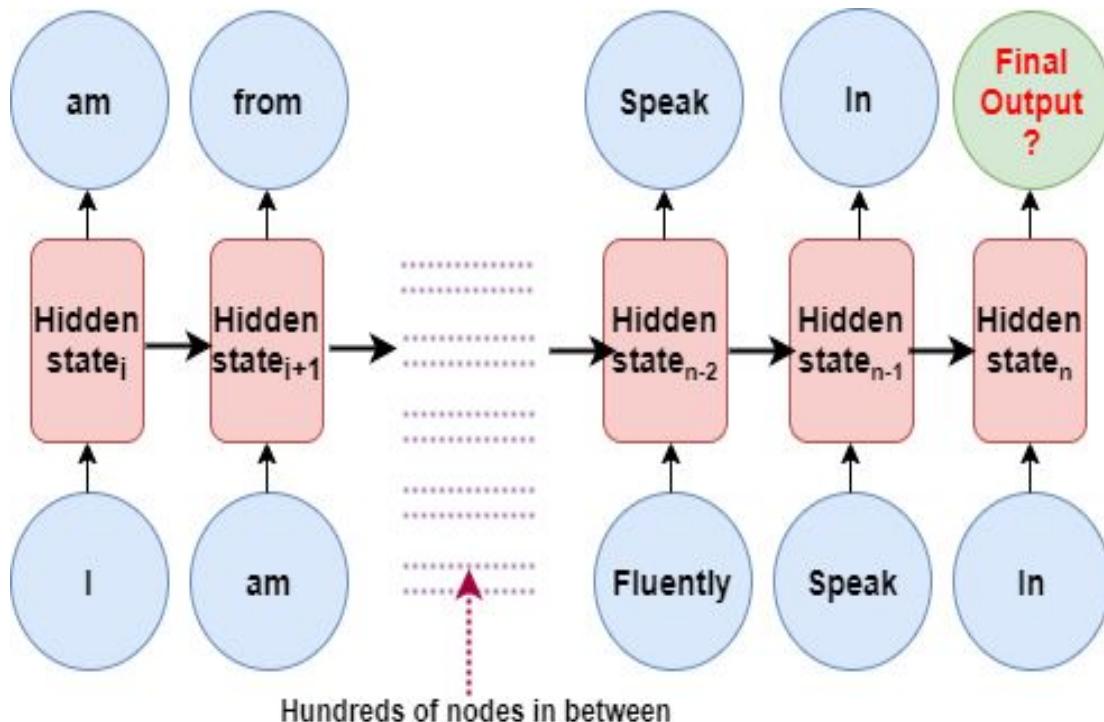
Let's understand this with example: **(Next word prediction)**

Scenario 1: “Your mobile number is selected as _____” : **RNN works fine**



RNN-Problem

Scenario 2: "I am from France. Once I visited India. I visited many places like Delhi: where people mostly speaks Hindi, Telangana: where the native language is Telegu, Tamilnadu: where the native language was Tamil. I also visited Taj Mahal, India Gate, and so on.

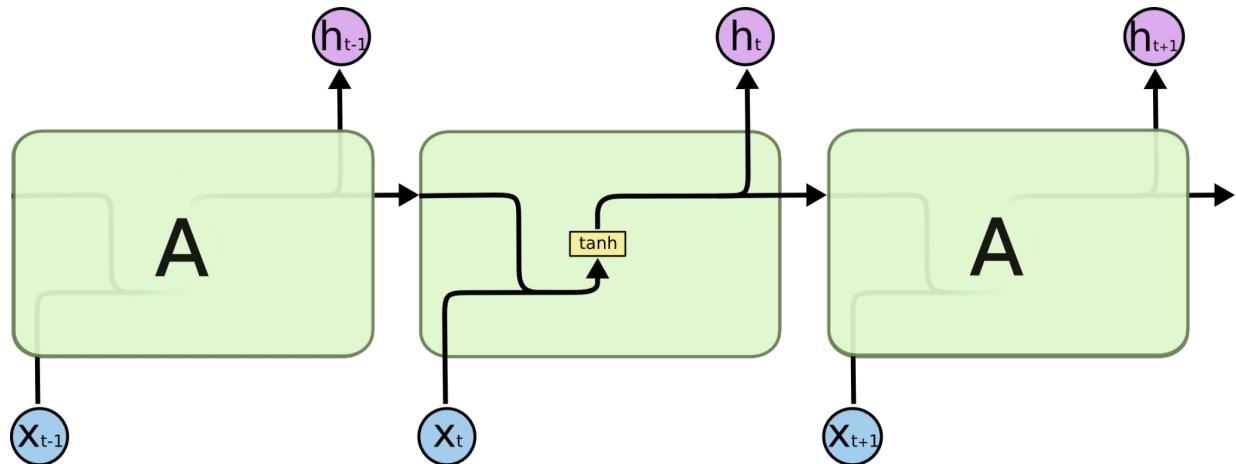


There are many tourist places in India where people from all over the world visit through the year. At last, I must say India is very diverse by culture and by language. Still, we never felt any difficulty, I got a right guide with whom I used to fluently speak in

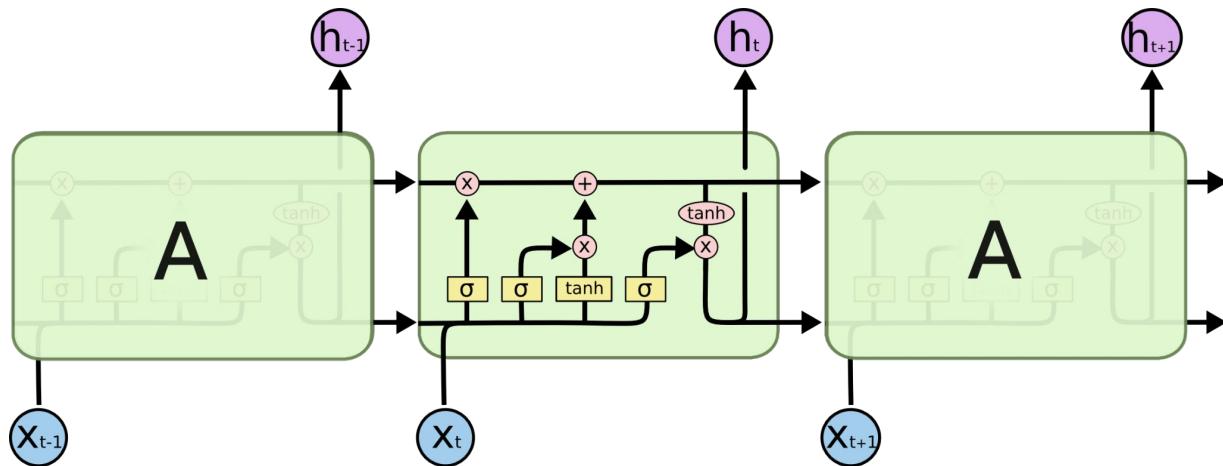
": **RNN fails when context is too long.**

RNN-Solution

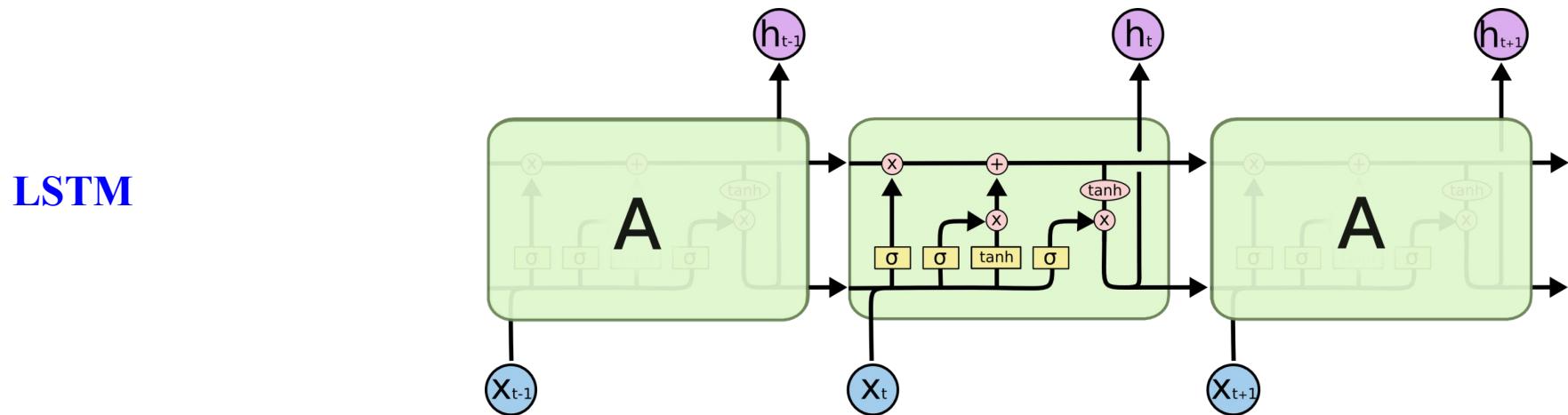
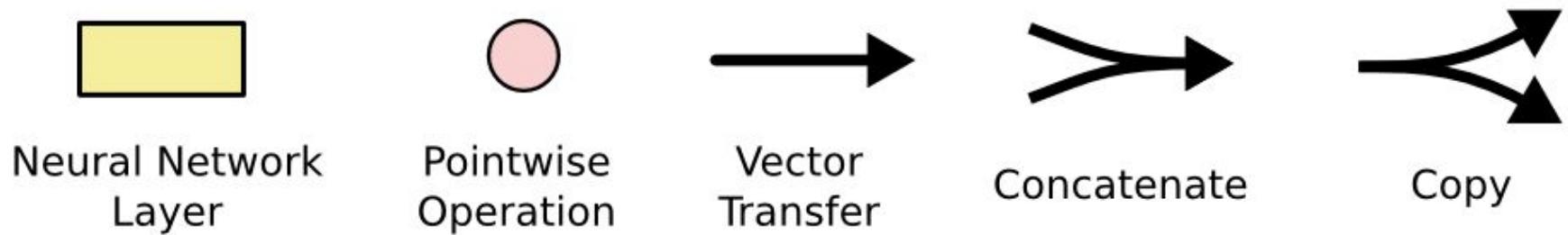
Vanilla RNN



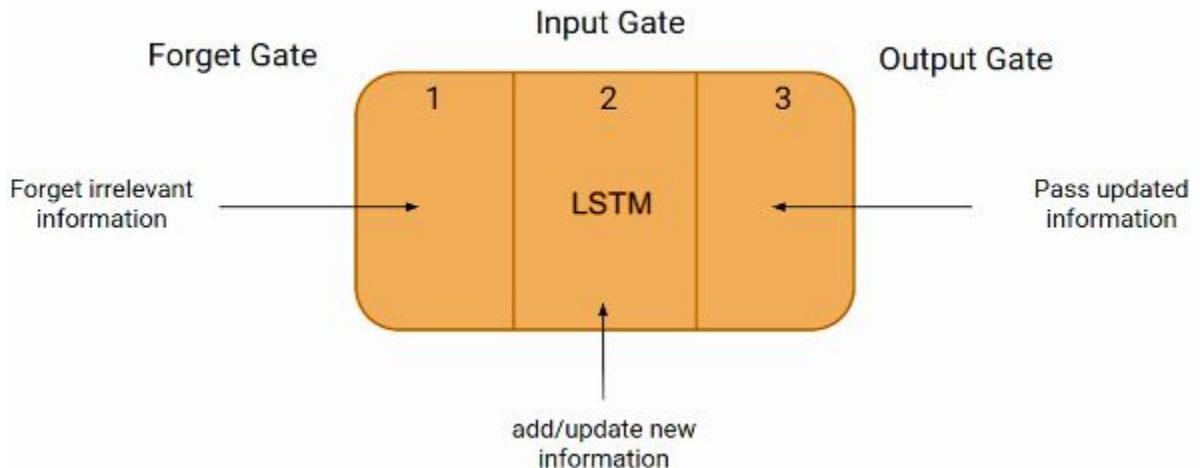
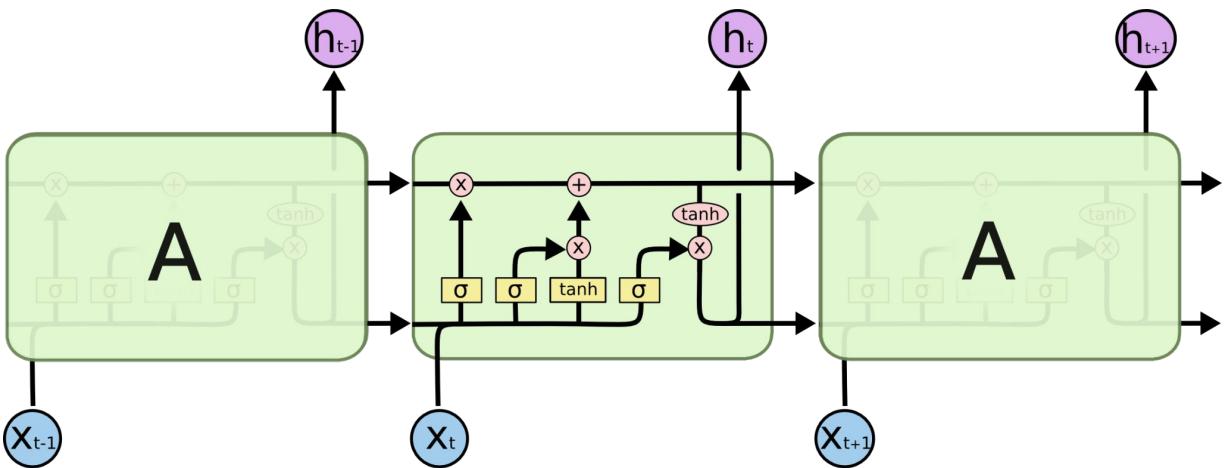
LSTM



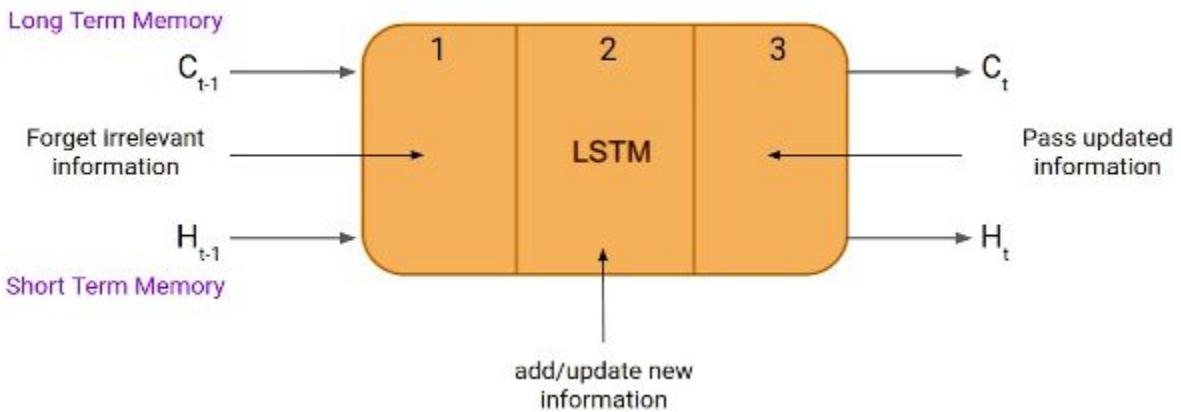
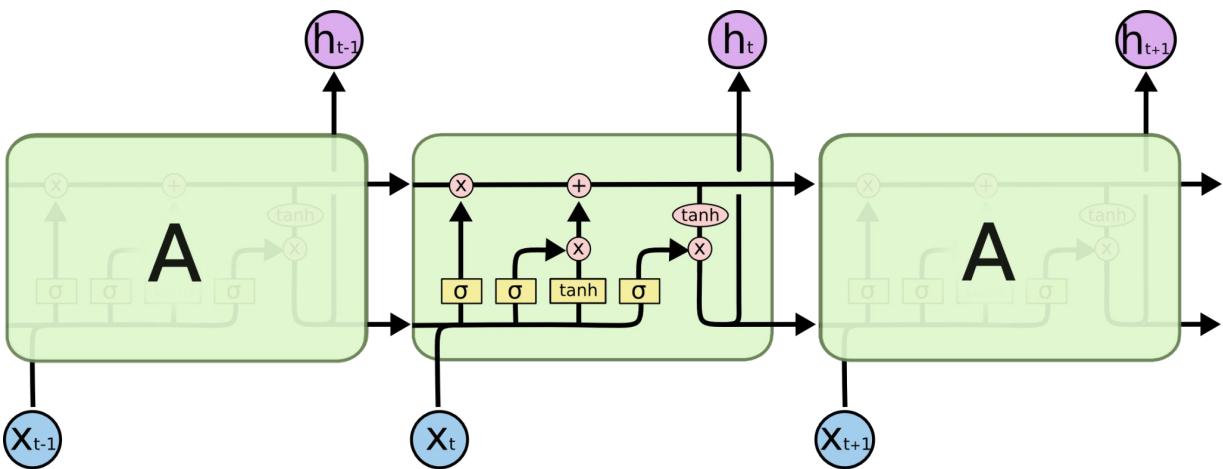
RNN-Solution



LSTM-core idea

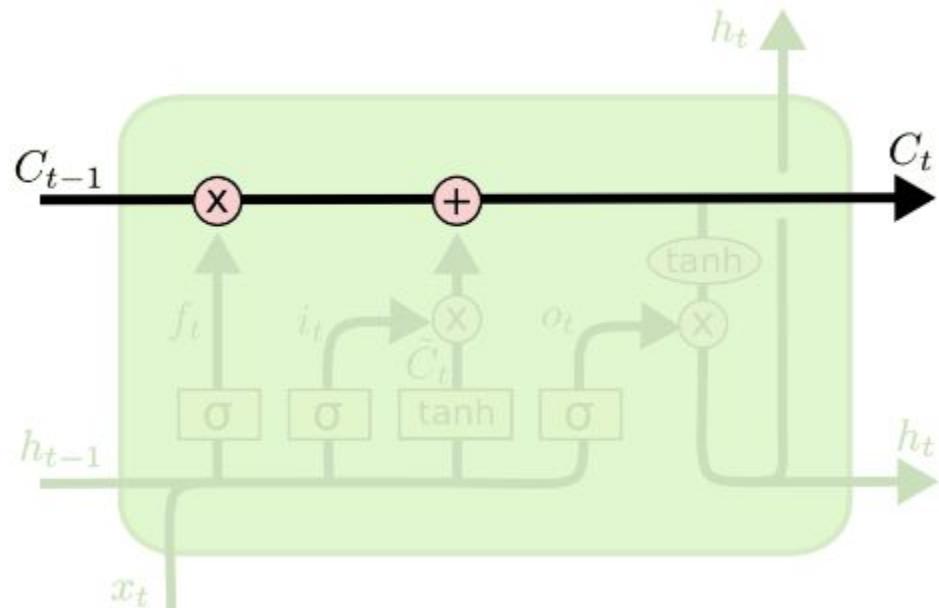


LSTM-core idea



LSTM-cell state

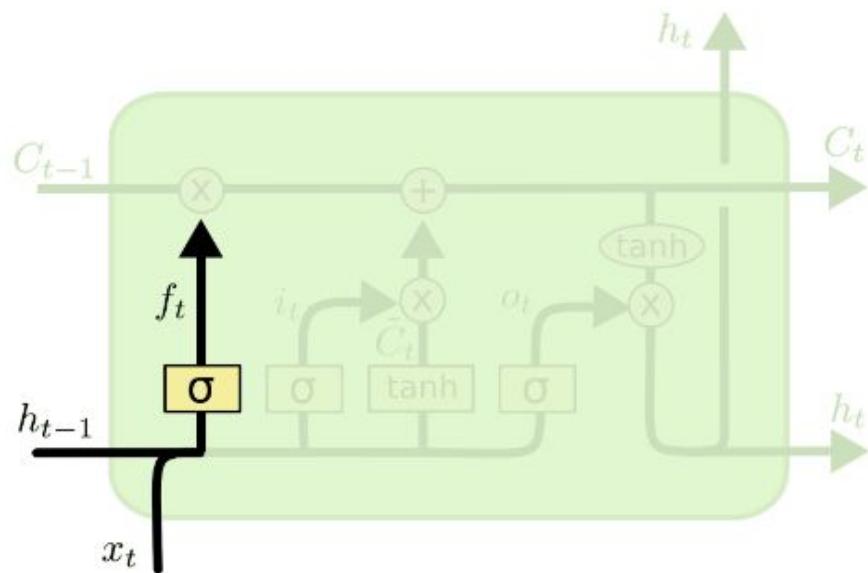
LSTM removes or adds information to the cell state, carefully regulated by structures called gates.



LSTM-gates

Each LSTM unit comprises of three gates.

- **Forget Gate:** Amount of memory it should forget.
- Input Gate
- Output Gate

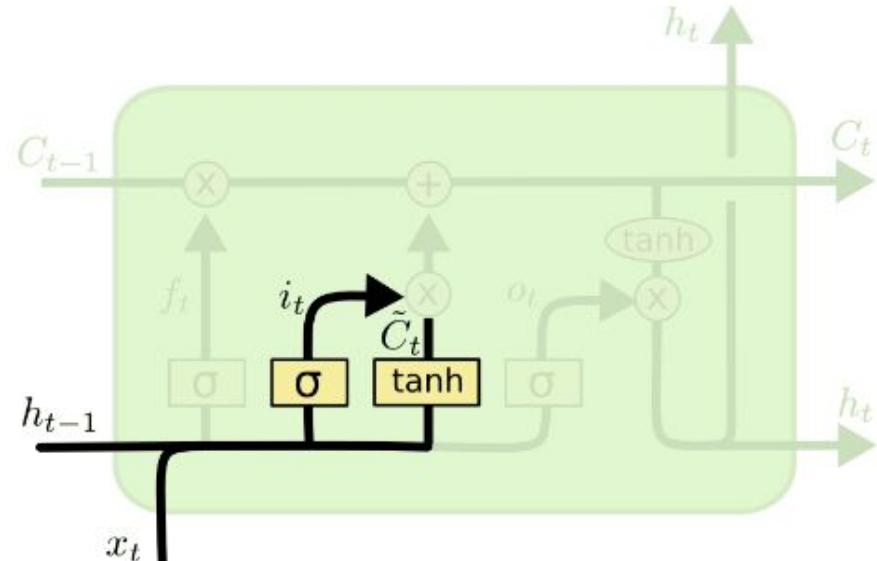


$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$

LSTM-gates

Each LSTM unit comprises of three gates.

- Forget Gate: Amount of memory it should forget.
- **Input Gate:** Amount of new information it should memorize.
- Output Gate



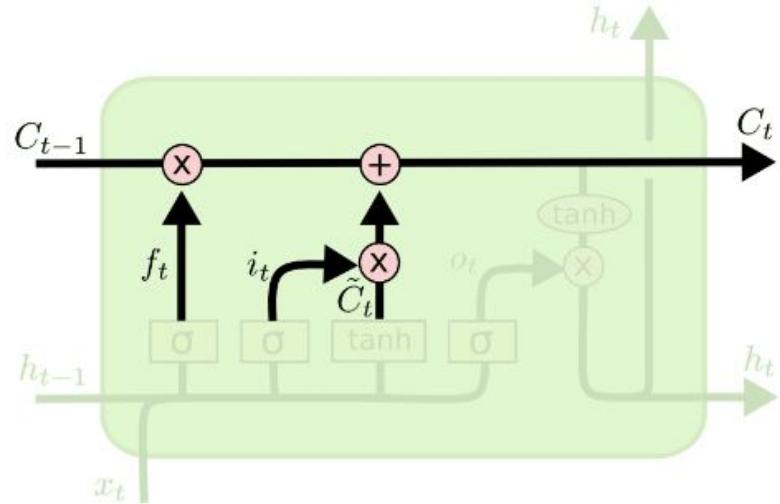
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

LSTM-gates

Each LSTM unit comprises of three gates.

- **Forget Gate:** Amount of memory it should forget.
- **Input Gate:** Amount of new information it should memorize.
- Output Gate:

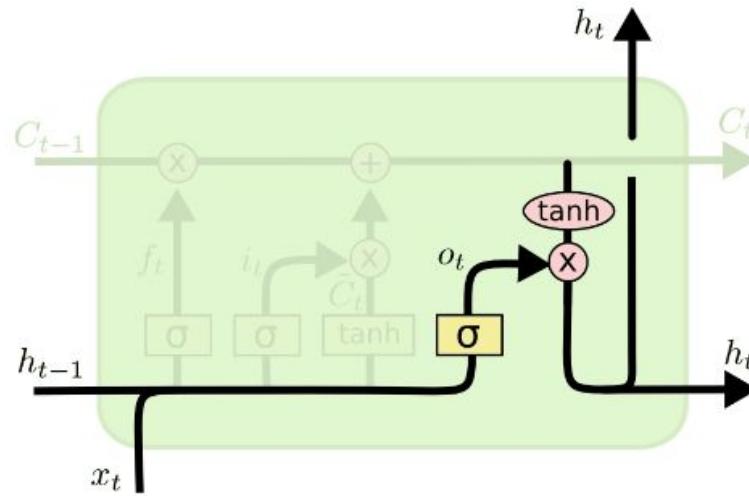


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM-gates

Each LSTM unit comprises of three gates.

- **Forget Gate:** Amount of memory it should forget.
- **Input Gate:** Amount of new information it should memorize.
- **Output Gate:** Amount of information it should pass to next unit.



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

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

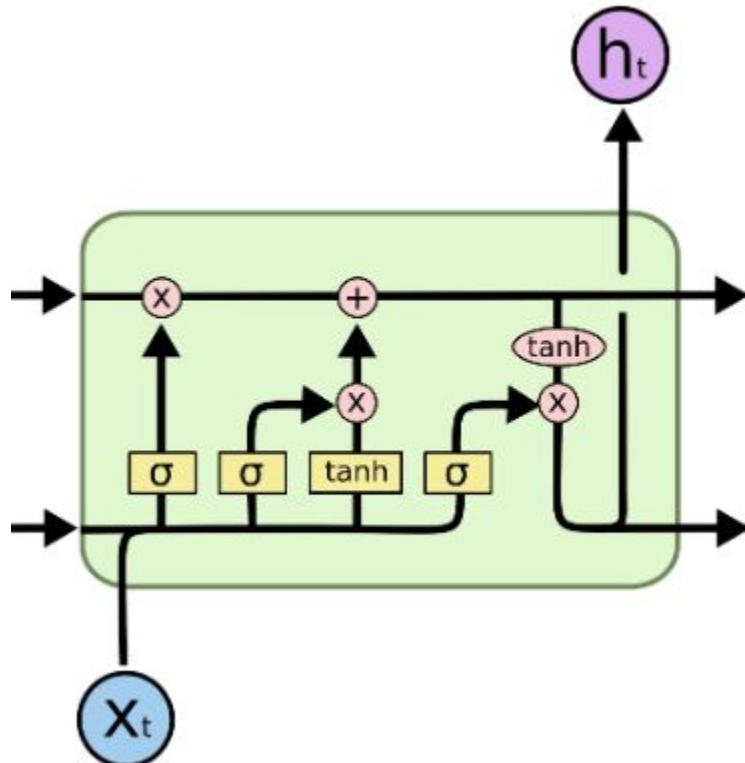
LSTM-mathematical intuition

What is h_t and c_t ?

h_t and c_t both are vectors of the same dimension.

Say $h_t = [0.5 \ 0.8 \ 0.3]$

Then, $c_t = [0.1 \ 0.5 \ 0.6]$



LSTM-mathematical intuition

What is x_t ?

x_t is an input vector. x_t Does not have any relation with h_t and c_t . Its size may be different.

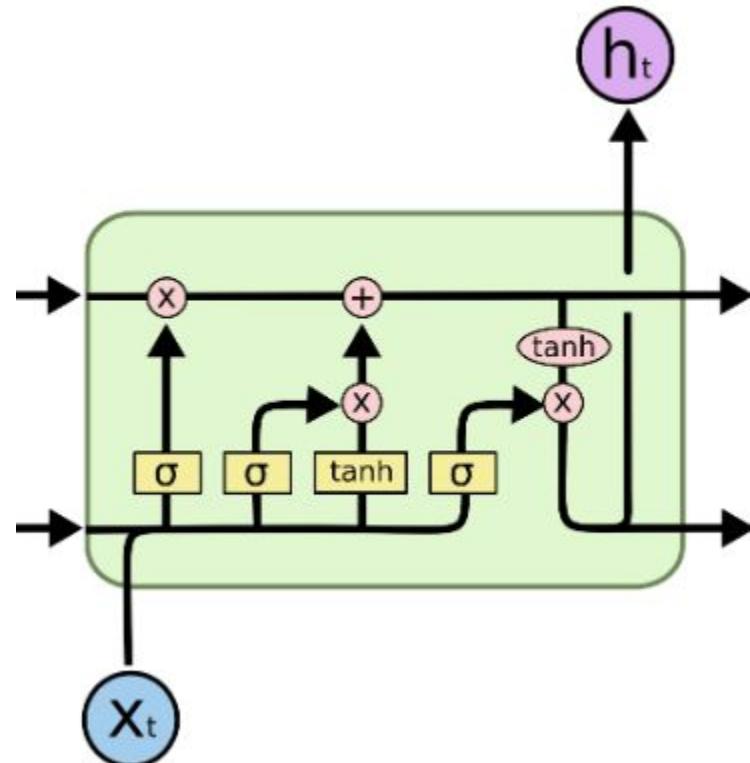
Say for the sentence:
This is classroom

This: [1 0 0]

Is: [0 1 0]

Class: [0 0 1]

This is classroom : [1 0 0] [0 1 0] [0 0 1]



LSTM-mathematical intuition

What is f_t , i_t , \tilde{C}_t , h_t

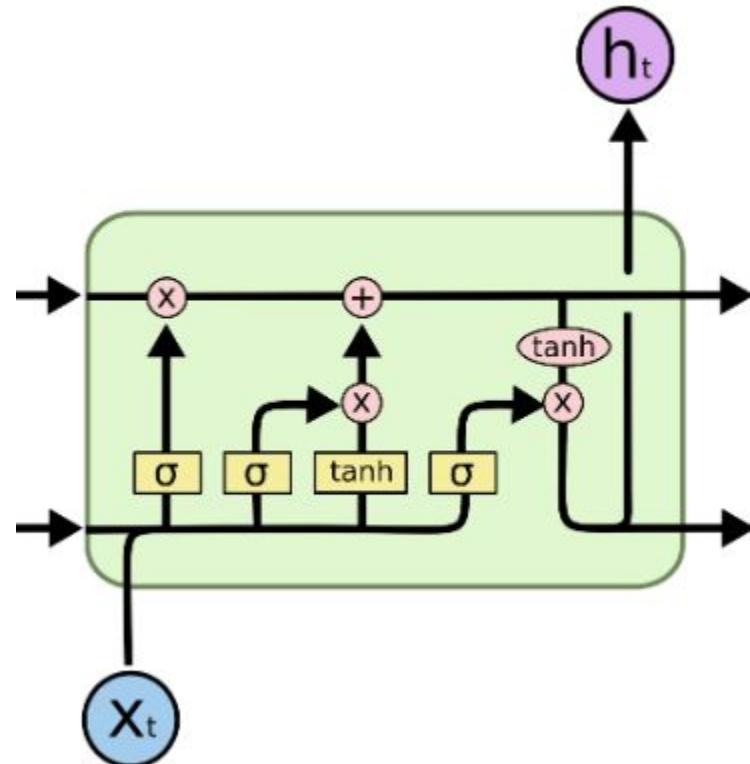
f_t : forget gate

i_t : input gate

\tilde{C}_t : input gate

h_t : output gate

All are vectors. All vectors have the same dimension as h_t and c_t .



LSTM vs GRU

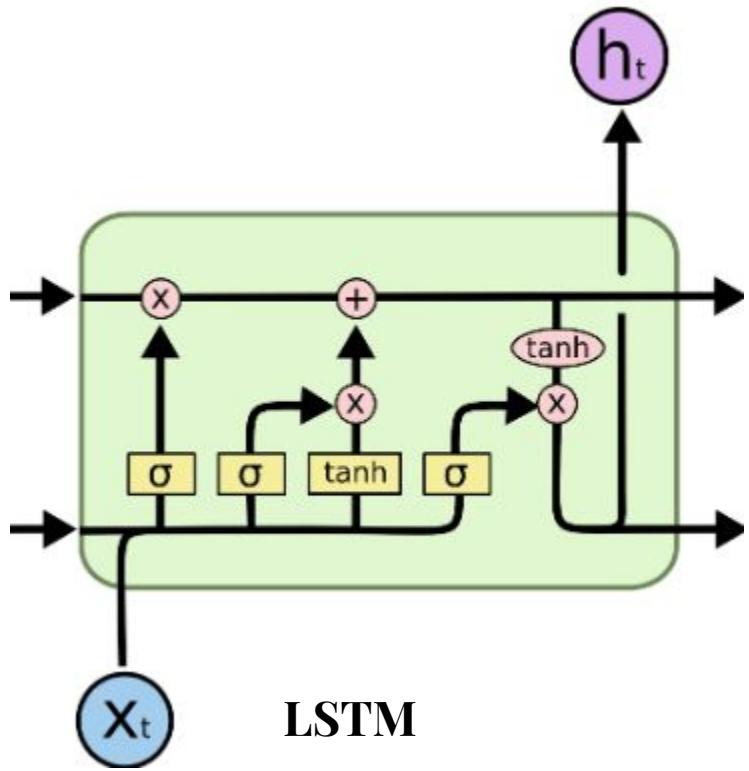
LSTM

- **Number of gates:** 3
 - Input, Forget, Output
- **Memory units:** 1
 - Cell state and hidden state
- **More parameters**
- **Computational complexity:** high
 - Due to extra gate and cell state
- **Empirical Performance:**
 - Slightly better with complex data

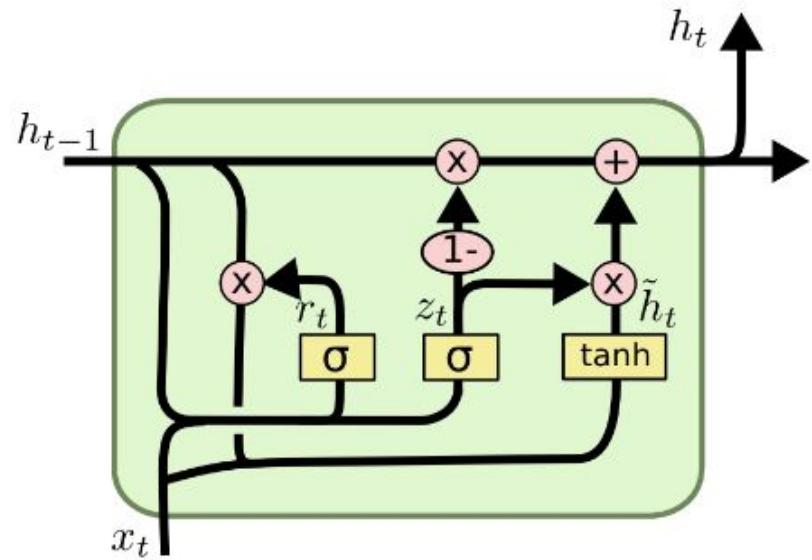
GRU

- **Number of gates:** 2
 - Reset and update
- **Memory units:** 2
 - Single hidden state
- **Less parameters**
- **Computational complexity:** low
 - Simpler and faster to compute
- **Empirical Performance:**
 - Comparable to LSTM with simple and small data

LSTM vs GRU



LSTM



GRU

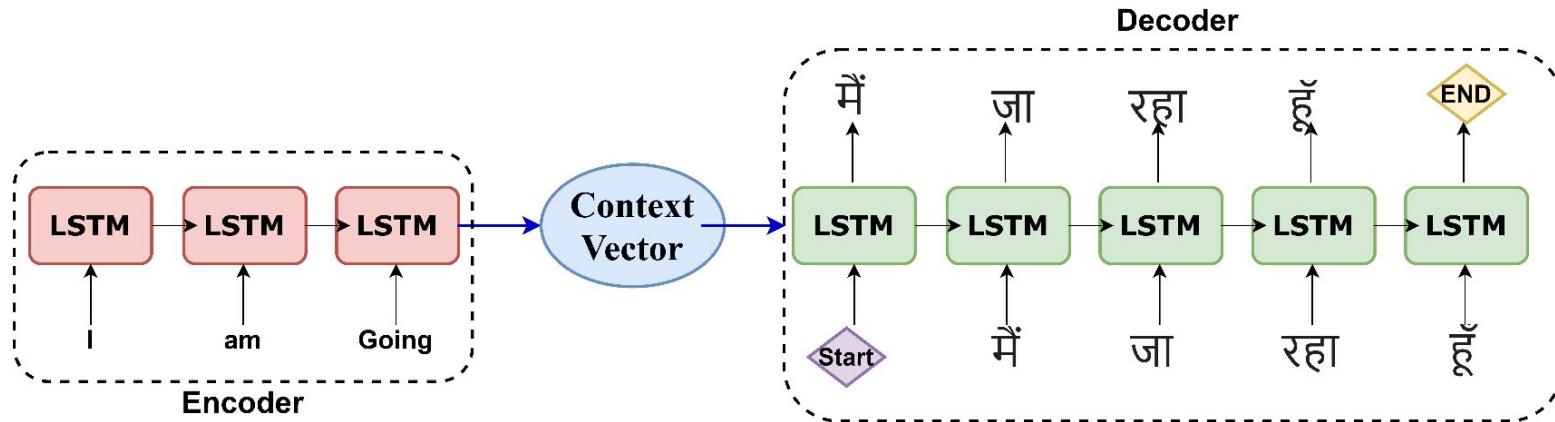
LLM developments

- Stage 1 (2014): Encoder- Decoder
- Stage 2 (2015): Attention
- Stage 3 (2017): Transformer
- Stage 4 (2018): Transfer Learning
- Stage 5 (2018): Large Language Model

LLM developments

- Stage 1 (2014): Encoder- Decoder
- Stage 2 (2015): Attention
- Stage 3 (2017): Transformer
- Stage 4 (2018): Transfer Learning
- Stage 5 (2018): Large Language Model

Sequence to Sequence Language Model



LLM evolution starts with Seq to Seq model¹

Challenges:

- Variable length Input and Output

Sequence to Sequence: Demo

Parallel corpus in my dataset:

1. I am going → मैं जा रहा हूँ
2. Think → सोचना

Preprocessing: English Tokens

['i', 'am', 'going', 'think']

Converting English tokens into one-hot vector:

'I': → [1, 0, 0, 0]

'am' → [0, 1, 0, 0]

'going': → [0, 0, 1, 0]

'think' → [0, 0, 0, 1]

Preprocessing: Hindi Tokens

[मैं, जा, रहा, हूँ, सोचना]

Converting English tokens into one-hot vector:

'Start': → [1, 0, 0, 0, 0, 0, 0]

'मैं': → [0, 1, 0, 0, 0, 0, 0]

'जा': → [0, 0, 1, 0, 0, 0, 0]

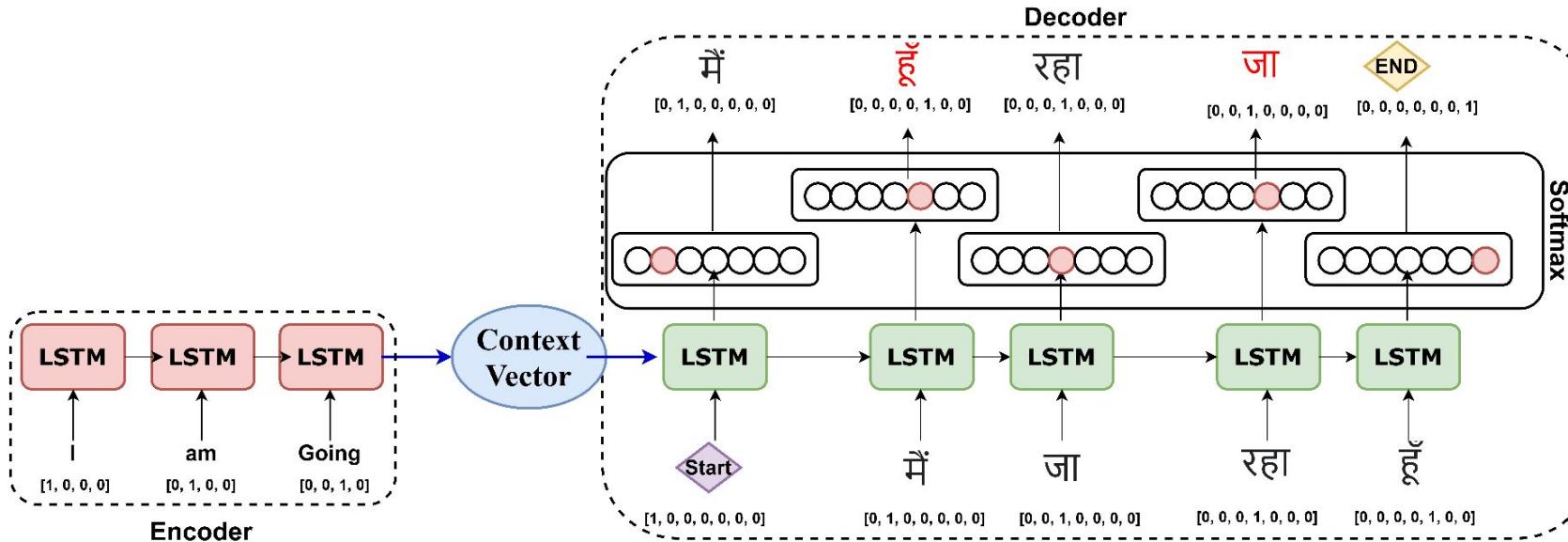
'रहा': → [0, 0, 0, 1, 0, 0, 0]

'हूँ': → [0, 0, 0, 0, 1, 0, 0]

'सोचना': → [0, 0, 0, 0, 0, 1, 0]

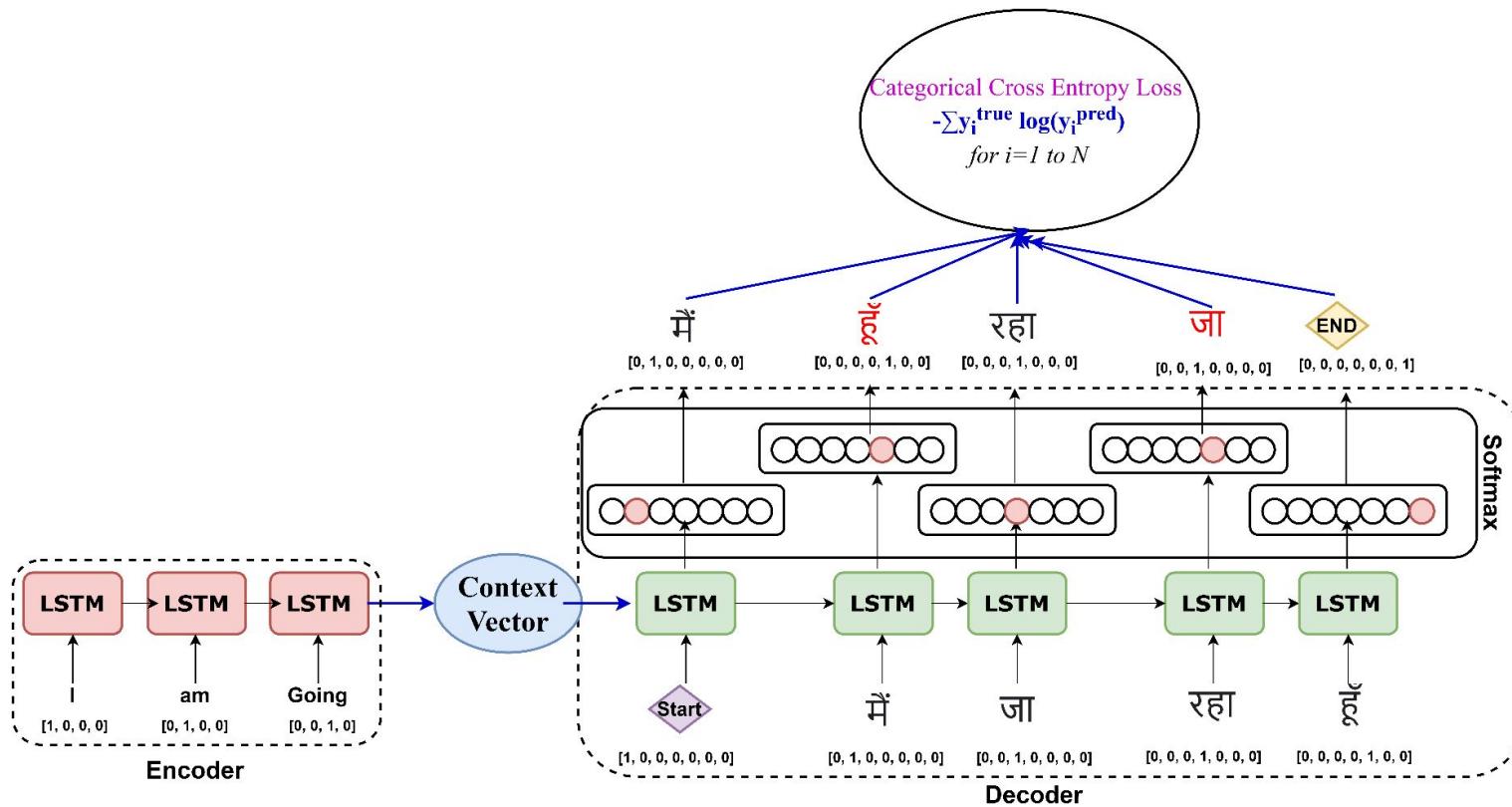
'End': → [0, 0, 0, 0, 0, 0, 1]

Sequence to Sequence: Training

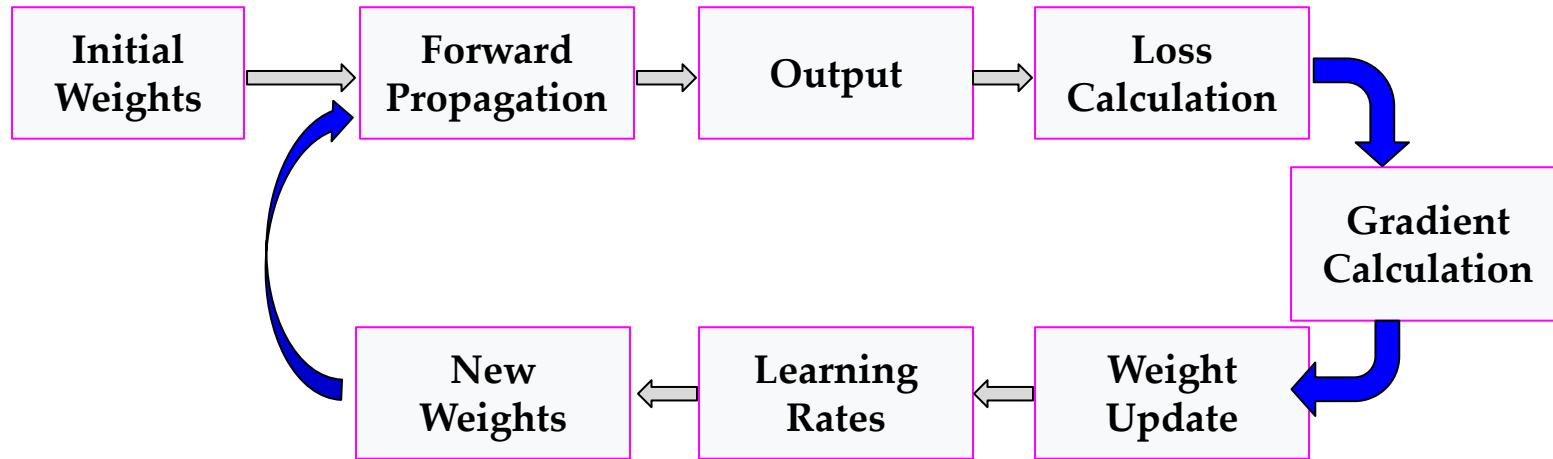


Teacher Forcing: input to the decoder is independent of individual time step output.

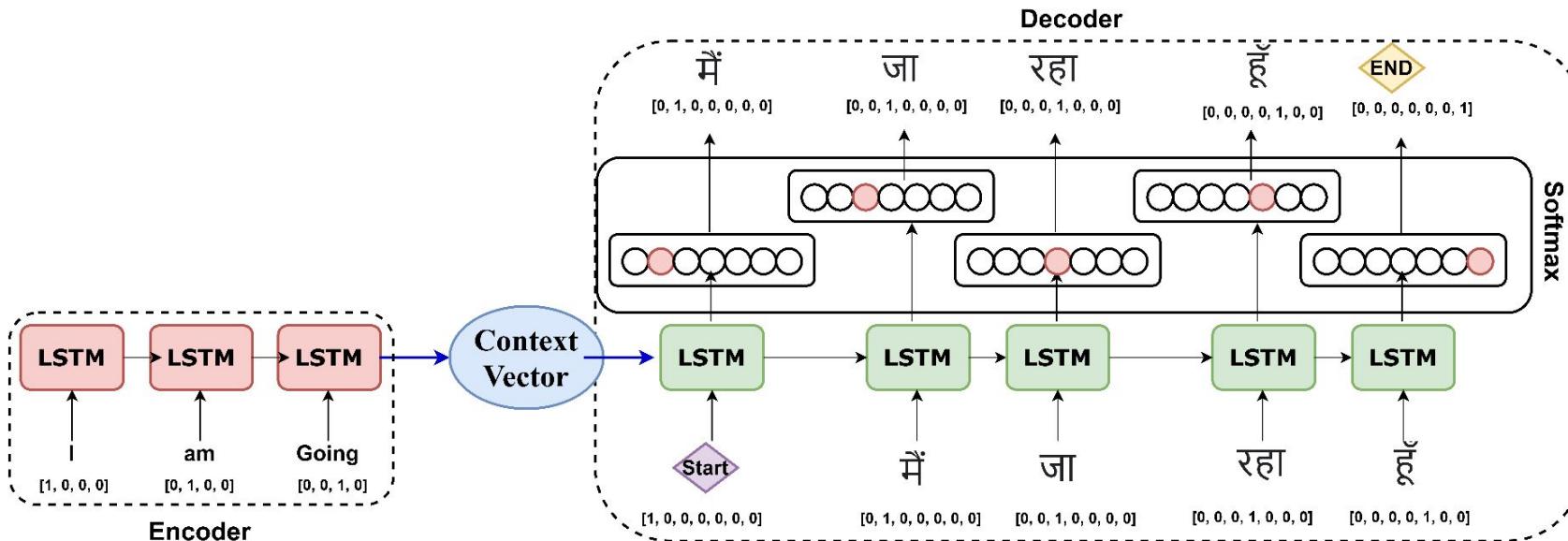
Sequence to Sequence: Training



Sequence to Sequence: Training



Sequence to Sequence: Inference

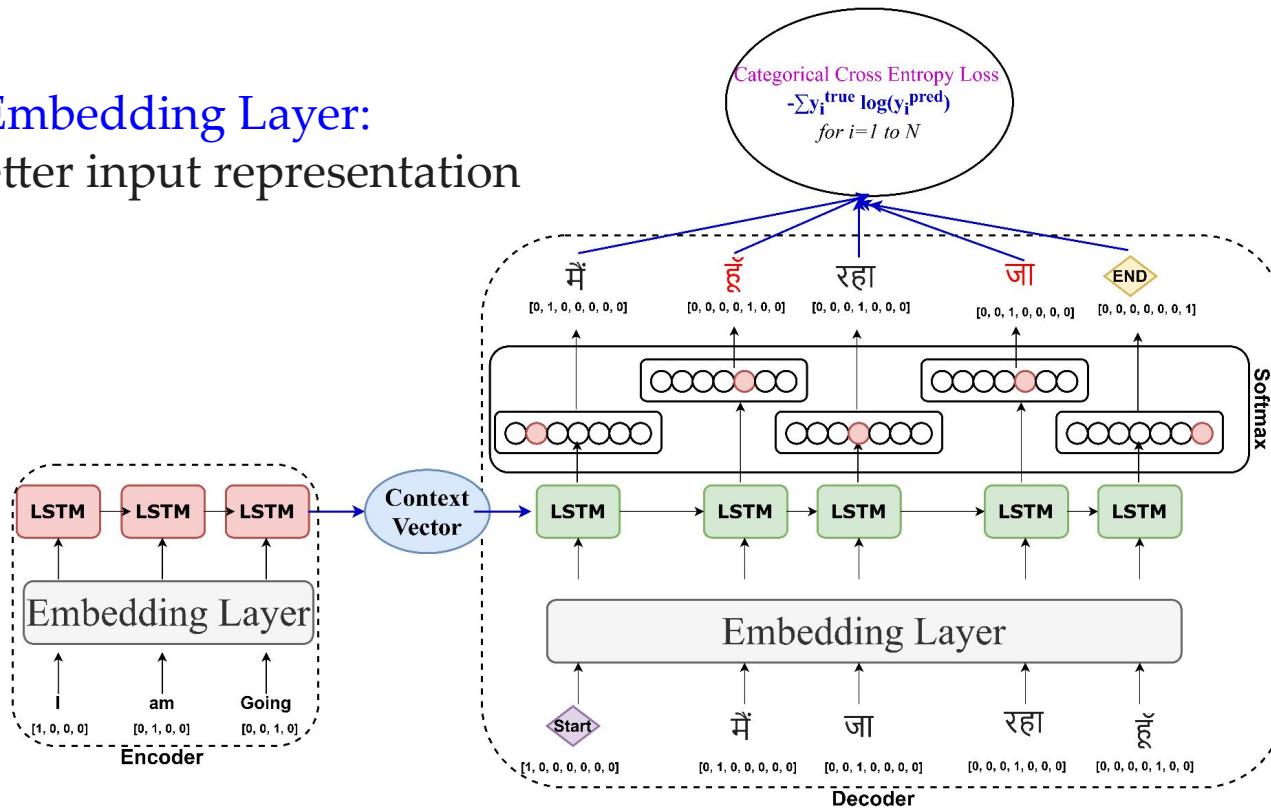


Input to the decoder's each time step is equivalent to the output of previous time step (except the 1st one).

Sequence to Sequence: Improvements 1

Deep Embedding Layer:

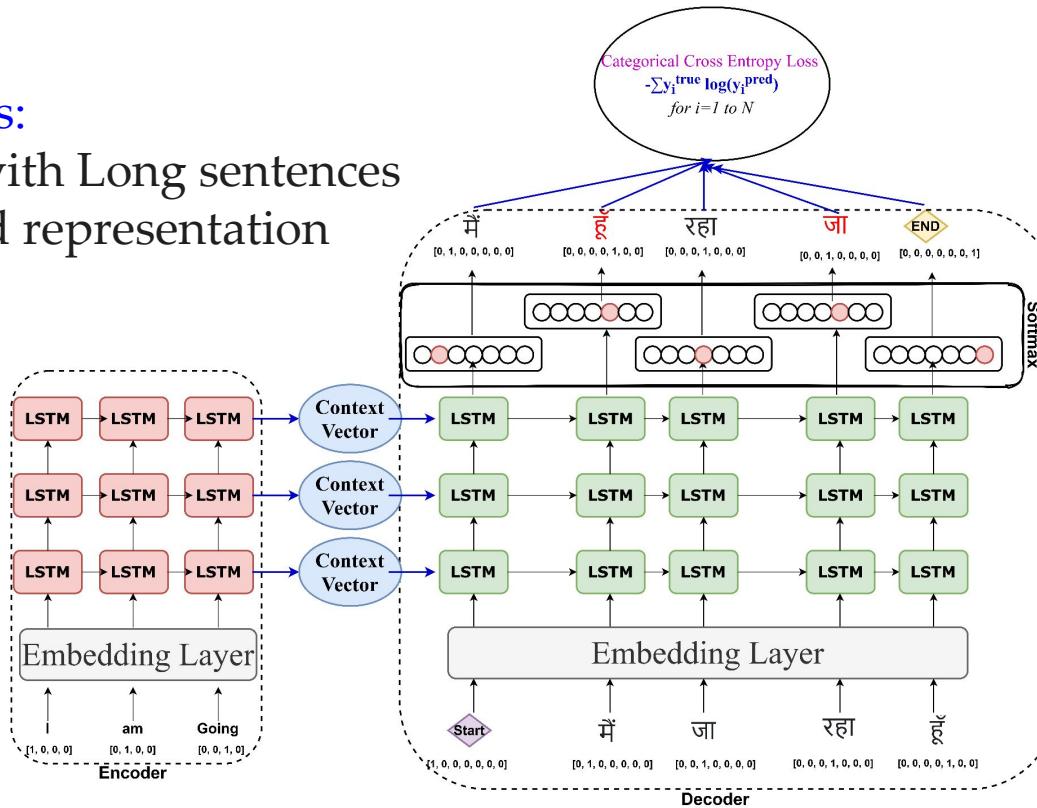
- Better input representation



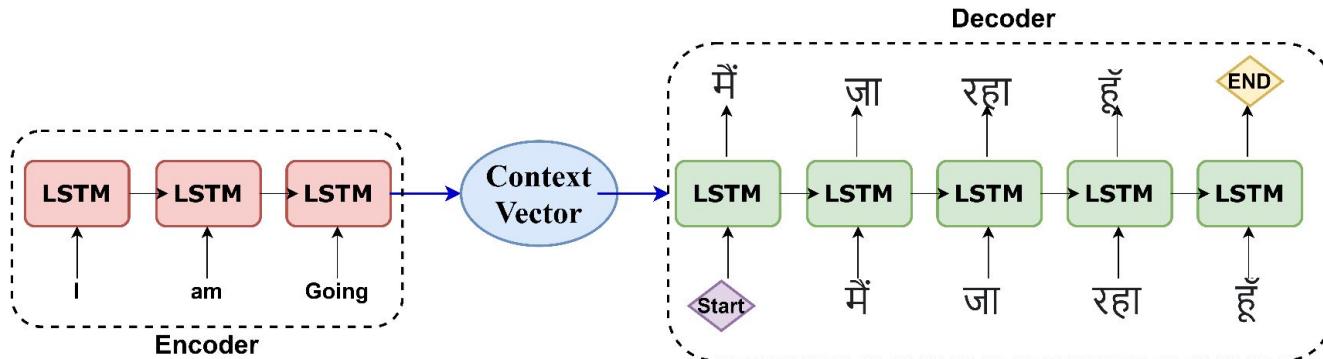
Sequence to Sequence: Improvements 2

Deep LSTMs:

- Good with Long sentences
- Layered representation



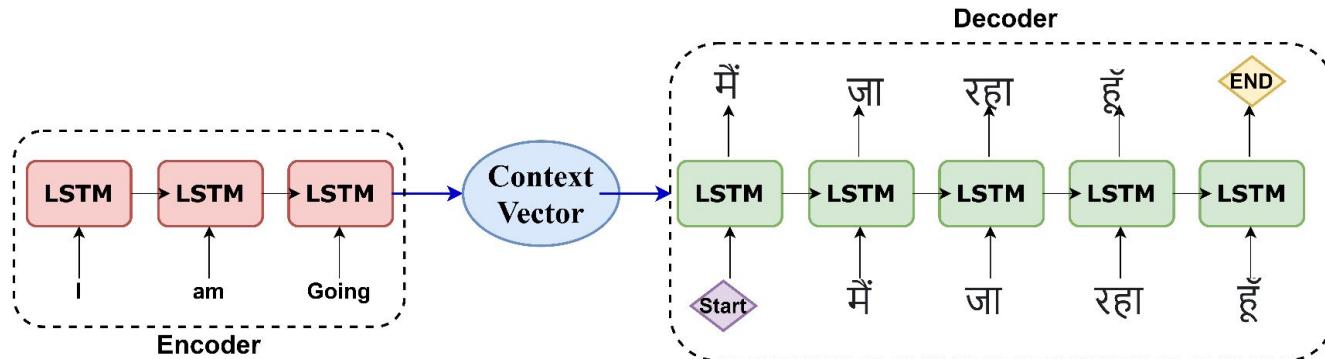
Sequence to Sequence: Limitations



Problem with Long Range Dependencies: (Converting following text into Hindi)

"Faculty development programs offer a double-edged sword. On the one hand, they can hone teaching skills, foster research, and spark innovation, leading to higher student engagement and success. However, poorly designed programs can be costly, time-consuming, and disruptive, potentially demotivating participants. Finding the right balance between effective training and respecting educators' autonomy is crucial for reaping the benefits of faculty development while minimizing drawbacks."

Sequence to Sequence: Limitations



Problem with Long Range Dependencies:

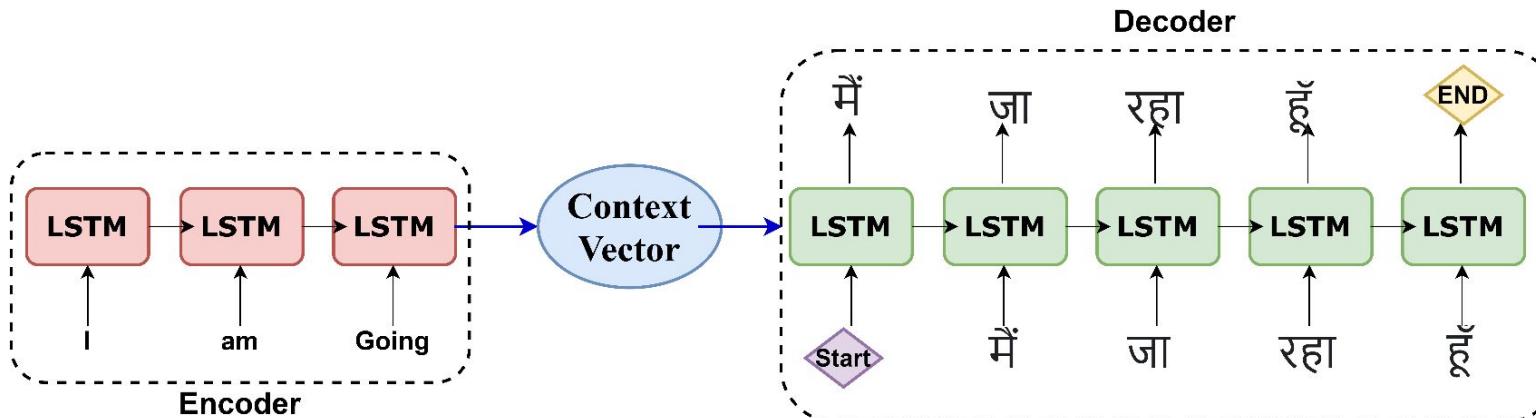
From Encoder side:

- Putting huge responsibility to **context vector** by summarizing the long sentence into a fixed length in one shot.

From Decoder side:

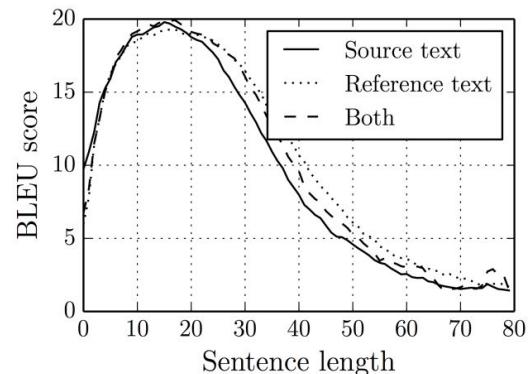
- At every time step in Decoder complete context vector is provided for next word generation which creates problem for decoder.
- Alternative would be to put attention to only a specific word/s in encoder for translation.

Sequence to Sequence: Limitations



Problem persists:

- Poor performance for longer sentences
- High Training Time



Attention in sequence



Human brain can put attention to only a word or span of words

Attention Mechanism Language model

At every time step of decoder, we need to focus on only a few words in encoder.

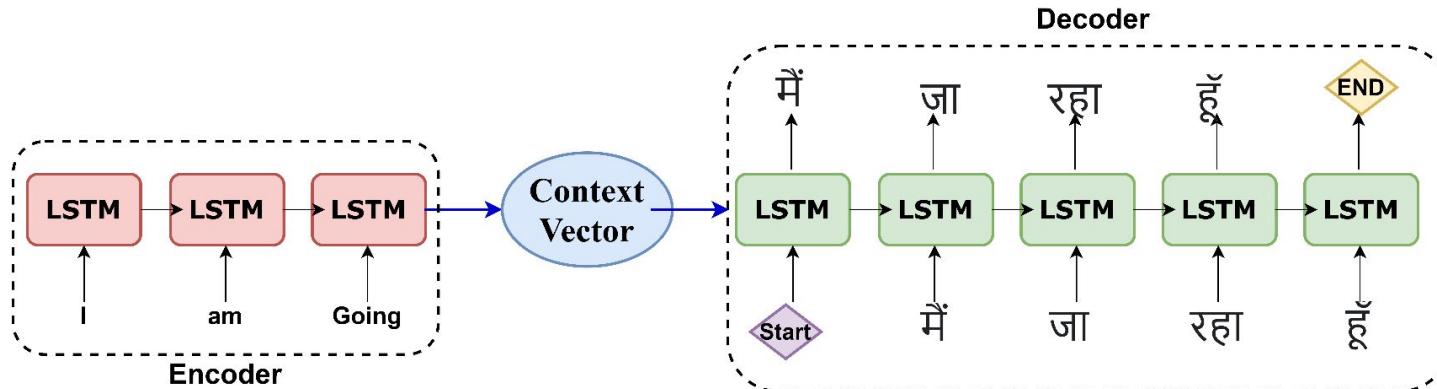
For <Start> → मैँ: Time step 1 in Decoder focus on word 'T' in Encoder

For मैँ → जा: Time step 2 in Decoder focus on word 'Going' in Encoder

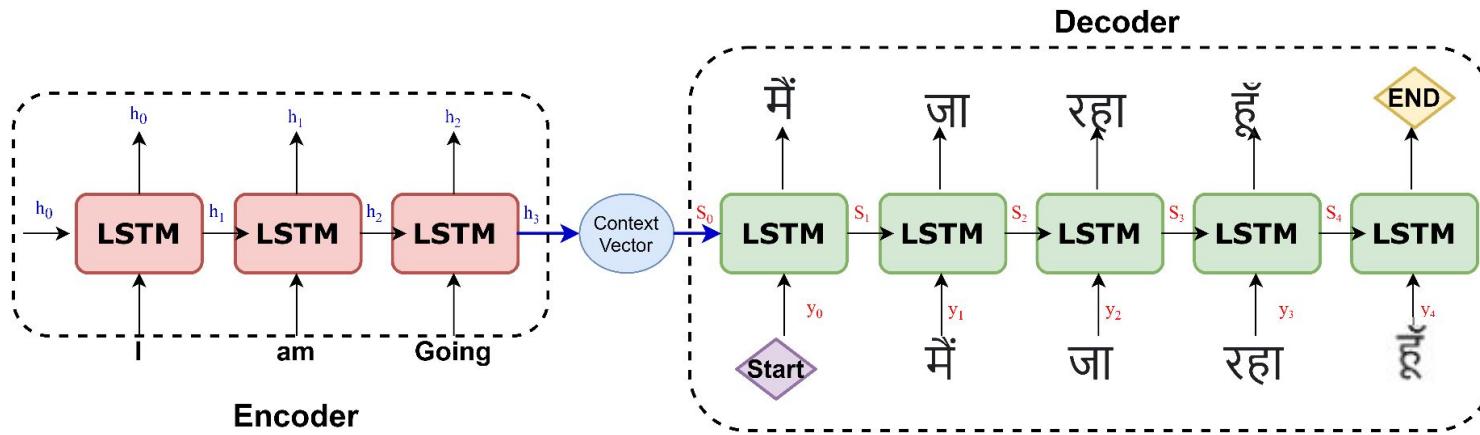
For जा → रहा: Time step 3 in Decoder focus on word 'Going' in Encoder

For रहा→हूँ: Time step 4 in Decoder focus on word 'am' in Encoder

For हूँ→<End>: Time step 5 in Decoder does not focus on any word in Encoder



Attention Mechanism Language model



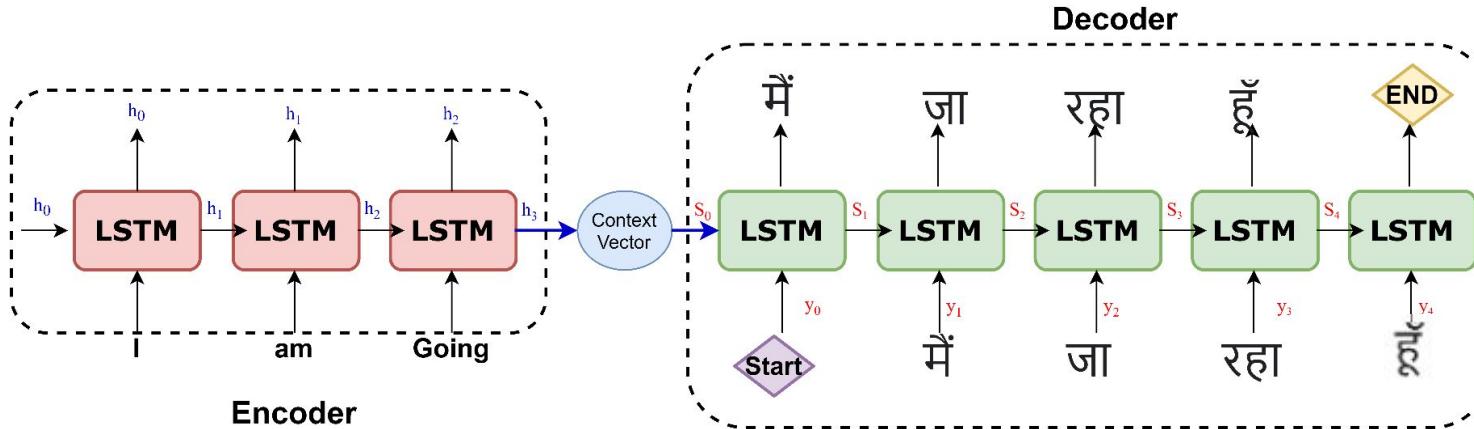
Attention Mechanism Language model

In Encoder-Decoder:

Input at step 1 of Decoder: $\rightarrow [y_0, S_0]$
Input at step 2 of Decoder: $\rightarrow [y_1, S_1]$
Input at step 3 of Decoder: $\rightarrow [y_2, S_2]$

In Attention-Mechanism:

Input at step 1 of Decoder: $\rightarrow [y_0, S_0, C_1]$
Input at step 2 of Decoder: $\rightarrow [y_0, S_0, C_2]$

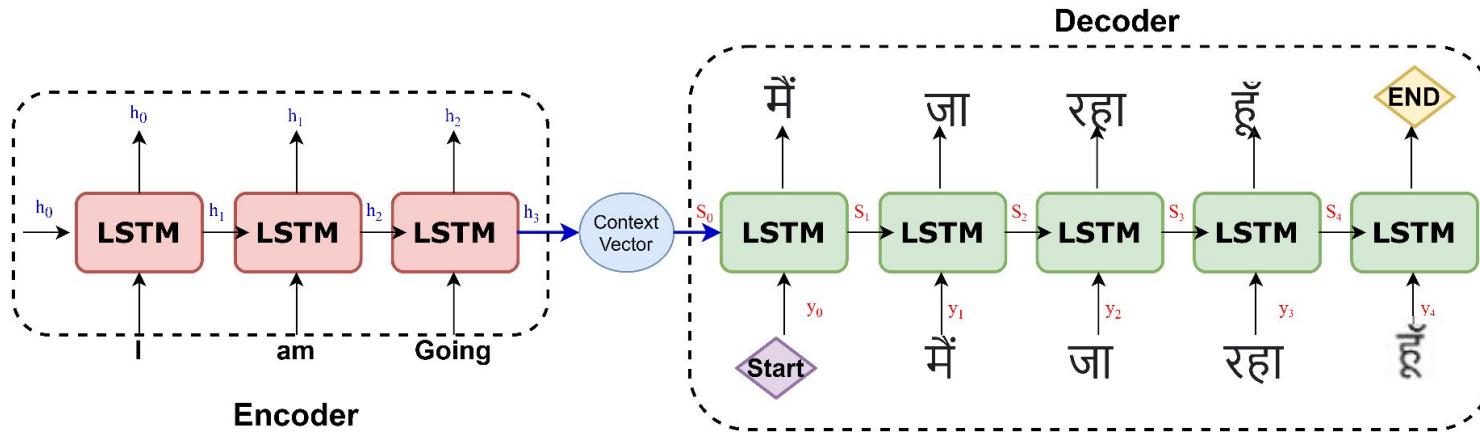


Attention Mechanism Language model

In general Attention-Mechanism:

Input at step i of Decoder: $\rightarrow [y_{i-1}, S_{i-1}, C_i]$

Input at step 1 of Decoder: $\rightarrow [y_0, S_0, C_1]$



Attention Mechanism Language model

What is this C_i :

Is it a Vector?

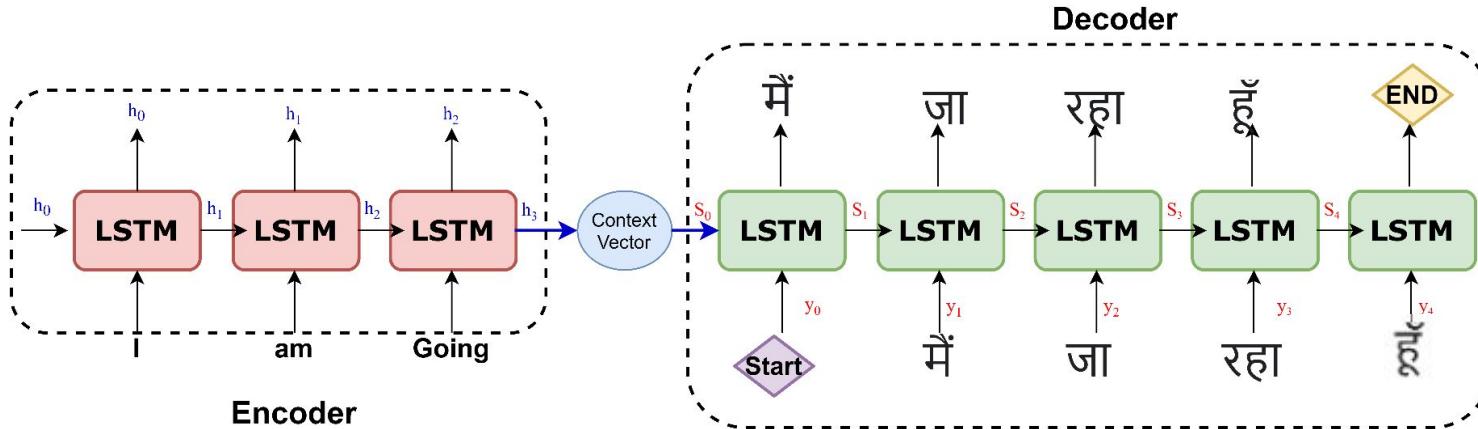
Is it a Scalar?

Is it a Matrix?

In general Attention-Mechanism:

Input at step i of Decoder: $\rightarrow [y_{i-1}, S_{i-1}, C_i]$

Input at step 1 of Decoder: $\rightarrow [y_0, S_0, C_2]$



Attention Mechanism Language model

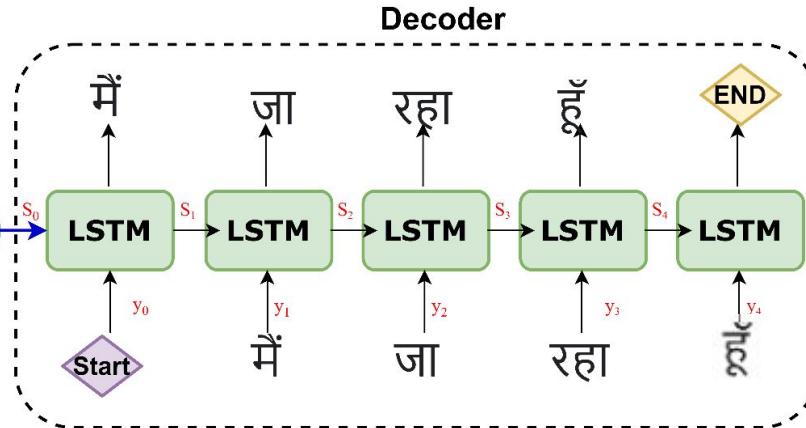
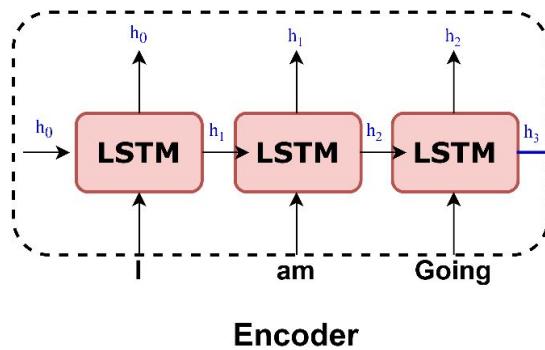
What is this C_i :

C_i represents the weightage of every Input at step i of Decoder: $\rightarrow [y_{i-1}, S_{i-1}, C_i]$ time stamp of Encoder.

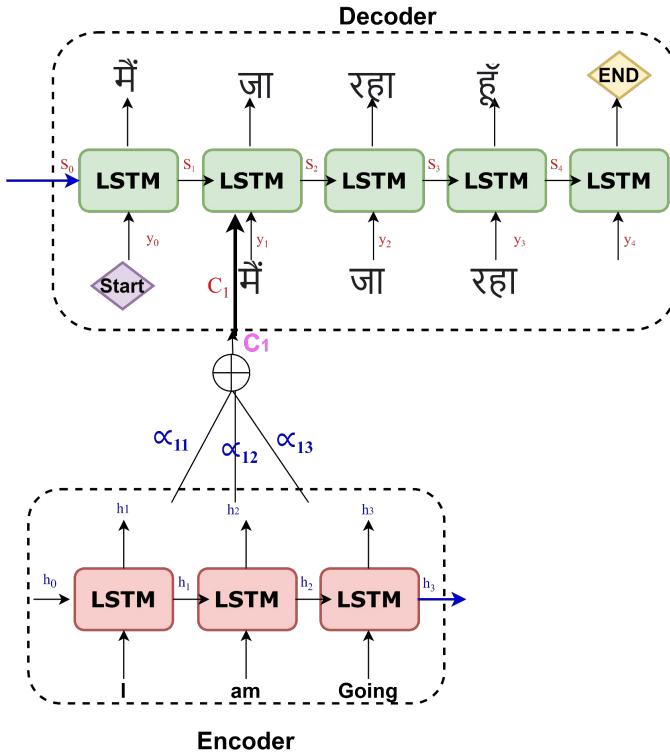
Therefore, C_i must be a vector.
Dimension of C_i depends on h_i .

In general Attention-Mechanism:

Input at step 1 of Decoder: $\rightarrow [y_0, S_0, C_2]$



Attention Mechanism Language model



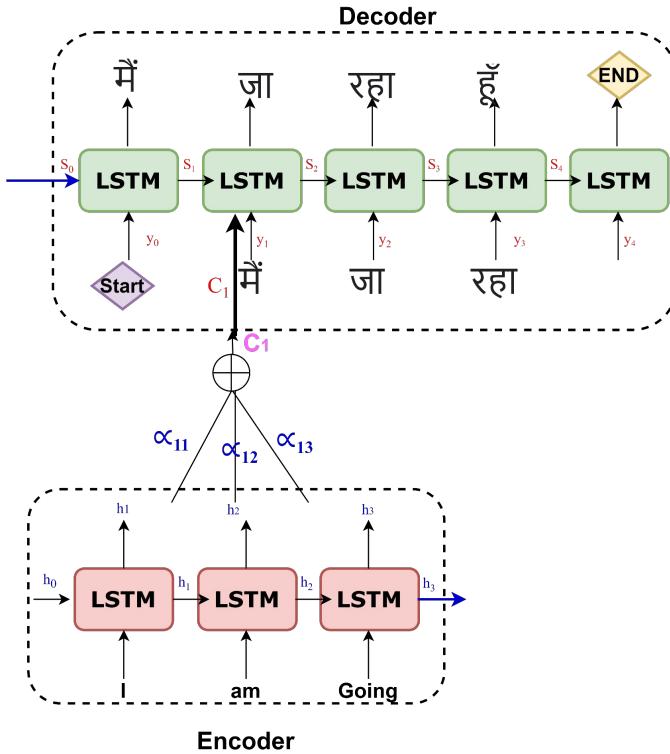
In general Attention-Mechanism:

Input at step i of Decoder: $\rightarrow [y_{i-1}, S_{i-1}, C_i]$

Input at step 1 of Decoder: $\rightarrow [y_0, S_0, C_1]$

$$\begin{aligned} C_1 &= \infty_{11} h_1 + \infty_{12} h_2 + \infty_{13} h_3 \\ C_2 &= \infty_{21} h_1 + \infty_{22} h_2 + \infty_{23} h_3 \end{aligned}$$

Attention Mechanism Language model



In general Attention-Mechanism:

Input at step i of Decoder: $\rightarrow [y_{i-1}, S_{i-1}, C_i]$

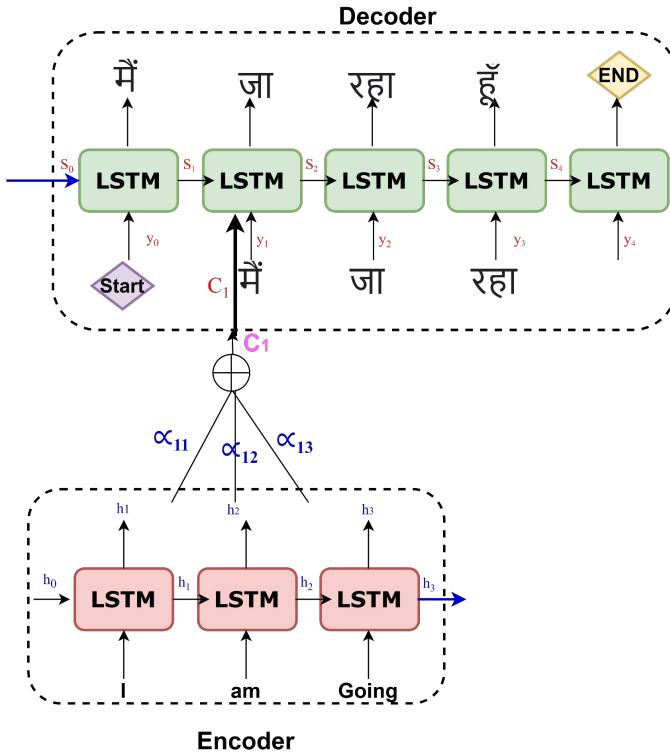
Input at step 1 of Decoder: $\rightarrow [y_0, S_0, C_1]$

$$C_1 = \alpha_{11} h_1 + \alpha_{12} h_2 + \alpha_{13} h_3$$
$$C_2 = \alpha_{21} h_1 + \alpha_{22} h_2 + \alpha_{23} h_3$$

$$C_i = \sum_j \alpha_{ij} h_j$$

How to calculate α_{ij} ?

Attention Mechanism Language model



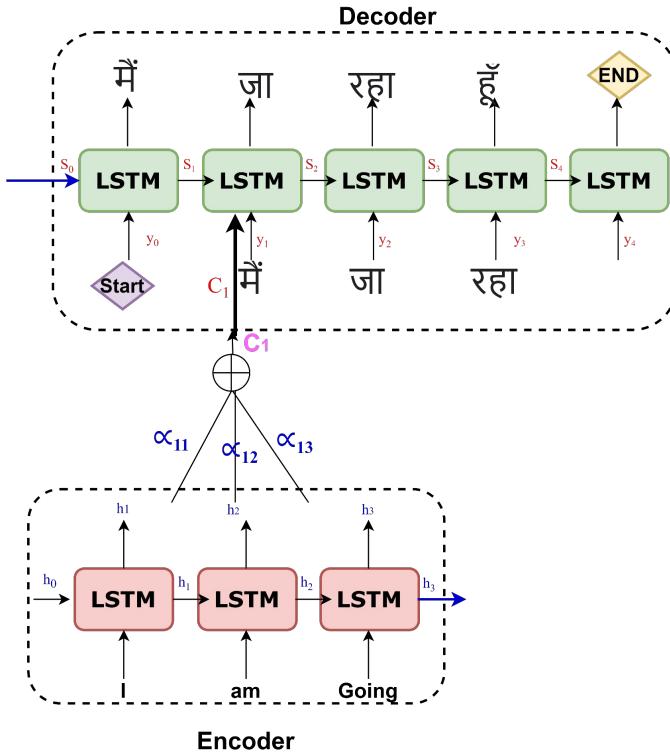
How to calculate α_{ij} ?

α_{ij} gives the alignment score:

For example α_{21} tells that:

- $i=2$ to print the output at 2nd time stamp of decoder
- $j=1$ what is the alignment/weightage of encoder's 1st time step

Attention Mechanism Language model



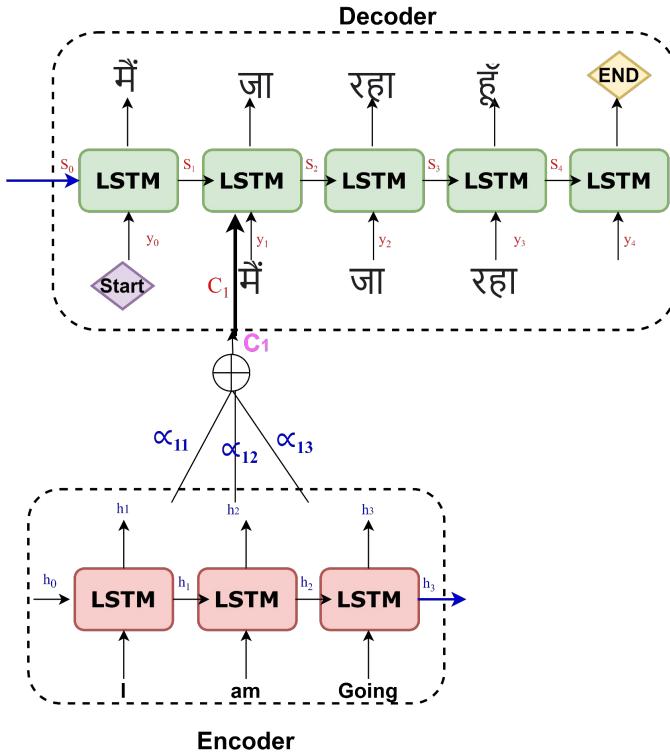
How to calculate ∞_{ij} ?

∞_{ij} gives the alignment score:

∞_{ij} depends on two factor h_j and S_{i-1} .

Given translation in the decoder (S_{i-1}) what is role of encoder's j th time step (h_j) to predict the output at decoder i th time step.

Attention Mechanism Language model



How to calculate ∞_{ij} ?

∞_{ij} gives the alignment score:

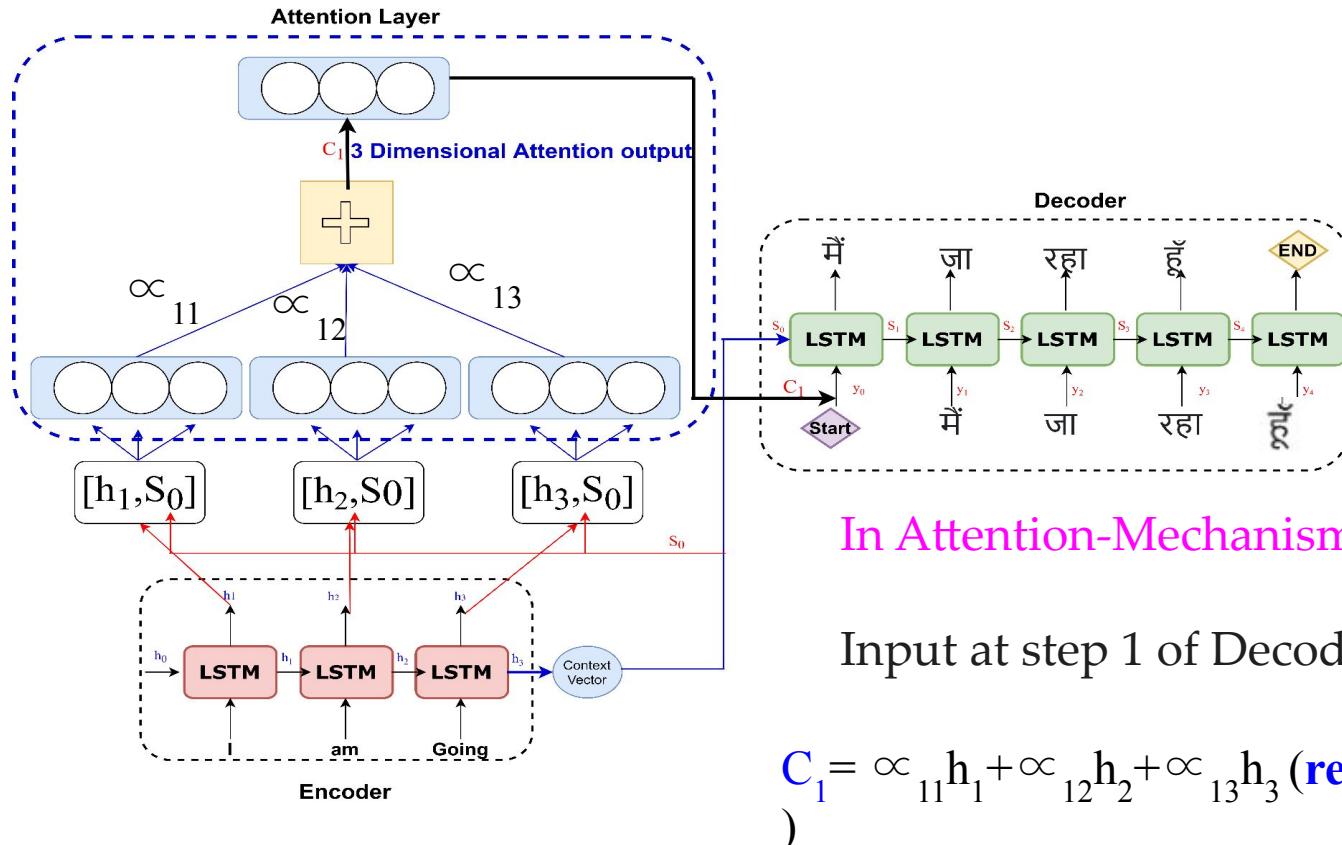
∞_{ij} depends on two factor h_j and S_{i-1} .

$$\infty_{11} = f(h_1, S_0)$$

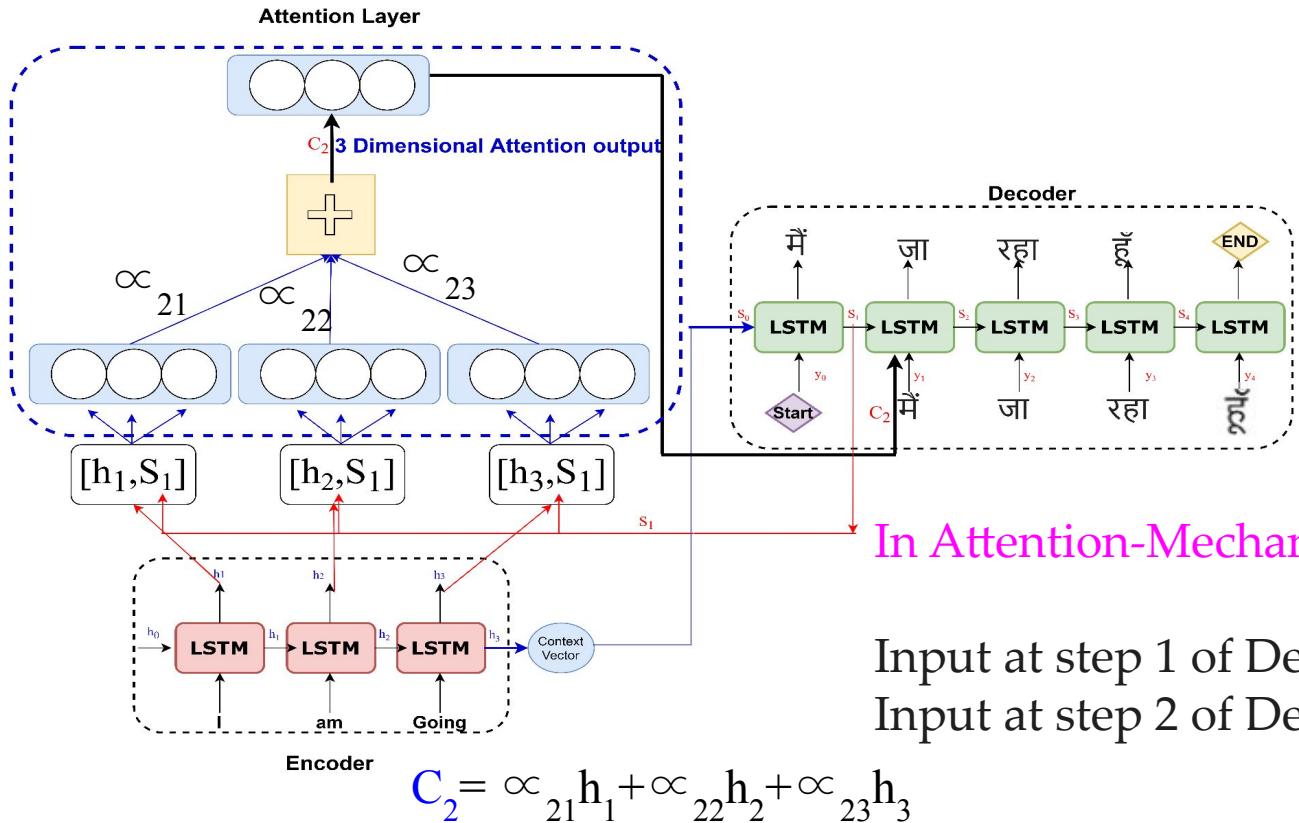
$$\infty_{12} = f(h_2, S_0)$$

$$\infty_{13} = f(h_3, S_0)$$

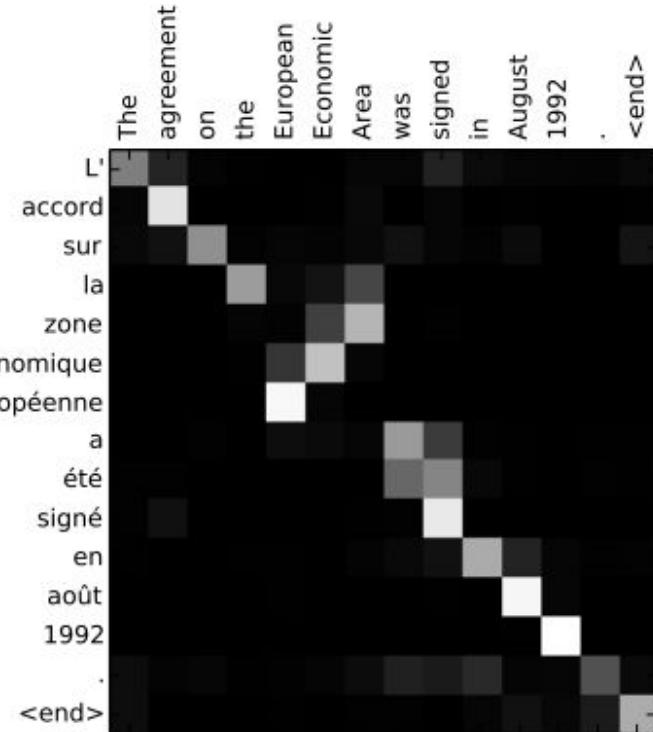
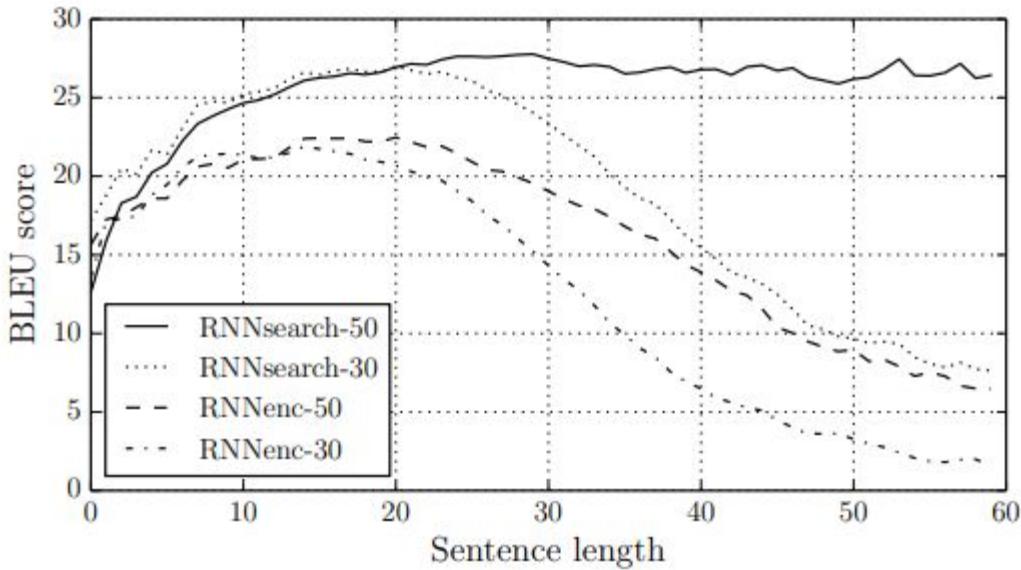
Attention Mechanism Language model



Attention Mechanism Language model



Attention Mechanism: performance



Attention Mechanism: Limitations

- Computation complexity is high.
 - at every time step output of decoder, attention of all words in encoder is being calculated.
 - For m words output in decoder, and n words in encoder, total matrix multiplication was $n*m$, $O(n^2)$
- Training time is high
 - Between 2015-2017, various attention mechanism were introduced to reduce this computational complexity.

Issue: Core problem was LSTMs (its sequential nature of computing output) and not attention.

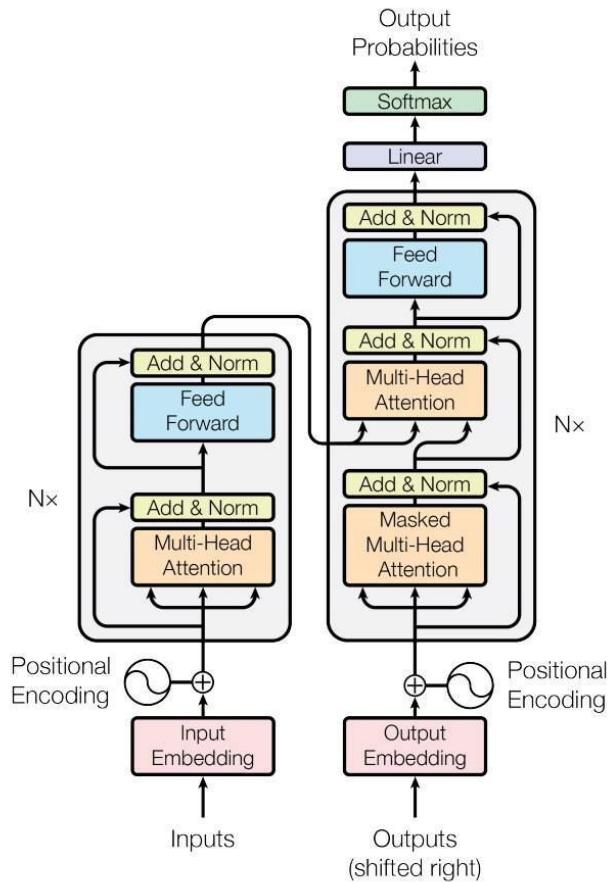
Next research was to bring parallel computing in the picture.

Transformer network for Language Modeling

- LSTM is no more required for sequential data.
- Attention is all you need.
 - Self-Attention
- Parallel processing to train the model.

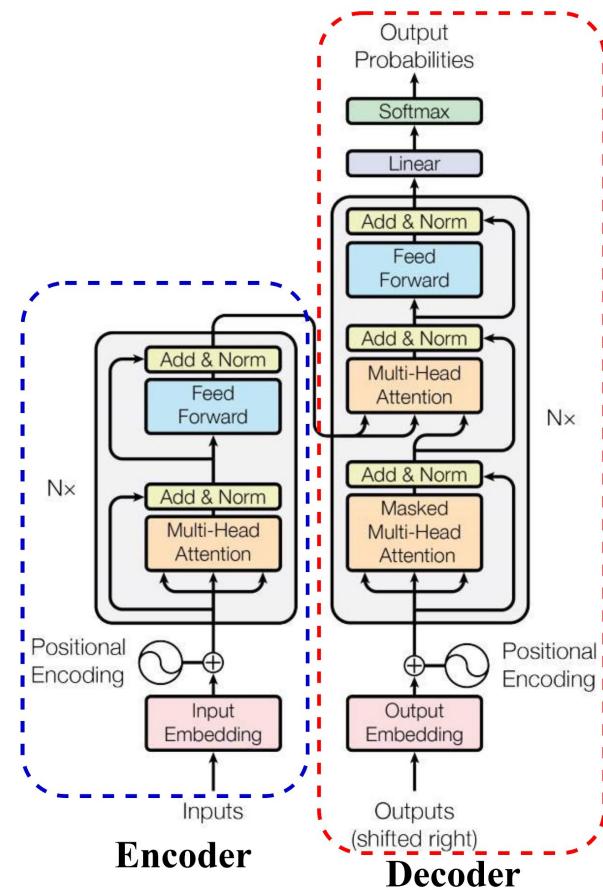
Transformer architecture

- LSTM is no more required for sequential data.
- Attention is all you need.
 - Self-Attention
- Parallel processing to train the model.



Transformer architecture

- Transformer network is made up of Encoder-Decoder modules.
- The objective of transformer encoder is same as Seq2Seq Encoder with parallelization of inputs.



Word Representation: Meaning of a word

- To perform language modelling effectively, it is necessary to capture the **meaning** of each word.
- We need effective representation of words.
- The simplest way to represent word is “**One-Hot Vector**”.
 - Means one 1, the rest 0s
 - For every word we create a vector where one position, the index of that word, is 1 and all other positions are 0.

Word Representation: One-Hot Vector

Parallel corpus in my dataset:

I am going → मैं जा रहा हूँ
Think → सोचना

Preprocessing: English Tokens
['i', 'am', 'going', 'think']

Word Embeddings (as one-hot vector):
'I': → [1, 0, 0, 0]
'am' → [0, 1, 0, 0]
'going': → [0, 0, 1, 0]
'think' → [0, 0, 0, 1]

Preprocessing: Hindi Tokens
[मैं, जा, रहा, हूँ, सोचना]

Word Embeddings (as one-hot vector):
'Start': → [1, 0, 0, 0, 0, 0, 0]
'मैं' → [0, 1, 0, 0, 0, 0, 0]
'जा': → [0, 0, 1, 0, 0, 0, 0]
'रहा' → [0, 0, 0, 1, 0, 0, 0]
'हूँ': → [0, 0, 0, 0, 1, 0, 0]
'सोचना' → [0, 0, 0, 0, 0, 1, 0]
'End': → [0, 0, 0, 0, 0, 0, 1]

Word Representation: Bow:- Count Vector

Parallel corpus in my dataset:

I am going → मैं जा रहा हूँ
Think → सोचना

Preprocessing: English Tokens

['i', 'am', 'going', 'think']

Word Embeddings (as count vector):

S1: → [1, 1, 1, 0]

S2 → [0, 0, 0, 1]

Preprocessing: Hindi Tokens

[Start, मैं, जा, रहा, हूँ, सोचना, End]

Word Embeddings (as count vector):

S1 → [1, 1, 1, 1, 1, 0, 1]

S2: → [1, 0, 1, 0, 0, 1, 1]

Word Representation: BoW:- TF-IDF Vector

Parallel corpus in my dataset:

I am going → मैं जा रहा हूँ
Think → सोचना

Preprocessing: English Tokens
['i', 'am', 'going', 'think']

Word Embeddings (as TF-IDF vector):

S1: → [1*log₂(2/1), 1*log₂(2/1), 1*log₂(2/1), 0*log₂(2/1)] → [1,1,1,0]

S2: → [0*log₂(2/1), 0*log₂(2/1), 0*log₂(2/1), 1*log₂(2/1)] → [0,0,0,1]

Preprocessing: Hindi Tokens

[Start, मैं, जा, रहा, हूँ, सोचना, End]

Word Embeddings (as TF-IDF vector):

S1 → [0, 1, 1, 1, 1, 0, 0]

S2: → [0, 0, 1, 0, 0, 1, 0]

Word Representation: BoW:- Co-occurrence Vector

Parallel corpus in my dataset:

I am going → मैं जा रहा हूँ
Think → सोचना

Preprocessing: English Tokens
['i', 'am', 'going', 'think']

Word Embeddings (window size =3):

	am	Going	think
I	1	1	0
Think	0	0	0

Preprocessing: Hindi Tokens

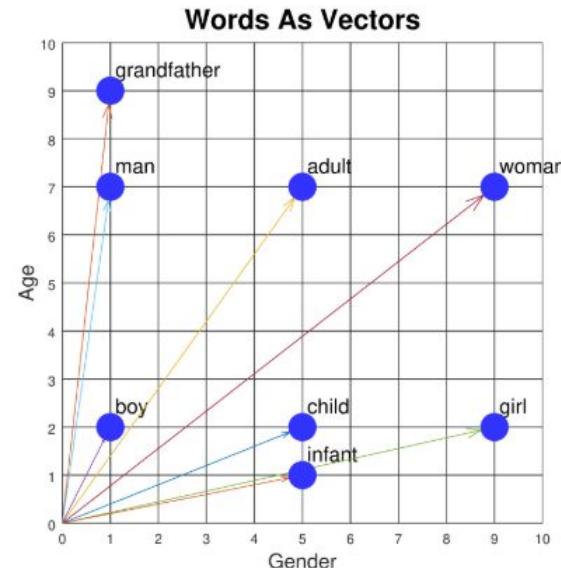
[Start, मैं, जा, रहा, हूँ, सोचना, End]

Word Embeddings (window size=3):

	मैं	सोचना	जा
जा	1	0	0
रहा	1	0	1

Word Representation

- So far we have seen frequency based word embedding techniques.
 - The word meaning is not able to capture.
- Other approach for word embedding is “**Prediction based word embeddings**”.
- Can capture semantic meaning.
 - **Similar context words have similar vectors**
- Similar words have similar representations.
- Word vectors are learned from the neural network
- Approaches:
 - Word2Vec
 - GloVe
 - FastText



Word Representation: NN approach

- Word Embeddings: word representation in vector

Parallel corpus in my dataset:

I am going → मैं जा रहा हूँ

Preprocessing: English Tokens

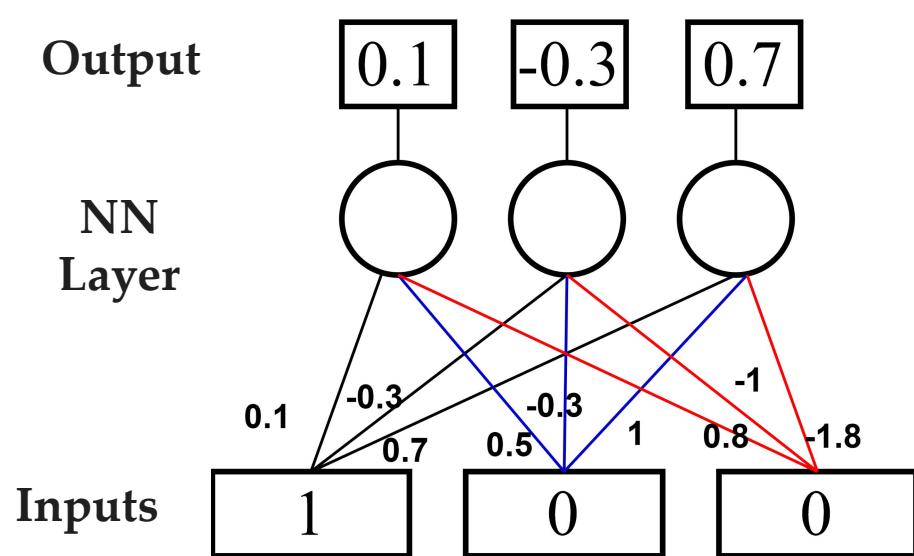
['i', 'am', 'going']

Word Embeddings (as dense vector):

'I': $\rightarrow [1, 0, 0] \rightarrow [0.1, -0.3, 0.7]$

'am' $\rightarrow [0, 1, 0]$

'going': $\rightarrow [0, 0, 1]$



Word Representation: NN approach

- Word Embeddings: word representation in vector

Parallel corpus in my dataset:

I am going → मैं जा रहा हूँ

Preprocessing: English Tokens

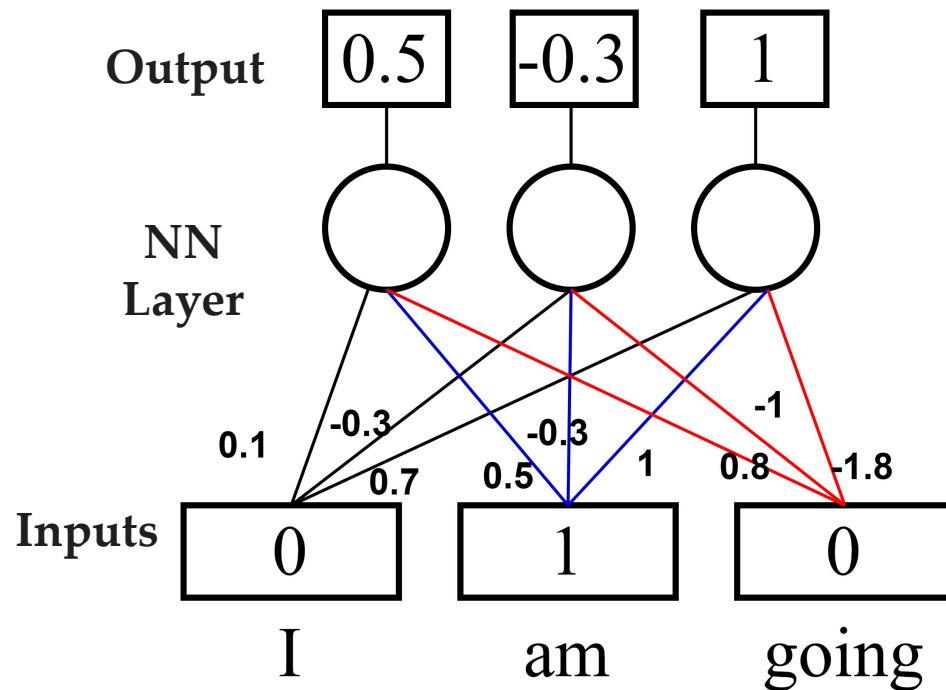
['i', 'am', 'going']

Word Embeddings (as dense vector):

'I': $\rightarrow [1, 0, 0, 0] \rightarrow [0.1, -0.3, 0.7]$

'am' $\rightarrow [0, 1, 0, 0] \rightarrow [0.5, -0.3, 1]$

'going': $\rightarrow [0, 0, 1, 0]$



Word Representation: NN approach

- Word Embeddings: word representation in vector

Parallel corpus in my dataset:

I am going → मैं जा रहा हूँ

Preprocessing: English Tokens

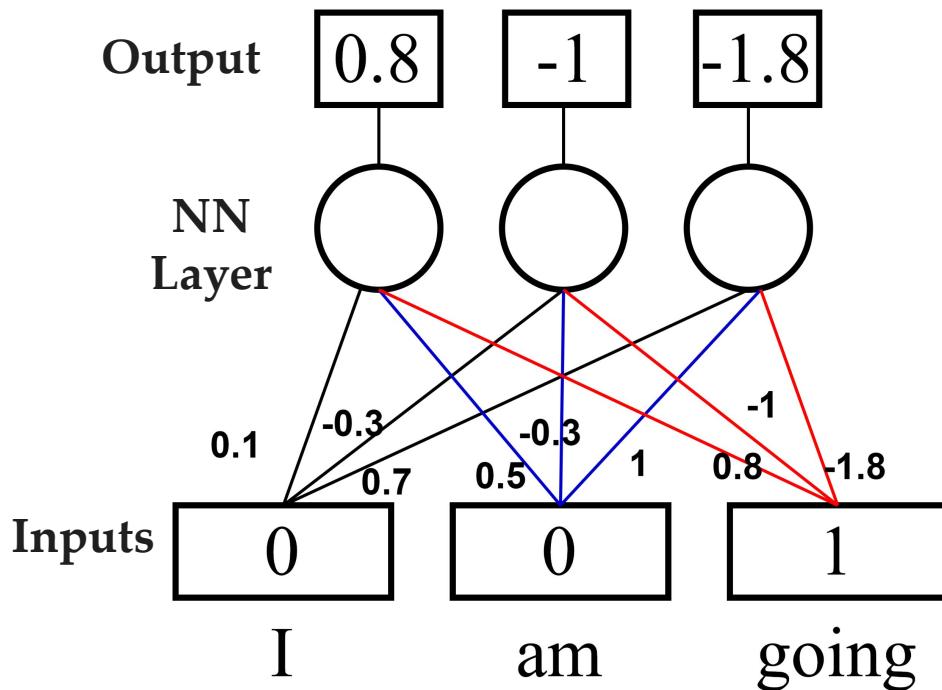
['i', 'am', 'going']

Word Embeddings (as dense vector):

'I': $\rightarrow [1, 0, 0, 0] \rightarrow [0.1, -0.3, 0.7]$

'am' $\rightarrow [0, 1, 0, 0] \rightarrow [0.5, -0.3, 1]$

'going': $\rightarrow [0, 0, 1, 0] \rightarrow [0.8, -1, -1.8]$



Word Representation: Skip-gram approach

Skip-gram predicts multiple context words based on the center word

Dataset: The pink horse is eating.

Step 1: Preparing One-hot vector

the: [1 0 0 0 0]

pink: [0 1 0 0 0]

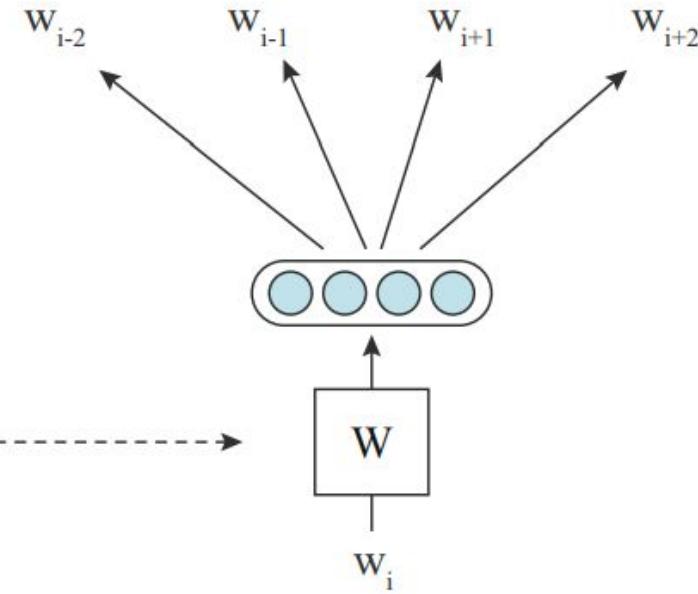
horse: [0 0 1 0 0]

is: [0 0 0 1 0]

eating: [0 0 0 0 1]

Classifier

Word Matrix



Word Representation: Skip-gram approach

Skip-gram predicts multiple context words based on the center word

Dataset: The pink horse is eating.

Step 1: Select window size =2, and make word pairs for training

Word in a sentence	pairs
The pink horse is eating	(the, pink), (the horse)
The pink horse is eating	(pink, the), (pink, horse), (pink, is)
The pink horse is eating	(horse, the), (horse, pink), (horse, is), (horse, eating)
The pink horse is eating	(is, pink), (is, horse), (is, eating)
The pink horse is eating	(eating, horse), (eating, is)

Word Representation: Skip-gram approach

Skip-gram predicts multiple context words based on the center word

Dataset: The pink horse is eating.

Step 1: Preparing One-hot vector

the: [1 0 0 0 0]

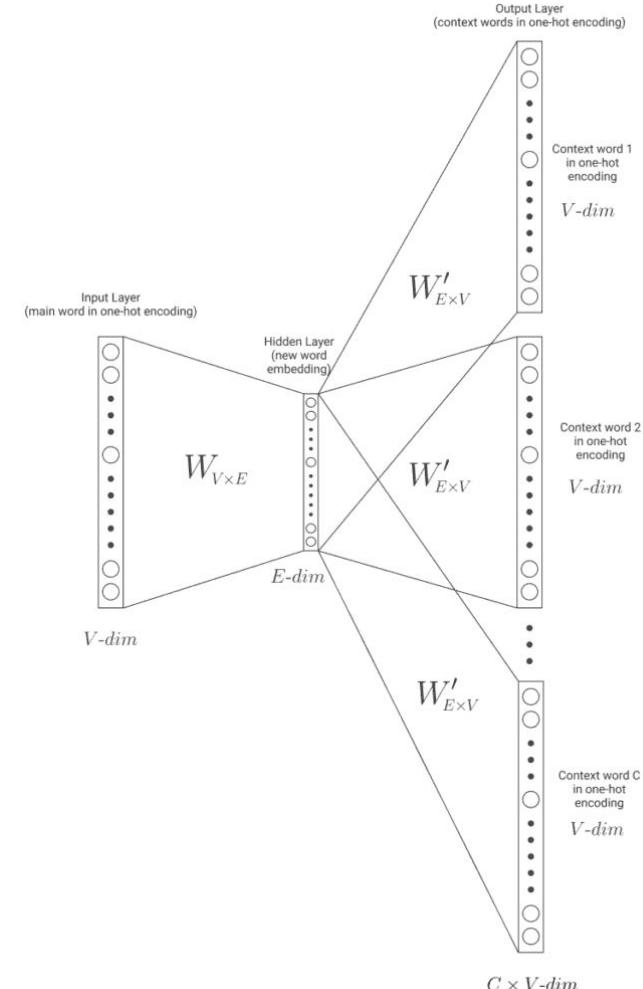
pink: [0 1 0 0 0]

horse: [0 0 1 0 0]

is: [0 0 0 1 0]

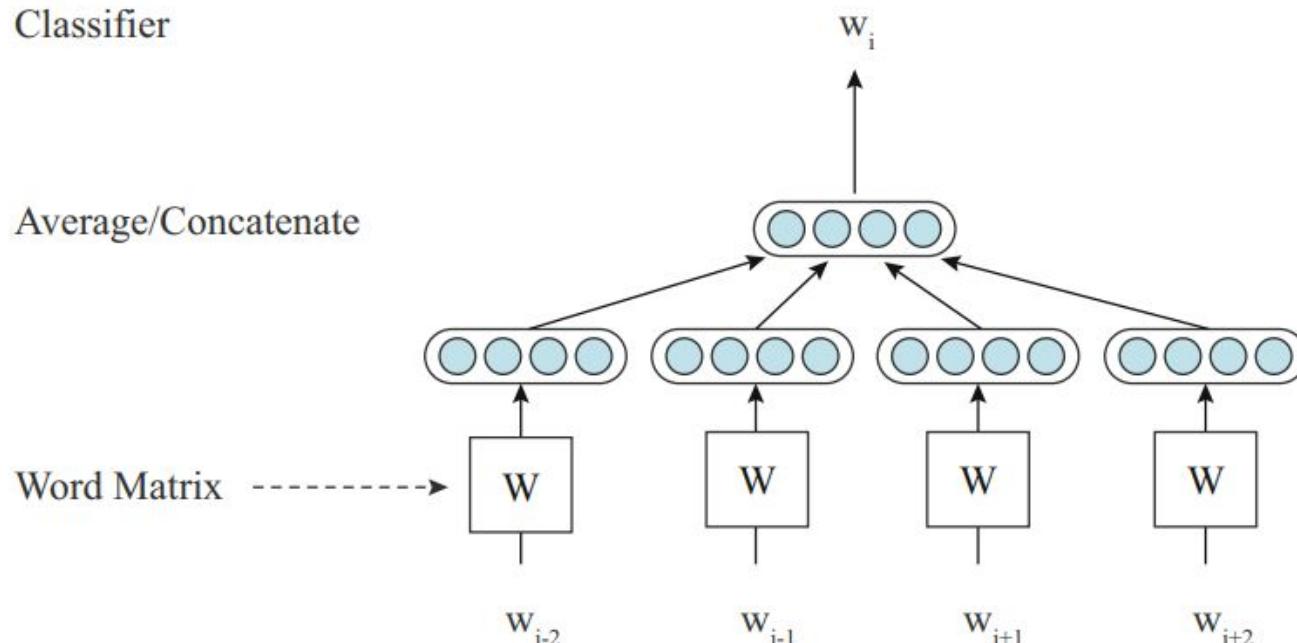
eating: [0 0 0 0 1]

pairs
(the, pink), (the horse)
(pink, the), (pink, horse), (pink, is)
(horse, the), (horse, pink), (horse, is), (horse, eating)
(is, pink), (is, horse), (is, eating)
(eating, horse), (eating, is)



Word Representation: CBOW approach

CBOW predicts a center word based on multiple context words



Word Representation: Word Embedding

- Can capture semantic meaning.
 - Similar context words have similar vectors
- Similar words have similar representations.

King: [0.2 0.9 0.9 0.9 0.1]

Queen: [0.9 0.1 1.0 0.8 0.9]

Cricketer: [0.9 0.1 0.4 0.7 1.0]

Word Representation: Word Embedding

- Can capture semantic meaning.
 - Similar context words have similar vectors
- Similar words have similar representations.

King: [0.9 0.9 0.9 0.9 0.1]

Queen: [0.9 0.1 1.0 0.8 0.9]

Cricketer: [0.2 0.1 0.4 0.7 1.0]

Royalty

Word Representation: Word Embedding

- Can capture semantic meaning.
 - Similar context words have similar vectors
- Similar words have similar representations.

King: [0.9 0.1 0.9 0.9 0.1]

Queen: [0.9 0.1 1.0 0.8 0.9]

Cricketer: [0.2 0.9 0.4 0.7 1.0]

Athletics

Word Representation: Word Embedding

- Can capture semantic meaning.
 - Similar context words have similar vectors
- Similar words have similar representations.

King: [0.9 0.1 0.9 **0.9** 0.1]

Queen: [0.9 0.1 1.0 **0.8** 0.9]

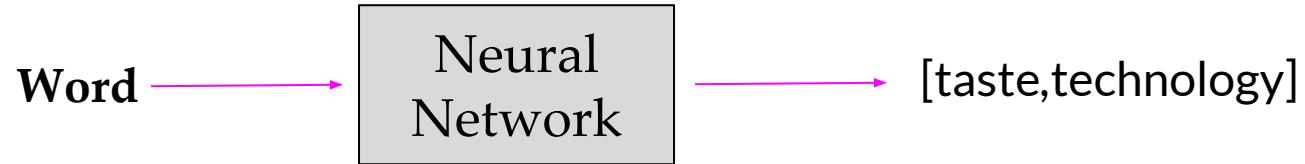
Cricketer: [0.2 0.9 0.4 **0.7** 1.0]

↑
Humanity

Word Embedding Limitations: Average Meaning

- Consider we have a dataset as:
 - An apple a day keeps the doctor away
 - Apple is healthy
 - Apple is better than Orange
 - Apple makes great phones

So on.

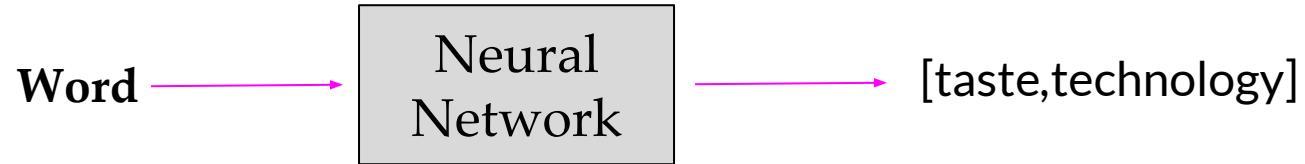


- Every word will be represented as 2-dimensional vector, where
 - 1st dimension represents taste, and
 - 2nd dimension represents technology

Word Embedding Limitations: Average Meaning

- Consider we have a dataset as:
 - An apple a day keeps the doctor away
 - Apple is healthy
 - Apple is better than Orange
 - Apple makes great phones

So on.

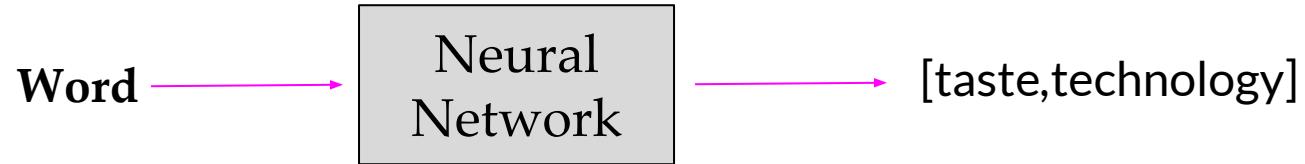


- Sentence1: [0.6,0]

Word Embedding Limitations: Average Meaning

- Consider we have a dataset as:
 - An apple a day keeps the doctor away
 - Apple is healthy
 - Apple is better than Orange
 - Apple makes great phones

So on.

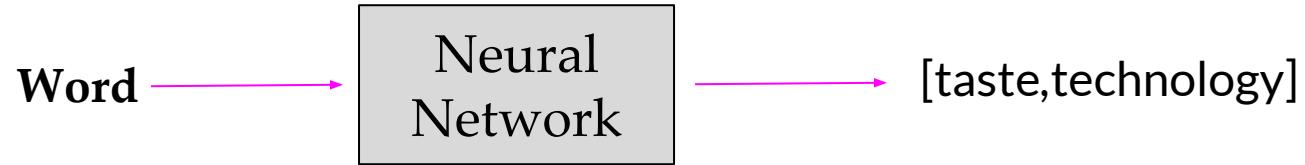


- Sentence1: [0.6,0]
- Sentence2: [0.7,0]

Word Embedding Limitations: Average Meaning

- Consider we have a dataset as:
 - An apple a day keeps the doctor away
 - Apple is healthy
 - Apple is better than Orange
 - Apple makes great phones

So on.

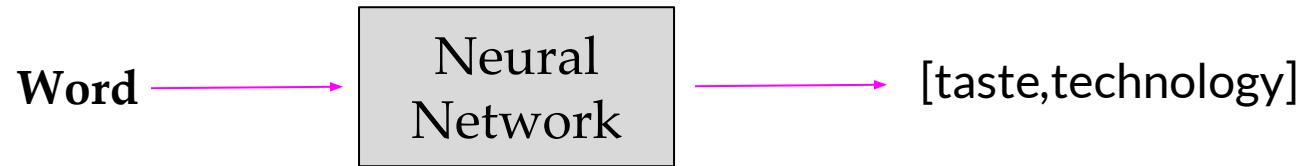


- Sentence1: [0.6,0]
- Sentence2: [0.7,0]
- Sentence3: [0.8,0]

Word Embedding Limitations: Average Meaning

- Consider we have a dataset as:
 - An apple a day keeps the doctor away
 - Apple is healthy
 - Apple is better than Orange
 - Apple makes great phones

So on.

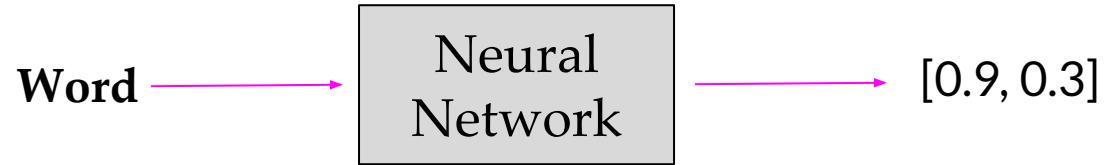


- Sentence1: [0.6,0]
- Sentence2: [0.7,0]
- Sentence3: [0.8,0]
- Sentence4: [0.8,0.2]

Word Embedding Limitations: Average Meaning

- Consider we have a dataset as:
 - An apple a day keeps the doctor away
 - Apple is healthy
 - Apple is better than Orange
 - Apple makes great phones

So on.



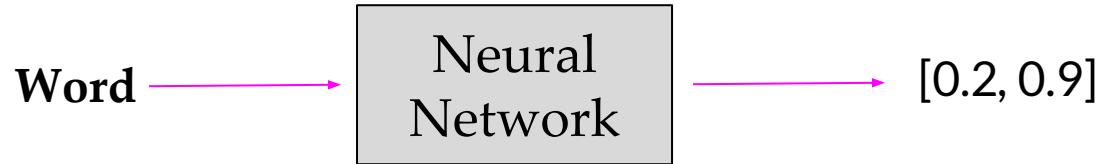
- Consider we have 10,000 sentences
 - In 9000 sentences, apple context is about taste
 - In 1000 sentences, apple context is about technology

Final **Average** Embedding vector of word **Apple** = [0.9, 0.3]

Word Embedding Limitations: Average Meaning

- But if we flip the instances of taste and technology and we have:
 - In 9000 sentences, apple context is about technology
 - In 1000 sentences, apple context is about taste

Final **Average** Embedding vector of word **Apple** = [0.2, 0.9]



Word Embedding vector depends on the Dataset.

- Because **Word Embedding** represents the **average meaning** (and not the actual meaning) of a word in the given dataset.

Word Embedding Limitations: Average Meaning

- Word Embeddings are created once and being used again and again for all datasets.
- Hence, these word embeddings are **static** in nature.

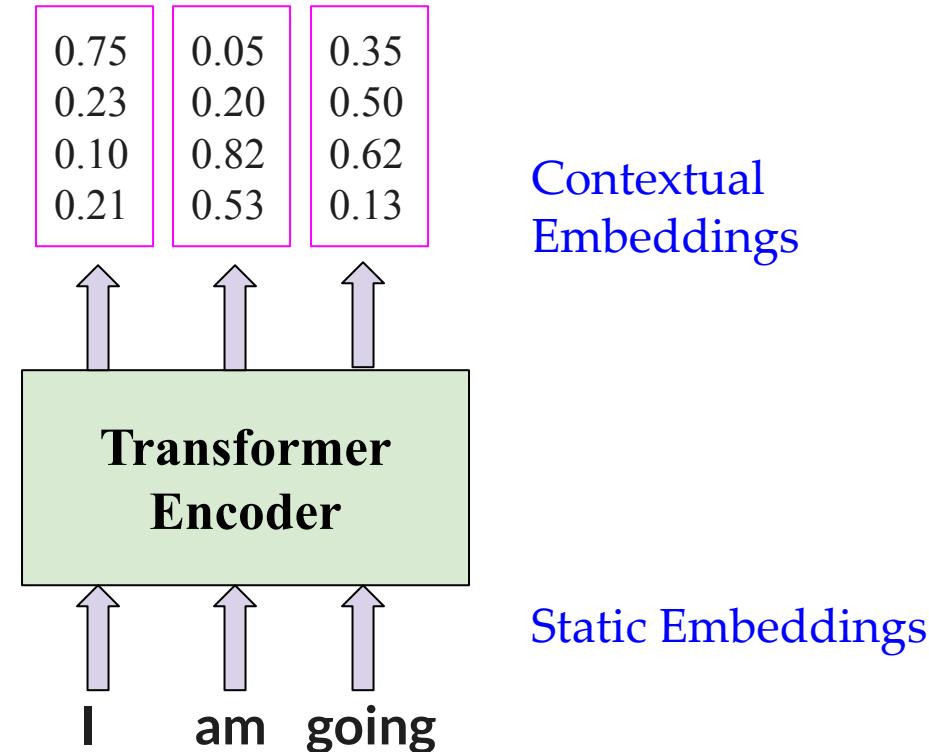
Use Case: Machine translation

Consider we need to convert a sentence in Hindi:

Apple launched a new phone while I was eating an Orange.

- Here for word Apple the embedding vector from previous slide: [0.9 0.2] which is wrong.
- Solution to this problem is **Self Attention**.
- Self attention converts **Static Embeddings** into **Contextual Embeddings..**

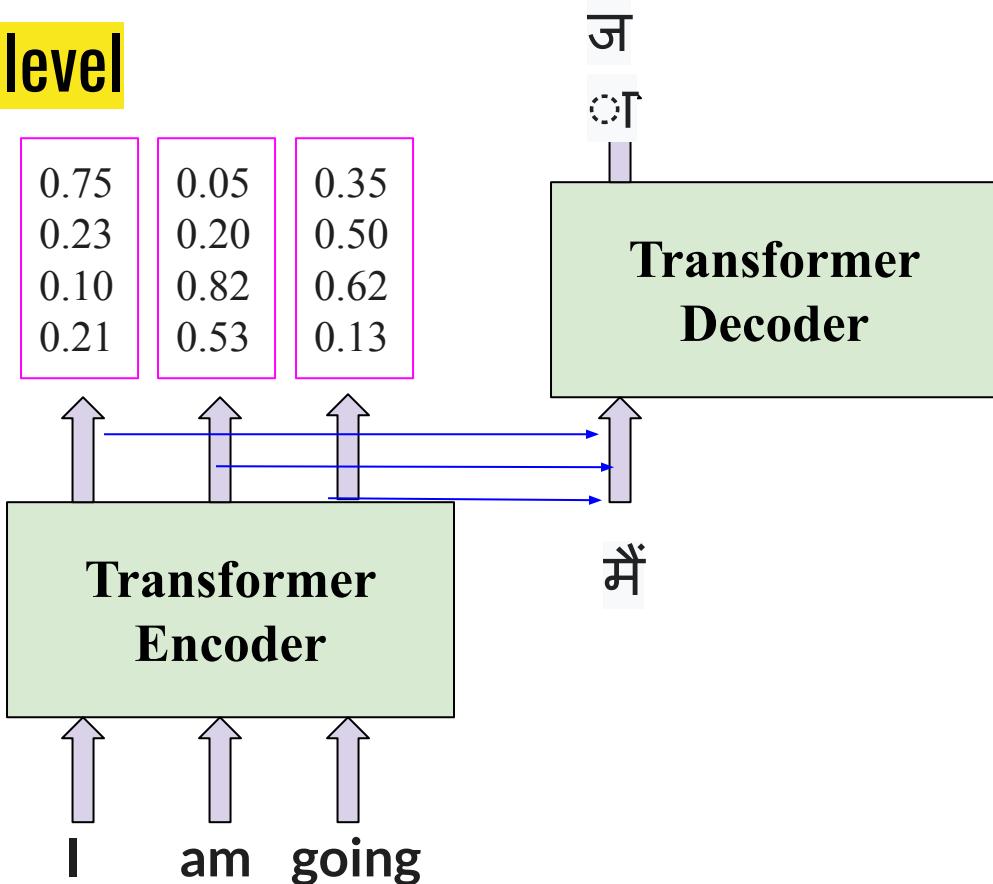
Transformer Self Attention



Transformer working: High level

English: I am going

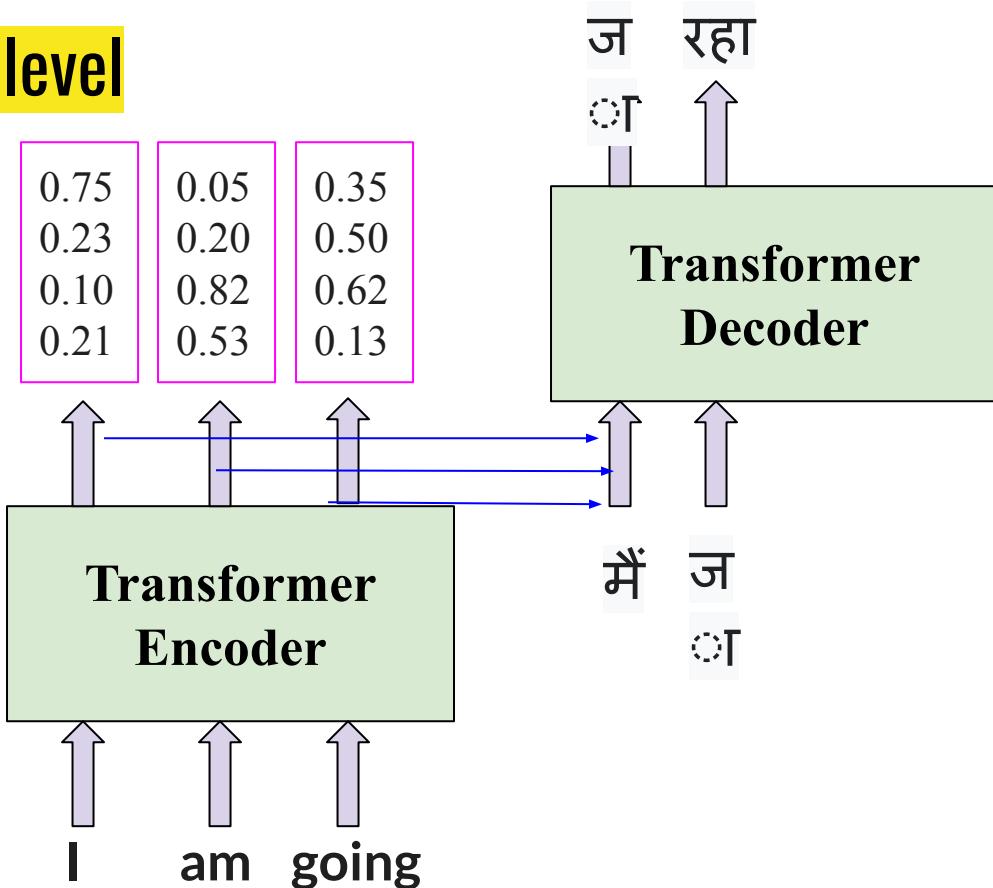
Hindi: मैं जा रहा हूँ



Transformer working: High level

English: I am going

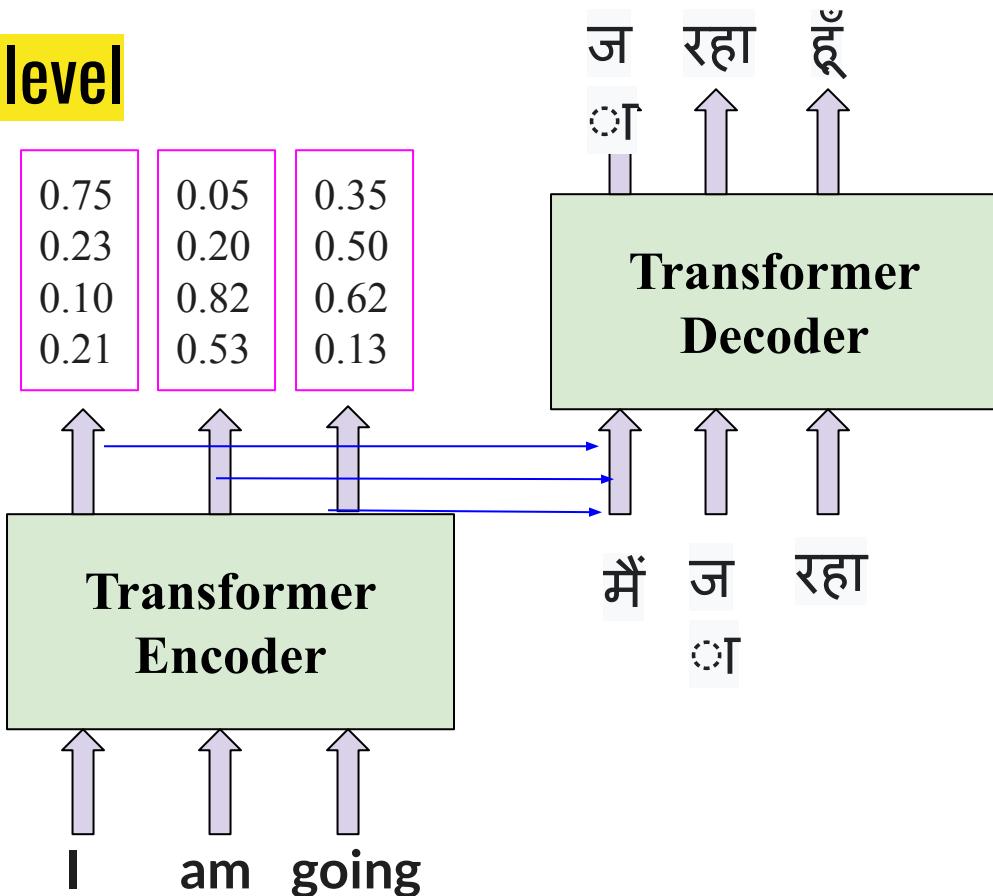
Hindi: मैं जा रहा हूँ



Transformer working: High level

English: I am going

Hindi: मैं जा रहा हूँ



Word Embedding: recap

- It is important to represent word as embeddings/vectors for any NLP task to be performed by Machine.
- Word embedding capture the semantic of a word.
- Two types of word embeddings:
 - **Frequency based:** OHE, BoW
 - **Prediction based:** Word2Vec, GloVe
- Limitation with word embedding:
 - **Static** in nature
 - Embedding of word bank is same in both the context below

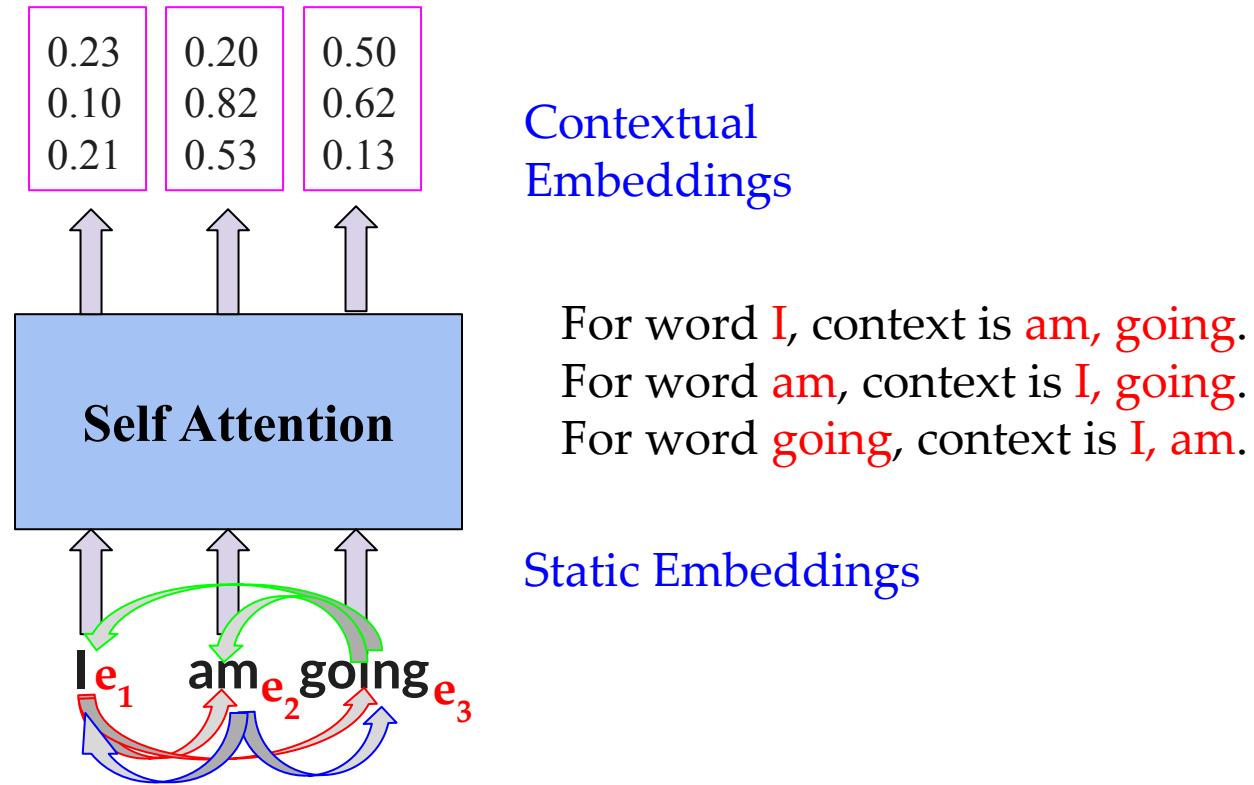
Money **bank**

[0.6, 0.2, 0.2, 0.7]

River **bank**

[0.6, 0.2, 0.2, 0.7]

Word Embedding: recap



Self Attention: How does it work?

S1: Money bank grows

Bank meaning is derived with the context of Money

S2: River bank flows

Bank meaning is derived with the context of River

- Static Embedding (like Word2Vec, GloVe, FastText) will give a similar vector for both bank.
- That will make difficult for the model to understand the context
- Therefore, we will use Transformer to create dynamic embedding for bank.

Self Attention: How does it work?

S1: Money bank grows

Let's represent the word bank as:

Contextual Embedding

bank: 0.25money+0.7bank+0.05grows

Self Attention: How does it work?

S1: Money bank grows

Let's represents the word bank as:

Contextual Embedding

bank: 0.25money+0.7bank+0.05grows

S2: River bank flows

Let's represents the word bank as:

bank: 0.2river+0.78bank+0.02flows

Self Attention: How does it work?

S1: Money bank grows

Let's represents the words as:

money: 0.7money+0.2bank+0.1grows

bank: 0.25money+0.7bank+0.05grows

grows: 0.1money+0.2bank+0.7grows

S2: River bank flows

Let's represents the words as:

river: 0.8river+0.15bank+0.05flows

bank: 0.2river+0.78bank+0.02flows

flows: 0.4river+0.01bank+0.59flows

Bank embedding is different for both sentence.

Self Attention: How does it work?

S1: Money bank grows

Let's represents the words as embeddings (e):

$$e_{\text{money}}: 0.7e_{\text{money}} + 0.2e_{\text{bank}} + 0.1e_{\text{grows}}$$

$$e_{\text{bank}}: 0.25e_{\text{money}} + 0.7e_{\text{bank}} + 0.05e_{\text{grows}}$$

$$e_{\text{grows}}: 0.1e_{\text{money}} + 0.2e_{\text{bank}} + 0.7e_{\text{grows}}$$

The numbers preceding the embeddings in RHS represents similarity

In the following equation:

$$e_{\text{bank}}: 0.25e_{\text{money}} + 0.7e_{\text{bank}} + 0.05e_{\text{grows}}$$

0.25: similarity of e_{bank} and e_{money}

0.7: similarity of e_{bank} and e_{bank}

0.05: similarity of e_{bank} and e_{grows}

Self Attention: How does it work?

In the equation below:

$$e_{\text{money}} : 0.7e_{\text{money}} + 0.2e_{\text{bank}} + 0.1e_{\text{grows}}$$

The **numbers** preceding the embeddings in RHS represents similarity

0.7:- similarity between e_{money} and e_{money}

0.2:- similarity between e_{money} and e_{bank}

0.1:- similarity between e_{money} and e_{grows}

Self Attention: How does it work?

S1: Money bank grows

Let's represents the words as embeddings (e):

$$e_{\text{money}}: 0.7e_{\text{money}} + 0.2e_{\text{bank}} + 0.1e_{\text{grows}}$$

$$e_{\text{bank}}: 0.25e_{\text{money}} + 0.7e_{\text{bank}} + 0.05e_{\text{grows}}$$

$$e_{\text{grows}}: 0.1e_{\text{money}} + 0.2e_{\text{bank}} + 0.7e_{\text{grows}}$$

The numbers preceding the embeddings in RHS represents similarity

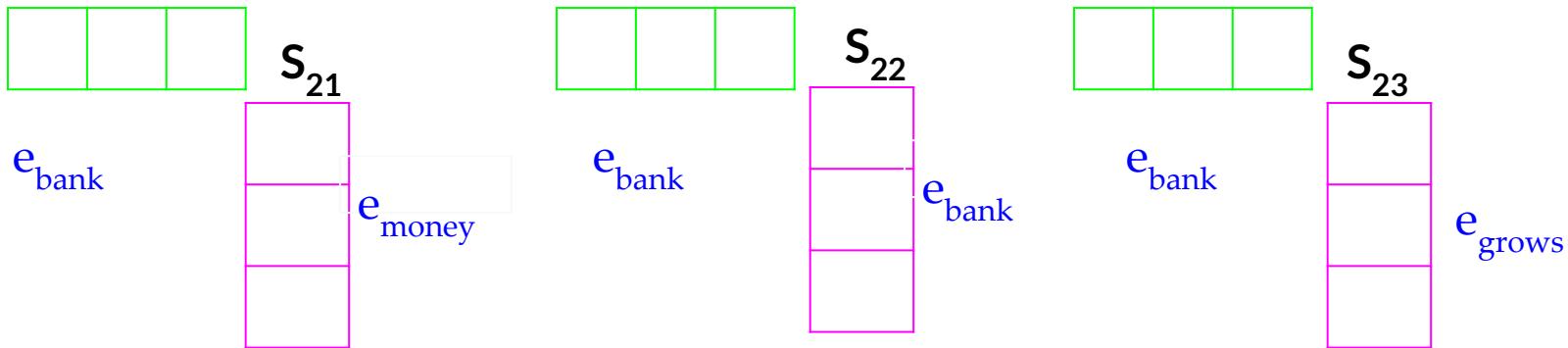
In order to get the similarity, the same equation (in RHS) can be written as:

$$\begin{aligned} e_{\text{bank}}^{\text{new:}} & [e_{\text{bank}} \cdot e_{\text{money}}] e_{\text{money}} + \\ & [e_{\text{bank}} \cdot e_{\text{bank}}] e_{\text{bank}} + \\ & [e_{\text{bank}} \cdot e_{\text{grows}}] e_{\text{grows}} + \end{aligned}$$

Self Attention: How does it work?

In order to get the similarity, the same equation (in right) can be written as:

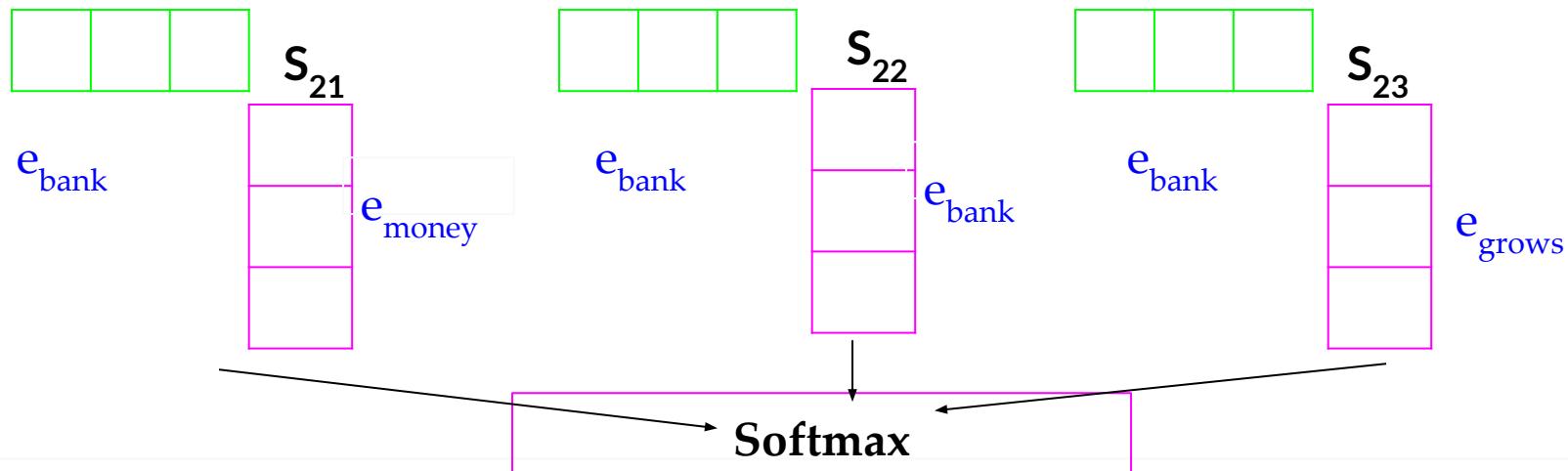
$$e_{\text{bank}}^{\text{new}}: [e_{\text{bank}} \cdot e_{\text{money}}^T]e_{\text{money}} + [e_{\text{bank}} \cdot e_{\text{bank}}^T]e_{\text{bank}} + [e_{\text{bank}} \cdot e_{\text{grows}}^T]e_{\text{grows}}$$



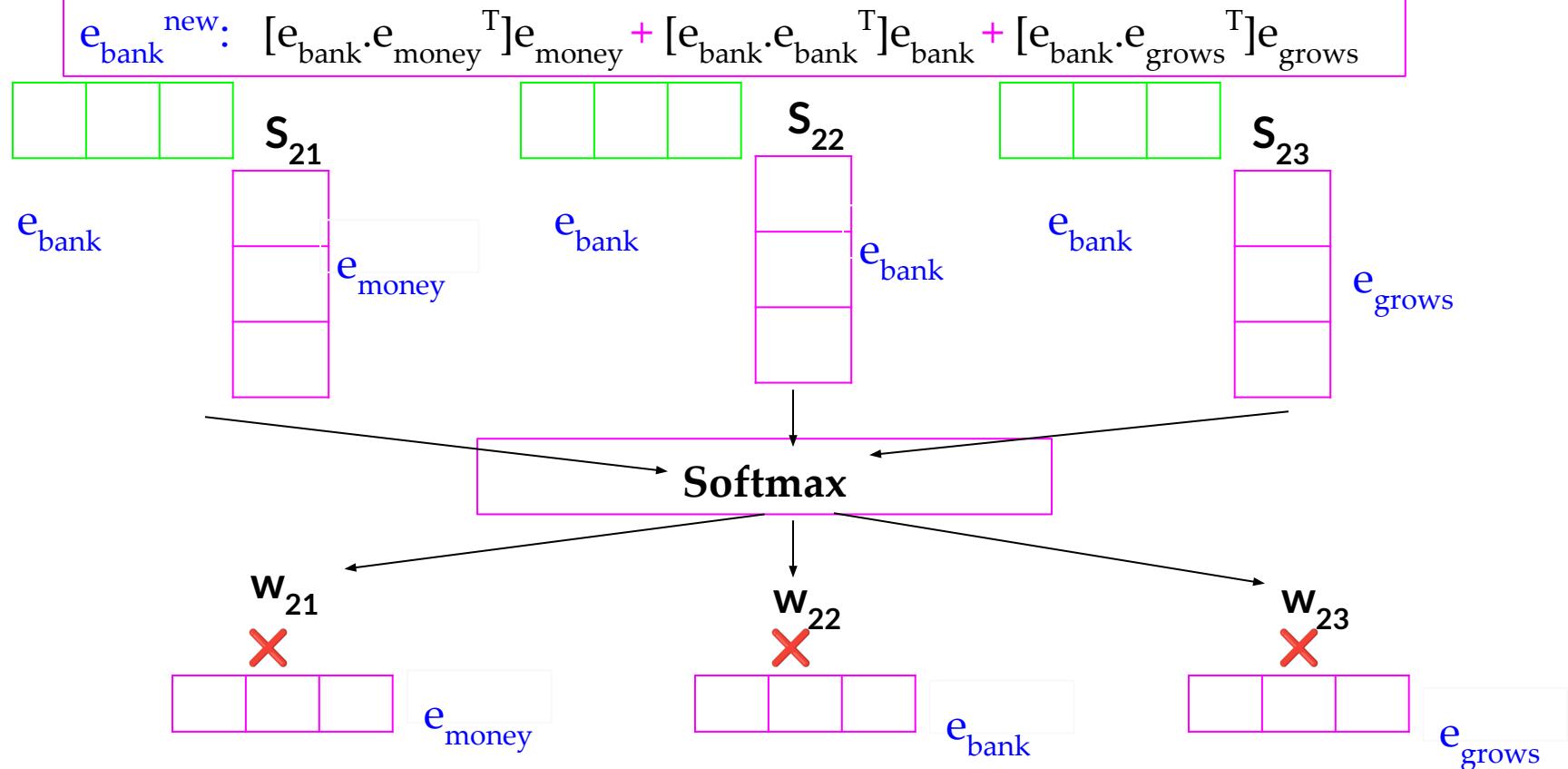
Self Attention: How does it work?

In order to get the similarity, the same equation (in right) can be written as:

$$e_{\text{bank}}^{\text{new}}: [e_{\text{bank}} \cdot e_{\text{money}}^T]e_{\text{money}} + [e_{\text{bank}} \cdot e_{\text{bank}}^T]e_{\text{bank}} + [e_{\text{bank}} \cdot e_{\text{grows}}^T]e_{\text{grows}}$$

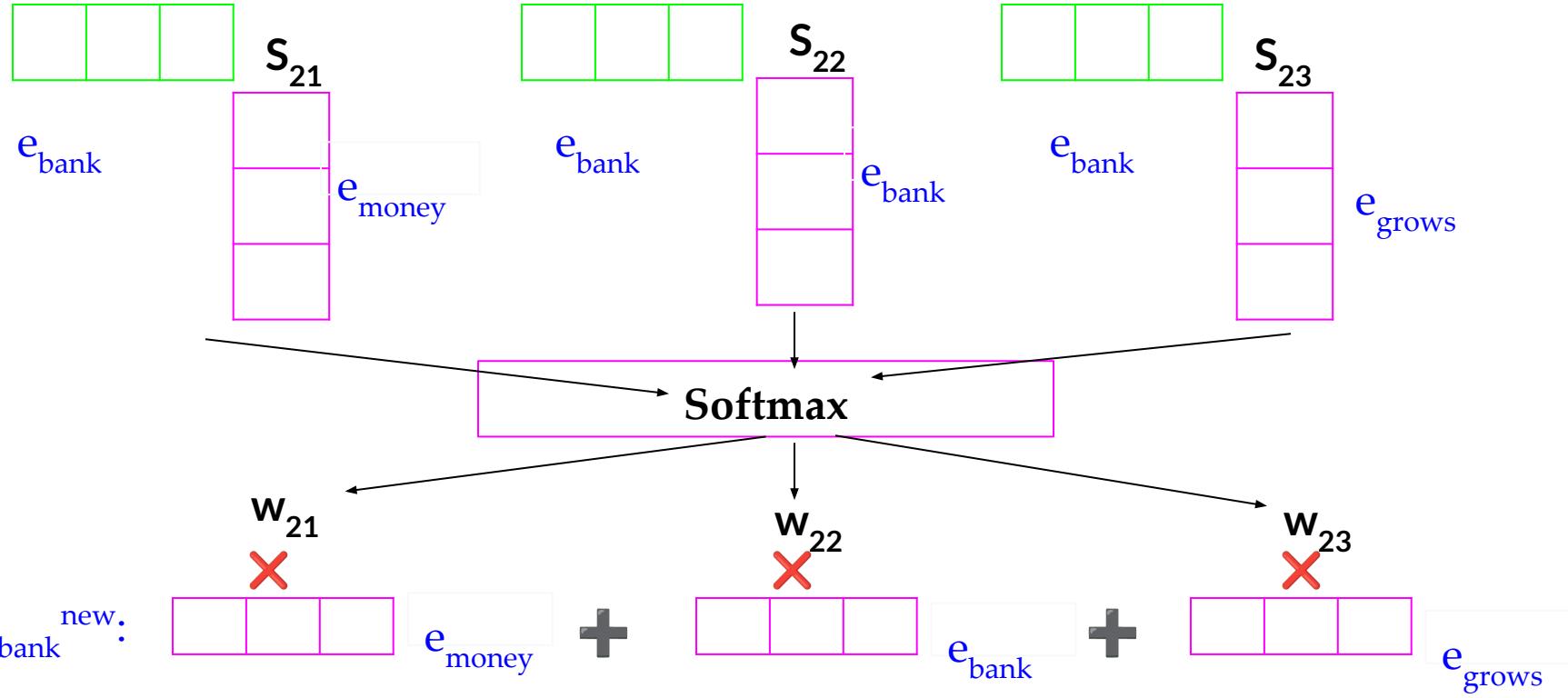


Self Attention: How does it work?



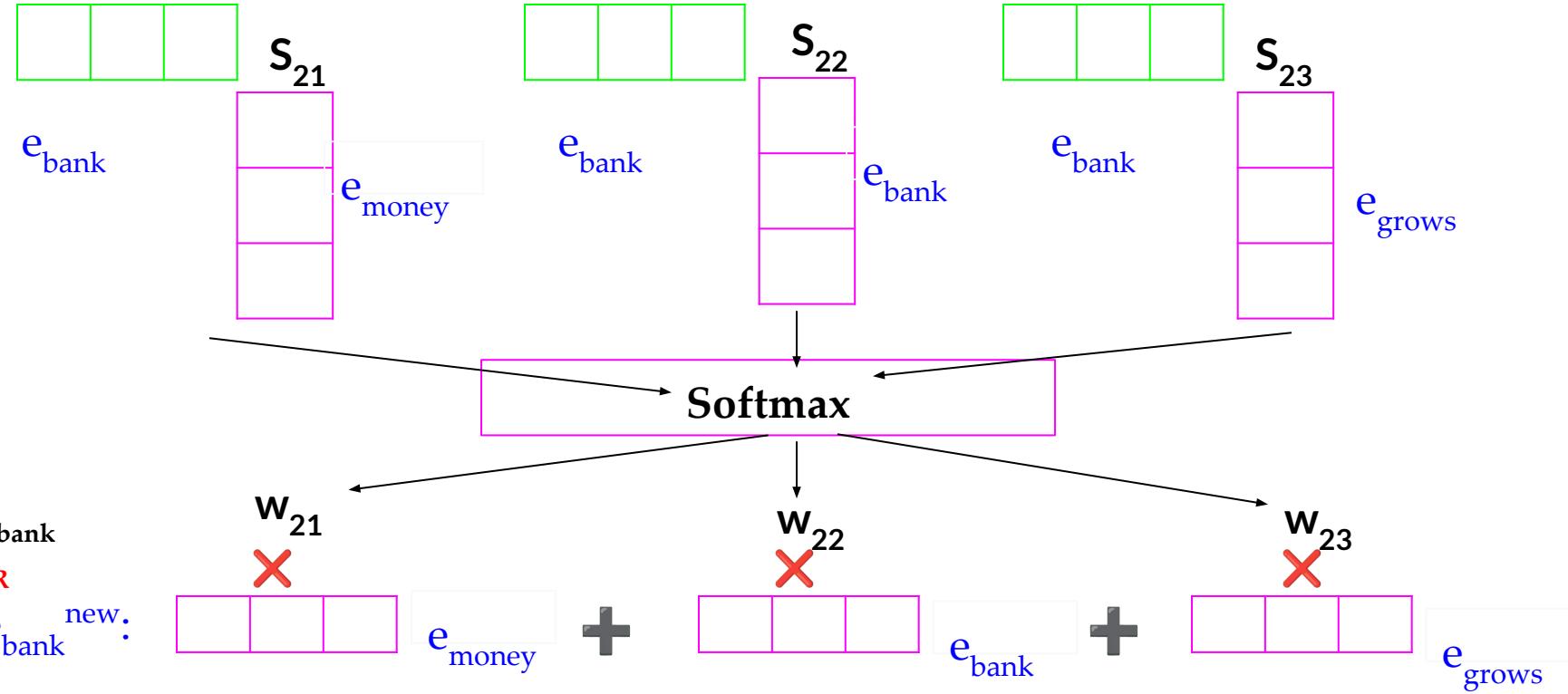
Self Attention: How does it work?

$$e_{\text{bank}}^{\text{new}}: [e_{\text{bank}} \cdot e_{\text{money}}^T] e_{\text{money}} + [e_{\text{bank}} \cdot e_{\text{bank}}^T] e_{\text{bank}} + [e_{\text{bank}} \cdot e_{\text{grows}}^T] e_{\text{grows}}$$

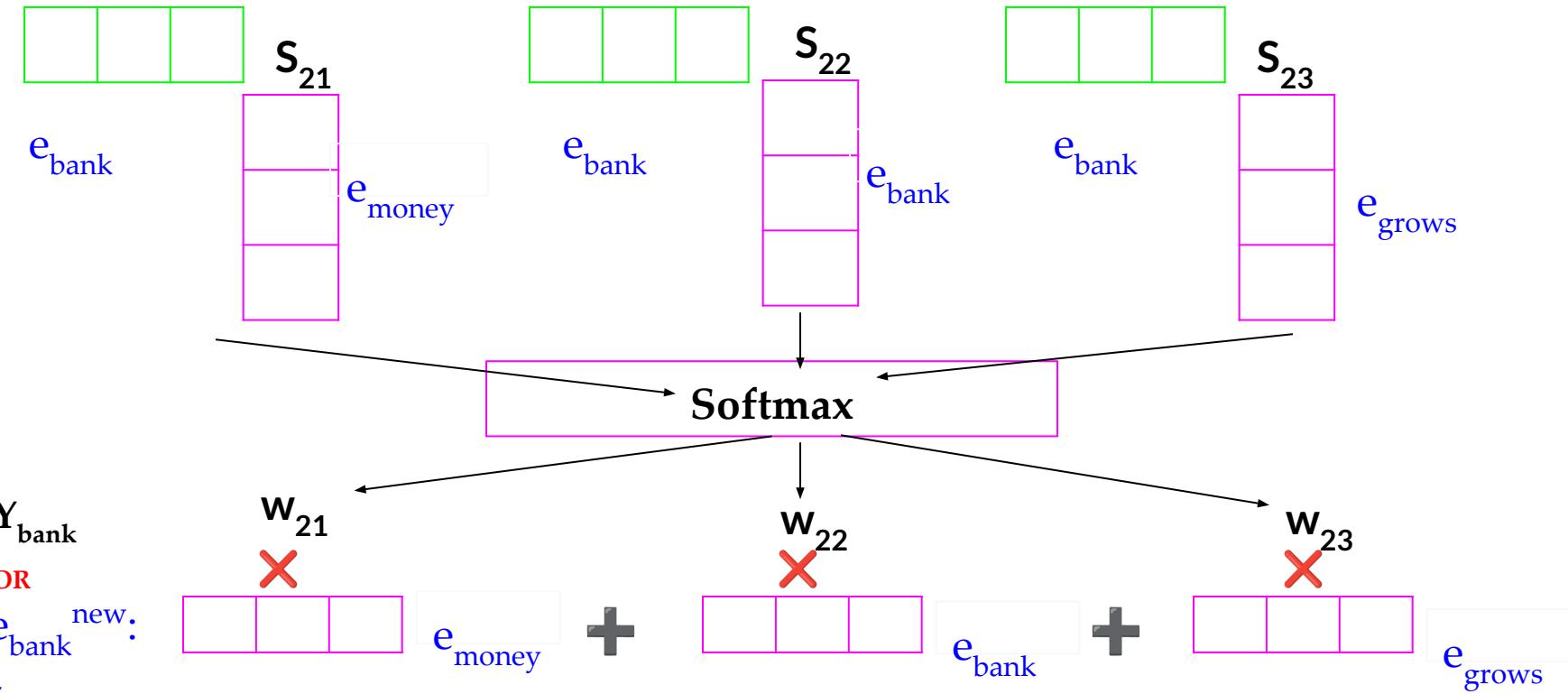


Self Attention: How does it work?

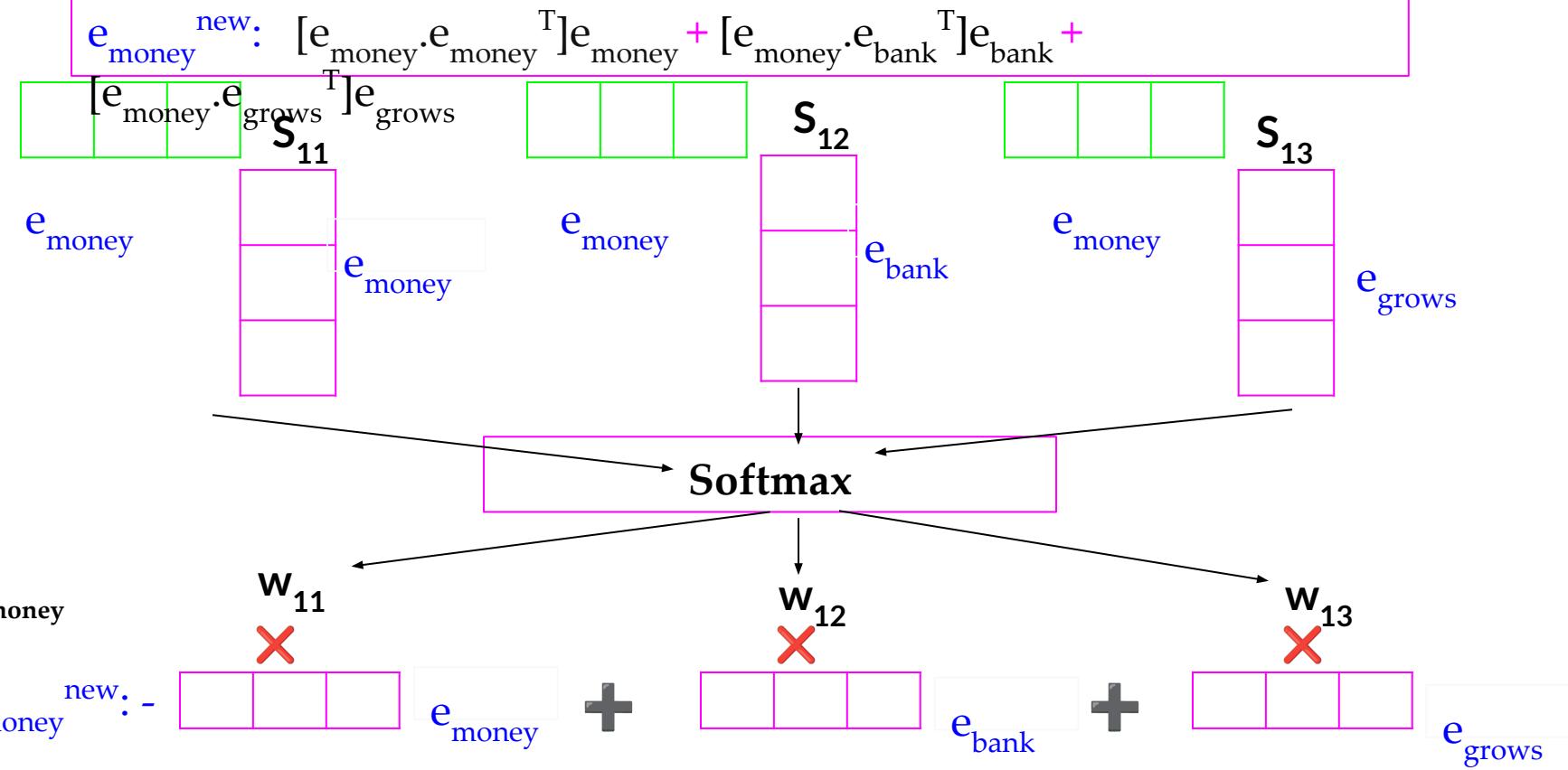
$$e_{\text{bank}}^{\text{new}}: [e_{\text{bank}} \cdot e_{\text{money}}^T] e_{\text{money}} + [e_{\text{bank}} \cdot e_{\text{bank}}^T] e_{\text{bank}} + [e_{\text{bank}} \cdot e_{\text{grows}}^T] e_{\text{grows}}$$



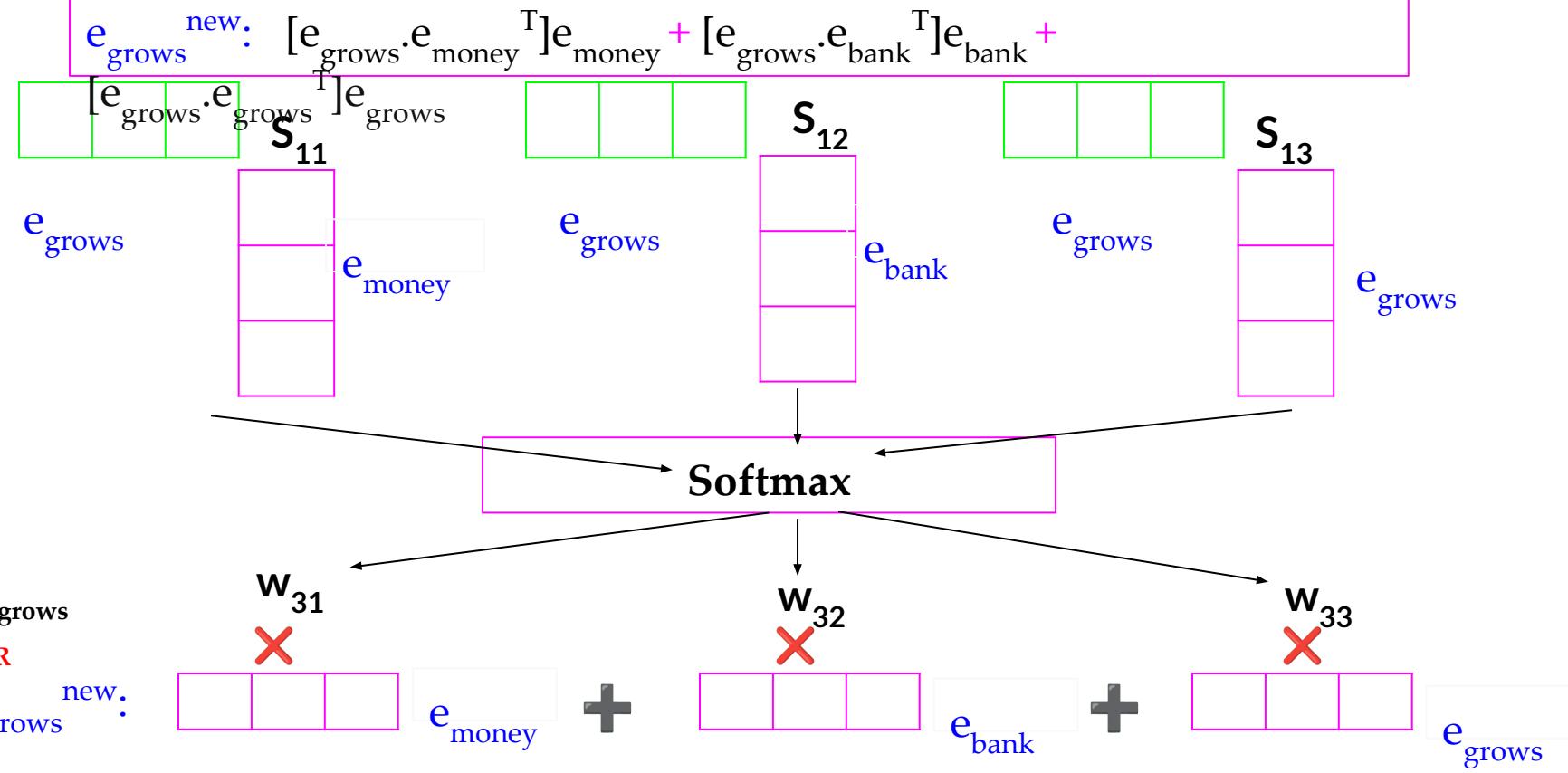
Self Attention: How does it work?



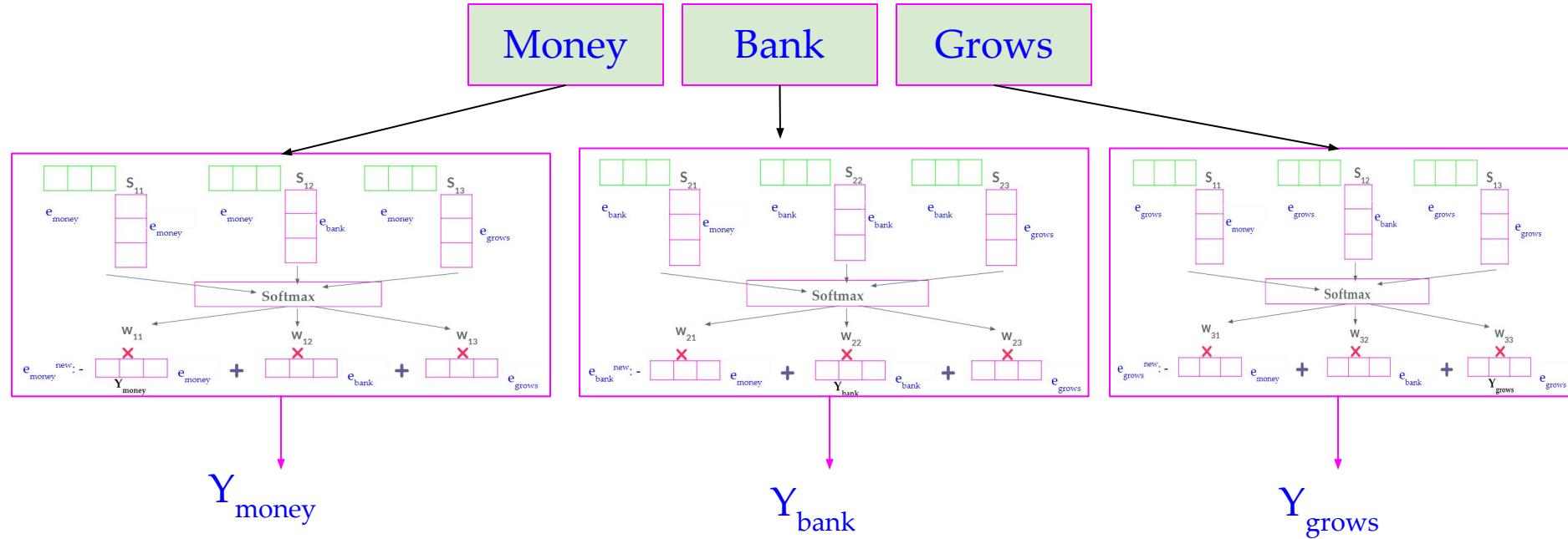
Self Attention: How does it work?



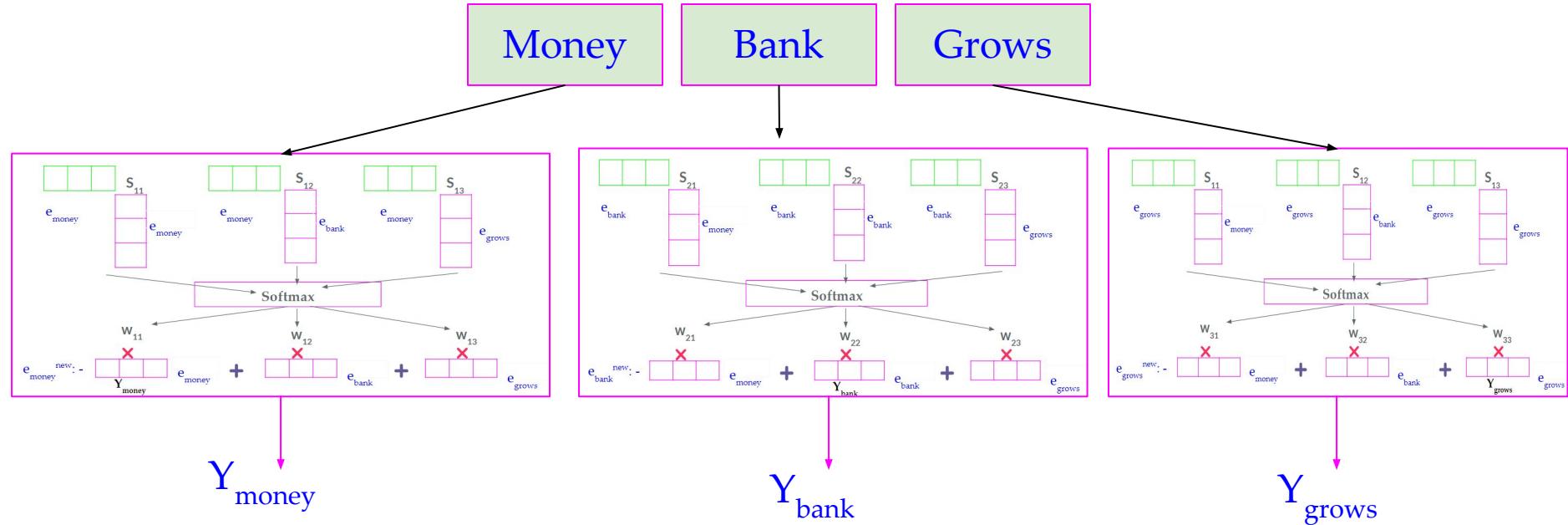
Self Attention: How does it work?



Self Attention: How does it work?



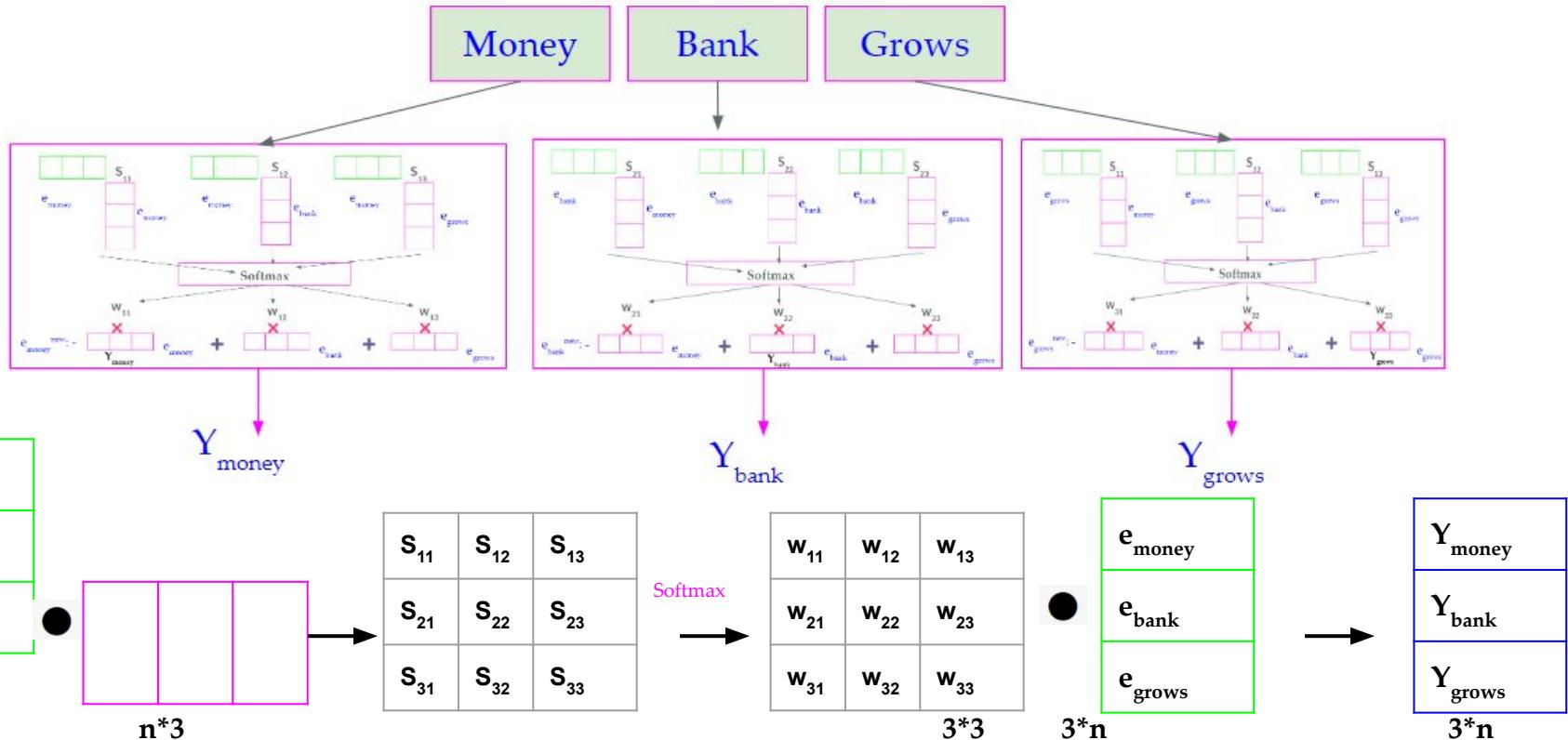
Self Attention: How does it work?



Advantage: Parallel Computing

Disadvantages: No parameters involved

Self Attention: Parallel operations

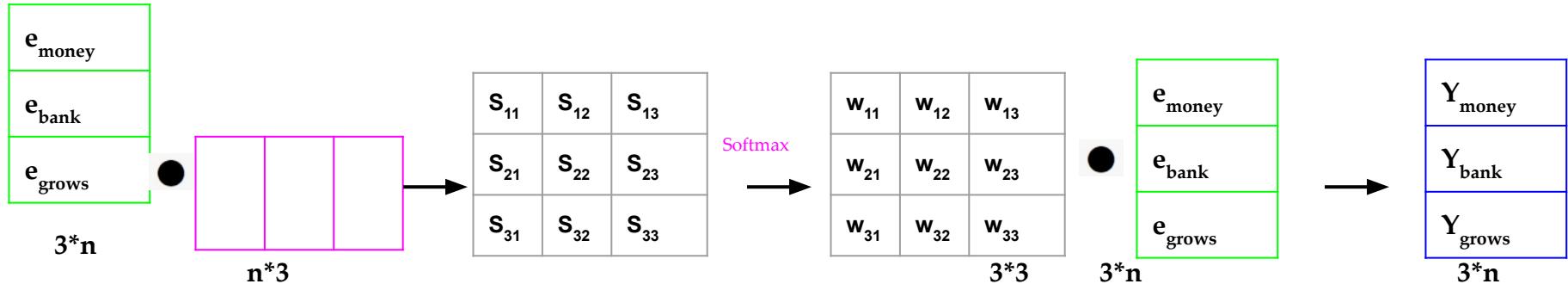


Self Attention: Naive approach

Advantage: Parallel Computing

Disadvantages: No parameters involved

No sequential information

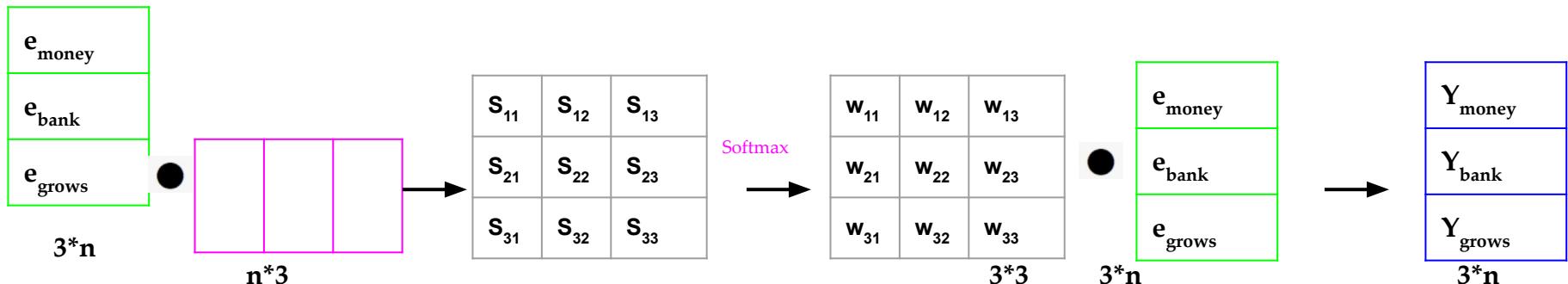


Self Attention: Naive approach Limitation

Dataset:

I am going → मैं जा रहा हूँ
Think → सोचना

- Self attention model we have just developed gives a **general contextual embedding**.
- It does not give a **task specific embedding**.

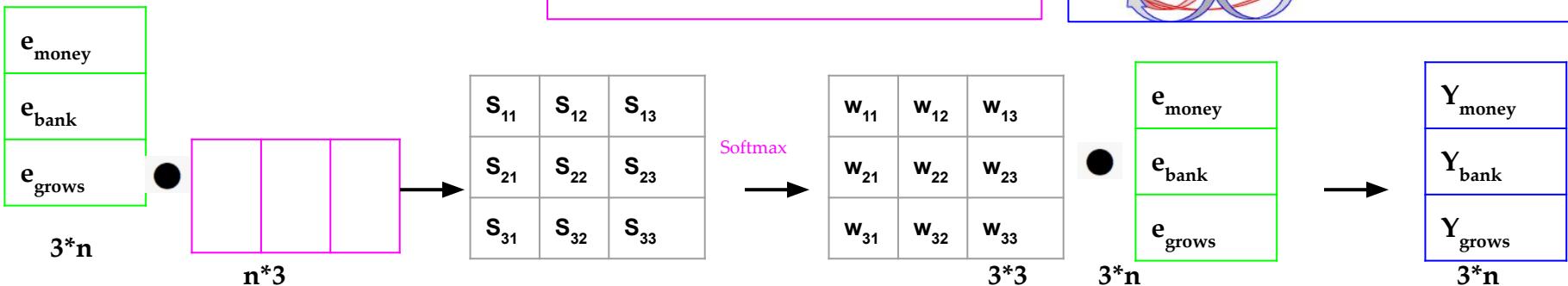
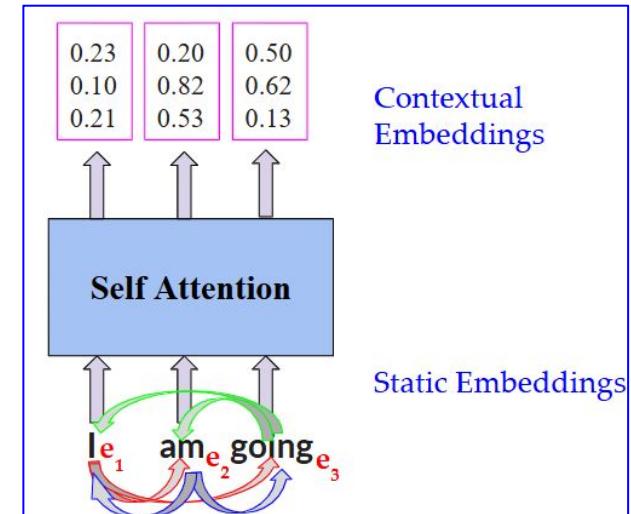


Self Attention: Naive approach Limitation

Dataset:

I am going → मैं जा रहा हूँ
Think → सोचना

- This embedding is a **general contextual embedding**.
- It does not depend on **corpus**.
- Therefore it is not giving **task specific embedding**.



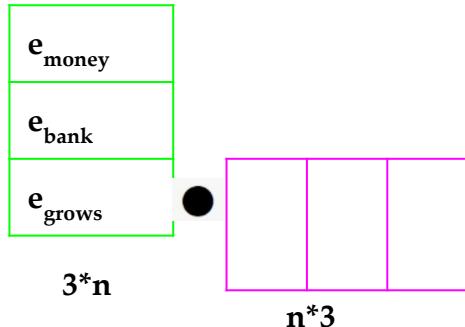
Self Attention: Naive approach Limitation

How it matters?

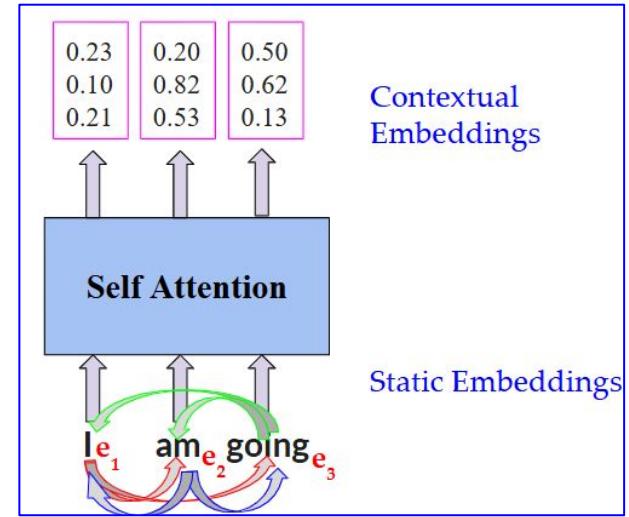
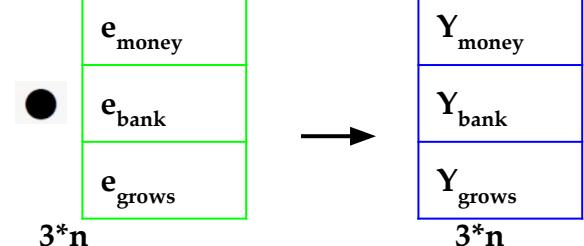
Ex: Suppose we have a sentence in our corpus:

Piece of cake → बहुत आसान काम

Piece of cake →



Softmax



Contextual
Embeddings

Static Embeddings

Self Attention: Naive approach Limitation

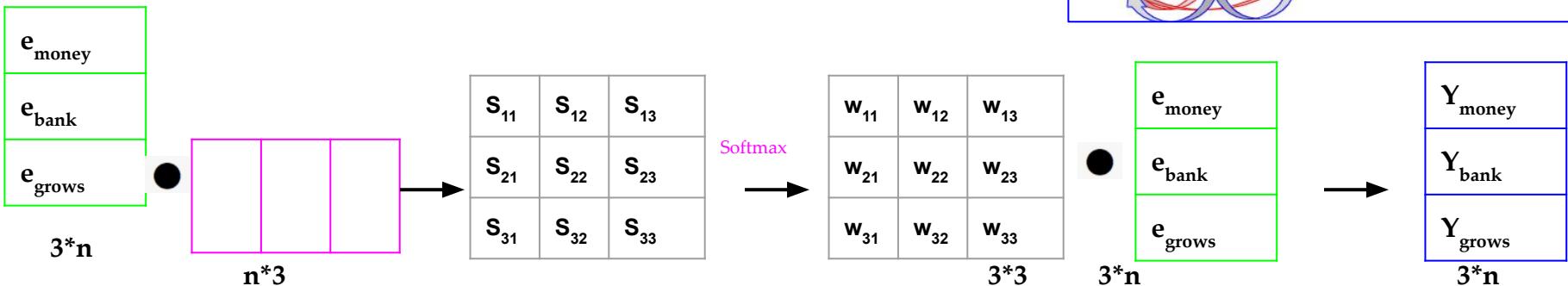
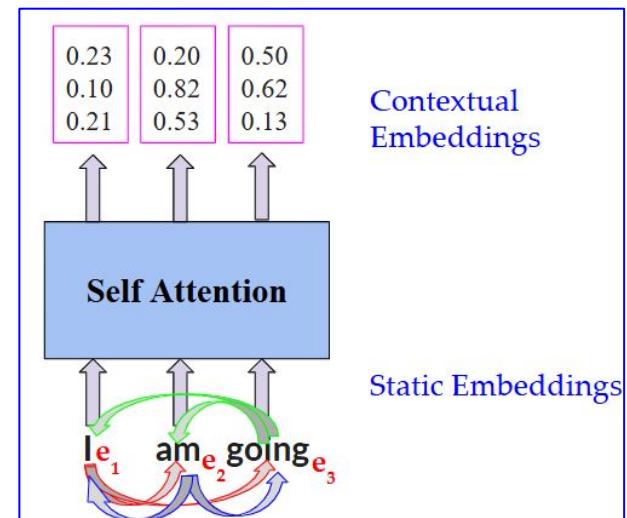
How it matters?

Ex: Suppose we have a sentence in our corpus:

Piece of cake → बहुत आसान काम

Piece of cake → General Contextual Embedding → केक का टुकड़ा

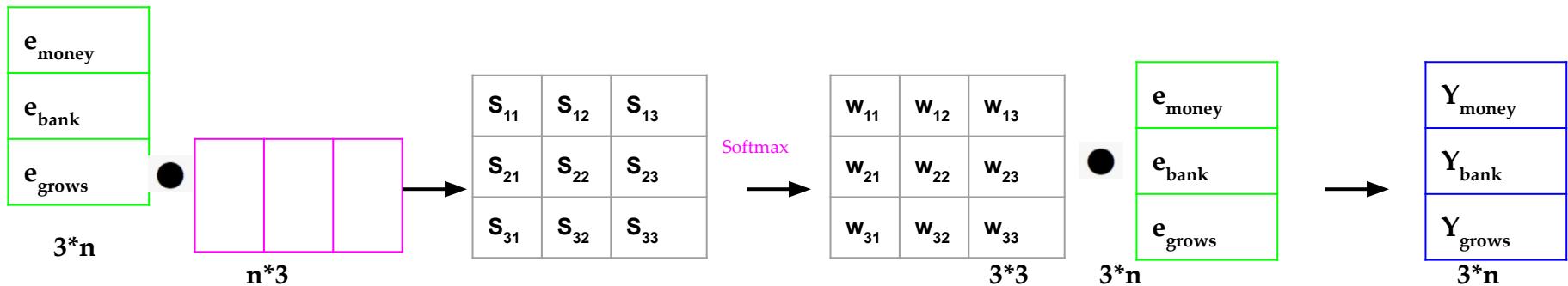
Piece of cake → Task specific Contextual Embedding → बहुत आसान काम



Self Attention: Naive approach Limitation

Solution: →

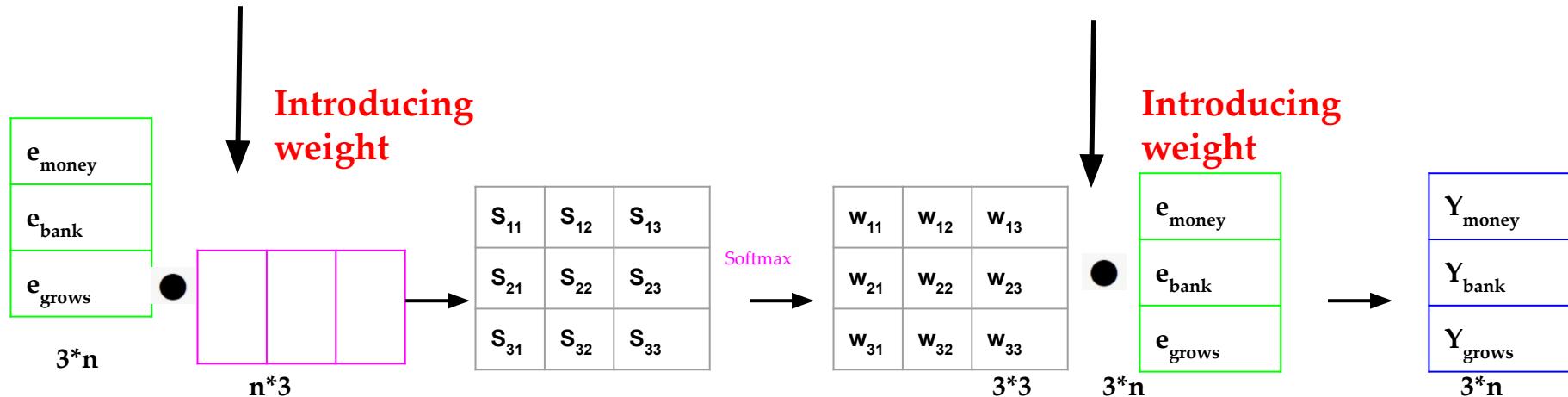
Introducing weight in this naive self attention approach



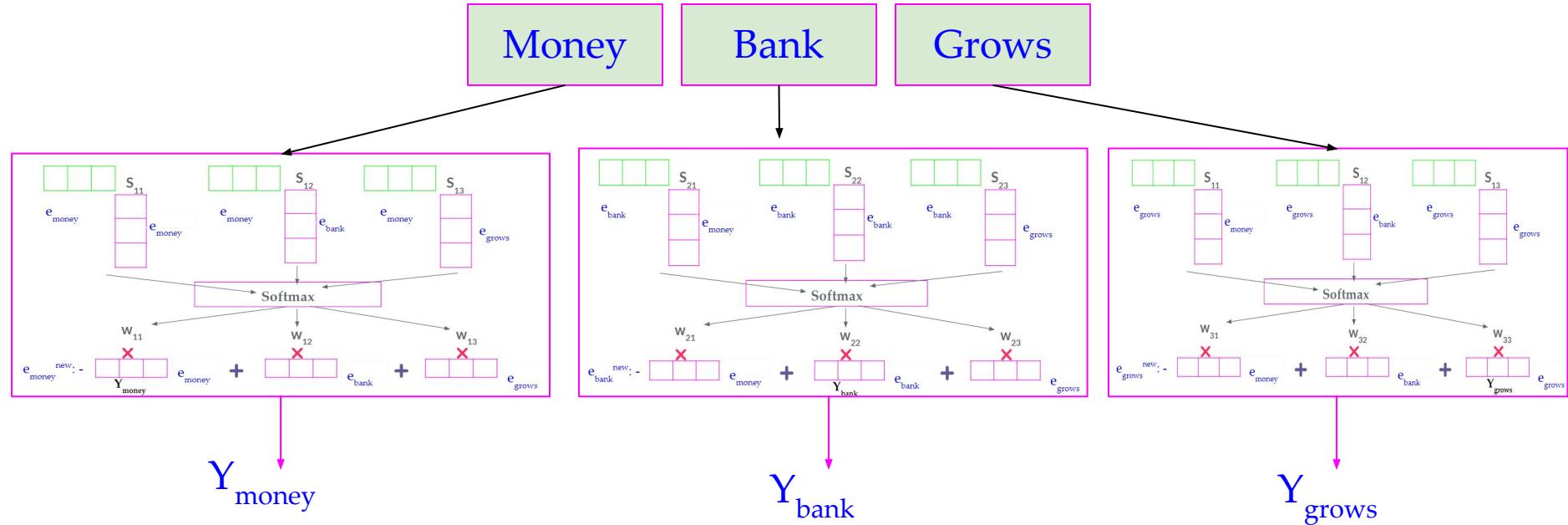
Self Attention: introducing weight

Solution: →

Introducing weight in this naive self attention approach

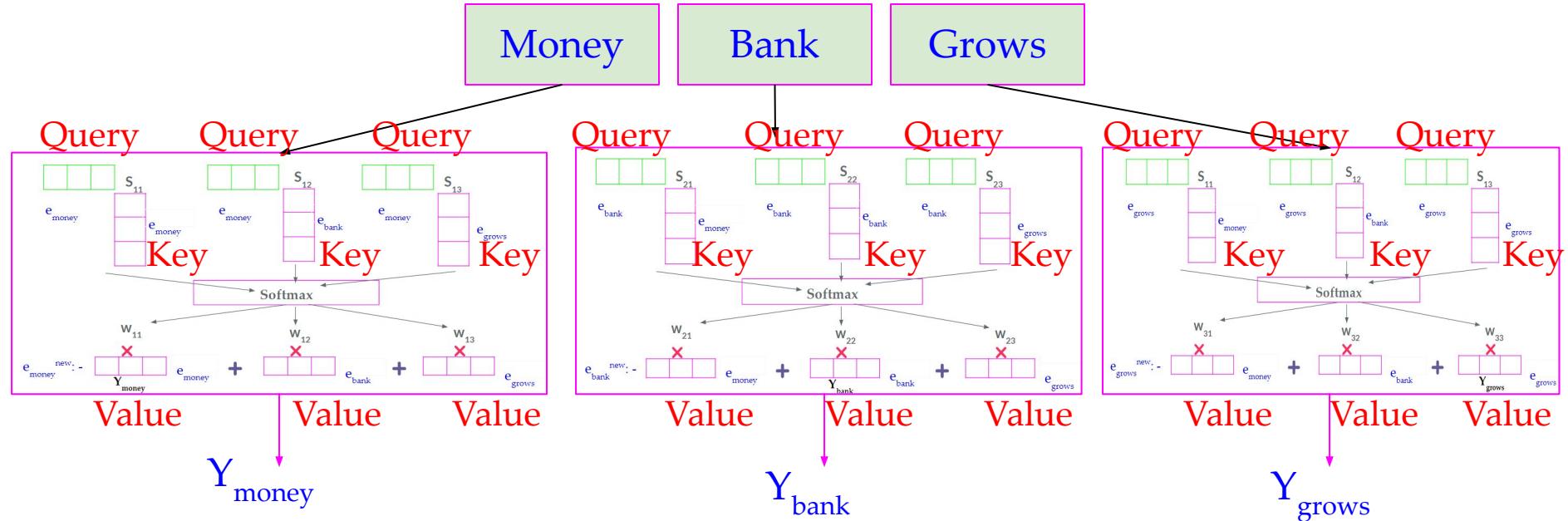


Self Attention: Introducing weight



The problem with the naive approach is that the **same initial embedding is used at three places and produces the contextual embeddings.**

Self Attention: Introducing weight



The problem with the naive approach is that the **same initial embedding is used at three places (as key, query, and value)** and produces the contextual embeddings.

Self Attention: Introducing weight

To calculate contextual vector e_{bank}^{new} :-
 e_{bank} is acting as Query, Key and Value.

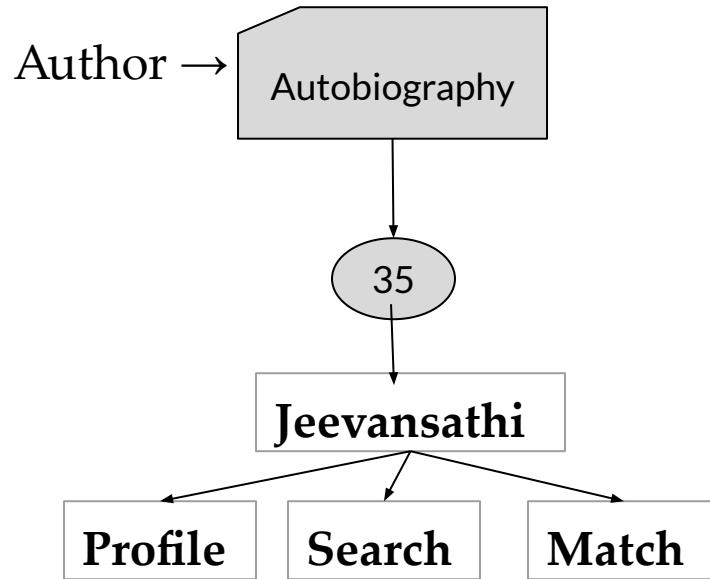
Problem: Separation of concern

It would be better, we will have three separate vectors transformed from e_{bank} as

Key vector: K_{bank}

Query vector: Q_{bank}

Value vector: V_{bank}



Self Attention: Introducing weight

To calculate contextual vector e_{bank}^{new} :-
 e_{bank} is acting as Query, Key and Value.

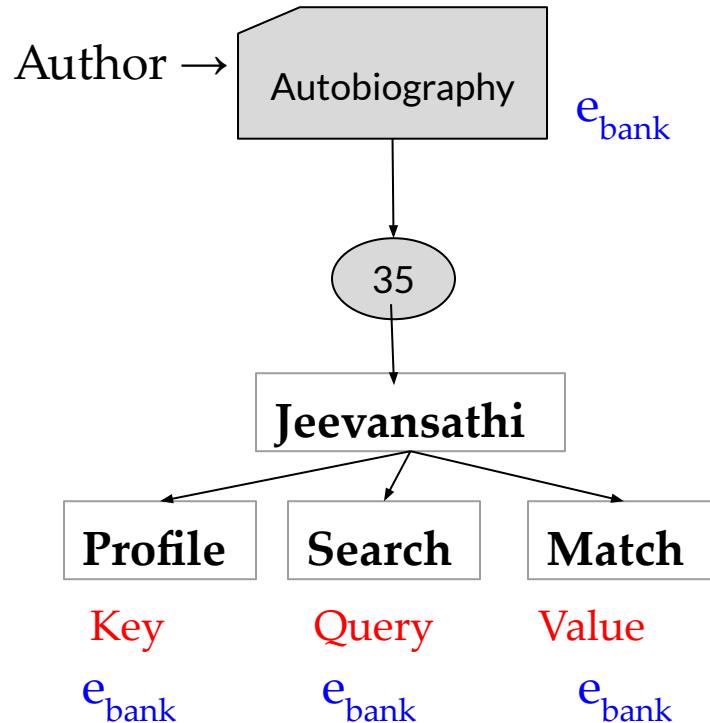
Problem: Separation of concern

It would be better, we will have three separate vectors transformed from e_{bank} as

Key vector: K_{bank}

Query vector: Q_{bank}

Value vector: V_{bank}



In the naive self attention model

Self Attention: Introducing weight

To calculate contextual vector e_{bank}^{new} :-
 e_{bank} is acting as Query, Key and Value.

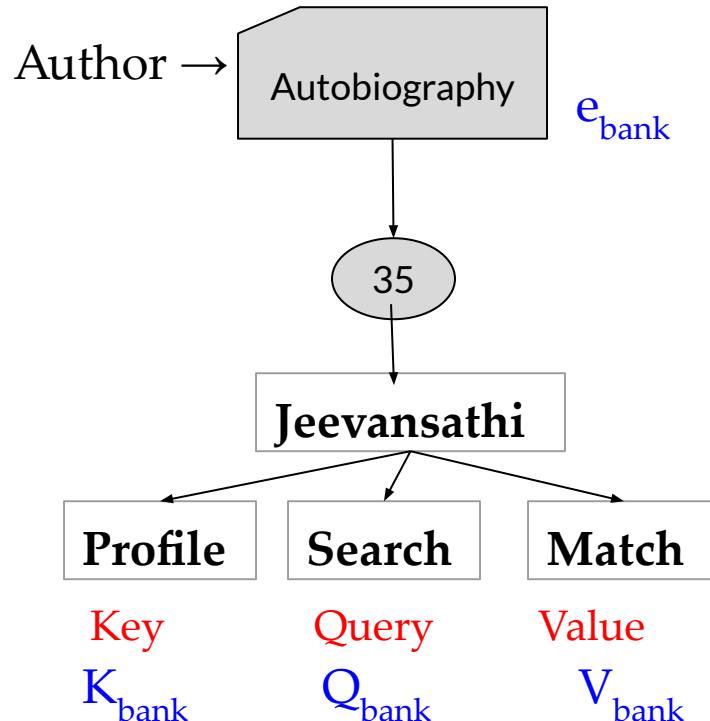
Problem: Separation of concern

It would be better, we will have three separate vectors transformed from e_{bank} as

Key vector: K_{bank}

Query vector: Q_{bank}

Value vector: V_{bank}



In the actual self attention model

Self Attention: Introducing weight

To calculate contextual vector e_{bank}^{new} :-
 e_{bank} is acting as Query, Key and Value.

Problem: Separation of concern

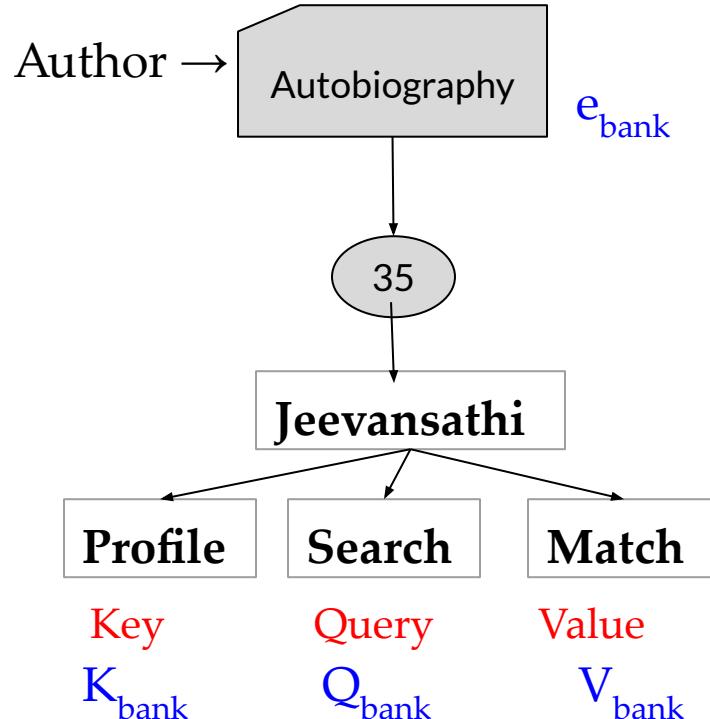
It would be better, we will have three separate vectors transformed from e_{bank} as

Key vector: K_{bank}

Query vector: Q_{bank}

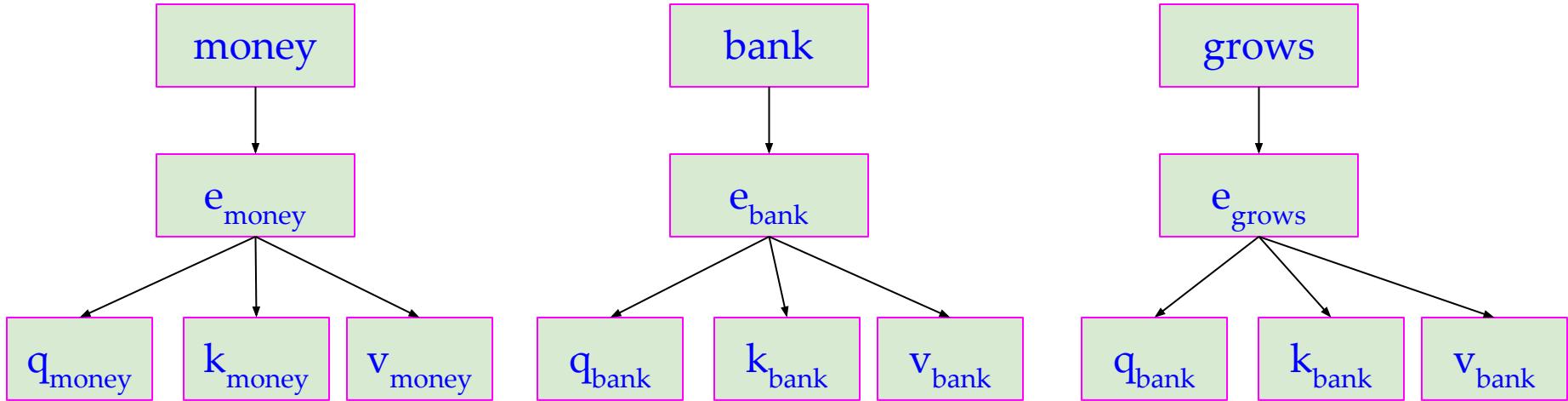
Value vector: V_{bank}

These vectors
are derived
from data



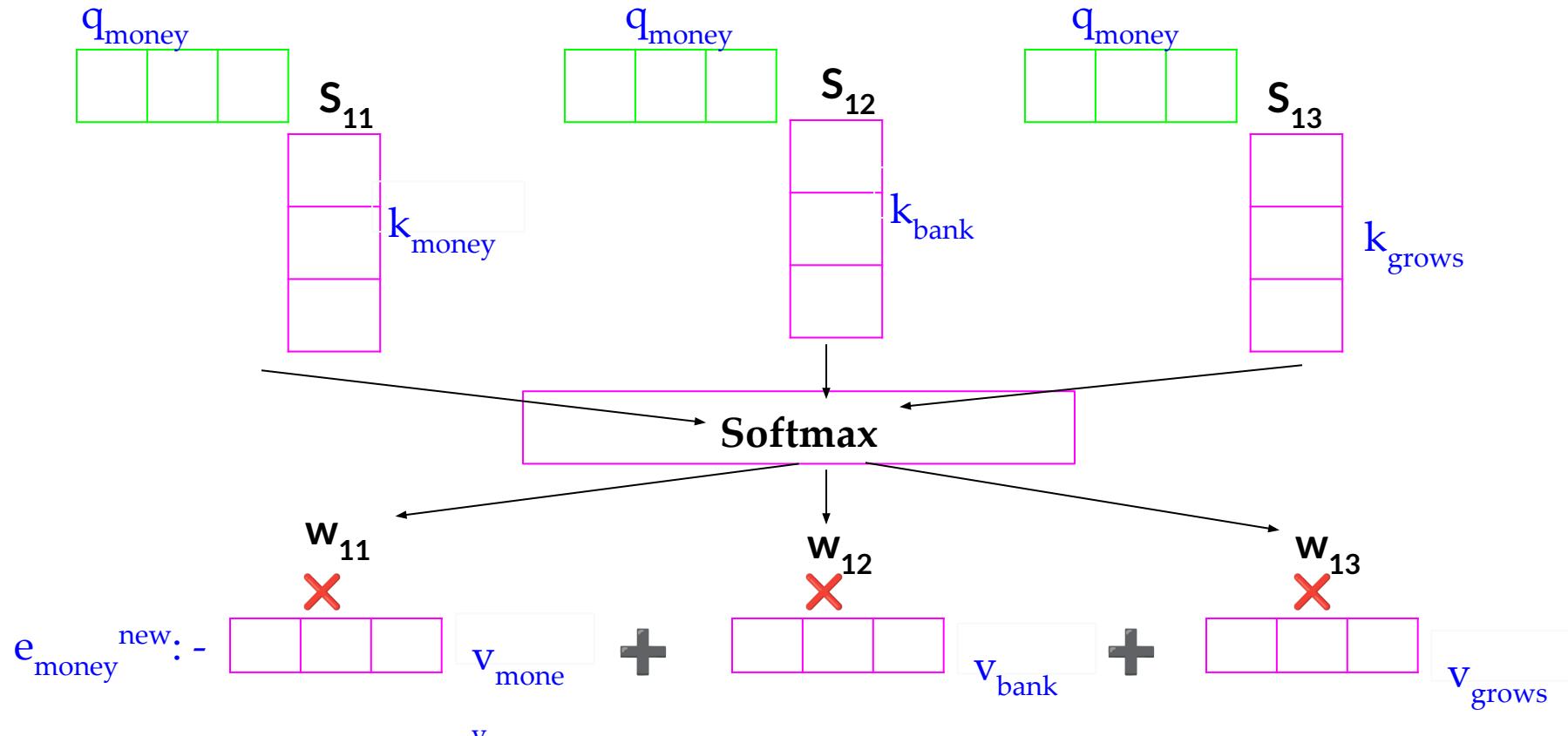
In the actual self attention model

Self Attention: redo



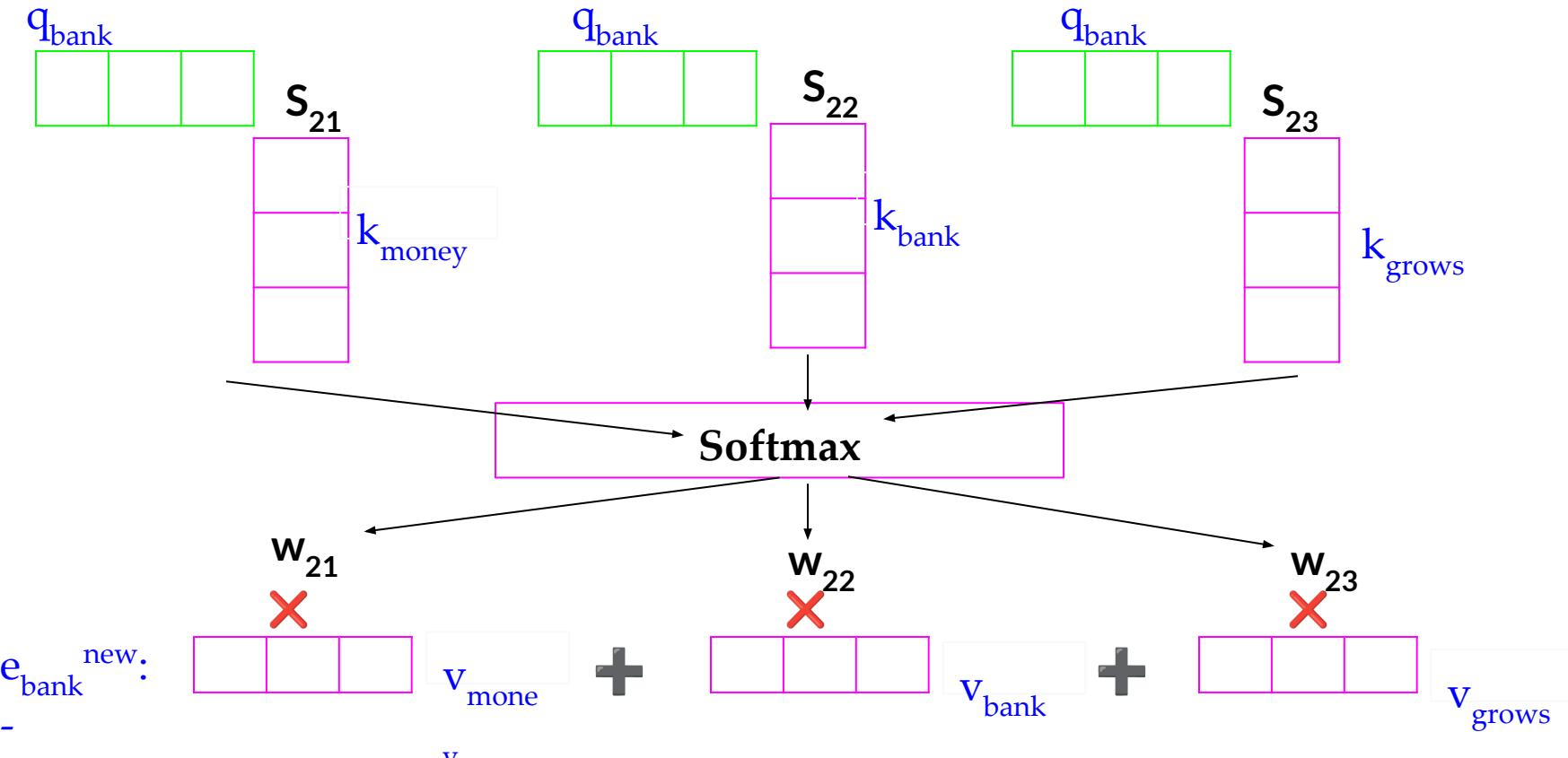
Self Attention: redo

Money Bank Grows



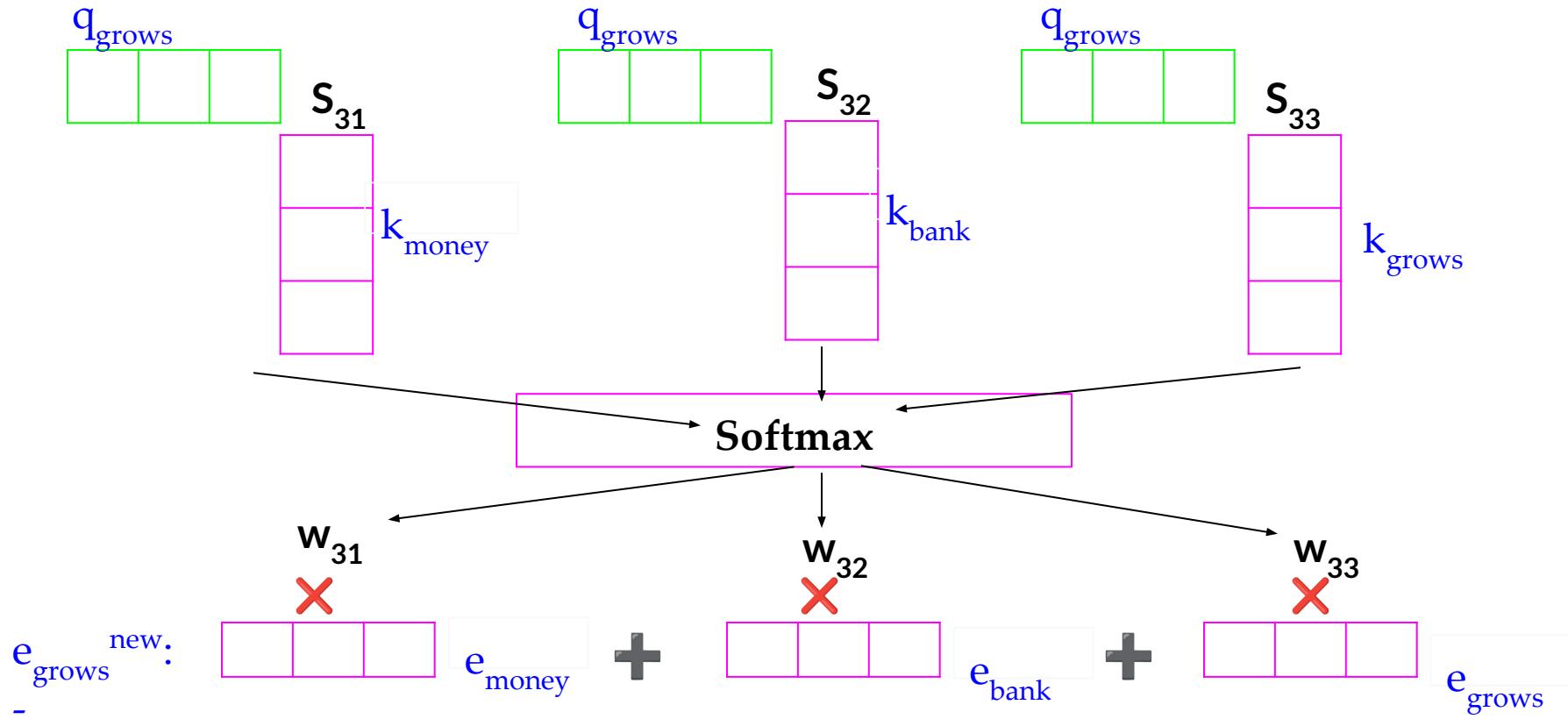
Self Attention: redo

Money Bank Grows

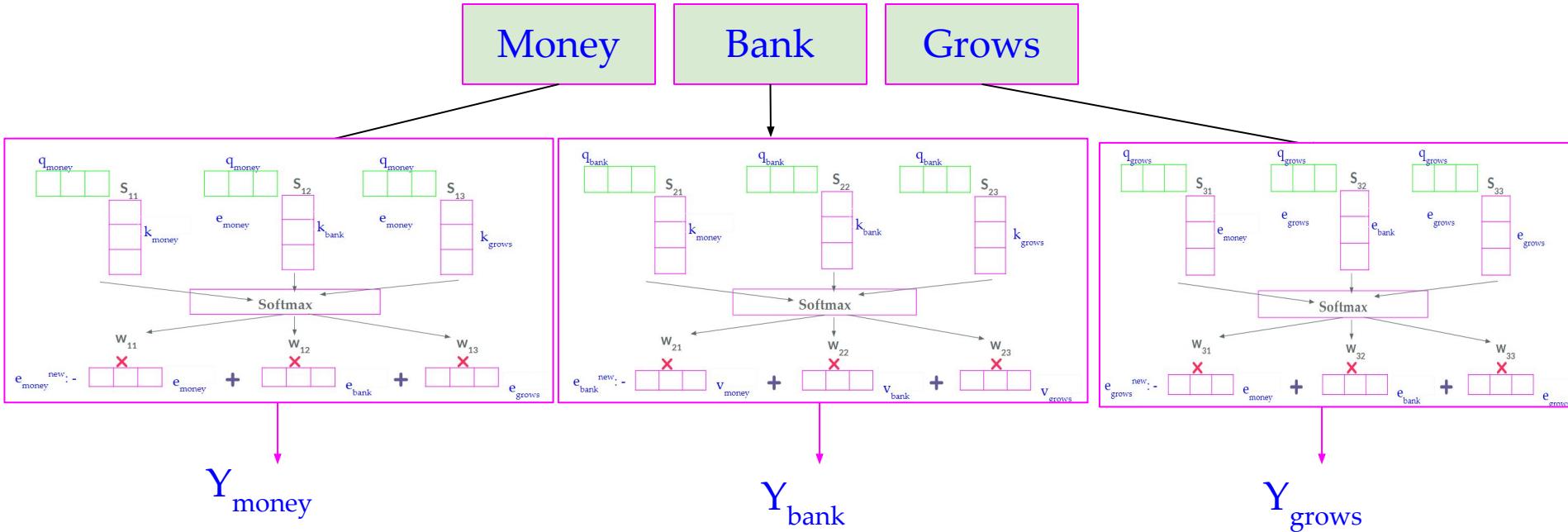


Self Attention: redo

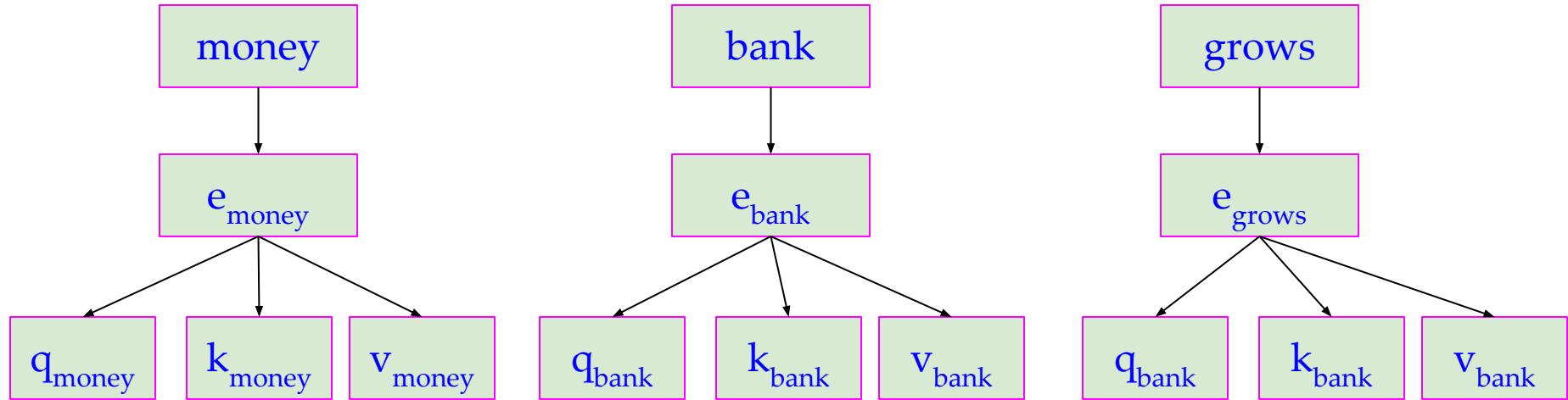
Money Bank Grows



Self Attention: redo



Self Attention: redo



Only one question remains unanswered that **how we create three new vectors from the given input embedding?**

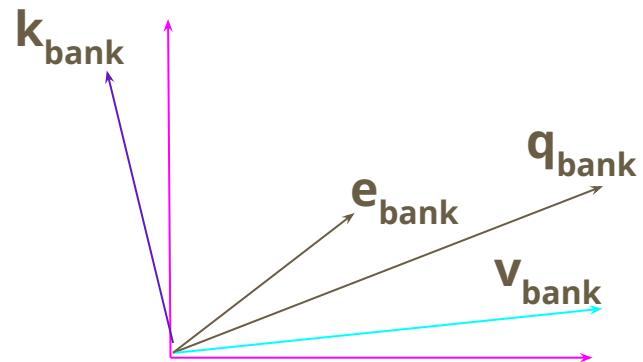
Self Attention: Key, Query and Value vectors

Only one question remains unanswered that **how we create three new vectors from the given input embedding?**

First way: Scaling (by varying magnitude)

Second way: Linear transformation

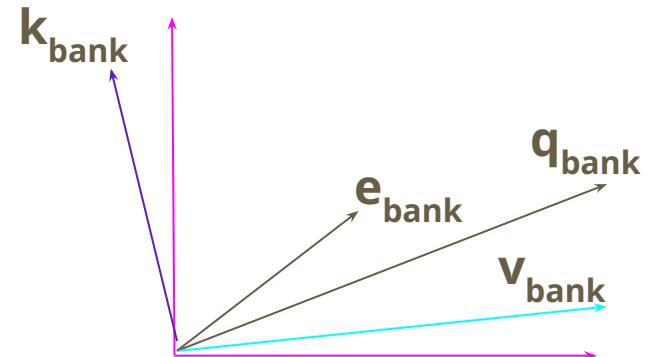
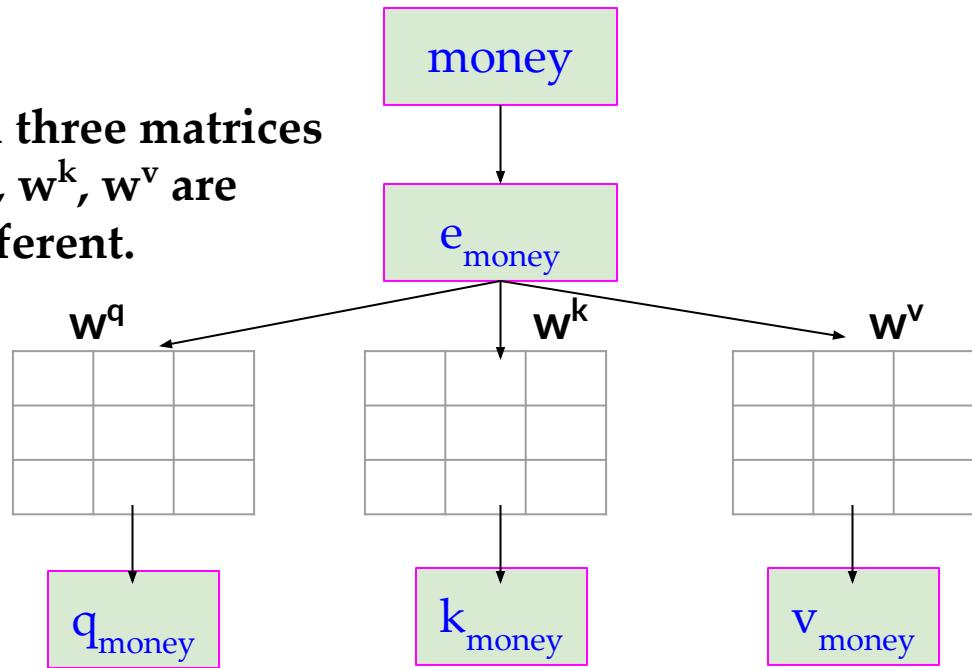
When a **vector** is multiplied by a matrix, it linearly transforms that coordinate system.



Self Attention: Key, Query and Value vectors

Second way: Linear transformation

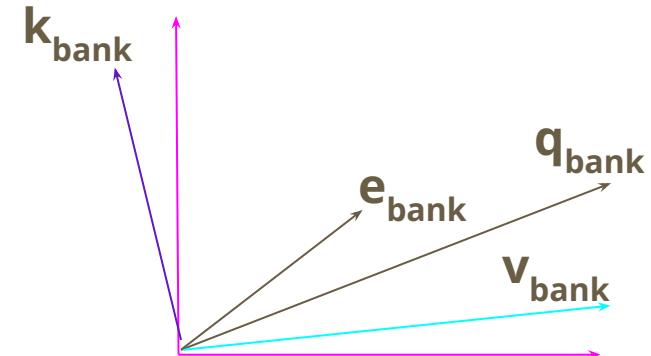
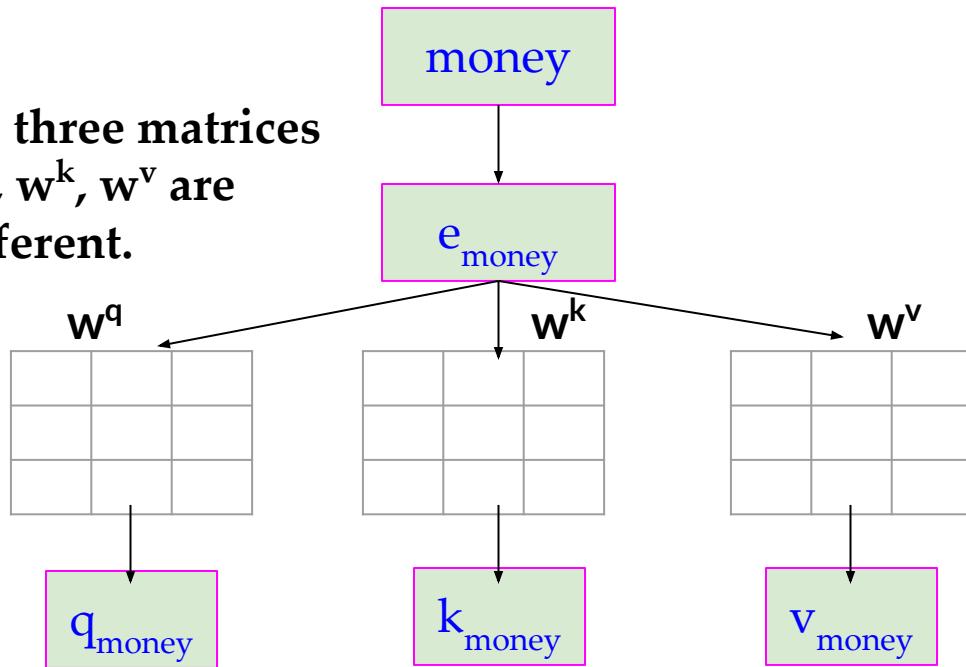
All three matrices
 w^q , w^k , w^v are
different.



Self Attention: Key, Query and Value vectors

Second way: Linear transformation

All three matrices
 w^q , w^k , w^v are
different.

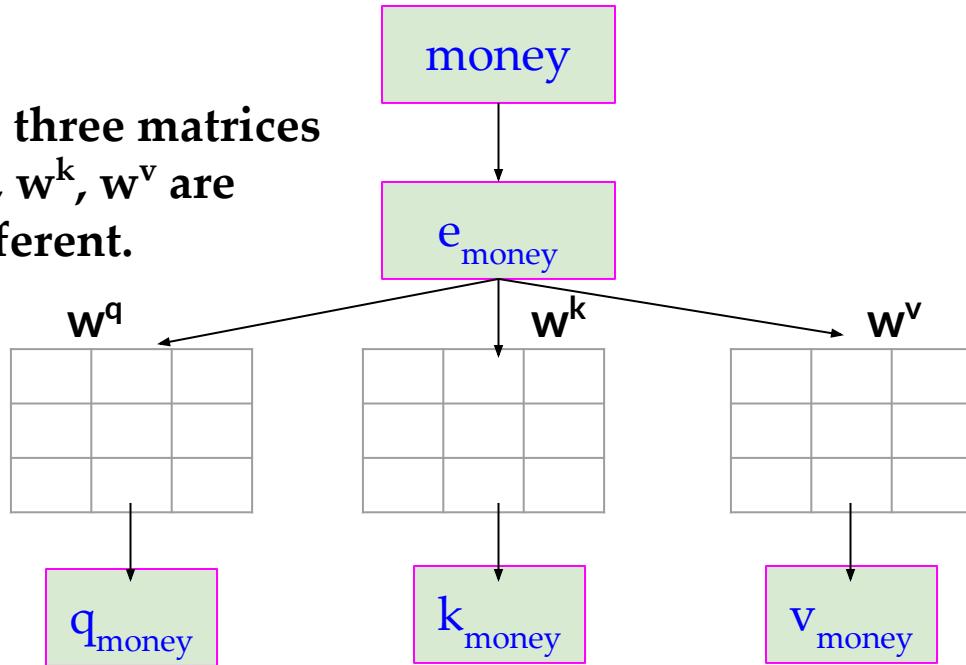


How do we get these matrices?

Self Attention: Key, Query and Value vectors

Second way: Linear transformation

All three matrices
 w^q , w^k , w^v are
different.



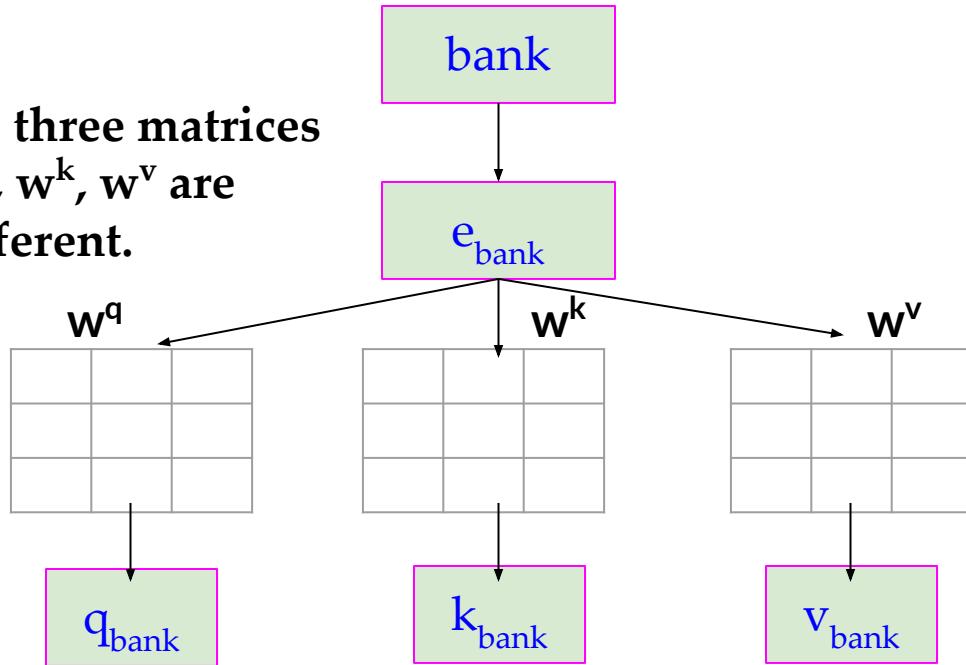
How do we get these matrices?

- Initially weights of these matrices will be random.
- It will learn from data with backpropagation.

Self Attention: Key, Query and Value vectors

Second way: Linear transformation

All three matrices
 w^q, w^k, w^v are
different.



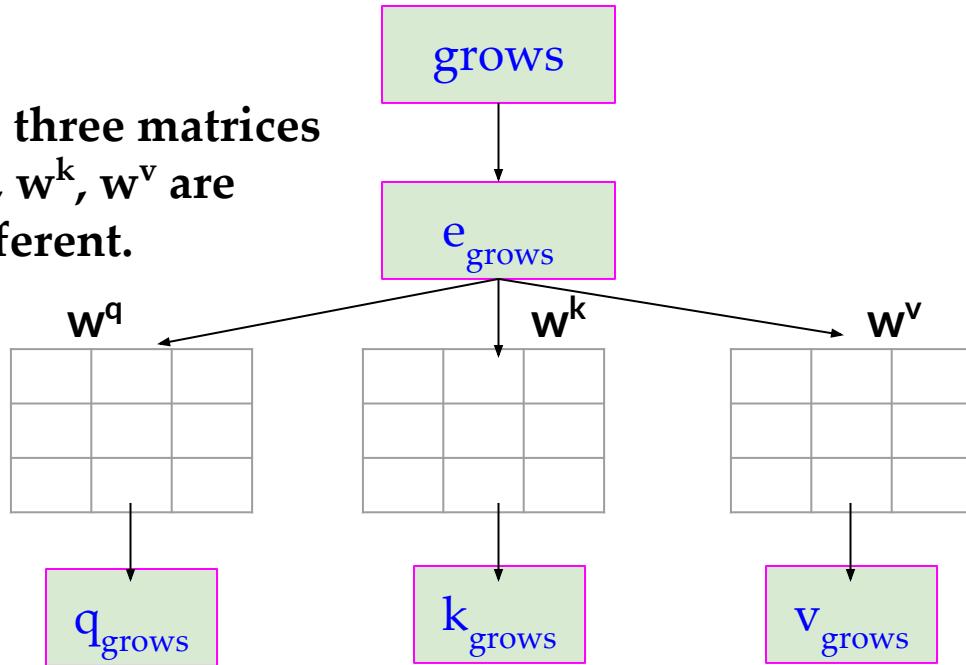
How do we get these matrices?

- Initially weights of these matrices will be random.
- It will learn from data with backpropagation.

Self Attention: Key, Query and Value vectors

Second way: Linear transformation

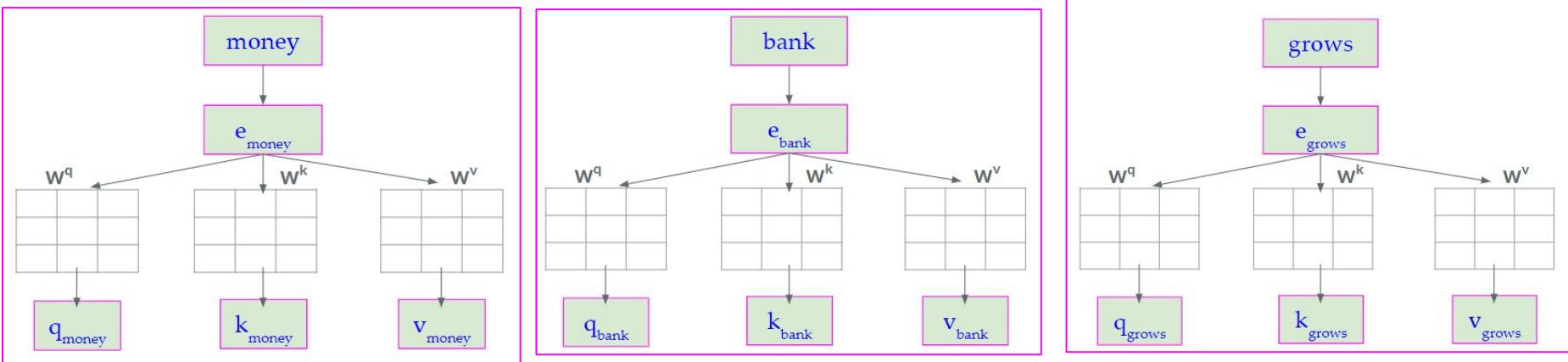
All three matrices
 w^q , w^k , w^v are
different.



How do we get these matrices?

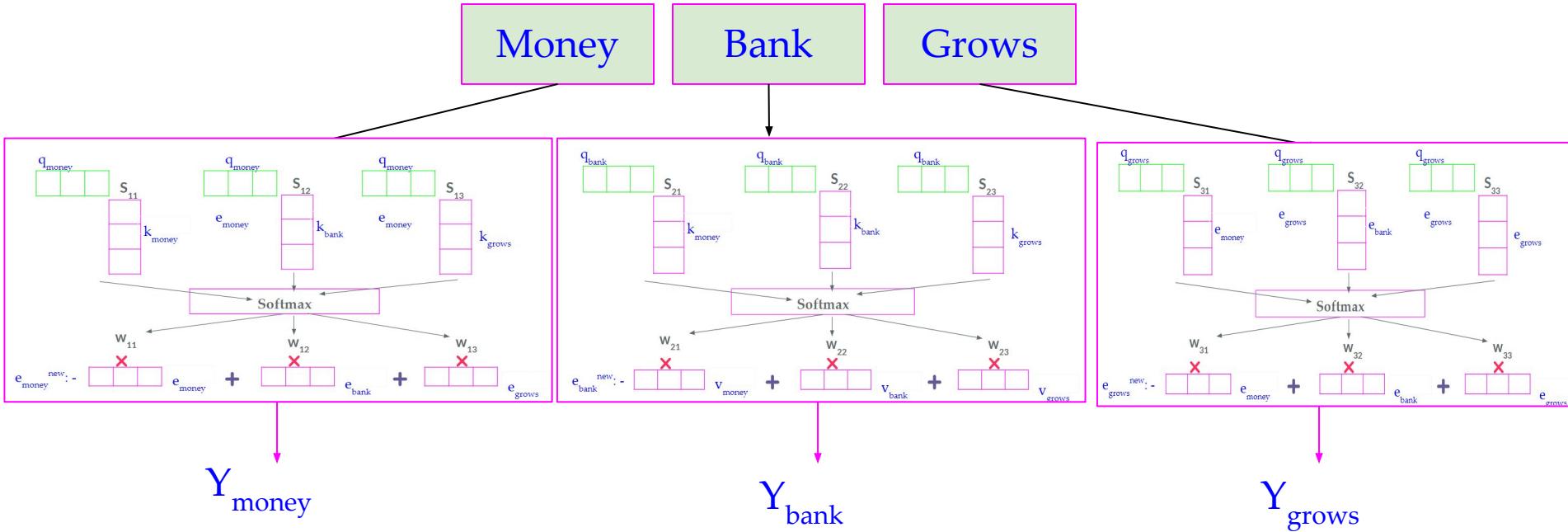
- Initially weights of these matrices will be random.
- It will learn from data with backpropagation.

Self Attention: Key, Query and Value vectors

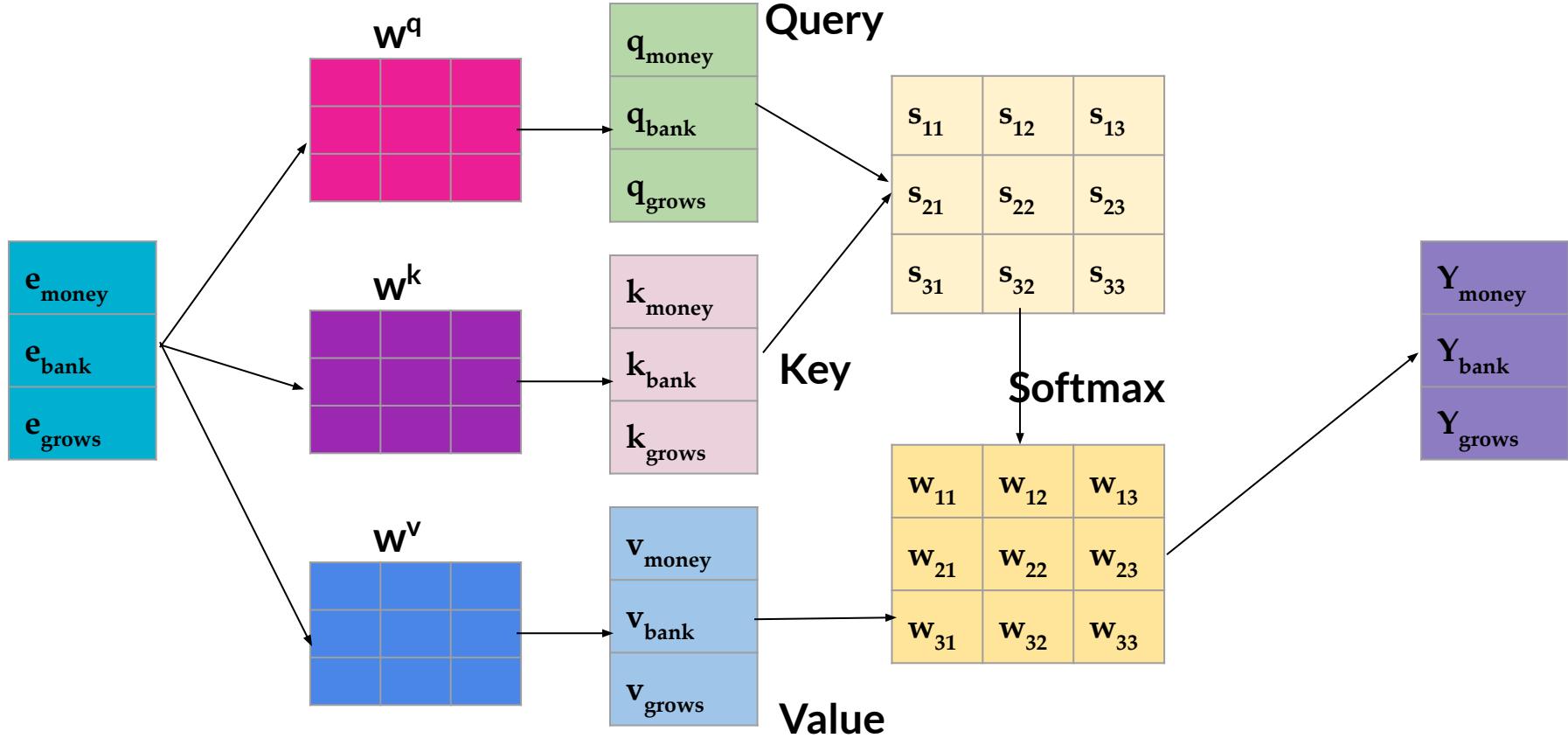


- All three matrices w^q , w^k , w^v are different.
- However, as we generate query, key and value for all words simultaneously, w^q is same in all the three word processing.
- The above statement is true for w^k and w^v also.

Self Attention: redo

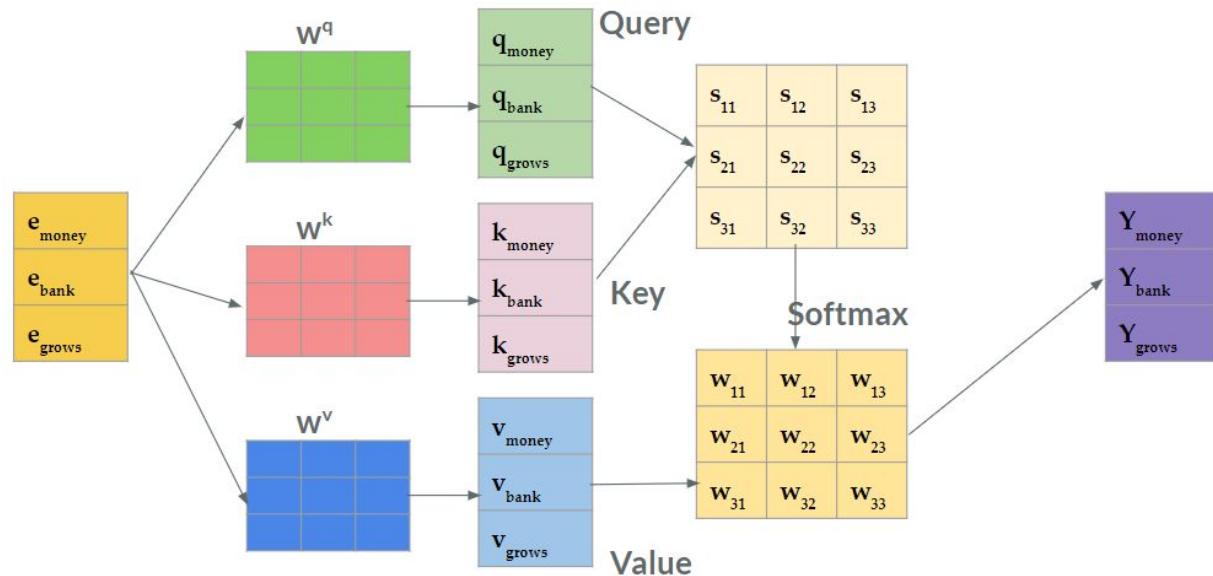


Self Attention: redo



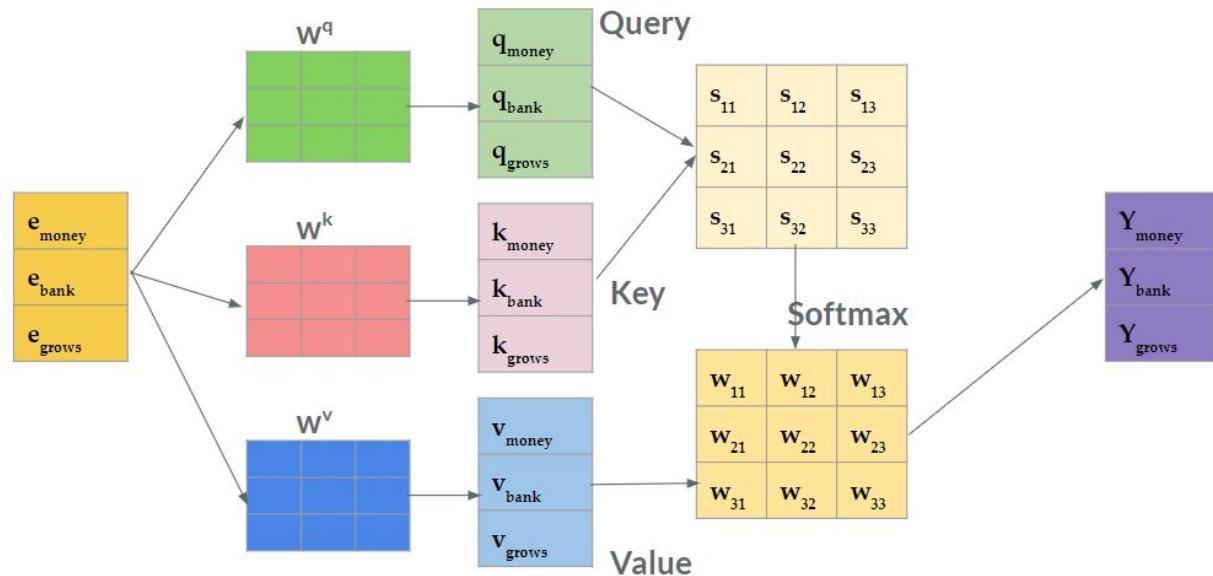
Self Attention: Mathematical form

Attention (Q,K,V) \Rightarrow (Q.K^T)



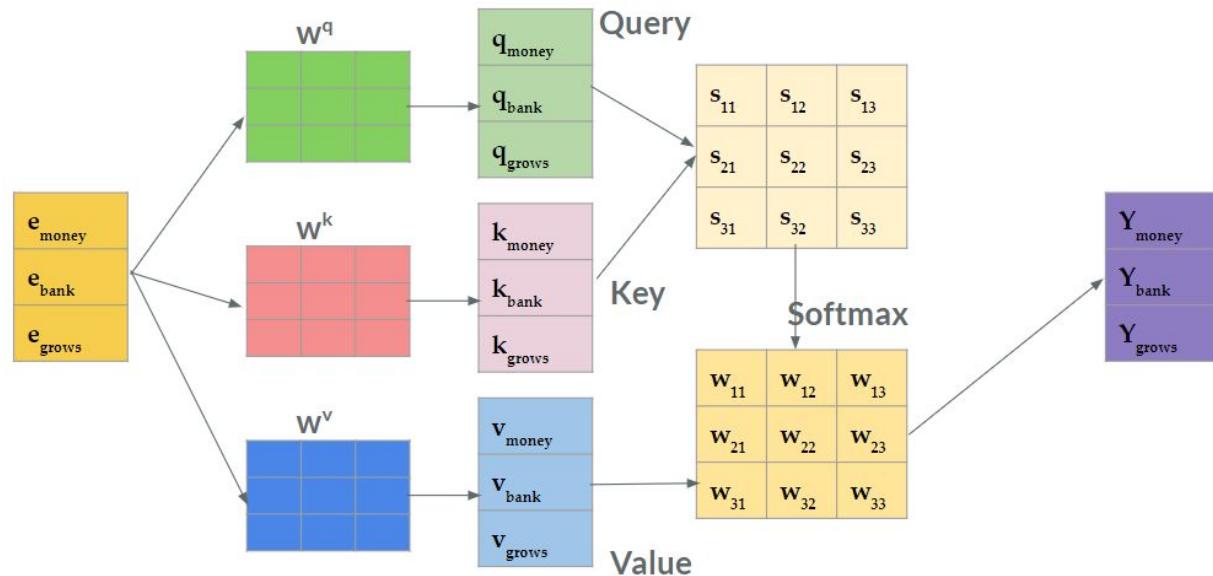
Self Attention: Mathematical form

Attention (Q,K,V) \Rightarrow Softmax(Q.K^T)



Self Attention: Mathematical form

Attention (Q,K,V) \Rightarrow Softmax(Q.K^T)V

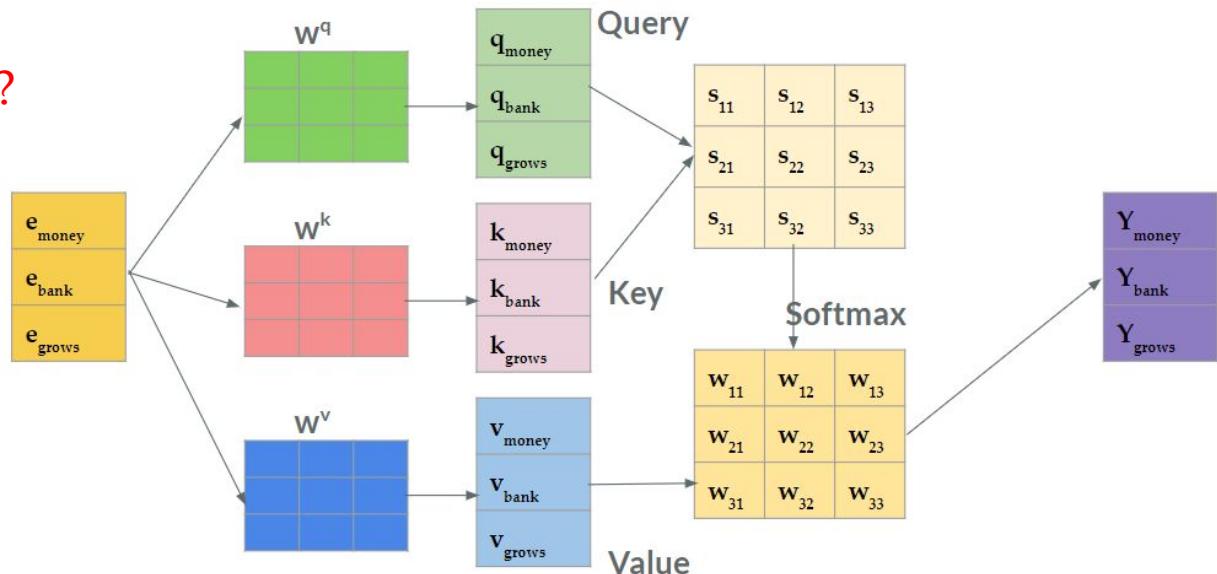


Self Attention: Mathematical form

Attention (Q,K,V) \Rightarrow Softmax(Q.K^T)V

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad \underline{\text{In the original paper}}$$

What is in the denominator?



Scaled dot product Attention

Attention (Q,K,V) \Rightarrow Softmax(Q.K^T)V

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad \text{In the original paper}$$

What is in the denominator?

We are scaling the factor QK^T with $\sqrt{d_k}$

We need to understand the following:

- What is d_k ?
- Why it was required to divide QK^T with $\sqrt{d_k}$

Scaled dot product Attention

Attention (Q,K,V) \Rightarrow Softmax(Q.K^T)V

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad \text{In the original paper}$$

We need to understand the following:

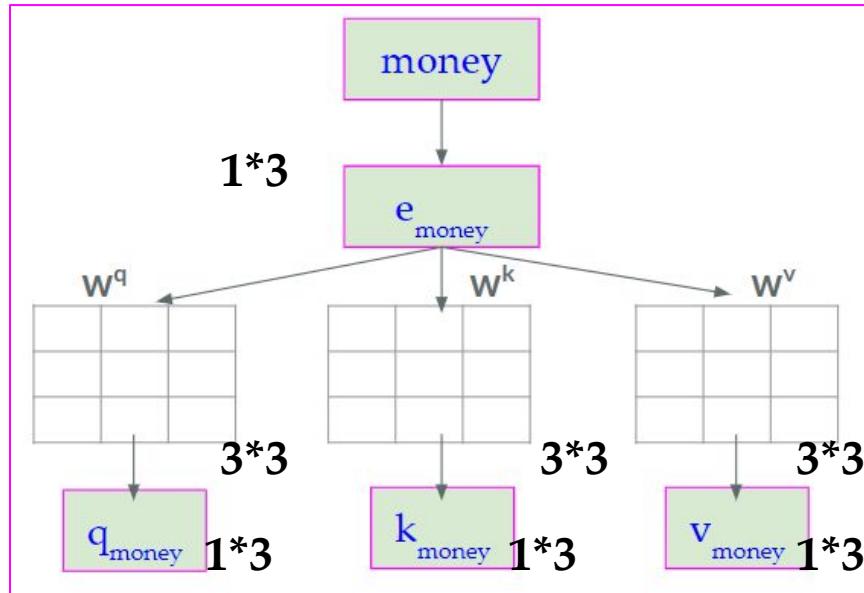
- **What is d_k :** Dimensionality of K vector
- Why it was required to divide QK^T with $\sqrt{d_k}$
 - to avoid the unstable gradient we divide QK^T with $\sqrt{d_k}$

Scaled dot product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

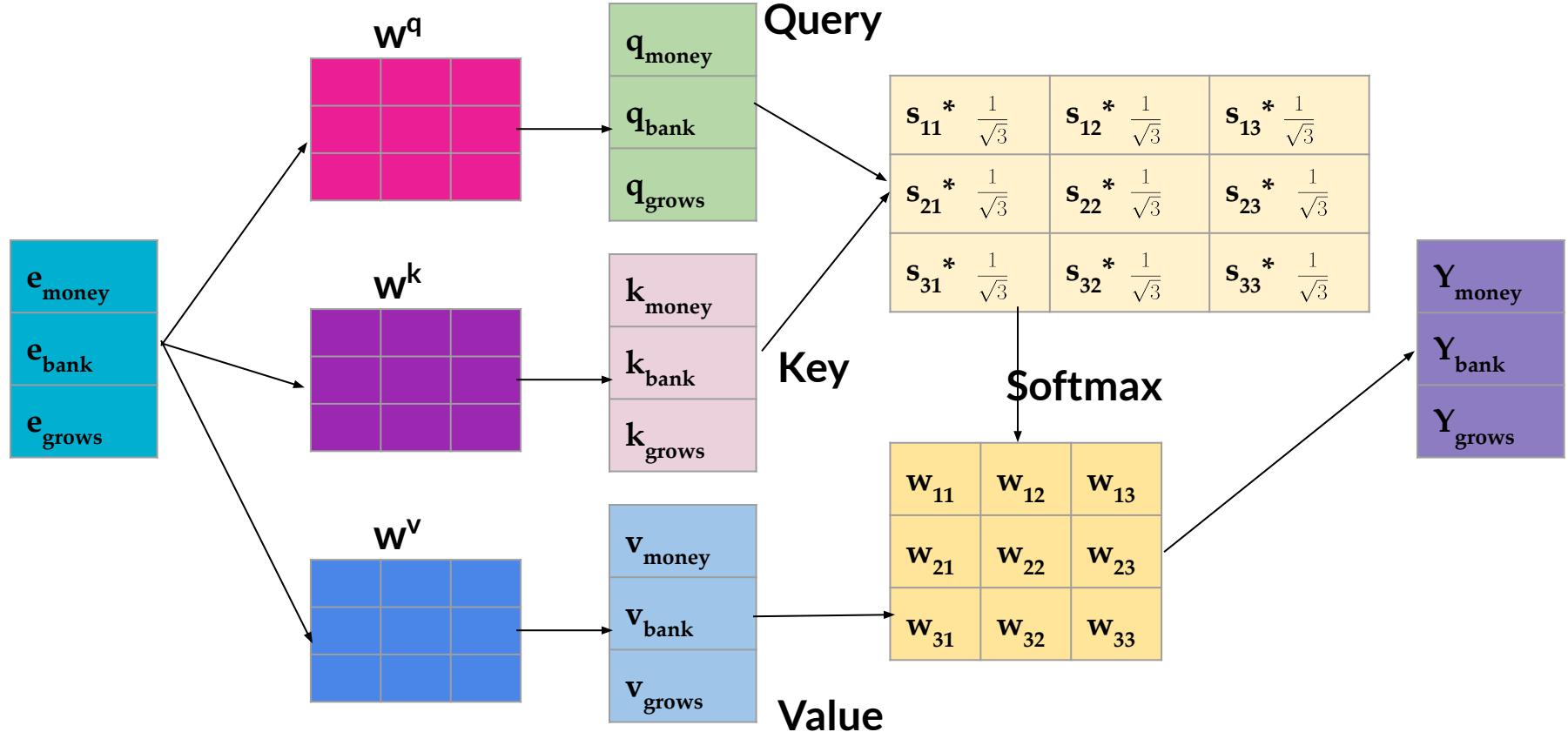
We need to understand the following:

- **What is d_k :** Dimensionality of K vector = 3
- Why it was required to divide QK^T with $\sqrt{d_k} = \sqrt{3}$
 - to avoid the unstable gradient we divide QK^T with $\sqrt{d_k}$



Scaled dot product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



Scaled dot product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

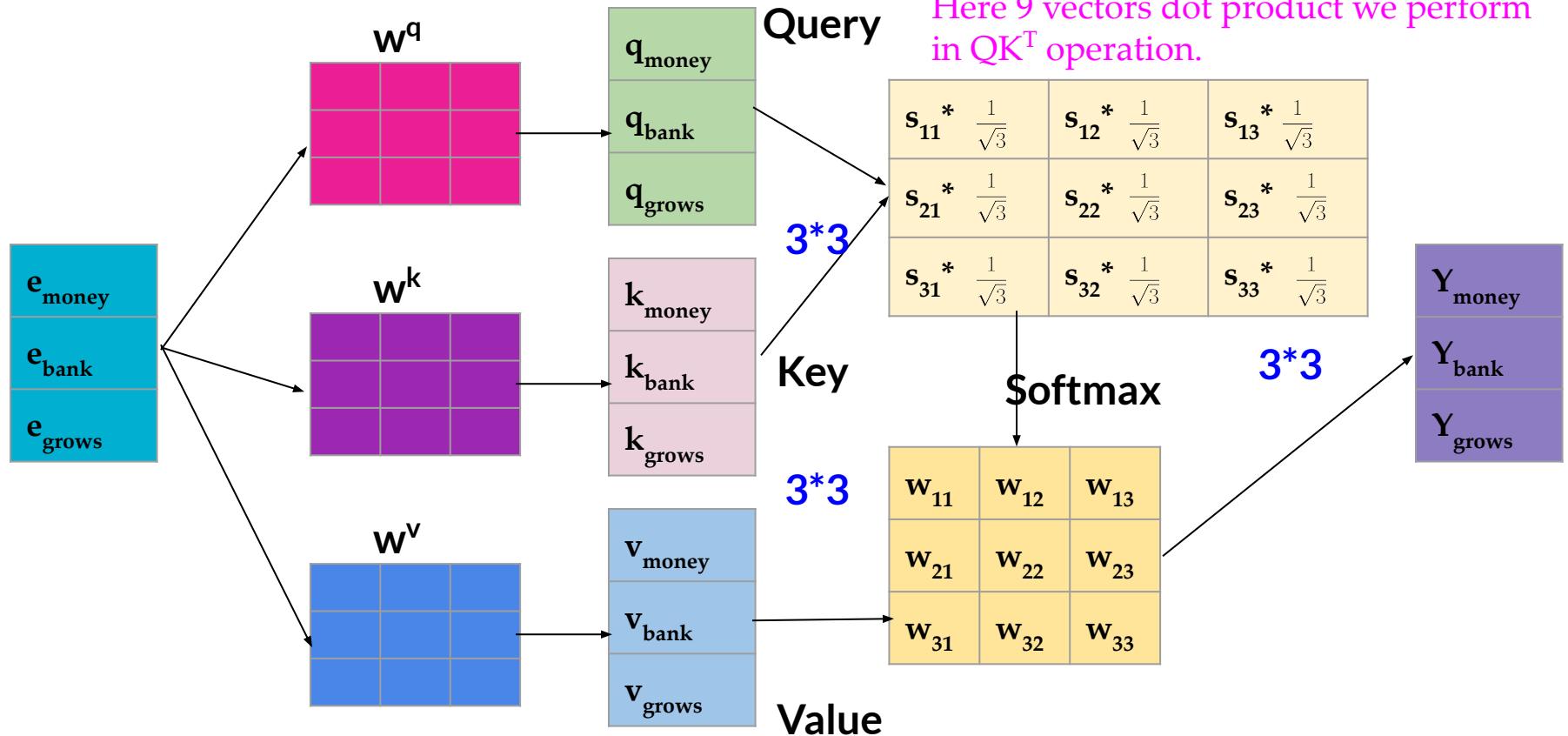
Why to scale with $\frac{1}{\sqrt{d_k}}$?

Ans: Due to the nature of dot product

While doing the dot product between Q and K^T , how many vectors dot product we are doing?

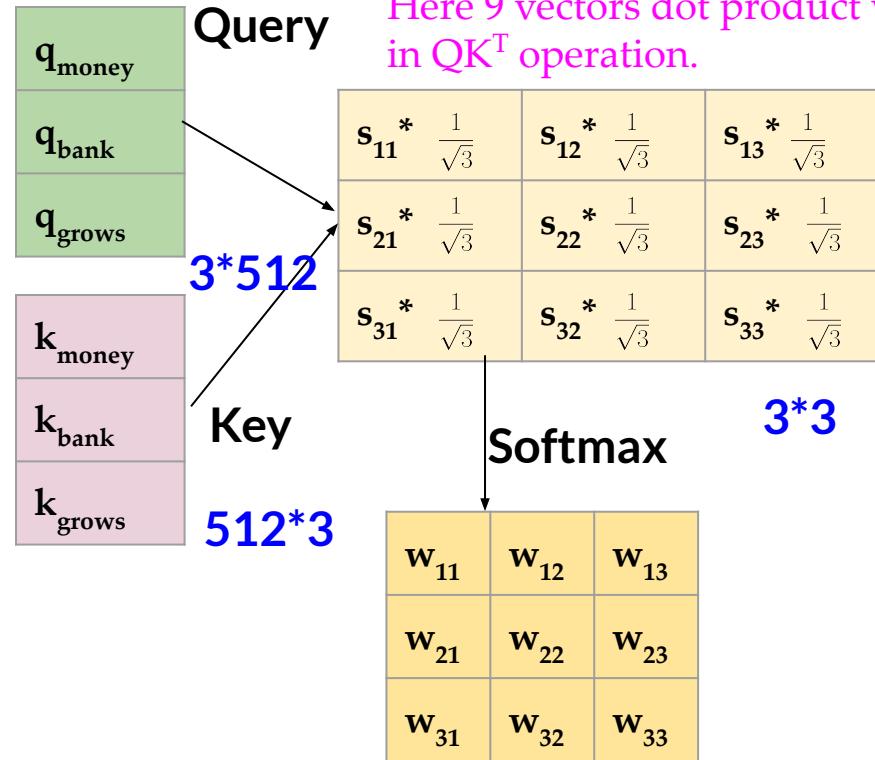
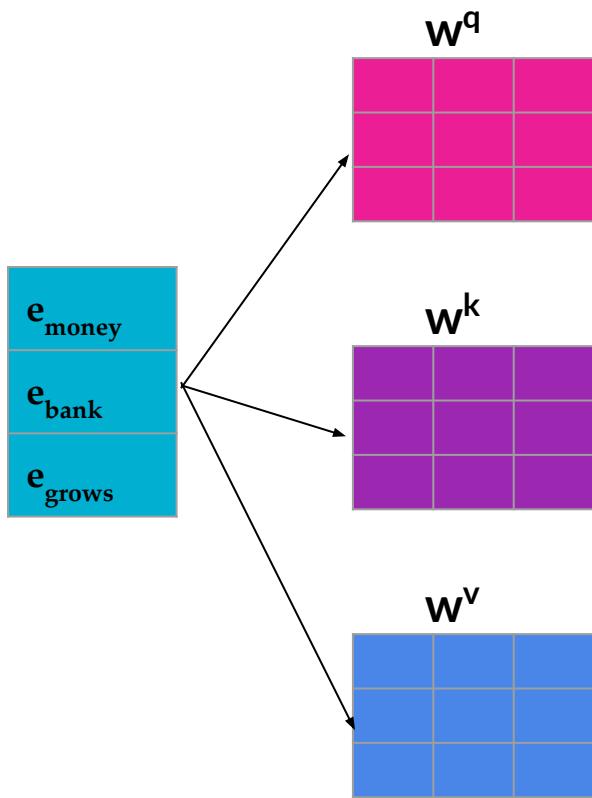
Scaled dot product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



Scaled dot product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$



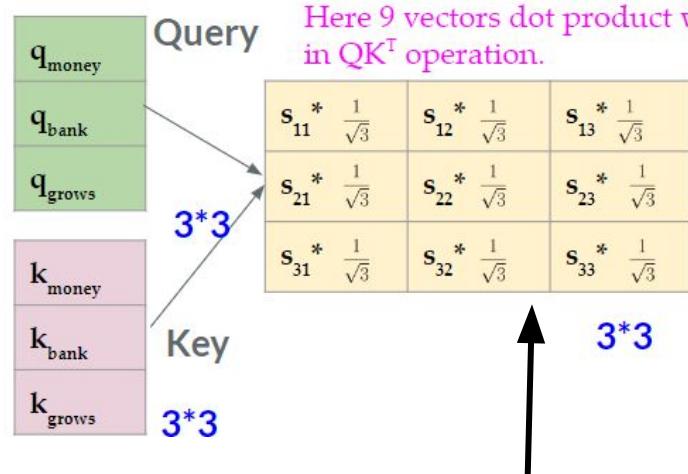
Scaled dot product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Why to scale with $\frac{1}{\sqrt{d_k}}$?

Ans: Due to the nature of dot product

While doing the dot product between Q and K^T , how many vectors dot product we are doing?



Here 9 vectors dot product we perform in QK^T operation.



For resultant matrix of QK^T we can calculate the μ and σ

Scaled dot product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

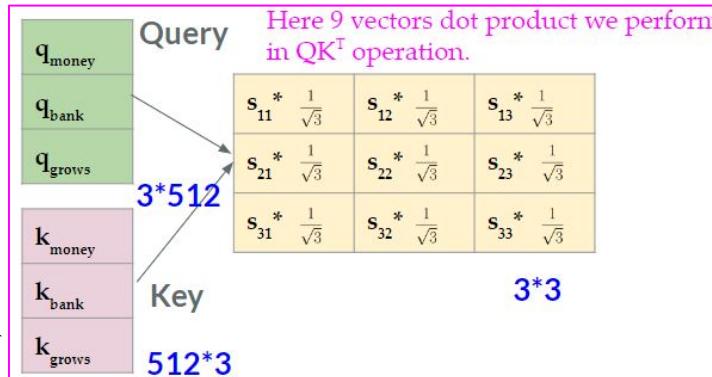
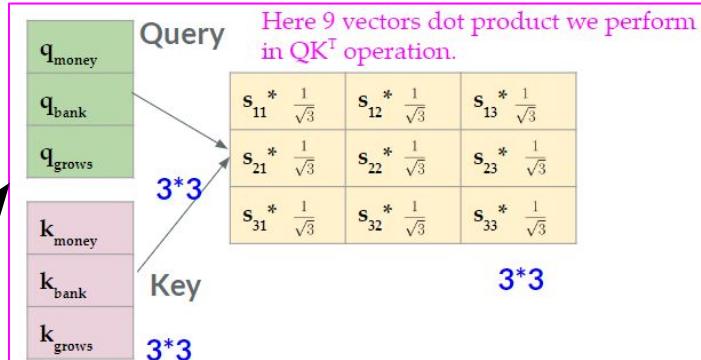
Why to scale with

$$\frac{1}{\sqrt{d_k}}$$

Ans: Due to the nature of dot product

Low dimensional → Dot Product → Low variance matrix

High dimensional → Dot Product → High variance matrix



Nature of dot product

Consider we have 1000 pairs of vectors each with **3 dimension**:

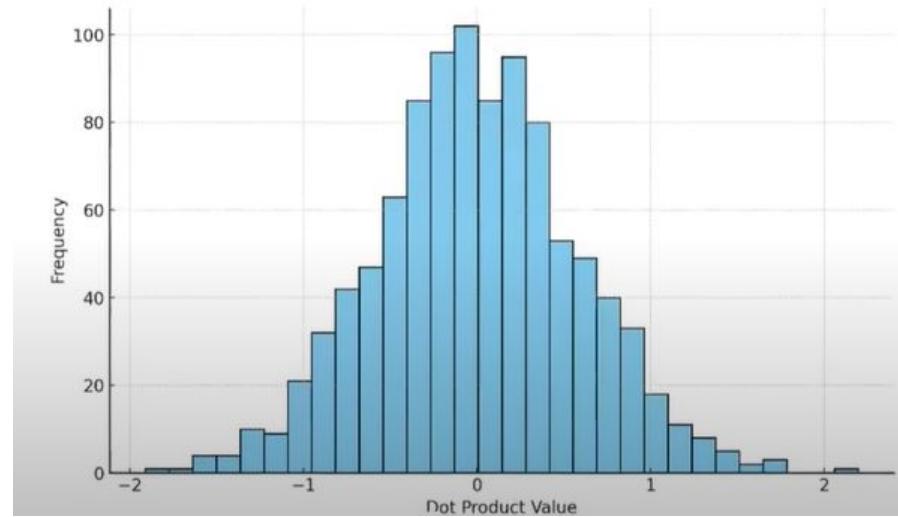
$[-,-,-] \cdot [-,-,-]$ → scalar

$[-,-,-] \cdot [-,-,-]$ → scalar

$[-,-,-] \cdot [-,-,-]$ → scalar

.....

$[-,-,-] \cdot [-,-,-]$ → scalar



Nature of dot product

Consider we have 1000 pairs of vectors
each with 100 dimension:

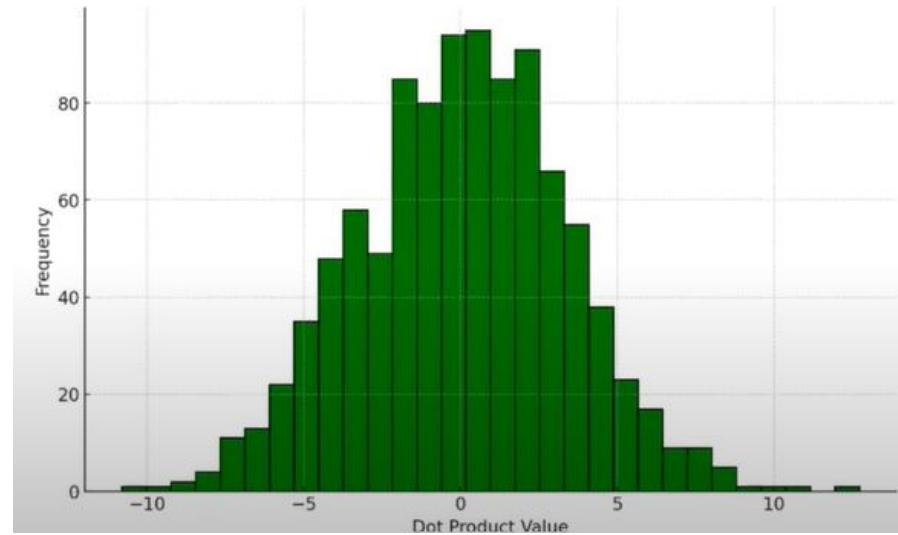
$[-,-,-,\dots,-] \cdot [-,-,-,\dots,-]$ → scalar

$[-,-,-,\dots,-] \cdot [-,-,-,\dots,-]$ → scalar

$[-,-,-,\dots,-] \cdot [-,-,-,\dots,-]$ → scalar

.....

$[-,-,-,\dots,-] \cdot [-,-,-,\dots,-]$ → scalar



Nature of dot product

Consider we have 1000 pairs of vectors
each with 1000 dimension:

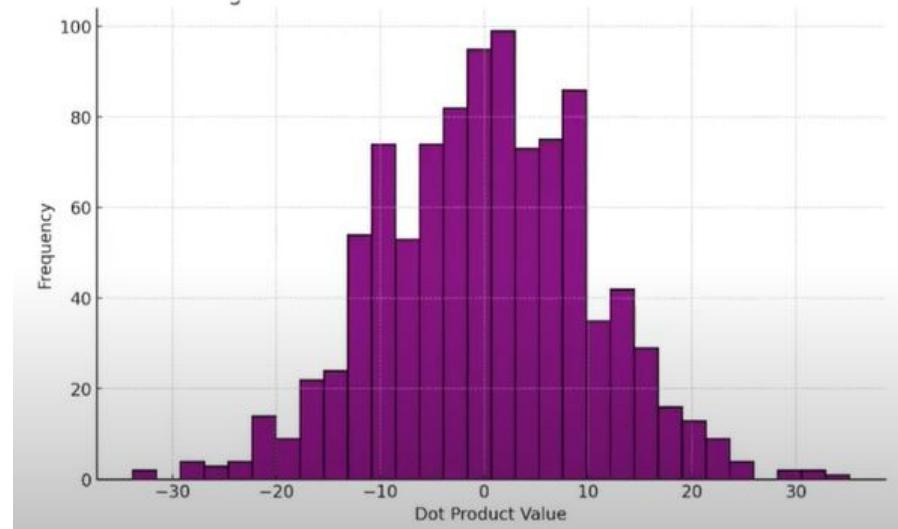
$[-,-,-,\dots,-] \cdot [-,-,-,\dots,-]$ → scalar

$[-,-,-,\dots,-] \cdot [-,-,-,\dots,-]$ → scalar

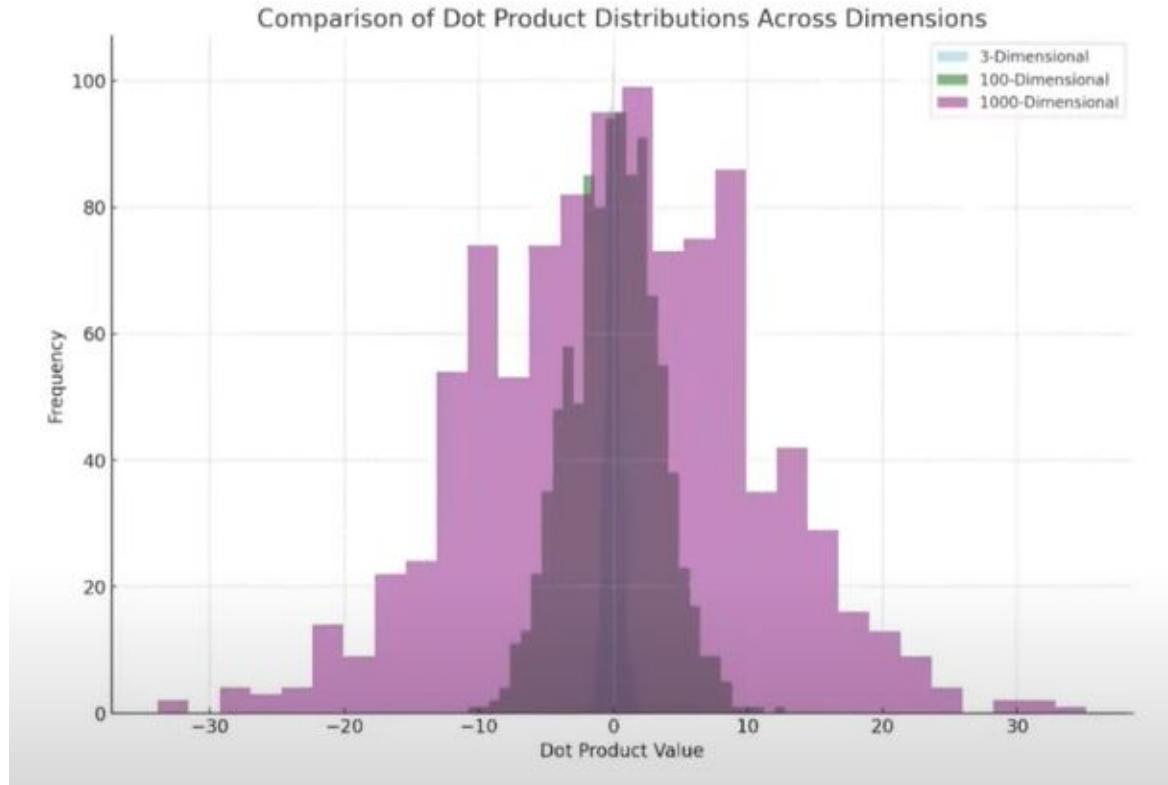
$[-,-,-,\dots,-] \cdot [-,-,-,\dots,-]$ → scalar

.....

$[-,-,-,\dots,-] \cdot [-,-,-,\dots,-]$ → scalar



Nature of dot product



Scaled dot product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

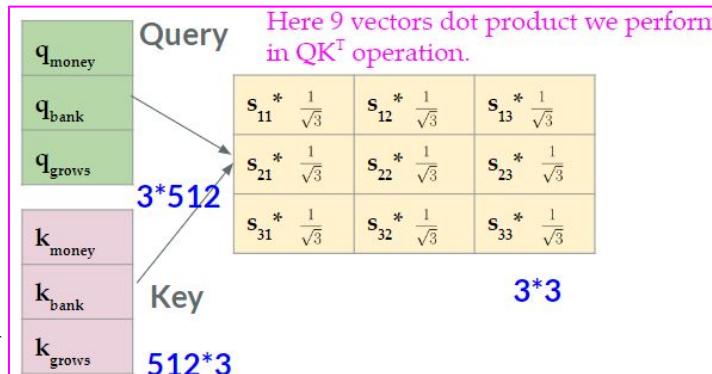
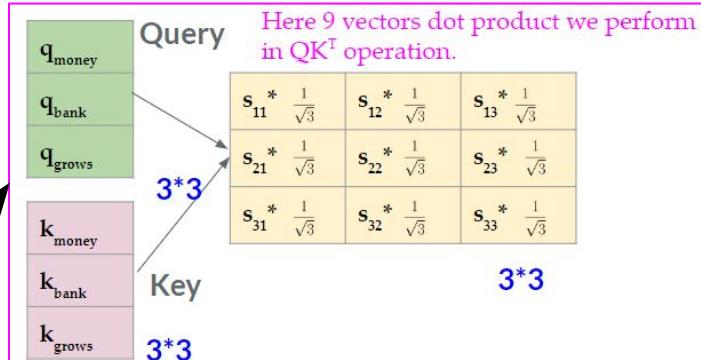
Why to scale with $\frac{1}{\sqrt{d_k}}$?

Ans: Due to the nature of dot product

Low dimensional \rightarrow Dot Product \rightarrow Low variance matrix

High dimensional \rightarrow Dot Product \rightarrow High variance matrix

Having high variance in QK^T matrix is a problem



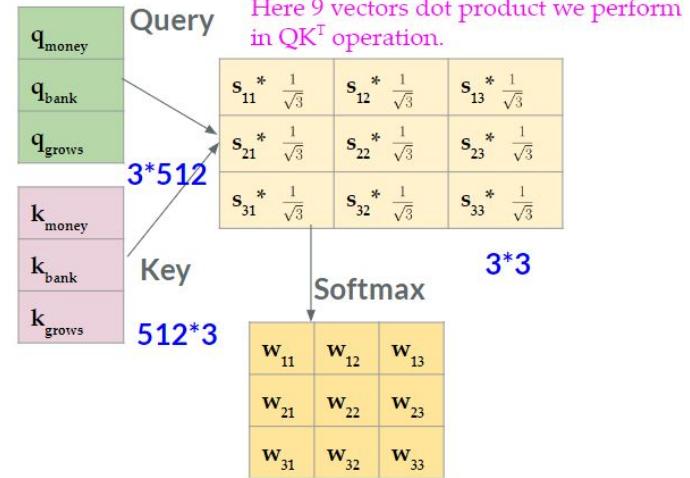
Scaled dot product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Why to scale with

$$\frac{1}{\sqrt{d_k}}$$

Ans: Due to the nature of dot product



Low dimensional → Dot Product → Low variance matrix

High dimensional → Dot Product → High variance matrix

Having high variance in QK^T matrix is a problem

Due to high variance in QK^T matrix, the softmax output will be very sparse.

For high value softmax give 99.9%

For low value softmax give 0.09%

During backpropagation, the lower number will be ignored.

Scaled dot product Attention

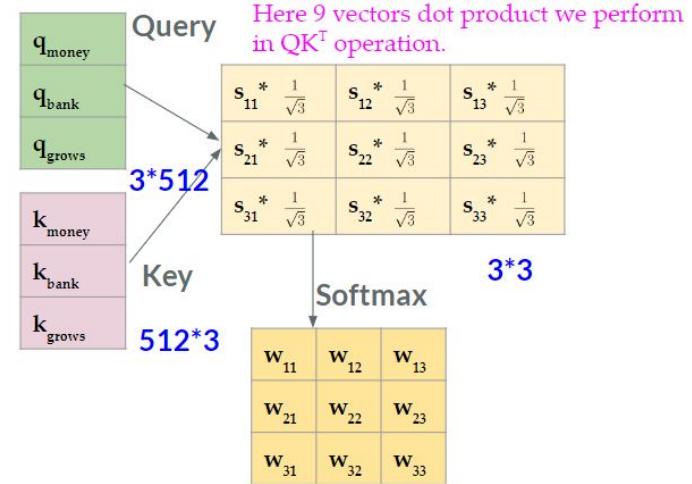
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Why to scale with

$$\frac{1}{\sqrt{d_k}}$$

Solution is to reduce the variance of QK^T matrix.

- Resultant softmax matrix elements will also be evenly distributed between 0-1.
- Therefore, all number based on its weightage will be considered during training.



How to reduce the variance of QK^T matrix?

Scaled dot product Attention

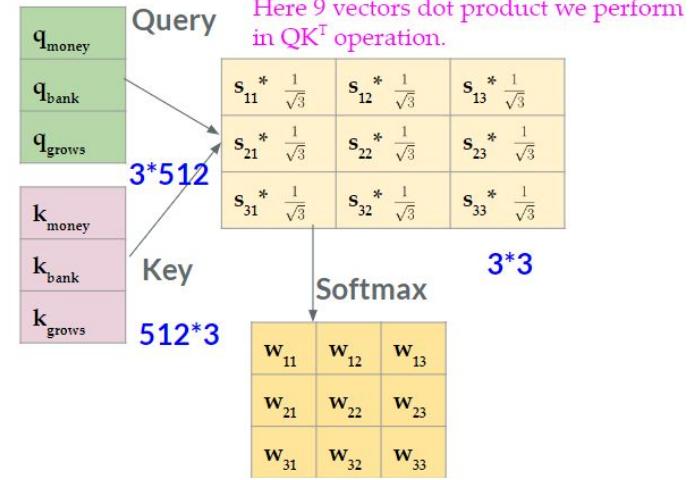
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Why to scale with

$$\frac{1}{\sqrt{d_k}}$$

Solution is to reduce the variance of QK^T matrix.

- Resultant softmax matrix elements will also be evenly distributed between 0-1.
- Therefore, all numbers based on its weightage will be considered during training.



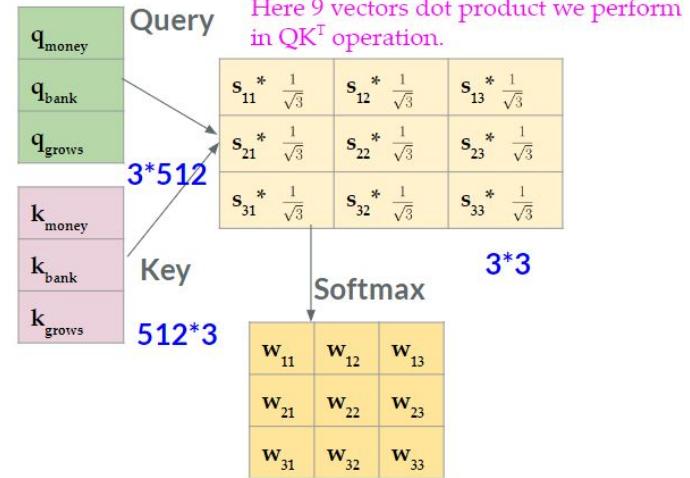
How to reduce the variance of QK^T matrix? → Scale the QK^T matrix.

Scaled dot product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Why to scale with

$$\frac{1}{\sqrt{d_k}}$$



How to reduce the variance of QK^T matrix? \rightarrow Scale the QK^T matrix.

What should be the scaling factor?

Scaled dot product Attention

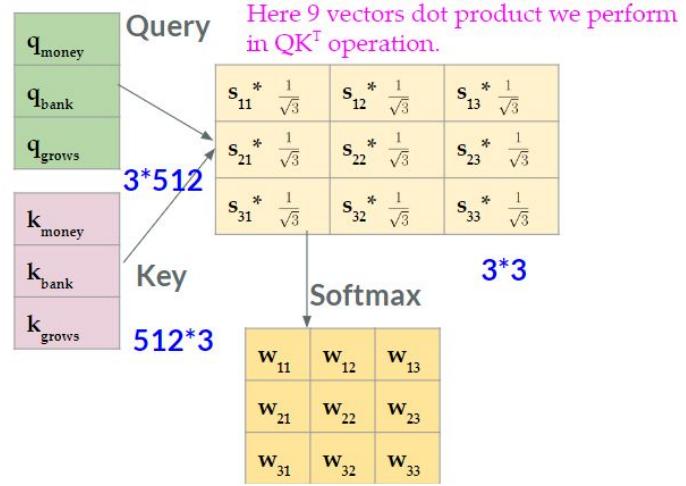
What should be the scaling factor?

To get the 1st row of QK^T matrix, we need to do:

$$q_{\text{money}} \cdot k_{\text{money}} \rightarrow \text{scalar}$$

$$q_{\text{money}} \cdot k_{\text{bank}} \rightarrow \text{scalar}$$

$$q_{\text{money}} \cdot k_{\text{grows}} \rightarrow \text{scalar}$$



Scaled dot product Attention

What should be the scaling factor?

Consider all vectors below is of 1 dimension

$$q_{\text{money}} \cdot k_{\text{money}} \rightarrow [a].[b] \rightarrow \text{scalar}$$

$$q_{\text{money}} \cdot k_{\text{bank}} \rightarrow [a].[c] \rightarrow \text{scalar}$$

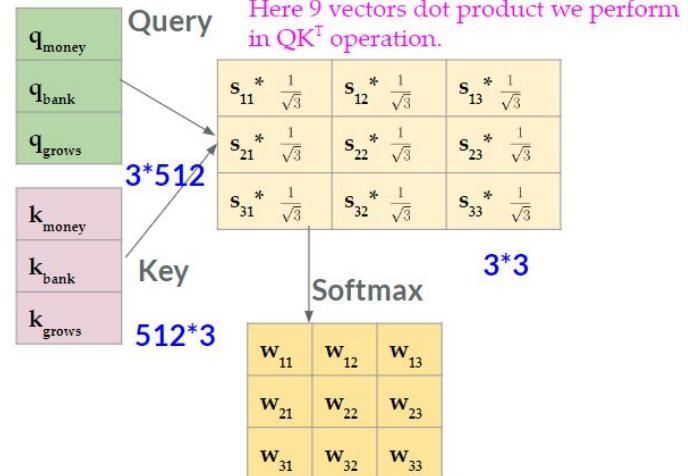
$$q_{\text{money}} \cdot k_{\text{grows}} \rightarrow [a].[d] \rightarrow \text{scalar}$$

These three are sample scalar values for given vector dot products.

If vector will change scalar will change.

So we calculate the variance in 1D as **Var (x)**

Where x is random variable representing all scalar values above.



Scaled dot product Attention

What should be the scaling factor?

Consider all vectors below is of 2 dimension

$$q_{\text{money}} \cdot k_{\text{money}} \rightarrow [a \ b].[c \ d] \rightarrow \text{scalar}$$

$$q_{\text{money}} \cdot k_{\text{bank}} \rightarrow [a \ b].[e \ f] \rightarrow \text{scalar}$$

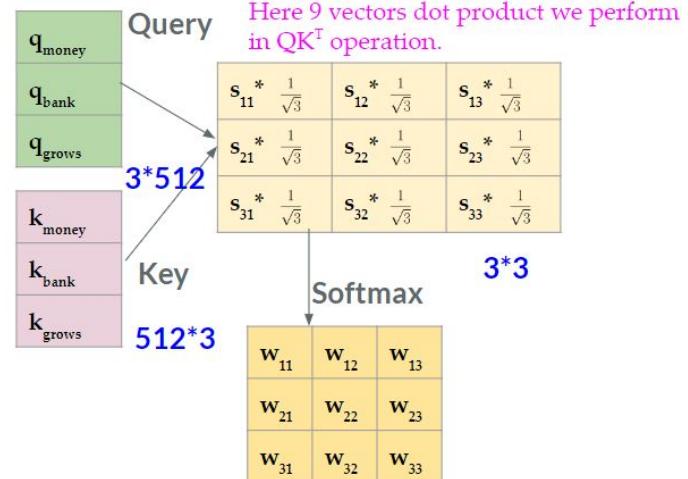
$$q_{\text{money}} \cdot k_{\text{grows}} \rightarrow [a \ b].[g \ h] \rightarrow \text{scalar}$$

These three are sample scalar values for given vector dot products.

If vector will change scalar will change.

So we calculate the variance in 2D as $\text{Var}(y) \approx 2 * \text{Var}(x)$

Where x is random variable representing all scalar values above.



Scaled dot product Attention

What should be the scaling factor?

Consider all vectors below is of 3 dimension

$$q_{\text{money}} \cdot k_{\text{money}} \rightarrow [a \ b \ c].[c \ d \ e] \rightarrow \text{scalar}$$

$$q_{\text{money}} \cdot k_{\text{bank}} \rightarrow [a \ b \ c].[f \ g \ h] \rightarrow \text{scalar}$$

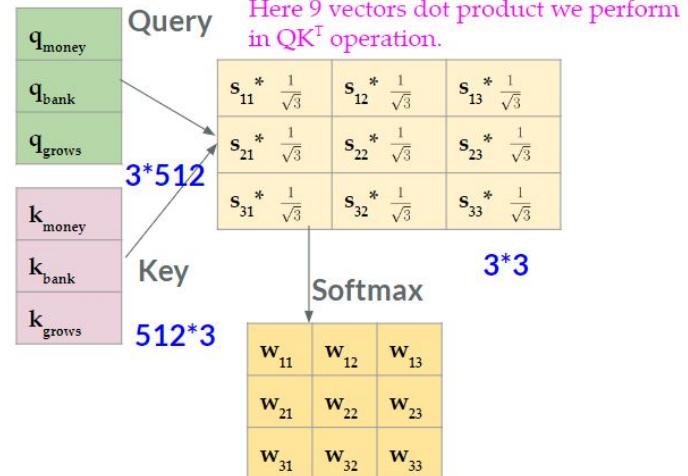
$$q_{\text{money}} \cdot k_{\text{grows}} \rightarrow [a \ b \ c].[i \ j \ k] \rightarrow \text{scalar}$$

These three are sample scalar values for given vector dot products.

If vector will change scalar will change.

So we calculate the variance in 2D as $\text{Var}(z) \approx 3 * \text{Var}(x)$

Where x is random variable representing all scalar values above.



Scaled dot product Attention

What should be the scaling factor?

Consider all vectors below is of d dimension

$$q_{\text{money}} \cdot k_{\text{money}} \rightarrow [a \ b \ c..].[c \ d \ e..] \rightarrow \text{scalar}$$

$$q_{\text{money}} \cdot k_{\text{bank}} \rightarrow [a \ b \ c..].[f \ g \ h..] \rightarrow \text{scalar}$$

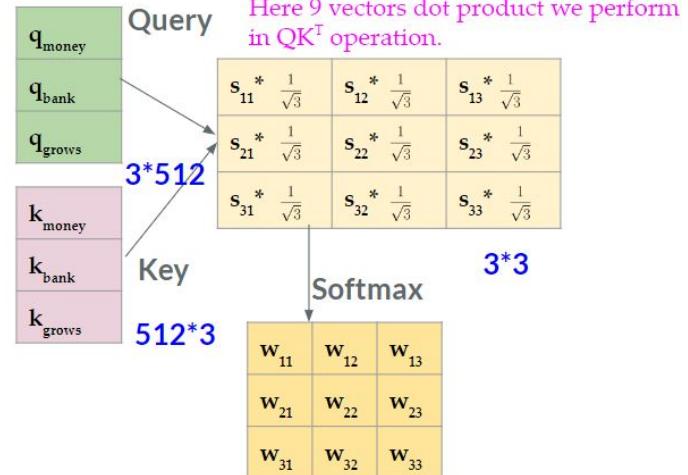
$$q_{\text{money}} \cdot k_{\text{grows}} \rightarrow [a \ b \ c..].[i \ j \ k..] \rightarrow \text{scalar}$$

These three are sample scalar values for given vector dot products.

If vector will change scalar will change.

So we calculate the variance in dD as $\text{Var}(d) \approx d * \text{Var}(x)$

Where x is random variable representing all scalar values above.



Scaled dot product Attention

What should be the scaling factor?

We understood that there is linear relationship between dimension and variance.

$$1D \rightarrow \text{Var}(x)$$

$$2D \rightarrow 2 * \text{Var}(x)$$

$$3D \rightarrow 3 * \text{Var}(x)$$

$$dD \rightarrow d * \text{Var}(x)$$

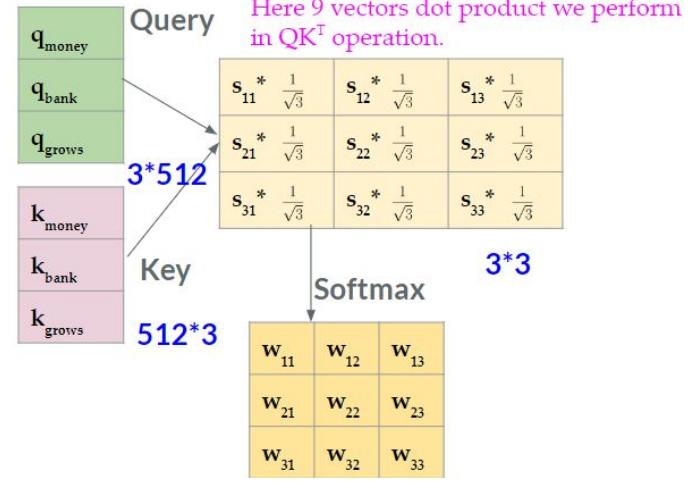
Ideally it should have been as follows:

$$1D \rightarrow \text{Var}(x)$$

$$2D \rightarrow \text{Var}(x)$$

$$3D \rightarrow \text{Var}(x)$$

$$dD \rightarrow \text{Var}(x)$$



Therefore, all the numbers in $\mathbf{Q}\mathbf{K}^T$ matrix should be divided by a number such that variance of the matrix will be same and it will be independent of dimension.

Scaled dot product Attention

What should be the scaling factor?

Consider for a random variable X , variance is $\text{Var}(x)$

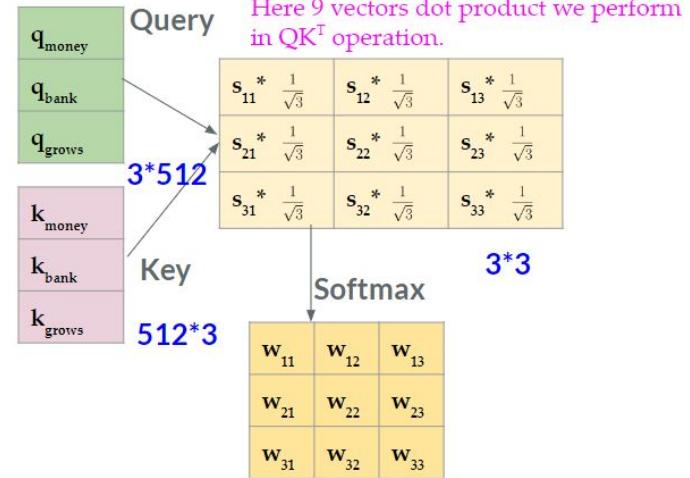
Now we create a new random variable Y by scaling X with constant c , such that $Y = cX$

Then, $\text{Var}(Y) = c^2 \text{Var}(x)$

We can also write it as:

$$X = \text{Var}(x)$$

$$cX = c^2 \text{Var}(x)$$



Here 9 vectors dot product we perform in QK^T operation.

Therefore, all the numbers in QK^T matrix should be divided by a number such that variance of the matrix will be same and it will be independent of dimension.

Scaled dot product Attention

What should be the scaling factor?

We understood that there is linear relationship between dimension and variance.

$$1D \rightarrow X \rightarrow \text{Var}(x)$$

$$2D \rightarrow Y \rightarrow \text{Var}(y) = 2 * \text{Var}(x)$$

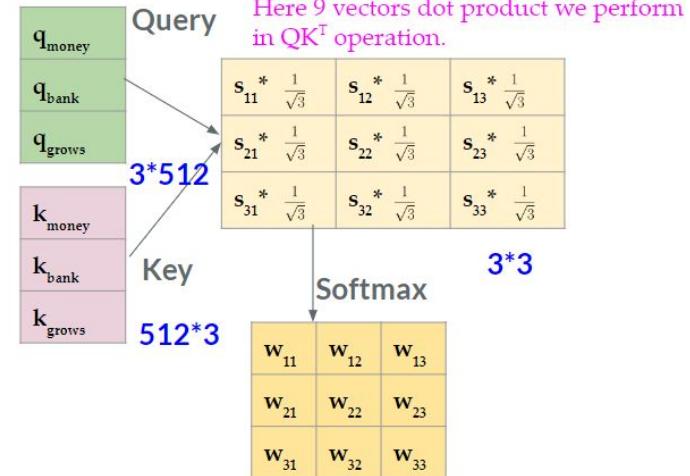
$$Y/\sqrt{2} = \text{Var}(y)/2 = (2 * \text{Var}(x))/2 = \text{Var}(x)$$

$$3D \rightarrow Z \rightarrow \text{Var}(z) = 3 * \text{Var}(x)$$

$$Z/\sqrt{3} = \text{Var}(z)/3 = (3 * \text{Var}(x))/3 = \text{Var}(x)$$

$$dD \rightarrow I \rightarrow \text{Var}(i) = d * \text{Var}(x)$$

$$I/\sqrt{d} = \text{Var}(i)/d = (d * \text{Var}(x))/d = \text{Var}(x)$$

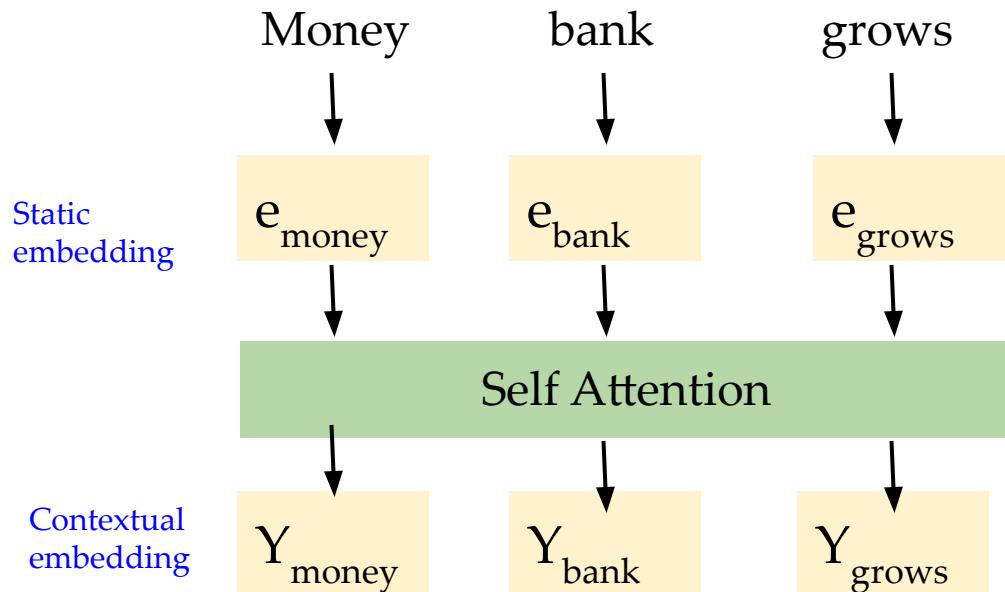


$$X = \text{Var}(x)$$
$$cX = c^2 \text{Var}(x)$$

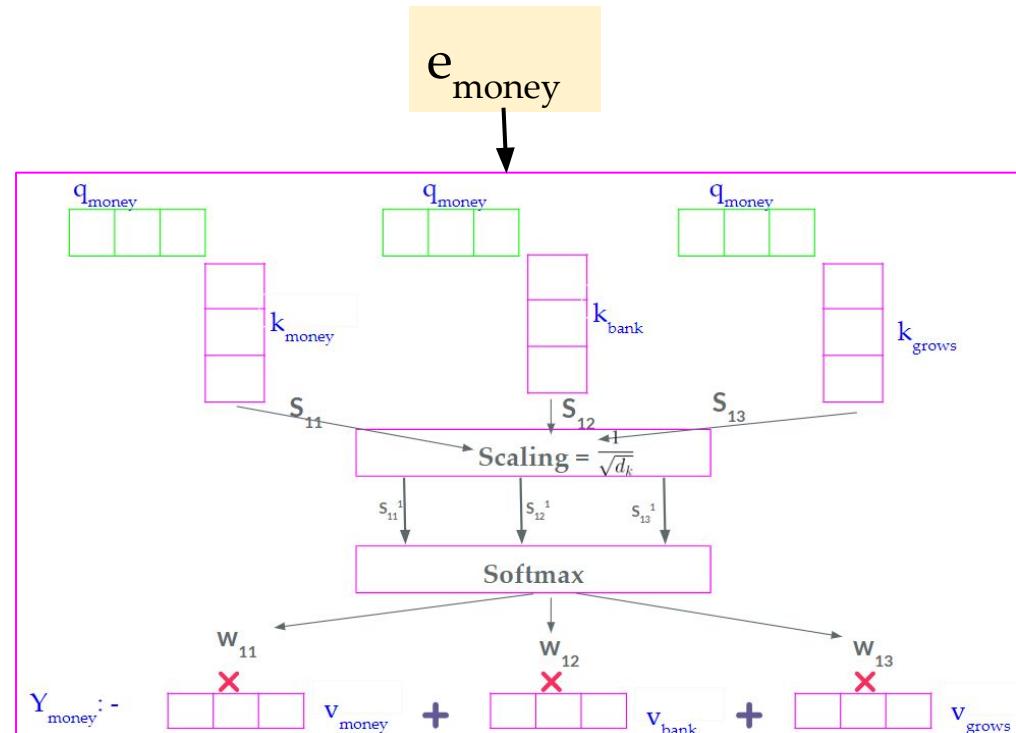
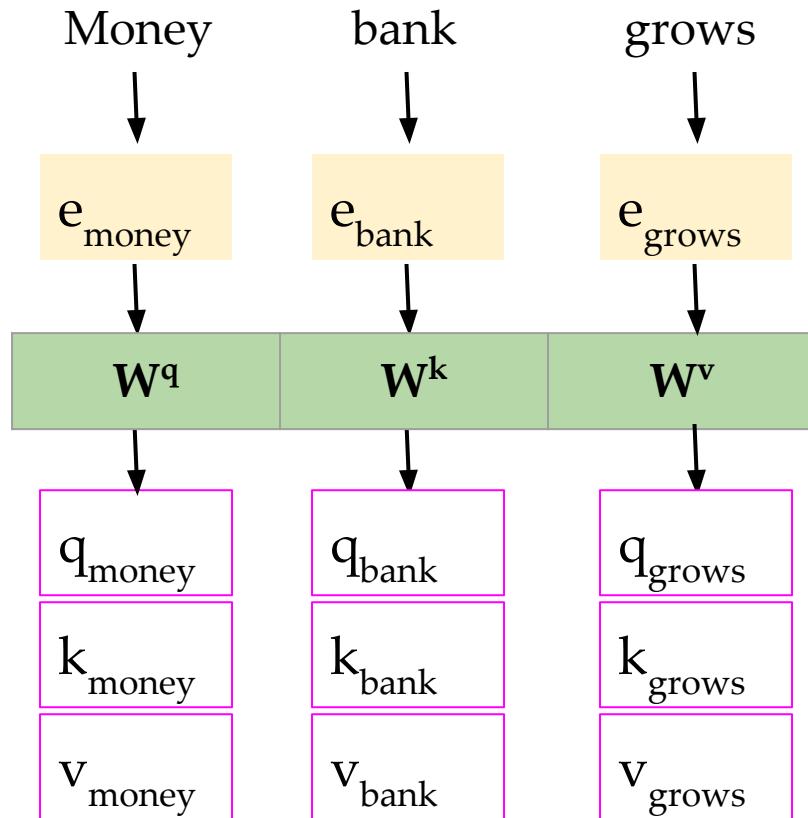
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Self Attention: Recap

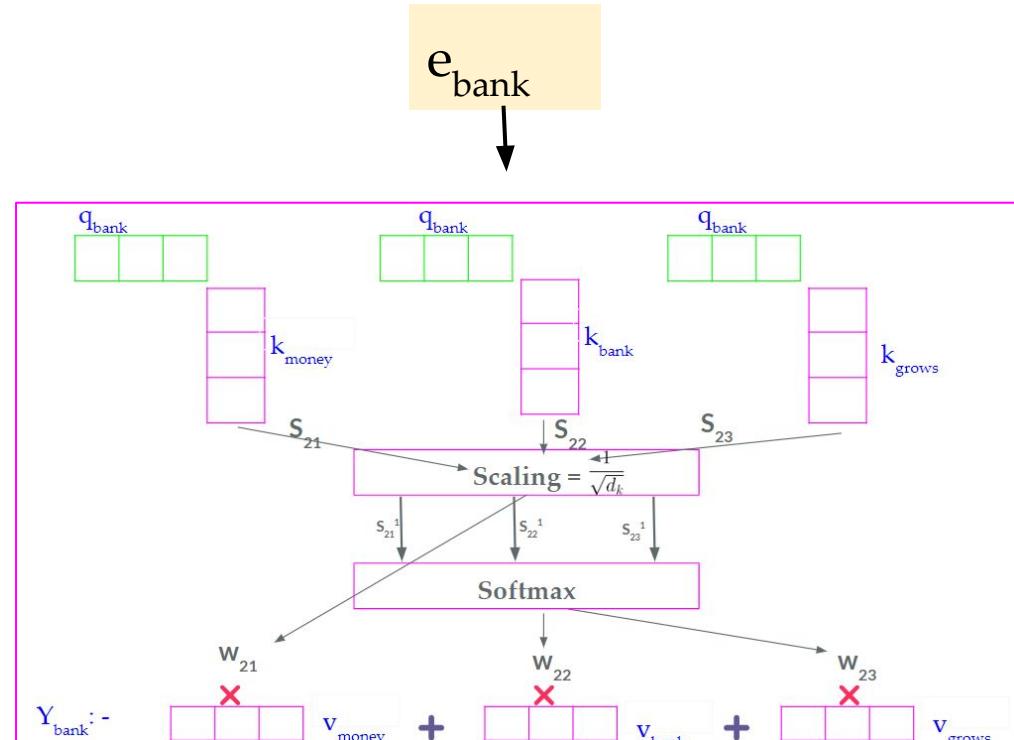
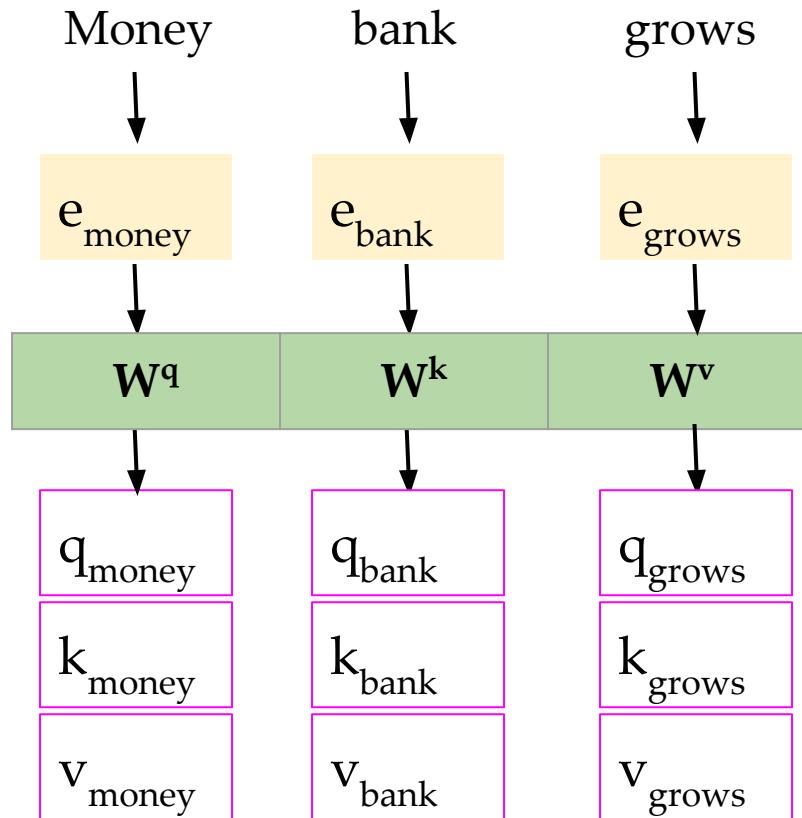
Self attention is a mechanism which is used to generate the contextual embedding.



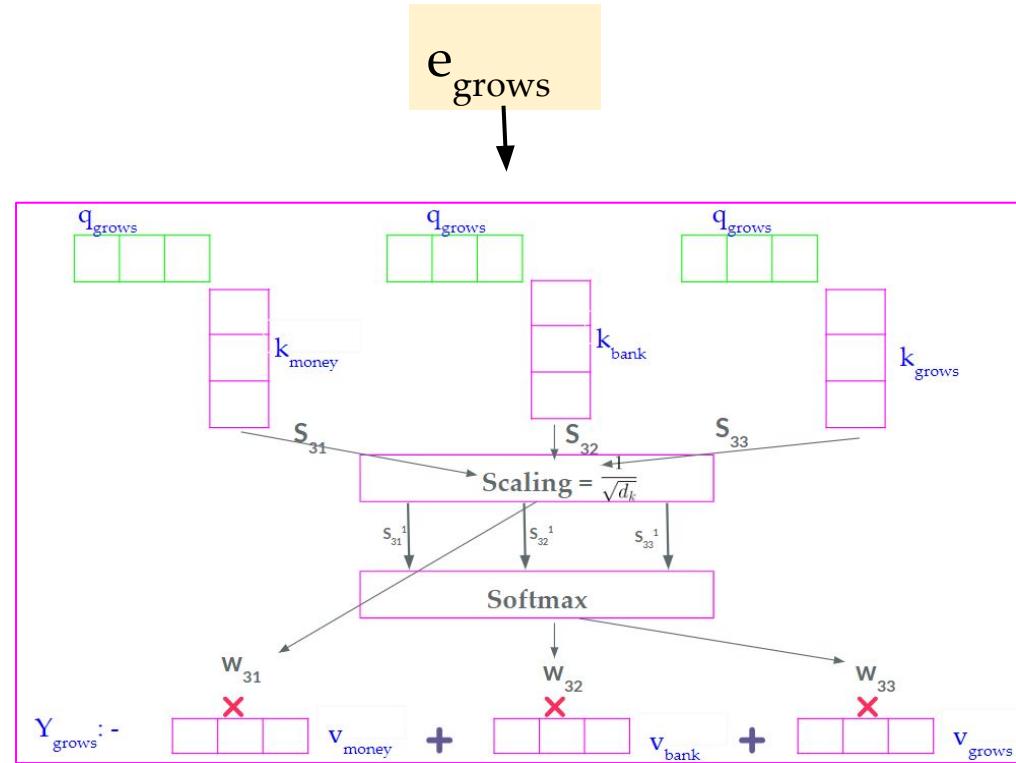
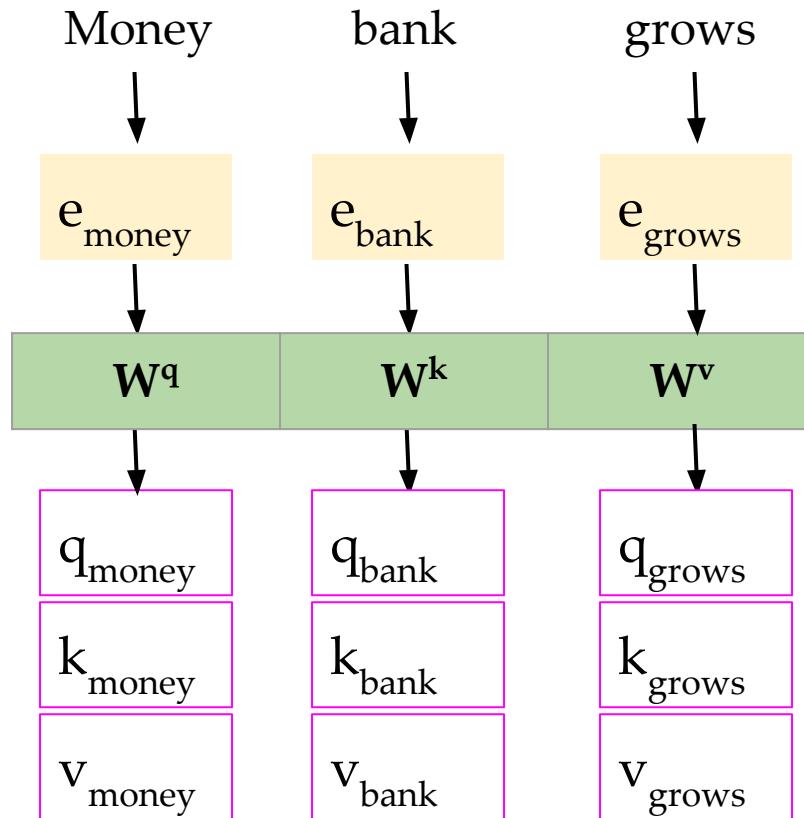
Self Attention: Recap



Self Attention: Recap



Self Attention: Recap



The problem with Self Attention

Consider the sentence:

The man saw the astronomer with a telescope

There are two possibilities here:

1: The man saw the astronomer with a telescope

2. The man saw the astronomer with a telescope

- The Self attention can capture only single perspective in a given sentence.
- But, often in NLP a sentence has multiple perspectives which machine has to understand.
 - Ex: Summarizing text in multiple ways.
 - Writing answers in different ways.

Multi-head Self Attention

Consider the sentence:

The man saw the astronomer with a telescope

There are two possibilities here:

1: The man saw the astronomer with a telescope

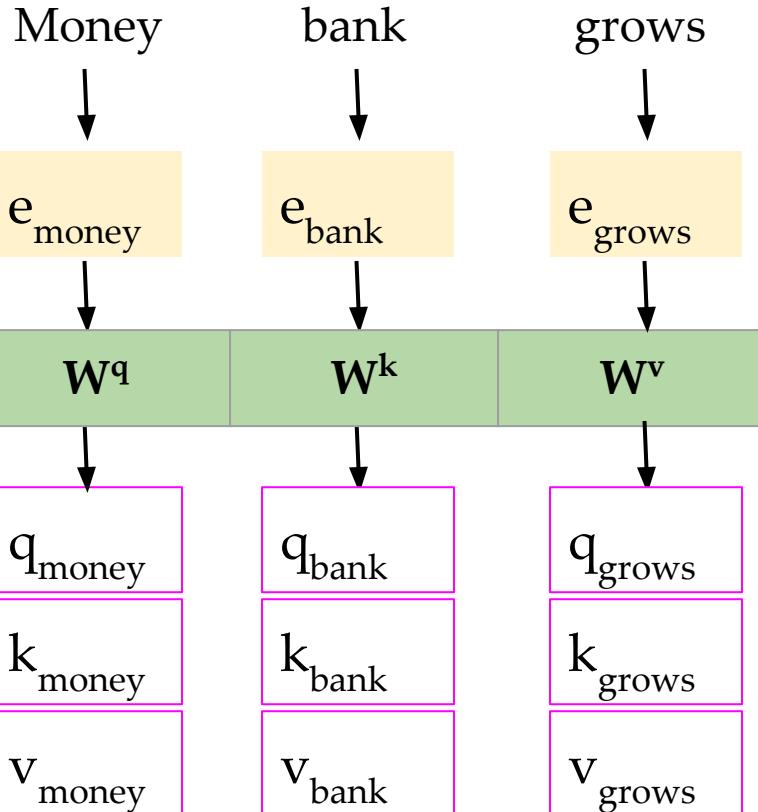
2. The man saw the astronomer with a telescope

- If there are two perspective of the given sentence, can we use two self attention?
- Where, 1st self attention will capture the context between
 $\text{Saw} \rightarrow \text{astronomer}$
 $\text{Saw} \rightarrow \text{telescope}$
- 2nd attention will capture the context between
 $\text{Saw} \rightarrow \text{astronomer}$
 $\text{Astronomer} \rightarrow \text{telescope}$

This setting of self attention with multi-head is known as
Multi-head attention.

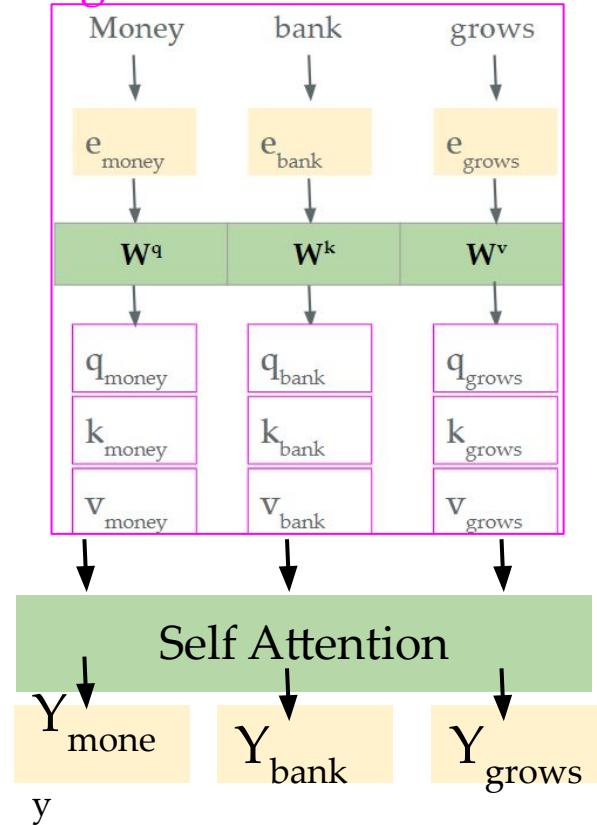
Multi-head Self Attention

Single-head Self attention: one set of W^q , W^k , and W^v



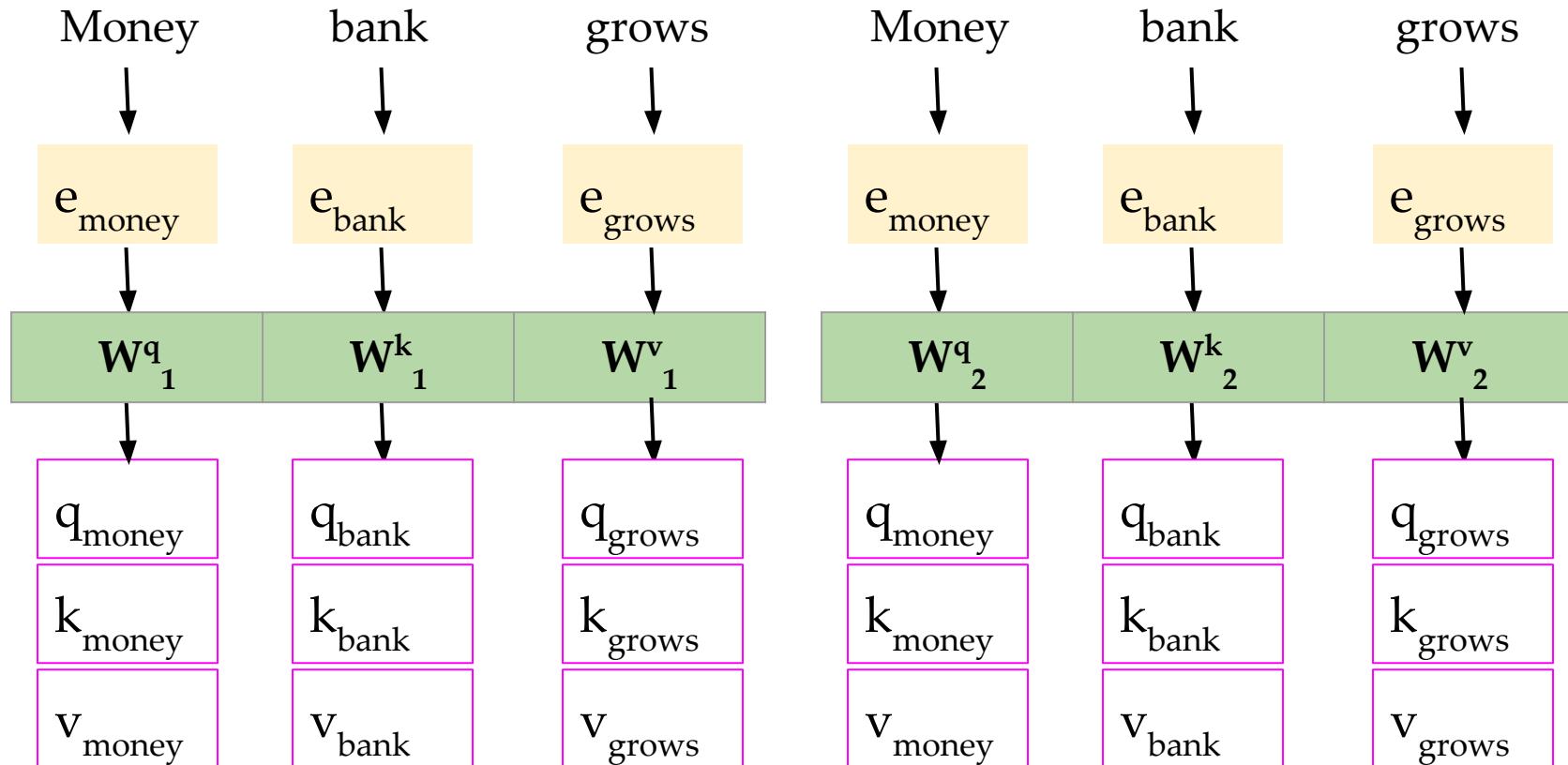
Multi-head Self Attention

Single-head Self attention: one set of W^q , W^k , and W^v



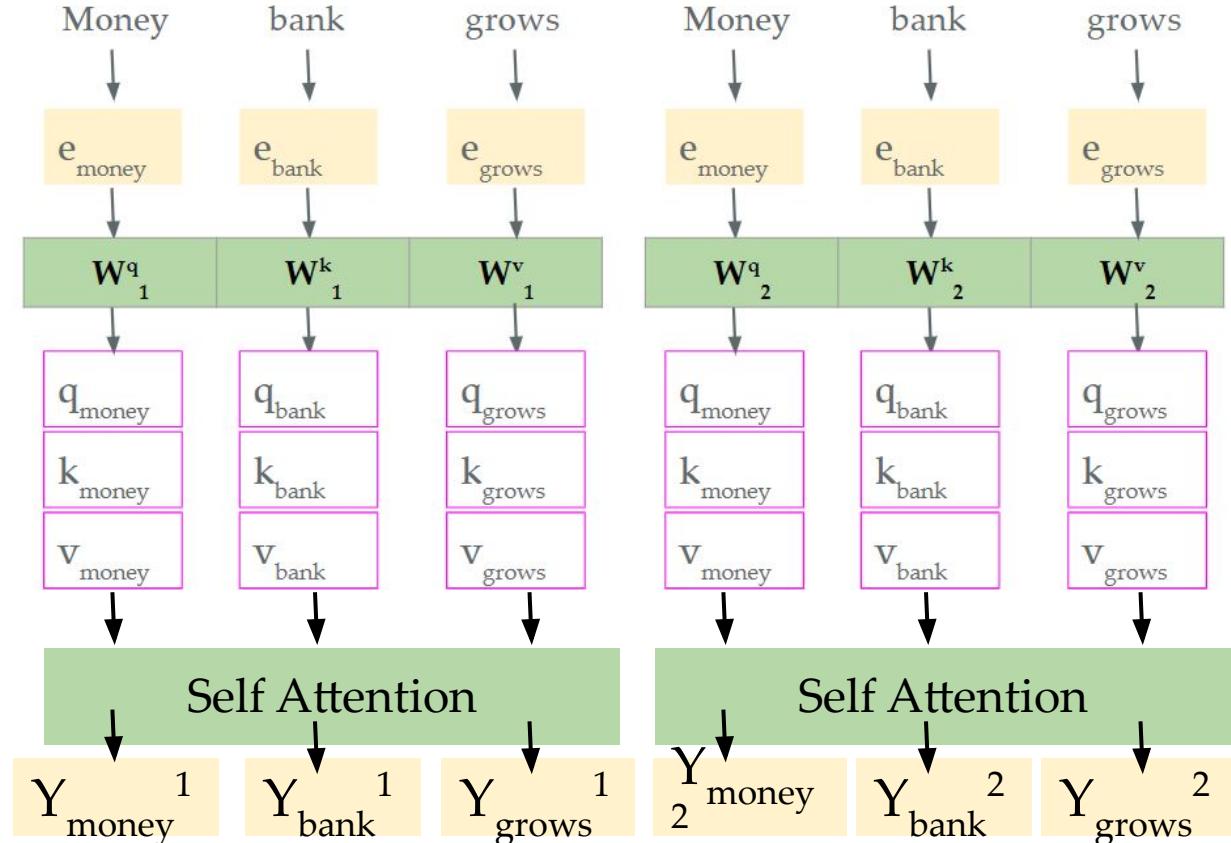
Multi-head Self Attention

Double-head Self attention: two set of W^q , W^k , and W^v

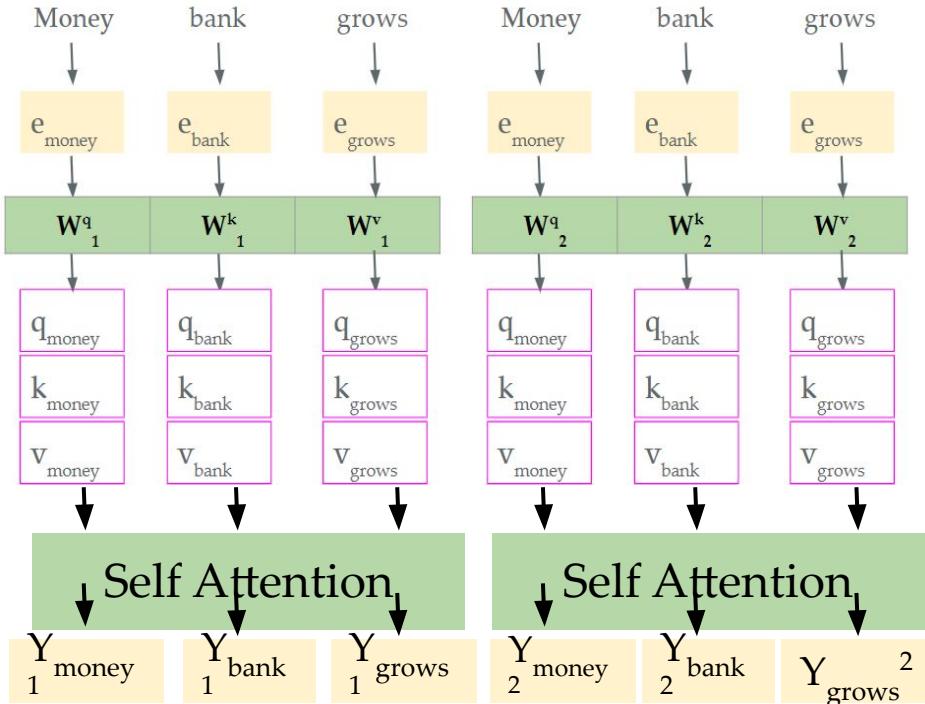


Multi-head Self Attention

Double-head Self attention: two set of W^q , W^k , and W^v

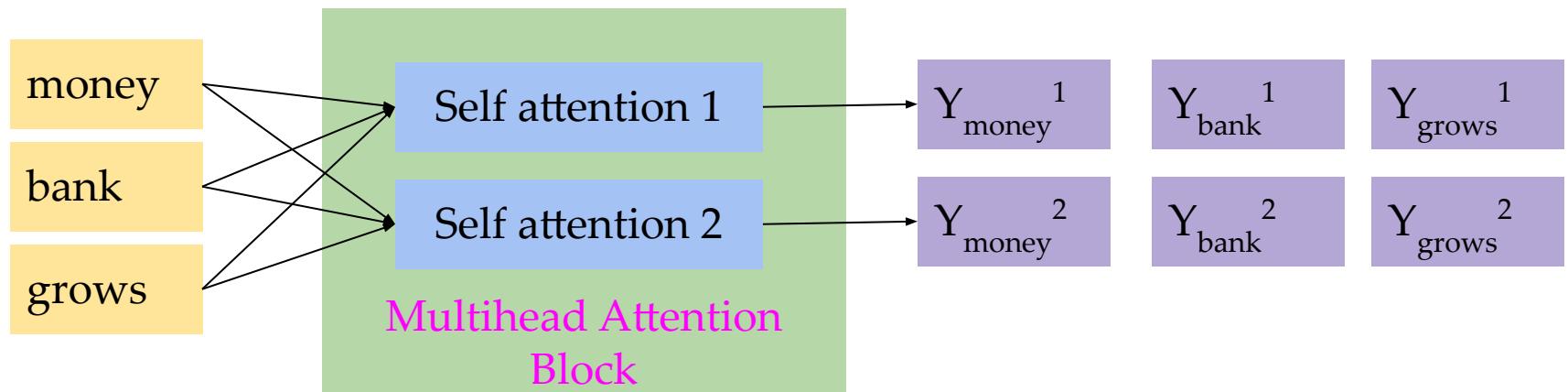


Multi-head Self Attention

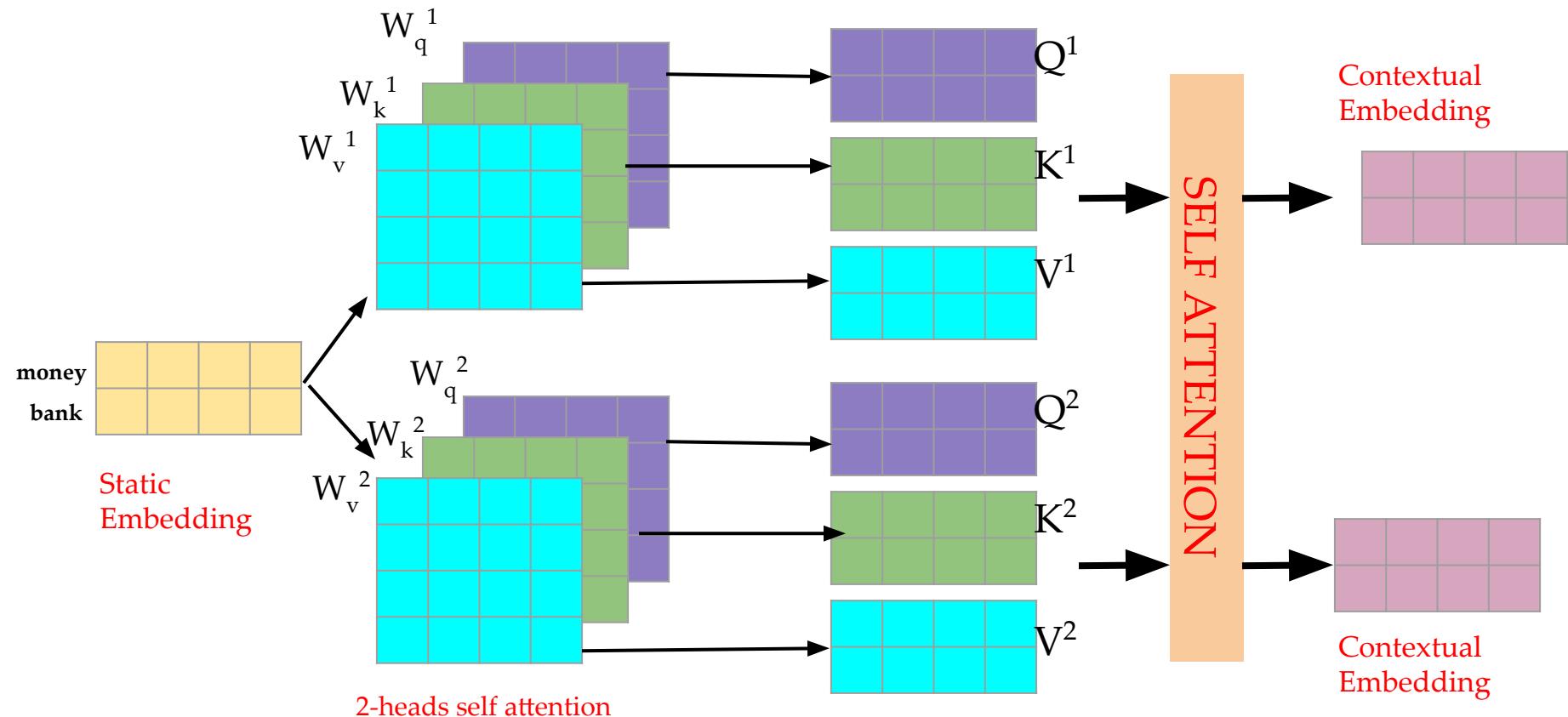


No. of heads	No. of set of matrices	No. of contextual embedding generated
1	1	1
2	2	2
..
8	8	8

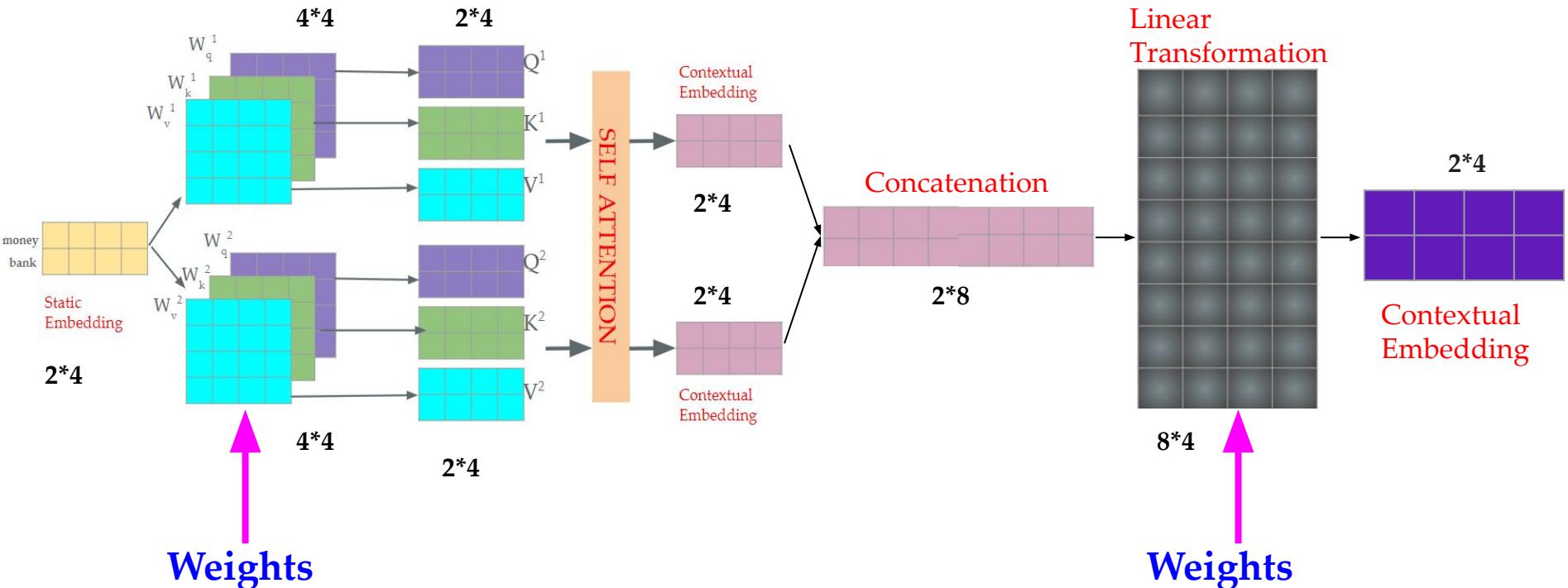
Multi-head Self Attention



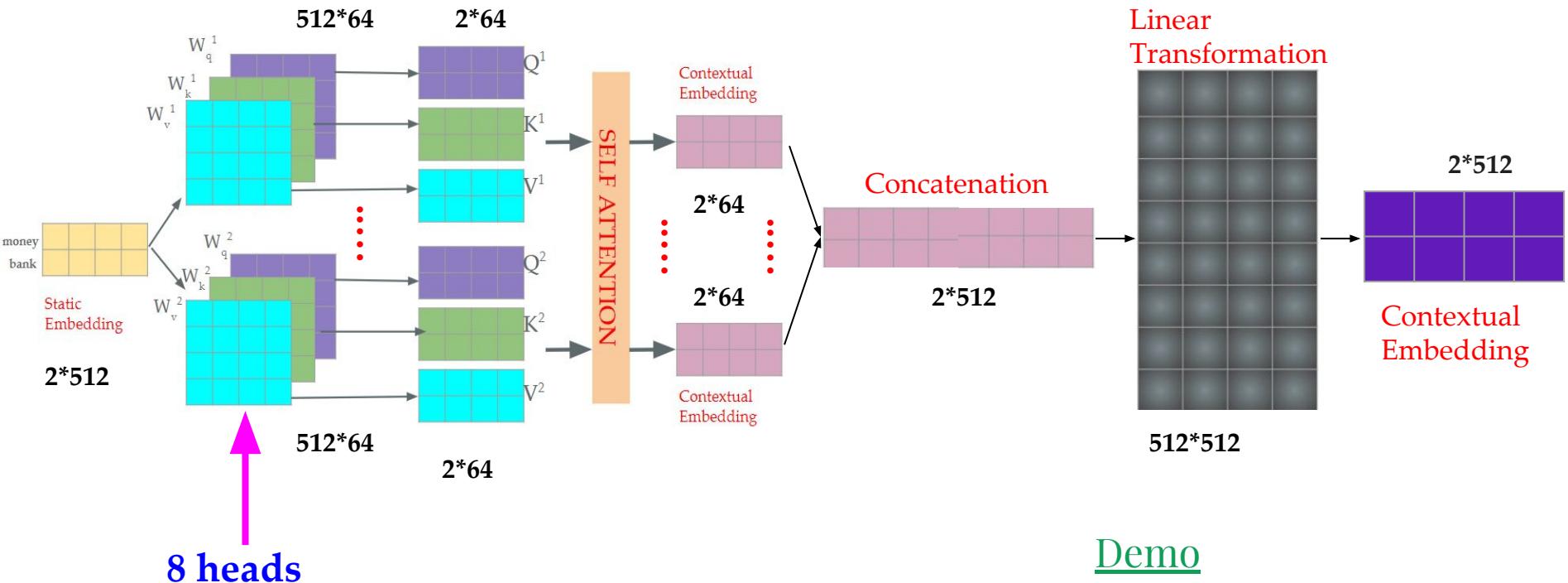
How is multi-head attention applied?



How is multi-head attention applied?



Multi-head attention in Transformer



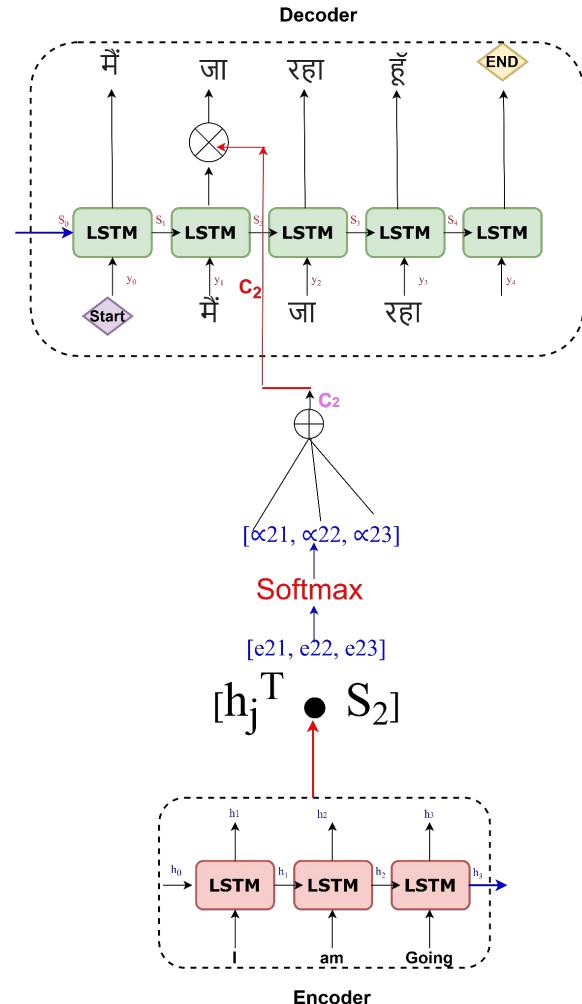
Why is self attention called as “Self”?

Luong Attention:

$$C_i = \sum_{j=1}^N h_j$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{i=1}^N \exp(e_{ij})}$$

$$e_{ij} = [S_i^T \quad h_j]$$



Why is self attention called as “Self”?

Luong Attention:

$$C_i = \sum_{j=1}^{\infty} h_j \exp(e_{ij})$$

$$\propto_{ij} = \frac{N}{\sum_{i=1}^{\infty} \exp(e_{ij})}$$

$$e_{ij} = [S_i^T \quad \square \quad h_j]$$

Money bank grows

q_{money} q_{bank} q_{grows}

s_{11} s_{12} s_{13}

Money bank grows

k_{money} k_{bank} k_{grows}

Similarities:

S_i = query

h_j = key

h_j = value

$$y_{\text{money}}: w_{11}v_{\text{money}} + w_{12}v_{\text{bank}} + w_{13}v_{\text{grows}}$$

$$\text{Here, } C_i \approx y_{\text{money}}$$

$$\propto_{ij} \approx w_{11}/w_{12}/w_{13}$$

$$w_{11}/w_{12}/w_{13} = \text{Softmax } (e_{ij} \approx S_{11}/S_{12}/S_{13})$$

$$S_{11} = q_{\text{money}} \cdot k_{\text{money}}, \quad S_{12} = q_{\text{money}} \cdot k_{\text{bank}}, \quad S_{13} = q_{\text{money}} \cdot k_{\text{grows}}$$

Transformer Encoder: Word Order

- Word order: order in which word appears in a text makes the context

I am going Vs going am I

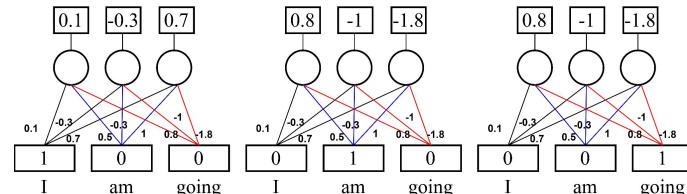
Both the above sentences are different in their meaning.

However, their word embeddings remains same.

I am going: $[[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]$

going am I: $[[0.8, -1, -1.8], [0.5, -0.3, 1], [0.1, -0.3, -0.3]]$

How to preserve word order?



Transformer Encoder: Positional Encoding

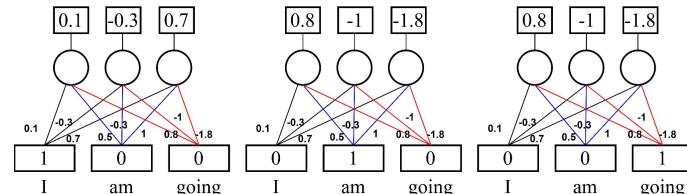
I am going: [[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]

Adding **absolute** position vector to word embedding:

$$e_p^0: [0.1, -0.3, 0.7] + [0, 0, 0] \rightarrow [0.1, -0.3, 0.7]$$

$$e_p^1: [0.5, -0.3, 1] + [1, 1, 1] \rightarrow [1.5, 0.7, 2]$$

$$e_p^2: [0.8, -1, -1.8] + [2, 2, 2] \rightarrow [2.8, 1, 0.2]$$



Transformer Encoder: Positional Encoding

I am going: [[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]

Adding **absolute** position vector to word embedding:

$$e_p^0: [0.1, -0.3, 0.7] + [0, 0, 0] \rightarrow [0.1, -0.3, 0.7]$$

$$e_p^1: [0.5, -0.3, 1] + [1, 1, 1] \rightarrow [1.5, 0.7, 2]$$

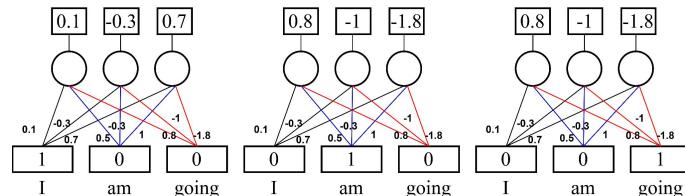
$$e_p^2: [0.8, -1, -1.8] + [2, 2, 2] \rightarrow [2.8, 1, 0.2]$$

.....

$$e_p^{30}: [0.8, -1, -1.8] + [30, 30, 30] \rightarrow [30.8, 29, 28.2]$$

Limitation:

- **Unbounded**: Words with higher position no. get high weightage.
- **Discrete**: NN prefer smooth changes.
- **Absolute position**: relative positioning cannot be captured.



Transformer Encoder: Positional Encoding

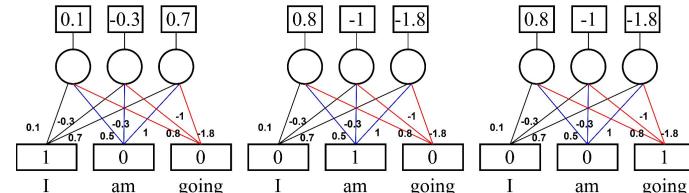
I am going: [[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]

Adding fraction position vector (fraction to total length = 1/N-1) to word embedding: to normalize the positional vector range between 0 to 1.

$$e_p^0: [0.1, -0.3, 0.7] + [0, 0, 0] \rightarrow [0.1, -0.3, 0.7] \quad (0^{1/2} = 0)$$

$$e_p^1: [0.5, -0.3, 1] + [0.5, 0.5, 0.5] \rightarrow [1, 0.2, 1.5] \quad (1^{1/2} = 0.5)$$

$$e_p^2: [0.8, -1, -1.8] + [1, 1, 1] \rightarrow [1.8, 0, -0.8] \quad (2^{1/2} = 1)$$



Transformer Encoder: Positional Encoding

I am going now: $[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8], [0.7, 0.6, 1]$

Adding fraction position vector (fraction to total length = $1/N-1$) to word embedding: to normalize the positional vector range between 0 to 1.

$$e_p^0: [0.1, -0.3, 0.7] + [0, 0, 0] \rightarrow [0.1, -0.3, 0.7] \quad (0^{*1/3} = 0)$$

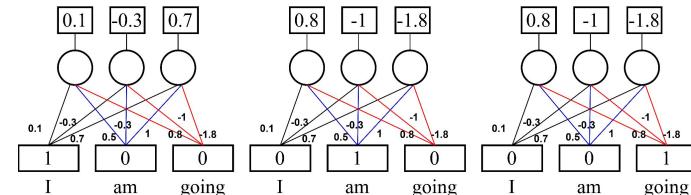
$$e_p^1: [0.5, -0.3, 1] + [0.3, 0.3, 0.3] \rightarrow [0.8, 0, 1.3] \quad (1^{*1/3} = 0.3)$$

$$e_p^2: [0.8, -1, -1.8] + [0.6, 0.6, 0.6] \rightarrow [1.4, -0.4, -1.2] \quad (2^{*1/3} = 0.6)$$

$$e_p^3: [0.7, 0.6, 1] + [1, 1, 1] \rightarrow [1.7, 1.6, 2] \quad (3^{*1/3} = 1)$$

Limitation:

- Position vector changes w.r.t. sentence length.
- Position vector should remains same irrespective of no. of words in a sentence.



Transformer Encoder: Positional Encoding

I am going: $[[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]$

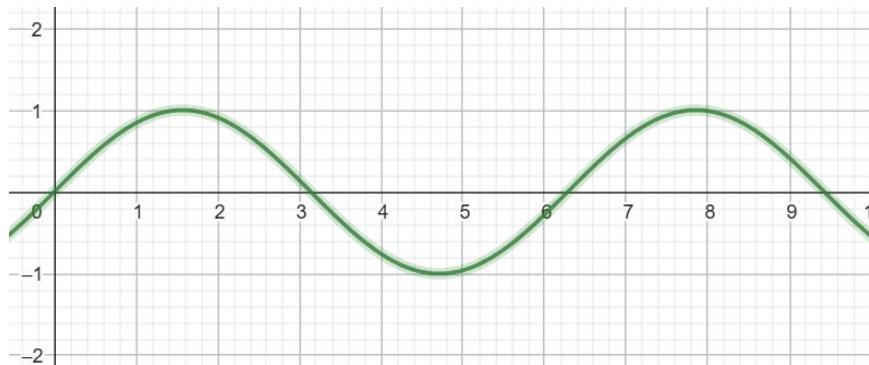
Adding wave frequency as position vector to word embedding

$$e_0^p: [0.1, -0.3, 0.7] + p_0$$

Let's calculate the 1st position embedding as an example.

Advantages:

- **Bounded:** -1 to +1.
- **Periodic:** For every x there is value of y .



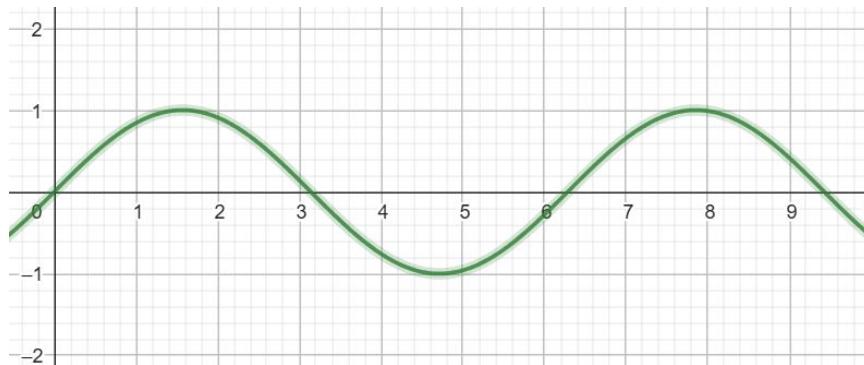
Transformer Encoder: Positional Encoding

I am going: $[[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]$

Adding wave frequency as position vector to word embedding

$$e_p^0: [0.1, -0.3, 0.7] + p_0$$

Let's calculate the 1st position embedding as an example.



Advantages:

- **Bounded:** -1 to +1.
- **Periodic:** For every x there is value of y .

We can use Sin function to get the position.

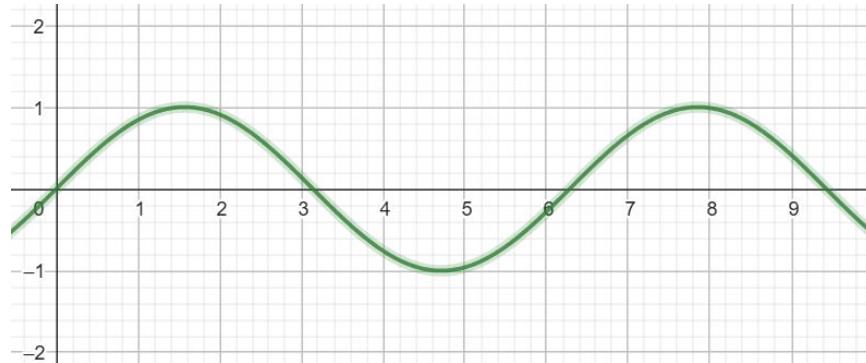
Transformer Encoder: Positional Encoding

I am going: $[[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]$

Adding wave frequency as position vector to word embedding

$$e_0^p: [0.1, -0.3, 0.7] + p_0$$

Let's calculate the 1st position embedding as an example.



$$\text{Sin}(1) = 0.84$$

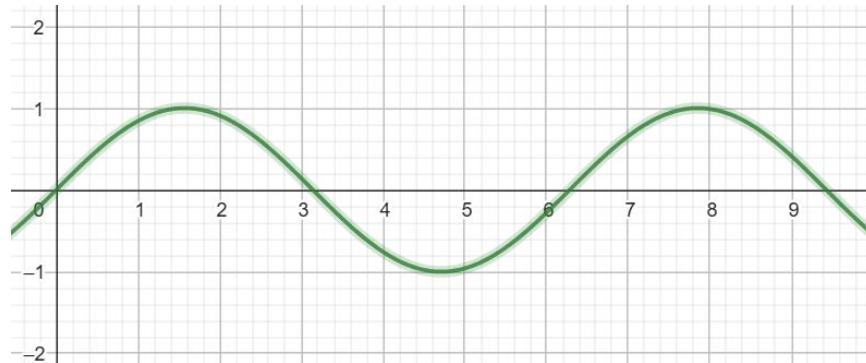
Transformer Encoder: Positional Encoding

I am going: [[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]

Adding wave frequency as position vector to word embedding

e_0^p : [0.1, -0.3, 0.7, 0.84]

Let's calculate the position embedding as an example.



$$\text{Sin}(1) = 0.84$$

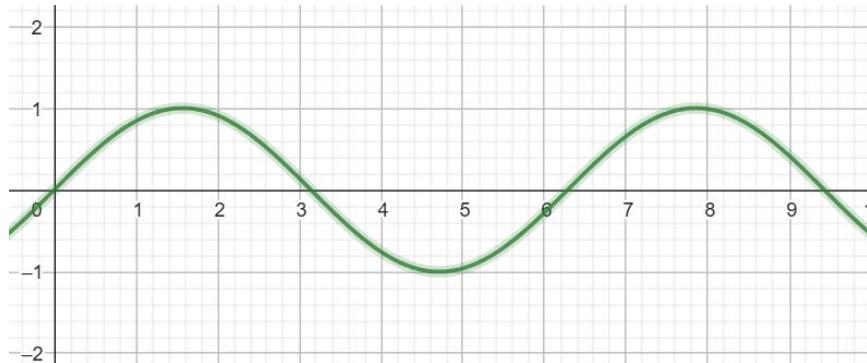
Transformer Encoder: Positional Encoding

I am going: [[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]

Adding wave frequency as position vector to word embedding

e_p^0 : [0.1, -0.3, 0.7, 0.84], e_p^1 : [0.5, -0.3, 1, 0.90]

Let's calculate the position embedding as an example.



$$\begin{aligned}\text{Sin}(1) &= 0.84 \\ \text{Sin}(2) &= 0.90\end{aligned}$$

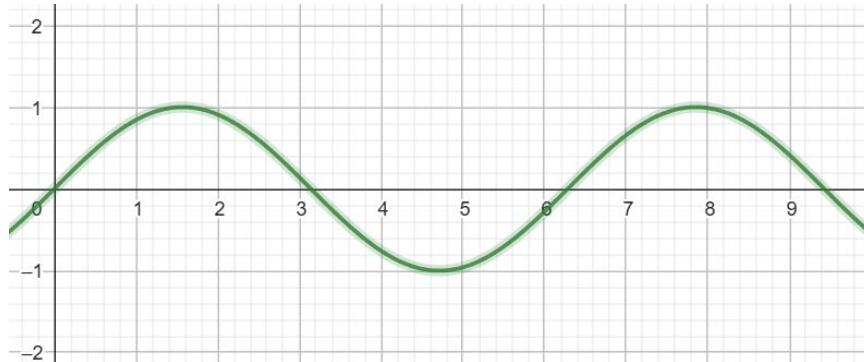
Transformer Encoder: Positional Encoding

I am going: [[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]

Adding wave frequency as position vector to word embedding

e_p^0 : [0.1, -0.3, 0.7, 0.84], e_p^1 : [0.5, -0.3, 1, 0.90], e_p^3 : [0.8, -1, -1.8, 0.14]

Let's calculate the position embedding as an example.



$$\text{Sin}(1) = 0.84$$

$$\text{Sin}(2) = 0.90$$

$$\text{Sin}(3) = 0.14$$

Transformer Encoder: Positional Encoding

I am going: $[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]$

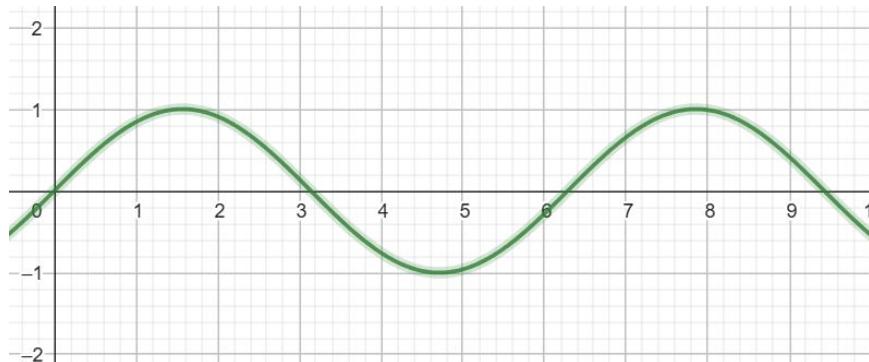
Limitation:

- **Periodic nature:** of Sin curve will return same position value for more than one word.
- May be position encoding of 2nd word and 44th word same.
- Then, it is a big problem for NN to differentiate the words.

Adding wave frequency as position vector to word embedding

$e_p^0: [0.1, -0.3, 0.7, 0.84], e_p^1: [0.5, -0.3, 1, 0.90], e_p^3: [0.8, -1, -1.8, 0.14]$

Let's calculate the position embedding as an example.



$$\text{Sin}(1) = 0.84$$

$$\text{Sin}(2) = 0.90$$

$$\text{Sin}(3) = 0.14$$

Transformer Encoder: Positional Encoding

Limitation:

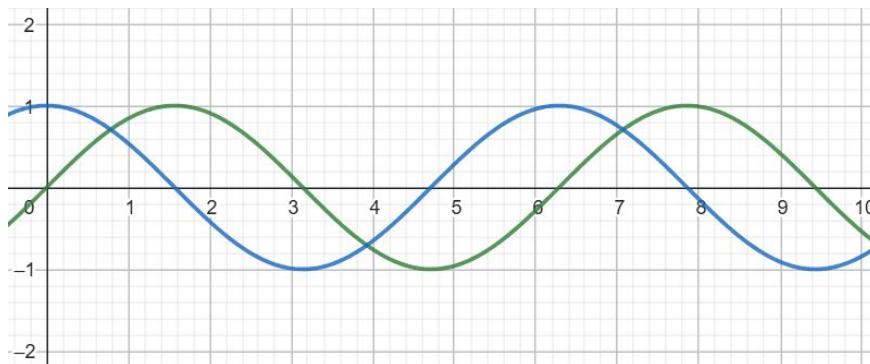
- Periodic nature: of Sin and Cos curve combination may also return same position value for more than one word.

I am going: [[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]

Adding wave frequency as position vector to word embedding

e_p^0 : [0.1, -0.3, 0.7, 0.84, 0.54], e_p^1 : [0.5, -0.3, 1, 0.90, -0.41], e_p^3 : [0.8, -1, -1.8, 0.14, -0.98]

Let's calculate the position embedding as a vector of Sin and Cos function.



$$\begin{array}{ll} \text{Sin}(1) = 0.84 & \text{Cos}(1) = 0.54 \\ \text{Sin}(2) = 0.90 & \text{Cos}(2) = -0.41 \\ \text{Sin}(3) = 0.14 & \text{Cos}(3) = -0.98 \end{array}$$

Transformer Encoder: Positional Encoding

I am going: $[[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]$

Adding wave frequency as position vector to word embedding

$$e_p^0: [0.1, -0.3, 0.7, \textcolor{red}{0.84}, 0.54, 0.47],$$

$$e_p^1: [0.5, -0.3, 1, \textcolor{red}{0.90}, -0.41, 0.84],$$

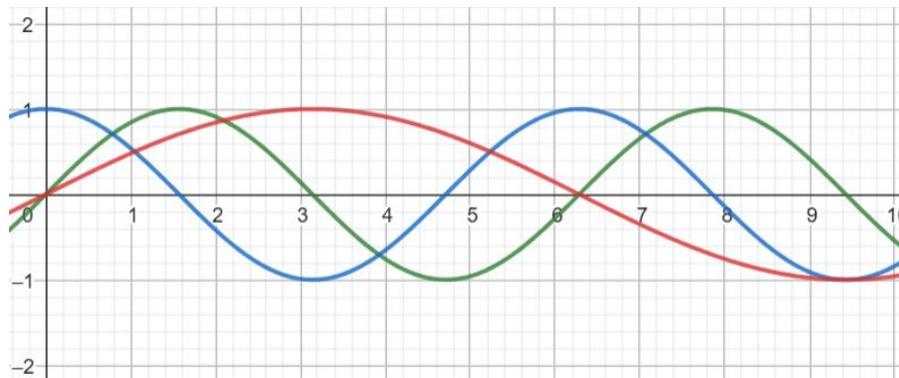
$$e_p^3: [0.8, -1, -1.8, \textcolor{red}{0.14}, -0.98, 0.99]$$

Limitation:

- Periodic nature: of Sin and Cos curve combination may also return same position value for more than one word.

Likewise, we can keep on increasing the curve to represent the position of a word uniquely.

Let's calculate the position embedding as a vector of Sin, Cos, and $\text{Sin}(\text{pos}/2)$ function.



$$\text{Sin}(1) = 0.84 \quad \text{Cos}(1) = 0.54$$

$$\text{Sin}(2) = 0.90 \quad \text{Cos}(2) = -0.41$$

$$\text{Sin}(3) = 0.14 \quad \text{Cos}(3) = -0.98$$

$$\text{Sin}(1/2) = 0.47$$

$$\text{Sin}(2/2) = 0.84$$

$$\text{Sin}(3/2) = 0.99$$

Transformer Encoder: Positional Encoding

I am going: $[[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]]$

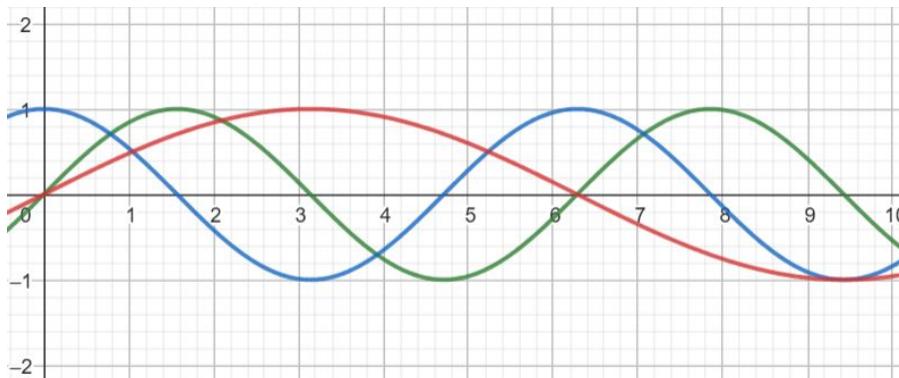
Adding wave frequency as position vector to word embedding

$$e_p^0: [0.1, -0.3, 0.7] + [0.84, 0.54, 0.47]$$

$$e_p^1: [0.5, -0.3, 1] + [0.90, -0.41, 0.84]$$

$$e_p^3: [0.8, -1, -1.8] + [0.14, -0.98, 0.99]$$

Let's calculate the position embedding as a vector of Sin, Cos, and $\text{Sin}(pos/2)$ function.



Advantage of vector addition over concatenation:

- In Concatenation, the size of word embedding gets double and complexity of network also gets double.
- In vector addition, we have less dimension and hence less complex network.

$$\text{Sin}(1) = 0.84 \quad \text{Cos}(1) = 0.54$$

$$\text{Sin}(2) = 0.90 \quad \text{Cos}(2) = -0.41$$

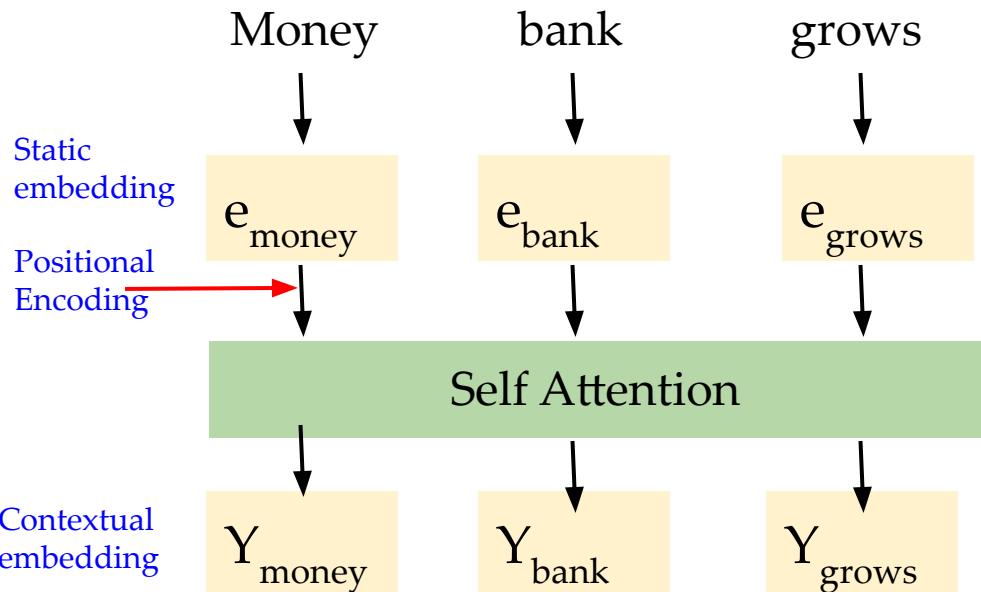
$$\text{Sin}(3) = 0.14 \quad \text{Cos}(3) = -0.98$$

$$\text{Sin}(1/2) = 0.47$$

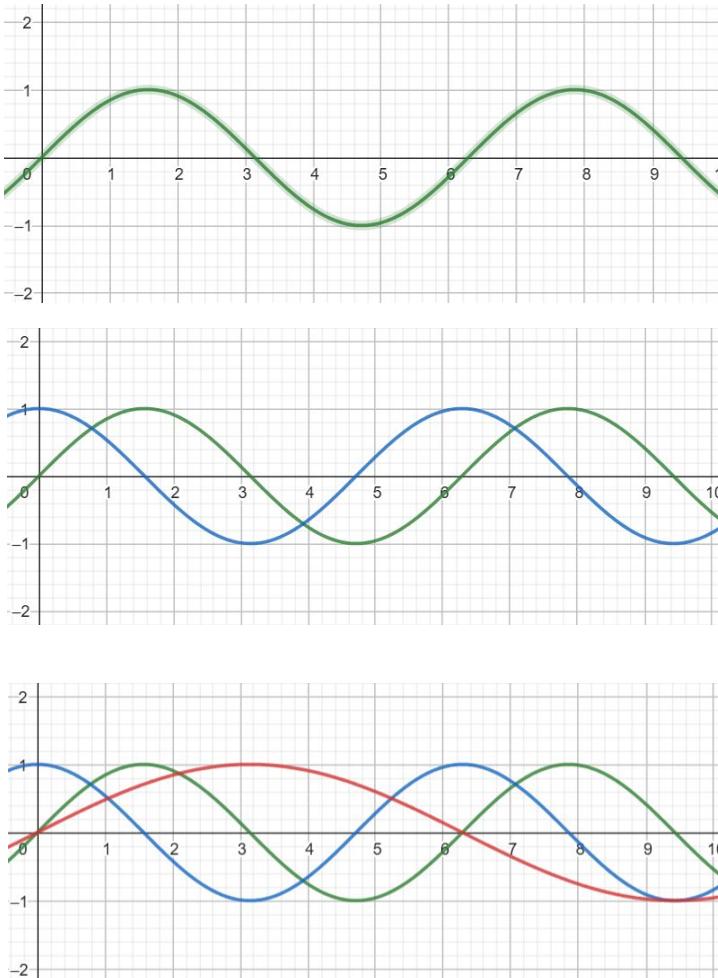
$$\text{Sin}(2/2) = 0.84$$

$$\text{Sin}(3/2) = 0.99$$

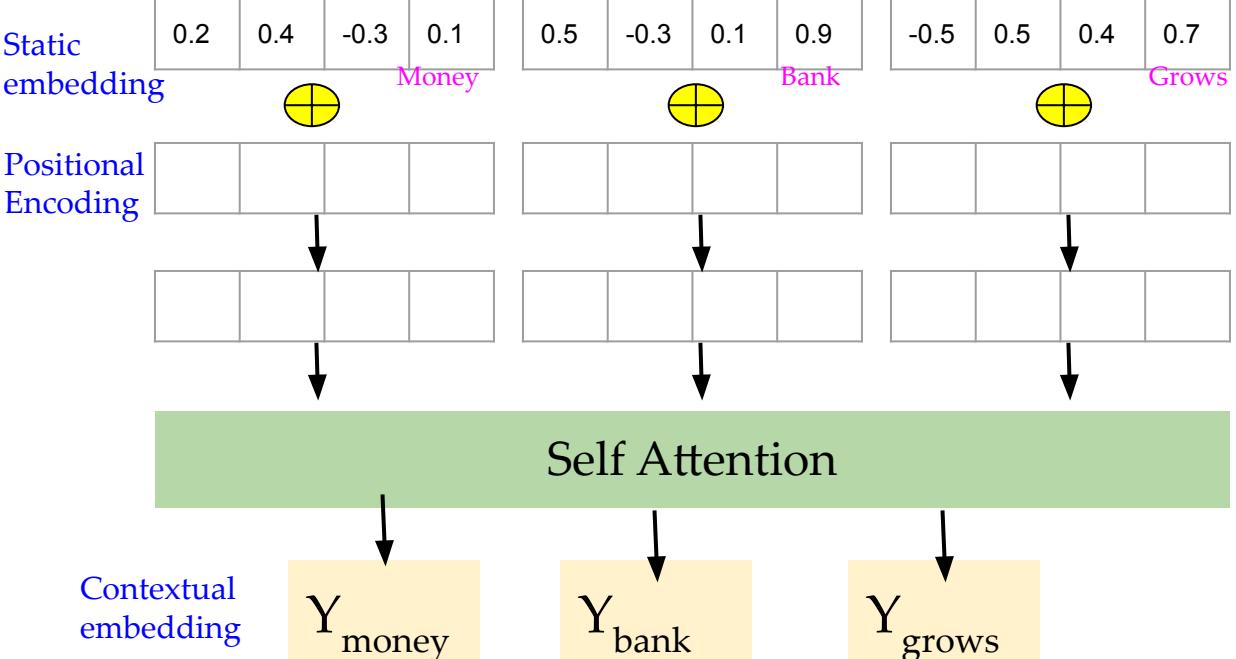
Transformer Encoder: Positional Encoding



- Positional embedding will be a vector
- Dimension of positional encoding vector = dimension of static embedding of a word
- We will take **pair of Sin and Cos** to determine the each dimension of position vector.

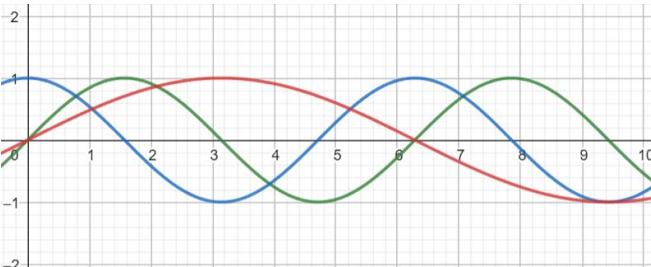
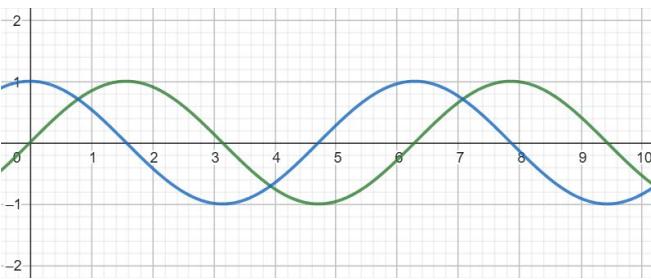
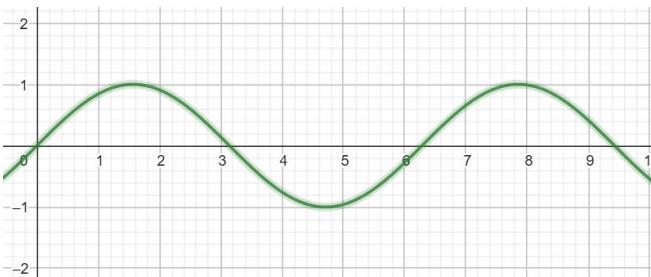


Transformer Encoder: Positional Encoding

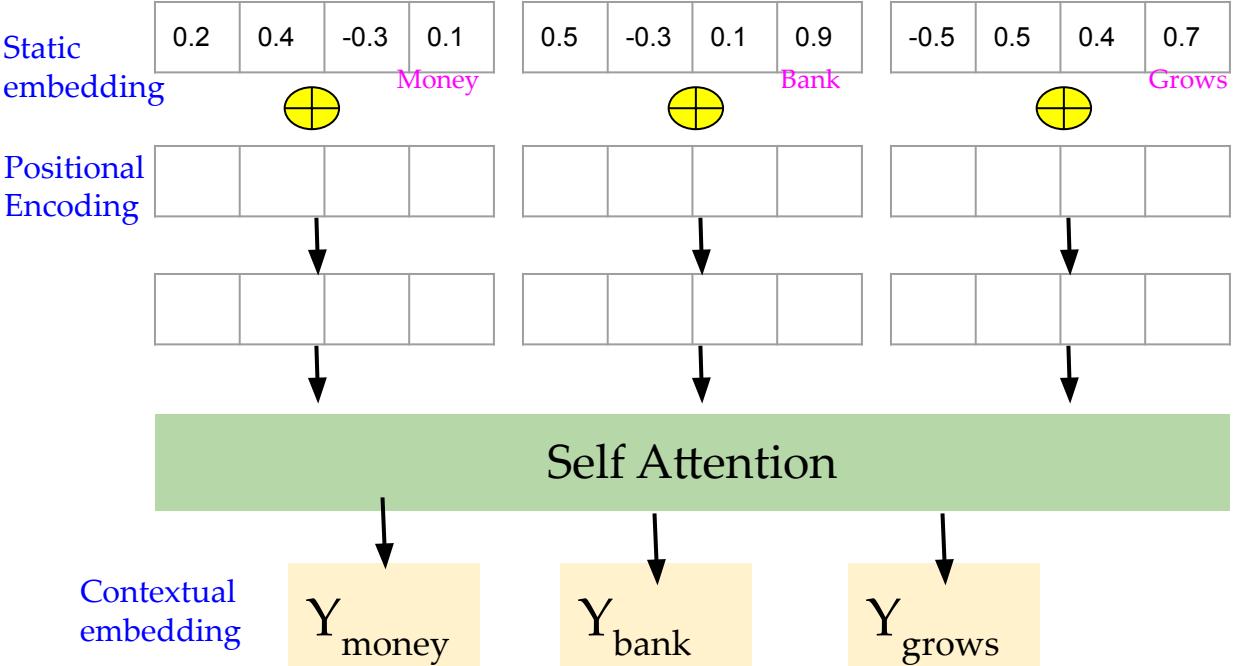


$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$



Transformer Encoder: Positional Encoding

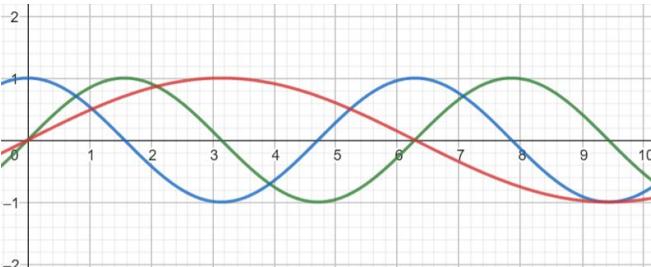
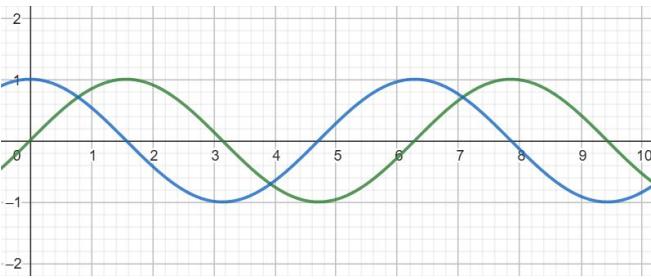
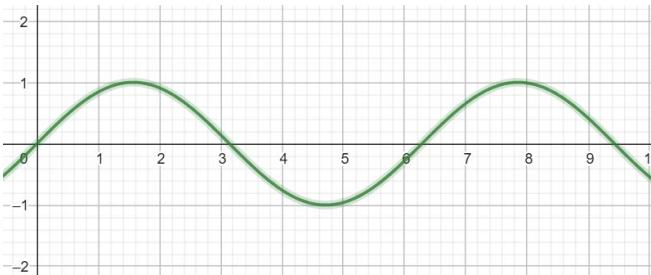


$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$d_{\text{model}} = 4$ (same as dimension of word embeddings)

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

$i = \text{ranges between } 0 \text{ to } (d_{\text{model}} - 1)$
(for each i we find sin and cosine value)



Transformer Encoder: Positional Encoding

Money

0			
---	--	--	--

For $i=Pos=0$

$$Pos(0,0) = \sin(0/10000^{0/4}) = 0$$

Bank

--	--	--	--

Grows

--	--	--	--

Let's calculate the 1st position embedding as an example.

Therefore,

$$d_{model} = 4 \text{ (same as dimension of word embeddings)}$$

i = ranges between 0 to $(d_{model}-1)$
(for each i we find sin and cosine value)

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

Transformer Encoder: Positional Encoding

Money

0	1		

For Pos=0, i=0

$$\text{Pos}(0,0) = \sin(0/10000^{0/4}) = 0$$

$$\text{Pos}(0,1) = \cos(0/10000^{0/4}) = 1$$

Bank

Grows

Let's calculate the 1st position embedding as an example.

Therefore,

$d_{\text{model}} = 4$ (same as dimension of word embeddings)

$i = \text{ranges between } 0 \text{ to } ((d_{\text{model}})/2) - 1$

(for each i we find sin and cosine value)

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

Transformer Encoder: Positional Encoding

Money

0	1	0	1
---	---	---	---

Bank

--	--	--	--

Grows

--	--	--	--

For Pos=0, i=0

$$\text{Pos}(0,0) = \sin(0/10000^{0/4}) = 0$$

$$\text{Pos}(0,1) = \cos(0/10000^{0/4}) = 1$$

For Pos=0, i=1

$$\text{Pos}(0,2) = \sin(0/10000^{2/4}) = 0$$

$$\text{Pos}(0,3) = \cos(0/10000^{2/4}) = 1$$

Let's calculate the 1st position embedding as an example.

Therefore,

$d_{\text{model}} = 4$ (same as dimension of word embeddings)

$i = \text{ranges between } 0 \text{ to } (d_{\text{model}} - 1)$

(for each i we find sin and cosine value)

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

Transformer Encoder: Positional Encoding

Money

0	1	0	1
---	---	---	---

Bank

0.84	0.54	0.009	0.99
------	------	-------	------

Grows

0.9	-0.41	0.01	0.99
-----	-------	------	------

Money positional Embedding:

For Pos=0, i=0

$$\text{Pos}(0,0) = \sin(0/10000^{0/4})=0$$

$$\text{Pos}(0,1) = \cos(0/10000^{0/4})=1$$

For Pos=1, i=1

$$\text{Pos}(0,2) = \sin(0/10000^{2/4})=0$$

$$\text{Pos}(0,3) = \cos(0/10000^{2/4})=1$$

Bank positional Embedding:

For Pos=1, i=0

$$\text{Pos}(1,0) = \sin(1/10000^{0/4})=0.84$$

$$\text{Pos}(1,1) = \cos(1/10000^{0/4})=0.54$$

For Pos=1, i=1

$$\text{Pos}(1,2) = \sin(1/10000^{2/4})=0.009$$

$$\text{Pos}(1,3) = \cos(1/10000^{2/4})=0.99$$

Let's calculate the 1st position embedding as an example.

Therefore,

$d_{\text{model}} = 4$ (same as dimension of word embeddings)
 $i =$ ranges between 0 to $(d_{\text{model}}/2) - 1 = 0, 1$
(for each i we find sin and cosine value)

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

Grows positional Embedding:

For Pos=2, i=0

$$\text{Pos}(2,0) = \sin(2/10000^{0/4})=0.9$$

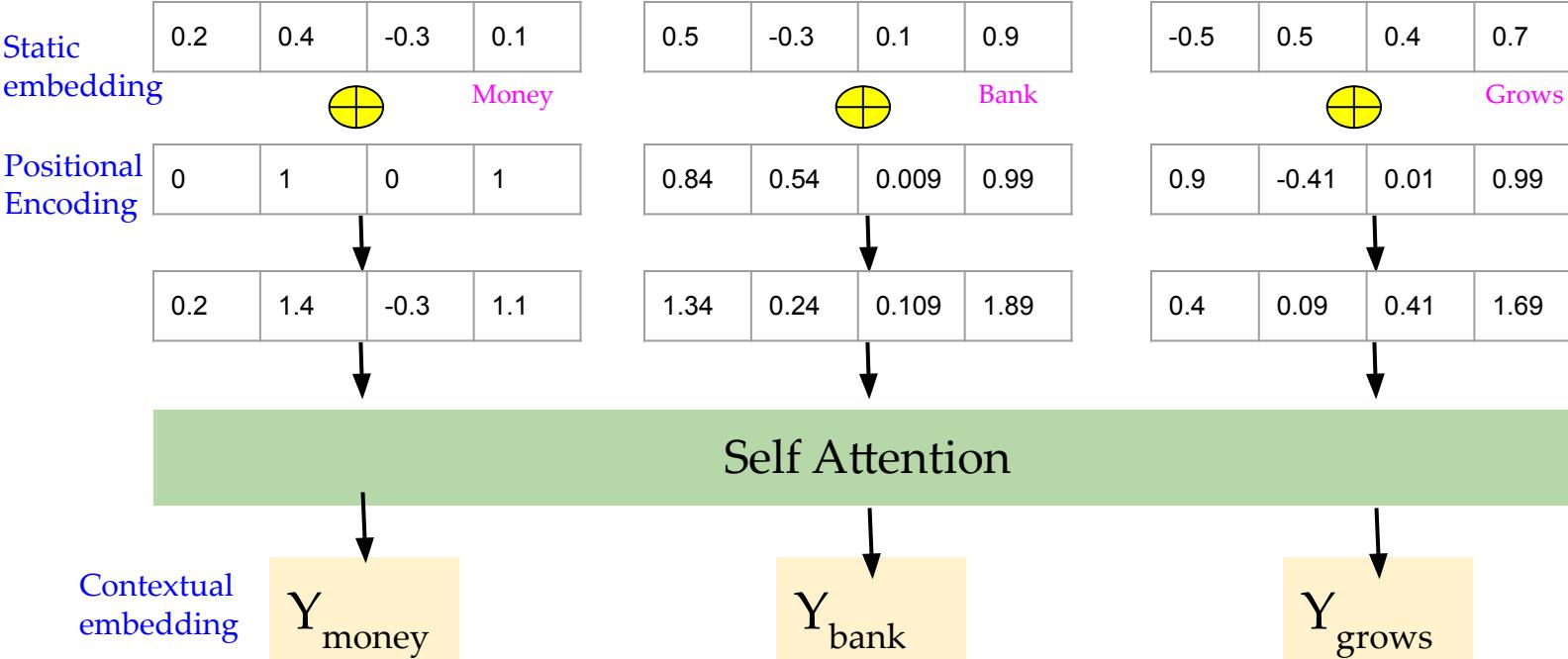
$$\text{Pos}(2,1) = \cos(2/10000^{0/4})= -0.41$$

For Pos=2, i=1

$$\text{Pos}(2,2) = \sin(2/10000^{2/4})=0.01$$

$$\text{Pos}(2,3) = \cos(2/10000^{2/4})=0.99$$

Transformer Encoder: Positional Encoding



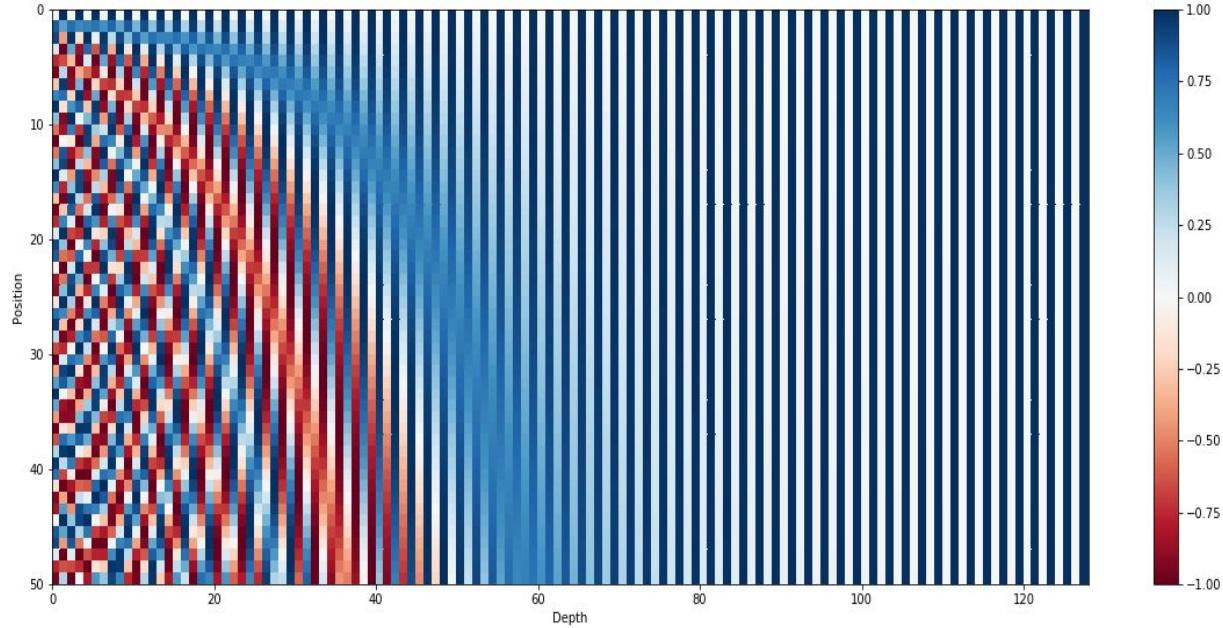
$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

d_{model} = 4 (same as dimension of word embeddings)

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

i = ranges between 0 to ($d_{\text{model}} - 1$)
(for each i we find sin and cosine value)

Positional Encoding behaviour



1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	0	1	1
5	0	1	0	0
6	0	1	0	1
7	0	1	1	0
8	0	1	1	1
9	1	0	0	0
10	1	0	0	1
11	1	0	1	0
12	1	0	1	1
13	1	1	0	0
14	1	1	0	1
15	1	1	1	0

Transformer Encoder: Positional Encoding

Sequence Index of token, k

	$i=0$	$i=0$	$i=1$	$i=1$
I	$P_{00}=\sin(0) = 0$	$P_{01}=\cos(0) = 1$	$P_{02}=\sin(0) = 0$	$P_{03}=\cos(0) = 1$
am	$P_{10}=\sin(1/1) = 0.84$	$P_{11}=\cos(1/1) = 0.54$	$P_{12}=\sin(1/10) = 0.10$	$P_{13}=\cos(1/10) = 1.0$
a	$P_{20}=\sin(2/1) = 0.91$	$P_{21}=\cos(2/1) = -0.42$	$P_{22}=\sin(2/10) = 0.20$	$P_{23}=\cos(2/10) = 0.98$
Robot	$P_{30}=\sin(3/1) = 0.14$	$P_{31}=\cos(3/1) = -0.99$	$P_{32}=\sin(3/10) = 0.30$	$P_{33}=\cos(3/10) = 0.96$

Positional Encoding Matrix for the sequence 'I am a robot'

Transformer Encoder: Positional Encoding

I am going

$[[0.1, -0.3, 0.7], [0.5, -0.3, 1], [0.8, -1, -1.8]] \rightarrow \text{Word embeddings}$



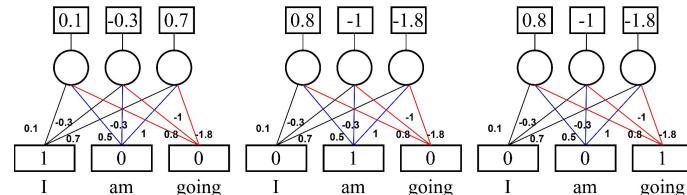
$[[0, 1, 0], [0.84, 0.54, 0.1], [0.91, 0.42, 0.2]] \rightarrow \text{Positional encoding remains same}$

$\Rightarrow [[0.1, 0.7, 0.7], [1.34, 0.24, 1.1], [1.71, -0.58, 0.8]] \rightarrow \text{New positional encoding}$

$$e_p^0: [0.1, 0.7, 0.7]$$

$$e_p^1: [1.34, 0.24, 1.1]$$

$$e_p^2: [1.71, -0.58, 0.8]$$



Transformer Encoder: Positional Encoding

going am i → Changed word order

$[[0.8, -1, -1.8], [0.5, -0.3, 1], [0.1, -0.3, 0.7]]$ → Word embeddings



$[[0, 1, 0], [0.84, 0.54, 0.1], [0.91, 0.42, 0.2]]$ → Positional encoding remains same

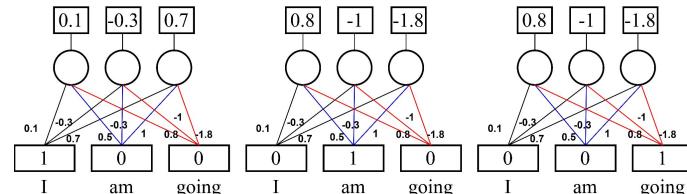
⇒ $[[0.8, 0, -1.8], [1.34, 0.24, 1.1], [1.01, 0.12, 0.9]]$ → New positional encoding

$$e_p^0: [0.8, 0, -1.8]$$

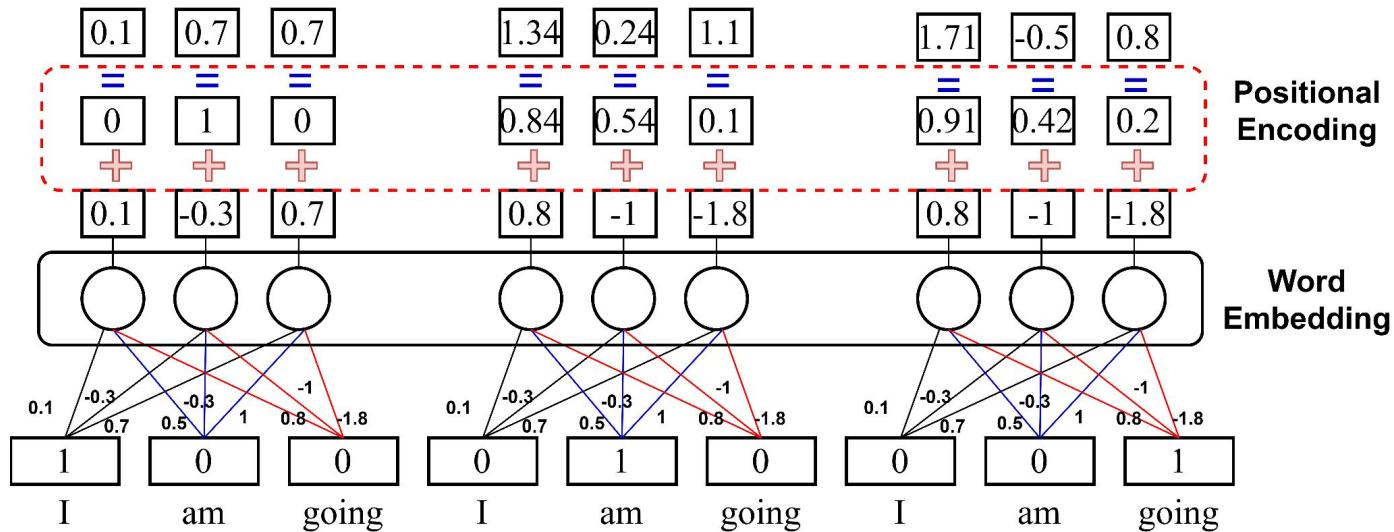
$$e_p^1: [1.34, 0.24, 1.1]$$

$$e_p^2: [1.01, 0.12, 0.9]$$

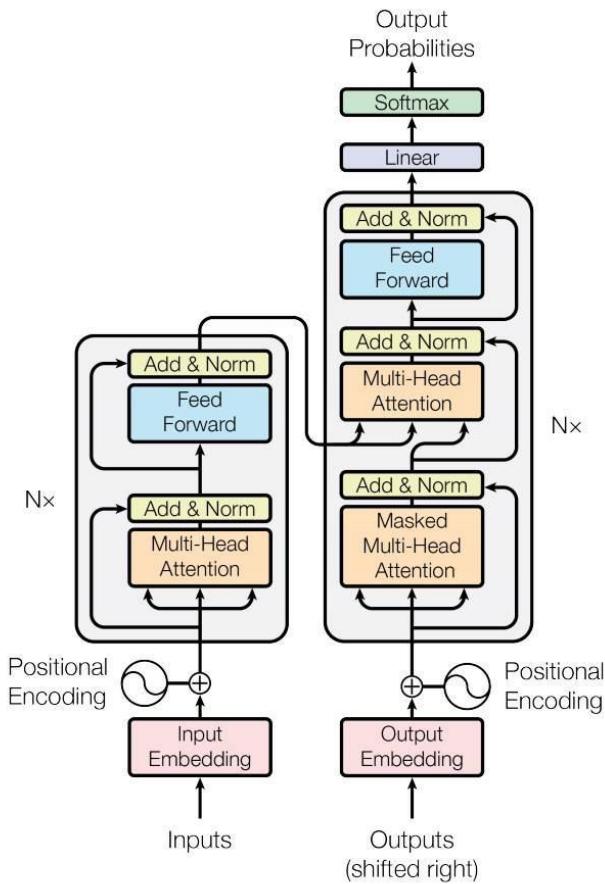
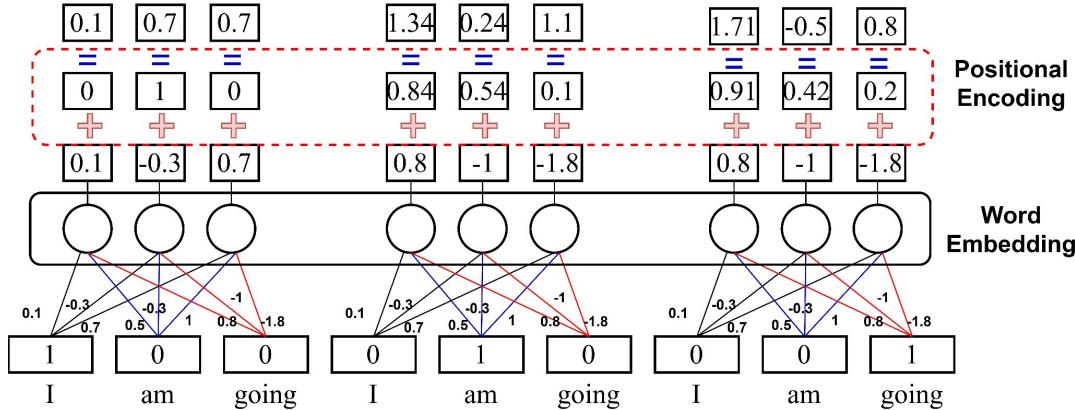
Thus, positional embedding allows Transformer to maintain the word order.



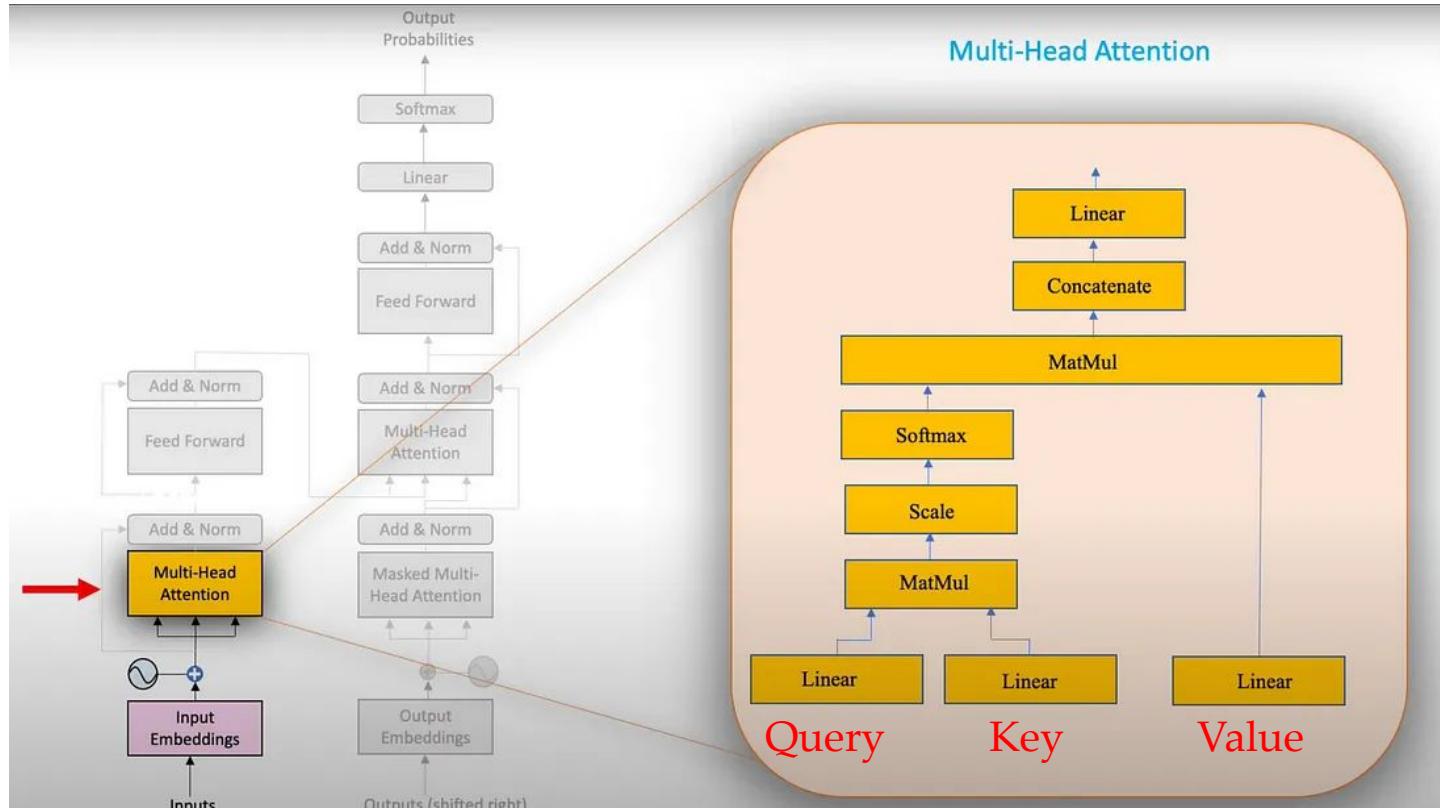
Transformer Encoder: Positional Encoding



Transformer: Self Attention

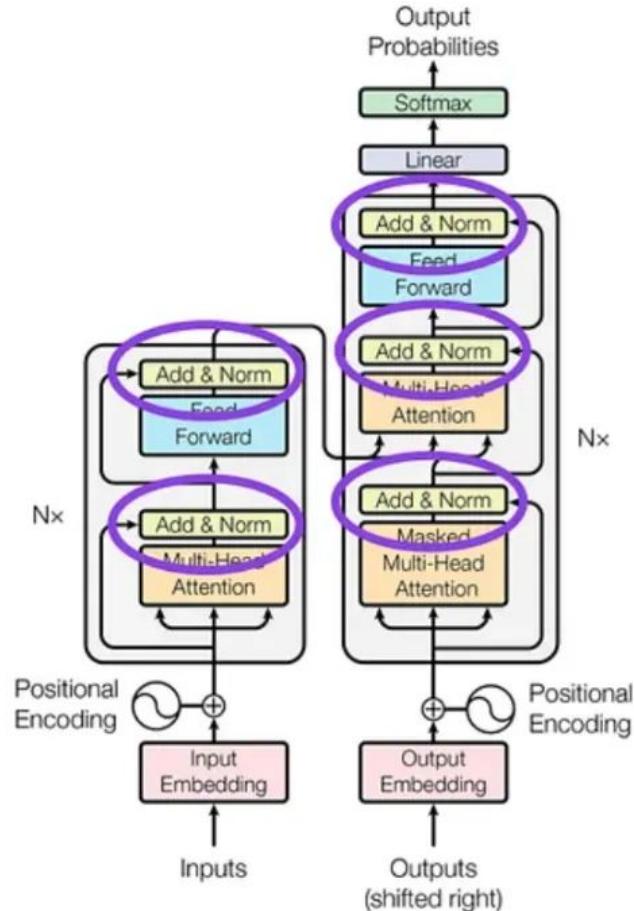


Transformer: Self Attention



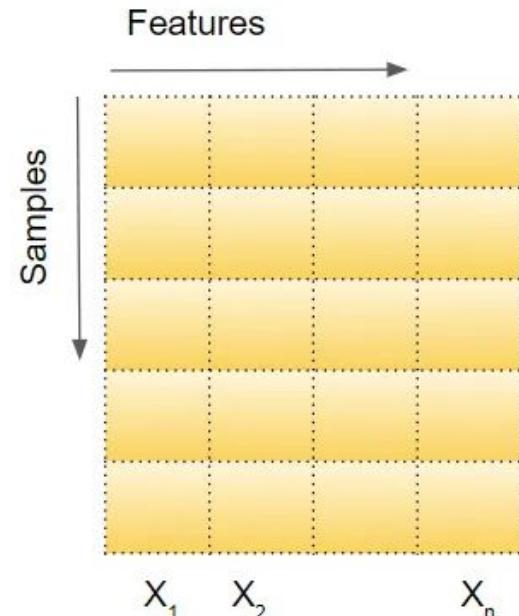
Transformer: Layer Normalization

Why normalization is required in Transformer?



Normalization in Deep Learning

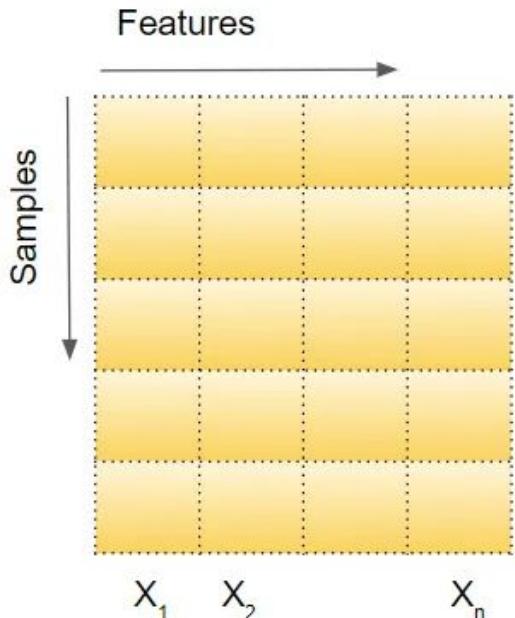
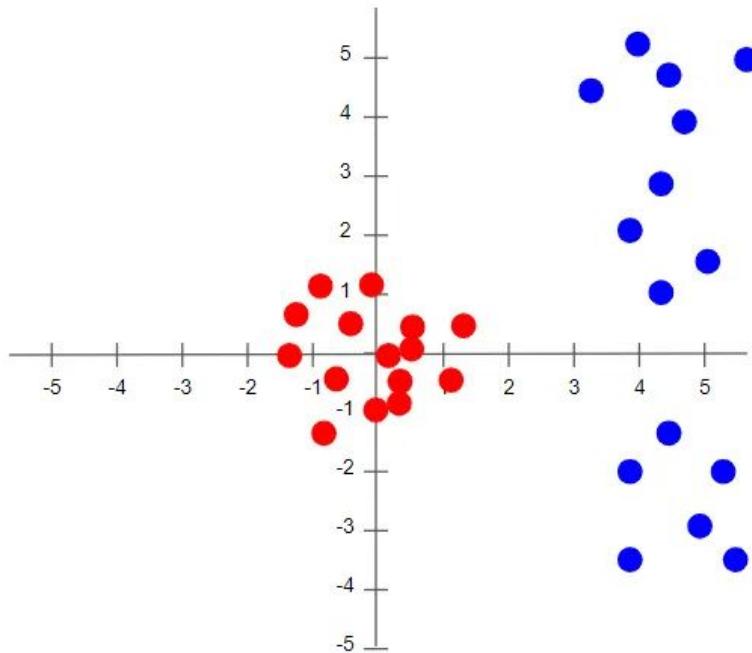
- Input data consists of several features x_1, x_2, \dots, x_n .
- Each feature might have a different range of values.
 - For instance, values for feature x_1 might range from 1 through 5,
 - while values for feature x_2 might range from 1000 to 99999.
- In Deep learning, we first normalize our features and then pass it to network.
- It helps the network to converge faster.



$$X_i = \frac{X_i - \text{Mean}_i}{\text{StdDev}_i}$$

Normalization in Deep Learning

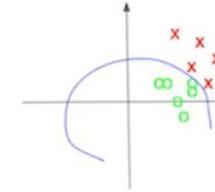
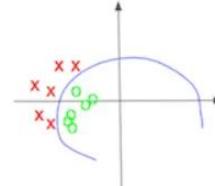
- Blue distribution: before normalization
- Red distribution: after normalization



$$X_i = \frac{X_i - \text{Mean}_i}{\text{StdDev}_i}$$

The need for batch normalization

- It helps in addressing Internal Covariate Shift
 - Internal covariate shift refers to this problem where the distribution of activations changes during training due to the constant updates to the network's weights.
 - This shifting can cause each subsequent layer to receive inputs with varying distributions, which can hinder stable learning.



The need for batch normalization

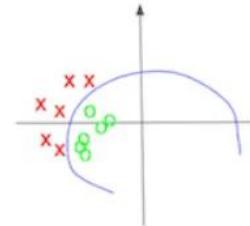
- Batch normalization mitigates this issue by normalizing the activations within each layer, ensuring they follow a consistent distribution with a mean of zero and a standard deviation of one.



Rose
(y=1)



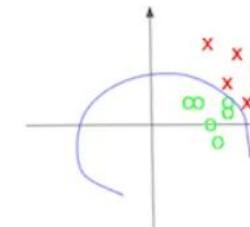
Not Rose
(y=0)



Rose
(y=1)

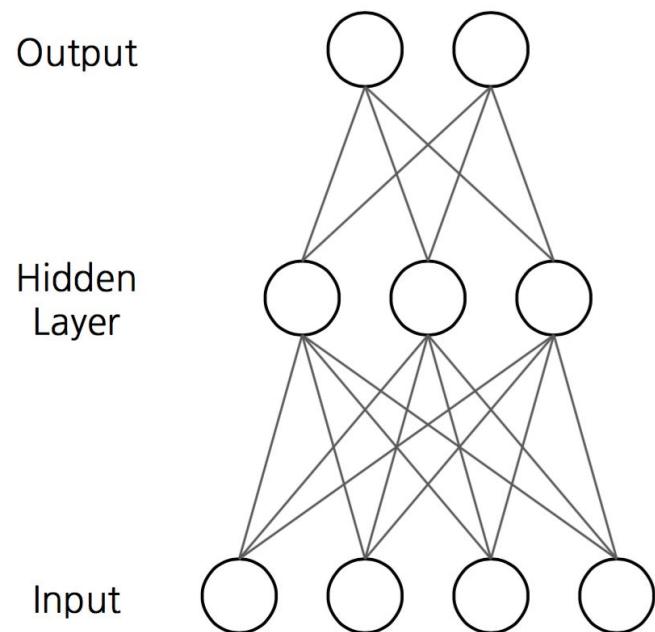


Not Rose
(y=0)

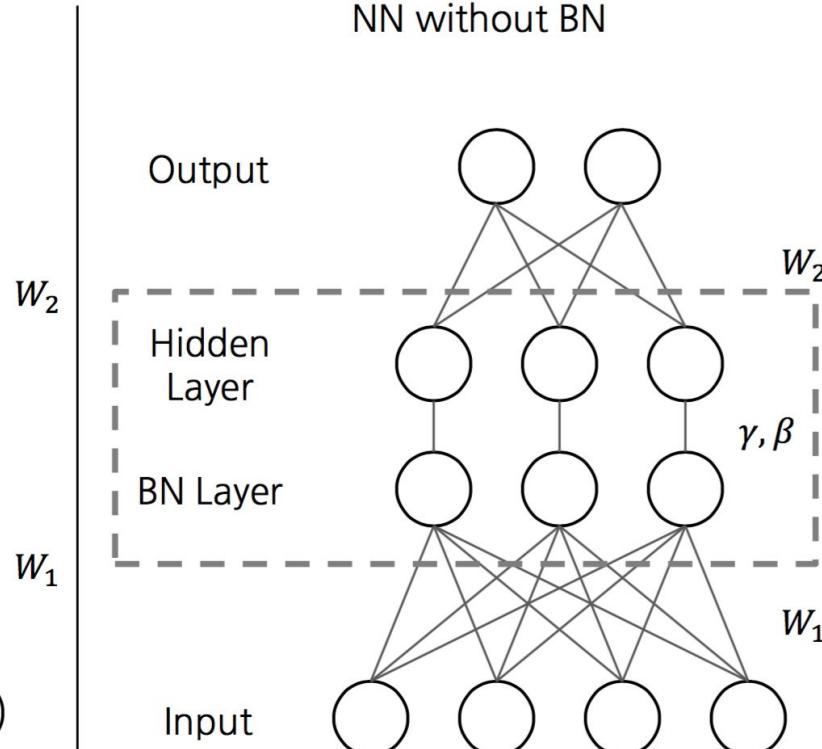


Batch normalization in NN

NN without BN

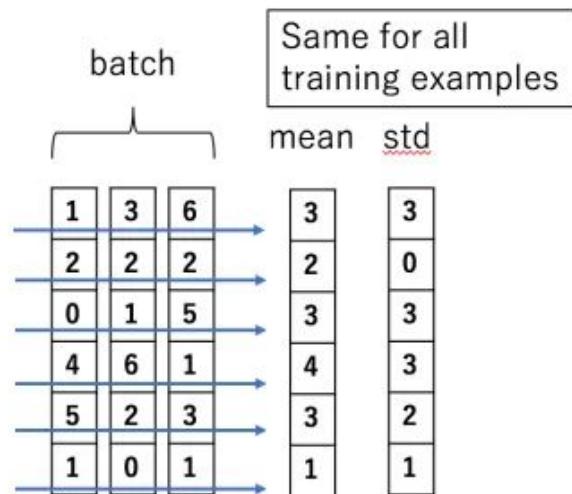


NN without BN

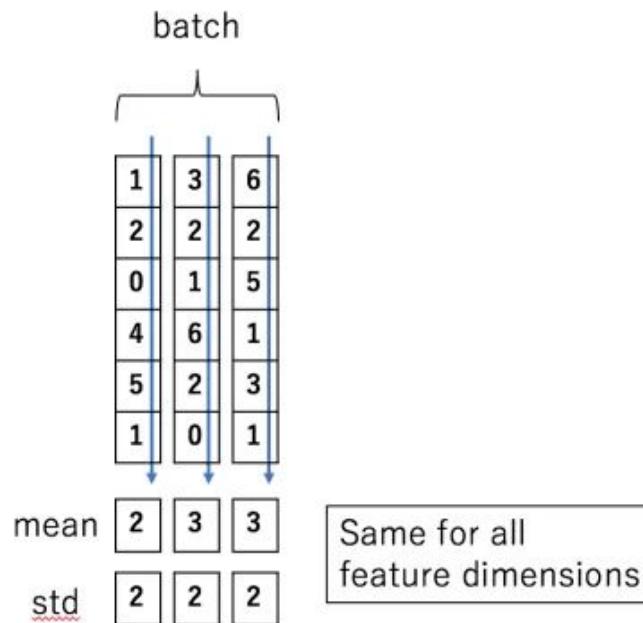


Layer normalization

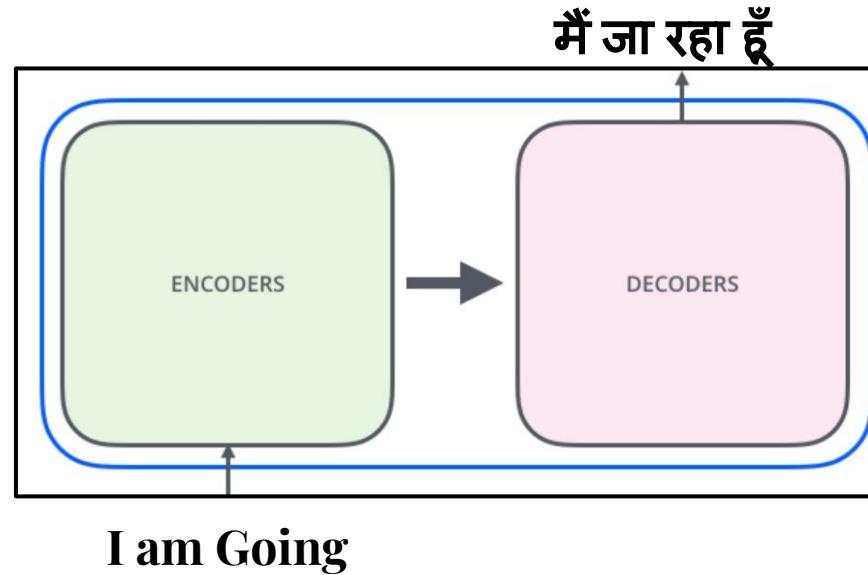
Batch Normalization



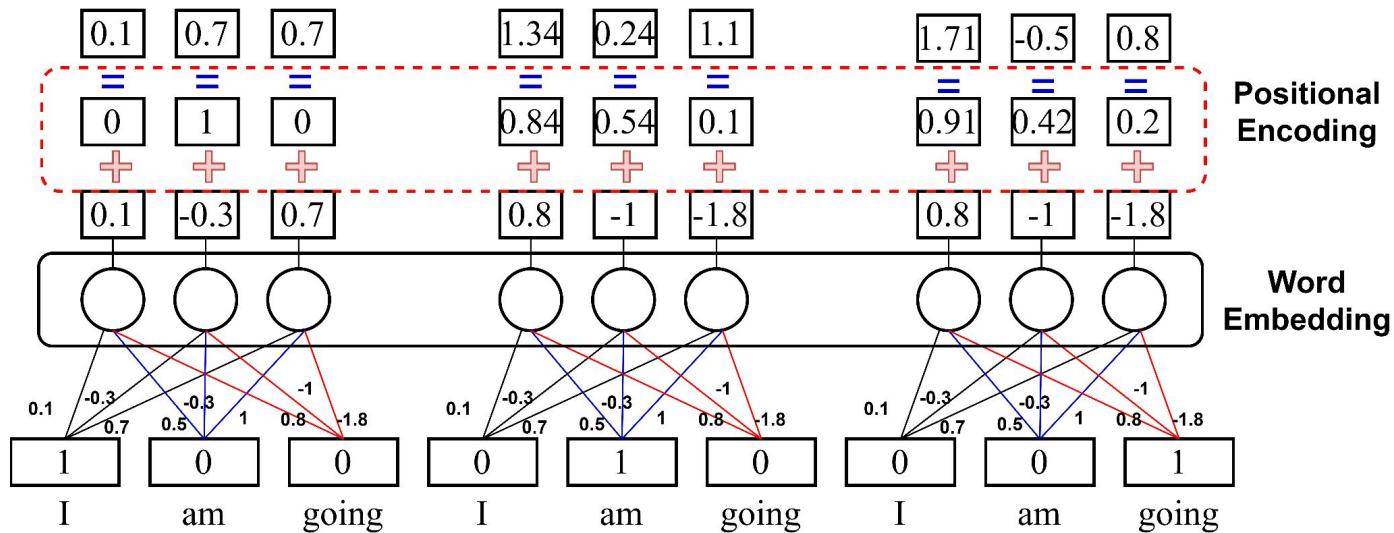
Layer Normalization



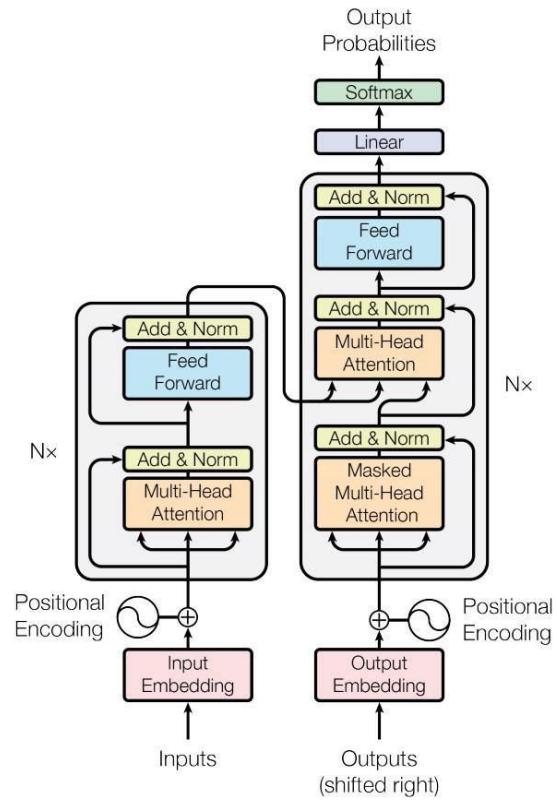
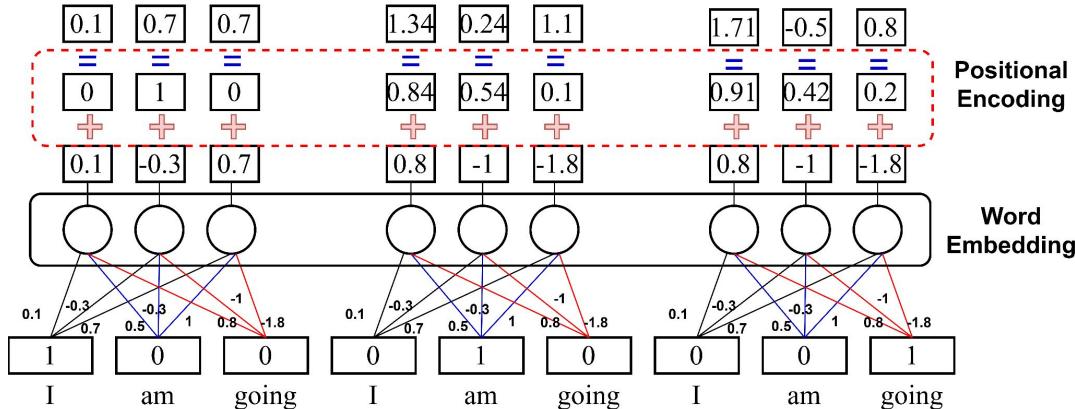
Transformer Architecture



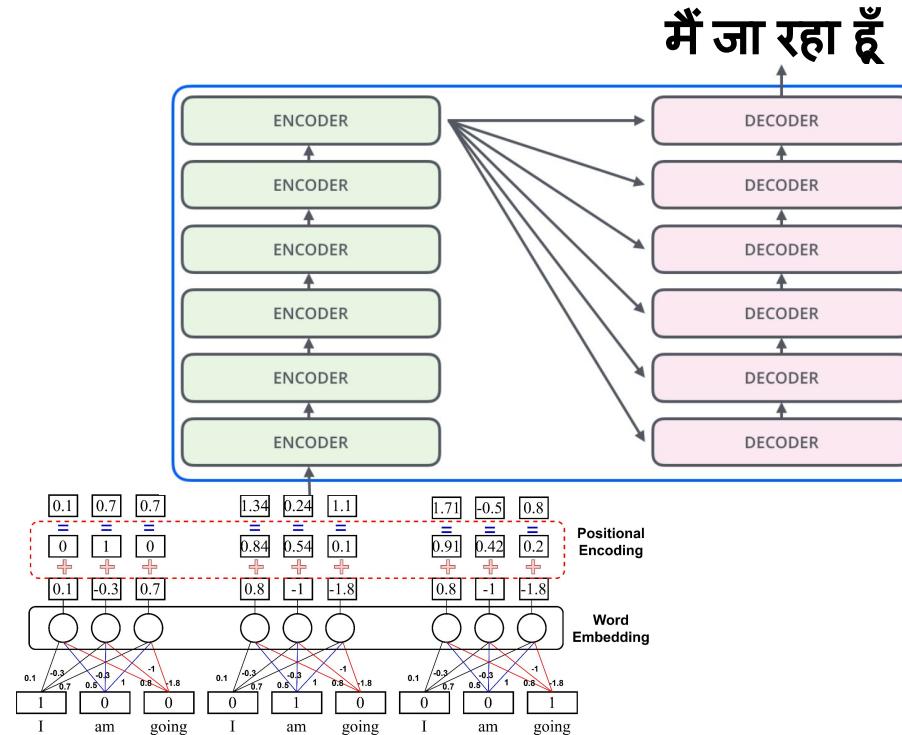
Transformer Architecture: Input



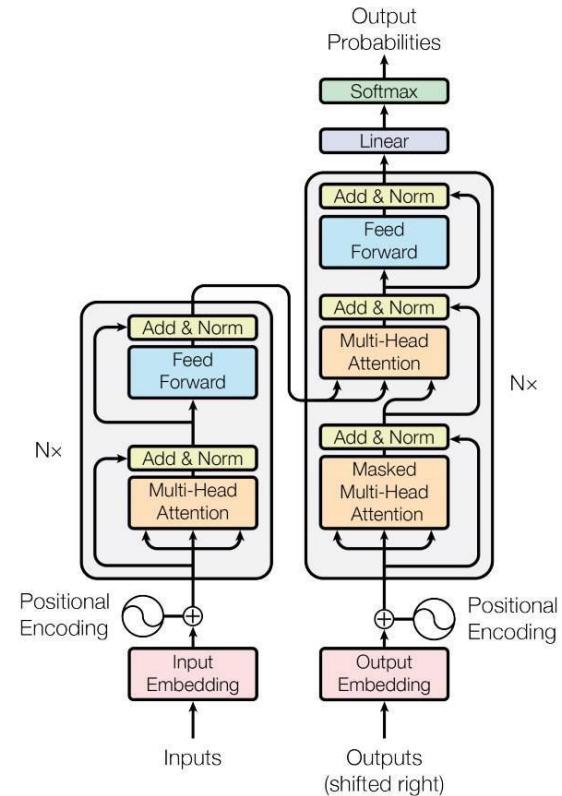
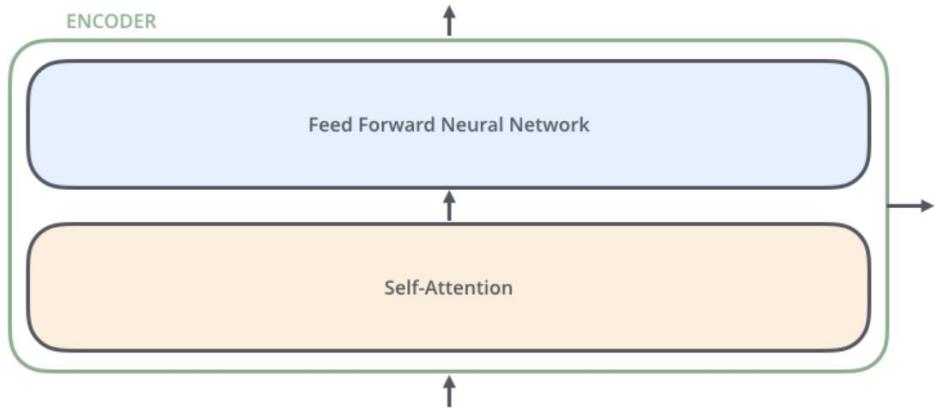
Transformer Architecture



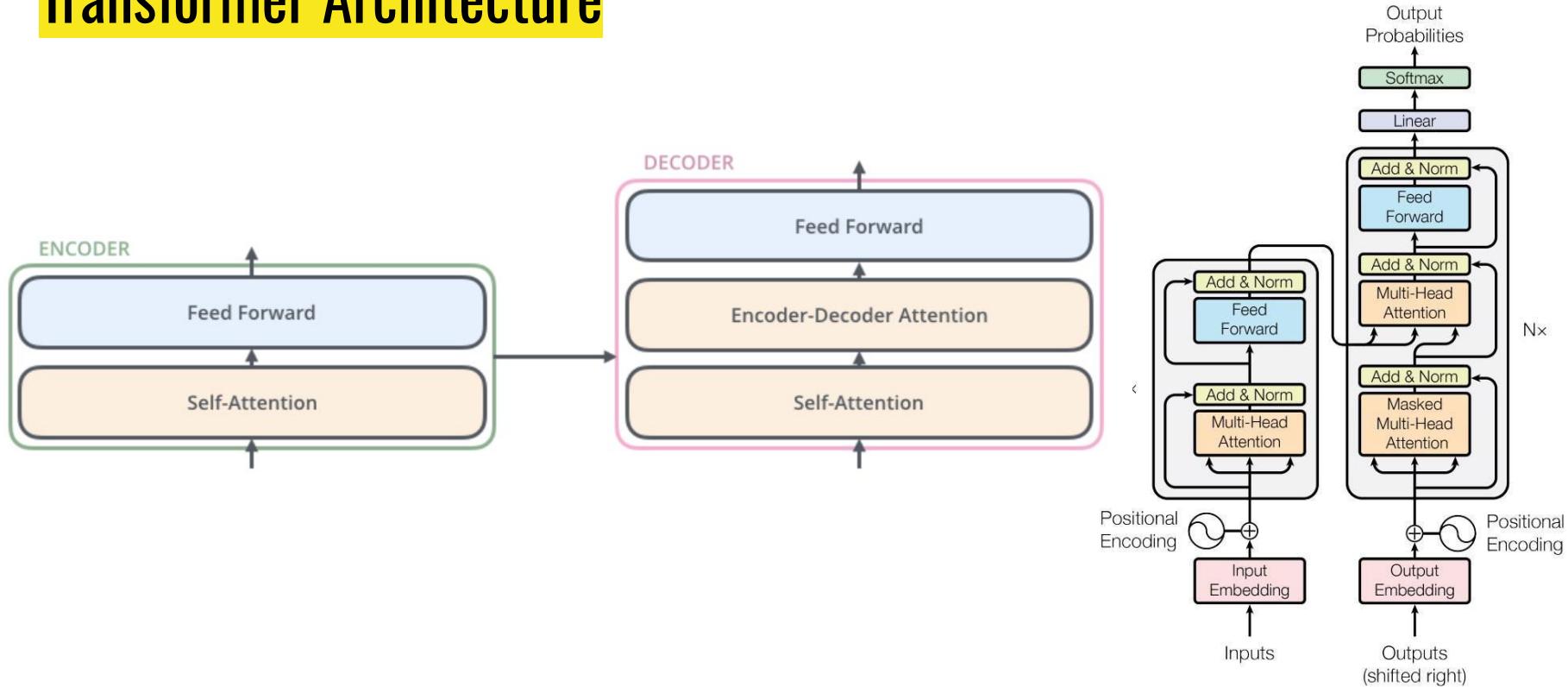
Transformer Architecture



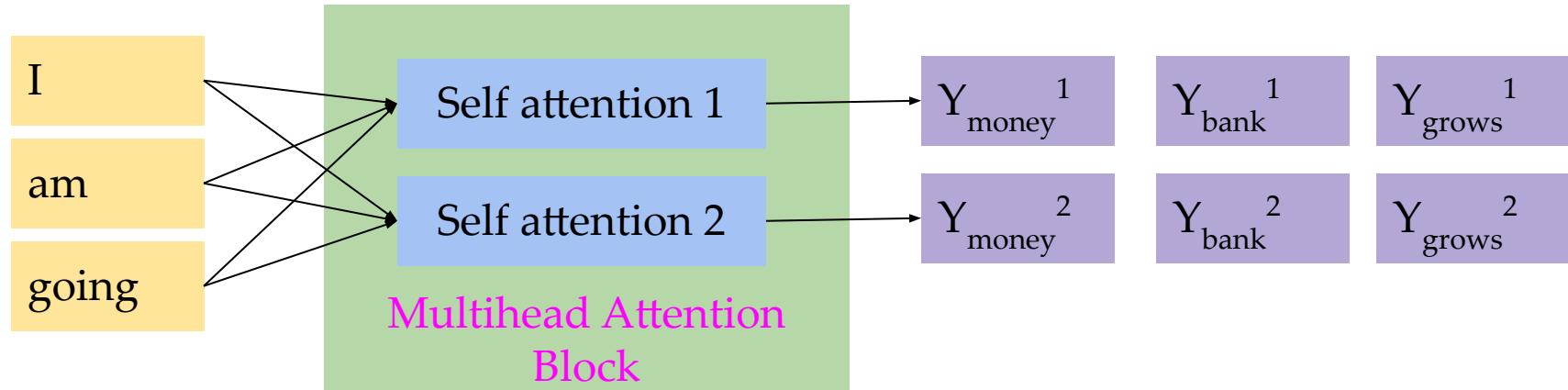
Transformer Architecture: Encoder



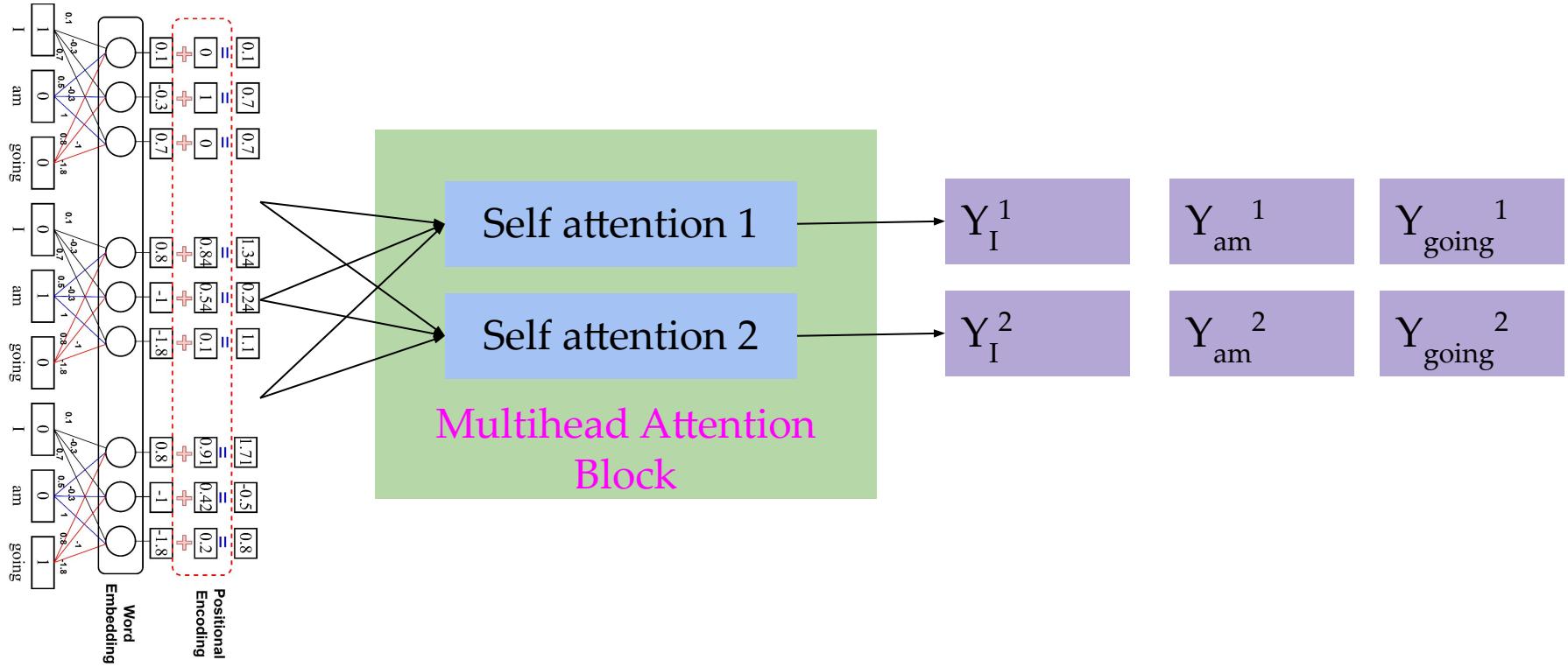
Transformer Architecture



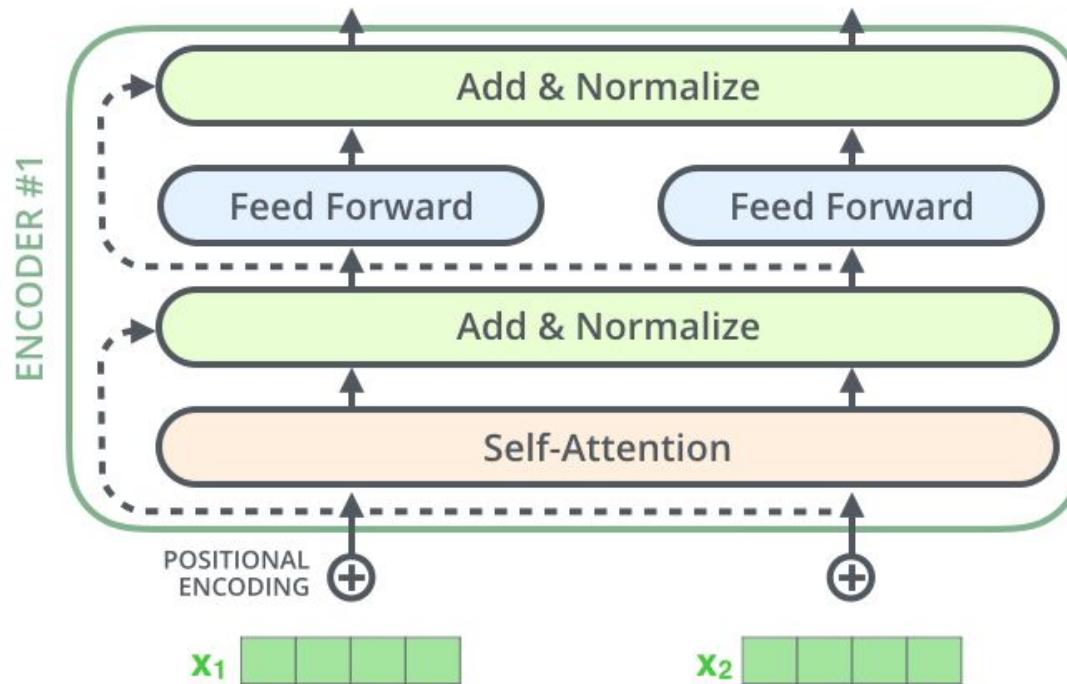
Transformer Encoder Architecture: Multi Head Attention



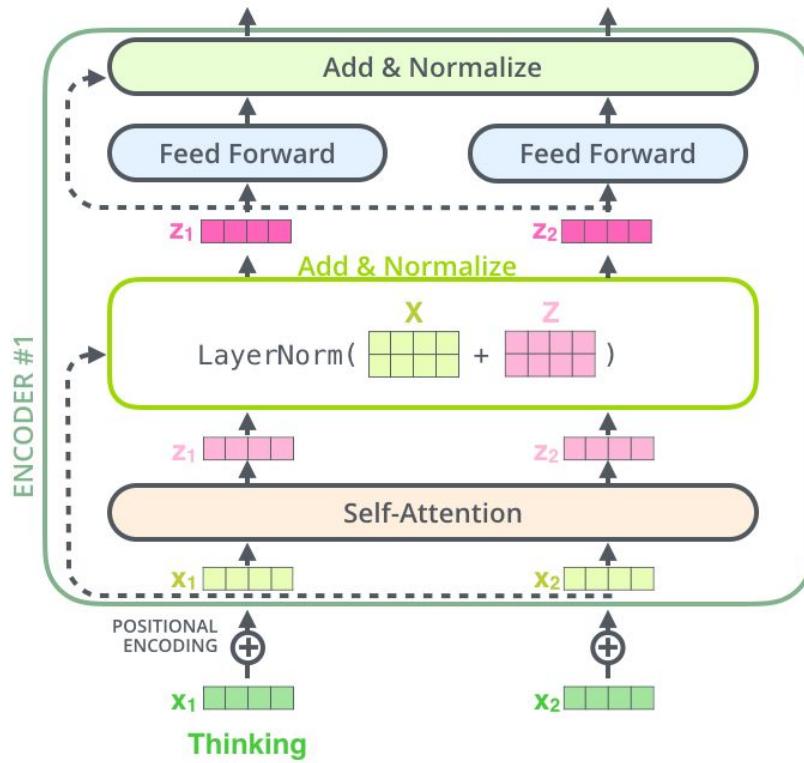
Transformer Encoder Architecture: Multi Head Attention



Transformer Encoder Architecture: Add & Normalization



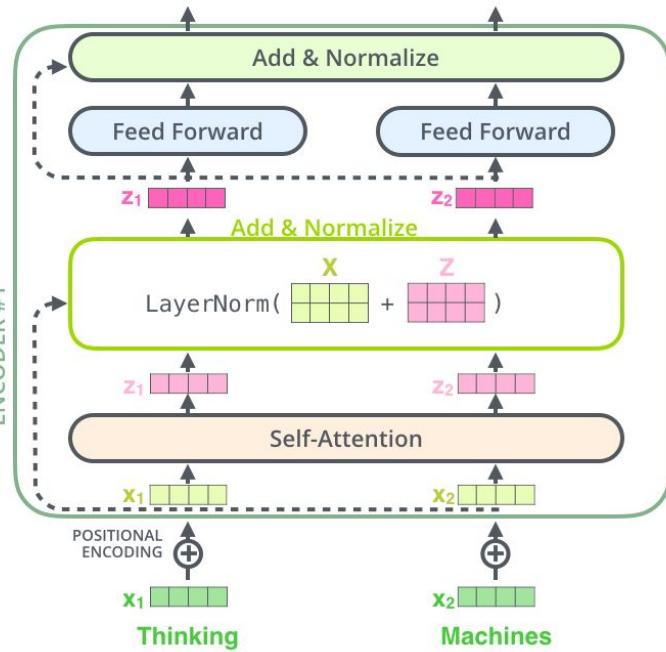
Transformer Encoder Architecture: Add & Normalization



Addition and Layer Normalization:

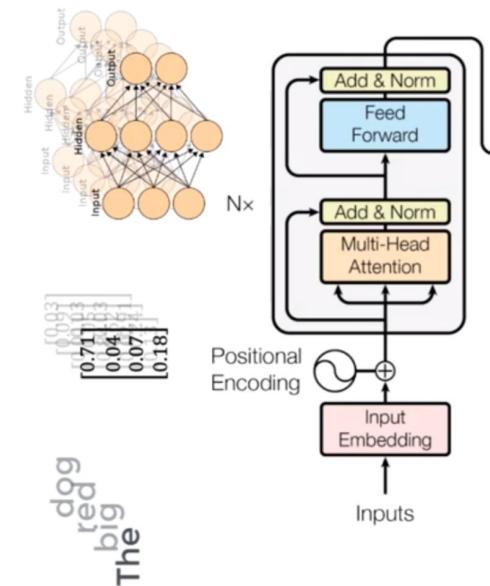
- Self attention output ($Z=Z_1+Z_2$) is first added with input ($X=X_1+X_2$) through skip connection.
- Later, Layer normalization is applied for smooth learning.

Transformer Encoder Architecture: Feed forward network

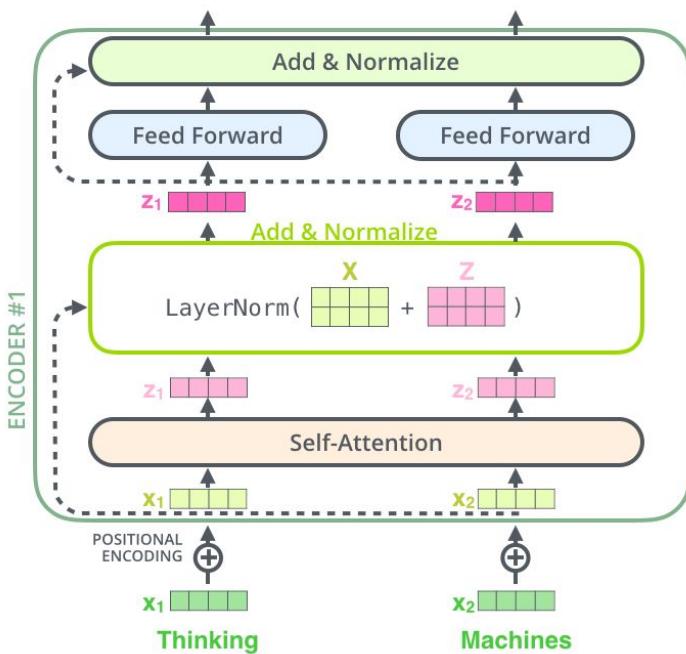


Two layer FFNN:

- Self attention is linear representation of input.
- To introduce non linearity FFNN is introduced.
- Input: 512 dimensional contextual embedding from add & norm
- Layer 1: 2048 neurons with ReLU
- Layer 2: 512 neurons



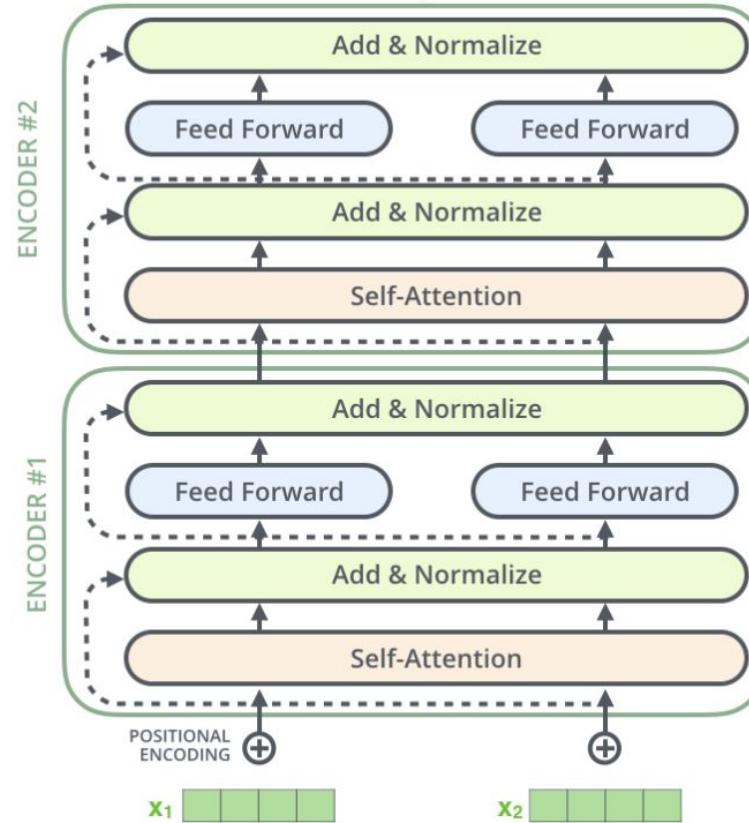
Transformer Encoder Architecture: Feed forward network



Addition and Layer Normalization after FFNN:

- Feed forward output ($Y=Y_1+Y_2$) is first added with input ($Z=Z_1+Z_2$) through skip connection.
- Later, Layer normalization is applied for smooth learning.

Transformer Encoder Architecture: 6 Encoders



Transformer Encoder Architecture: Queries

Why residual connections are required?

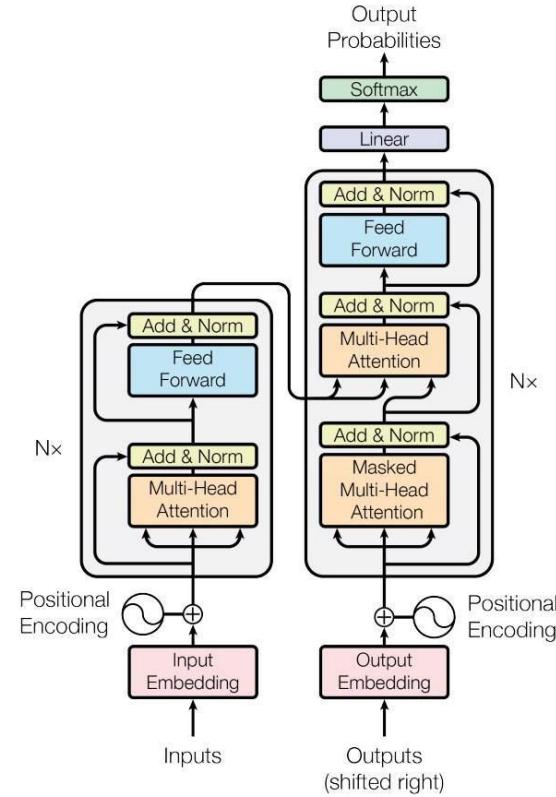
- Stable training.
- Original input embeddings are fed to the next layer
 - Helps in case if Self-attention does not work effectively.

Why Feed Forward NN?

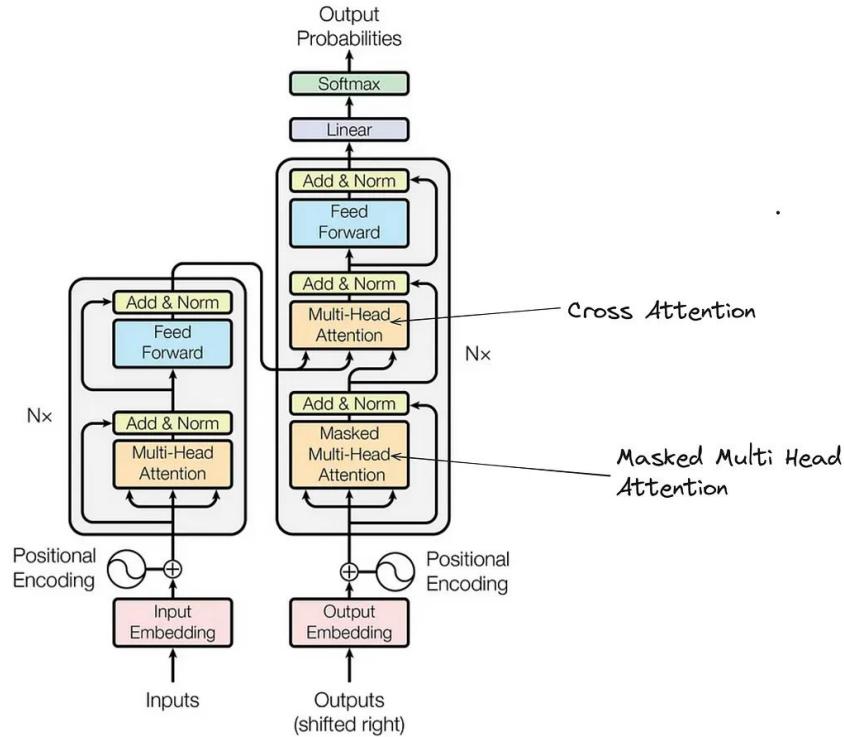
- To capture non-linearity in the data.
- Self attention is linear in nature.

Why 6 Encoders?

- To capture the natural language perspectives in a better way.
- The no. of encoders may vary in different cases.



Transformer Decoder Architecture



Transformer Decoder Architecture

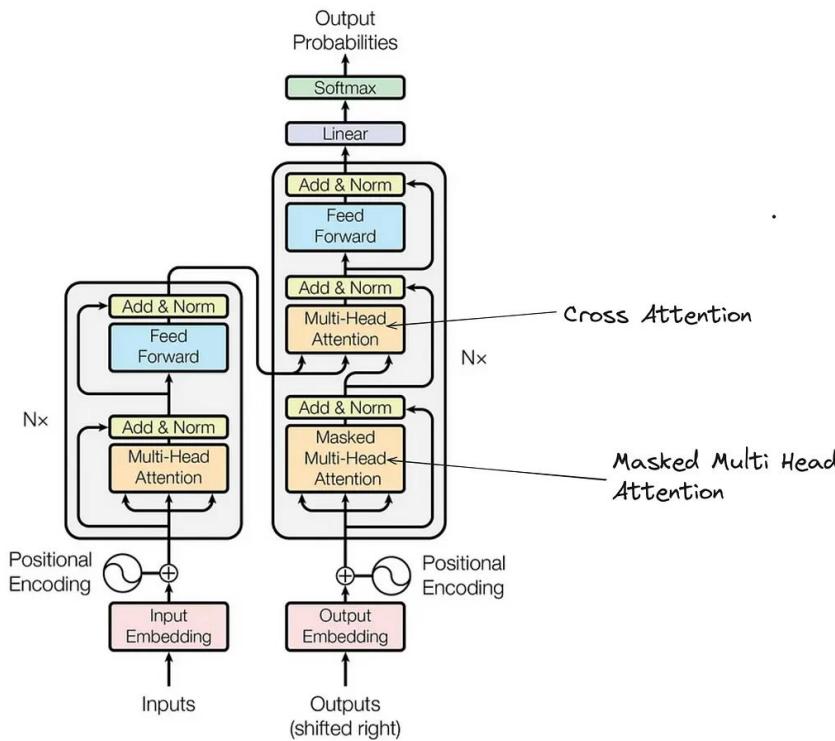


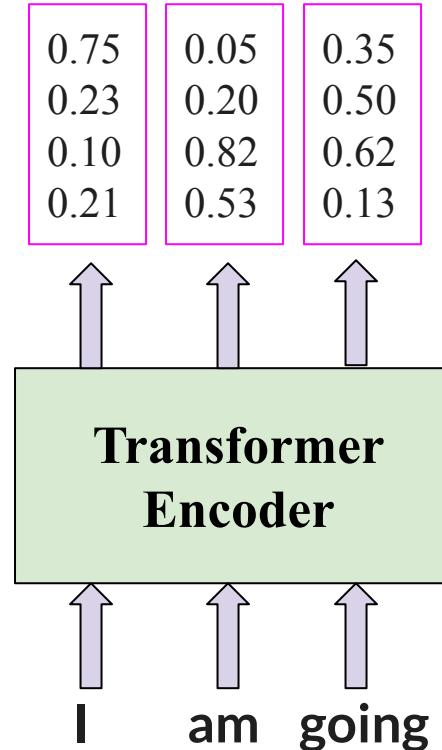
Figure 1: The Transforn

The Transformer decoder is autoregressive at inference time and non-autoregressive at training time.

[Link](#): Autoregressive LM.

Transformer for MT task: Inference Time

English: I am going

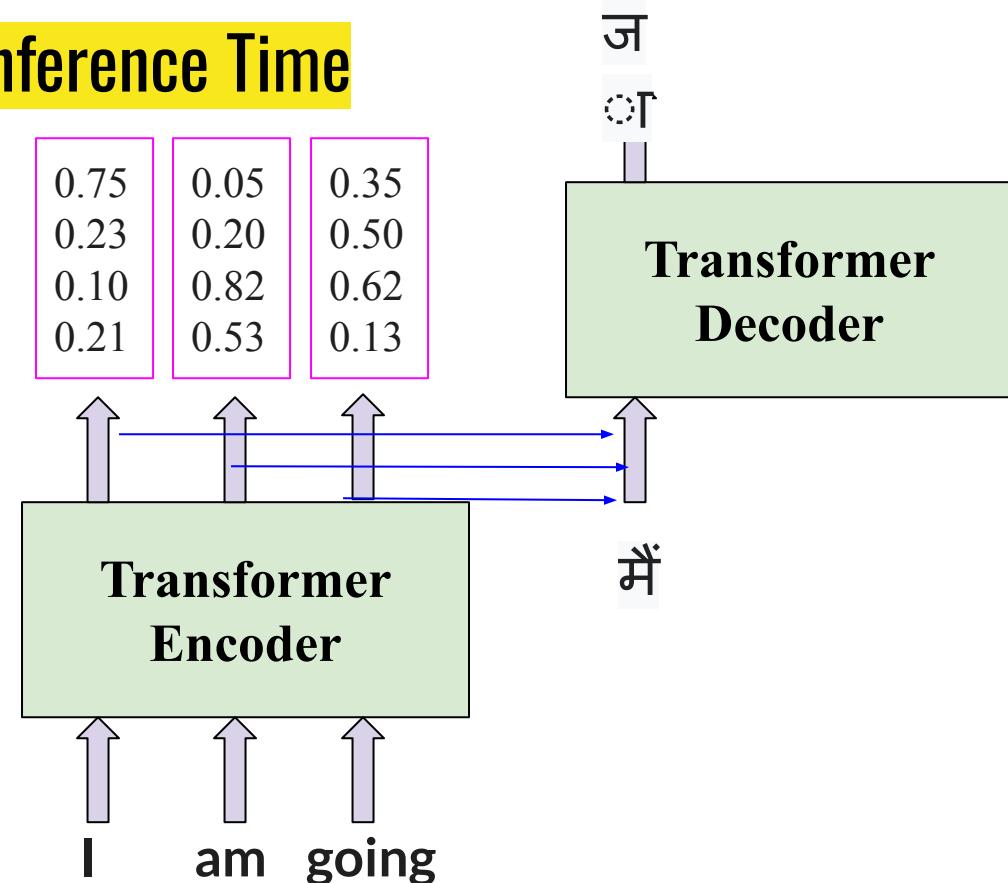


Contextual
Embeddings

Static Embeddings

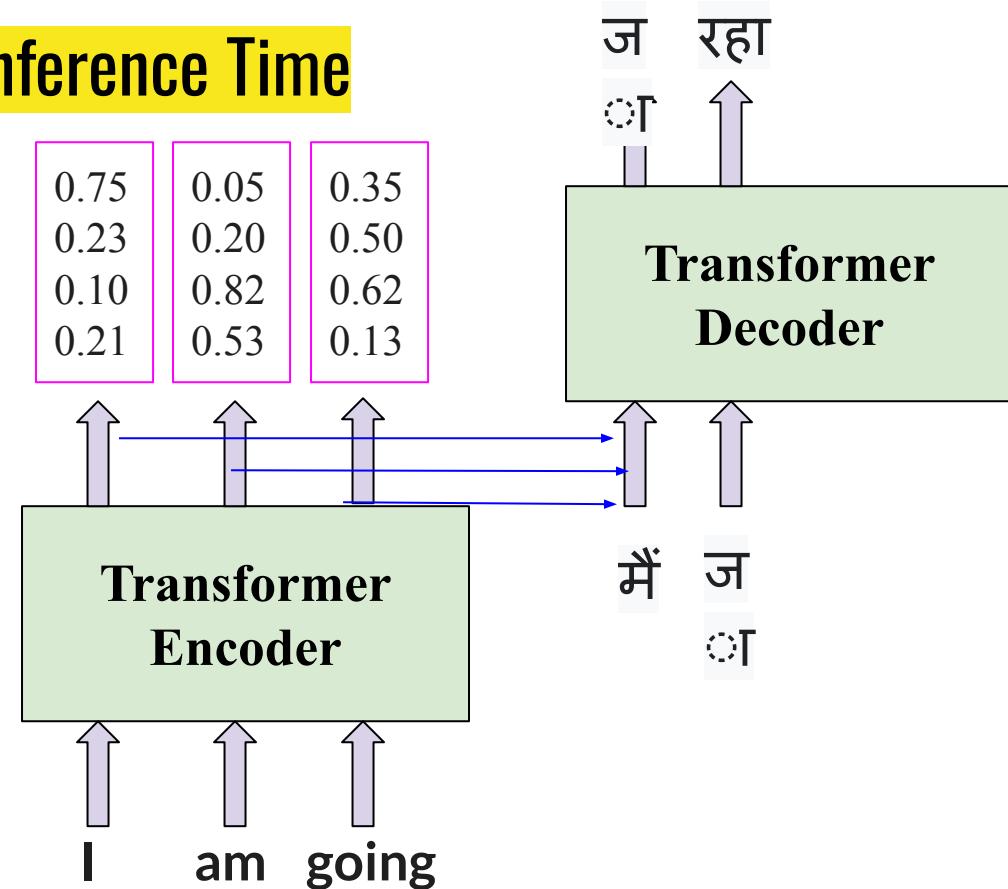
Transformer for MT task: Inference Time

English: I am going



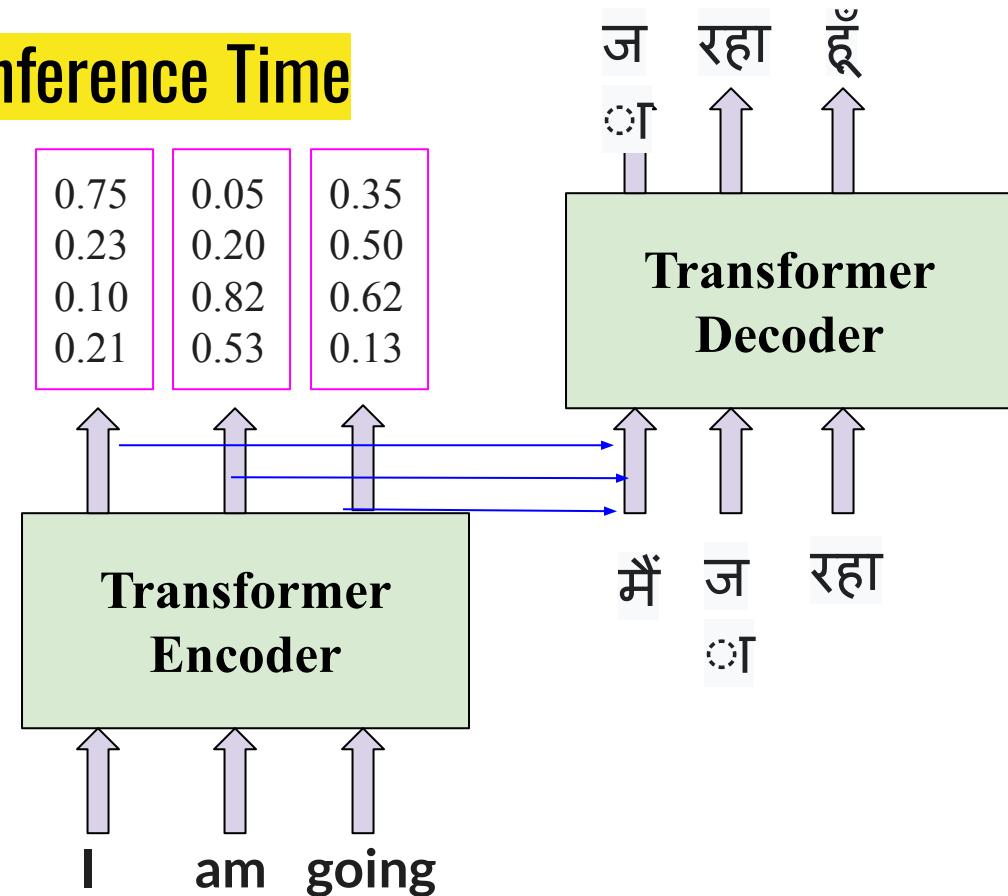
Transformer for MT task: Inference Time

English: I am going



Transformer for MT task: Inference Time

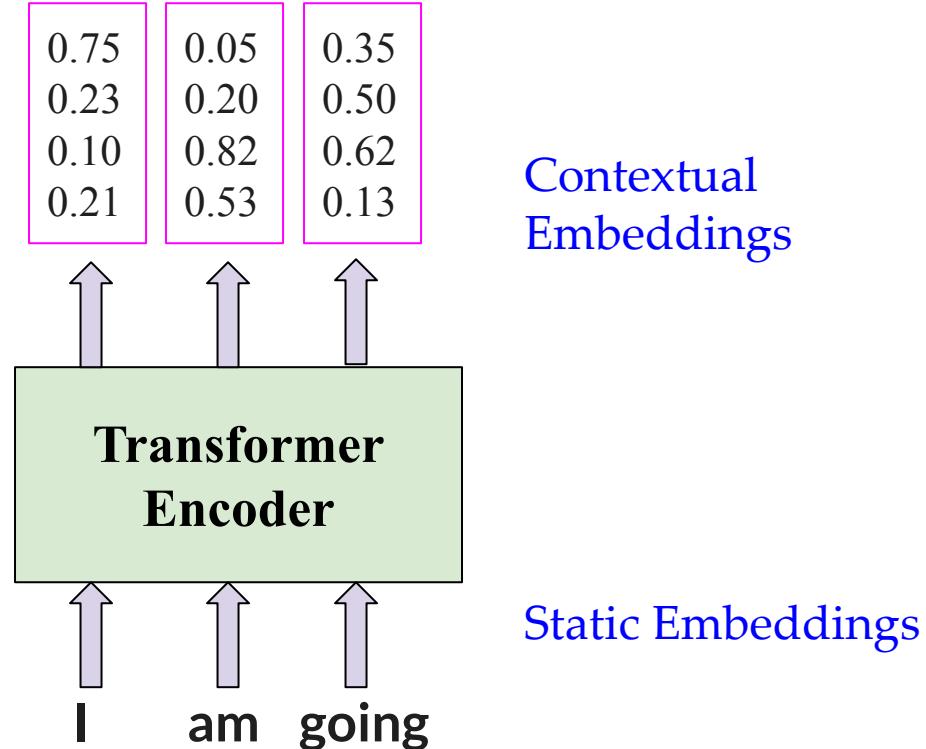
English: I am going



Transformer for MT task: Training Time

English: I am going
Hindi: मैं जा रहा हूँ

Autoregressive Approach

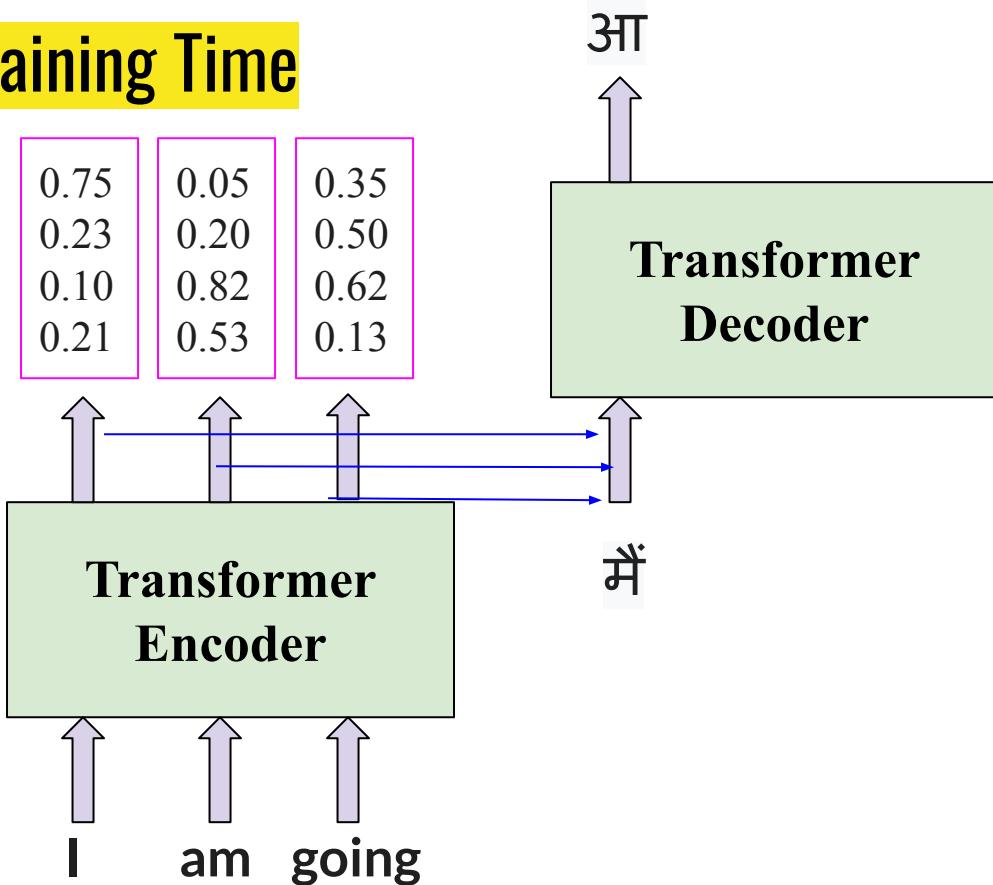


Transformer for MT task: Training Time

English: I am going

Hindi: मैं जा रहा हूँ

Autoregressive Approach

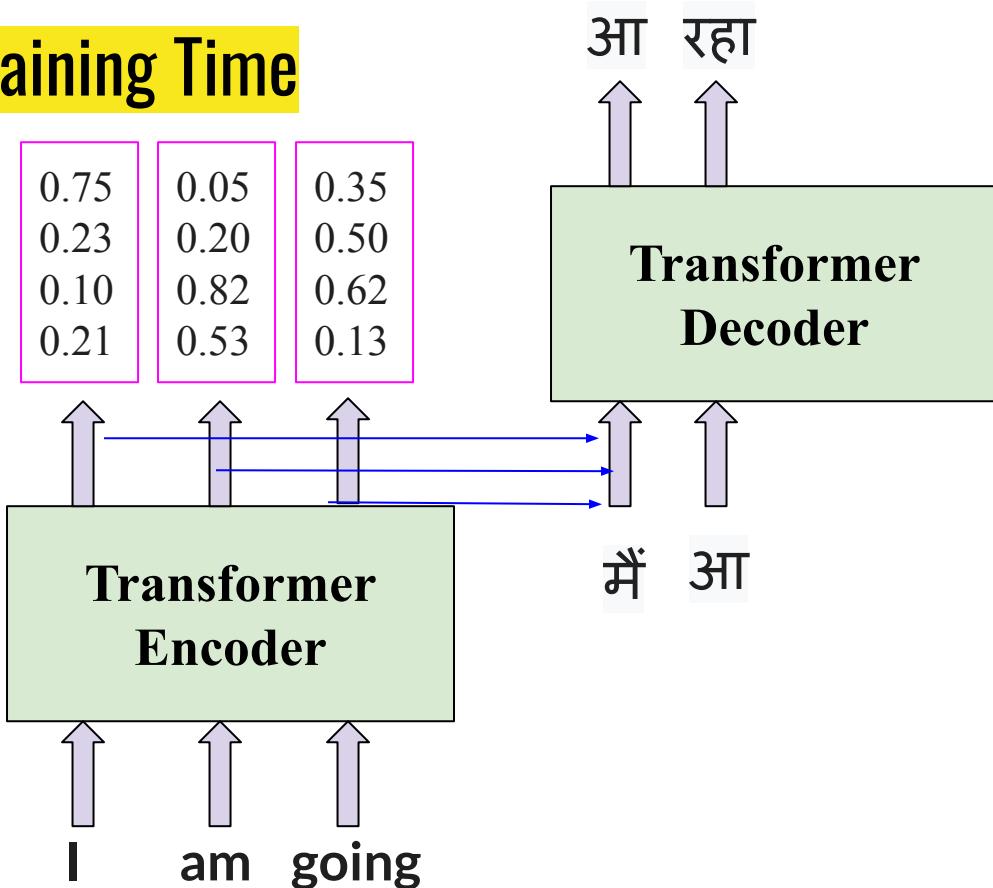


Transformer for MT task: Training Time

English: I am going

Hindi: मैं जा रहा हूँ

Autoregressive Approach



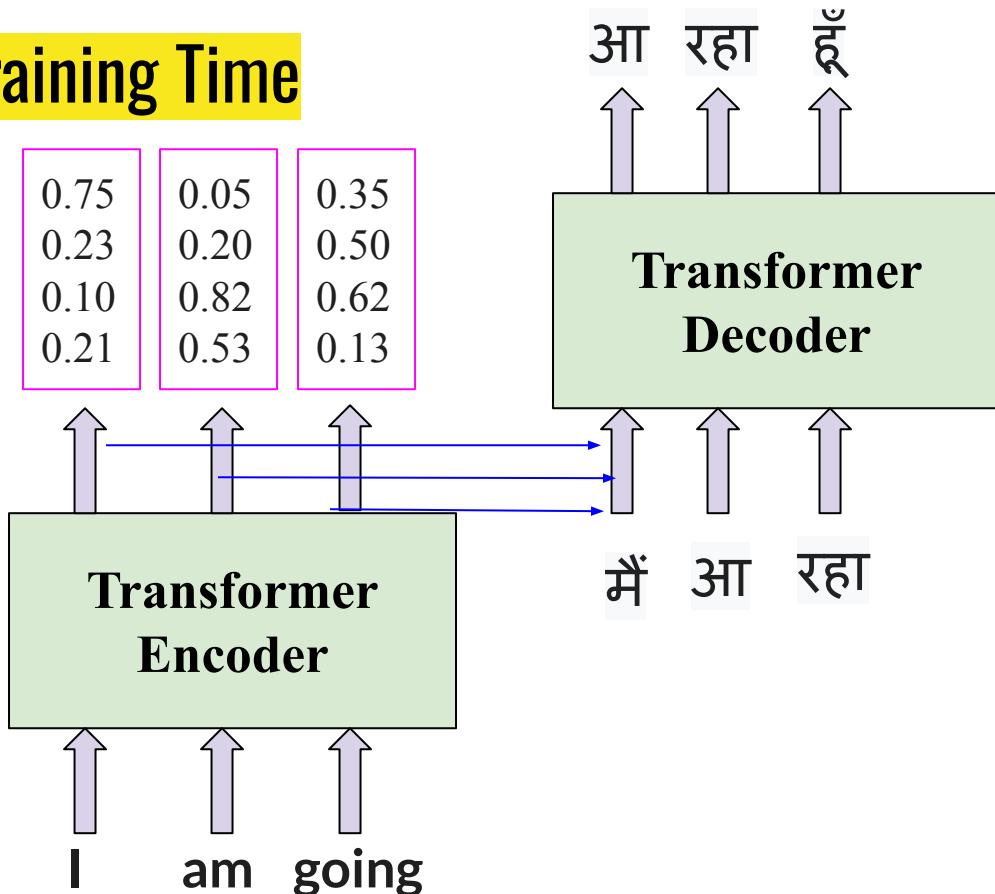
Transformer for MT task: Training Time

Autoregressive Approach Limitations:

1. With autoregressive approach, we will end up feeding wrong input.
2. **Sequential processing of the input:** will take more time for longer sentences.

Solution: non-autoregressive approach

1. Hence we don't have to be dependent on last step output.
2. We can process all inputs in the decoder in parallel.

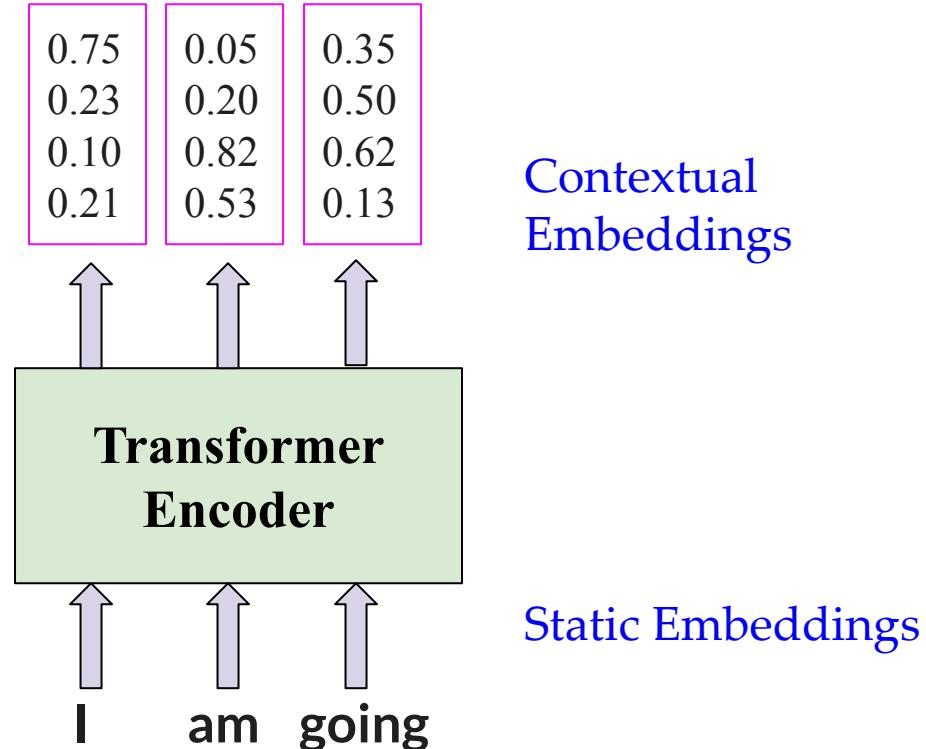


Transformer for MT task: Training Time

Non-Autoregressive Approach

English: I am going

Hindi: मैं जा रहा हूँ



Transformer for MT task: Training Time

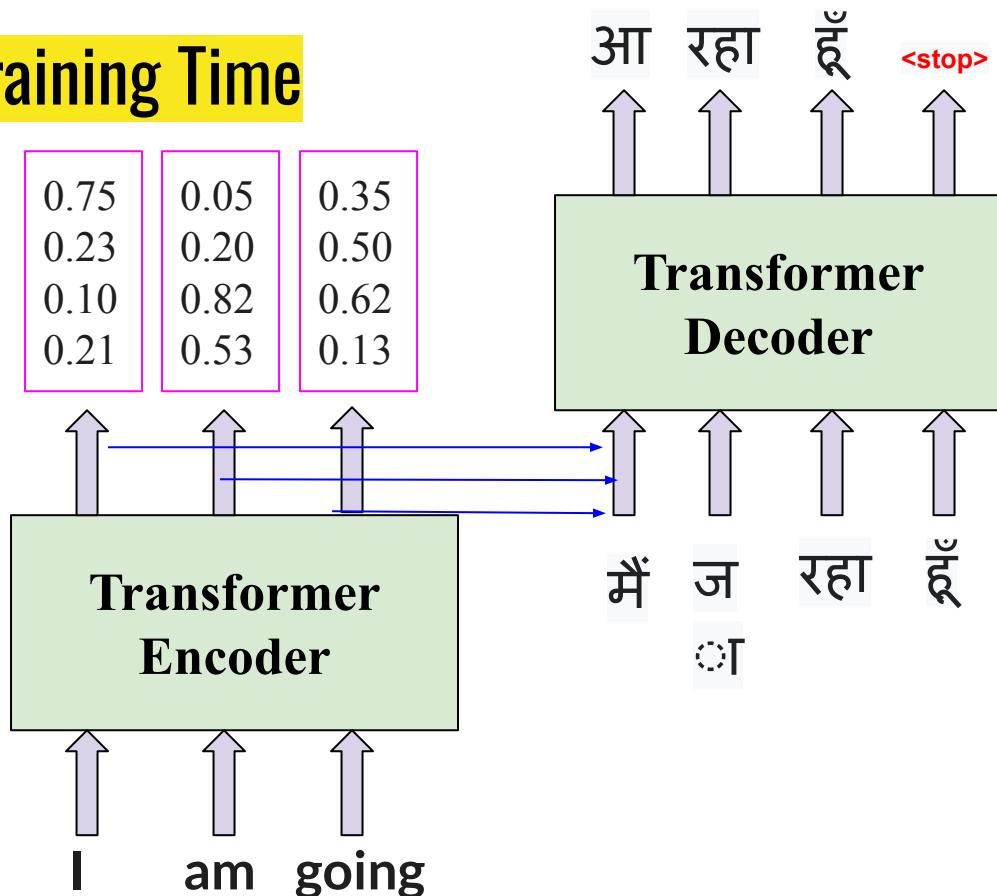
Non-Autoregressive Approach:

1. Teacher Forcing:

- at training time we have parallel corpus available we will feed them as an input.
- There is no dependency on the prediction of decoder.

2. Parallel processing of the input:

All input are processed simultaneously.

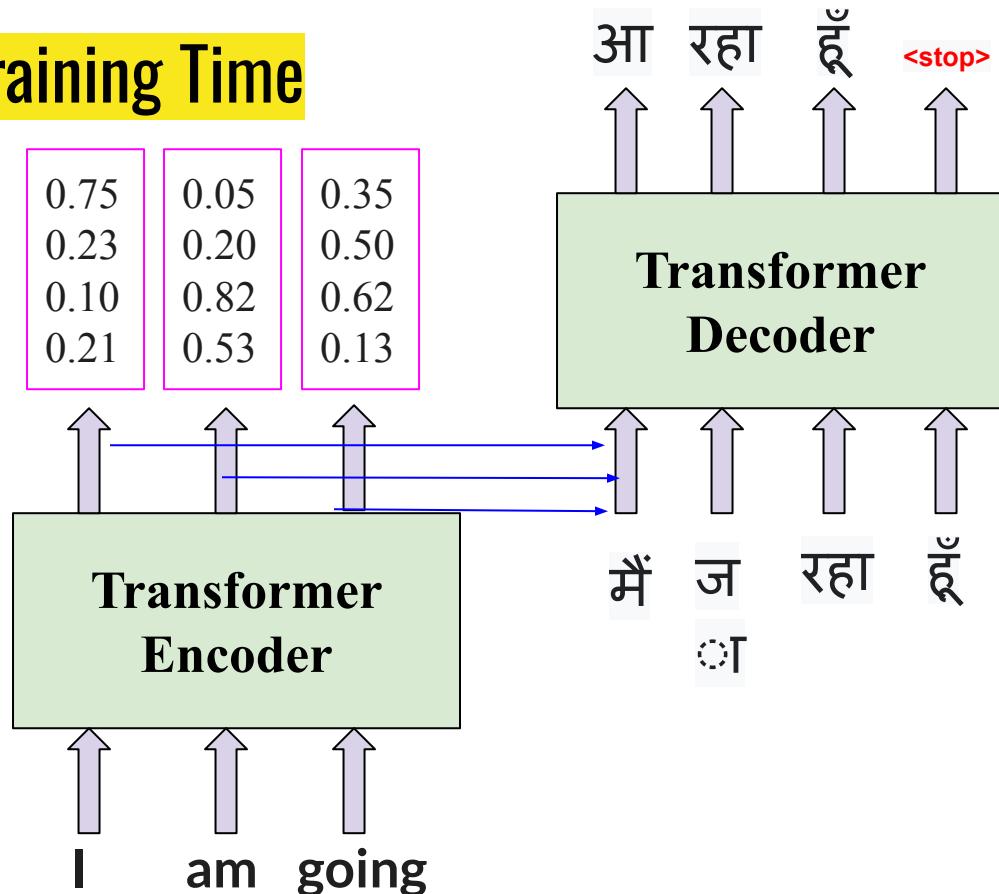


Transformer for MT task: Training Time

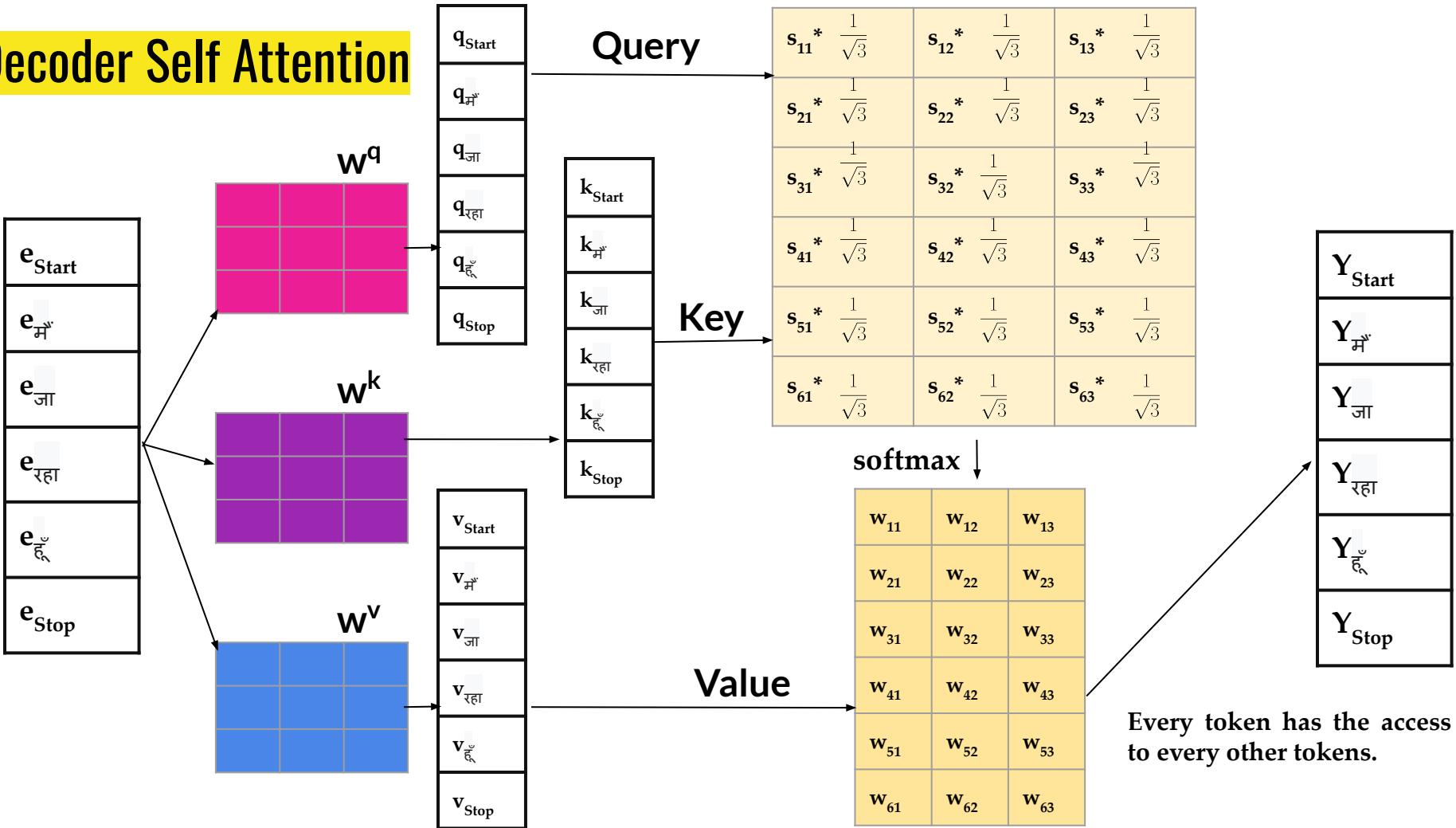
Non-Autoregressive Approach:

1. Teacher Forcing:
 - at training time we have parallel corpus available we will feed them as an input.
 - There is no dependency on the prediction of decoder.
2. Parallel processing of the input:
All input are processed simultaneously.

Limitation: Self attention in the transformer knows the future tokens.



Decoder Self Attention



Decoder Self Attention

Every token has the access to every other tokens while calculating the attention.

	<Start>	मैं	जा	रहा	हूँ	<Stop>
<Start>	0.7	0.2	0.025	0.025	0.025	0.025
मैं	0.025	0.8	0.025	0.025	0.1	0.025
जा	0.05	0.05	0.6	0.2	0.05	0.05
रहा	0.05	0.05	0.2	0.6	0.05	0.05
हूँ	0.075	0.075	0.075	0.075	0.6	0.075
<Stop>	0.02	0.02	0.02	0.02	0.02	0.9

$$\text{मैं} : 0.025 * V_{\text{<Start>}} + 0.8 * V_{\text{मैं}} + 0.025 * V_{\text{जा}} + 0.025 * V_{\text{रहा}} + 0.025 * V_{\text{हूँ}} + 0.025 * V_{\text{<Stop>}}$$

Decoder Self Attention

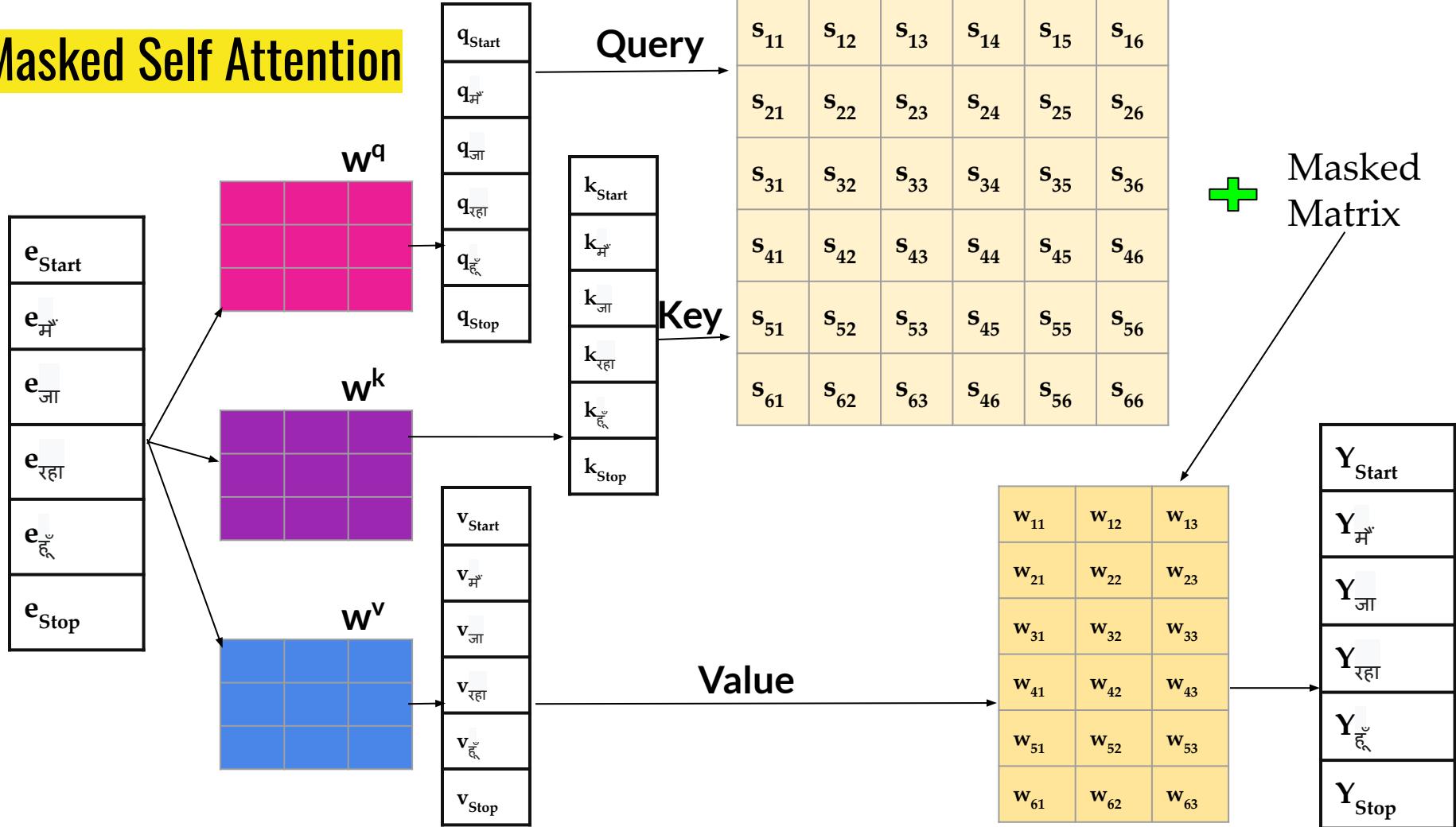
Every token has the access to every other tokens while calculating the attention.

	<Start>	मैं	जा	रहा	हूँ	<Stop>
<Start>	0.7	0	0	0	0	0
मैं	0.025	0.8	0	0	0	0
जा	0.05	0.05	0.6	0	0	0
रहा	0.05	0.05	0.2	0.6	0	0
हूँ	0.075	0.075	0.075	0.075	0.6	0
<Stop>	0.02	0.02	0.02	0.02	0.02	0.9

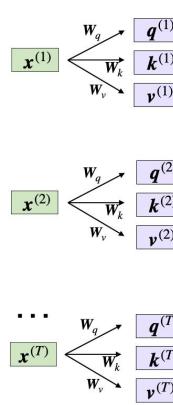
Ideally, a token should not have access to the future tokens.

मैं: $0.025*V_{<\text{Start}>} + 0.8*V_{\text{मैं}} + 0*V_{\text{जा}} + 0*V_{\text{रहा}} + 0*V_{\text{हूँ}} + 0*V_{<\text{Stop}>}$

Masked Self Attention



Masked Self Attention



s_{11}	s_{12}	s_{13}	s_{14}	s_{15}	s_{16}
s_{21}	s_{22}	s_{23}	s_{24}	s_{25}	s_{26}
s_{31}	s_{32}	s_{33}	s_{34}	s_{35}	s_{36}
s_{41}	s_{42}	s_{43}	s_{44}	s_{45}	s_{46}
s_{51}	s_{52}	s_{53}	s_{45}	s_{55}	s_{56}
s_{61}	s_{62}	s_{63}	s_{46}	s_{56}	s_{66}

$$S = \frac{QK^T}{\sqrt{d_k}}$$

$$S_{\text{masked}} = S + M$$

s_{11}'	s_{12}'	s_{13}'	s_{14}'	s_{15}'	s_{16}'	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
s_{21}'	s_{22}'	s_{23}'	s_{24}'	s_{25}	s_{26}'	0	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$
s_{31}'	s_{32}'	s_{33}'	s_{34}'	s_{35}	s_{36}'	0	0	0	$-\infty$	$-\infty$	$-\infty$
s_{41}'	s_{42}'	s_{43}'	s_{44}'	s_{45}	s_{46}'	0	0	0	0	$-\infty$	$-\infty$
s_{51}'	s_{52}'	s_{53}'	s_{45}'	s_{55}'	s_{56}'	0	0	0	0	0	$-\infty$
s_{61}'	s_{62}'	s_{63}'	s_{46}'	s_{56}'	s_{66}'	0	0	0	0	0	0

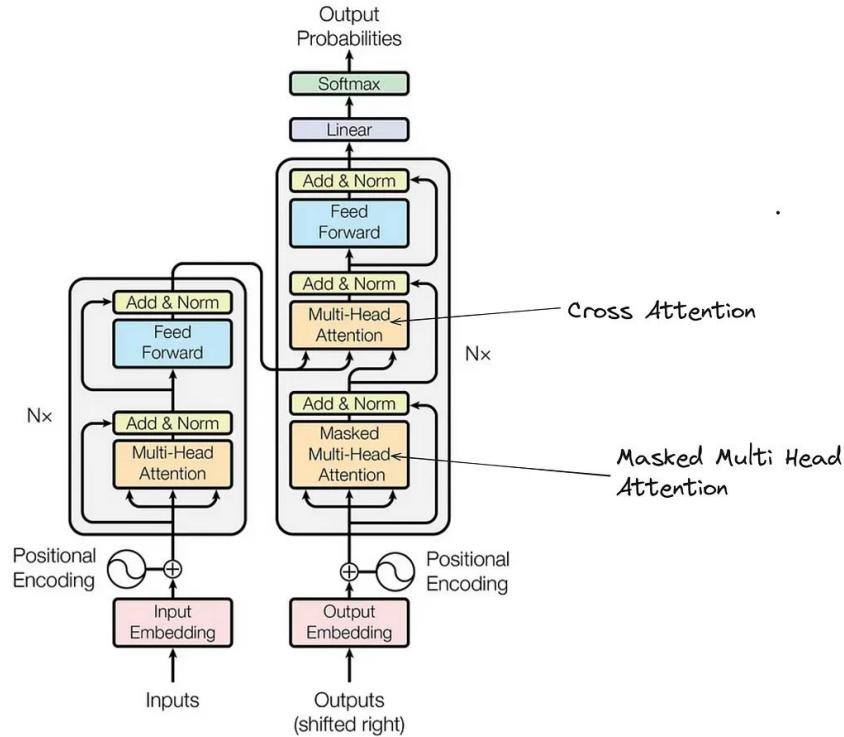
$$A = \text{softmax}(S_{\text{masked}})$$

w_{11}	0	0	0	0	0	s_{11}'	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$
w_{21}	w_{22}	0	0	0	0	s_{21}'	s_{22}'	$-\infty$	$-\infty$	$-\infty$	$-\infty$
w_{31}	w_{32}	w_{33}	0	0	0	s_{31}'	s_{32}'	s_{33}'	$-\infty$	$-\infty$	$-\infty$
w_{41}	w_{42}	w_{43}	w_{44}	0	0	s_{41}'	s_{42}'	s_{43}'	s_{44}'	$-\infty$	$-\infty$
w_{51}	w_{52}	w_{53}	w_{45}	w_{55}	0	s_{51}'	s_{52}'	s_{53}'	s_{45}'	s_{55}'	$-\infty$
w_{61}	w_{62}	w_{63}	w_{46}	w_{56}	w_{66}	s_{61}'	s_{62}'	s_{63}'	s_{46}'	s_{56}'	s_{66}'

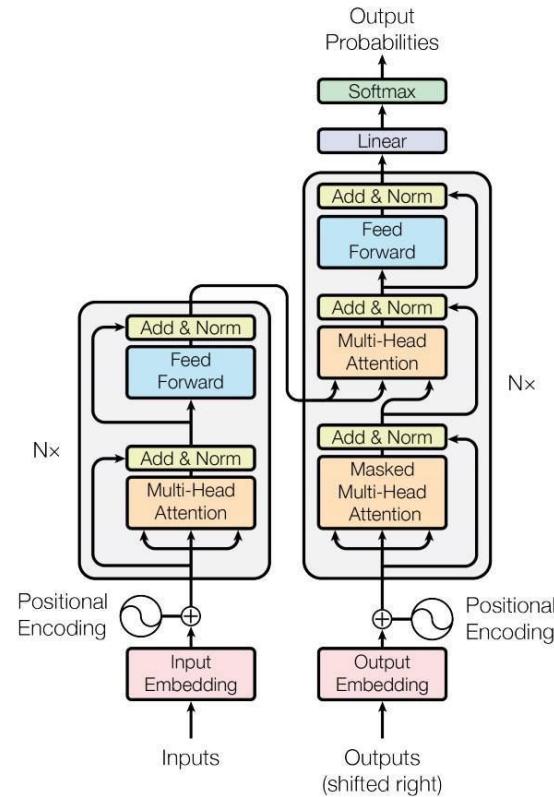
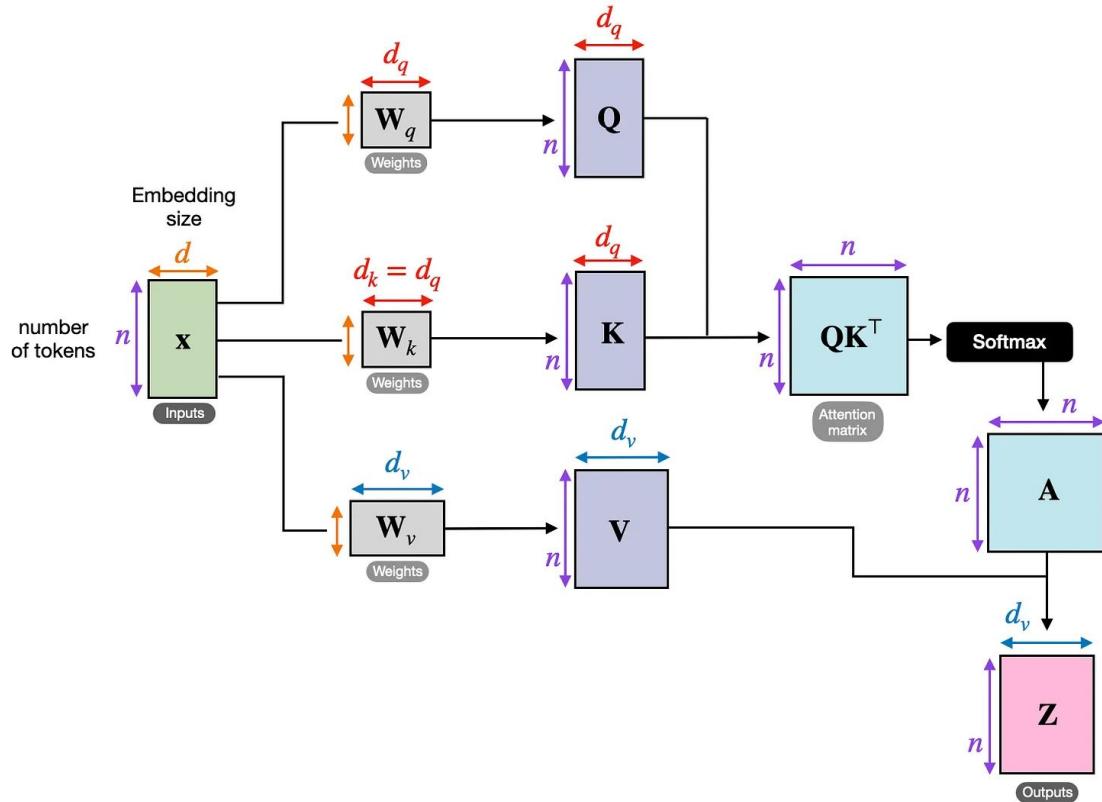
$$\text{Output} = AV$$

$$A = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} + M \right) V$$

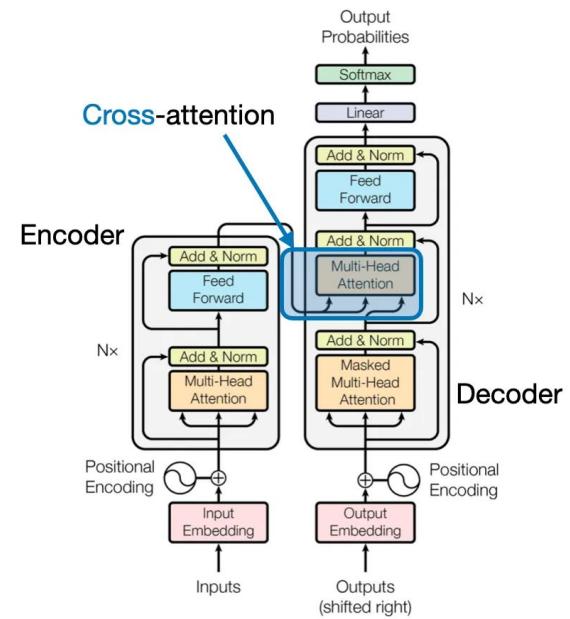
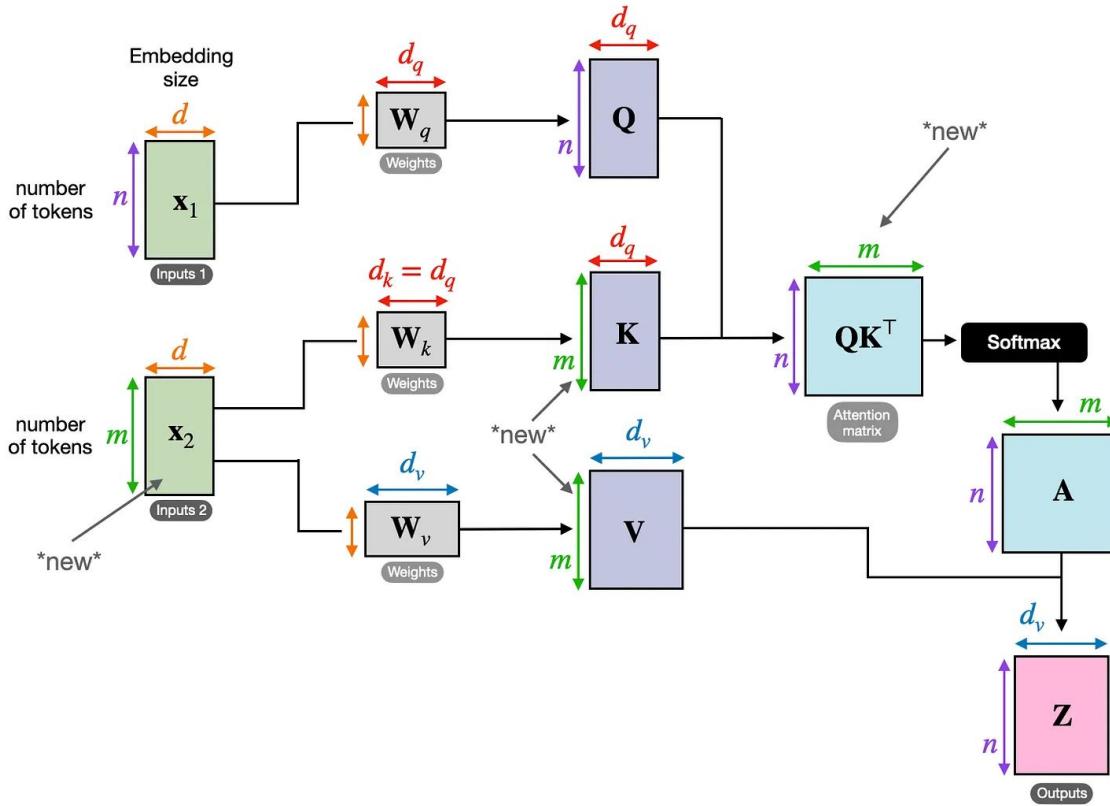
Transformer Decoder Architecture



Self Attention in Encoder



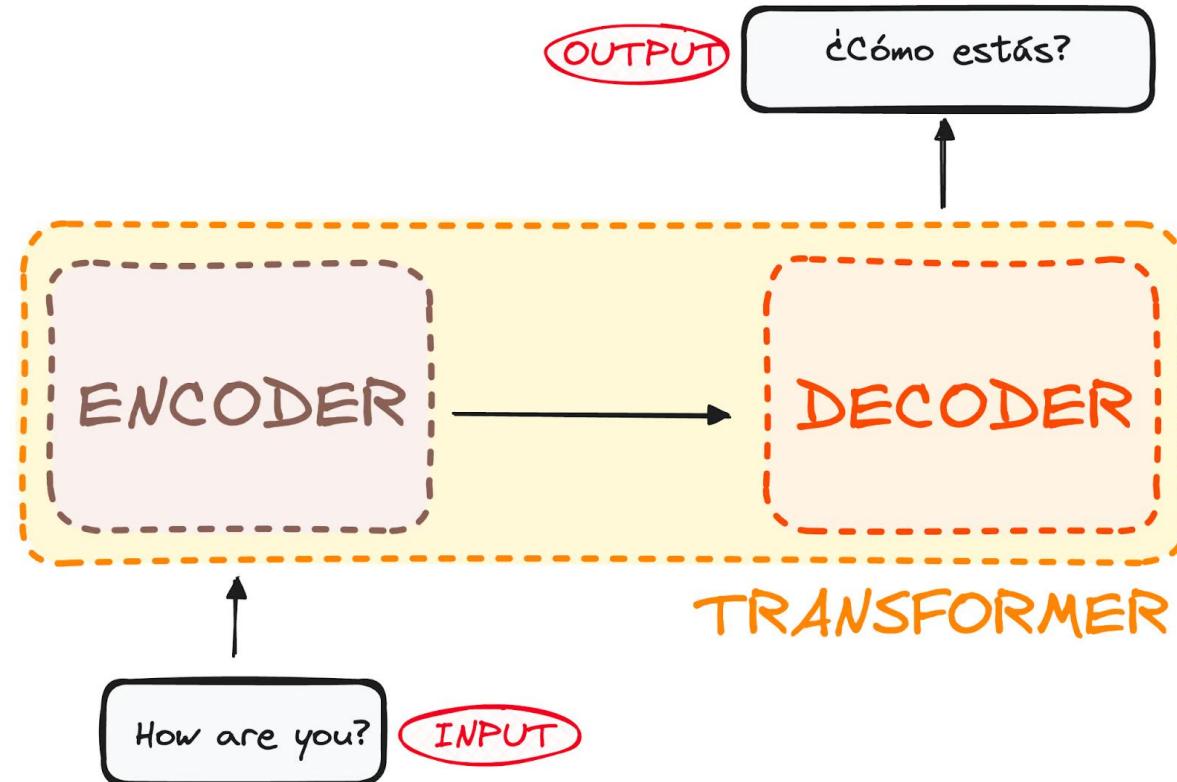
Self Attention in Decoder: Cross Attention



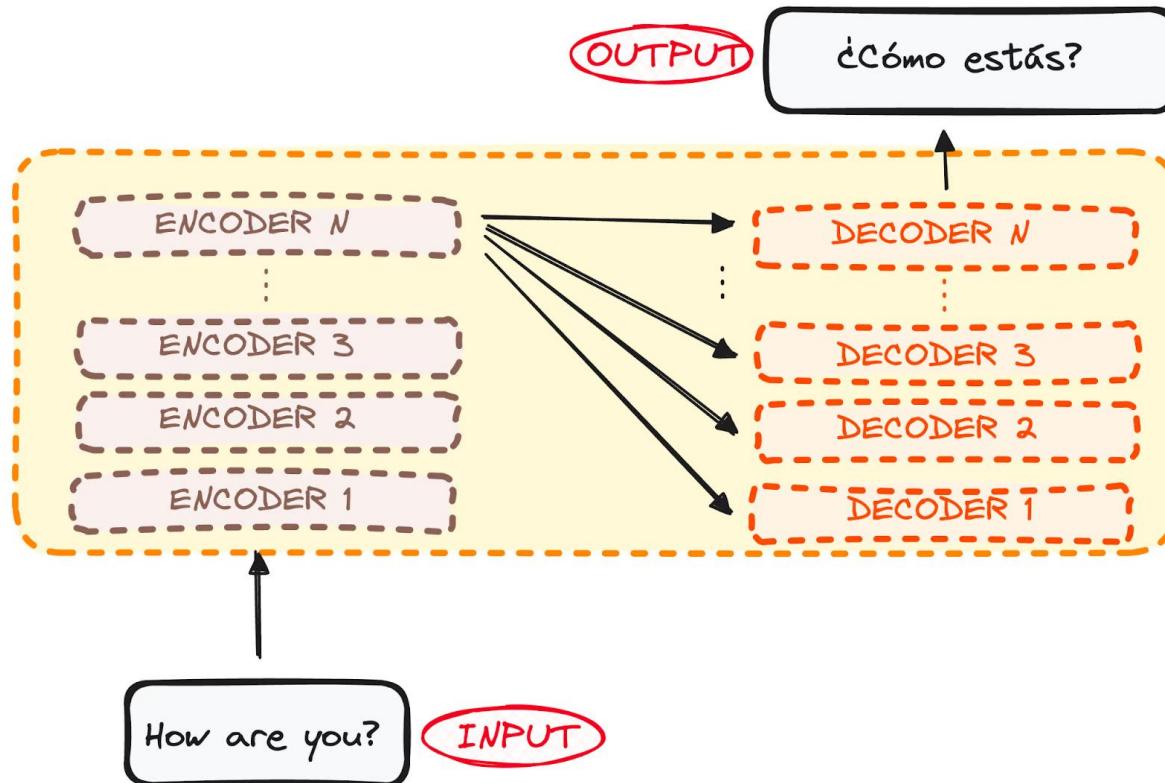
Transformer Architecture



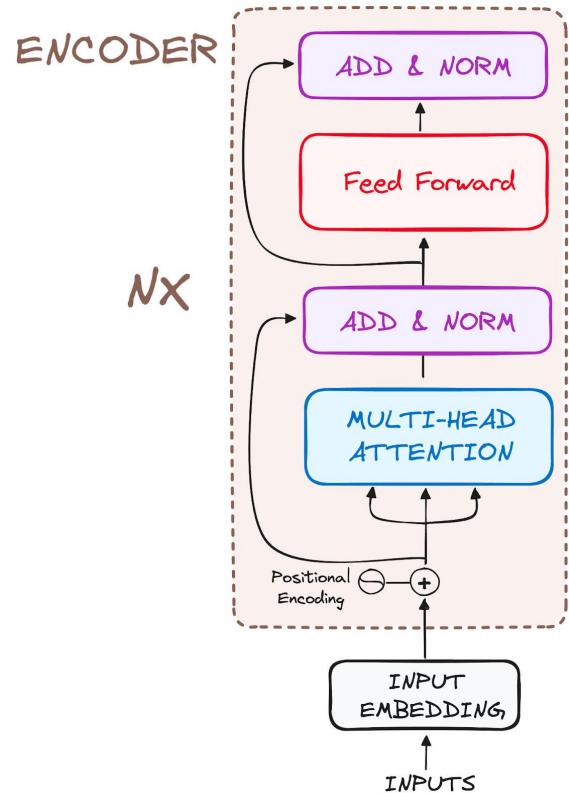
Transformer Architecture



Transformer Architecture

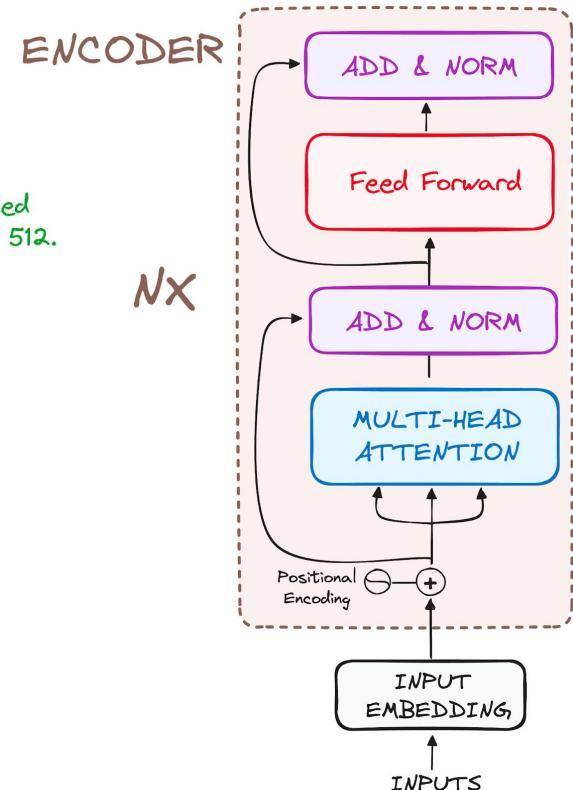
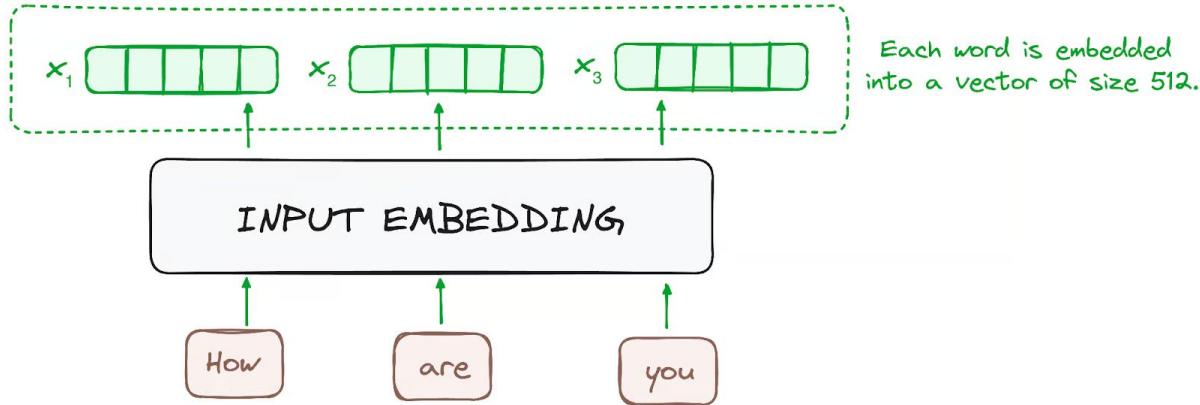


Transformer Architecture: Encoder workflow



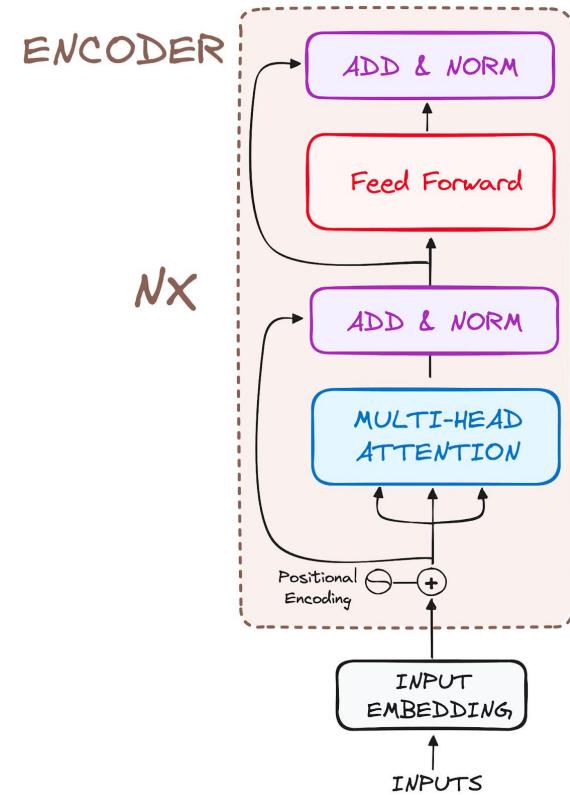
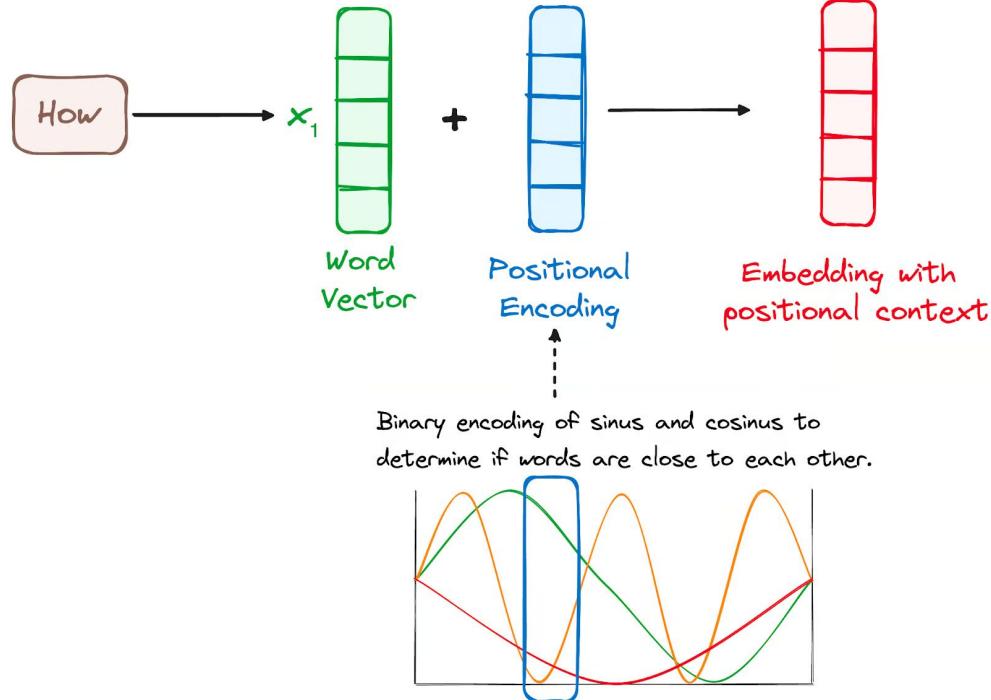
Transformer Architecture: Encoder workflow

Step 1: Input Embeddings



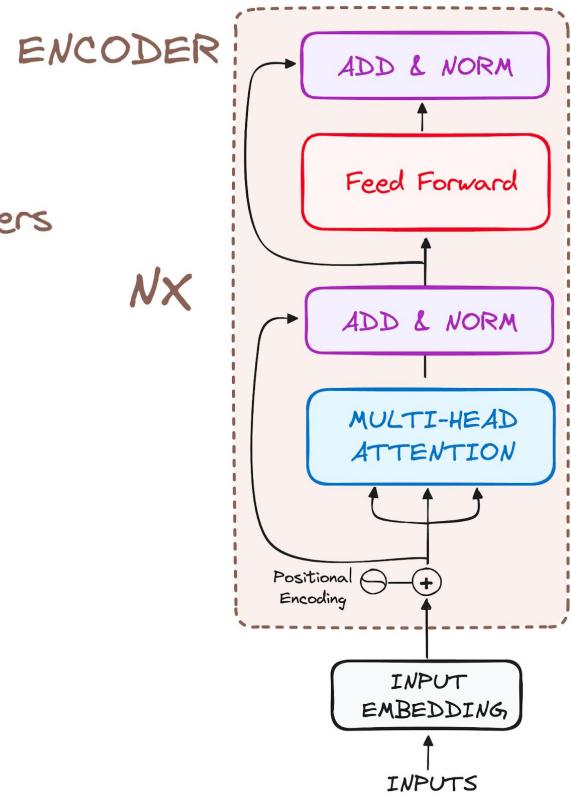
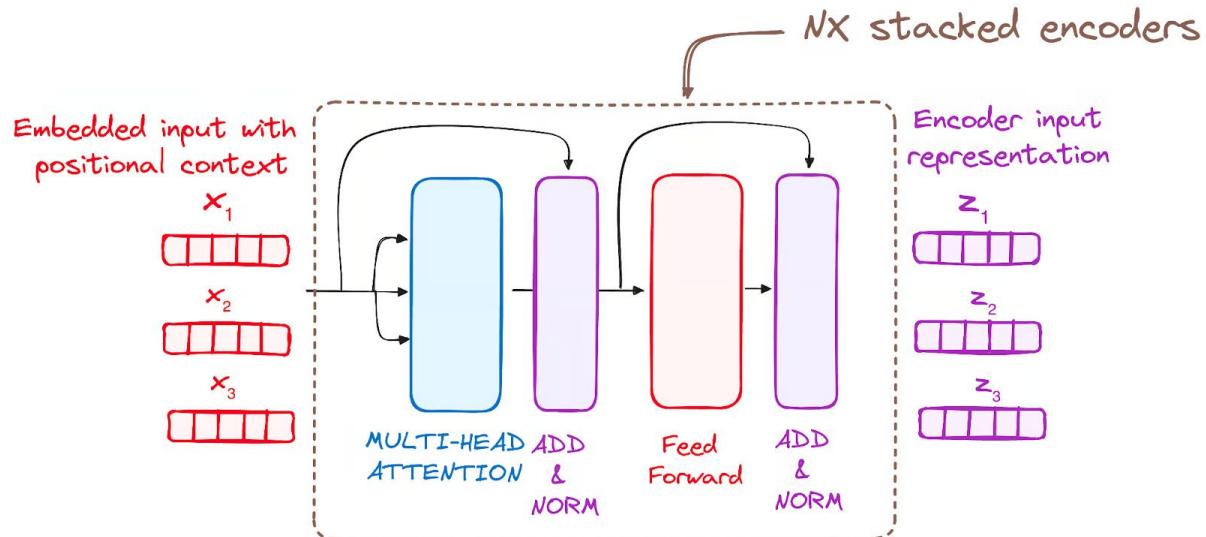
Transformer Architecture: Encoder workflow

Step 2: Positional Encoding



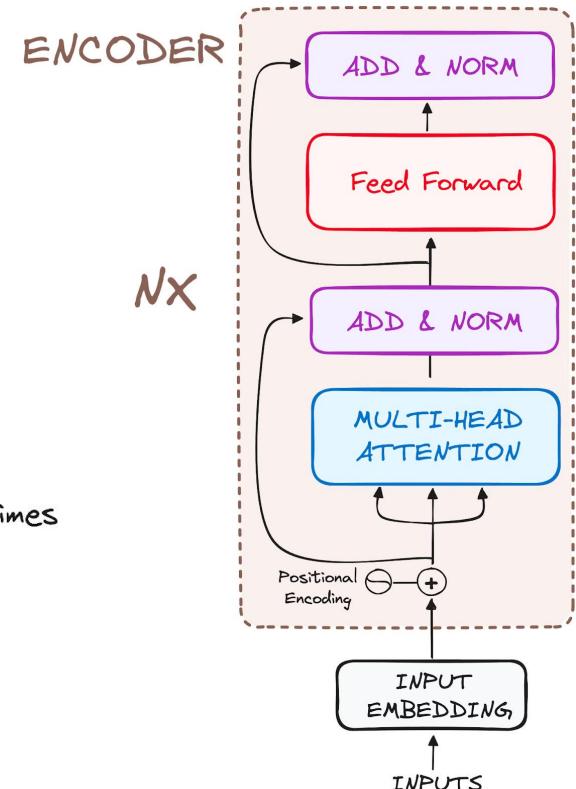
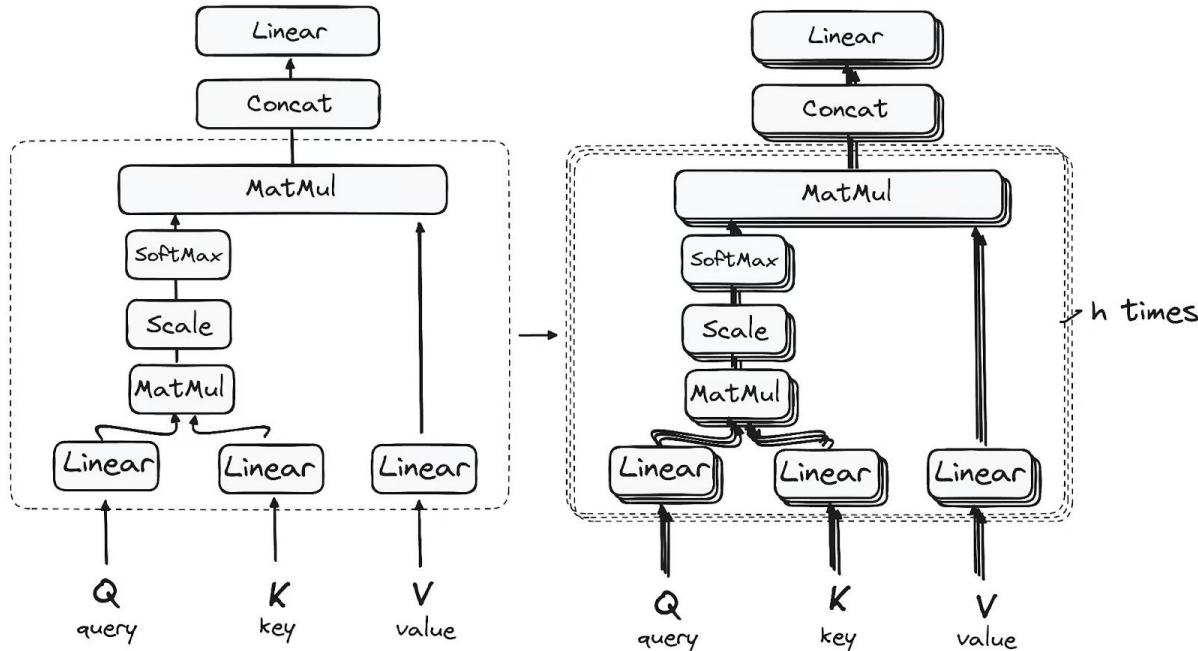
Transformer Architecture: Encoder workflow

Step 3: Stack of Encoder Layers



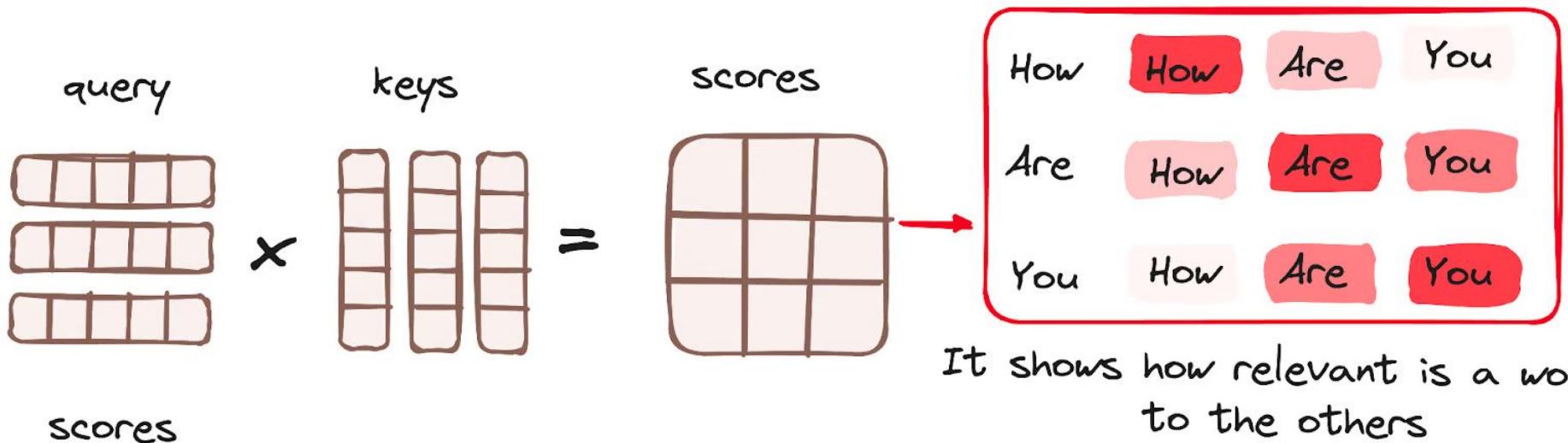
Transformer Architecture: Encoder workflow

Step 3.1: Multi-Headed Self-Attention Mechanism



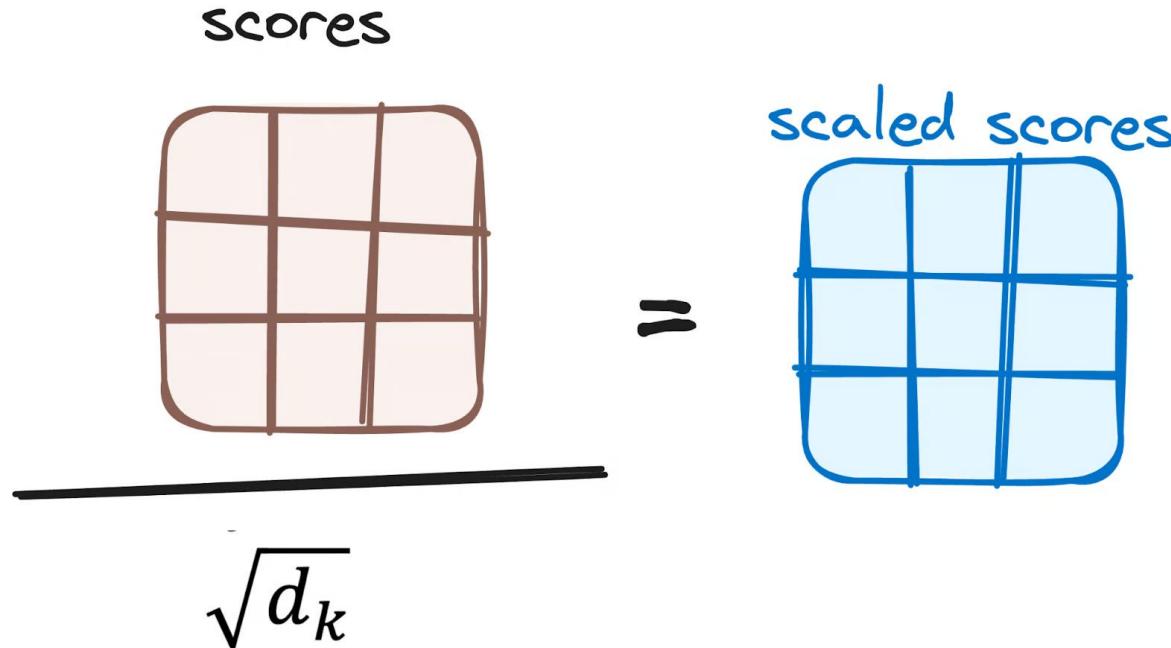
Transformer Architecture: Encoder workflow

Step 3.1: Multi-Headed Self-Attention Mechanism Dot Product of Query and Key



Transformer Architecture: Encoder workflow

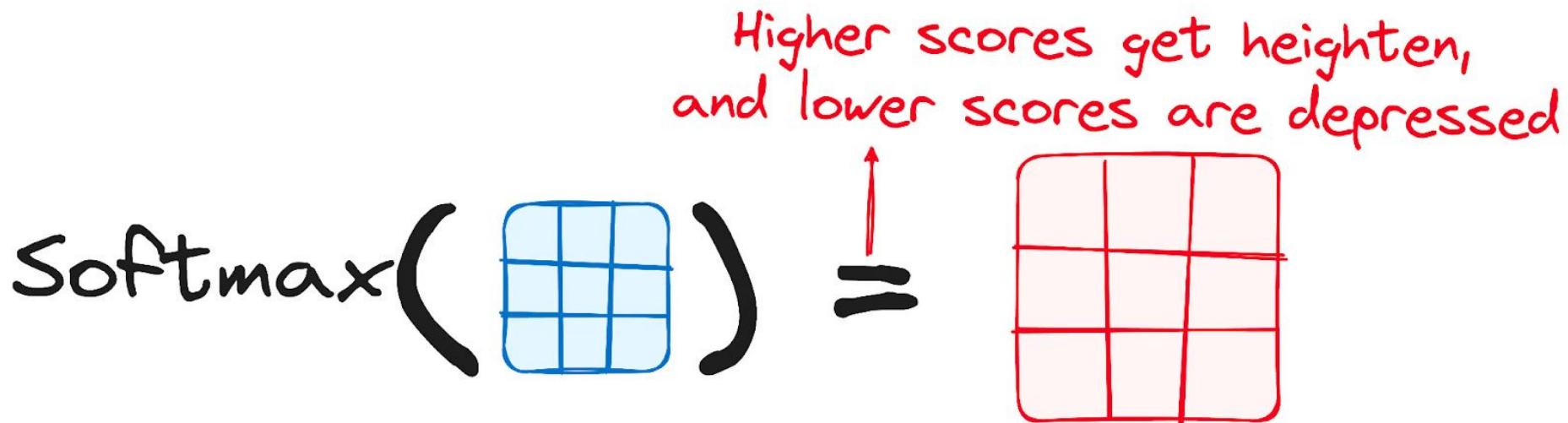
Step 3.1: Multi-Headed Self-Attention Mechanism Scaled Scores



Transformer Architecture: Encoder workflow

Step 3.1: Multi-Headed Self-Attention Mechanism

Higher scores get heightened,
and lower scores are depressed

$$\text{Softmax} \left(\begin{matrix} & & \\ & & \\ & & \\ & & \\ & & \\ & & \end{matrix} \right) = \begin{matrix} & & \\ & & \\ & & \\ & & \\ & & \\ & & \end{matrix}$$


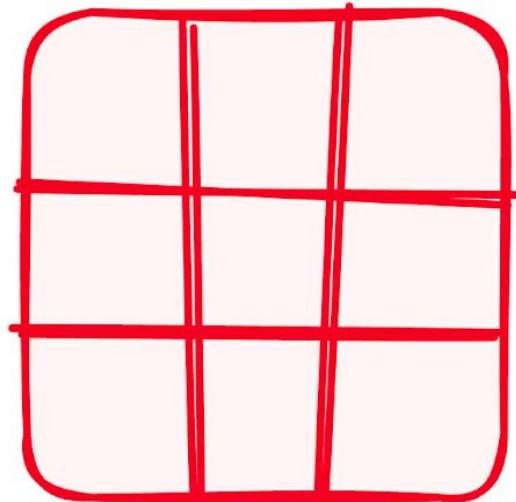
0.1	0.2	0.3	0.4	0.5	0.6
0.2	0.4	0.6	0.8	1.0	0.9
0.3	0.6	0.9	1.2	1.5	1.3
0.4	0.8	1.2	1.6	1.8	1.7
0.5	1.0	1.5	1.8	2.0	1.9
0.6	0.9	1.3	1.7	1.9	1.8

0.1	0.2	0.3	0.4	0.5	0.6
0.2	0.4	0.6	0.8	1.0	0.9
0.3	0.6	0.9	1.2	1.5	1.3
0.4	0.8	1.2	1.6	1.8	1.7
0.5	1.0	1.5	1.8	2.0	1.9
0.6	0.9	1.3	1.7	1.9	1.8

Transformer Architecture: Encoder workflow

Step 3.1: Multi-Headed Self-Attention Mechanism

Attention weights



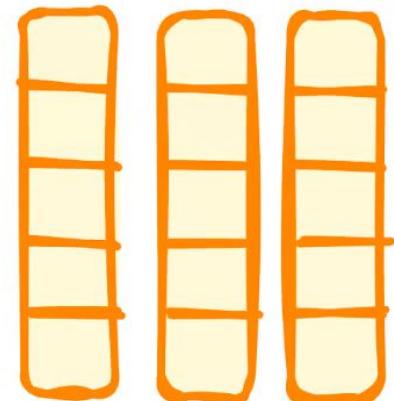
\times

Values



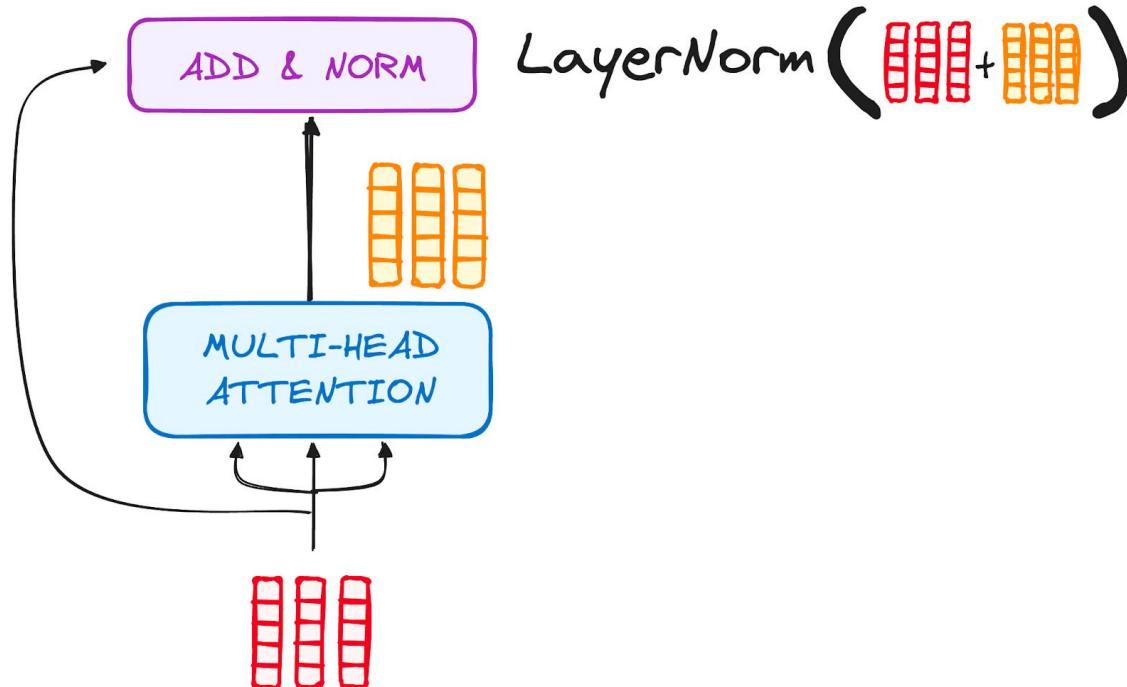
=

Output



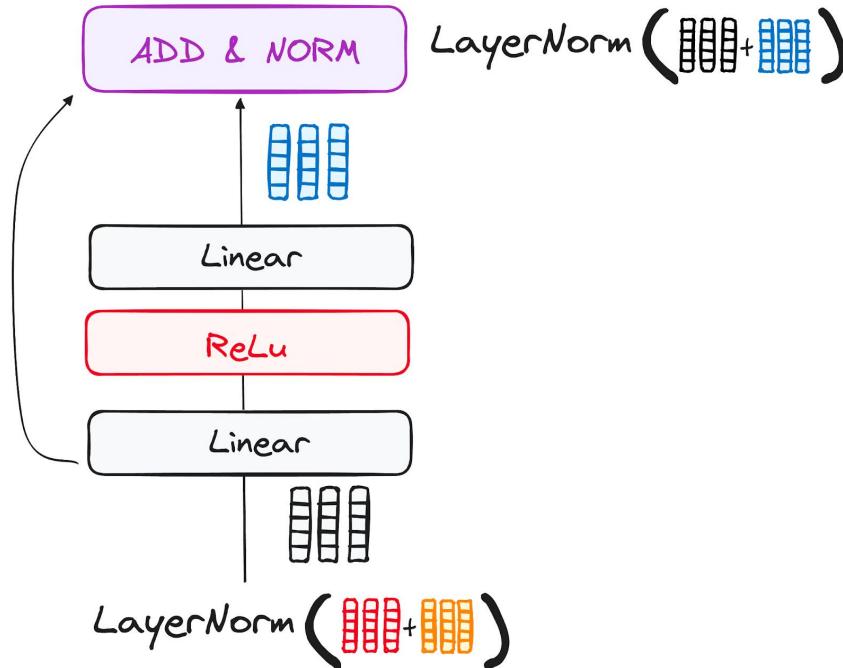
Transformer Architecture: Encoder workflow

Step 3.2: Normalization and Residual Connections



Transformer Architecture: Encoder workflow

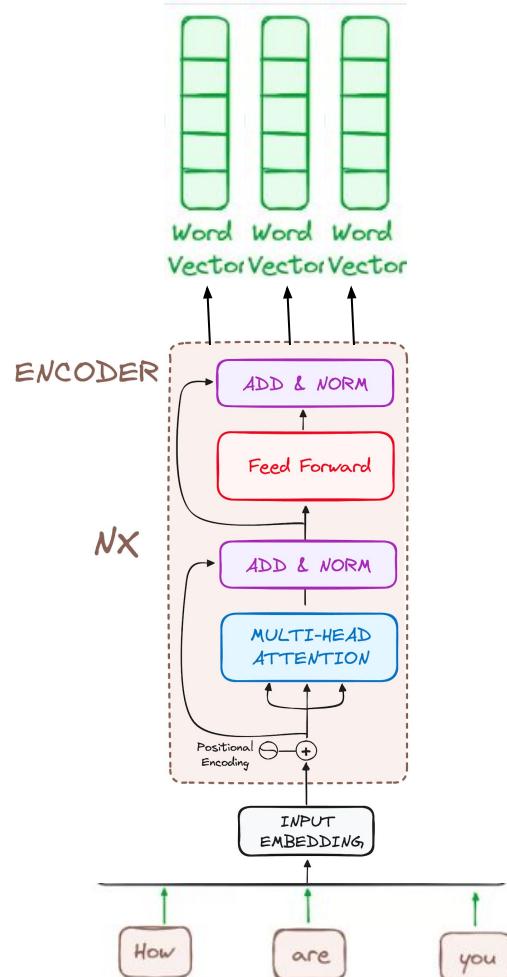
Step 3.2: Feed-Forward Neural Network



Transformer Architecture: Encoder workflow

Step 4: Encoder Output

- The output of the final encoder layer is a set of vectors, each representing the input sequence with a rich contextual understanding.
- This output is then used as the input for the decoder in a Transformer model.



Transformer Architecture: Decoder workflow

Step 1: Output embeddings

Step 2: Positional Encoding

Step 3: Stack of decoder Layers

3.1: Masked Self-Attention Mechanism

scaled scores

0.5	0.2	0.1
0.1	0.6	0.2
0.1	0.2	0.3

Look-ahead mask

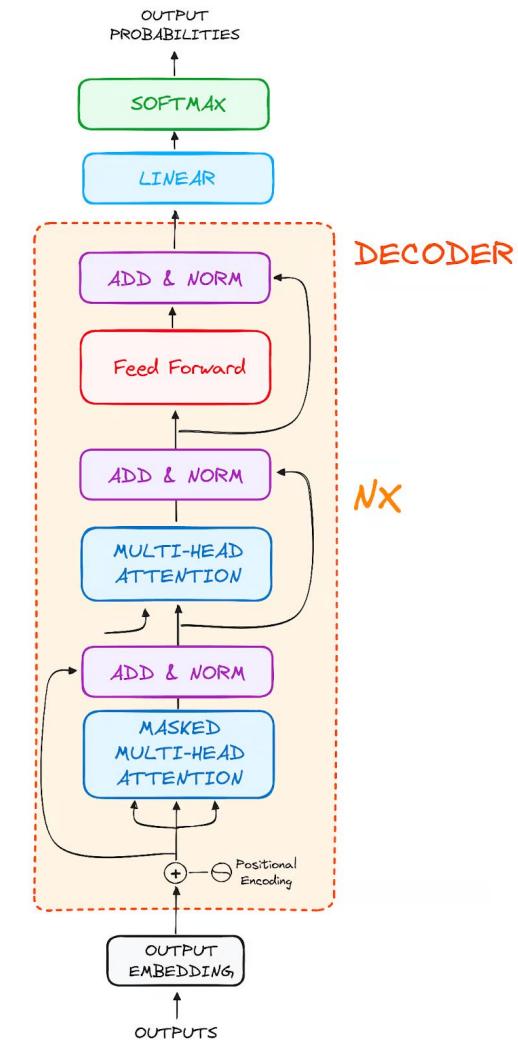
0	-inf	-inf
0	0	-inf
0	0	0

+

=

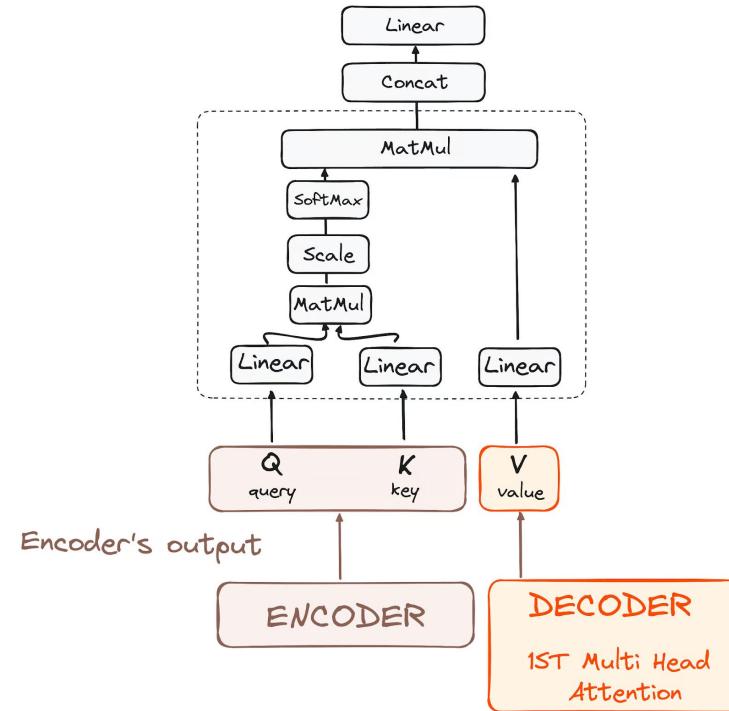
Masked Scores

0.5	-inf	-inf
0.1	0.6	-inf
0.1	0.2	0.3

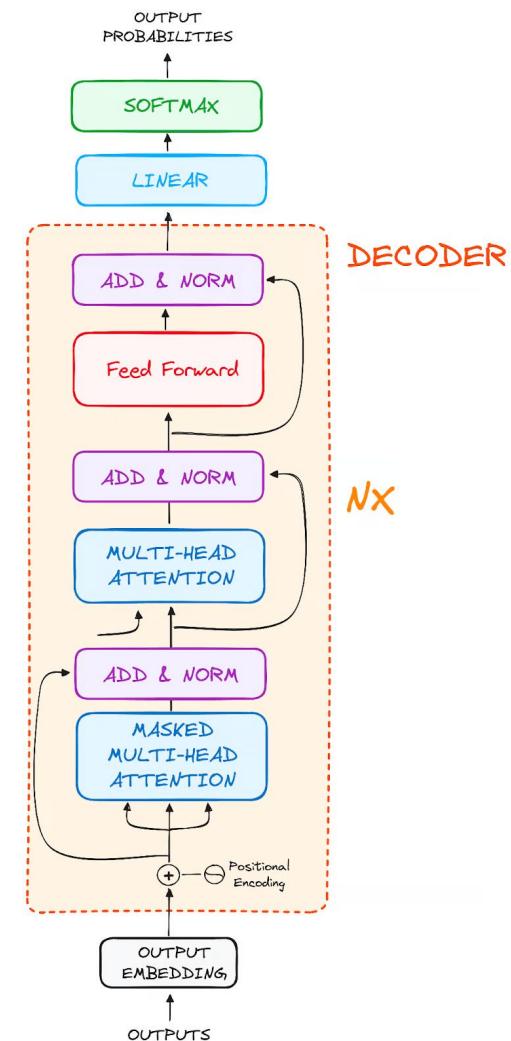


Transformer Architecture: Decoder workflow

3.2: Cross Attention

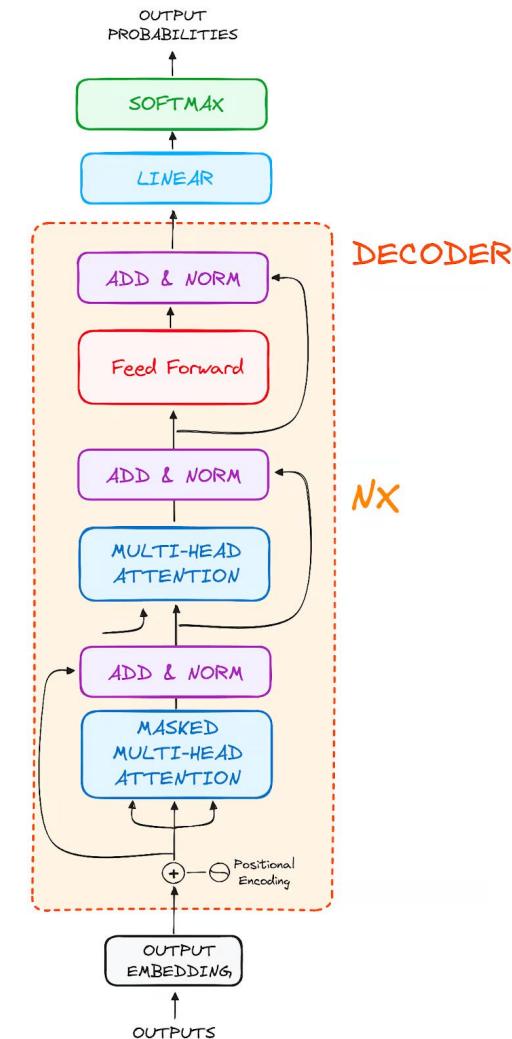
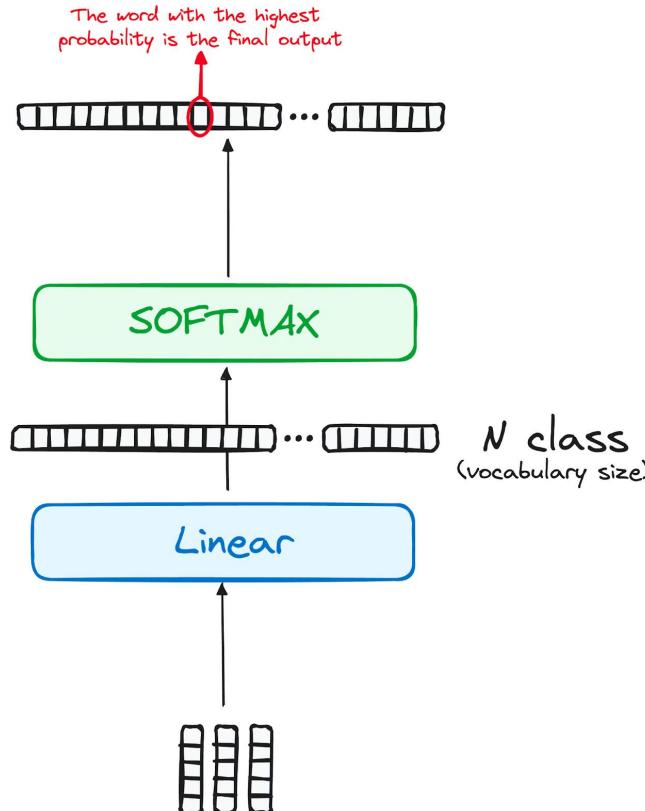


3.3: FF Neural Network

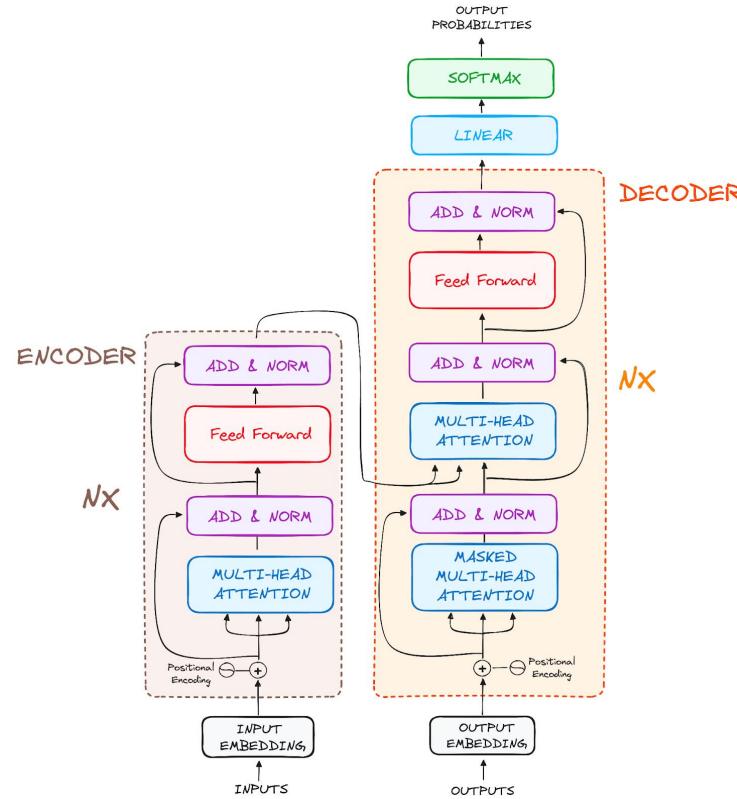


Transformer Architecture: Decoder workflow

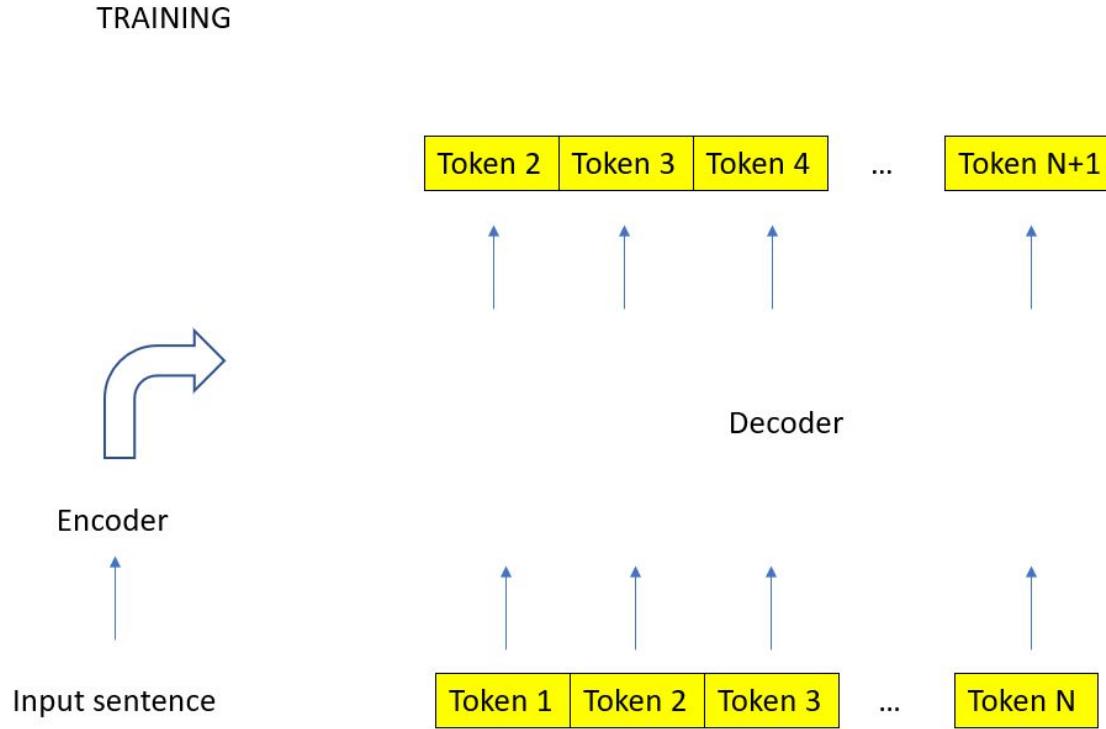
Step 4: Linear Classifier



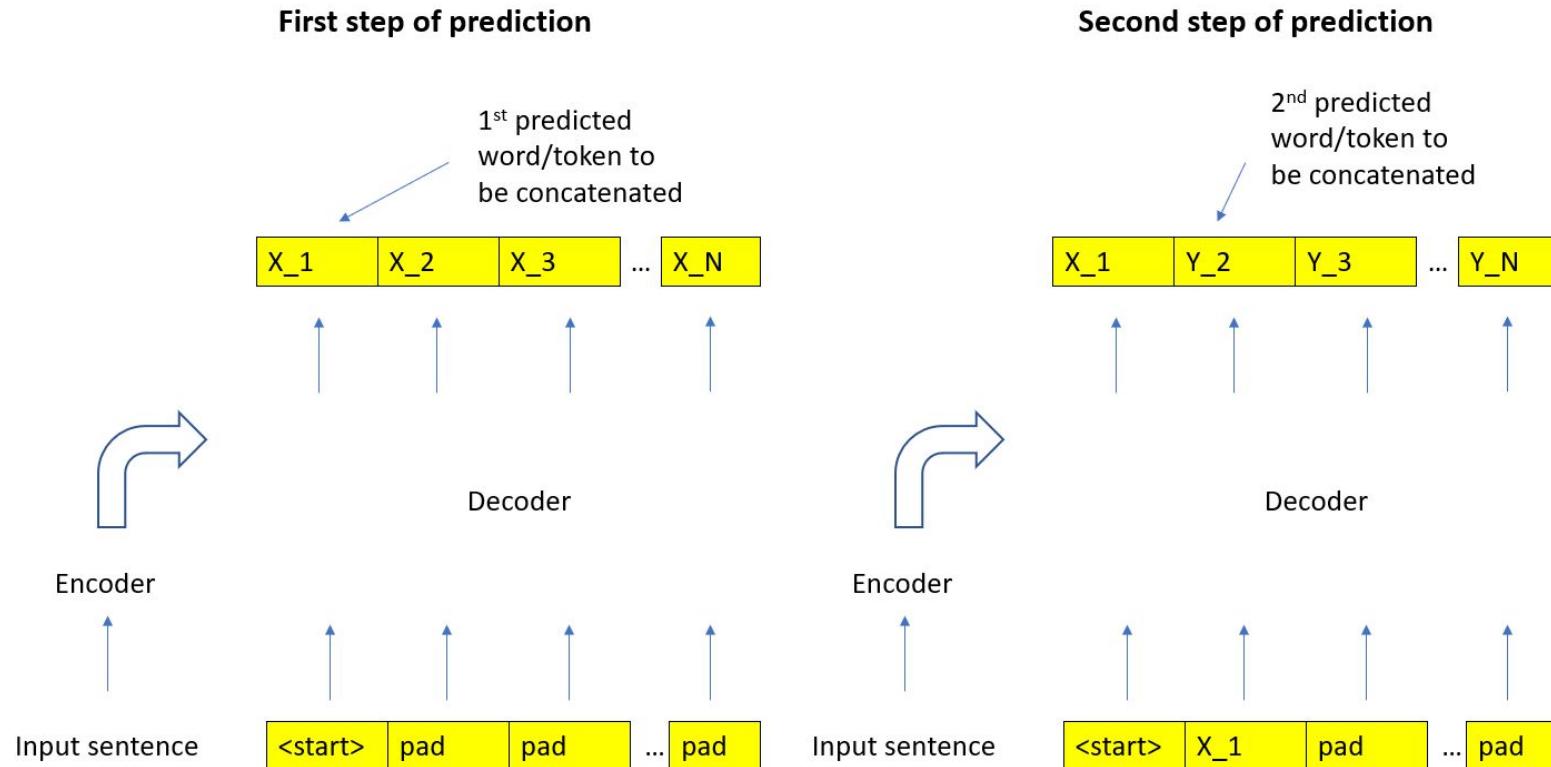
Transformer Architecture: Encoder-Decoder workflow



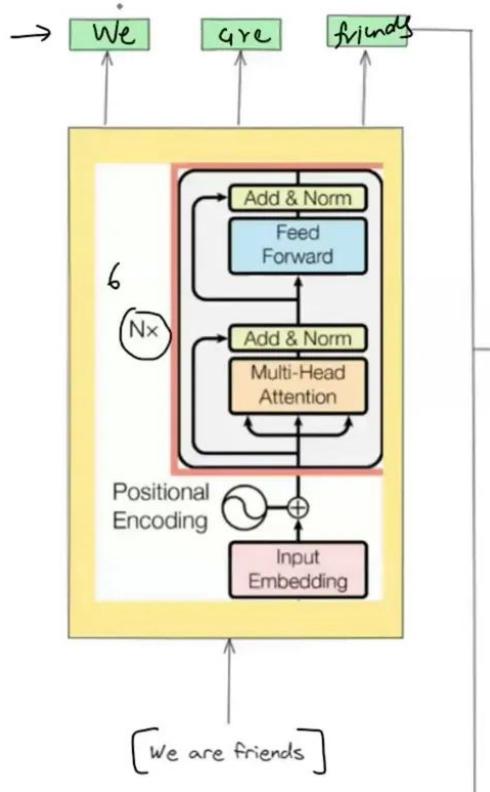
Transformer Architecture: Encoder-Decoder Training



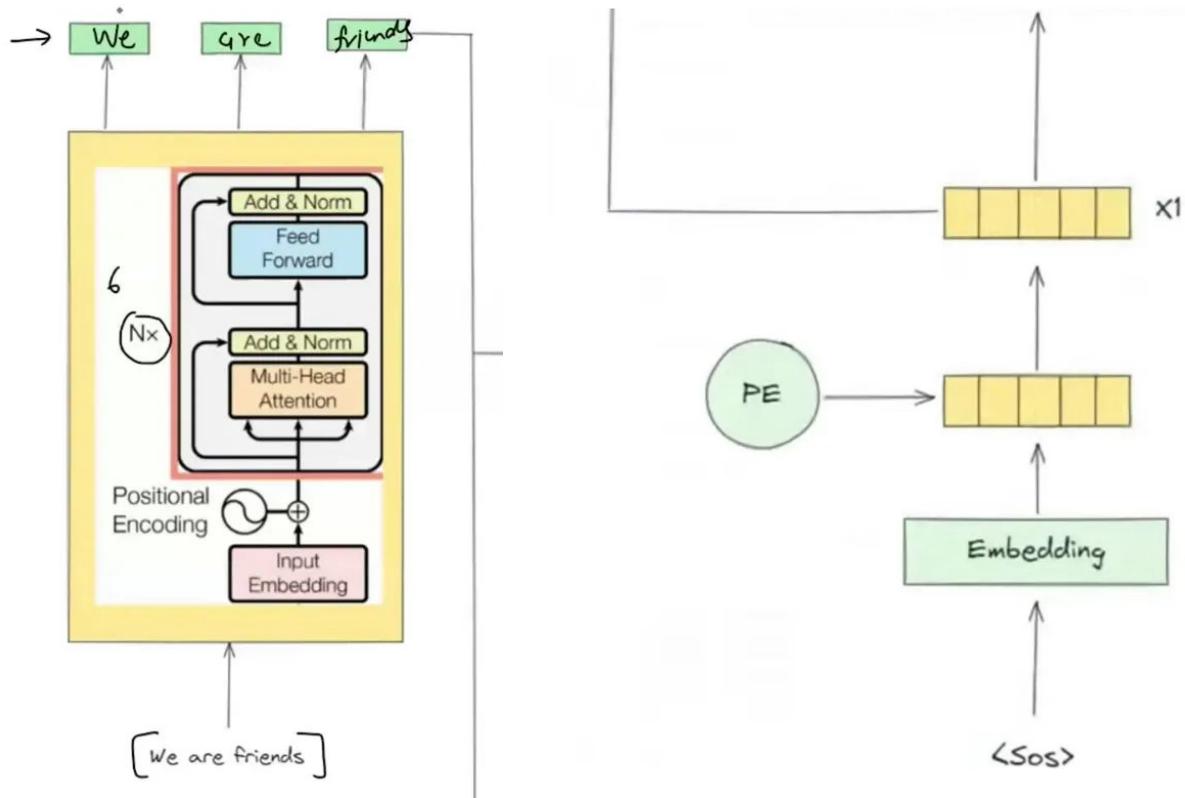
Transformer Architecture: Encoder-Decoder Inference



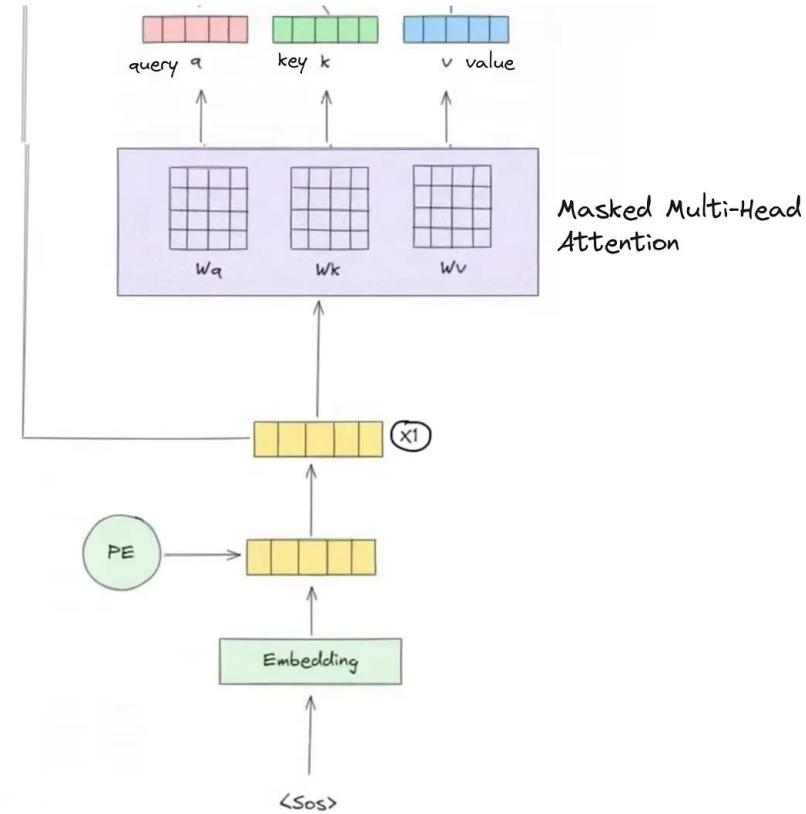
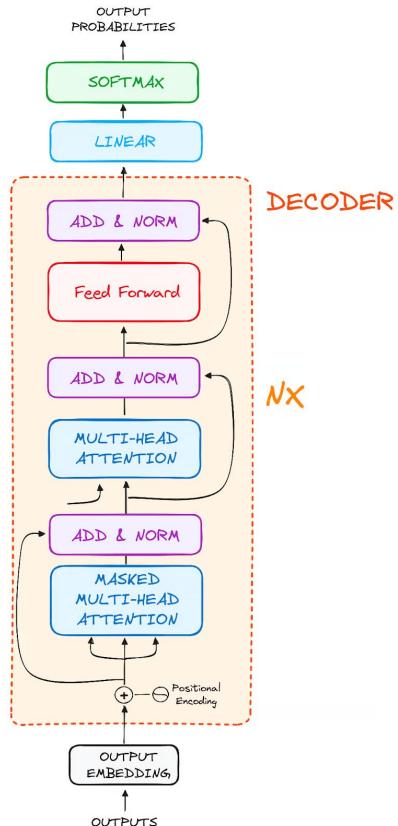
Transformer Architecture: Inference



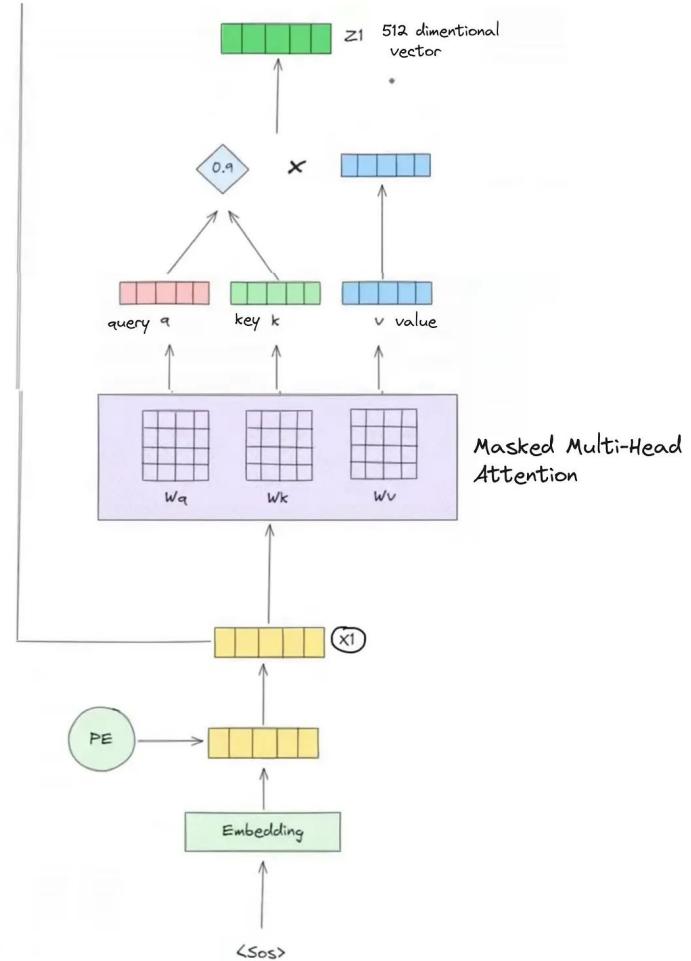
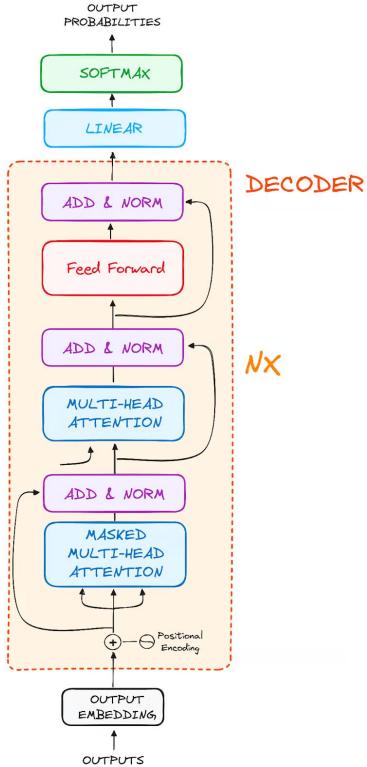
Transformer Architecture: Inference



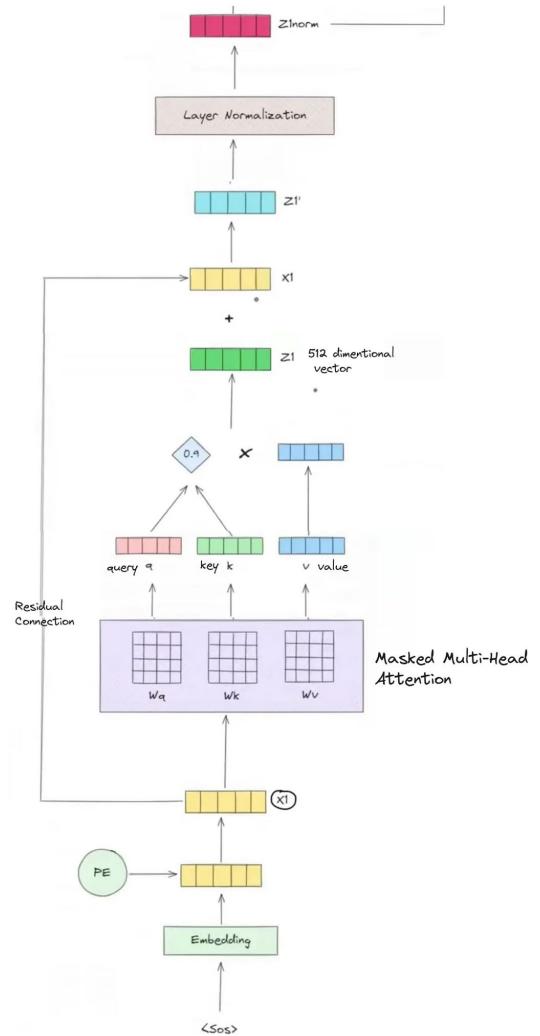
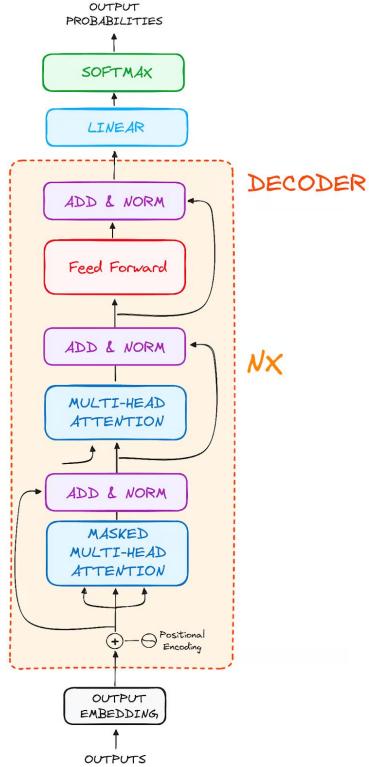
Transformer Architecture: Inference



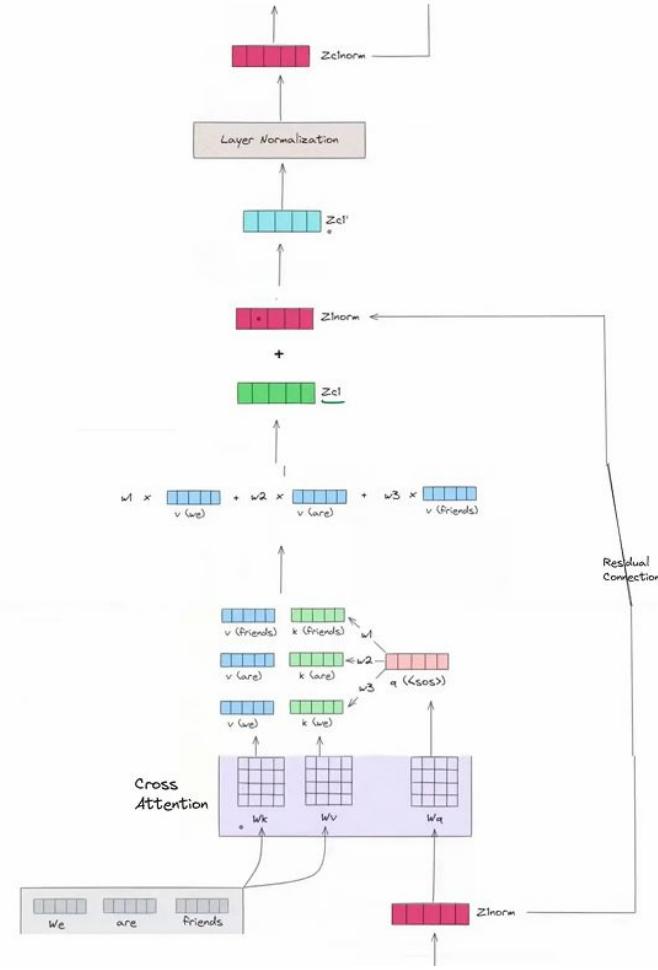
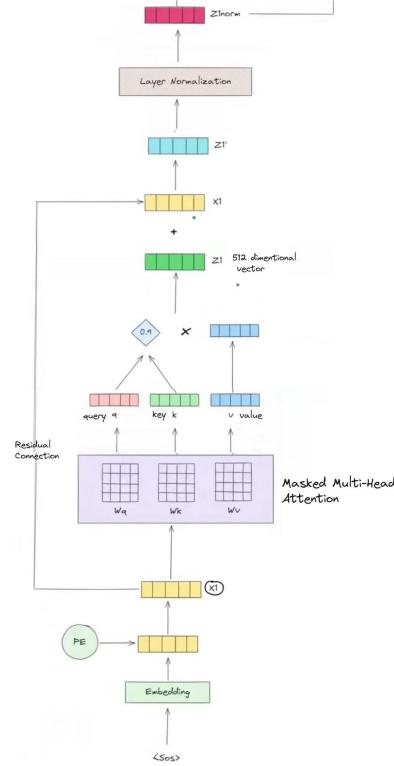
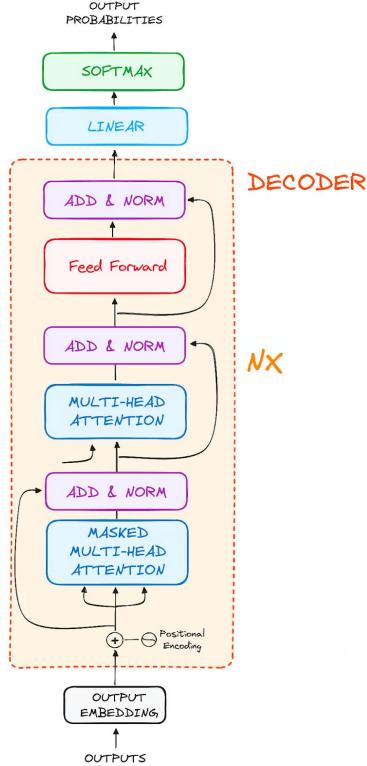
Transformer Architecture: Inference



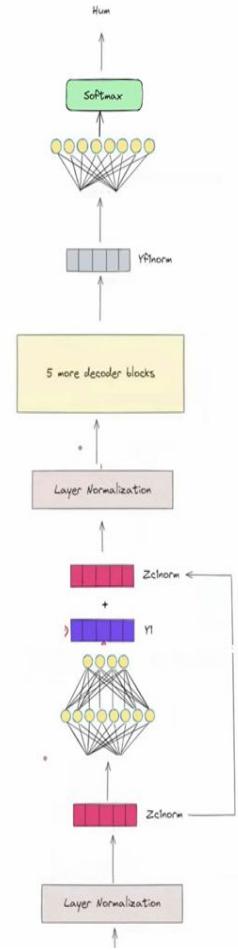
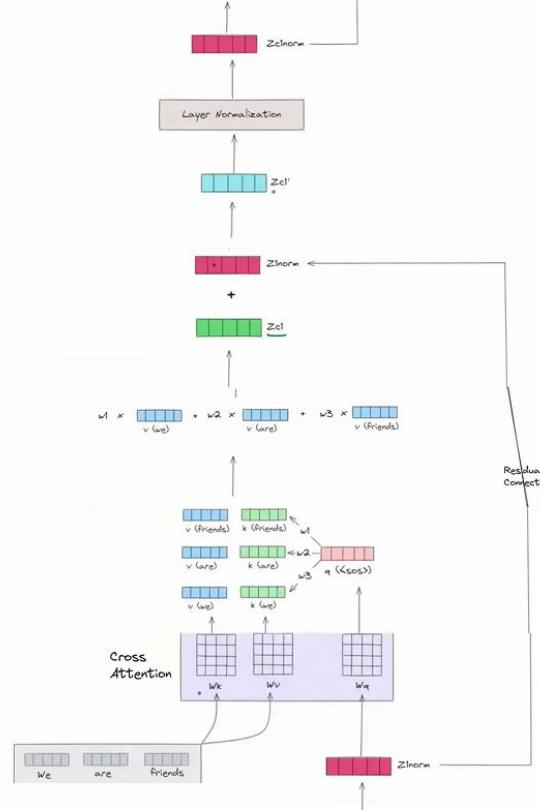
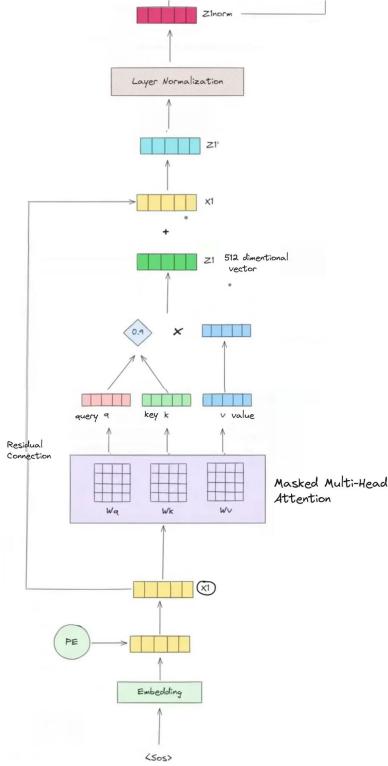
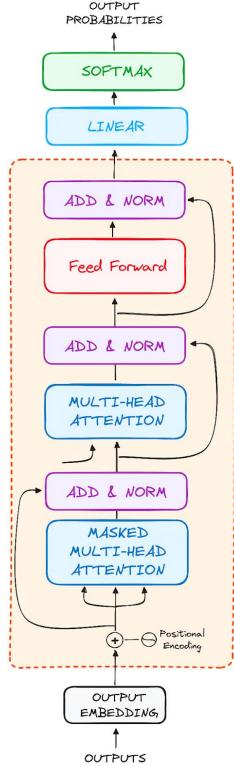
Transformer Architecture: Inference



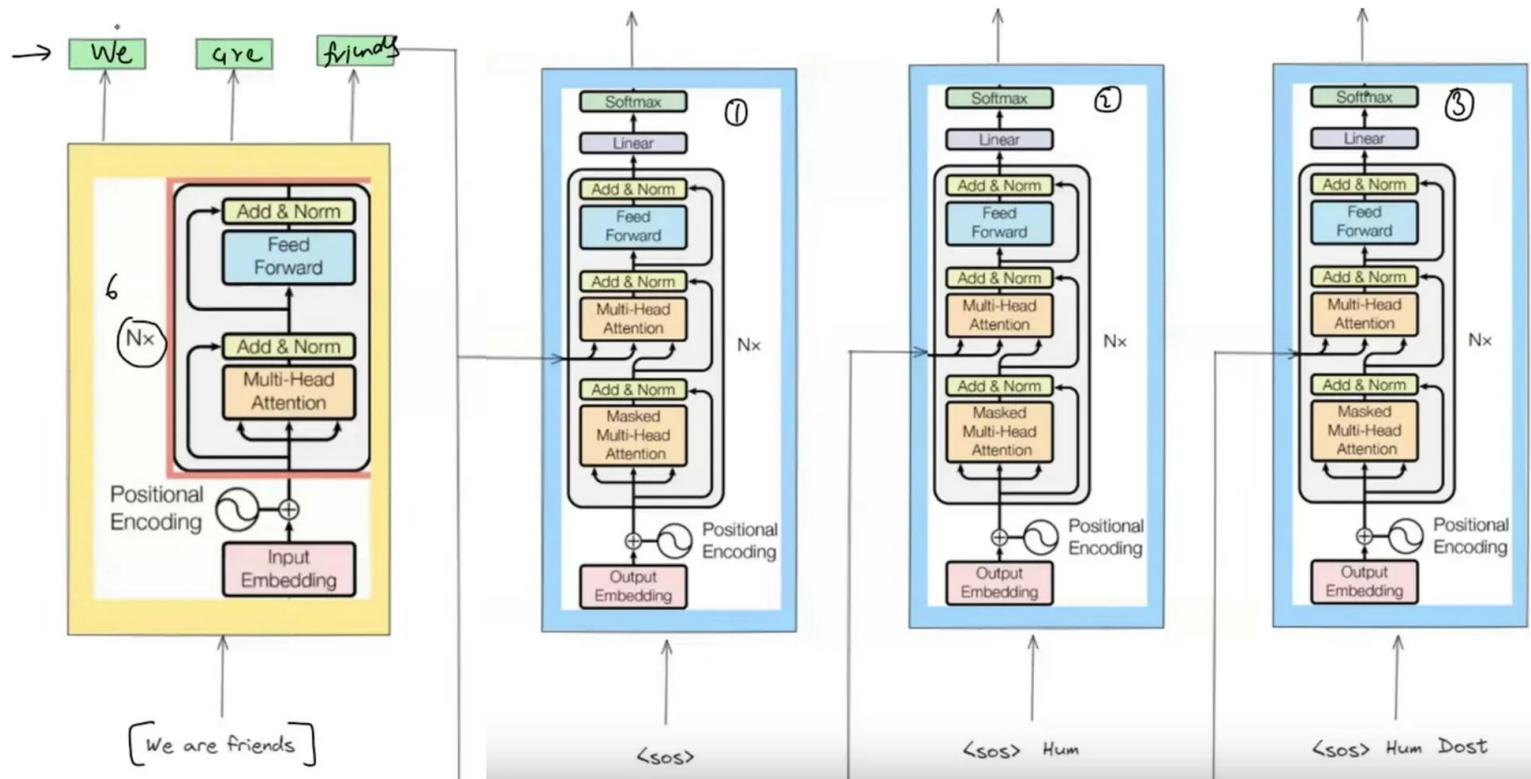
Transformer Architecture: Inference



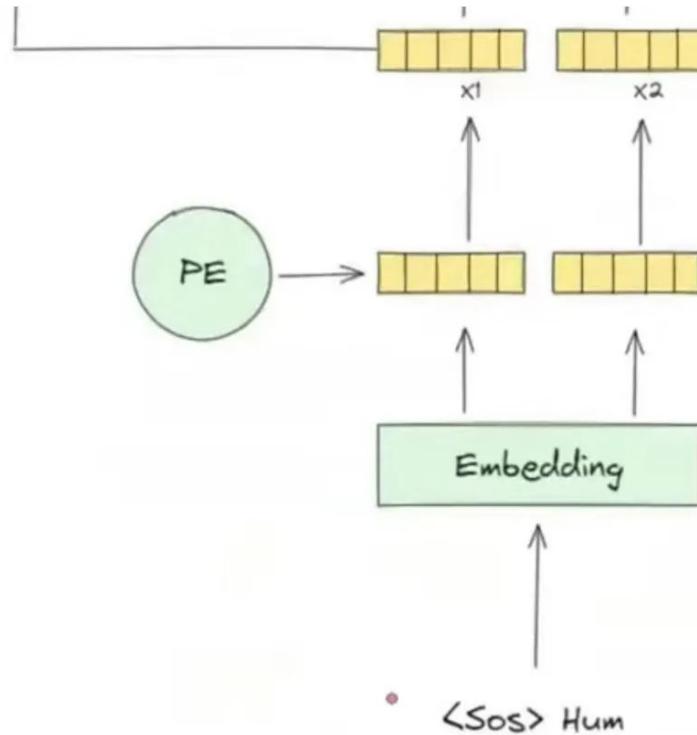
Transformer Architecture: Inference



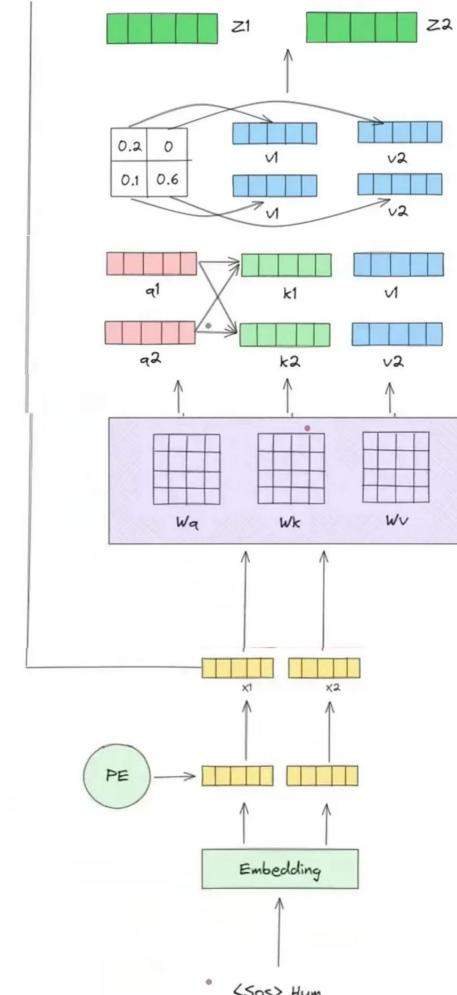
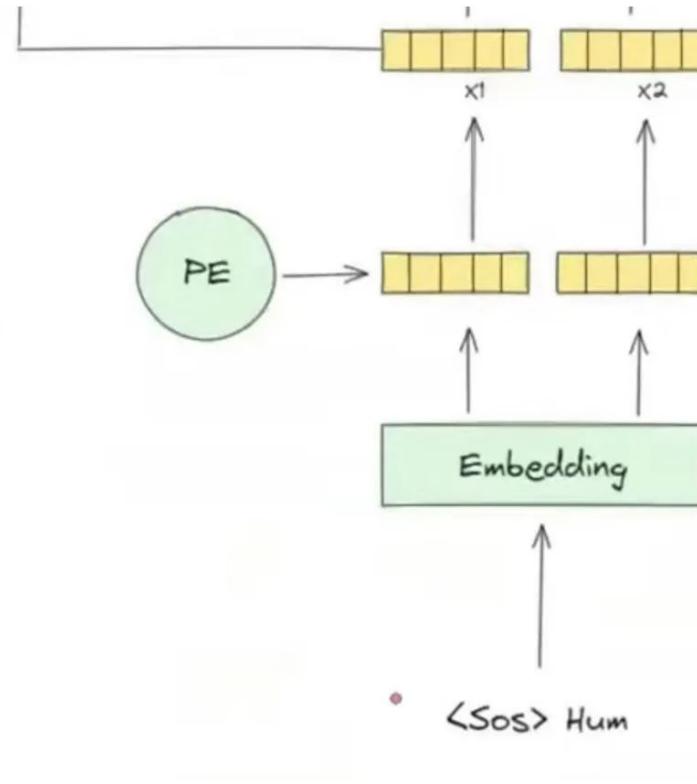
Transformer Architecture: Inference



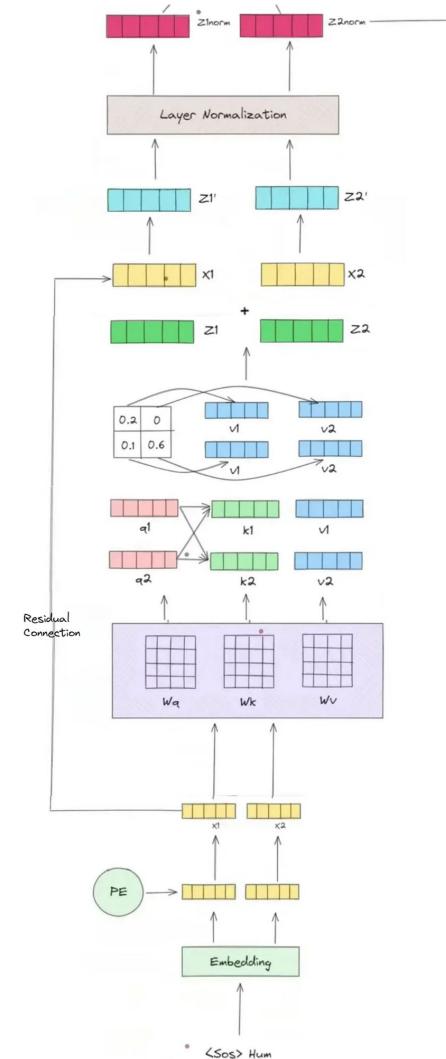
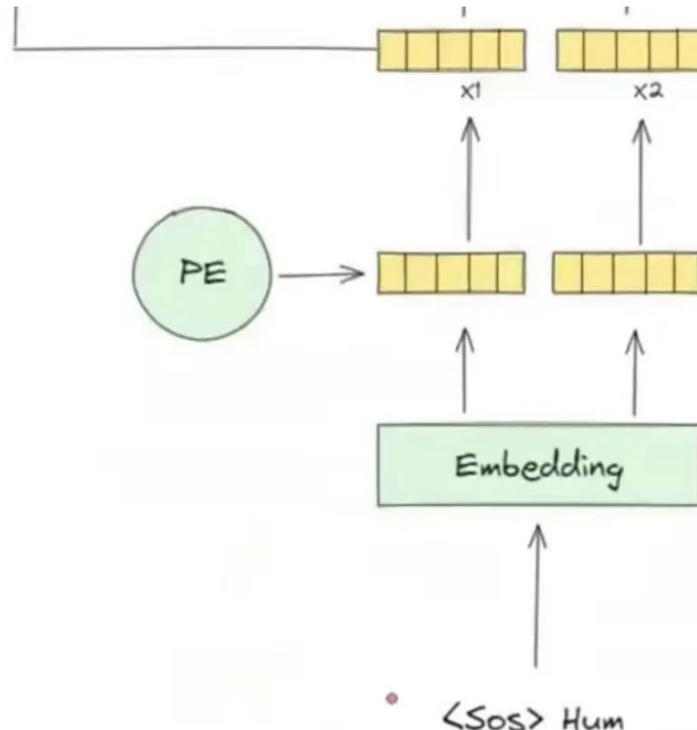
Transformer Architecture: Inference



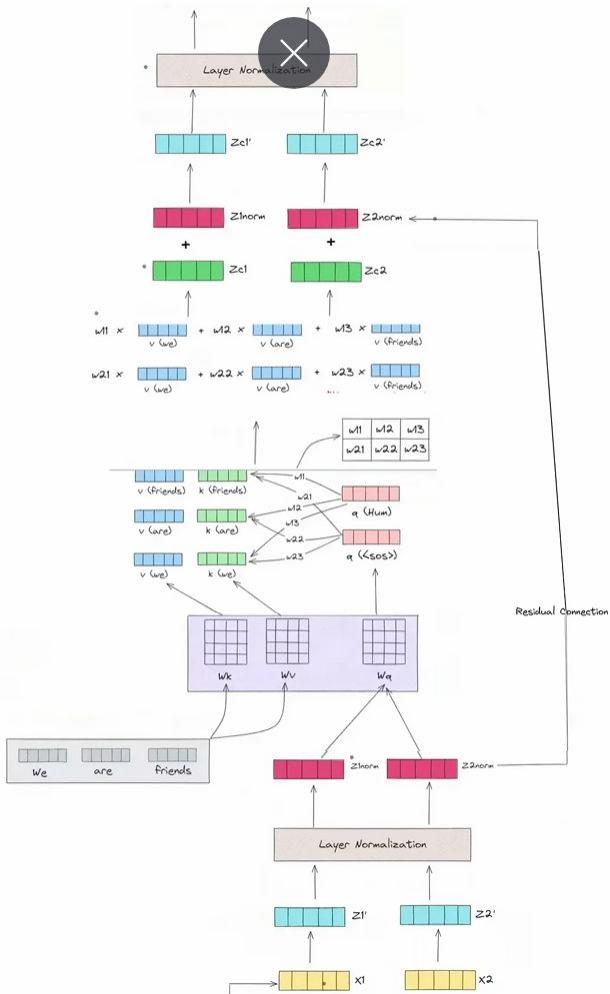
Transformer Architecture: Inference



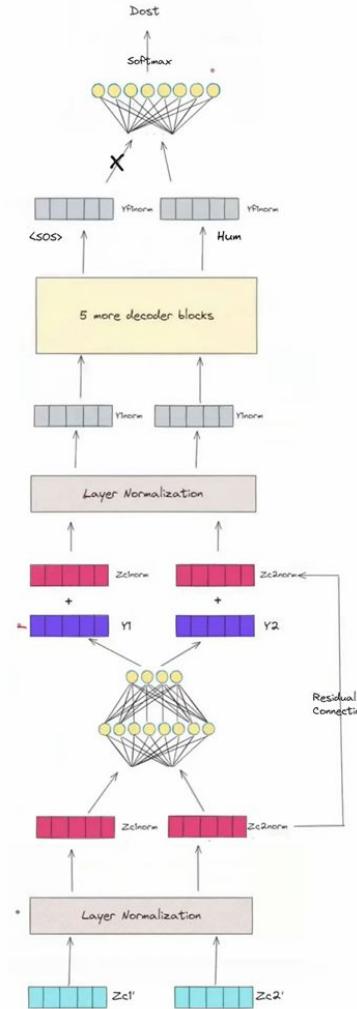
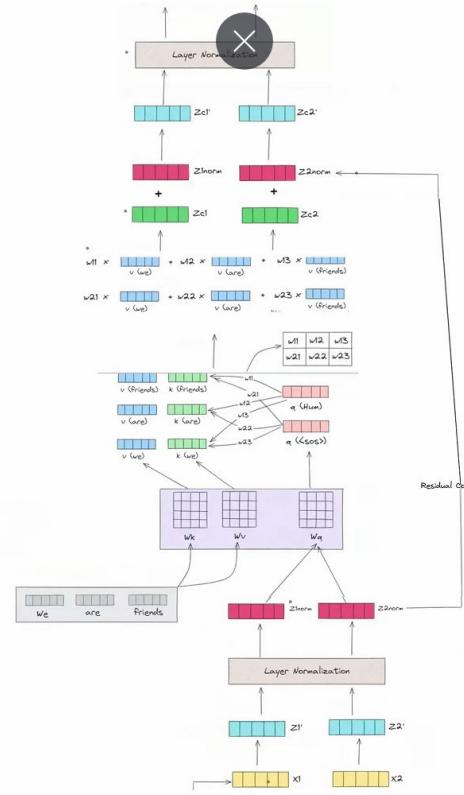
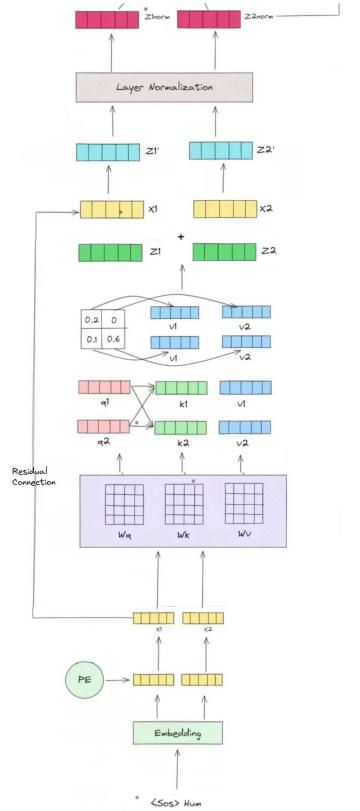
Transformer Architecture: Inference



Transformer Architecture: Inference



Transformer Architecture: Inference



Encoder Only Transformer: BERT

BERT: Bidirectional Encoder Representations from Transformers

- Unlike common language models, BERT does not handle “special tasks” with prompts, but rather, it can be specialized on a particular task by means of fine-tuning.
- Unlike common language models, BERT has been trained using the left context and the right context.
- Unlike common language models, BERT is not built specifically for text generation.
- Unlike common language models, BERT has not been trained on the Next Token Prediction task, but rather, on the Masked Language Model and Next Sentence Prediction task.

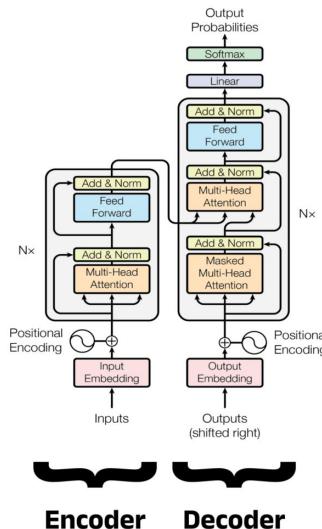
BERT: Bidirectional Encoder Representations from Transformers

- BERT model can be defined by four main features:
 - Encoder-only architecture
 - Use of bidirectional context
 - Pre-training approach
 - Model fine-tuning

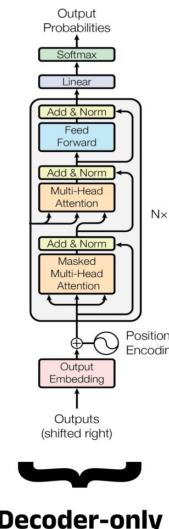
BERT: Bidirectional Encoder Representations from Transformers

- BERT is Encoder-only model of Vanilla Transformer.

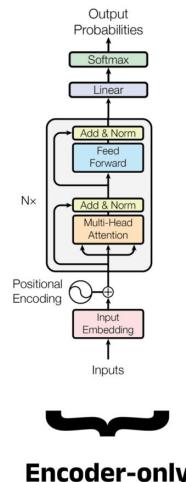
Transformer



GPT*



BERT*

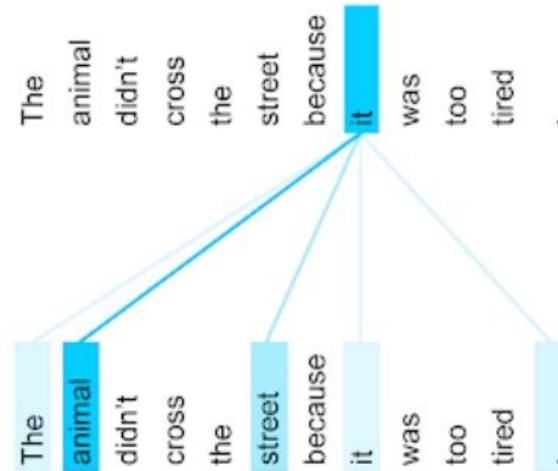


*Illustrative example, exact model architecture may vary slightly

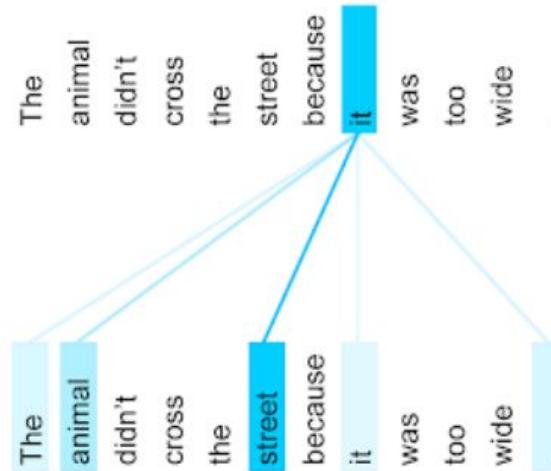
Transformer Encoder Self Attention: Inherently Bidirectional

Imagine the following sentences:

Sentence A: The animal didn't cross the street because **it** was too tired.



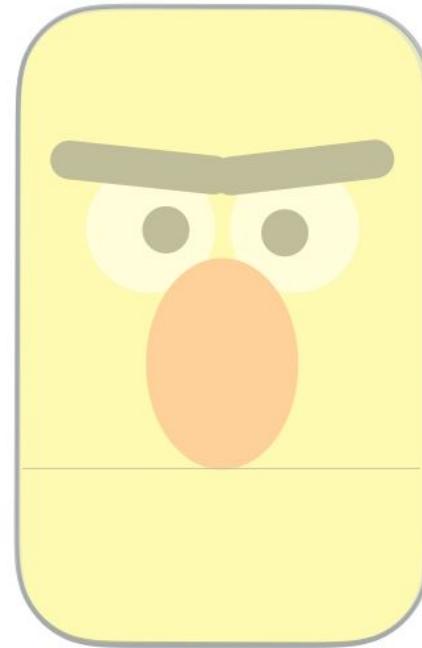
Sentence B: The animal didn't cross the street because **it** was too wide.



BERT model Architecture

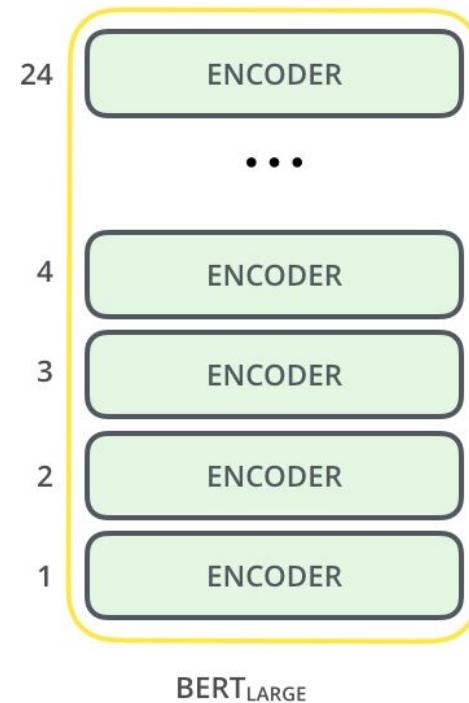
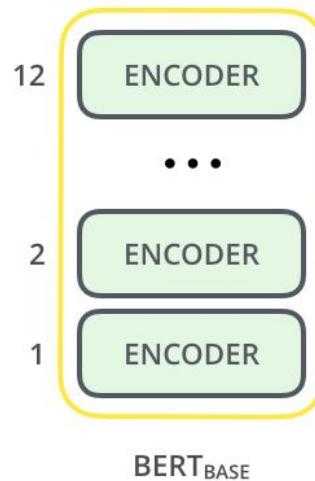


BERT_{BASE}

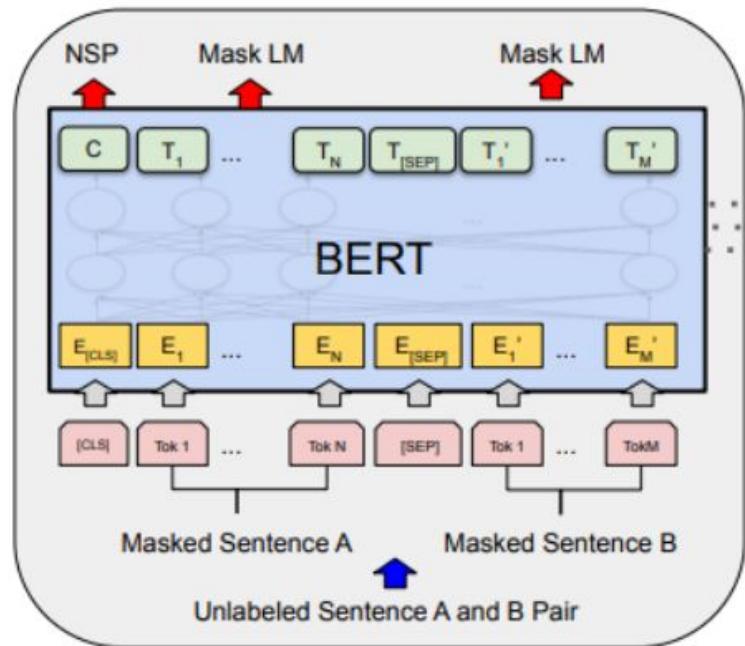


BERT_{LARGE}

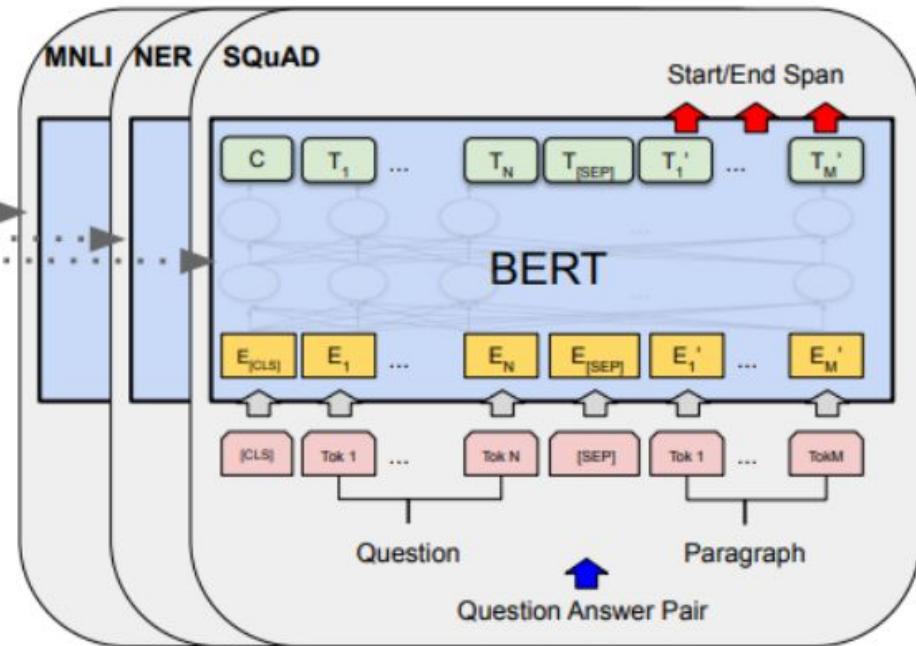
BERT model Architecture



BERT pre-training and fine tuning



Pre-training



Fine-Tuning

BERT pre-training

BERT was pre-trained on two tasks:

- Masked Language Model (MLM) Tasks
- Next Sentence Prediction (NLP) Tasks

Masked Language Model

- In Masked Language Modeling (MLM), one word in each sentence is masked by replacing it with a **[MASK]** token, and the task is predicting it.

Rome is the **capital** of Italy, which is why it hosts many government buildings.

Randomly select one or more tokens and replace them with the special token **[MASK]**

Rome is the **[MASK]** of Italy, which is why it hosts many government buildings.



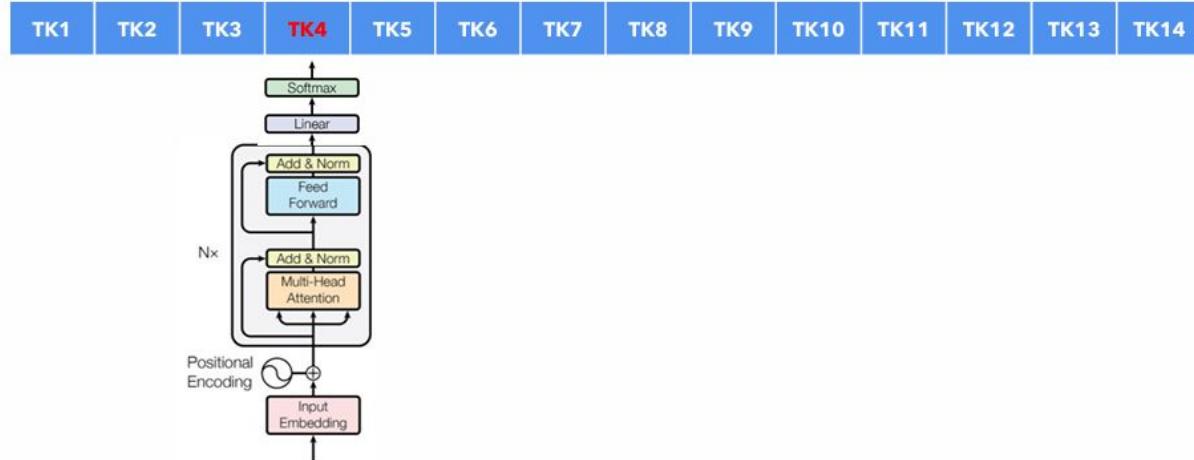
capital

Masked Language Model

Target (1 token):



Output (14 tokens):



Input (14 tokens):

Rome is the [mask] of Italy, which is why it hosts many government buildings.

Masked Language Model (MLM) : details

- Let us consider the following sample dataset:

"the sun rises in the east"

"it provides warmth to the earth"

"birds chirp in the morning"

"the moon appears at night"

"stars twinkle in the dark sky"

Step 1: Initialize vocabulary

Example vocabulary (simplified):

- Whole words: the, sun, rises, in, east, it, provides, warmth, to, earth, birds, chirp, morning, moon, appears, night, stars, twinkle, dark, sky
- Subwords: ##es, ##ing, ##ar, ##th, ##le, ##ight, ##ov, ##id, ##rk

(## denotes a subword that is not standalone)

Masked Language Model (MLM) : details

- Let us consider the following sample dataset:

"the sun rises in the east"

"it provides warmth to the earth"

"birds chirp in the morning"

"the moon appears at night"

"stars twinkle in the dark sky"

Step 2: Apply WordPiece Tokenization

Each word is checked in the vocabulary. If a word is not found, it is broken into subwords.

"the sun rises in the east"

→ ["the", "sun", "rises", "in", "the", "east"]

(All words are in the vocabulary)

"it provides warmth to the earth"

→ ["it", "provid", "#es", "warmth", "to", "the", "earth"]

("provides" is split into **provid** + **#es**)

Masked Language Model (MLM) : details

- Let us consider the following sample dataset:

"the sun rises in the east"

"it provides warmth to the earth"

"birds chirp in the morning"

"the moon appears at night"

"stars twinkle in the dark sky"

Step 3: Final Tokenized Representation

["the", "sun", "rises", "in", "the", "east"]

["it", "provid", "#es", "warmth", "to", "the", "earth"]

["birds", "chirp", "in", "the", "morn", "#ing"]

["the", "moon", "appears", "at", "night"]

["stars", "twink", "#le", "in", "the", "dark", "sky"]

Masked Language Model (MLM) : details

Step 4: Preparing Masked data

BERT's Masked Language Model (MLM) randomly selects **15% of tokens** in a sentence for masking.

For example, consider the sentence:

Original: "The sun rises in the east"

Tokenized: [CLS, "the", "sun", "rises", "in", "the", "east", "SEP"]

- **Total tokens:** 8
- **15% of 8 ≈ 1 token to modify**

Let's assume "rises" is selected.

Step 3: Final Tokenized Representation

1. ["[CLS]", "the", "sun", "rises", "in", "the", "east", "[SEP]"]
2. ["[CLS]", "it", "provid", "#es", "warmth", "to", "the", "earth", "[SEP]"]
3. ["[CLS]", "birds", "chirp", "in", "the", "morn", "#ing", "[SEP]"]
4. ["[CLS]", "the", "moon", "appears", "at", "night", "[SEP]"]
5. ["[CLS]", "stars", "twink", "#le", "in", "the", "dark", "sky", "[SEP]"]

Masked Language Model (MLM) : details

Step 4: Preparing Masked data

BERT's Masked Language Model (MLM) randomly selects **15% of tokens** in a sentence for masking.

For example, consider the sentence:

Original: "The sun rises in the east"

Tokenized: [CLS, "the", "sun", "rises", "in", "the", "east", "SEP"]

- **Total tokens:** 8
- **15% of 8 \approx 1 token to modify**

Let's assume "rises" is selected.

Applying the 80-10-10 Rule

For every **selected token**, we generate a random number between 0 and 1.

- **If the number is ≤ 0.8 ,** we replace it with [MASK]. (80% probability)
- **If the number is between 0.8 and 0.9,** we replace it with a random word. (10% probability)
- **If the number is between 0.9 and 1.0,** we keep it unchanged. (10% probability)

Masked Language Model (MLM) : details

Step 4: Preparing Masked data

BERT's Masked Language Model (MLM) randomly selects **15% of tokens** in a sentence for masking.

For example, consider the sentence:

Original: "The sun rises in the east"

Tokenized: [CLS, "the", "sun", "rises", "in", "the", "east", "SEP"]

- **Total tokens:** 8
- **15% of 8 \approx 1 token to modify**

Let's assume "rises" is selected.

Applying the 80-10-10 Rule

Let's say a random number generator gives 0.72.

→ replace "rises" with "[MASK]" token

Original Tokenized: [CLS, "the", "sun", "rises", "in", "the", "east", "SEP"]

Maked Tokenized: [CLS, "the", "sun", "[MASK]", "in", "the", "east", "SEP"]

Masked Language Model (MLM) : details

Step 4: Preparing Masked data

BERT's Masked Language Model (MLM) randomly selects **15% of tokens** in a sentence for masking.

For example, consider the sentence:

Original: "The sun rises in the east"

Tokenized: [CLS, "the", "sun", "rises", "in", "the", "east", "SEP"]

- **Total tokens:** 8
- **15% of 8 \approx 1 token to modify**

Let's assume "rises" is selected.

Applying the 80-10-10 Rule

Let's say a random number generator gives 0.88.

→ replace "rises" with token "sets"

Original Tokenized: [CLS, "the", "sun", "rises", "in", "the", "east", "SEP"]

Maked Tokenized: [CLS, "the", "sun", "sets", "in", "the", "east", "SEP"]

Masked Language Model (MLM) : details

Step 4: Preparing Masked data

BERT's Masked Language Model (MLM) randomly selects **15% of tokens** in a sentence for masking.

For example, consider the sentence:

Original: "The sun rises in the east"

Tokenized: [CLS, "the", "sun", "rises", "in", "the", "east", "SEP"]

- **Total tokens:** 8
- **15% of 8 \approx 1 token to modify**

Let's assume "rises" is selected.

Applying the 80-10-10 Rule

Let's say a random number generator gives 0.93.

→ replace "rises" with token "rises"

Original Tokenized: [CLS, "the", "sun", "rises", "in", "the", "east", "SEP"]

Maked Tokenized: [CLS, "the", "sun", "rises", "in", "the", "east", "SEP"]

Masked Language Model (MLM) : details

Step 4: Preparing Masked data

BERT's Masked Language Model (MLM) randomly selects **15% of tokens** in a sentence for masking.

For example, consider the sentence:

Original: "The sun rises in the east"

Tokenized: [CLS, "the", "sun", "rises", "in", "the", "east", "SEP"]

- **Total tokens:** 8
- **15% of 8 \approx 1 token to modify**

Let's assume "rises" is selected.

Applying the 80-10-10 Rule

1. Let's say a random number generator gives 0.72.
Maked Tokenized: [CLS, "the", "sun", "[MASK]", "in", "the", "east", "SEP"]
2. Let's say a random number generator gives 0.88.
Maked Tokenized: [CLS, "the", "sun", "sets", "in", "the", "east", "SEP"]
3. Let's say a random number generator gives 0.93.
Maked Tokenized: [CLS, "the", "sun", "rises", "in", "the", "east", "SEP"]

Masked Language Model (MLM) : training

Step 4: Preparing Masked data

Original Sentence	Masked Sentence	Label
<i>the sun rises in the east</i>	[CLS] <i>the sun [MASK] in the east [SEP]</i>	[102, -100, -100, rises , -100, -100, -100, 104]
<i>it provides warmth to the earth</i>	[CLS] <i>it provides warmth to the earth [SEP]</i>	[102, -100, -100, -100, -100, -100, earth , 104]
<i>birds chirp in the morning</i>	[CLS] dogs chirp in the morning [SEP]	[102, birds , -100, -100, -100, -100, 104]
<i>the moon appears at night</i>	[CLS] <i>the [MASK] appears at night [SEP]</i>	[102, -100, moon , -100, -100, -100, 104]
<i>stars twinkle in the dark sky</i>	[CLS] <i>stars twinkle in the [MASK] sky [SEP]</i>	[102, -100, -100, -100, -100, dark , -100, -100, 104]

With Masked Language and Label pair we train the BERT model

Masked Language Model: backpropagation

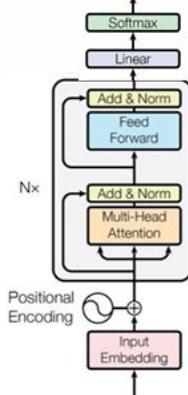
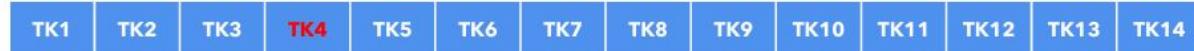
Target (1 token):

capital



Run **backpropagation** to update the weights

Output (14 tokens):



Input (14 tokens):

Rome is the [mask] of Italy, which is why it hosts many government buildings.

Next Sentence Prediction (NSP)

- Many downstream applications require learning relationships between sentences rather than single tokens

Sentence A → Before my bed lies a pool of moon bright
I could imagine that it's frost on the ground
Sentence B → I look up and see the bright shining moon
Bowing my head I am thinking of home



- 50% of the time, we select the actual next sentence.
- 50% of the time we select a random sentence from the text.

Sentence A = Before my bed lies a pool of moon bright
Sentence B = I look up and see the bright shining moon



IsNext

NotNext

Next Sentence Prediction (NSP)

- Let us consider the following sample dataset:

"the sun rises in the east"

"it provides warmth to the earth"

"birds chirp in the morning"

"the moon appears at night"

"stars twinkle in the dark sky"

Next Sentence Prediction (NSP)

- consider the following sample dataset:
"the sun rises in the east"
"it provides warmth to the earth"
"birds chirp in the morning"
"the moon appears at night"
"stars twinkle in the dark sky"

Sentence A	Sentence B	Label
<i>the sun rises in the east</i>	<i>it provides warmth to the earth</i>	<i>Is Next (1)</i>
<i>it provides warmth to the earth</i>	<i>stars twinkle in the dark sky</i>	<i>Not Next (0)</i>
<i>birds chirp in the morning</i>	<i>birds chirp in the morning</i>	<i>Not Next (0)</i>
<i>the moon appears at night</i>	<i>stars twinkle in the dark sky</i>	<i>Is Next (1)</i>
<i>stars twinkle in the dark sky</i>	<i>the sun rises in the east</i>	<i>Not Next (0)</i>

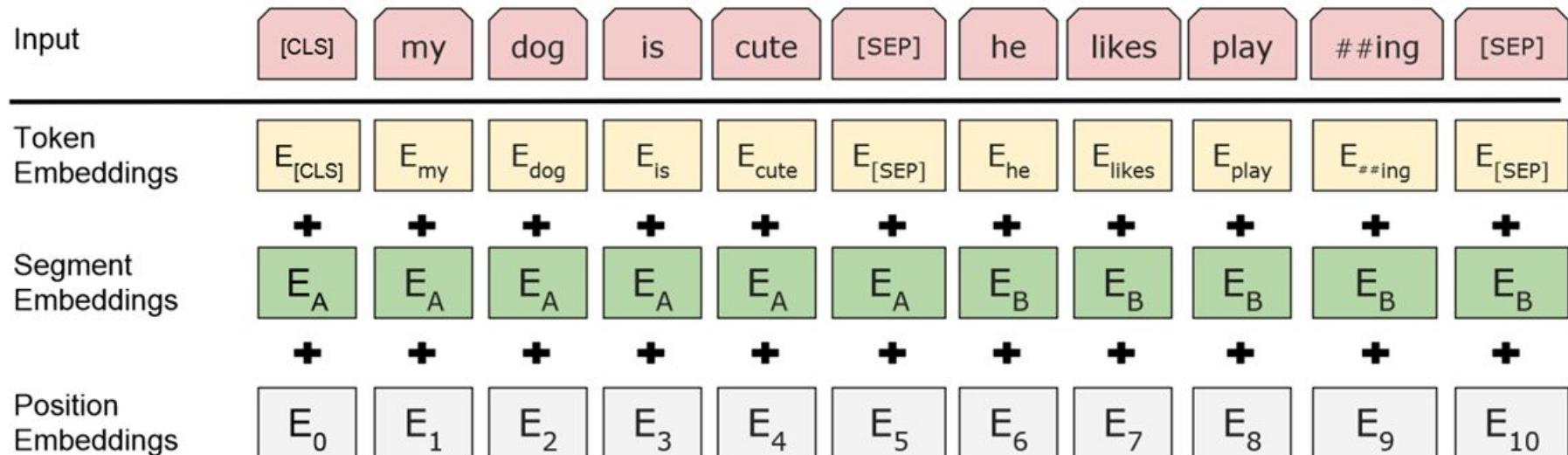
Next Sentence Prediction (NSP)

- 50% of the time, we select the actual next sentence.
- 50% of the time we select a random sentence from the text.

Final Input	Label
[CLS] the sun rises in the east [SEP] it provides warmth to the earth [SEP]	Is Next (1)
[CLS] it provides warmth to the earth [SEP] stars twinkle in the dark sky [SEP]	Not Next (0)
[CLS] birds chirp in the morning [SEP] birds chirp in the morning [SEP]	Not Next (0)
[CLS] the moon appears at night [SEP] stars twinkle in the dark sky [SEP]	Is Next (1)
[CLS] stars twinkle in the dark sky [SEP] the sun rises in the east [SEP]	Not Next (0)
[CLS] my dog is cute [SEP] he likes play ##ing [SEP]	Is Next (1)

Next Sentence Prediction (NSP): Segmentation embeddings

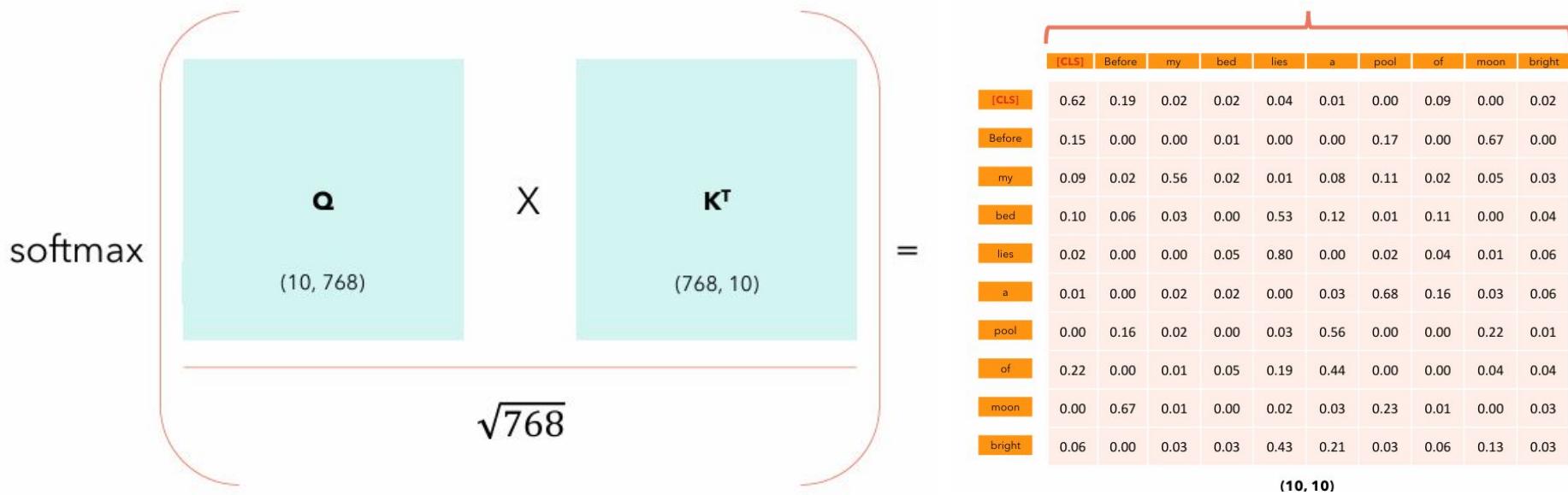
- Given the sentence A and the sentence B, how can BERT understand which tokens belongs to the sentence A and which to the sentence B?
 - the segmentation embeddings! Is integrated



Next Sentence Prediction (NSP): [CLS] token

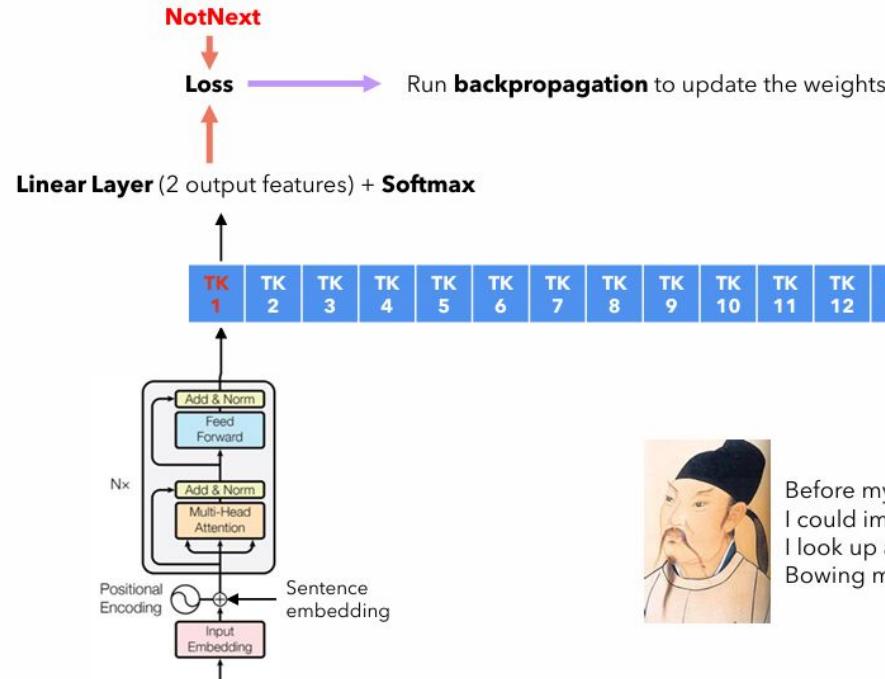
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- The [CLS] token always interacts with all the other tokens,
- So, we can consider the [CLS] token as a token that “captures” the information from all the other tokens.



Next Sentence Prediction (NSP): training

Target (1 token):



Output (20 tokens):



Before my bed lies a pool of moon bright
I could imagine that it's frost on the ground
I look up and see the bright shining moon
Bowing my head I am thinking of home

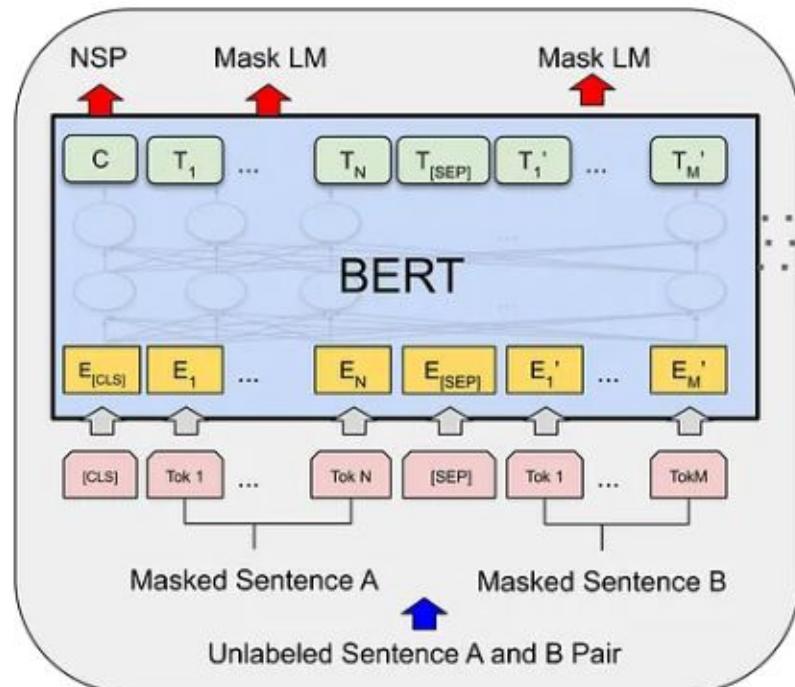
Input (20 tokens):

[CLS] Before my bed lies a pool of moon bright [SEP] I look up and see the bright shining moon

BERT: Generated NSP + MLM Training Data

Sentence A	Sentence B	NSP Label	MLM Input (Masked Sentences)	MLM Labels
The sun rises in the east.	It provides warmth to the Earth.	1 (Positive)	"The [MASK] rises in the east. It [MASK] warmth to the Earth."	["sun", -100, -100, -100, -100, "provides", -100, -100, -100, -100]
AI is transforming industries.	It is used in healthcare and finance.	1 (Positive)	"AI is [MASK] industries. It is used in [MASK] and finance."	[-100, -100, "transforming", -100, -100, -100, -100, "healthcare", -100, -100]
Dogs are known for their loyalty.	Many modern applications use machine learning.	0 (Negative)	"Dogs [MASK] known for their loyalty. Many [MASK] applications use machine learning."	[-100, "are", -100, -100, -100, -100, "modern", -100, -100, -100, -100]

BERT: Pretraining on NSP + MLM Task



Step 1: Create Training Pairs for NSP (Next Sentence Prediction)

Sentence A: "The sun rises in the east."

Sentence B: "It provides warmth to the Earth."

NSP Label: Is Next (1)

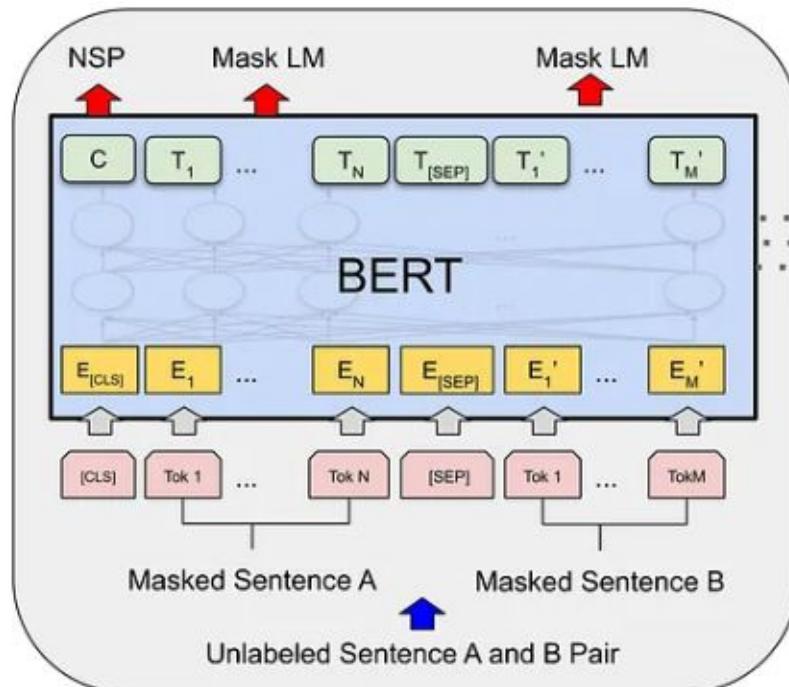
Sentence A: "The moon appears at night."

Sentence B: ""Birds chirp in the morning."

NSP Label: Not Next (0)

Pre-training

BERT: Pretraining on NSP + MLM Task



Step 2: Apply Masked Language Modeling (MLM)

After pairing sentences for NSP, we apply MLM to Sentence A and/or Sentence B by masking 15% of the tokens.

Original Input:

Sentence A → "The sun rises in the east."

Sentence B → "It provides warmth to the Earth."

Masked Input:

Sentence A → "The sun [MASK] in the east."

Sentence B → "It [MASK] warmth to the Earth."

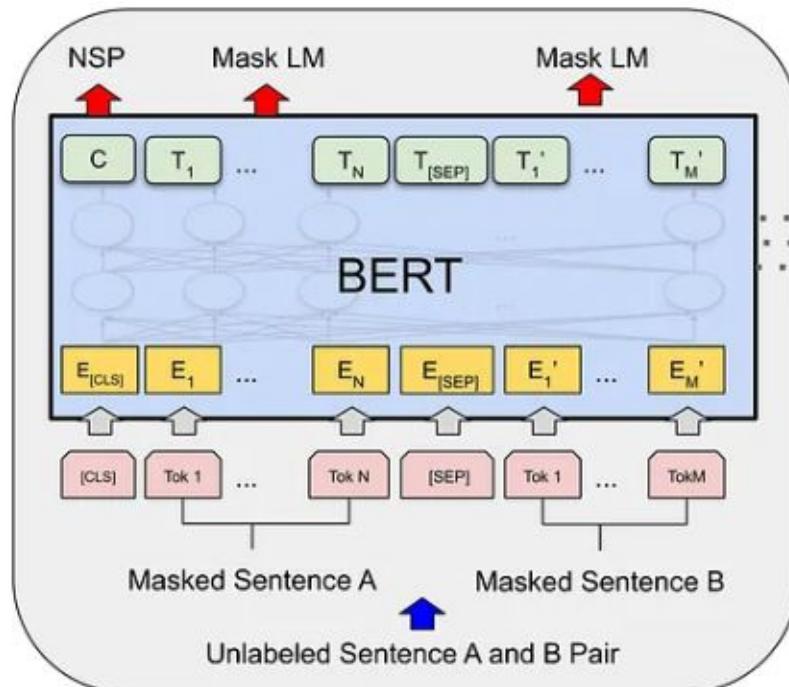
MLM Label Output:

["rises", -100, -100, -100, -100]

["provides", -100, -100, -100, -100]

(-100 means no prediction loss for those words.)

BERT: Pretraining on NSP + MLM Task



Pre-training

Step 3: Tokenization & Input Formatting

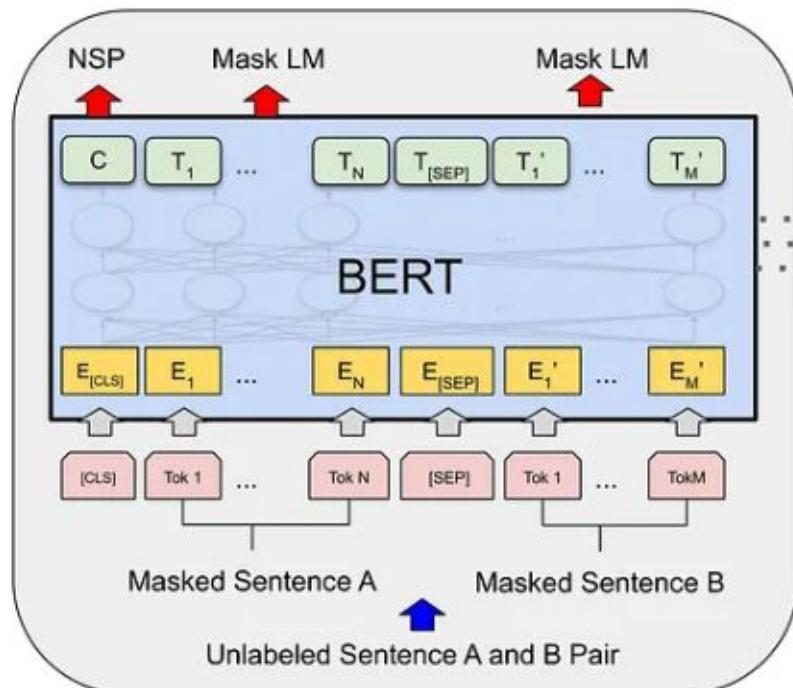
The input is tokenized with [CLS] and [SEP] tokens.

[CLS] The sun [MASK] in the east . [SEP] It [MASK] warmth to the Earth . [SEP]

Step 4: BERT Processes the Input

Embeddings
Self Attention
Feed Forward layers

BERT: Pretraining on NSP + MLM Task



Step 5: Simultaneous MLM + NSP Loss Computation

- **MLM Loss** → Cross-Entropy loss over **only masked words**.
- **NSP Loss** → Binary classification loss on the **[CLS]** token.

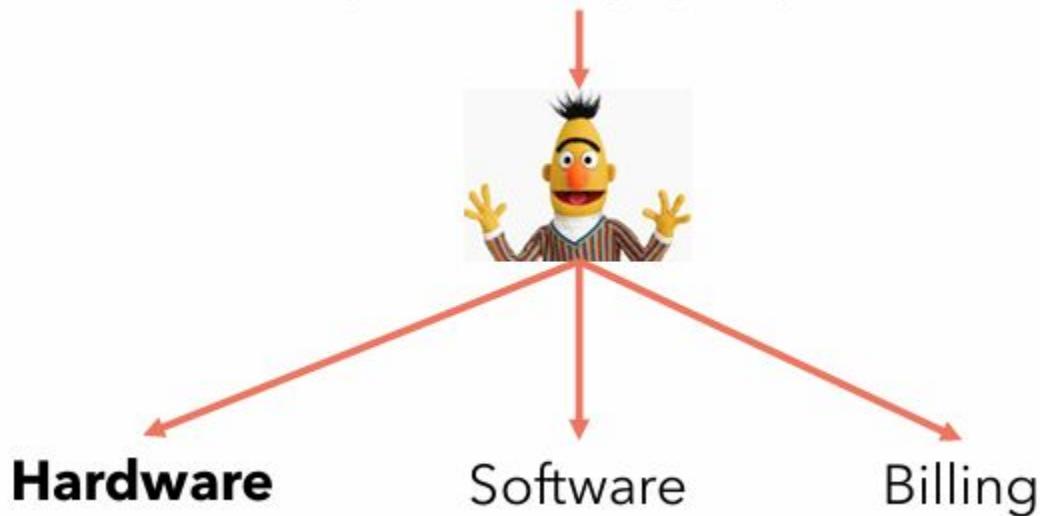
$$\text{Total Loss} = \text{MLM Loss} + \text{NSP Loss}$$

Backpropagation updates BERT's weights for both tasks simultaneously.

Pre-training

BERT Fine Tuning: Text Classification task

My router's led is not working, I tried changing the power socket but still nothing.

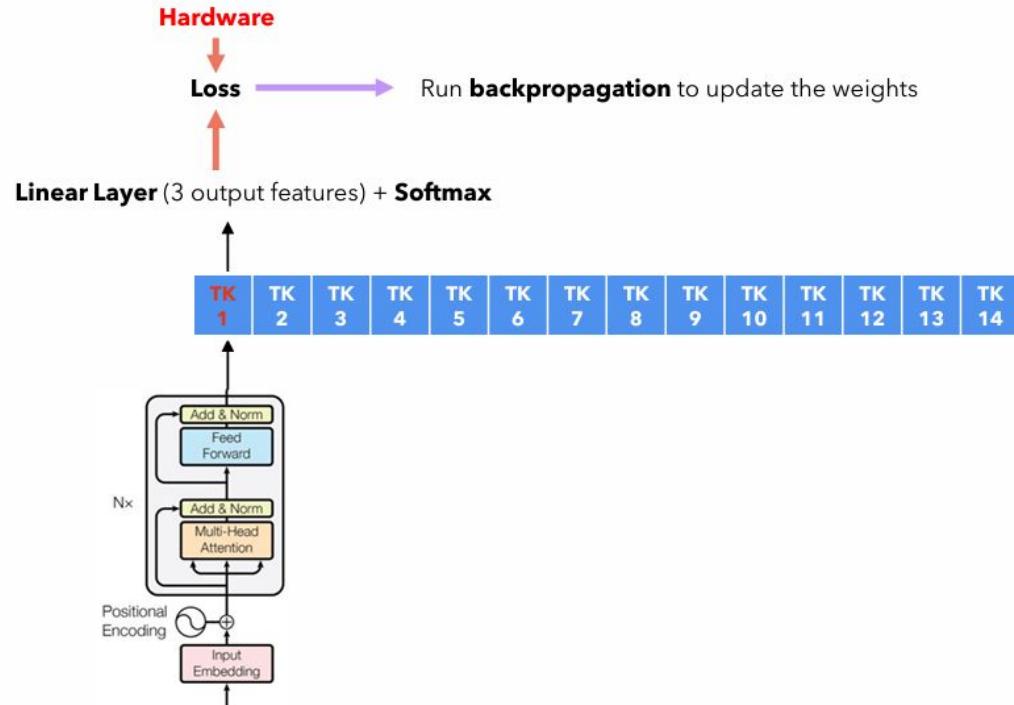


BERT Fine Tuning: Text Classification task

Target (1 token):

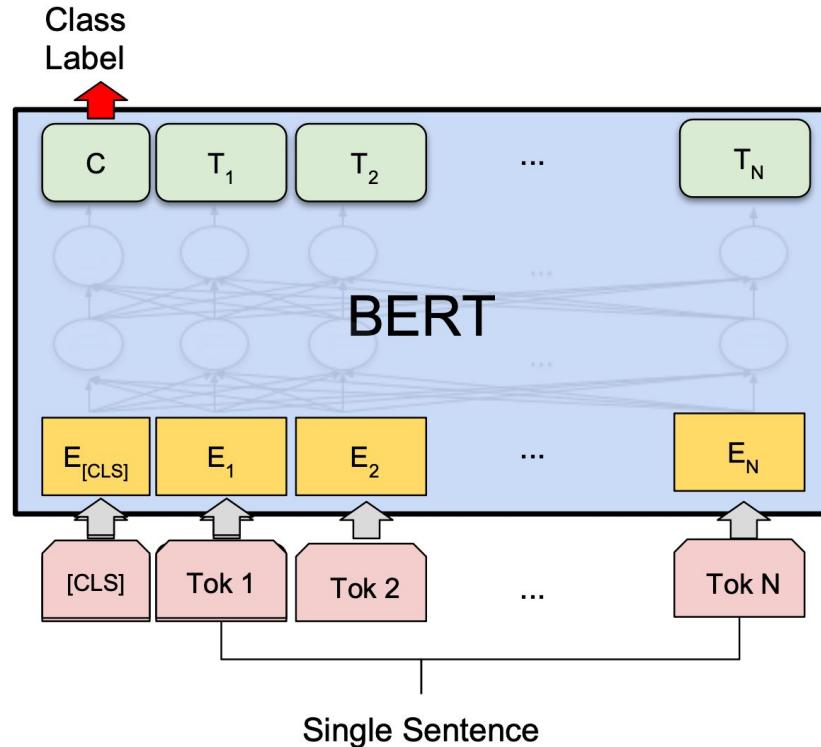
Output (16 tokens):

Input (16 tokens):



[CLS] My router's led is not working, I tried changing the power socket but still nothing.

BERT Fine Tuning: Text Classification task



BERT Fine Tuning: Question Answering

Question answering is the task of answering questions given a context.

Context: "Shanghai is a City in China, it is also a financial center, its fashion capital and industrial city."

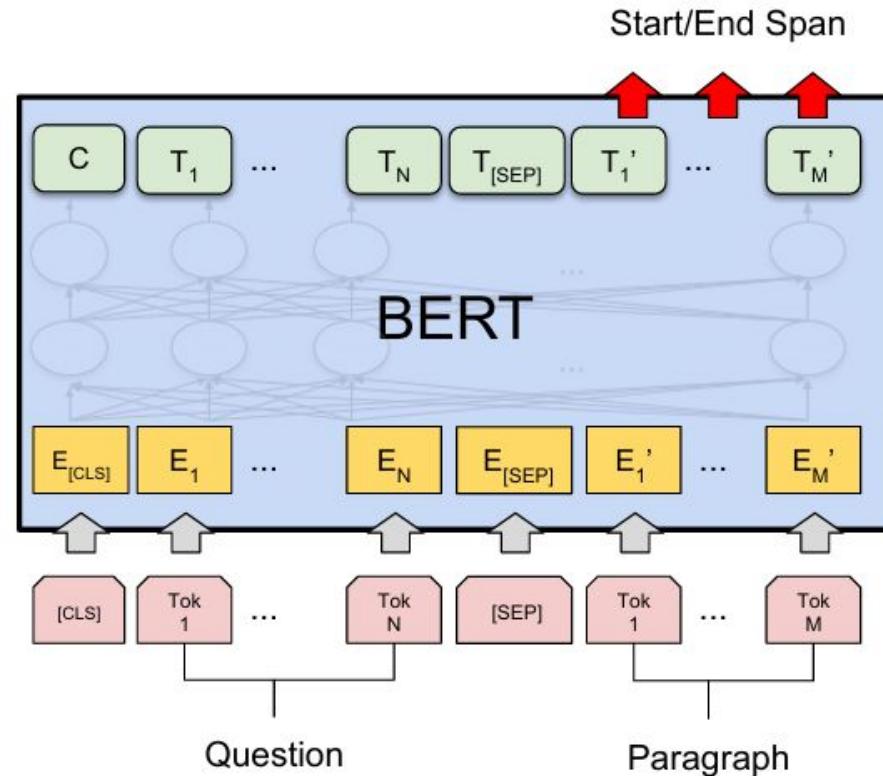
Question: "What is the fashion capital of China?"

Answer: "Shanghai is a City in China, it is also a financial center, its fashion capital and industrial city."

The model has to highlight the part of text that contains the answer. Two problems:

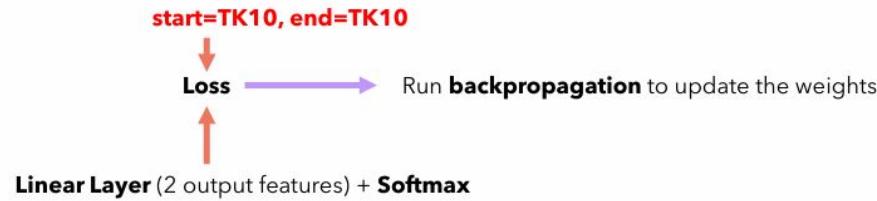
1. We need to find a way for BERT to understand which part of the input is the context, which one is the question.
2. We also need to find a way for BERT to tell us where the answer starts and where it ends in the context provided

BERT Fine Tuning: Question Answering

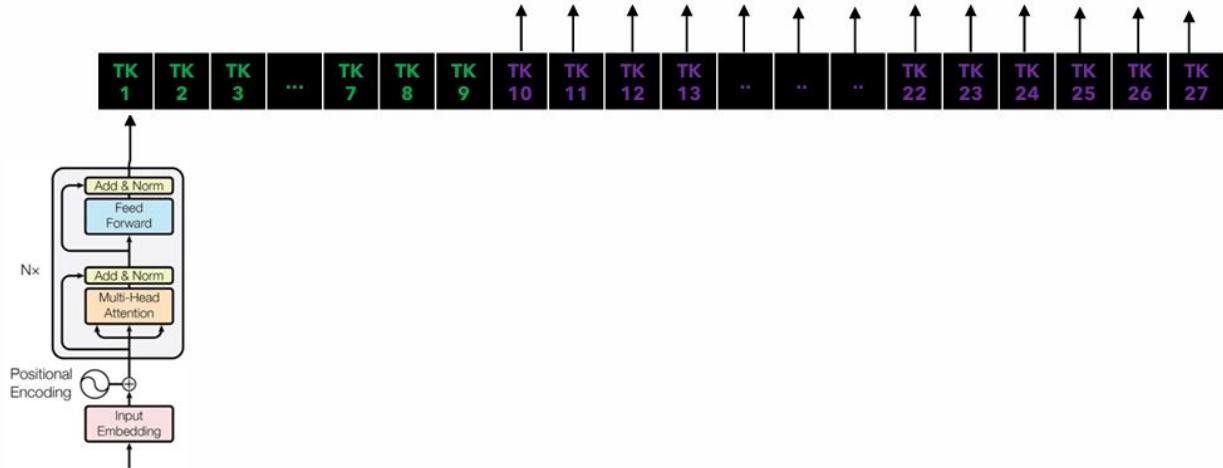


BERT Fine Tuning: Question Answering

Target (1 token):



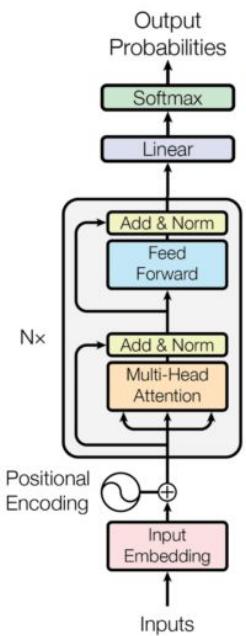
Output (27 tokens):



Input (27 tokens):

[CLS] What is the fashion capital of China? [SEP] Shanghai is a City in China, it is also a financial center, its fashion capital and industrial city.

BERT Variants



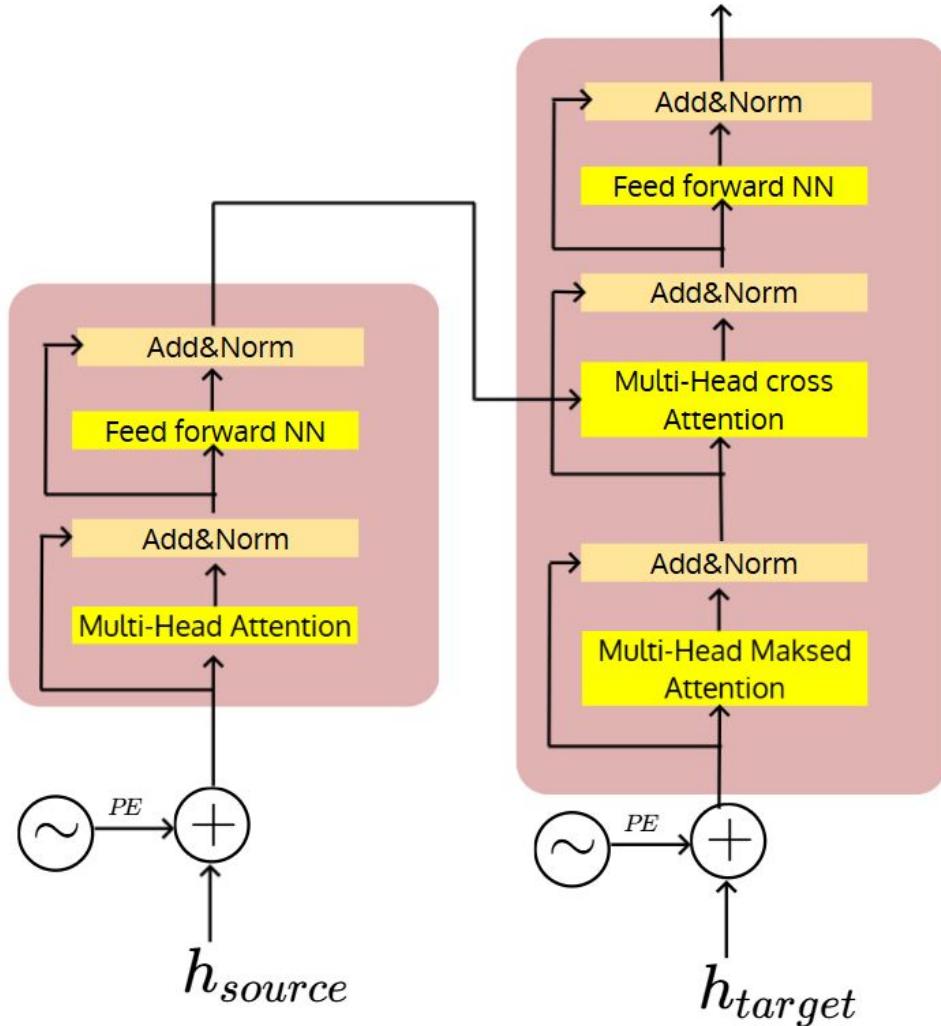
BERT Base BERT Large

Number of embedding dimensions, d_model	768	1024
Number of encoder blocks, N:	12	24
Number of attention heads per encoder block:	12	16
Size of hidden layer in feedforward network:	3072	4096
Total parameters:	110 million	340 million

Decoder Only Transformer: GPT

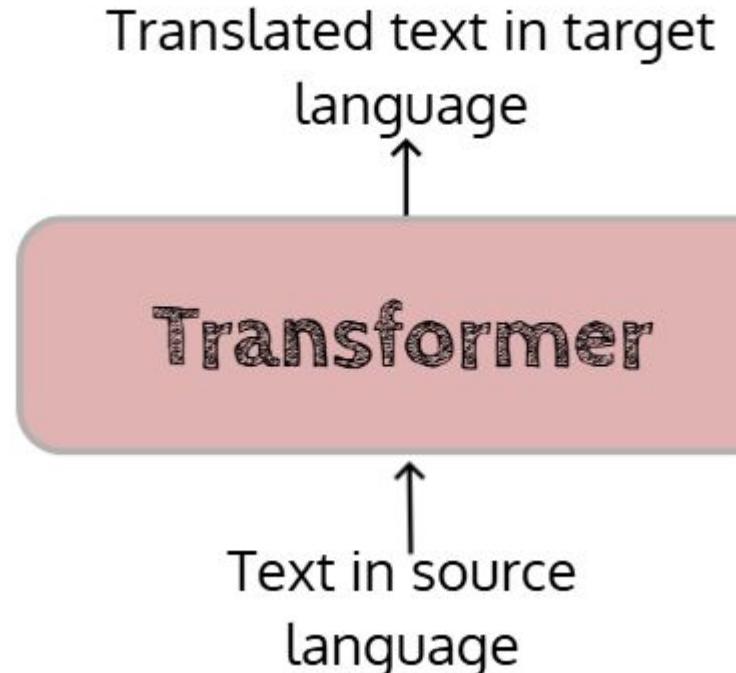
Transformer Recap

- In the previous lecture, we learned about the components of the transformer architecture in the context of machine translation.



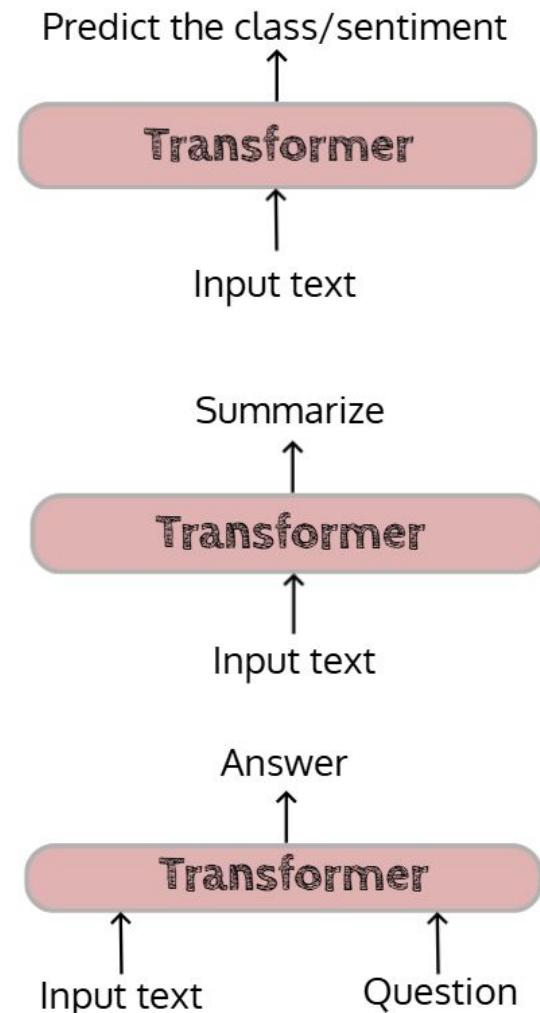
Transformer Recap

- In the previous lecture, we learned about the components of the transformer architecture in the context of machine translation.
- At a higher level of abstraction, we can think of it as a black box that receives an input and produces an output.



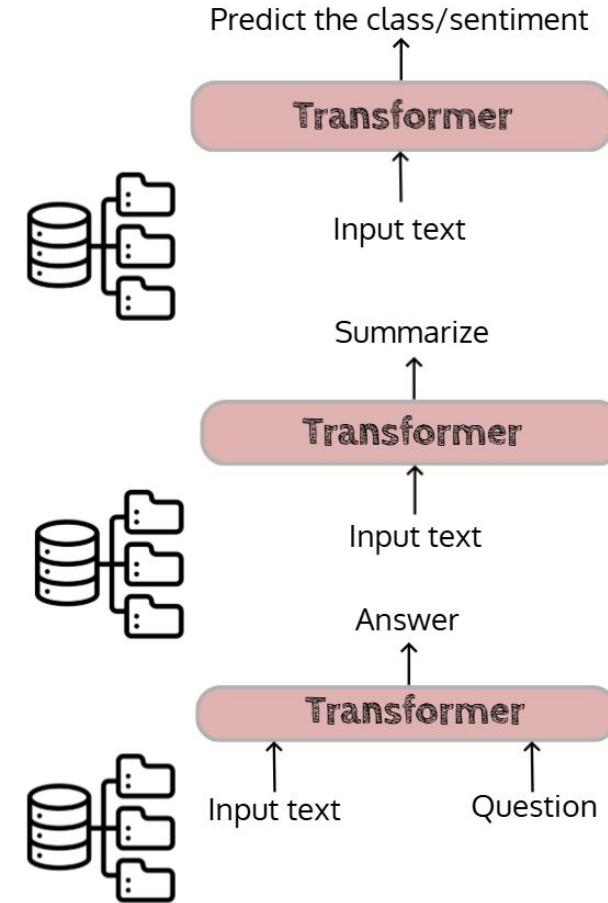
Transformer Recap

- In the previous lecture, we learned about the components of the transformer architecture in the context of machine translation.
- At a higher level of abstraction, we can think of it as a black box that receives an input and produces an output.
- What if we want to use the transformer architecture for other NLP tasks?



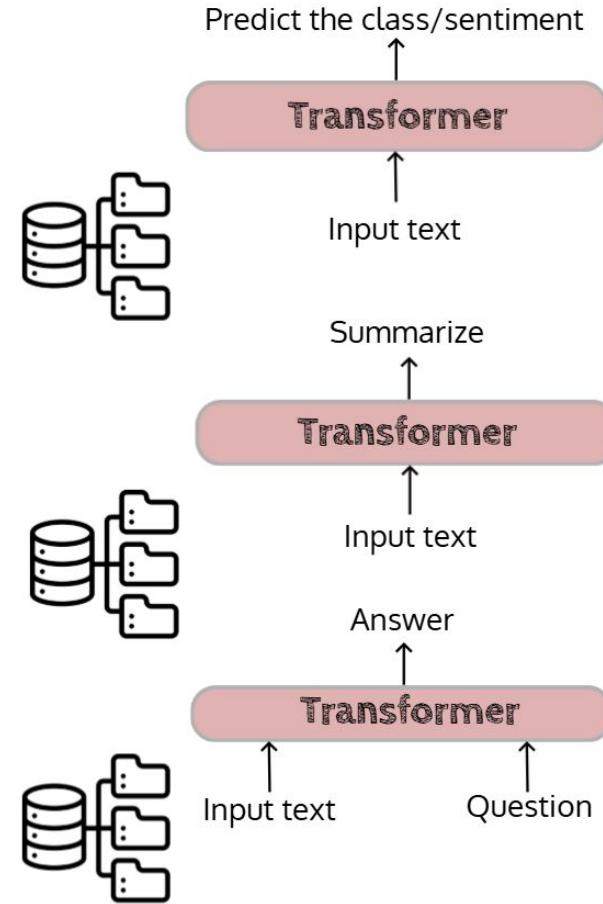
Transformer Recap

- In the previous lecture, we learned about the components of the transformer architecture in the context of machine translation.
- At a higher level of abstraction, we can think of it as a black box that receives an input and produces an output.
- What if we want to use the transformer architecture for other NLP tasks?
- We need to train a separate model for each task using dataset specific to the task



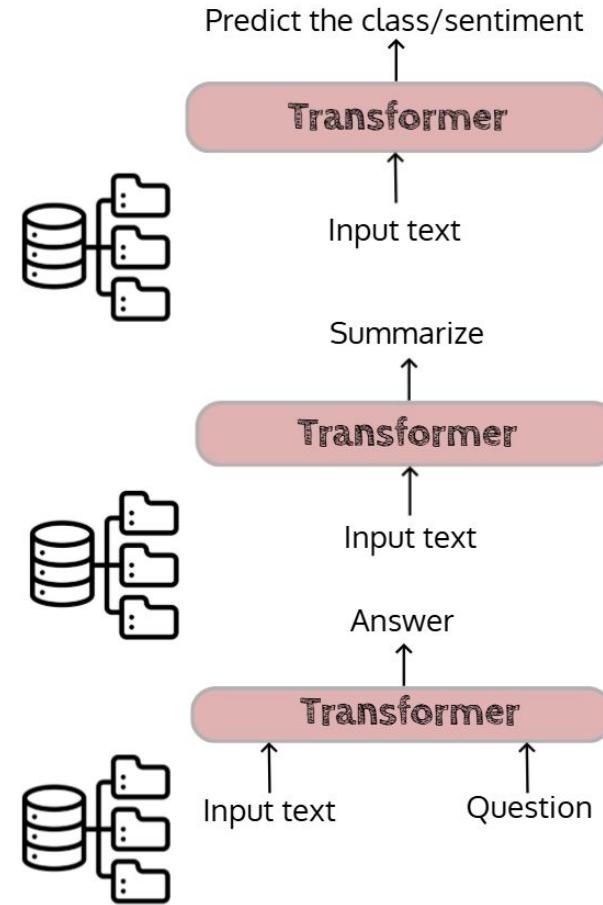
Transformer: Limitation

- If we train the architecture from scratch (that is, by randomly initializing the parameters) for each task, it takes a long time for convergence.
- Often, we may not have enough **labelled** samples for many NLP tasks.
- Moreover, preparing labelled data is laborious and costly.



Transformer: as a foundational model

- On the other hand,
- We have a large amount of unlabelled text easily available on the internet

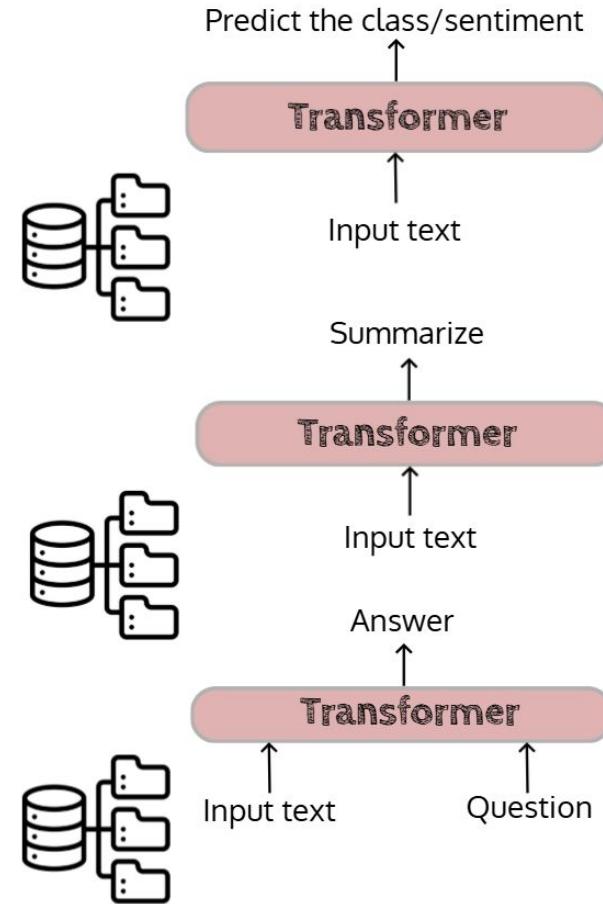


Transformer: as a foundational model

- On the other hand,
- We have a large amount of unlabelled text easily available on the internet



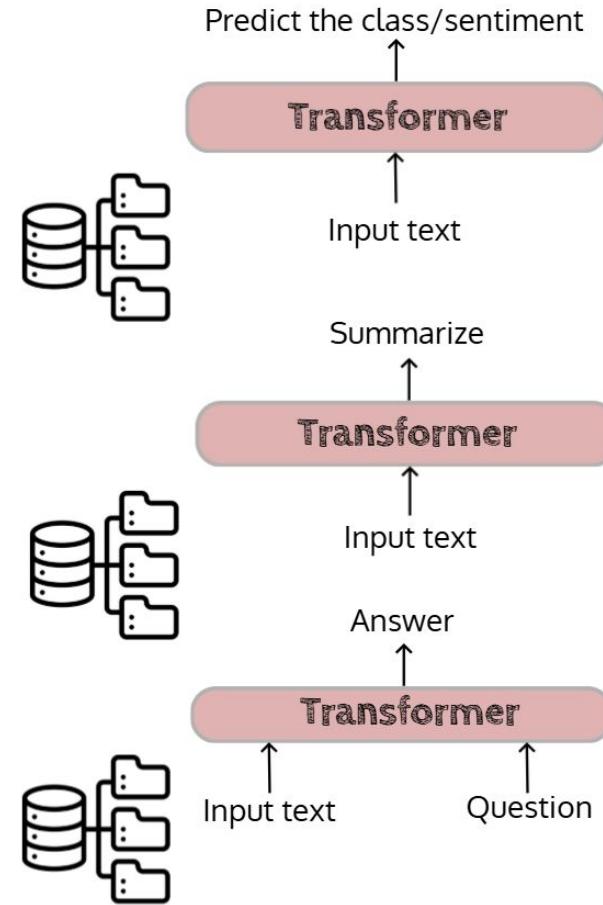
Can we make use of such unlabelled data to train a model?



Transformer: as a foundational model

Can we make use of such unlabelled data to train a model?

- What will be the training objective in that case?
- Would that be helpful in adapting the model to downstream tasks with minimal fine-tuning (with zero or a few samples)?



Motivation

Assume that we ask questions to a lay person?

- " Wow, India has now reached the moon"

What is the sentiment in above sentence?

Motivation

Assume that we ask questions to a lay person?

- "Wow, India has now reached the moon"
- What sets this mission apart is the pivotal role of artificial intelligence (AI) in guiding the spacecraft during its critical descent to the moon's surface.

What is the sentiment in above sentence?

Did the lander use AI for soft landing on the moon?

Motivation

Assume that we ask questions to a lay person?

- " Wow, India has now reached the moon"
- What sets this mission apart is the pivotal role of artificial intelligence (AI) in guiding the spacecraft during its critical descent to the moon's surface.
- He likes to stay
He likes to stray
He likes to sway

What is the sentiment in above sentence?

Did the lander use AI for soft landing on the moon?

Are these meaningful sentences?

- The person will most likely answer all the questions, even though he/she may not be explicitly trained on any of these tasks. **How?**

Motivation

Assume that we ask questions to a lay person?

- " Wow, India has now reached the moon"
- What sets this mission apart is the pivotal role of artificial intelligence (AI) in guiding the spacecraft during its critical descent to the moon's surface.
- He likes to stay
He likes to stray
He likes to sway

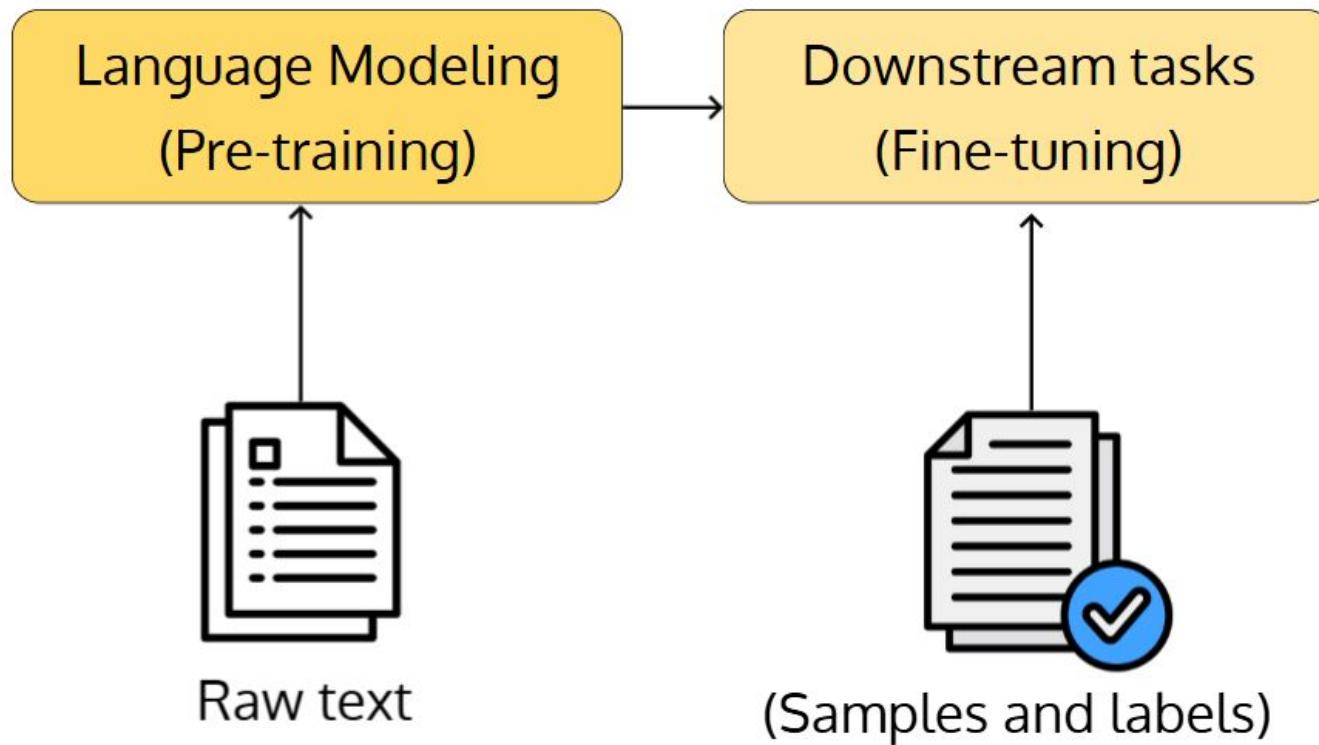
What is the sentiment in above sentence?

Did the lander use AI for soft landing on the moon?

Are these meaningful sentences?

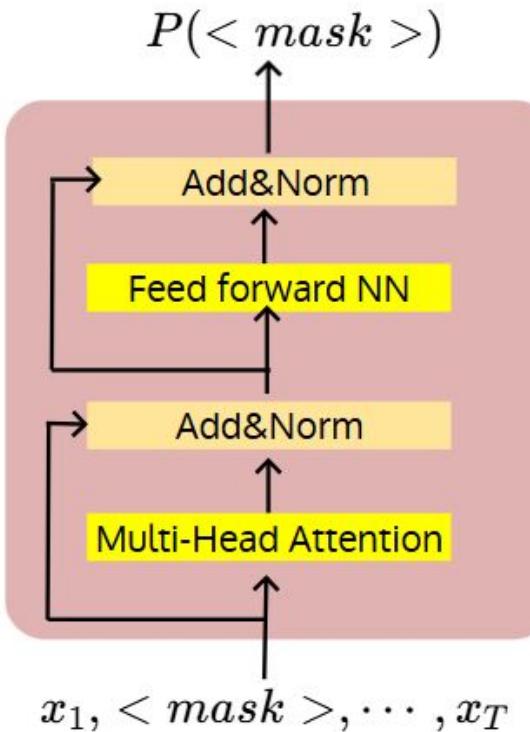
- The person will most likely answer all the questions, even though he/she may not be explicitly trained on any of these tasks. **How?**
- We develop a **strong understanding of language** through various language based interactions(listening/reading) over our life time without any explicit supervision

Idea



Solution 1: using only Encoder

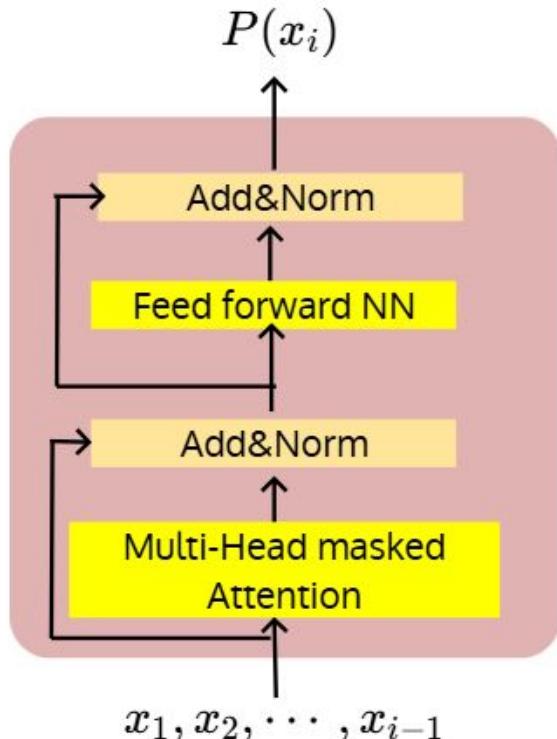
- Encoder only model is non-autoregressive.
- Can be used for masked language modeling.



Solution 2: using only Decoder

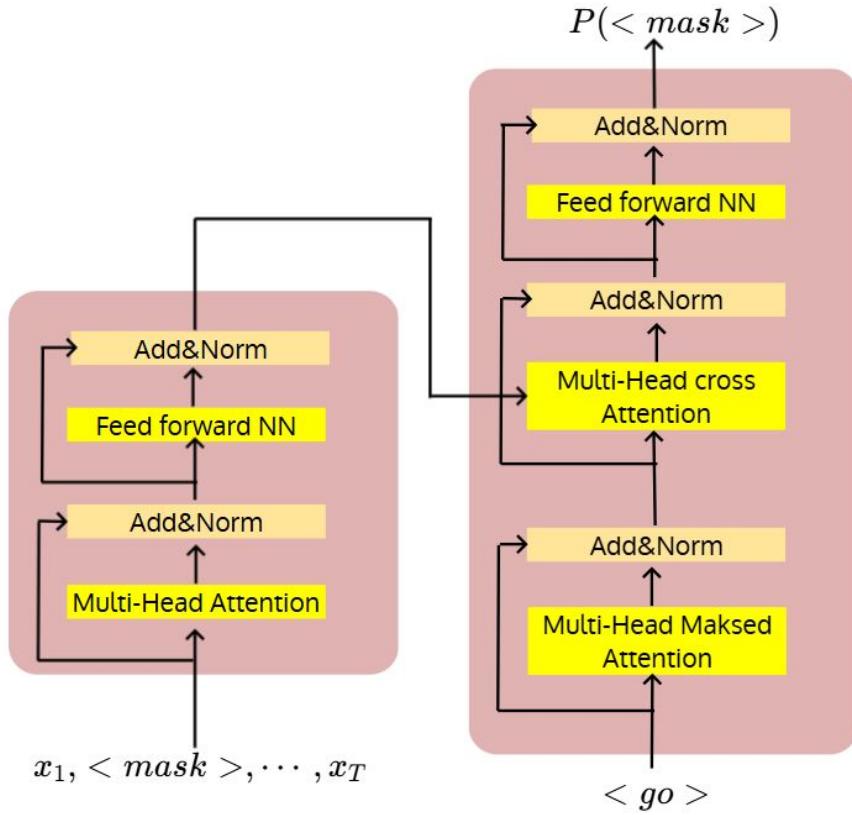
- Decoder only model is autoregressive.
- Can do the next word prediction task based on previous context.
 - Language model
 - Question Answer
 - Summarization

And so on.



Solution 3: using Encoder-Decoder

- Encoder-Decoder model for various NLP tasks.

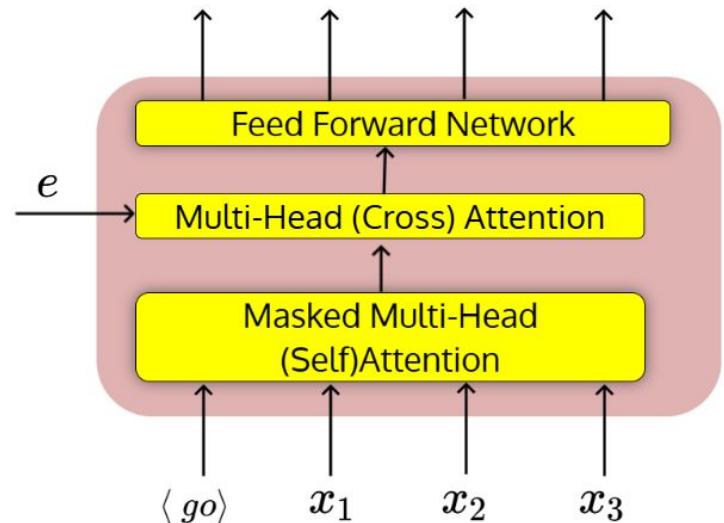


Decoder only Transformer

- The input is a sequence of words.
- We want the model to see only the present and past inputs.
- We can achieve this by applying the mask.

$$M = \begin{bmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

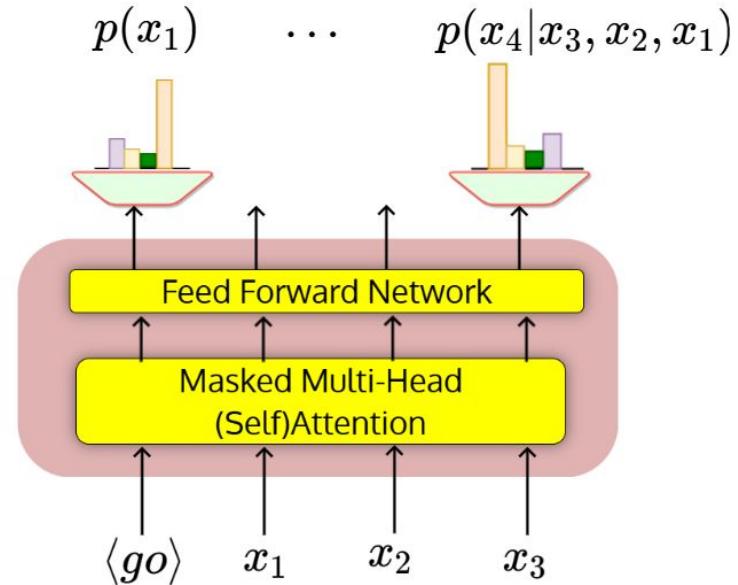
- The masked multi-head attention layer is required.
- However, we do not need multi-head cross attention layer (as there is no encoder).



Decoder only Transformer

- The outputs represent each term in the chain rule:

$$P(x_1)P(x_2|x_1)P(x_3|x_2, x_1)P(x_4|x_3, x_2, x_1)$$



Generative Pretrained Transformer

- In GPT, authors stacked (n) modified decoder layers.

Let, X denote the input sequence

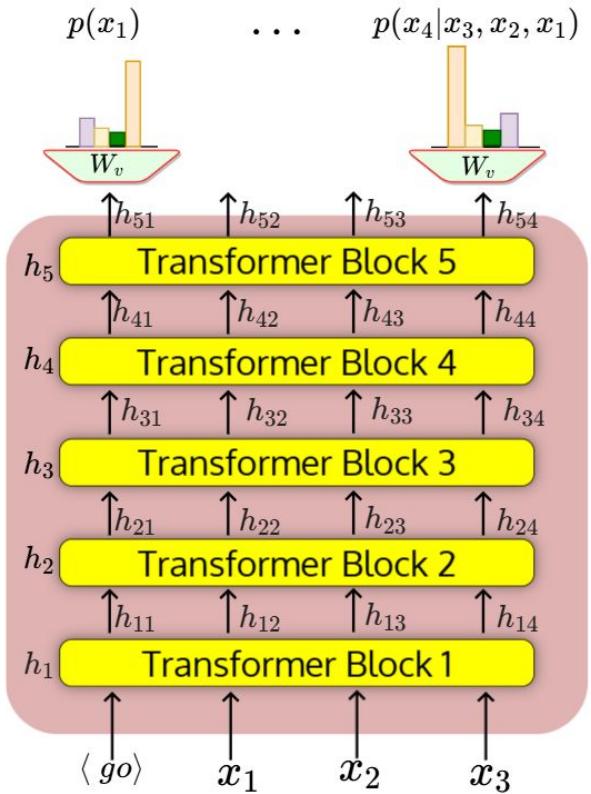
$$h_0 = X \in \mathbb{R}^{T \times d_{model}}$$

$$h_l = \text{transformer_block}(h_{l-1}), \forall l \in [1, n]$$

Where $h_n[i]$ is the i -the output vector in h_n block.

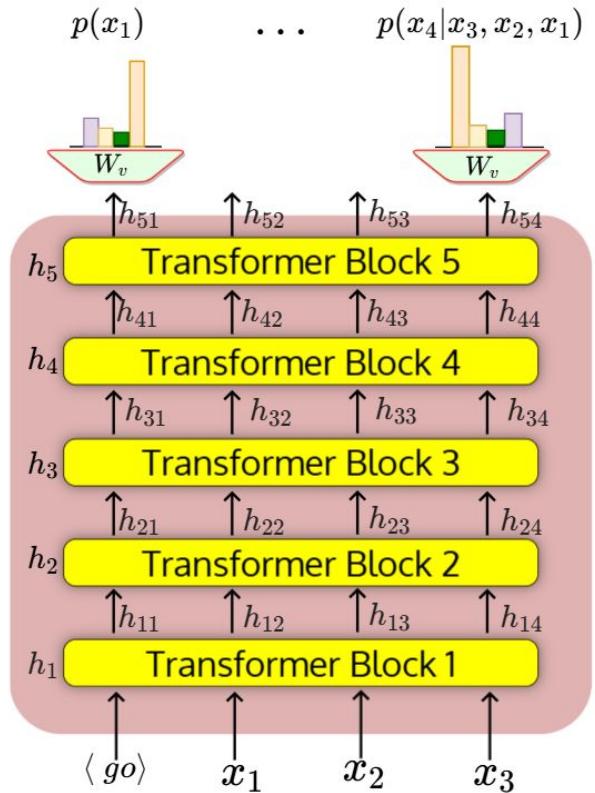
$$P(x_i) = \text{softmax}(h_n[i]W_v)$$

$$\mathcal{L} = \sum_{i=1}^T \log(P(x_i|x_1, \dots, x_{i-1}))$$



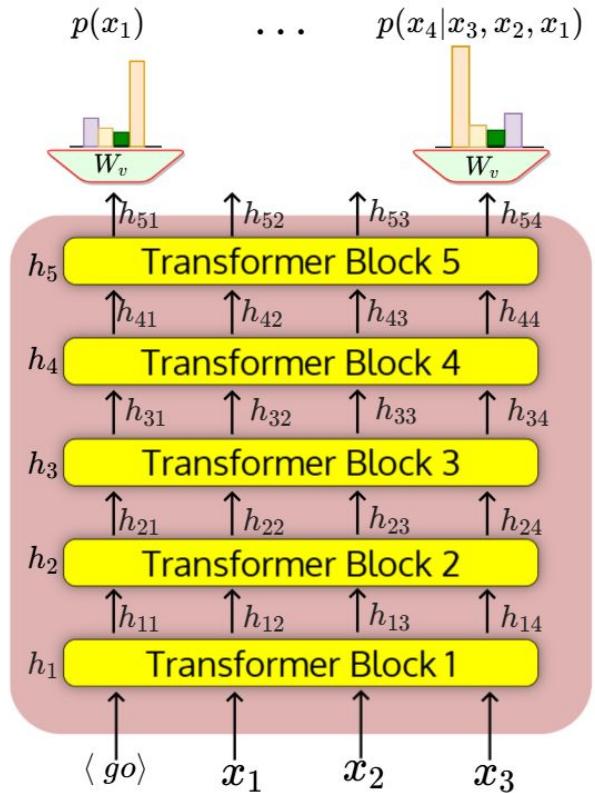
GPT: Input data

- The corpus contains 7000 unique books,
 - 74 Million sentences and
 - approximately 1 Billion words across 16 genres
- Vocab size: 40478
- Tokenizer: Byte Pair encoding
- Embedding dimension: 768



GPT: Model Architecture

- Contains 12 decoder layers (transformer blocks)
- Context size: 512
- Attention head: 12
- FFN hidden layer size: $768 * 4 = 3072$
- Activation: Gussaian Error Linear Unit
- Dropout, layer normalization, and residual connections were implemented to enhance convergence during training.



GPT: end to end flow

At the Bell Labs Hamming shared an office for a time with Claude Shannon. The Mathematical Research Department also included John Tukey and Los Alamos veterans Donald Ling and Brockway McMillan. Shannon, Ling, McMillan and Hamming came to call themselves the Young Turks.^[4] "We were first-class troublemakers," Hamming later recalled. "We did unconventional things in unconventional ways and still got valuable results. Thus management had to tolerate us and let us alone a lot of the time."^[2]

Although Hamming had been hired to work on elasticity theory, he still spent much of his time with the calculating machines.^[7] Before he went home on one Friday in 1947, he set the machines to perform a long and complex series of calculations over the weekend, only to find when he arrived on Monday morning that an error had occurred early in the process and the calculation had errored off.^[8] Digital machines manipulated information as sequences of zeroes and ones, units of information that Tukey would christen "bits."^[9] If a single bit in a sequence was wrong, then the whole sequence would be. To detect this, a parity bit was used to verify the correctness of each sequence. "If the computer can tell when an error has occurred," Hamming reasoned, "surely there is a way of telling where the error is so that the computer can correct the error itself."^[8]

Hamming set himself the task of solving this problem,^[3] which he realised would have an enormous range of applications. Each bit can only be a zero or a one, so if you know which bit is wrong, then it can be corrected. In a landmark paper published in 1950, he introduced a concept of the number of positions in which two code words differ, and therefore how many changes are required to transform one code word into another, which is today known as the Hamming distance.^[10] Hamming thereby created a family of mathematical error-correcting codes, which are called Hamming codes. This not only solved an important problem in telecommunications and computer science, it opened up a whole new field of study.^{[10][11]}

The Hamming bound, also known as the sphere-packing or volume bound is a limit on the parameters of an arbitrary block code. It is from an interpretation in terms of sphere packing in the Hamming distance into the space of all possible words. It gives an important limitation on the efficiency with which any error-correcting code can utilize the space in which its code words are embedded. A code which attains the Hamming bound is said to be a perfect code. Hamming codes are perfect codes.^{[12][13]}

Returning to differential equations, Hamming studied means of numerically integrating them. A popular approach at the time was Milne's Method, attributed to Arthur Milne.^[14] This had the drawback of being unstable, so that under certain conditions the result could be swamped by roundoff noise. Hamming developed an improved version, the Hamming predictor-corrector. This was in use for many years, but has since been superseded by the Adams method.^[15] He did extensive research into digital filters, devising a new filter, the Hamming window, and eventually writing an entire book on the subject, *Digital Filters* (1977).^[16]

A sample Data

GPT: end to end flow

Transformer Block 1

At the Bell Labs Hamming shared an office for a time with Claude Shannon. The Mathematical Research Department also included John Tukey and Los Alamos veterans Donald Ling and Brockway McMillan. Shannon, Ling, McMillan and Hamming came to call themselves the Young Turks.^[4] "We were first-class troublemakers," Hamming later recalled. "We did unconventional things in unconventional ways and still got valuable results. Thus management had to tolerate us and let us alone a lot of the time."^[2]

Although Hamming had been hired to work on elasticity theory, he still spent much of his time with the calculating machines.^[7] Before he went home on one Friday in 1947, he set the machines to perform a long and complex series of calculations over the weekend, only to find when he arrived on Monday morning that an error had occurred early in the process and the calculation had errored off.^[8] Digital machines manipulated information as sequences of zeroes and ones, units of information that Tukey would christen "bits".^[9] If a single bit in a sequence was wrong, then the whole sequence would be. To detect this, a parity bit was used to verify the correctness of each sequence. "If the computer can tell when an error has occurred," Hamming reasoned, "surely there is a way of telling where the error is so that the computer can correct the error itself."^[8]

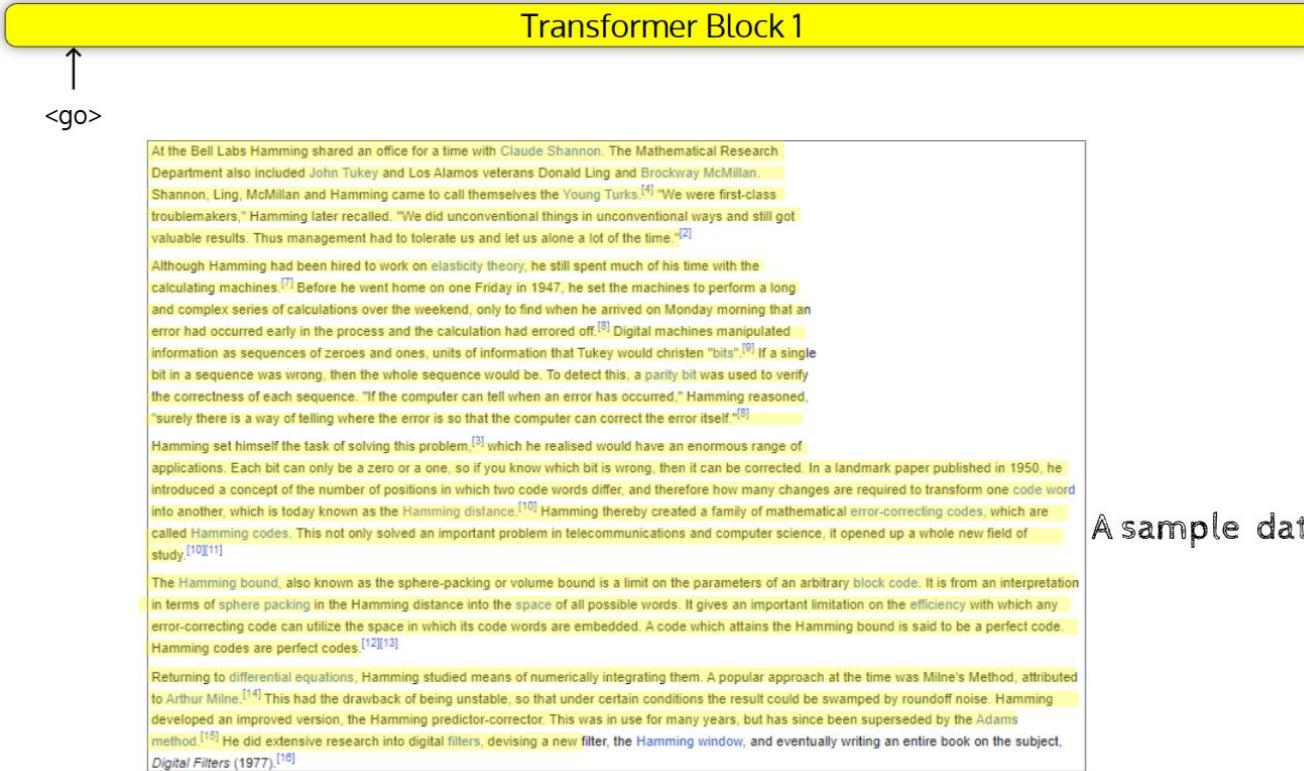
Hamming set himself the task of solving this problem,^[3] which he realised would have an enormous range of applications. Each bit can only be a zero or a one, so if you know which bit is wrong, then it can be corrected. In a landmark paper published in 1950, he introduced a concept of the number of positions in which two code words differ, and therefore how many changes are required to transform one code word into another, which is today known as the Hamming distance.^[10] Hamming thereby created a family of mathematical error-correcting codes, which are called Hamming codes. This not only solved an important problem in telecommunications and computer science, it opened up a whole new field of study.^{[10][11]}

The Hamming bound, also known as the sphere-packing or volume bound is a limit on the parameters of an arbitrary block code. It is from an interpretation in terms of sphere packing in the Hamming distance into the space of all possible words. It gives an important limitation on the efficiency with which any error-correcting code can utilize the space in which its code words are embedded. A code which attains the Hamming bound is said to be a perfect code. Hamming codes are perfect codes.^{[12][13]}

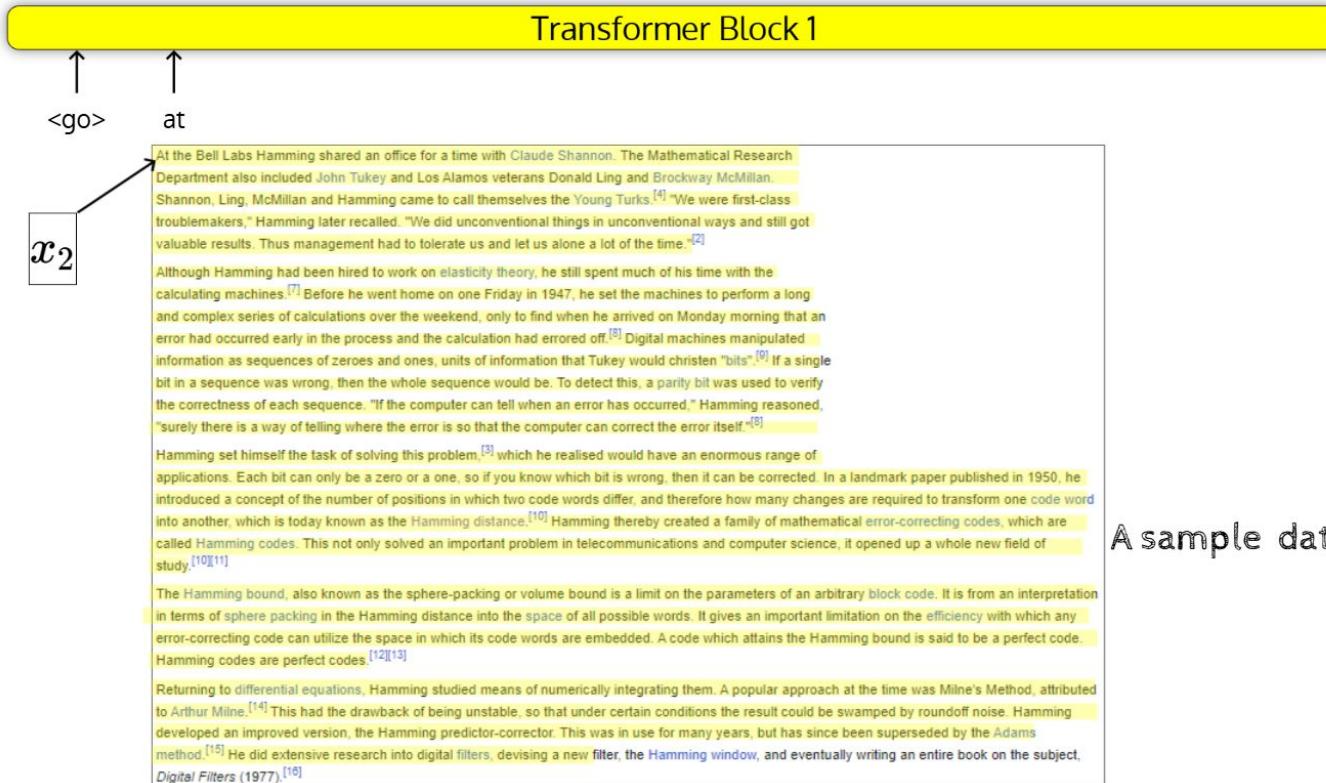
Returning to differential equations, Hamming studied means of numerically integrating them. A popular approach at the time was Milne's Method, attributed to Arthur Milne.^[14] This had the drawback of being unstable, so that under certain conditions the result could be swamped by roundoff noise. Hamming developed an improved version, the Hamming predictor-corrector. This was in use for many years, but has since been superseded by the Adams method.^[15] He did extensive research into digital filters, devising a new filter, the Hamming window, and eventually writing an entire book on the subject, *Digital Filters* (1977).^[16]

A sample Data

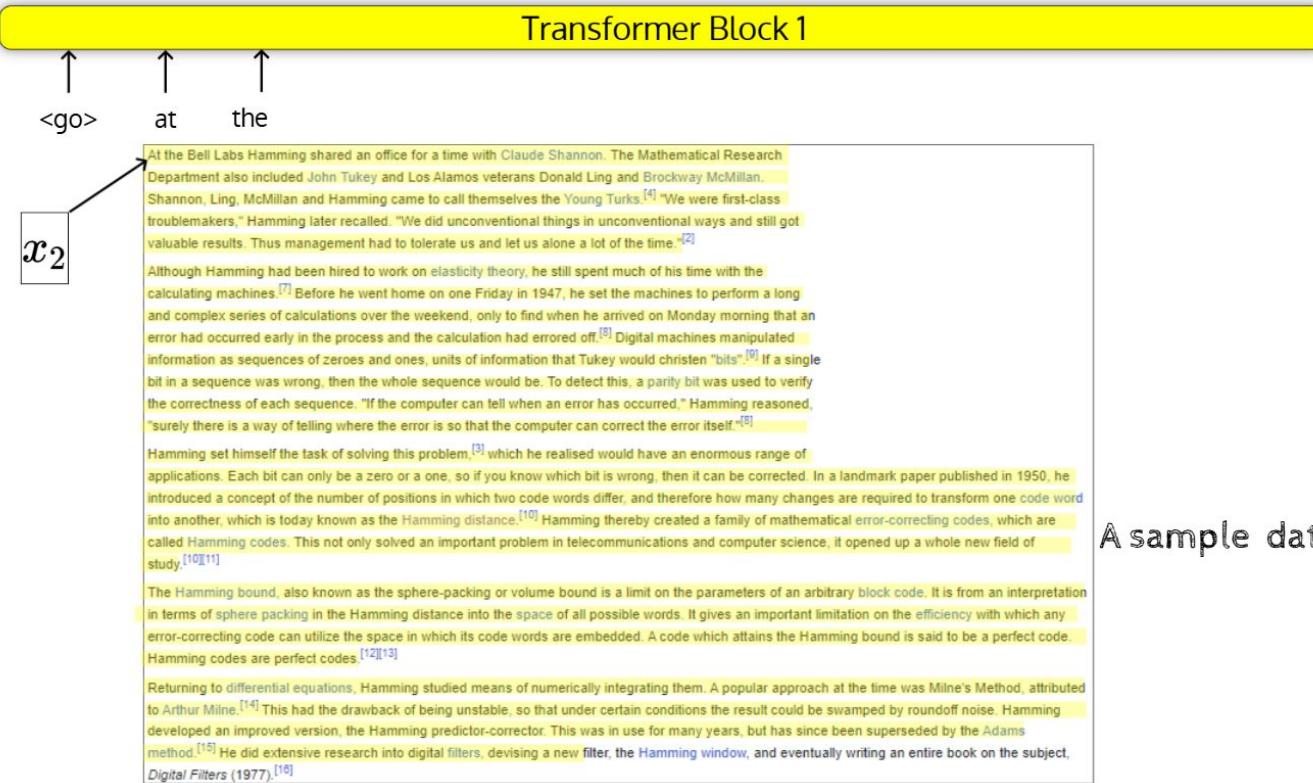
GPT: end to end flow



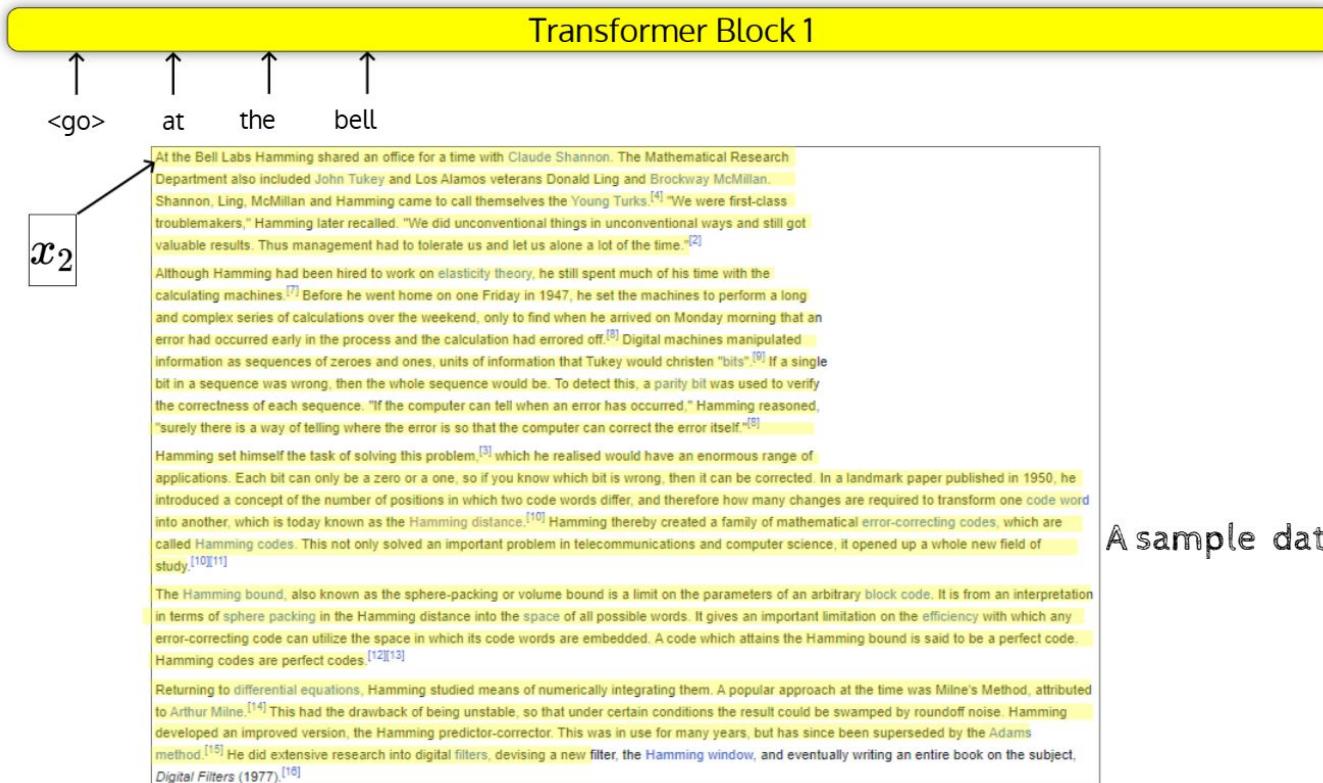
GPT: end to end flow



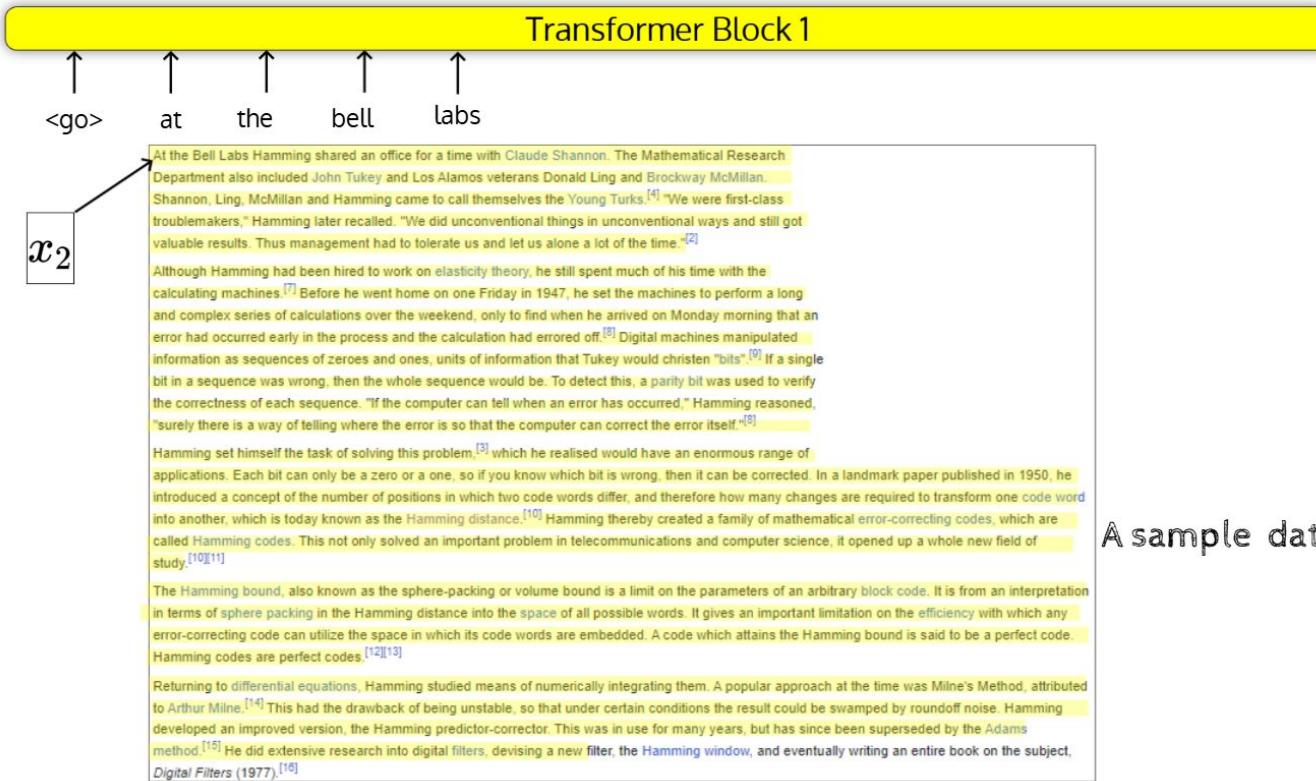
GPT: end to end flow



GPT: end to end flow

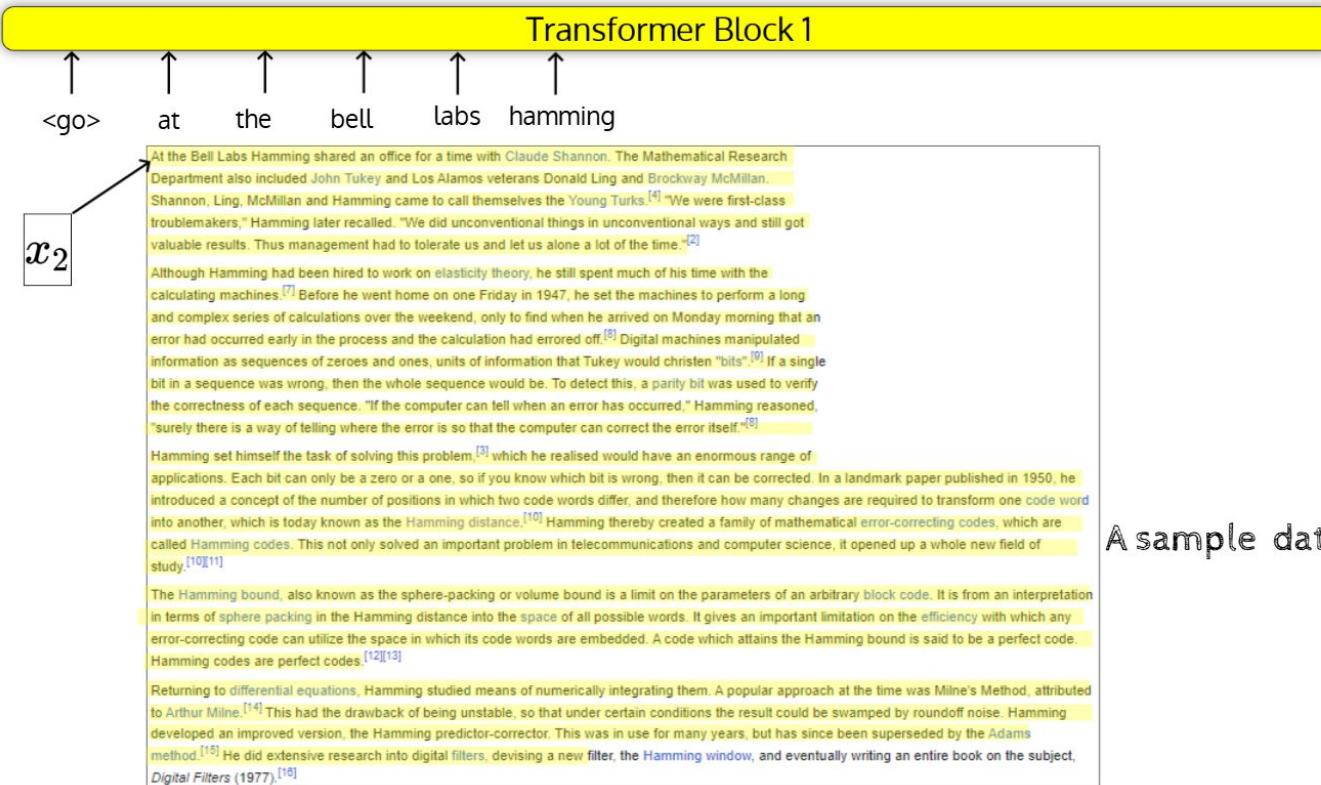


GPT: end to end flow

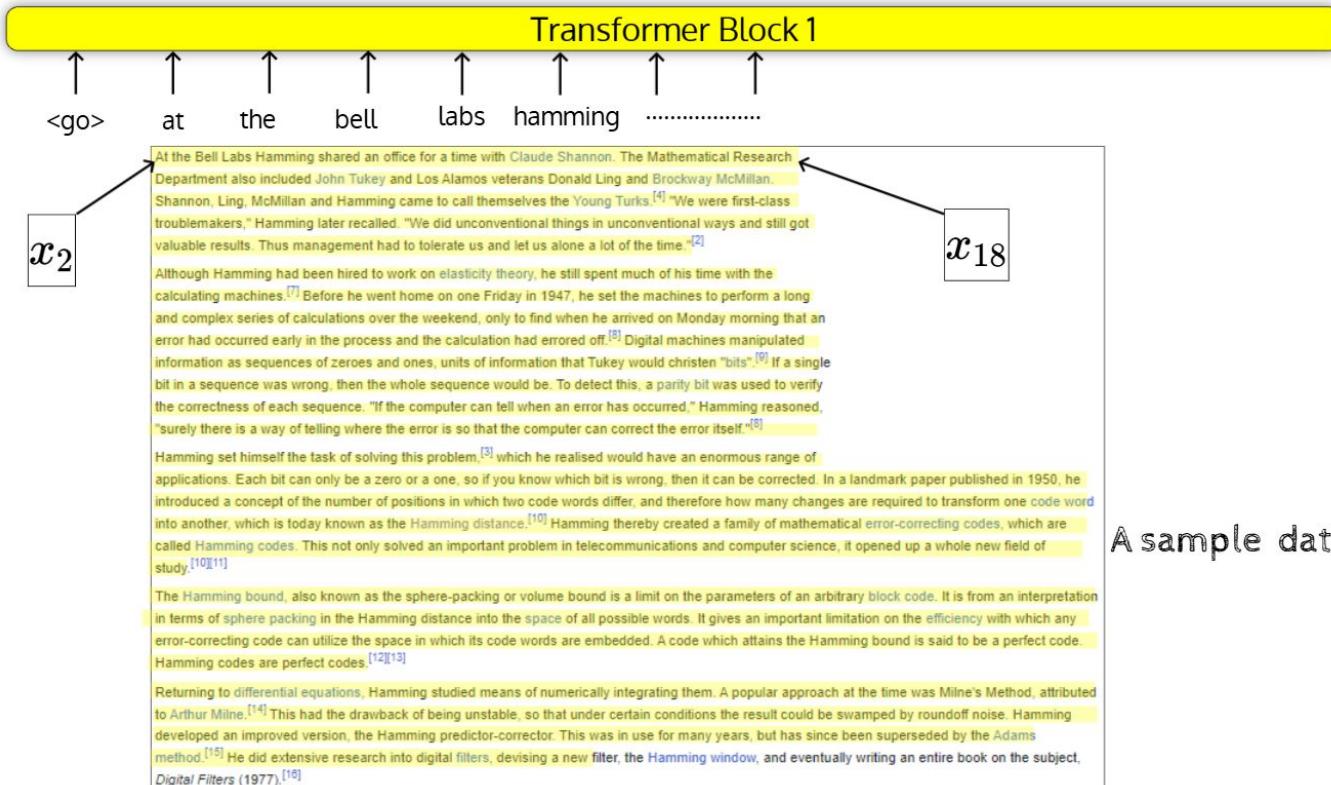


A sample data

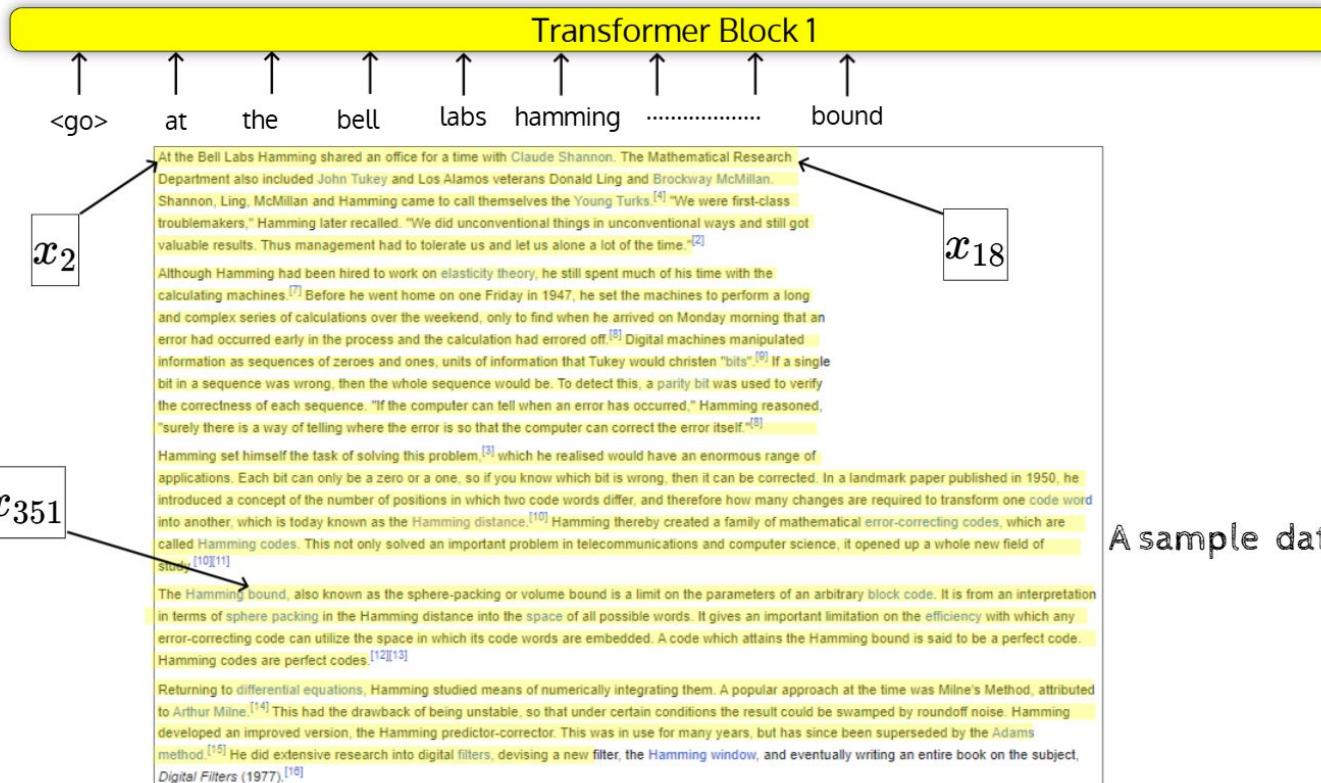
GPT: end to end flow



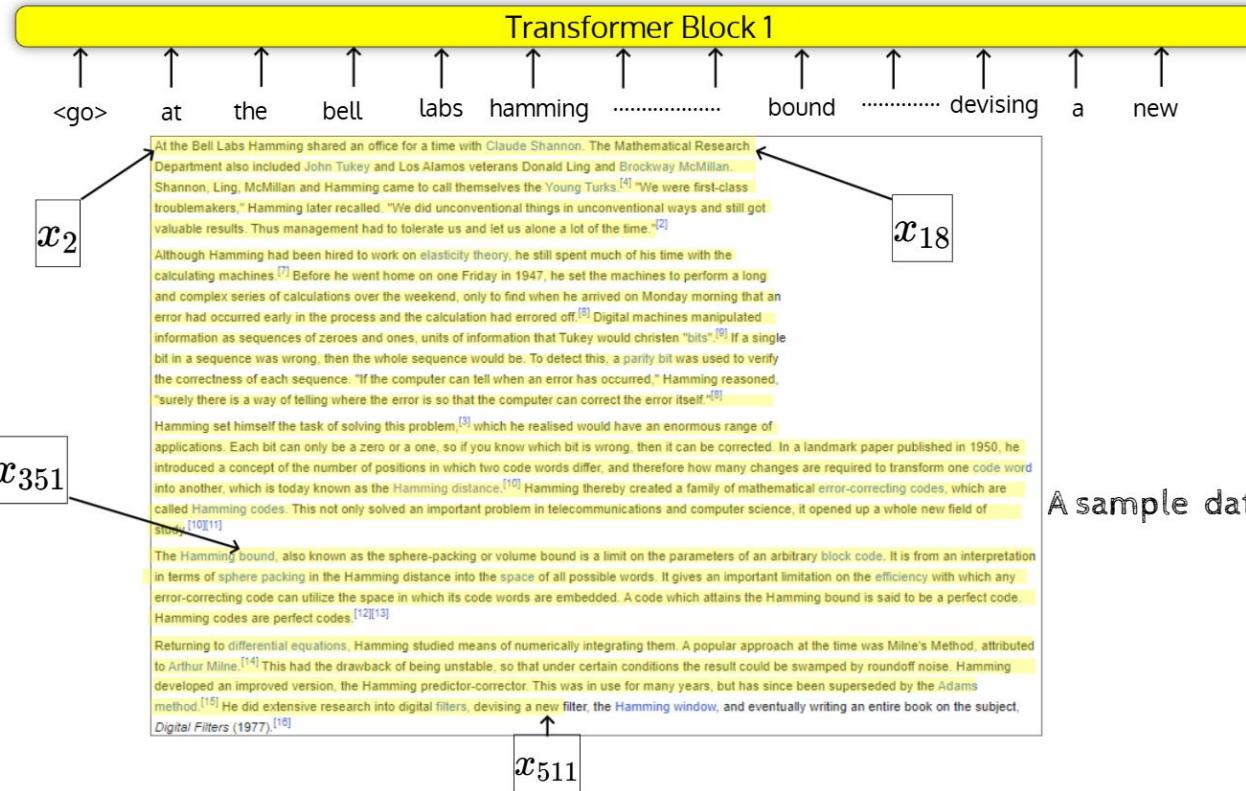
GPT: end to end flow



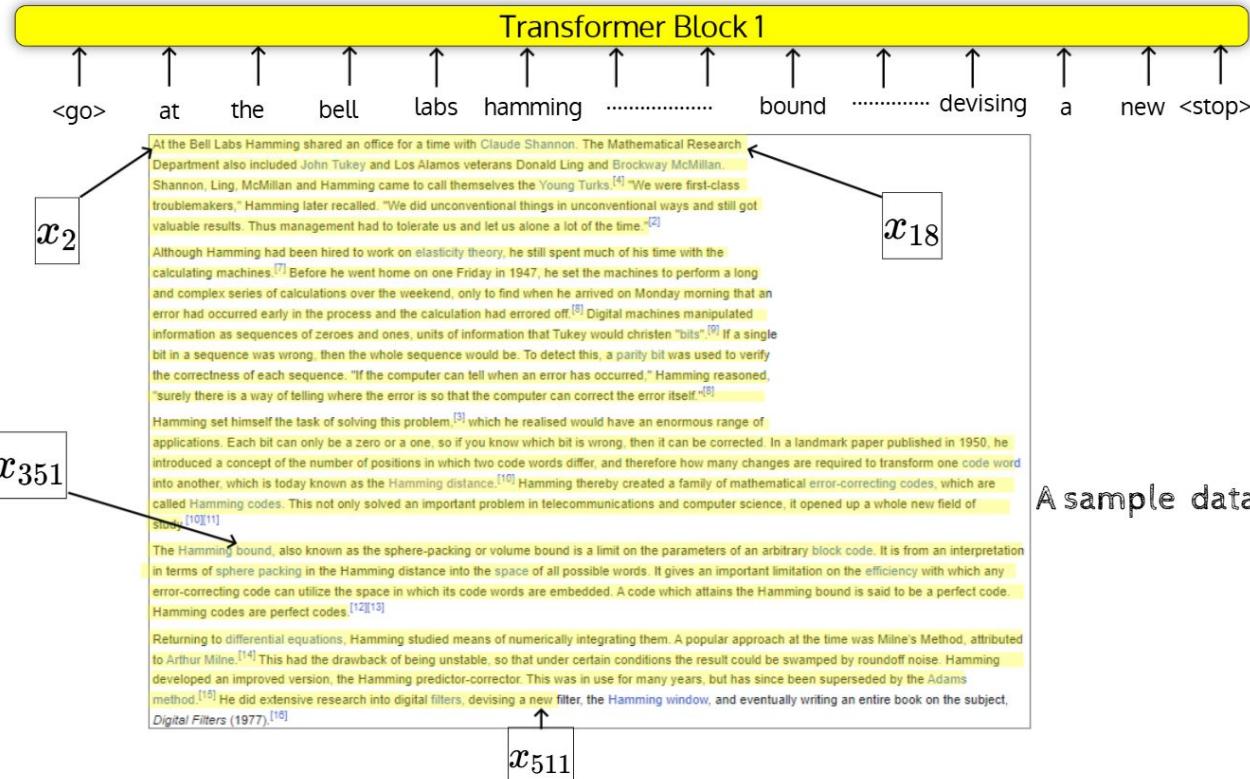
GPT: end to end flow



GPT: end to end flow



GPT: end to end flow



GPT: end to end flow

Multi-head masked attention

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
<go> at the bell labs hamming bound devising a new <stop>

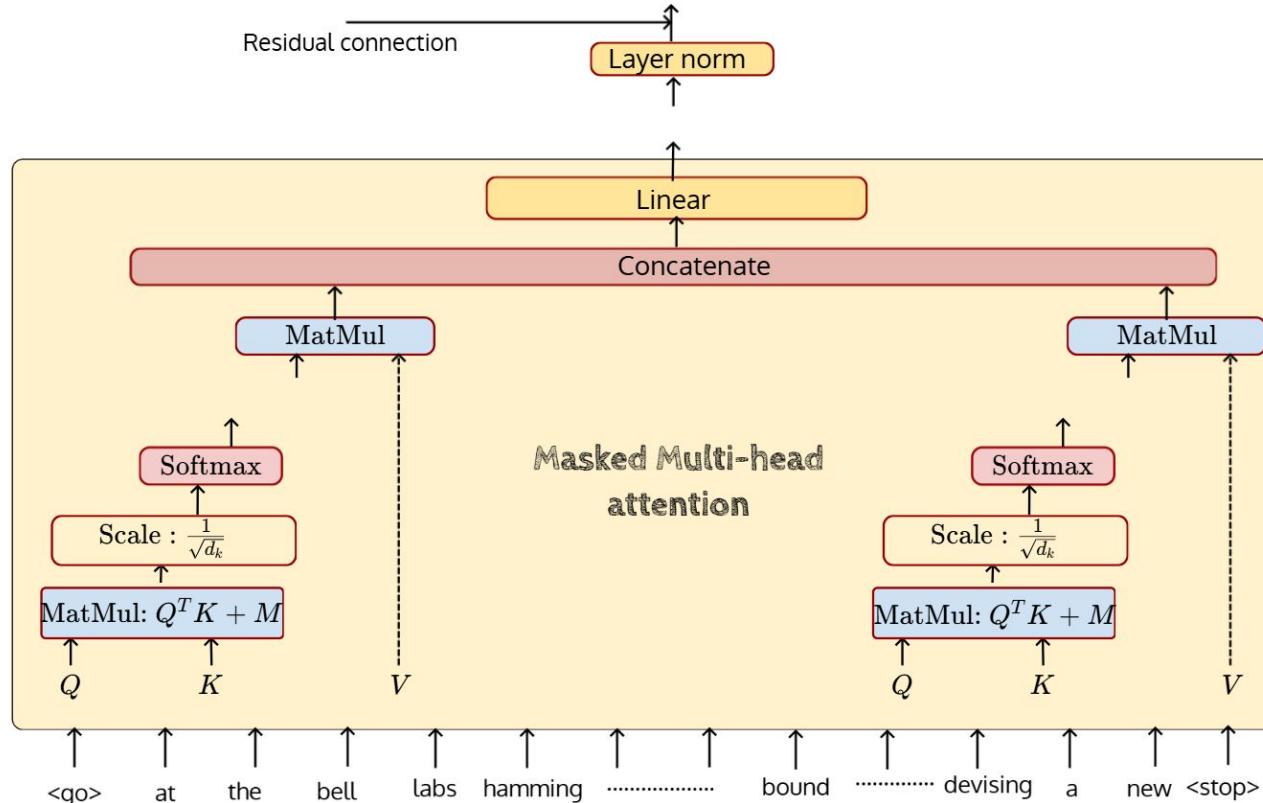
GPT: end to end flow

Feed Forward Neural Network

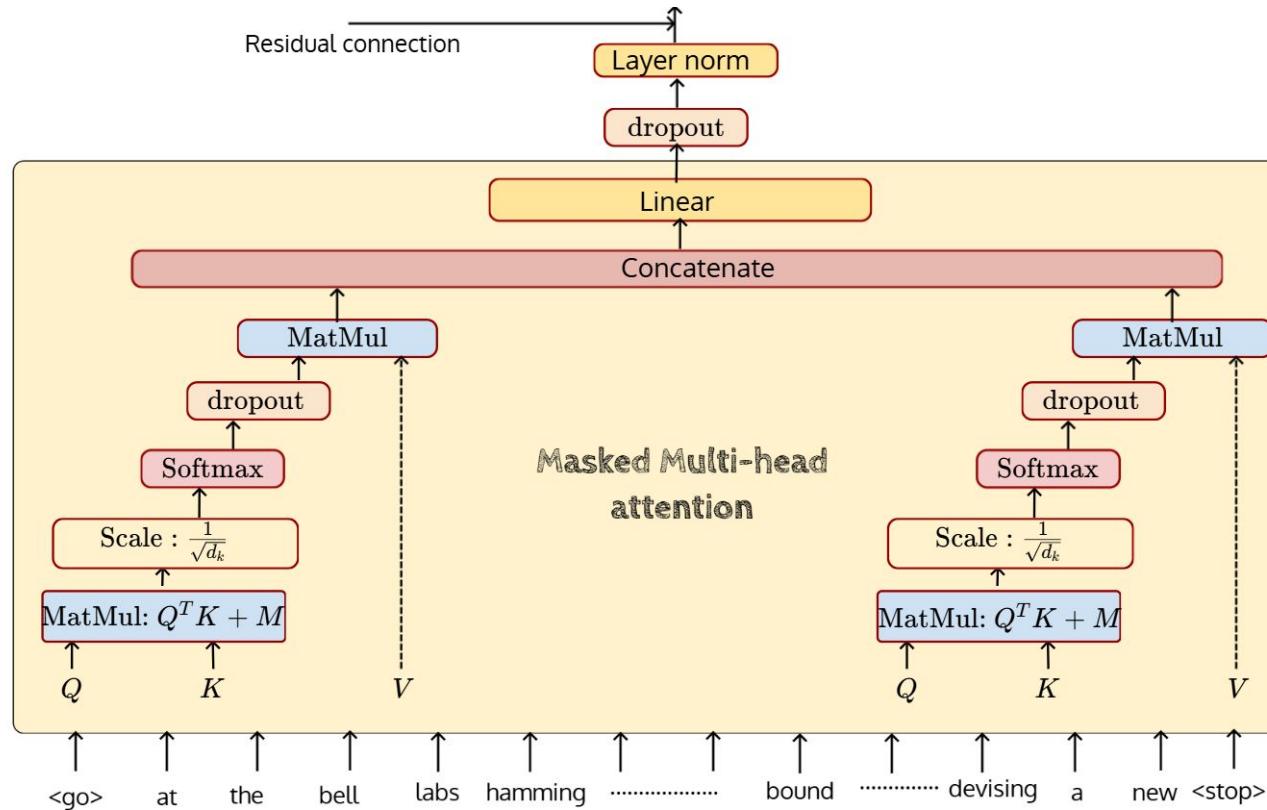
Multi-head masked attention

↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
<go> at the bell labs hamming bound devising a new <stop>

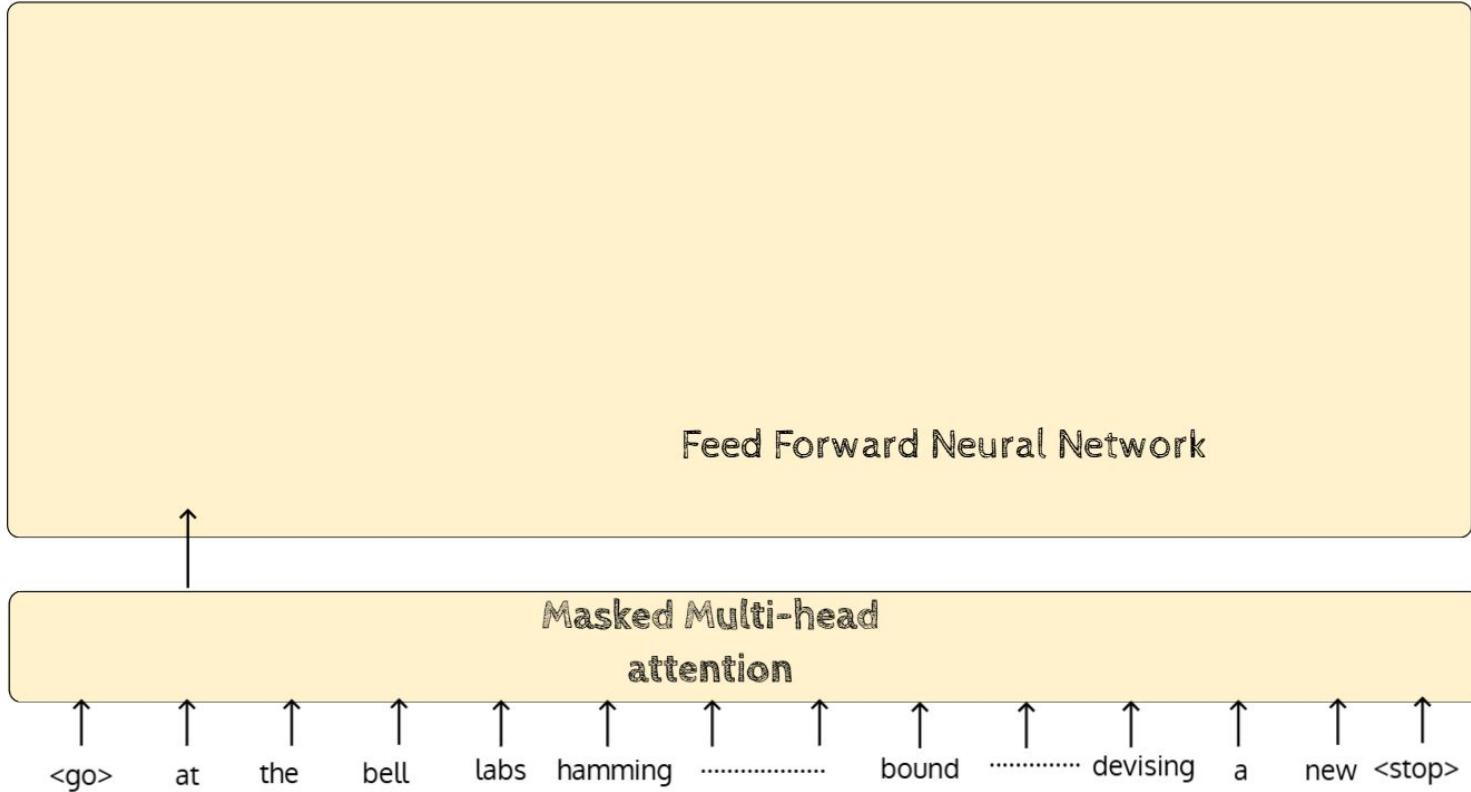
GPT: end to end flow



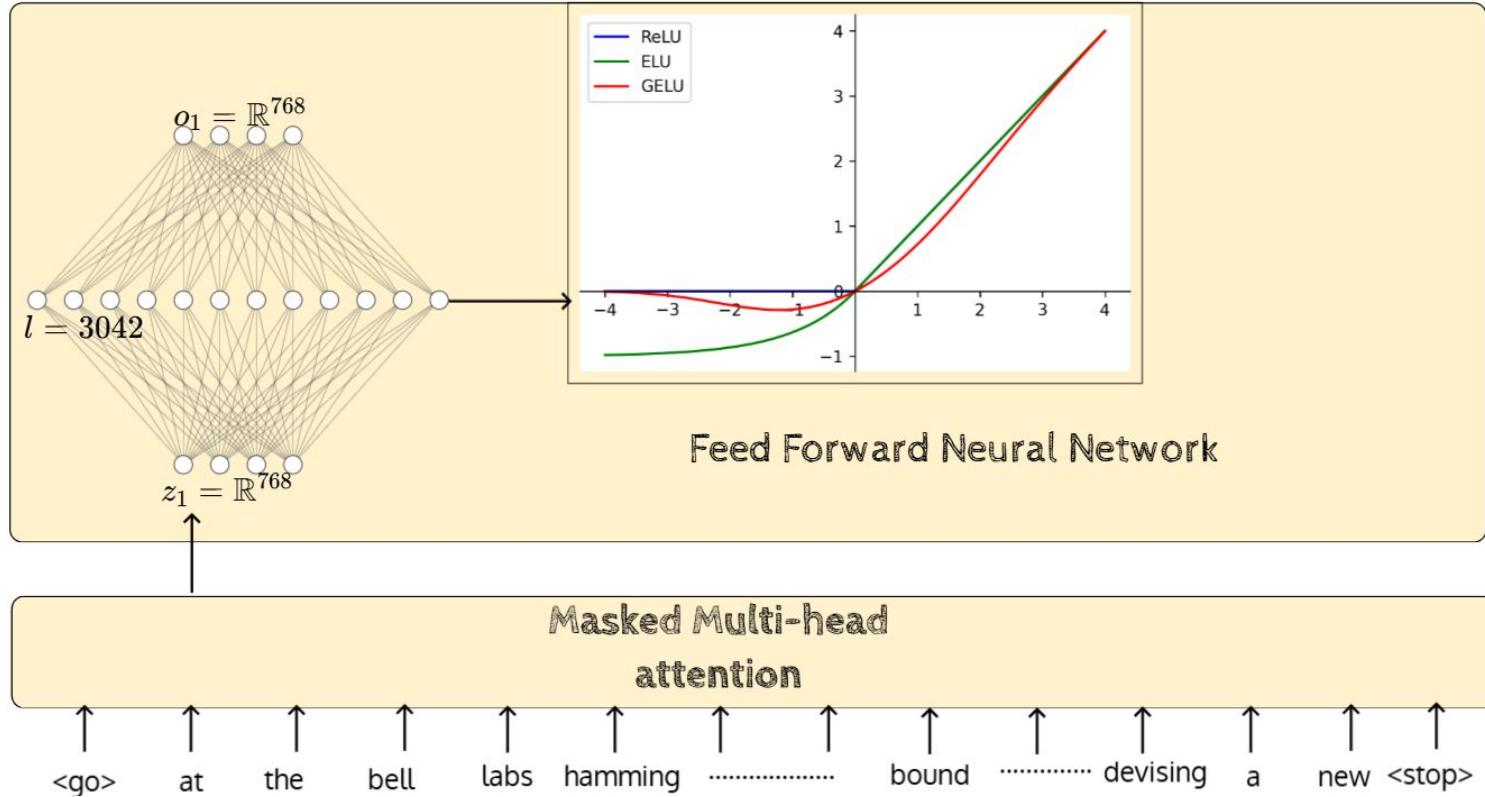
GPT: end to end flow



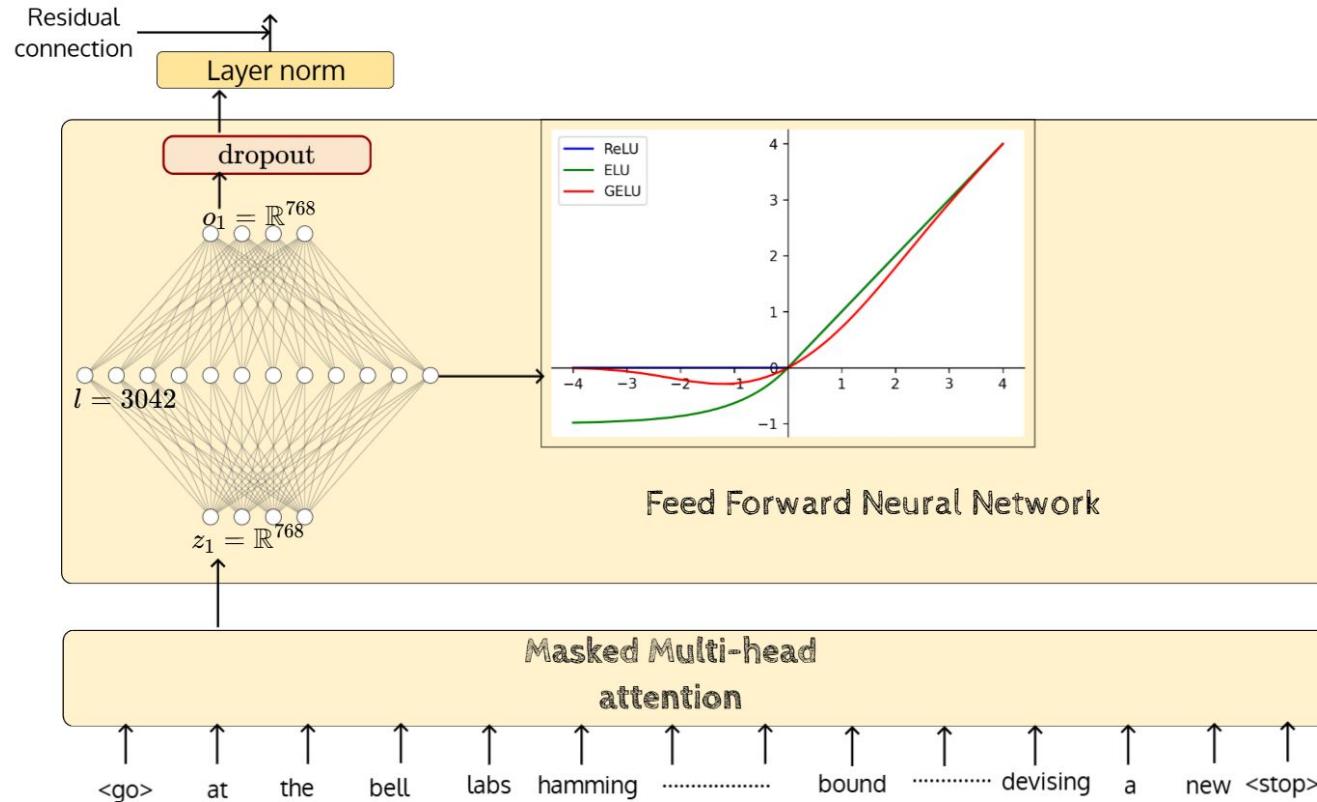
GPT: end to end flow



GPT: end to end flow



GPT: end to end flow



GPT: no. of parameters

The positional embeddings are also learned, unlike the original transformer which uses fixed sinusoidal embeddings to encode the positions.

- **Total Embeddings:**

⇒ $|V| * \text{embedding dimension:}$

⇒ $40478 * 768$

⇒ $31 * 10^6$

⇒ 31M

- **Position Embeddings:**

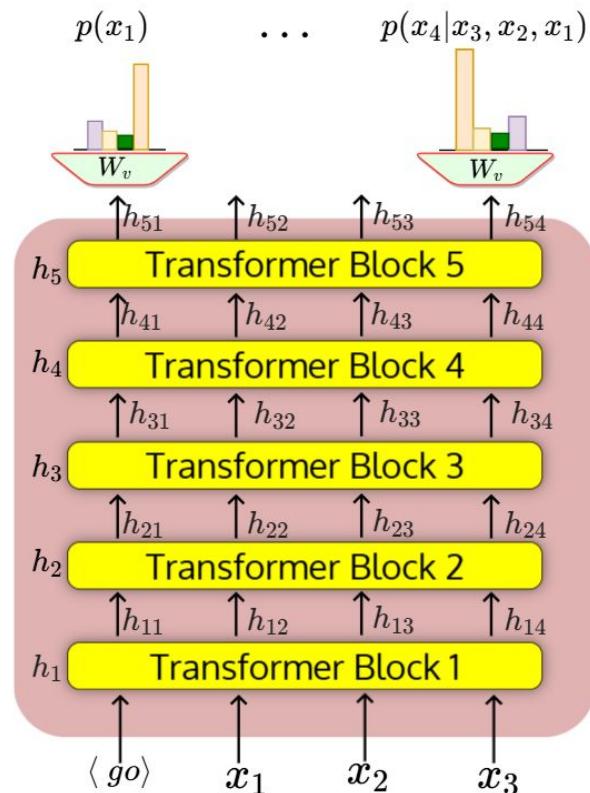
⇒ Context length * Embedding dimension

⇒ $512 * 768$

⇒ $0.3 * 10^6$

⇒ 0.3M

Total : $31M + 0.3M \Rightarrow 31.3M$



GPT: no. of parameters

- Attention Parameter per block:

$$W_q = W_k = W_v = (768 \times 64)$$

Per attention head

$$\Rightarrow 3 \times (768 \times 64) \approx 147 \times 10^3$$

For 12 heads

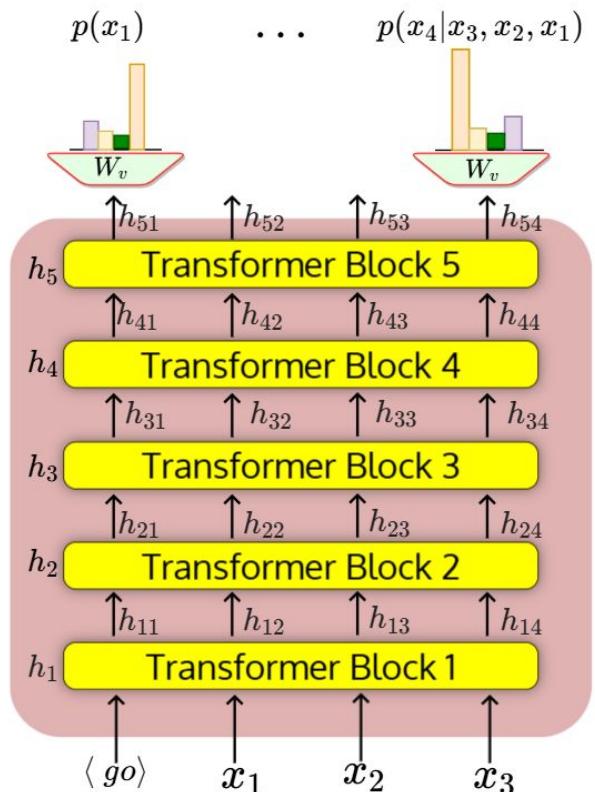
$$\Rightarrow 12 \times 147 \times 10^3 \approx 1.7M$$

For a linear layer

$$\Rightarrow 768 \times 768 \approx 0.6M$$

For all 12 blocks

$$\Rightarrow 12 \times 2.3M = 27.6M$$



GPT: no. of parameters

- FFN Parameter per block:

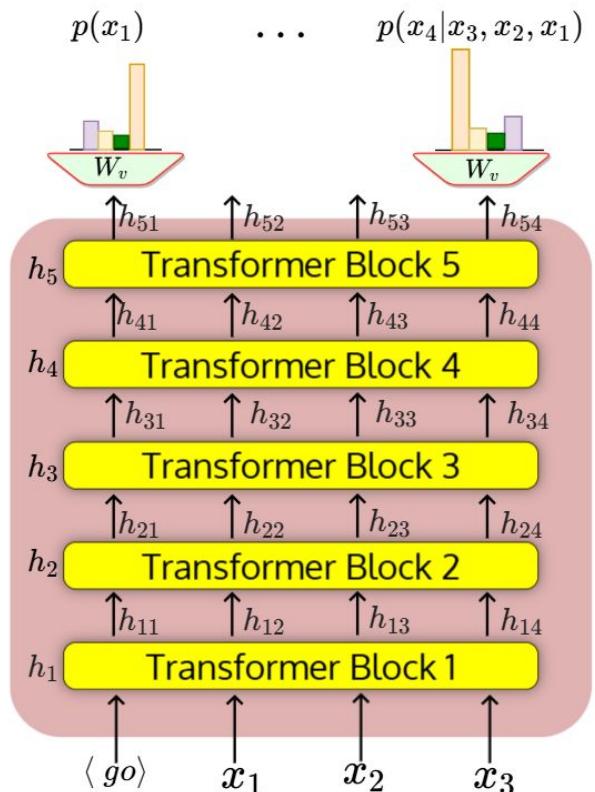
$$2 \times (768 \times 3072) + 3072 + 768$$

$$\Rightarrow 4.7 \times 10^6$$

$$\Rightarrow 4.7\text{M}$$

For all 12 blocks

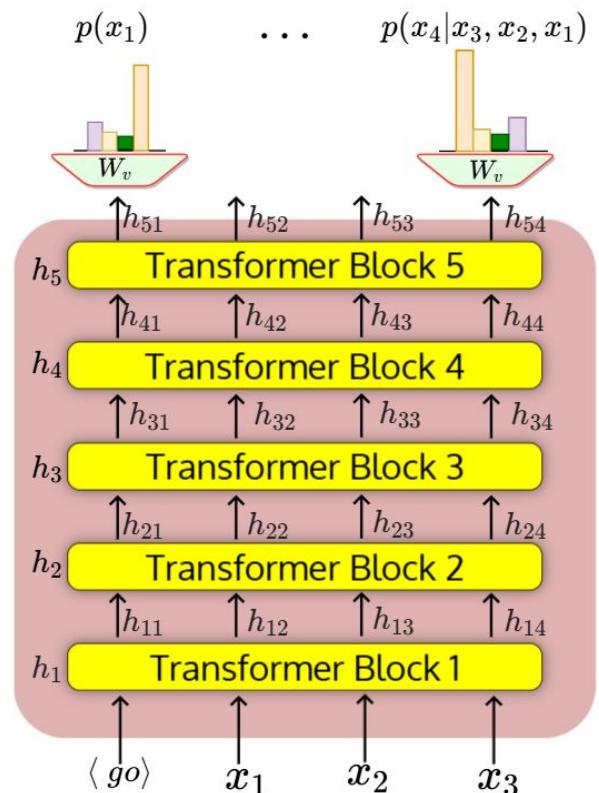
$$\Rightarrow 12 \times 4.7\text{M} = 56.4\text{M}$$



GPT: no. of parameters

Layer	Parameters (in Millions)
Embedding Layer	31.3
Attention layers	27.6
FFN Layers	56.4
Total	116.461056*

That means GPT-1 has around 117M parameters



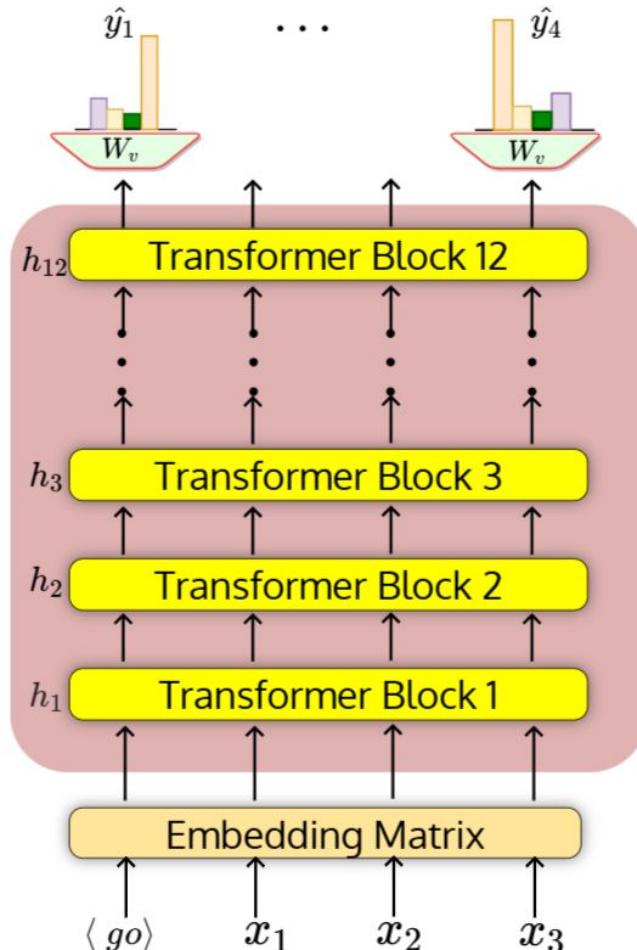
GPT: Pre-training

Batch Size: 64

Input size: $(B, T, C) = (64, 512, 768)$, where, C is an embedding dimension

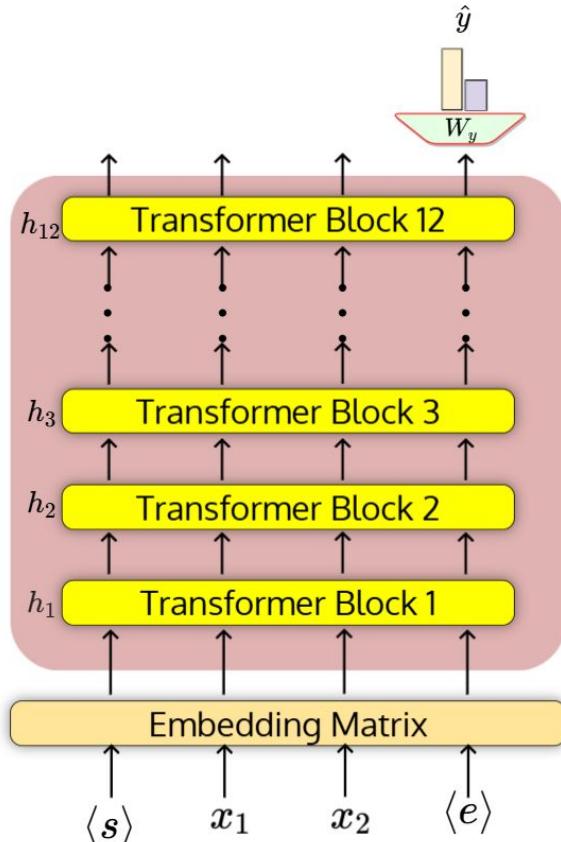
Optimizer: Adam

Strategy: Teacher forcing (instead of auto-regressive training) for quicker and stable convergence



GPT: Fine Tuning

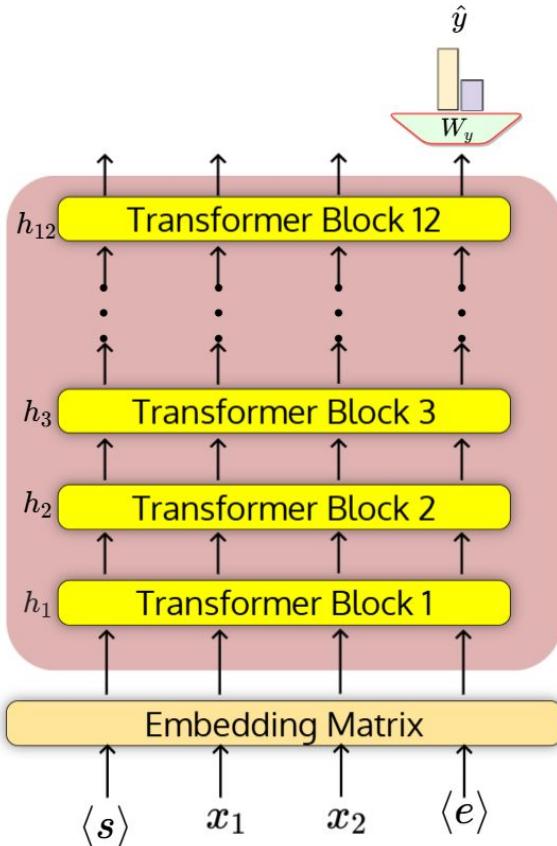
- Fine-tuning involves adapting model for various downstream tasks (with a minimal change in the architecture).
- Each sample in a labelled data set C consists of a sequence of tokens x_1, x_2, \dots, x_m with the label y .
- Initialize the parameters with the parameters learned by solving the pre-training objective.
- At the input side, add additional tokens based on the type of downstream task. For example, start $\langle s \rangle$ and end $\langle e \rangle$ tokens for classification tasks.
- At the output side, replace the pre-training LM head with the classification head (a linear layer W_y).



GPT: Fine Tuning

- Now our objective is to predict the label of the input sequence
- Note that we take the output representation at the last time step of the last layer
- It makes sense as the entire sentence is encoded only at the last time step due to causal masking.
- Then we can minimize the following objective

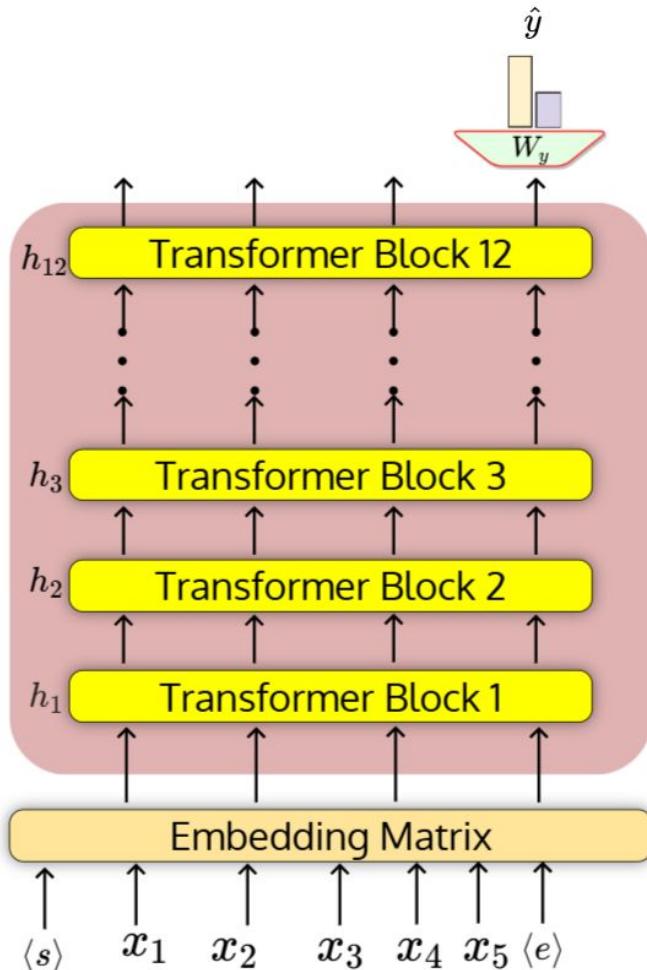
$$\mathcal{L} = - \sum_{(x,y)} \log(\hat{y}_i)$$



GPT: Fine Tuning for Sentiment Analysis

- **Text:** Wow, India has now reached the moon
- **Sentiment:** Positive

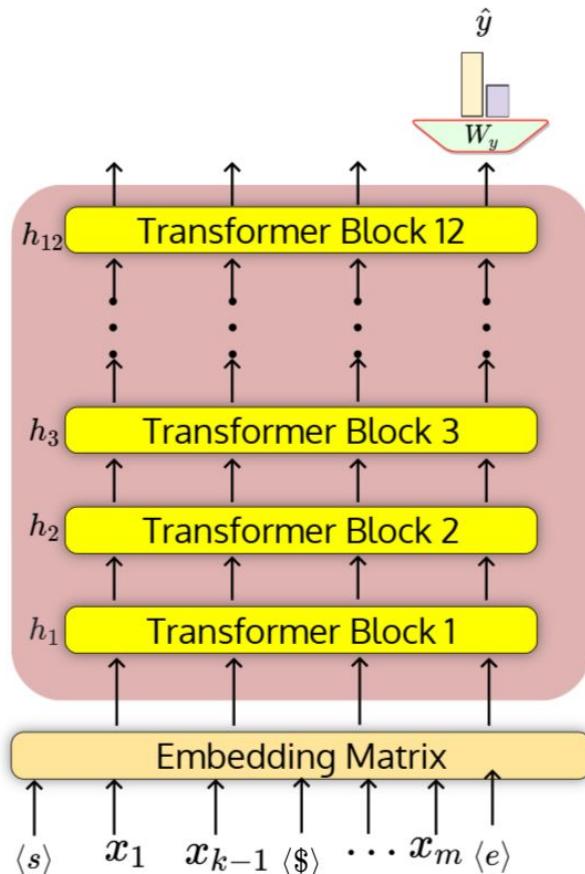
$$\hat{y} \in \{+1, -1\}$$



GPT: Fine Tuning for Textual Entailment/Contradiction

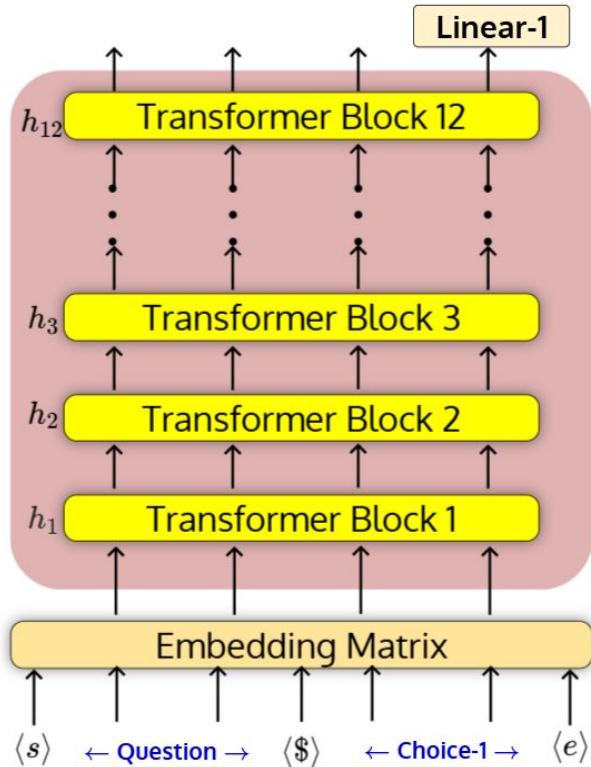
- **Text:** A soccer game with multiple males playing
- **Hypothesis:** Some men are playing a sport
- **Entailment:** True

In this case, we need to use a delimiter token $\langle \$ \rangle$ to differentiate the text from the hypothesis



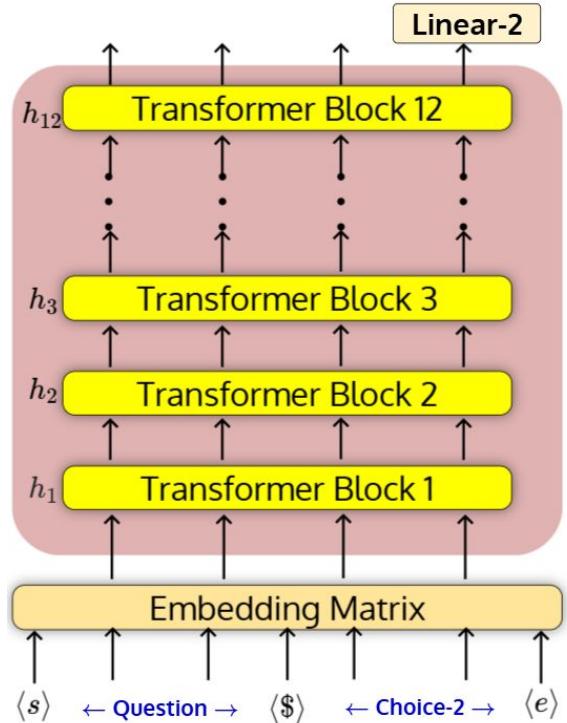
GPT: Fine Tuning for Multiple choice

- **Question:** Which of the following animals is an amphibian?
 - **Choice:** Frog
 - **Choice:** Fish
- Feed in the question along with the choice-1



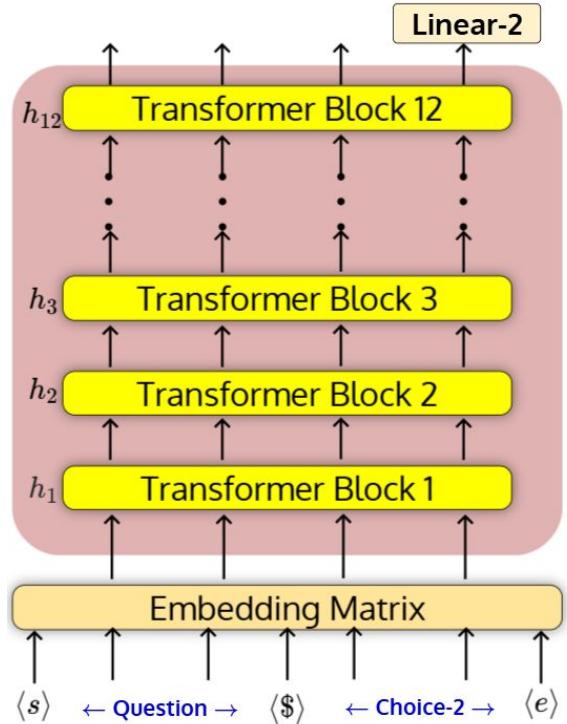
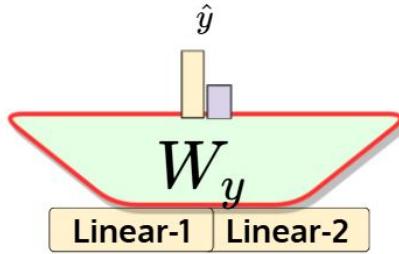
GPT: Fine Tuning for Multiple choice

- **Question:** Which of the following animals is an amphibian?
 - **Choice:** Frog
 - **Choice:** Fish
- Feed in the question along with the choice-2



GPT: Fine Tuning for Multiple choice

- Repeat this for all choices
- Normalize via softmax



GPT: Fine Tuning for Multiple choice

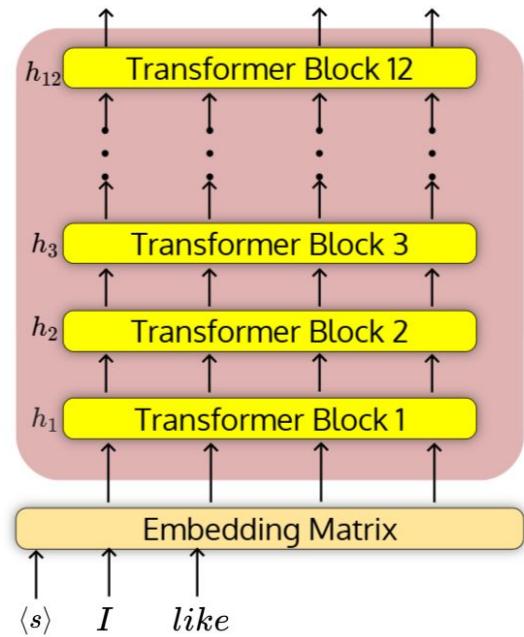
- **Prompt:** I like

$$M = \begin{bmatrix} 0 & 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & 0 & -\infty \end{bmatrix}$$

Feed in the prompt along with the mask and run the model in autoregressive mode.

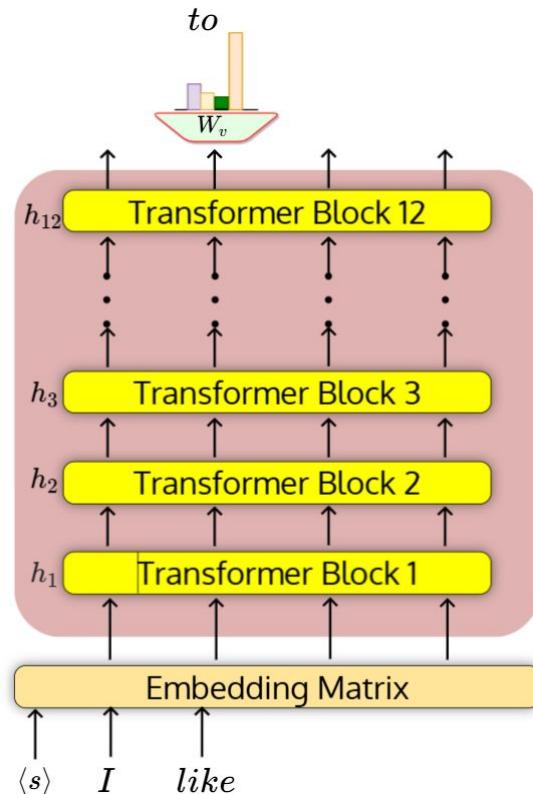
- **Stoping Criteria:**

- Sequence length: 5
- or outputting a token: <e>



GPT: Fine Tuning for Multiple choice

- **Prompt:** I like
- **Output:**
 - I like **to think that**

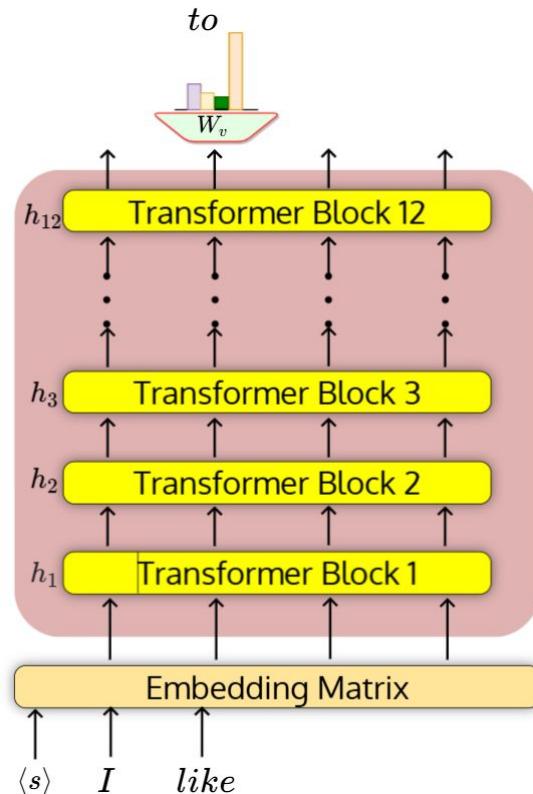


GPT: Fine Tuning for Multiple choice

- **Prompt:** I like
- **Output:**
 - I like **to think that**

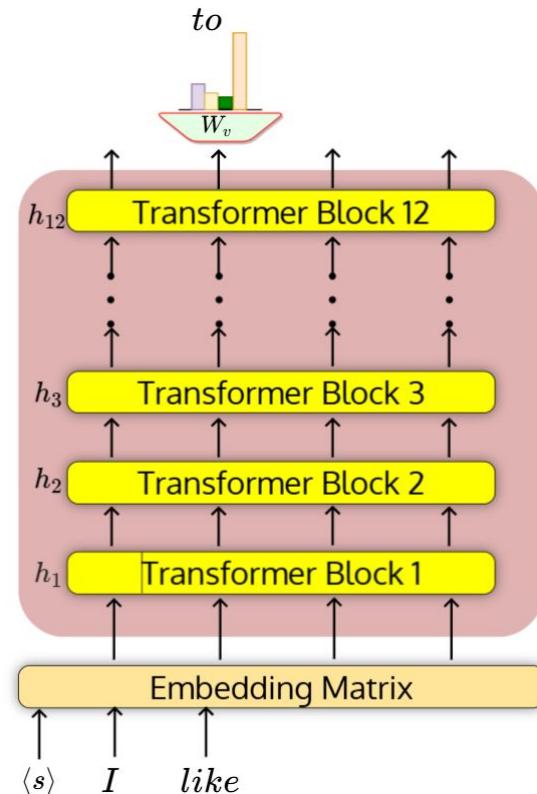
Does it produce the same output sequence for the given prompt?

or Will it be creative ?

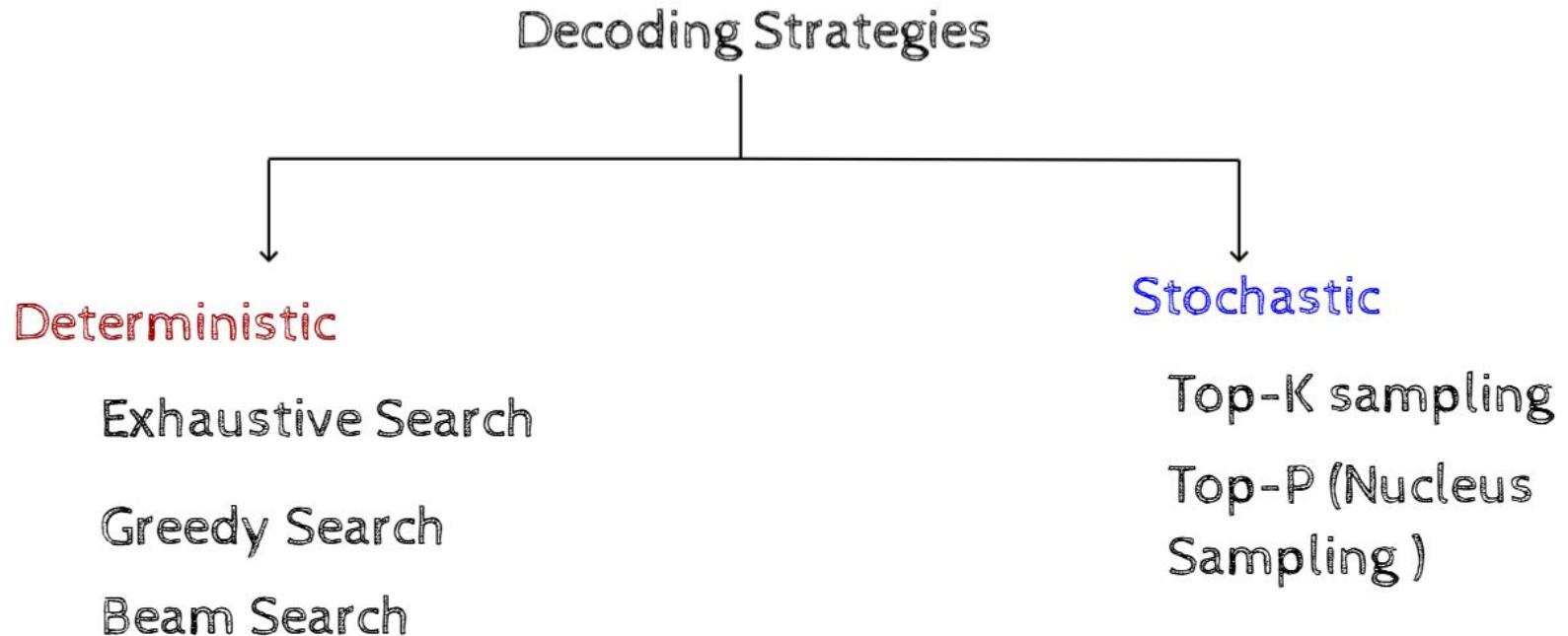


GPT: Fine Tuning for Multiple choice

- Discourage degenerative (that is, repeated or incoherent) texts
 - I like to think that I like to think...
 - I like to think that reference know how to think best selling book
- Encourage it to be Creative in generating a sequence for the same prompt
 - I like to read a book
 - I like to buy a hot beverage
 - I like a person who cares about others



Text Generation Decoding strategy



Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

Ideally, we can generate $|V|^5$ sequences.

Here $|V| = 6$,

So $|6|^5 = 7776$ sequences

Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

Ideally, we can generate $|V|^5$ sequences. In GPT,

Here $|V| = 6$,

$|V| = 40478$,

So $|6|^5 = 7776$ sequences

So, $|40478|^5 = 1.0866639e+23$ sequences

Exhaustive search for generation is practically not possible.

Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

I like cold water

I like cold coffee

coffee like cold coffee

I like I like

coffee coffee coffee coffee

Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:

$$V = \{\text{Cold, Coffee, I, Like, Water, <stop>}\}$$

Generate a sequence of 5 words.

I like cold water <eos>

I like cold coffee <eos>

coffee like cold coffee <eos>

I like I like <eos>

coffee coffee coffee coffee <eos>

for each of the sequence output the probability can be calculated as:

$$P(x_1, x_2, x_3, \dots, x_n) =$$

$$P(x_1).P(x_2/x_1), \dots, P(x_n/x_1, x_2, \dots, x_{n-1})$$

Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:

$$V = \{\text{Cold, Coffee, I, Like, Water, <stop>}\}$$

Generate a sequence of 5 words.

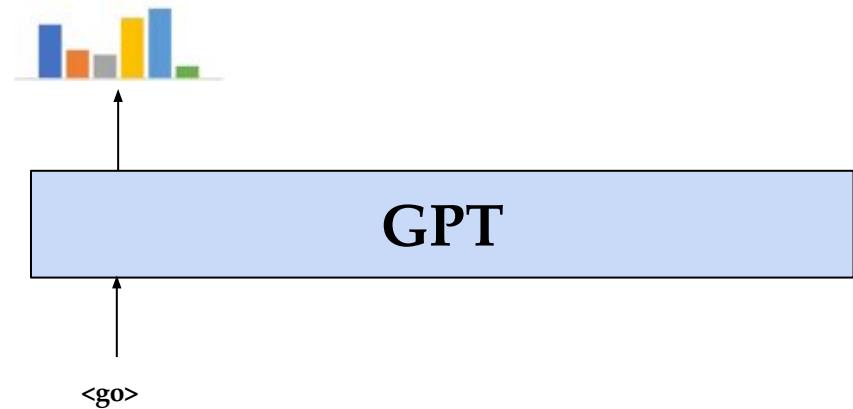
<go> I like cold water <eos>

<go> I like cold coffee <eos>

<go> coffee like cold coffee <eos>

<go> I like I like <eos>

<go> coffee coffee coffee coffee <eos>



Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:

$$V = \{\text{Cold, Coffee, I, Like, Water, <stop>}\}$$

Generate a sequence of 5 words.

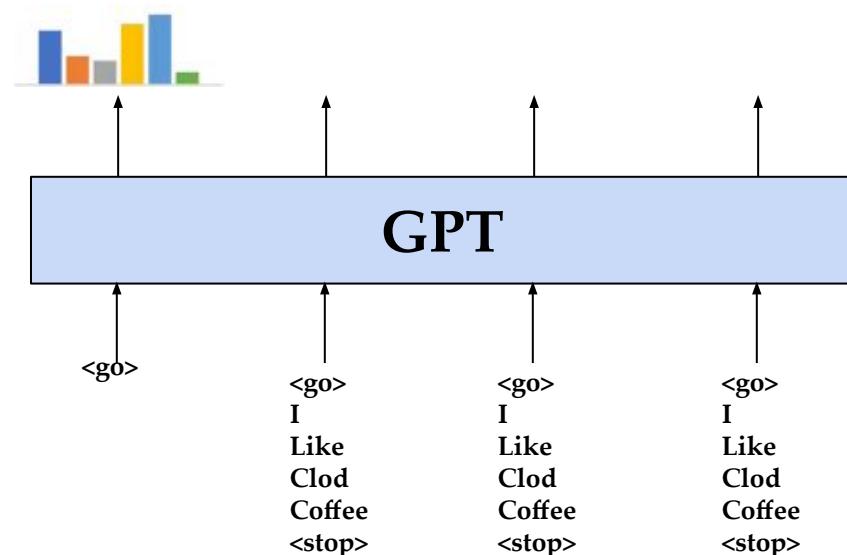
<go> I like cold water <eos>

<go> I like cold coffee <eos>

<go> coffee like cold coffee <eos>

<go> I like I like <eos>

<go> coffee coffee coffee coffee <eos>



Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

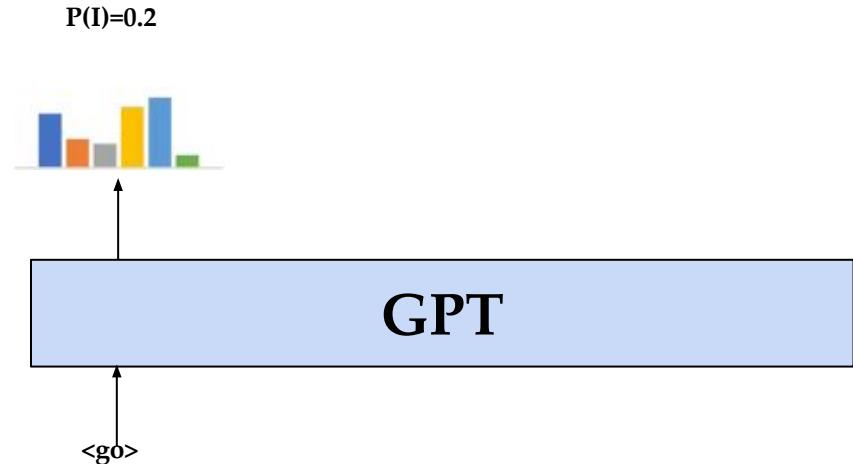
<go> I like cold water <eos>

<go> I like cold coffee <eos>

<go> coffee like cold coffee <eos>

<go> I like I like <eos>

<go> coffee coffee coffee coffee <eos>



Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

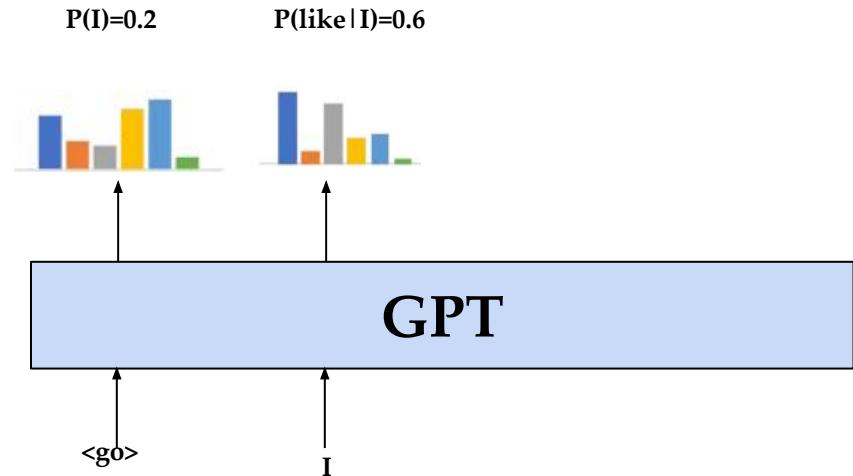
<go> I like cold water <eos>

<go> I like cold coffee <eos>

<go> coffee like cold coffee <eos>

<go> I like I like <eos>

<go> coffee coffee coffee coffee <eos>



Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

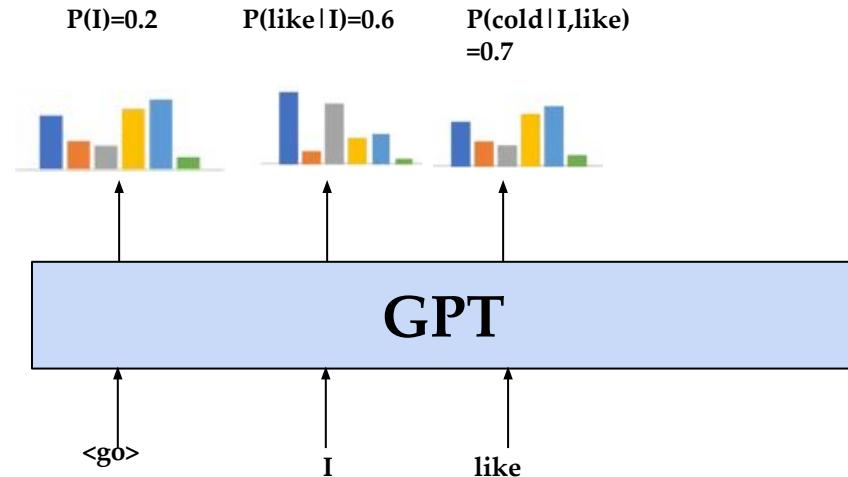
<go> I like cold water <eos>

<go> I like cold coffee <eos>

<go> coffee like cold coffee <eos>

<go> I like I like <eos>

<go> coffee coffee coffee coffee <eos>

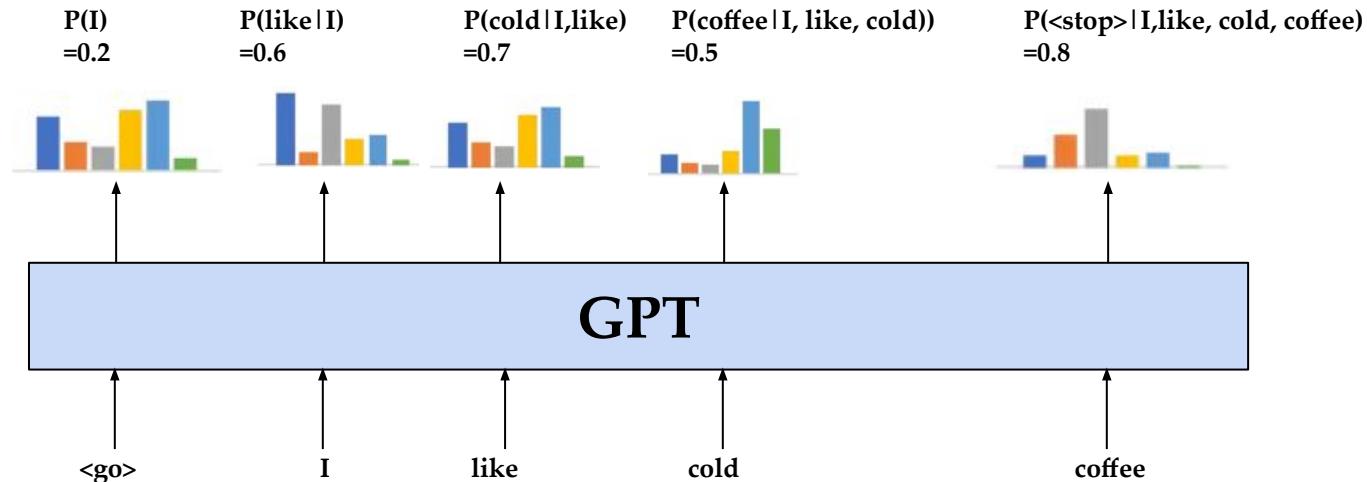


Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words as follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

$$P(\text{I like cold coffee}) = P(\text{I}) * P(\text{like}|\text{I}) * P(\text{cold}|\text{I, like}) * P(\text{coffee}|\text{I, like, cold})$$



Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

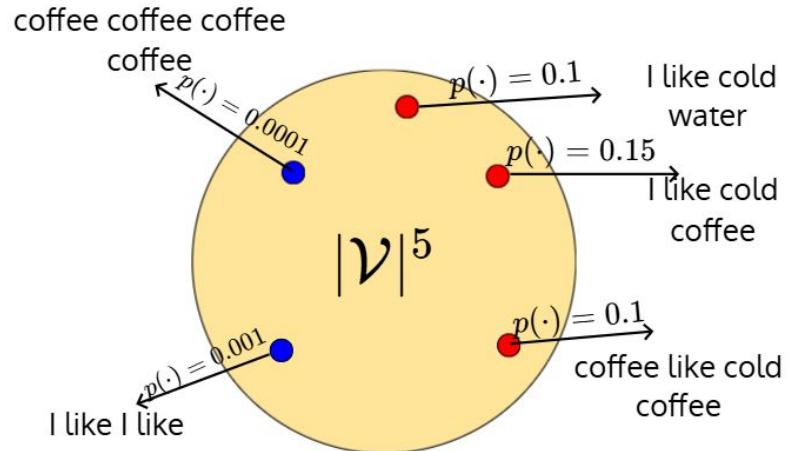
I like cold water

I like cold coffee

coffee like cold coffee

I like I like

coffee coffee coffee coffee



Decoding Strategy: Exhaustive Search

Consider a vocabulary of 5 words s follows:
 $V = \{\text{Cold, Coffee, I, Like, Water, <stop>}\}$

Generate a sequence of 5 words.

I like cold water

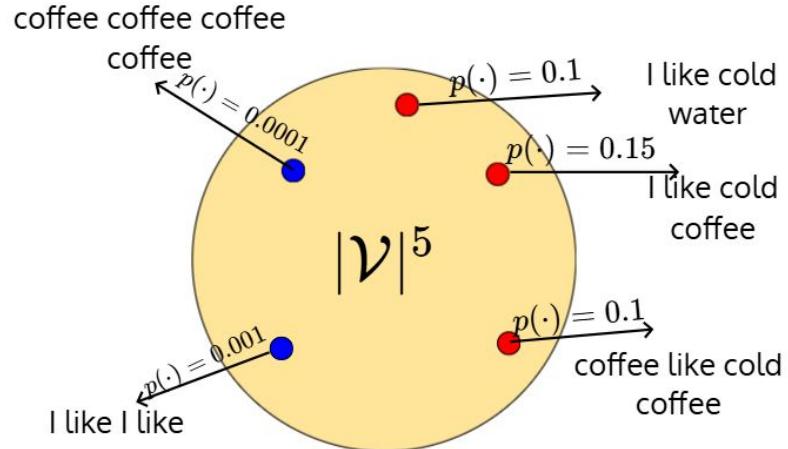
I like cold coffee **Final Output**

coffee like cold coffee

I like I like

coffee coffee coffee coffee

Deterministic strategy:
Output is always same



Decoding Strategy: Exhaustive Search

Consider a sequence of length 2 with the vocabulary of size 3

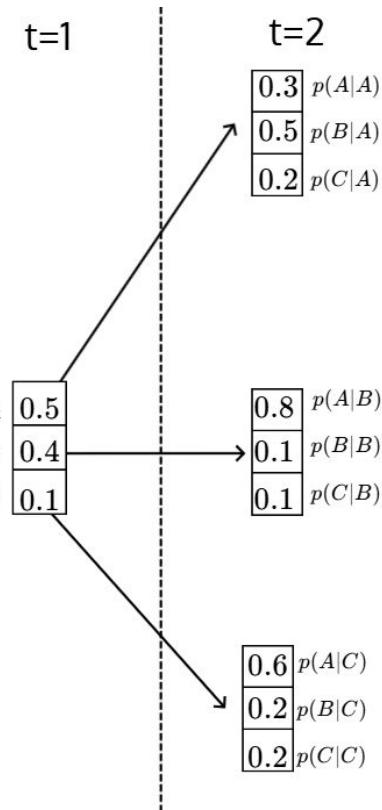
- **At time step-1,**
 - input is <go> and
 - Output is probability for all 3 tokens.

A	0.5
B	0.4
C	0.1

Decoding Strategy: Exhaustive Search

Consider a sequence of length 2 with the vocabulary of size 3

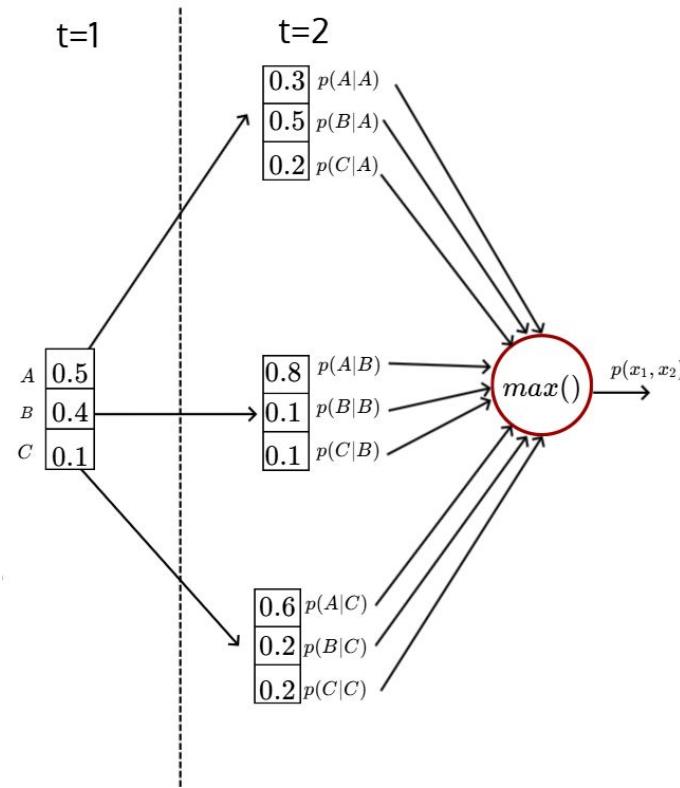
- **At time step-1,**
 - input is <go> and
 - Output is probability for all 3 tokens.
- **At time step-2, GPT runs 3 times**
 - input is A, B, and C and
 - Output is probability for all 3 tokens for A, B, and C.



Decoding Strategy: Exhaustive Search

Consider a sequence of length 2 with the vocabulary of size 3

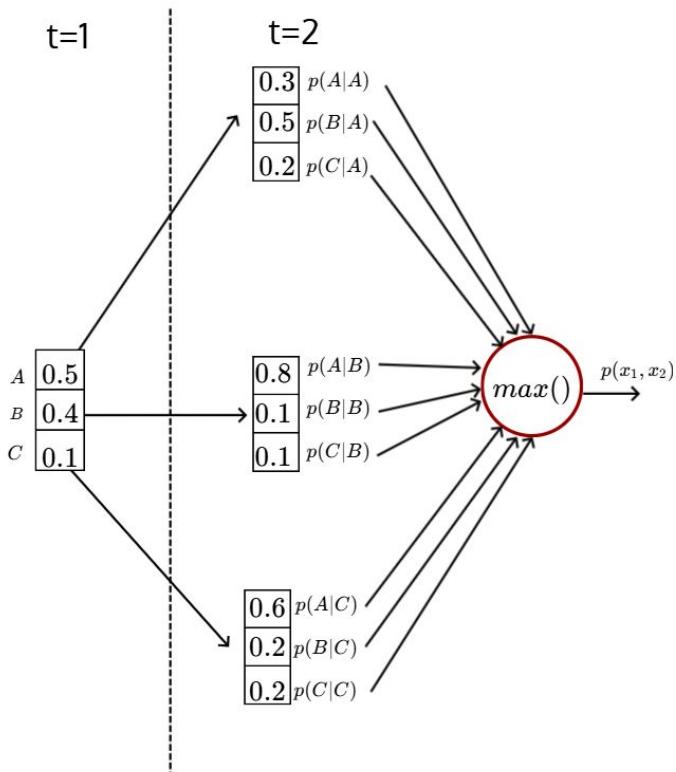
- **At time step-1,**
 - input is <go> and
 - Output is probability for all 3 tokens.
- **At time step-2, GPT runs 3 times**
 - input is A, B, and C and
 - Output is probability for all 3 tokens for A, B, and C.
- **At time step-3, GPT runs 9 times**
 - Output is probability for all 3 tokens for A, B, and C.



Decoding Strategy: Exhaustive Search

Consider a sequence of length 2 with the vocabulary of size 3

- **At time step-1,**
 - input is <go> and
 - Output is probability for all 3 tokens.
- **At time step-2, GPT runs 3 times**
 - input is A, B, and C and
 - Output is probability for all 3 tokens for A, B, and C.
- **At time step-3, GPT runs 9 times**
 - Output is probability for all 3 tokens for A, B, and C.



For $|V| = 40478$, at each time step, we need to run the decoder 40000 times in parallel

Decoding Strategy: Greedy Search

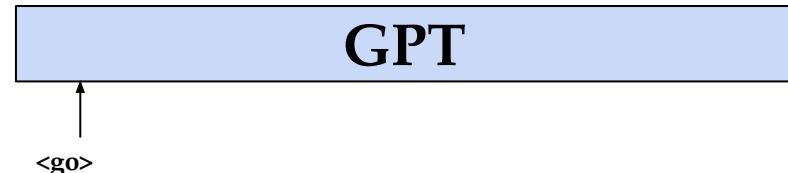
Consider a vocabulary of 5 words as follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

	Time Step				
	1	2	3	4	5
Cold					
<stop>					
coffee					
I					
like					
water					

At each time step, we always output the token with the highest probability (Greedy).



Decoding Strategy: Greedy Search

Consider a vocabulary of 5 words as follows:

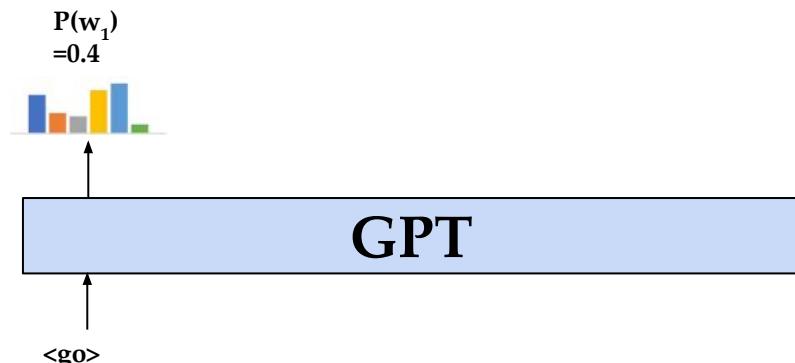
$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

	Time Step				
	1	2	3	4	5
Cold	0.1				
<stop>	0.15				
coffee	0.25				
I	0.4				
like	0.05				
water	0.05				

At each time step, we always output the token with the highest probability (Greedy).

$$P(w_1 = I) = 0.4$$



Decoding Strategy: Greedy Search

Consider a vocabulary of 5 words as follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

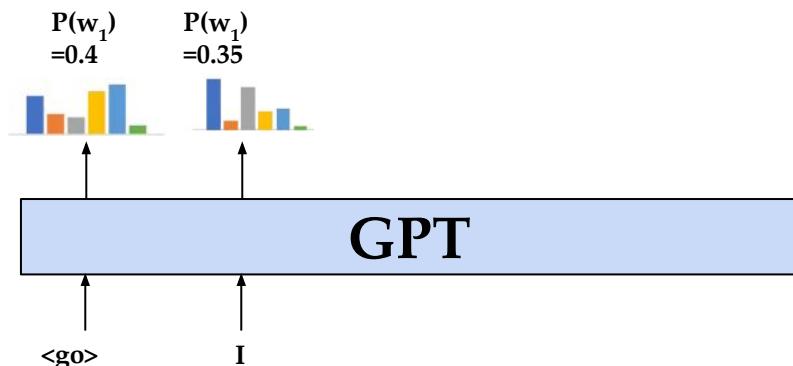
Generate a sequence of 5 words.

	Time Step				
	1	2	3	4	5
Cold	0.1	0.1			
<stop>	0.15	0.15			
coffee	0.25	0.25			
I	0.4	0.05			
like	0.05	0.35			
water	0.05	0.1			

At each time step, we always output the token with the highest probability (Greedy).

$$P(w_1 = I) = 0.4$$

$$P(w_2 = \text{like} | w_1 = I) = 0.35$$



Decoding Strategy: Greedy Search

Consider a vocabulary of 5 words as follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

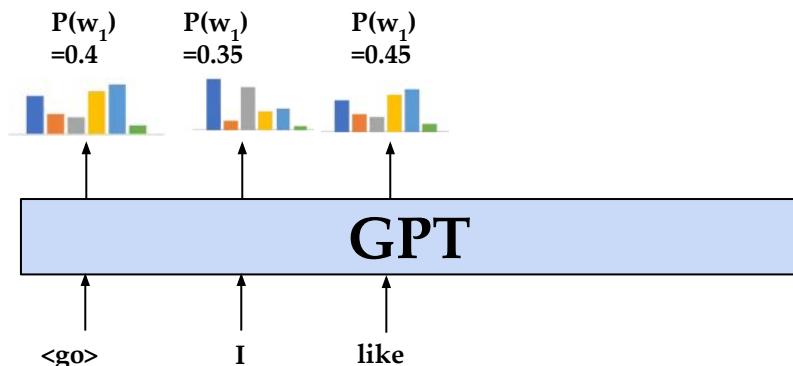
	Time Step				
	1	2	3	4	5
Cold	0.1	0.1	0.45		
<stop>	0.15	0.15	0.05		
coffee	0.25	0.25	0.1		
I	0.4	0.05	0.05		
like	0.05	0.35	0.15		
water	0.05	0.1	0.2		

At each time step, we always output the token with the highest probability (Greedy).

$$P(w_1 = \text{I}) = 0.4$$

$$P(w_2 = \text{like} | w_1 = \text{I}) = 0.35$$

$$P(w_3 = \text{cold} | w_1 = \text{I}, w_2 = \text{like}) = 0.45$$



Decoding Strategy: Greedy Search

Consider a vocabulary of 5 words as follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \text{<stop>}\}$$

Generate a sequence of 5 words.

	Time Step				
	1	2	3	4	5
Cold	0.1	0.1	0.45	0.15	
<stop>	0.15	0.15	0.05	0.3	
coffee	0.25	0.25	0.1	0.35	
I	0.4	0.05	0.05	0.01	
like	0.05	0.35	0.15	0.09	
water	0.05	0.1	0.2	0.1	

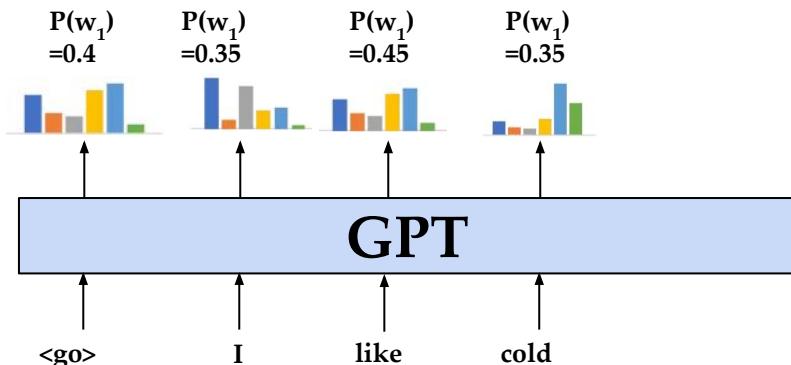
At each time step, we always output the token with the highest probability (Greedy).

$$P(w_1 = \text{I}) = 0.4$$

$$P(w_2 = \text{like} | w_1 = \text{I}) = 0.35$$

$$P(w_3 = \text{cold} | w_1 = \text{I}, w_2 = \text{like}) = 0.45$$

$$P(w_4 = \text{coffee} | w_1 = \text{I}, w_2 = \text{like}, w_3 = \text{cold}) = 0.35$$



Decoding Strategy: Greedy Search

Consider a vocabulary of 5 words as follows:

$$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$$

Generate a sequence of 5 words.

	Time Step				
	1	2	3	4	5
Cold	0.1	0.1	0.45	0.15	0.1
<stop>	0.15	0.15	0.05	0.3	0.5
coffee	0.25	0.25	0.1	0.35	0.2
I	0.4	0.05	0.05	0.01	0.1
like	0.05	0.35	0.15	0.09	0.05
water	0.05	0.1	0.2	0.1	0.05

At each time step, we always output the token with the highest probability (Greedy).

$$P(w_1=I)=0.4$$

$$P(w_2=\text{like} | w_1=I)=0.35$$

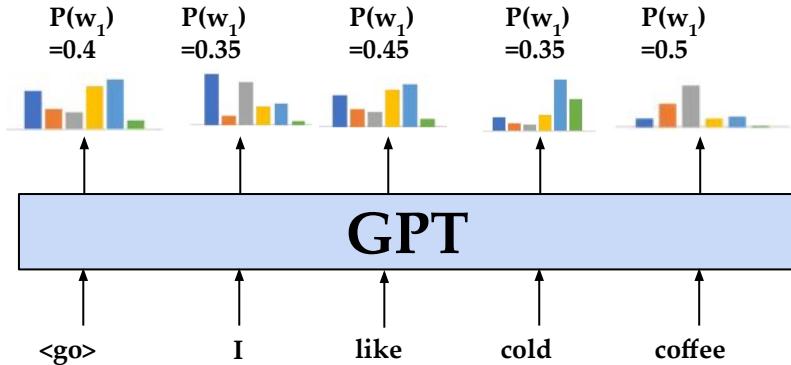
$$P(w_3=\text{cold} | w_1=I, w_2=\text{like})=0.45$$

$$P(w_4=\text{coffee} | w_1=I, w_2=\text{like}, w_3=\text{cold})=0.35$$

$$P(w_5=\text{stop} | w_1=I, w_2=\text{like}, w_3=\text{cold}, w_4=\text{coffee})=0.5$$

$$P(\text{I like cold coffee } \langle\text{stop}\rangle) =$$

$$0.4 * 0.35 * 0.45 * 0.35 * 0.5 = 0.011$$



Decoding Strategy: Greedy Search

Consider a vocabulary of 5 words as follows:

$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$

Generate a sequence of 5 words.

	Time Step				
	1	2	3	4	5
Cold	0.1	0.1	0.45	0.15	0.1
<stop>	0.15	0.15	0.05	0.3	0.5
coffee	0.25	0.25	0.1	0.35	0.2
I	0.4	0.05	0.05	0.01	0.1
like	0.05	0.35	0.15	0.09	0.05
water	0.05	0.1	0.2	0.1	0.05

At each time step, we always output the token with the highest probability (Greedy).

$$P(w_1=I)=0.4$$

$$P(w_2=\text{like} | w_1=I)=0.35$$

$$P(w_3=\text{cold} | w_1=I, w_2=\text{like})=0.45$$

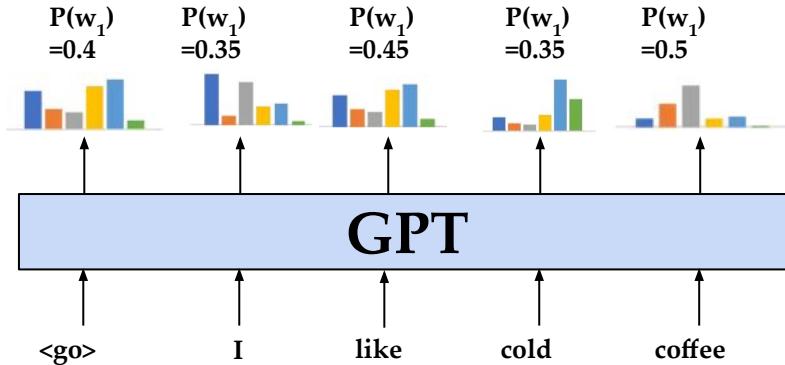
$$P(w_4=\text{coffee} | w_1=I, w_2=\text{like}, w_3=\text{cold})=0.35$$

$$P(w_5=\text{stop} | w_1=I, w_2=\text{like}, w_3=\text{cold}, w_4=\text{coffee})=0.5$$

Deterministic strategy:
Output is always same

$$P(I \text{ like cold coffee } \langle\text{stop}\rangle) =$$

$$0.4 * 0.35 * 0.45 * 0.35 * 0.5 = 0.011$$



Decoding Strategy: Greedy Search Limitations

- Is this the **most likely** sequence?
- What if we want to get a variety of sequences of the same length?
 - Not possible
- If the starting token is the word "I", then it will always end up producing the same sequence:
 - I like cold coffee

	Time Step				
	1	2	3	4	5
Cold	0.1	0.1	0.45	0.15	0.1
<stop>	0.15	0.15	0.05	0.3	0.5
coffee	0.25	0.25	0.1	0.35	0.2
I	0.4	0.05	0.05	0.01	0.1
like	0.05	0.35	0.15	0.09	0.05
water	0.05	0.1	0.2	0.1	0.05
	I	like	cold	coffee	<stop>

Decoding Strategy: Greedy Search Limitations

- What if we picked the second most probable token in the first time step?
 - we would have ended up with a different sequence.

	Time Step				
	1	2	3	4	5
Cold	0.1				
<stop>	0.15				
coffee	0.25				
I	0.4				
like	0.05				
water	0.05				
	coffee				

Decoding Strategy: Greedy Search

Consider a vocabulary of 5 words as follows:

$V = \{\text{Cold}, \text{Coffee}, \text{I}, \text{Like}, \text{Water}, \langle\text{stop}\rangle\}$

Generate a sequence of 5 words.

	Time Step				
	1	2	3	4	5
Cold	0.1	0.15	0.65	0.04	0.1
<stop>	0.15	0.1	0.05	0.01	0.5
coffee	0.25	0.05	0.05	0.1	0.2
I	0.4	0.05	0.05	0.03	0.1
like	0.05	0.55	0.1	0.02	0.05
water	0.05	0.1	0.1	0.8	0.05

At each time step, we always output the token with the highest probability (Greedy).

$$P(w_1 = \text{coffee}) = 0.25$$

$$P(w_2 = \text{like} | w_1 = \text{coffee}) = 0.55$$

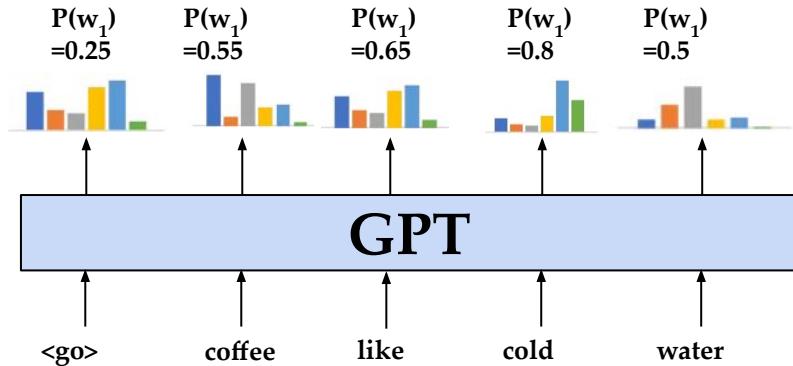
$$P(w_3 = \text{cold} | w_1 = \text{coffee}, w_2 = \text{like}) = 0.65$$

$$P(w_4 = \text{water} | w_1 = \text{coffee}, w_2 = \text{like}, w_3 = \text{cold}) = 0.8$$

$$P(w_5 = \text{stop} | w_1 = \text{coffee}, w_2 = \text{like}, w_3 = \text{cold}, w_4 = \text{water}) = 0.5$$

$$P(\text{coffee like cold water } \langle\text{stop}\rangle) =$$

$$0.25 * 0.55 * 0.65 * 0.8 * 0.5 = 0.035$$

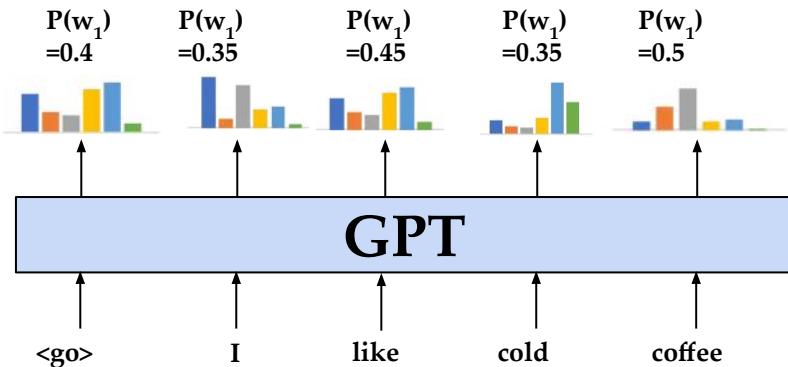


Decoding Strategy: Greedy Search Limitations

- We could output the second sequence instead of the one generated by greedy search.
- That means greedy search does not guarantee to give a sequence with maximum probability.
- Why not follow this idea at every time step?

Decoding Strategy: Beam Search

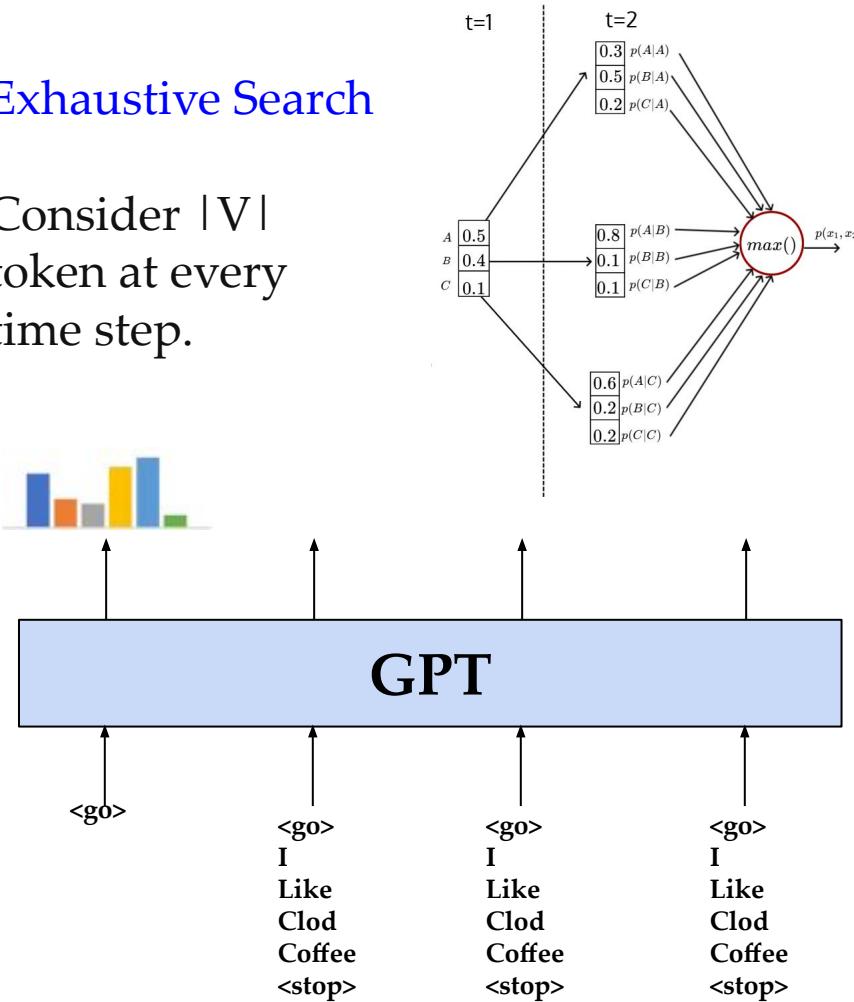
Greedy Search



Selecting only one token for the next time stamp.

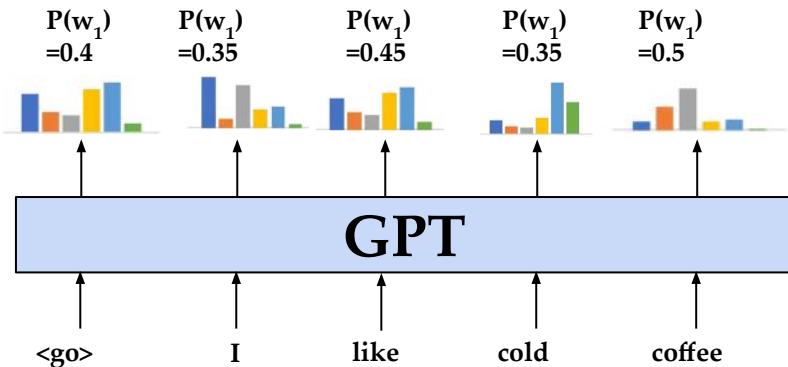
Exhaustive Search

Consider $|V|$ token at every time step.



Decoding Strategy: Beam Search

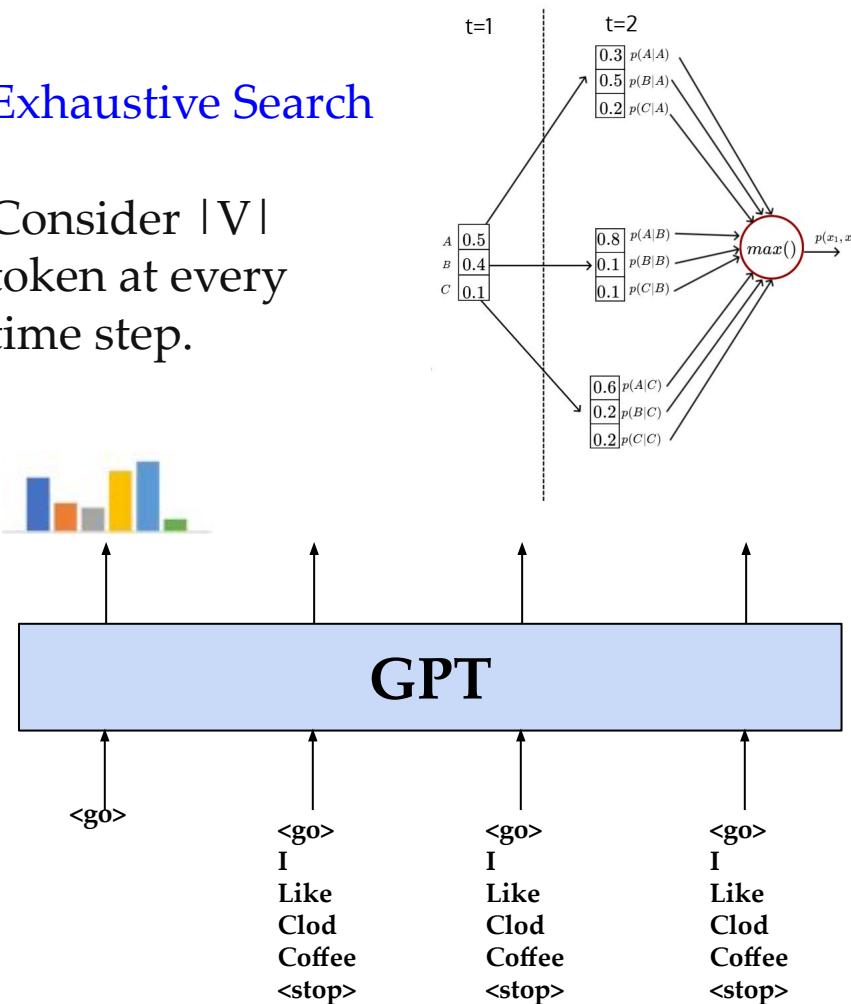
Greedy Search



Selecting only one token for the next time stamp.

Exhaustive Search

Consider $|V|$ token at every time step.



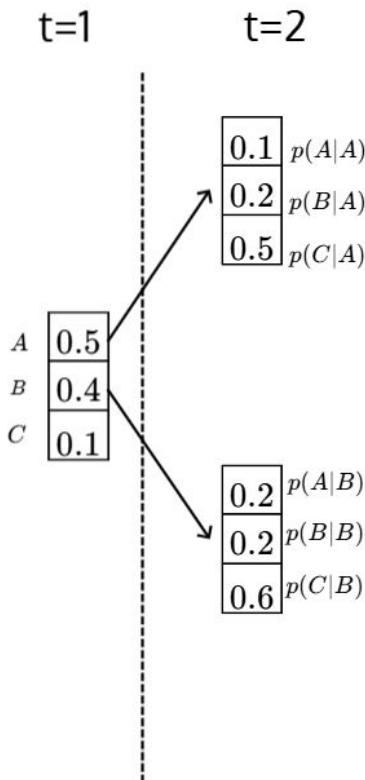
Decoding Strategy: Beam Search

t=1

A	0.5
B	0.4
C	0.1

Instead of considering probability for all the tokens at every time step (as in exhaustive search), consider only **top-k tokens**.

Decoding Strategy: Beam Search



- Instead of considering probability for all the tokens at every time step (as in exhaustive search), consider only **top-k tokens**.
- Suppose ($k = 2$)
- Now we have to choose the tokens that maximize the probability of the sequence.
- It requires $k \times |V|$ computations at each time step.

$$p(A)p(A|A) = 0.5 \times 0.1 = 0.05$$

$$p(A)p(B|A) = 0.5 \times 0.2 = 0.01$$

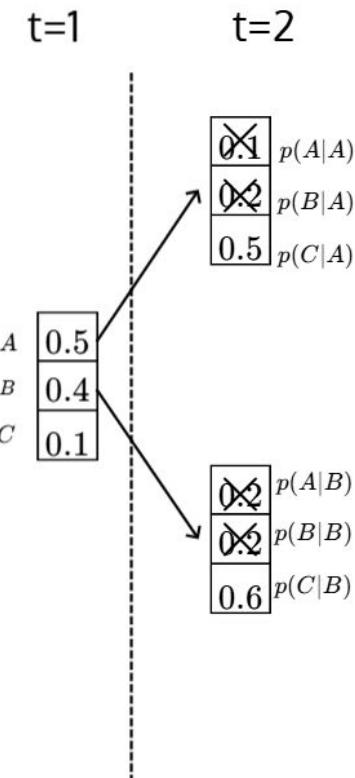
$$p(A)p(C|A) = 0.5 \times 0.5 = 0.25$$

$$p(B)p(A|B) = 0.4 \times 0.2 = 0.08$$

$$p(B)p(B|B) = 0.4 \times 0.2 = 0.08$$

$$p(B)p(C|B) = 0.4 \times 0.6 = 0.24$$

Decoding Strategy: Beam Search



- Instead of considering probability for all the tokens at every time step (as in exhaustive search), consider only **top-k tokens**.
- Suppose ($k = 2$)
- Now we have to choose the tokens that maximize the probability of the sequence.
- It requires $k \times |V|$ computations at each time step.

$$p(A)p(A|A) = 0.5 \times 0.1 = 0.05$$

$$p(A)p(B|A) = 0.5 \times 0.2 = 0.1$$

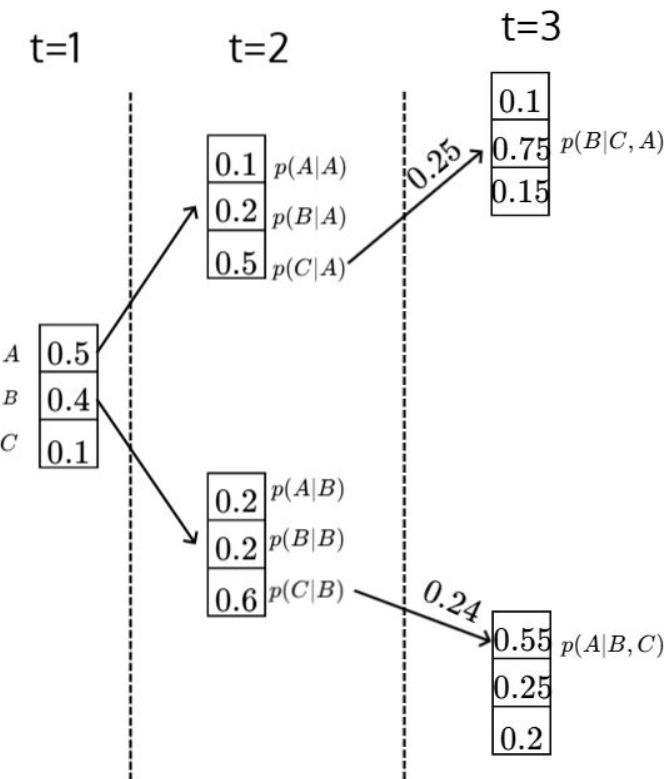
$$p(A)p(C|A) = 0.5 \times 0.5 = 0.25$$

$$p(B)p(A|B) = 0.4 \times 0.2 = 0.08$$

$$p(B)p(B|B) = 0.4 \times 0.2 = 0.08$$

$$p(B)p(C|B) = 0.4 \times 0.6 = 0.24$$

Decoding Strategy: Beam Search

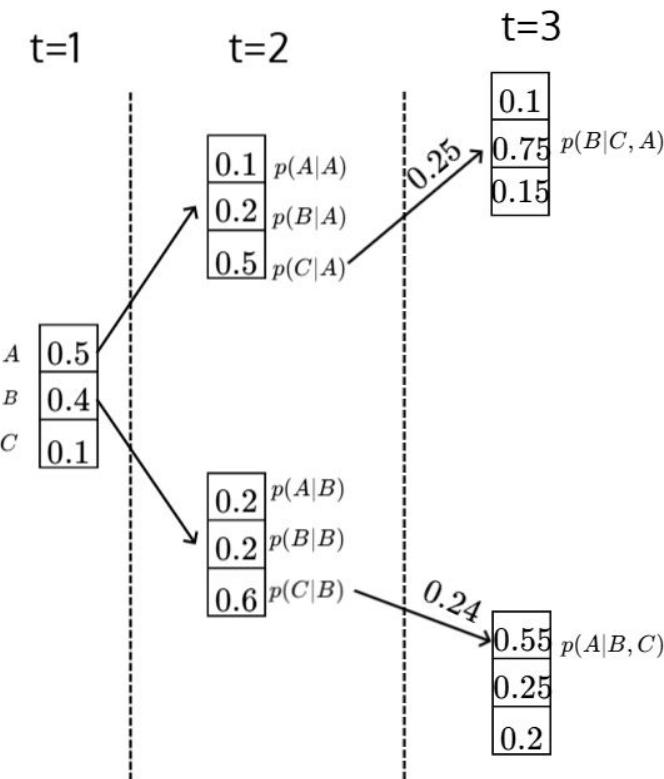


- Instead of considering probability for all the tokens at every time step (as in exhaustive search), consider only **top-k tokens**.
- Suppose ($k = 2$)
- Following a similar calculation, we end up choosing:

$$P(A, C, B) = 0.18$$

$$P(C, B, A) = 0.13$$

Decoding Strategy: Beam Search



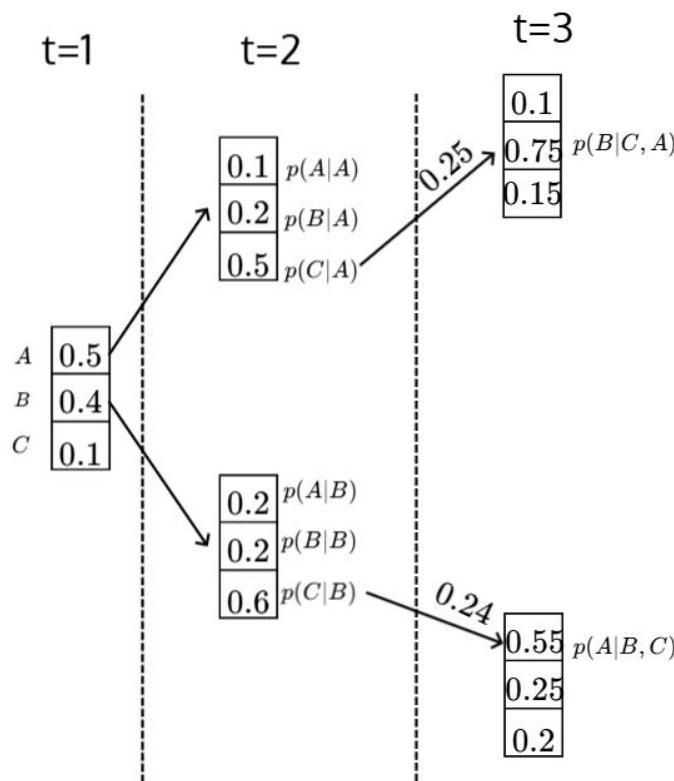
- Instead of considering probability for all the tokens at every time step (as in exhaustive search), consider only **top-k tokens**.
- Suppose ($k = 2$)
- Following a similar calculation, we end up choosing:

$$P(A, C, B) = 0.18$$

$$P(C, B, A) = 0.13$$

- Now we will have k sequences at the end of time step T and output the sequence which **has the highest probability**.
- Here, the final sequence would be the sequence A, C, B.

Decoding Strategy: Beam Search



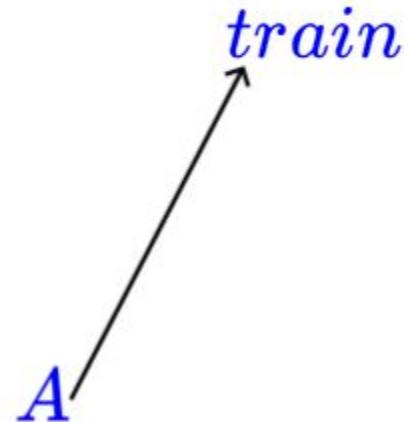
- Parameter k is called beam size.
- It is an approximation to exhaustive search.
- If k= 1, it becomes greedy.
- If k= |V|, it becomes exhaustive.
- At every time step 2 (for k=2) * |V| probabilities are generated, and we pick the top two probabilities.

Decoding Strategy: Beam Search

A

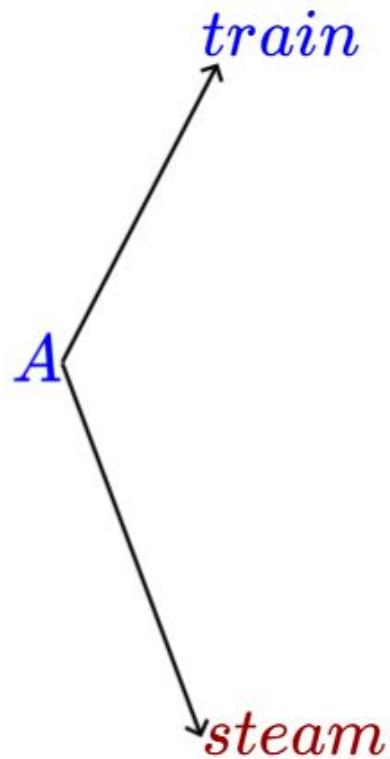
Decoding Strategy: Beam Search

train

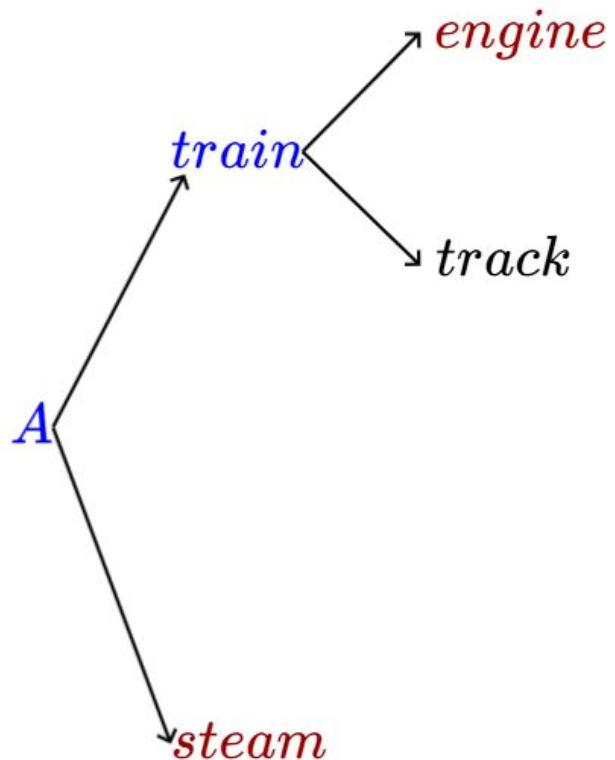


A

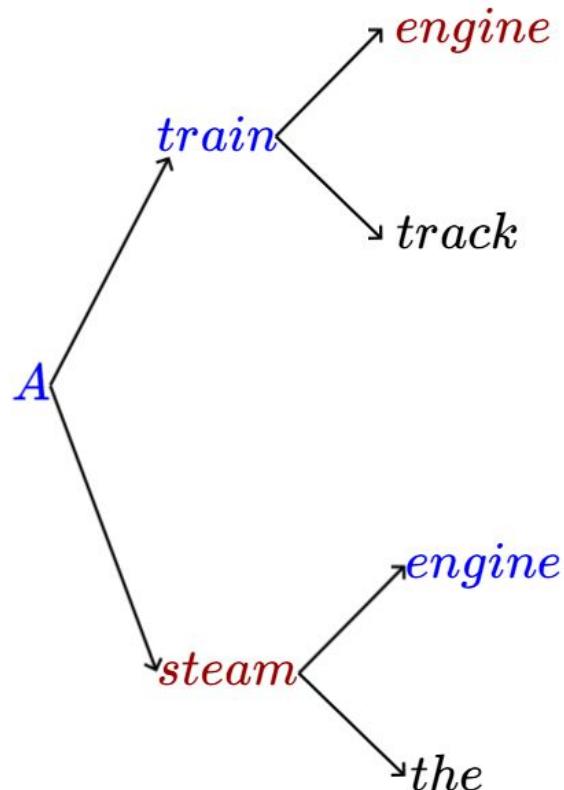
Decoding Strategy: Beam Search



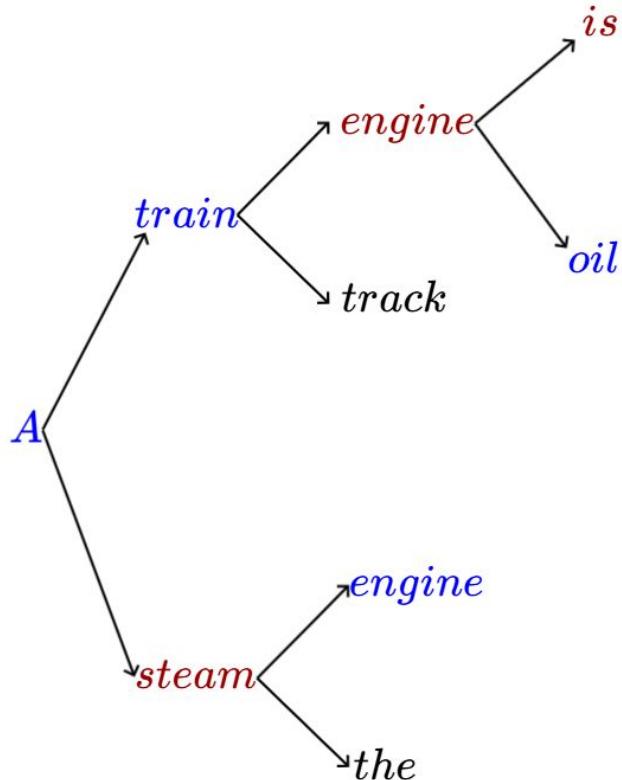
Decoding Strategy: Beam Search



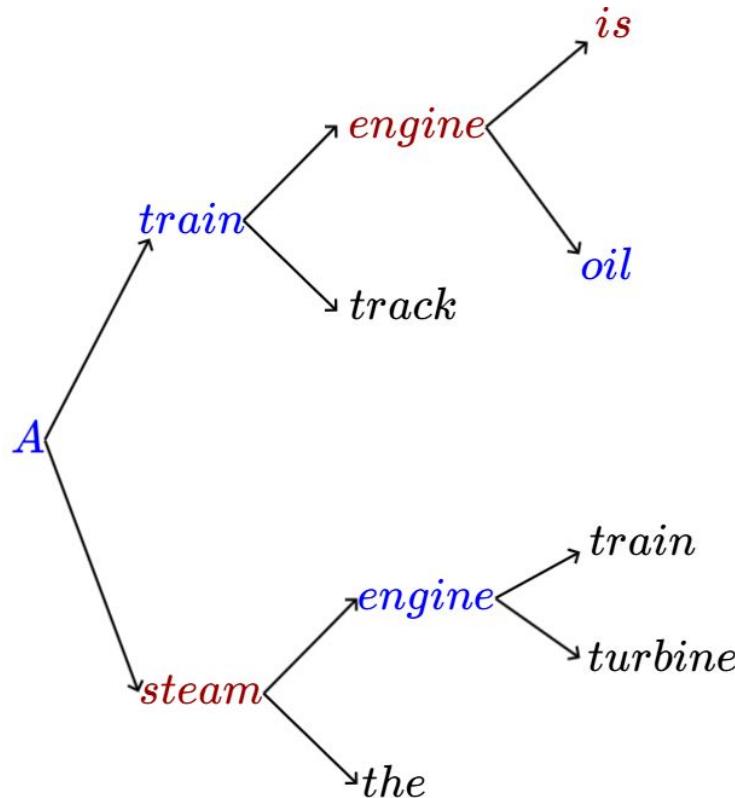
Decoding Strategy: Beam Search



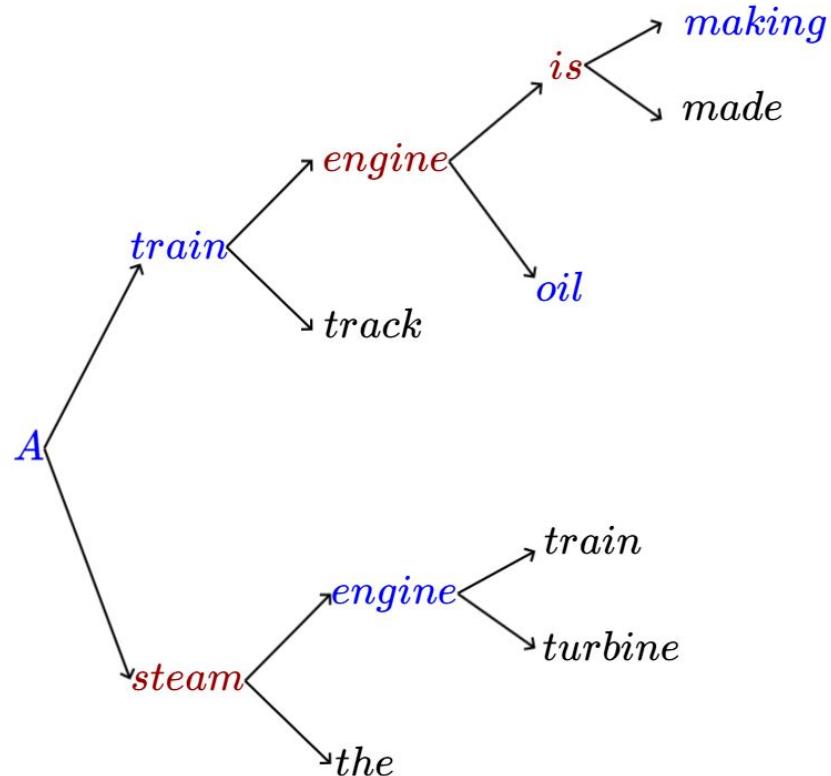
Decoding Strategy: Beam Search



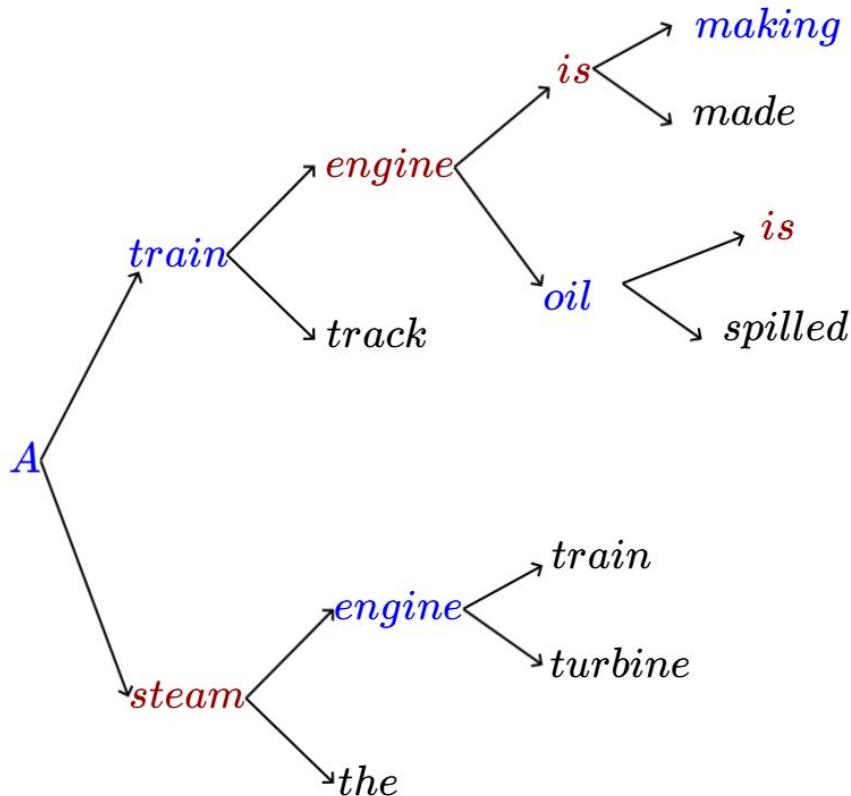
Decoding Strategy: Beam Search



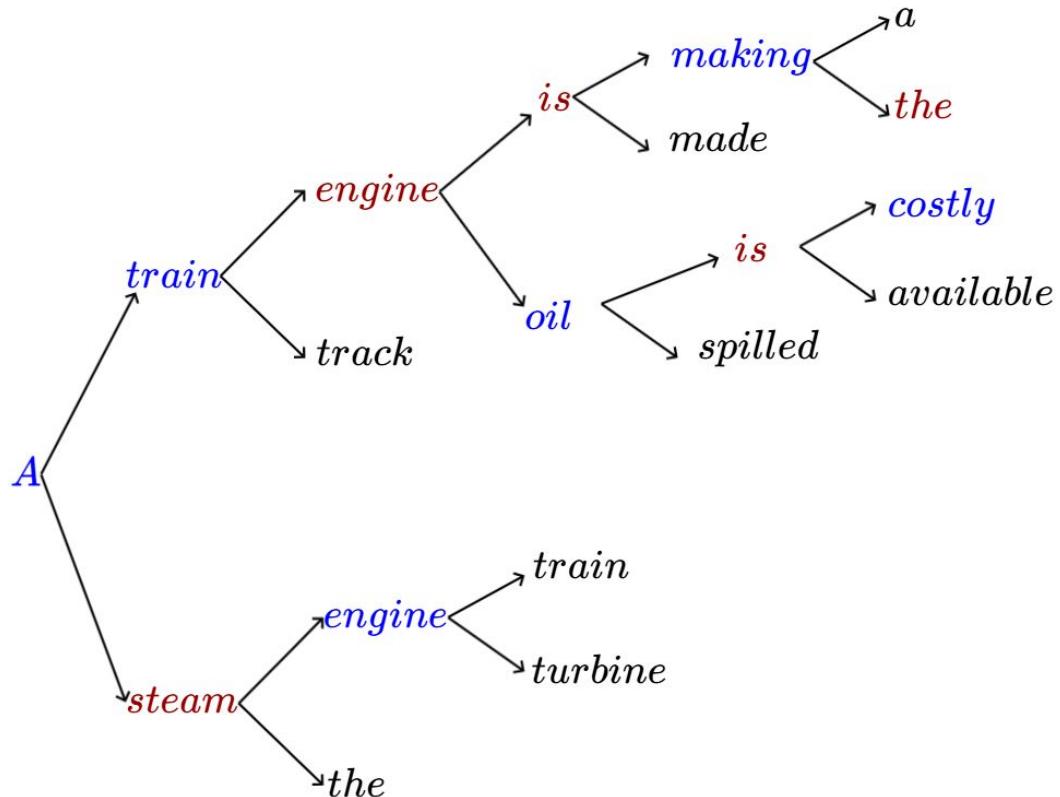
Decoding Strategy: Beam Search



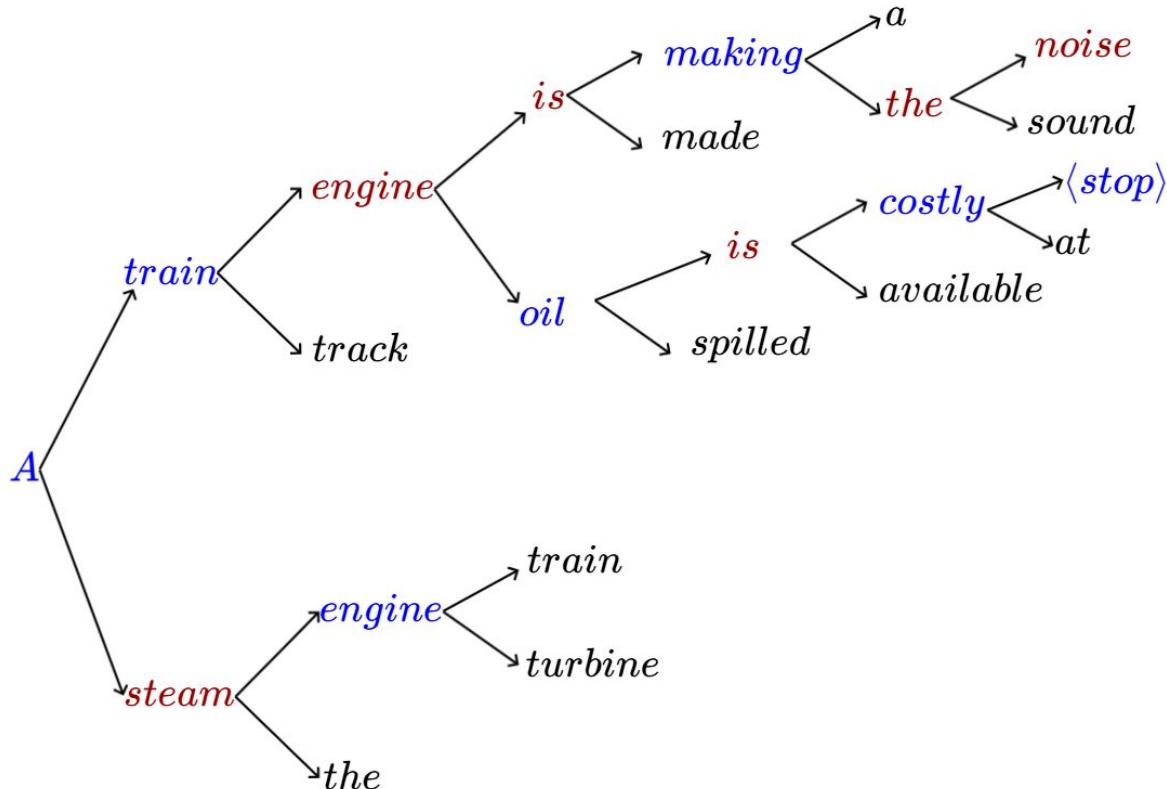
Decoding Strategy: Beam Search



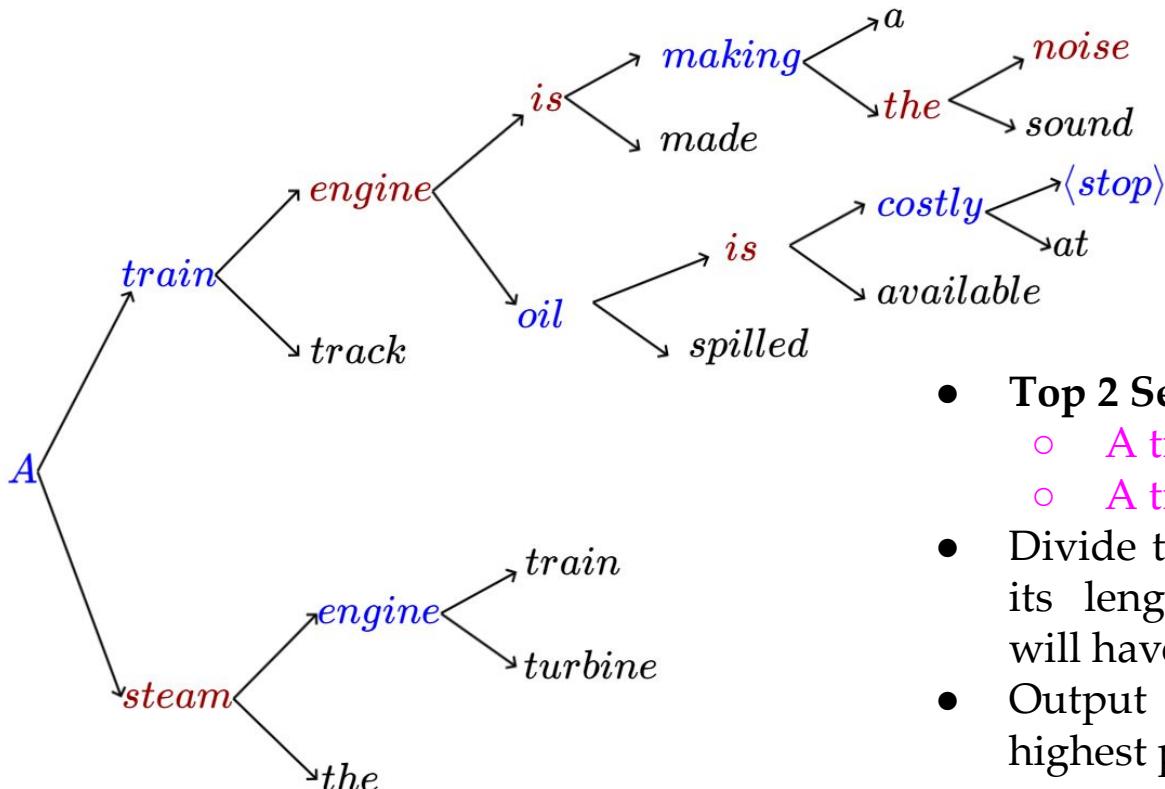
Decoding Strategy: Beam Search



Decoding Strategy: Beam Search



Decoding Strategy: Beam Search



- **Top 2 Sequences:**
 - A train engine is making the noise
 - A train engine oil is costly
- Divide the probability of the sequence by its length (otherwise longer sequences will have lower probability)
- Output the sequence which has the highest probability

Deterministic Decoding Strategy: Summary

- Both the greedy search and the beam search are prone to be degenerative
- Latency for greedy search is lower than beam search
- Neither greedy search nor beam search can result in creative outputs
- Note however that the beam search strategy is highly suitable for tasks like translation and summarization
- We are surprised when something is creative!

Surprise = uncertainty

Stochastic Strategy: Top-k sampling

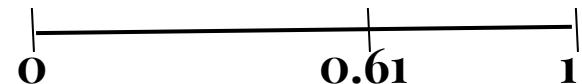
	Time Step				
	1	2	3	4	5
Cold	0.1	0.1	0.65	0.04	0.1
<stop>	0.15	0.15	0.05	0.01	0.5
coffee	0.25	0.05	0.05	0.1	0.2
I	0.4	0.05	0.05	0.03	0.1
like	0.05	0.55	0.1	0.02	0.05
water	0.05	0.1	0.1	0.8	0.05

- At every time step, consider top-k tokens from the probability distribution.
- **Sample** a token from the top-k tokens!
Say, k=2
 - Let's generate a sequence using Top-K sampling
 - The probability of top-k tokens will be normalized relatively,
 $P(I)=0.61, P(coffee)=0.39$ before sampling a token.

Stochastic Strategy: Top-k sampling

	Time Step				
	1	2	3	4	5
Cold	0.1	0.1	0.65	0.04	0.1
<stop>	0.15	0.15	0.05	0.01	0.5
coffee	0.25	0.05	0.05	0.1	0.2
I	0.4	0.05	0.05	0.03	0.1
like	0.05	0.55	0.1	0.02	0.05
water	0.05	0.1	0.1	0.8	0.05
	I				

- The probability of top-k tokens will be normalized relatively,
 $P(I) = 0.4/(0.25+0.4) = 0.61$
 $P(coffee) = 0.25/(0.25+0.4) = 0.39$ before sampling a token.



- We generate a random number:
 $\text{rand}(0,1) = 0.58$
The next word will be I
(any number between 0 to 0.6)

Stochastic Strategy: Top-k sampling

	Time Step				
	1	2	3	4	5
Cold	0.1	0.1	0.65	0.04	0.1
<stop>	0.15	0.15	0.05	0.01	0.5
coffee	0.25	0.05	0.05	0.1	0.2
I	0.4	0.05	0.05	0.03	0.1
like	0.05	0.55	0.1	0.02	0.05
water	0.05	0.1	0.1	0.8	0.05
Coffee					

- The probability of top-k tokens will be normalized relatively,
 $P(I) = 0.4/(0.25+0.4) = 0.61$
 $P(coffee) = 0.25/(0.25+0.4) = 0.39$ before sampling a token.

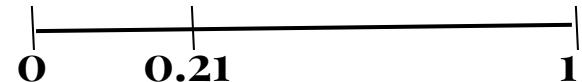


- We generate a random number:
 $\text{rand}(0,1) = 0.7$
The word will be coffee
(any number between 0.61 to 1)

Stochastic Strategy: Top-k sampling

	Time Step				
	1	2	3	4	5
Cold	0.1	0.1	0.65	0.04	0.1
<stop>	0.15	0.15	0.05	0.01	0.5
coffee	0.25	0.05	0.05	0.1	0.2
I	0.4	0.05	0.05	0.03	0.1
like	0.05	0.55	0.1	0.02	0.05
water	0.05	0.1	0.1	0.8	0.05
	I	<stop>			

- Suppose at t=1, we get word I.
- At t=2, the probability of top-k tokens will be normalized relatively,
 $P(\text{stop}) = 0.15/(0.15+0.55) = 0.21$
 $P(\text{like}) = 0.55/(0.15+0.55) = 0.79$ before sampling a token.



- We generate a random number:
 $\text{rand}(0,1) = 0.18$
The next word will be <Stop>
(any number between 0 to 0.21)

I <Stop>

Stochastic Strategy: Top-k sampling

	Time Step				
	1	2	3	4	5
Cold	0.1	0.1	0.65	0.04	0.1
<stop>	0.15	0.15	0.05	0.01	0.5
coffee	0.25	0.05	0.05	0.1	0.2
I	0.4	0.05	0.05	0.03	0.1
like	0.05	0.55	0.1	0.02	0.05
water	0.05	0.1	0.1	0.8	0.05
	Coffee	like	cold	coffee	<stop>

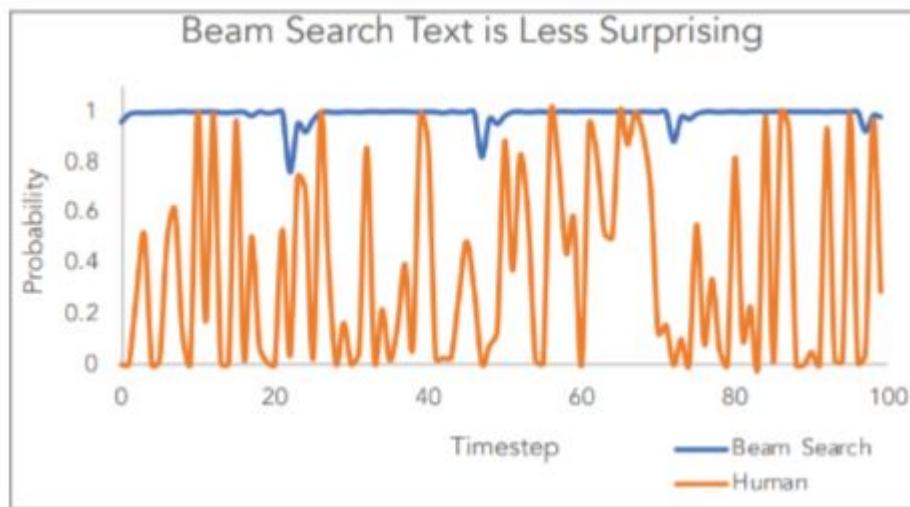
- We generate a random number:
 $\text{rand}(0,1) = ?$
The next word will be

Coffee like cold coffee <stop>

Stochastic Strategy: Top-k sampling

Surprise is an outcome of being random

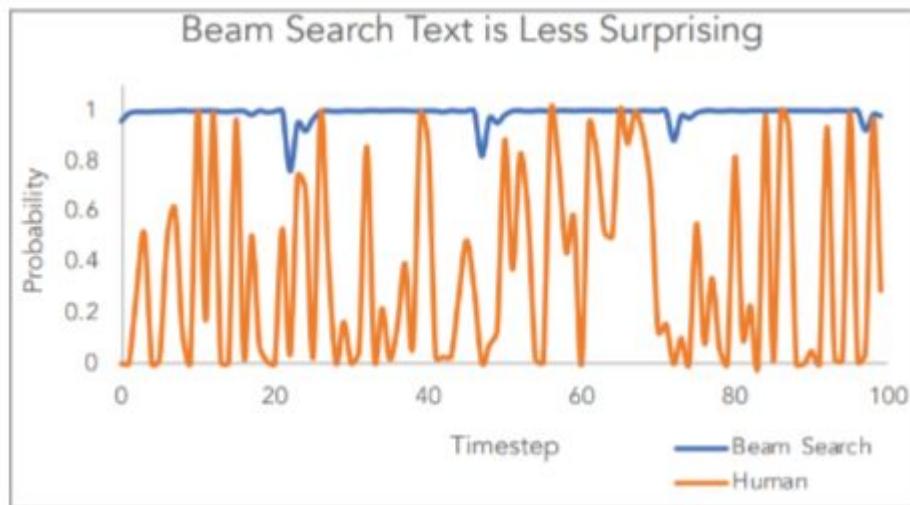
How does beam search compare with human prediction at every time step?



Stochastic Strategy: Top-k sampling

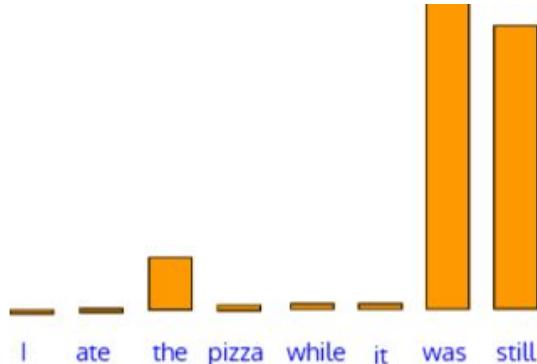
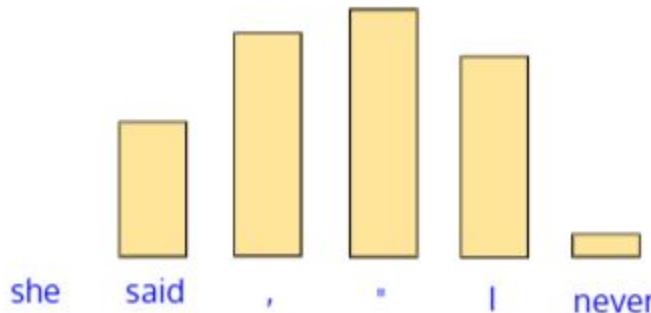
Surprise is an outcome of being random

How does beam search compare with human prediction at every time step?

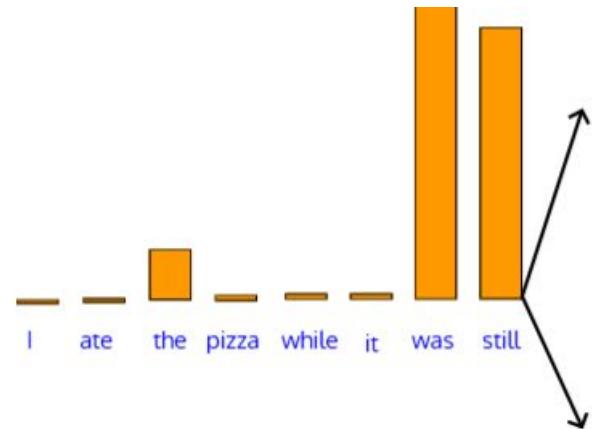
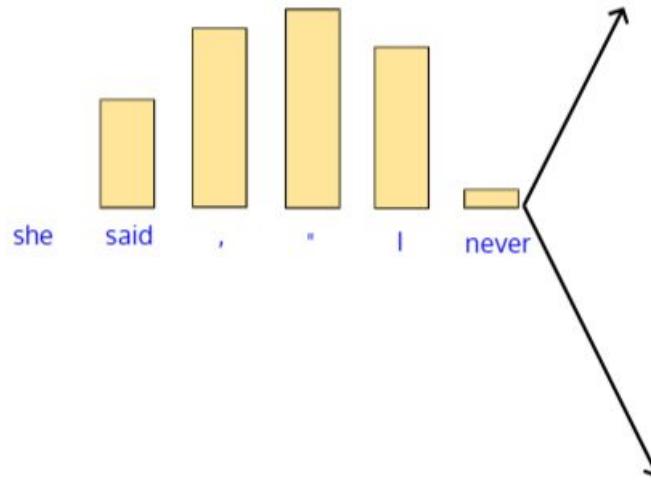


- Human predictions have a high variance whereas beam search predictions have a low variance.
- Giving a chance to other highly probable tokens leads to a variety in the generated sequences.

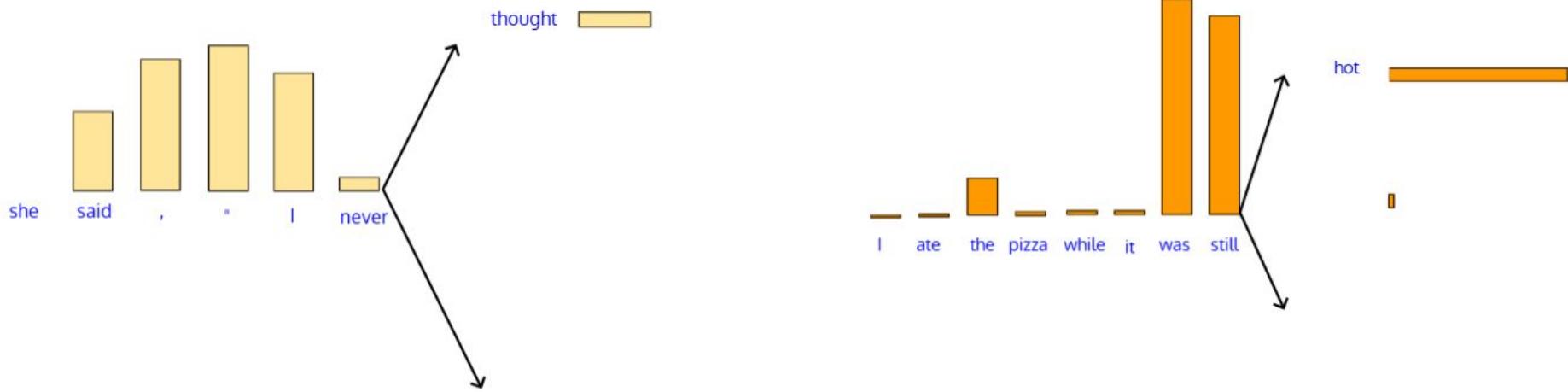
What should the optimal value of k be?



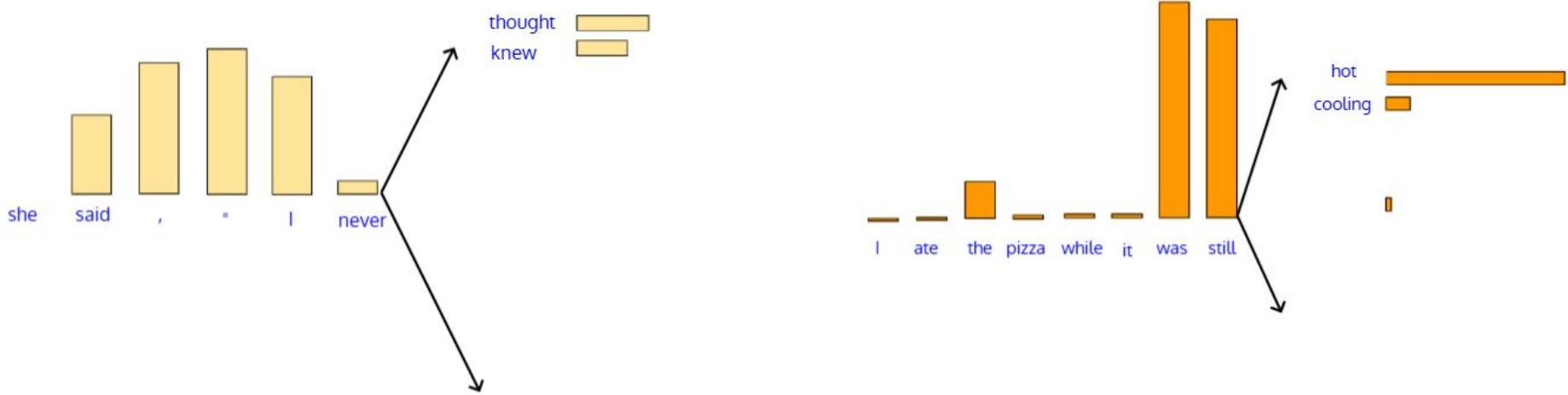
What should the optimal value of k be?



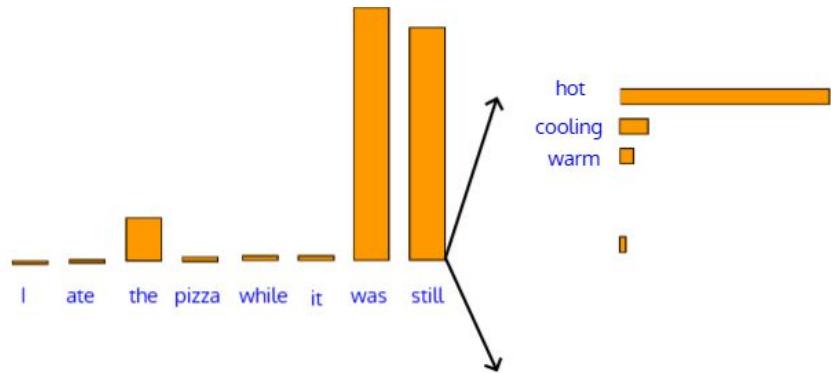
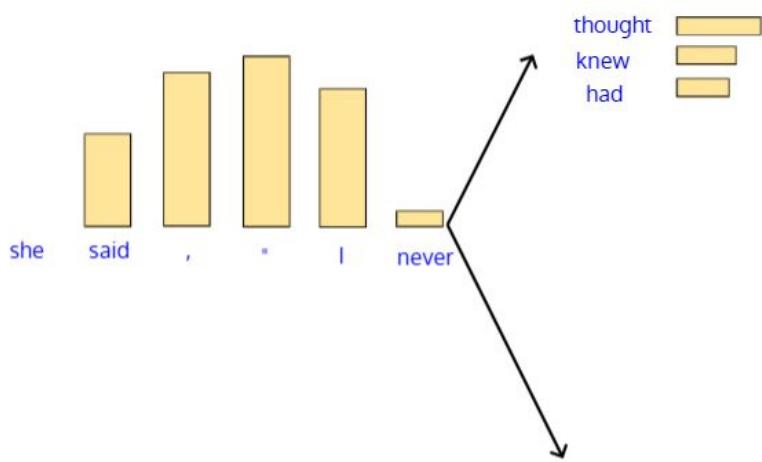
What should the optimal value of k be?



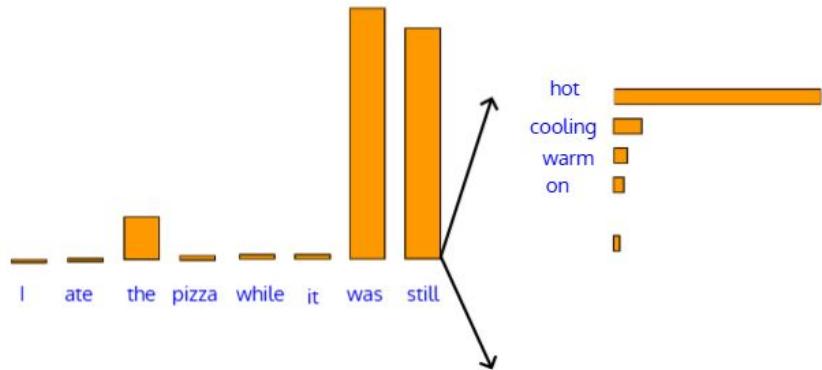
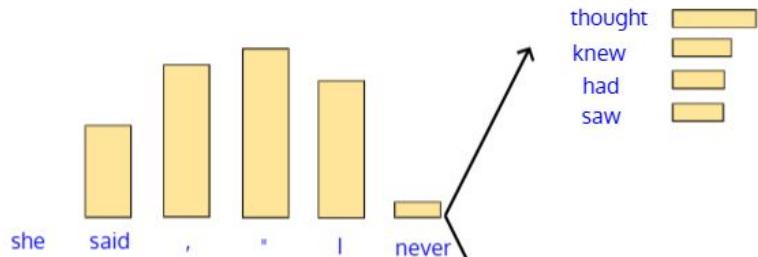
What should the optimal value of k be?



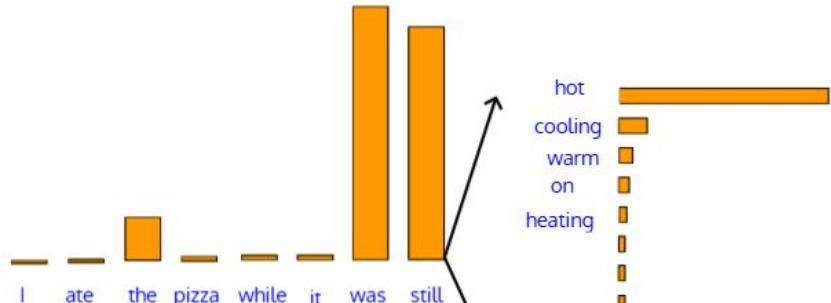
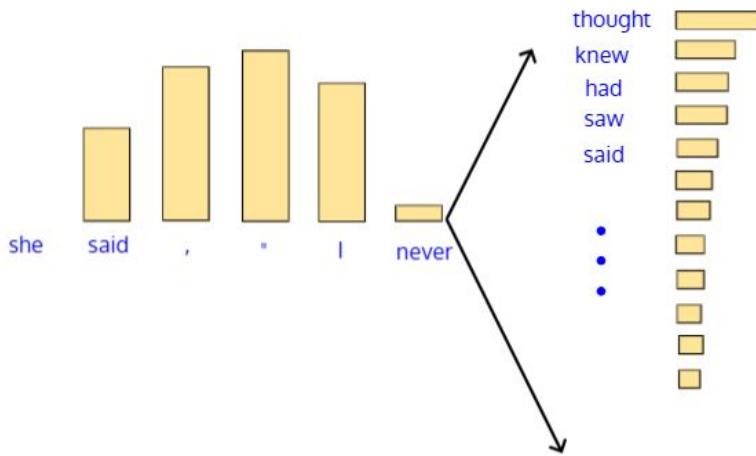
What should the optimal value of k be?



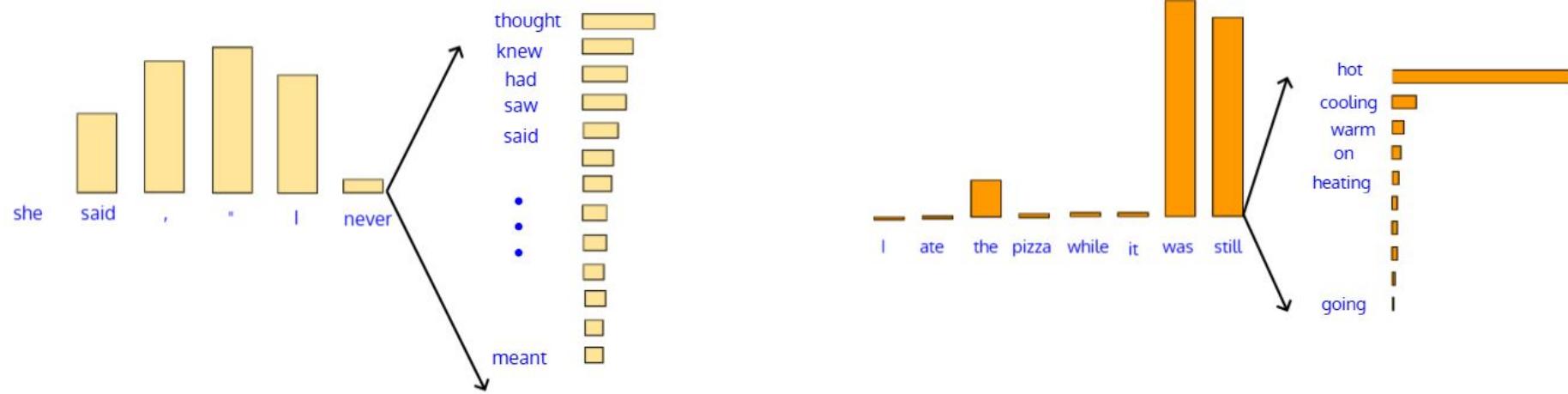
What should the optimal value of k be?



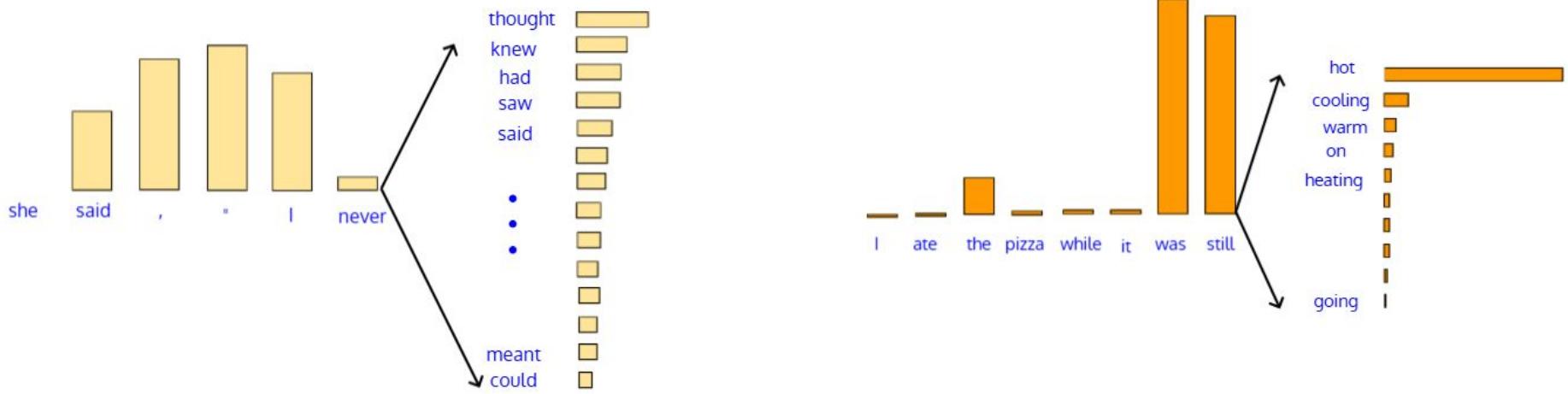
What should the optimal value of k be?



What should the optimal value of k be?



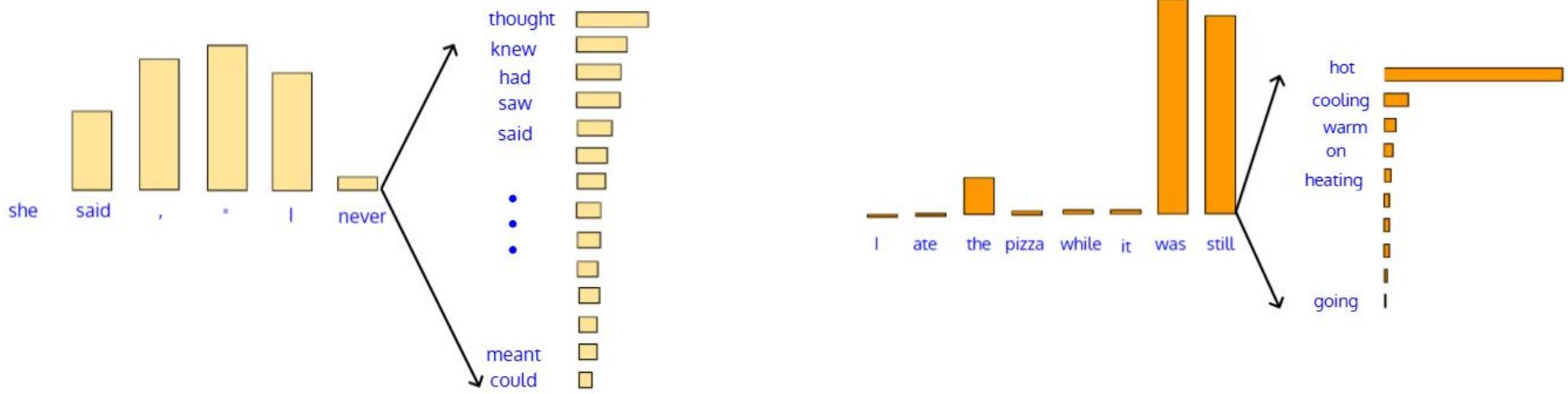
What should the optimal value of k be?



If we fix the value of, say, $k=5$, then we are missing out other equally probable tokens from the flat distribution.

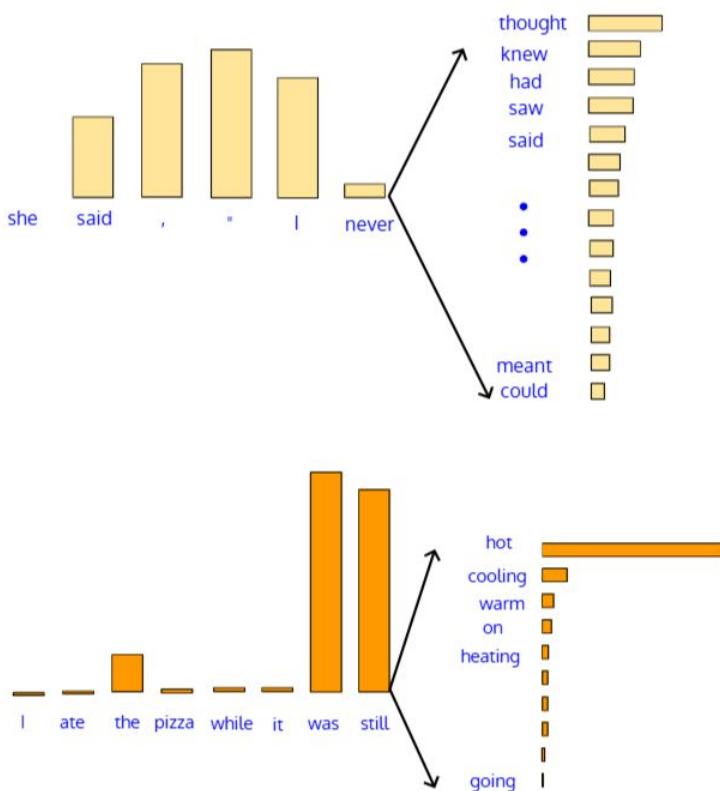
It will miss to generate a variety of sentences (less creative)

What should the optimal value of k be?



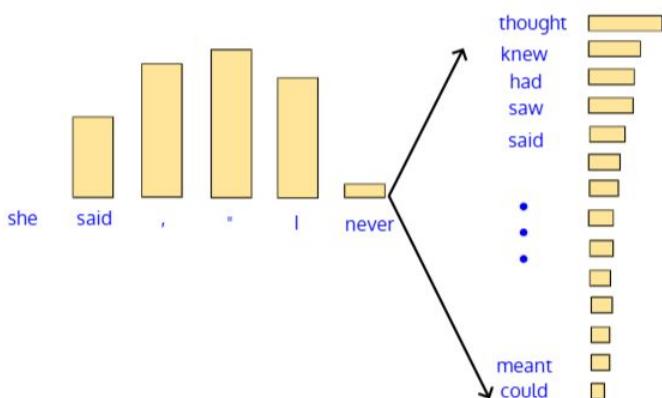
- For a peaked distribution, using the same value of $k=5$, we might end up creating some meaningless sentences
 - as we are taking tokens that are less likely to come next.

Top-P Nucleus sampling

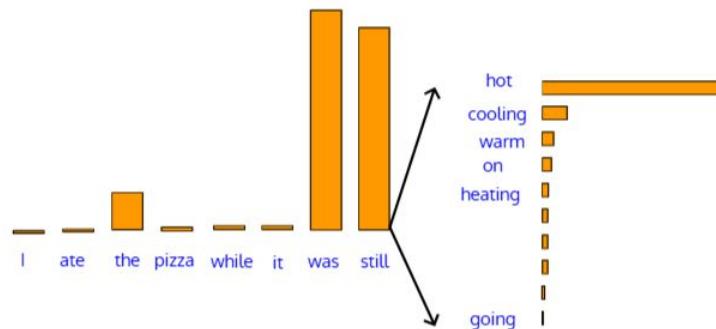
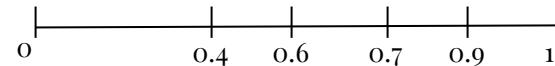


- Sort the probabilities in descending order
- Set a value for the parameter p ,
 $0 < p < 1$
- Sum the probabilities of tokens starting from the top token.
- If the sum exceeds p , then sample a token from the selected tokens.
- It is similar to top- k with k being dynamic.

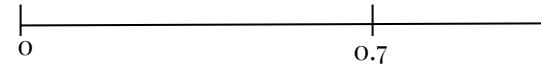
Top-P Nucleus sampling



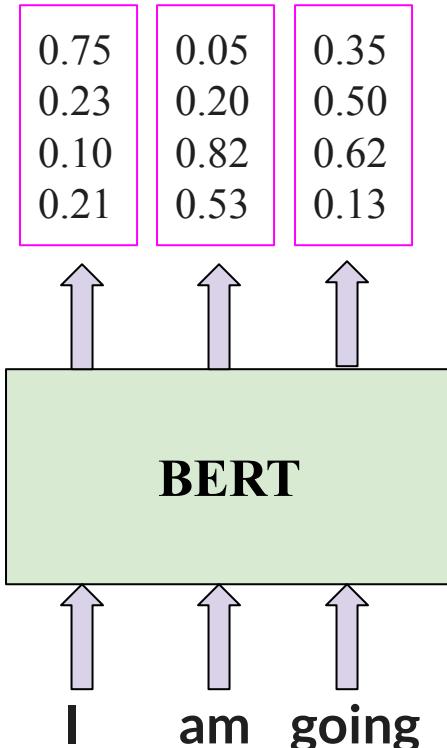
- Suppose we set $p=0.6$
- For top left distribution: the model will sample from the tokens (thought,knew,had,saw,said)



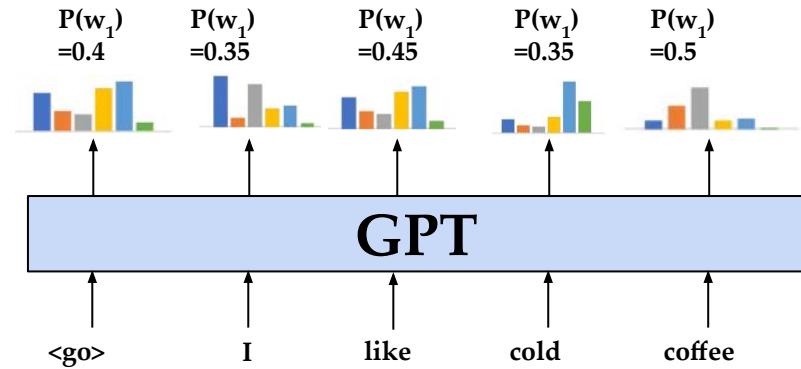
- For bottom left distribution: the model will sample from the tokens (hot,cooling)



LLM Tokenization



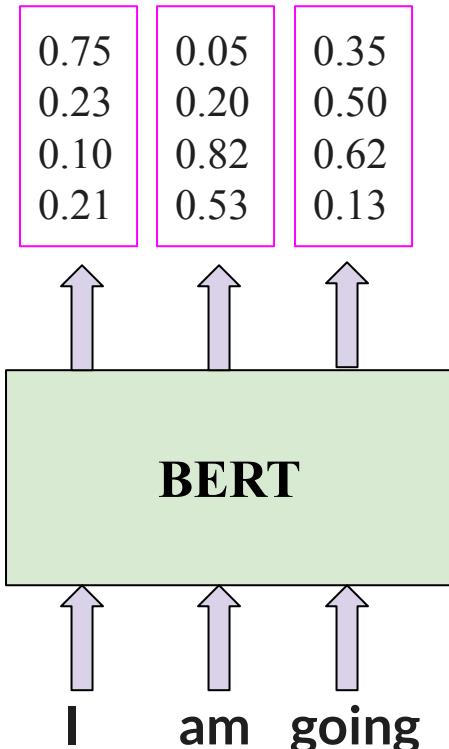
Contextual
Embeddings



Static Embeddings

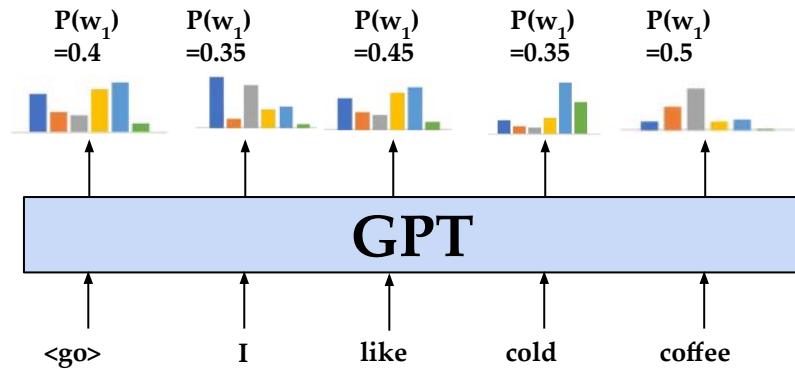
We treated each word as token.

LLM Tokenization



Contextual
Embeddings

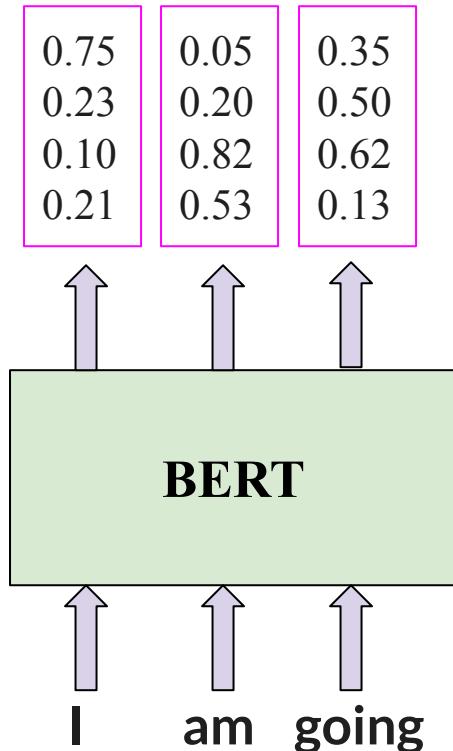
- This requires us to split the input text into tokens in a consistent way.
- A simple approach is to use whitespace for splitting the text..



Static Embeddings

We treated each word as token.

LLM Tokenization



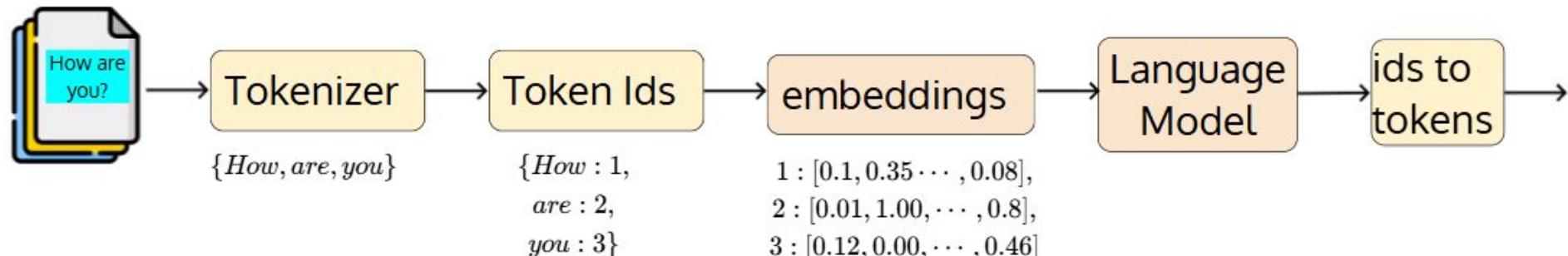
- What if a language does not have white space?

Static Embeddings

We treated each word as token.

LLM Tokenization

Role of Tokenizer at training and inference time



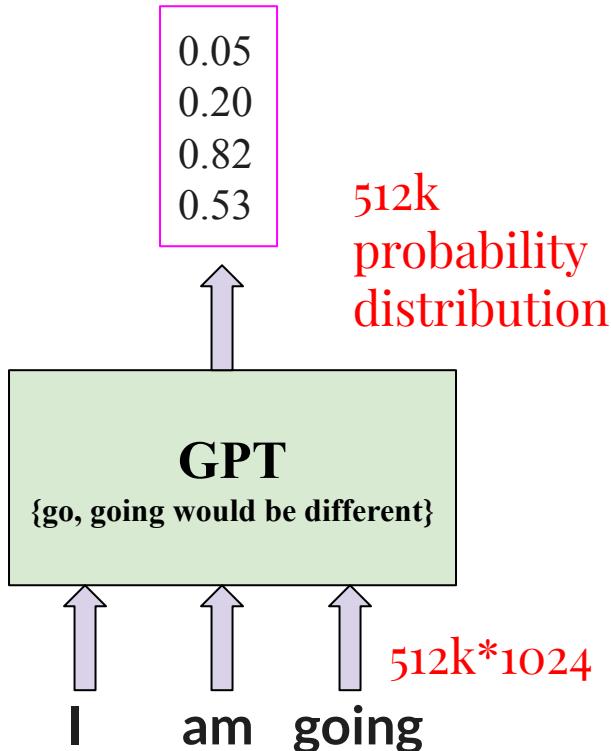
LLM Tokenization: Questions

- Is splitting the input text into words using whitespace a good approach?
 - Vocabulary will be large in 1000s.
- In that case, Do we treat the words "go" and "going" as separate tokens?
- What about languages like Japanese, which do not use any word delimiters like space?
- Why not treat each individual character in a language as a vocabulary?
 - Vocabulary will be small in 100s.
- What should be the size of vocabulary?
- Finally, what are good tokens?

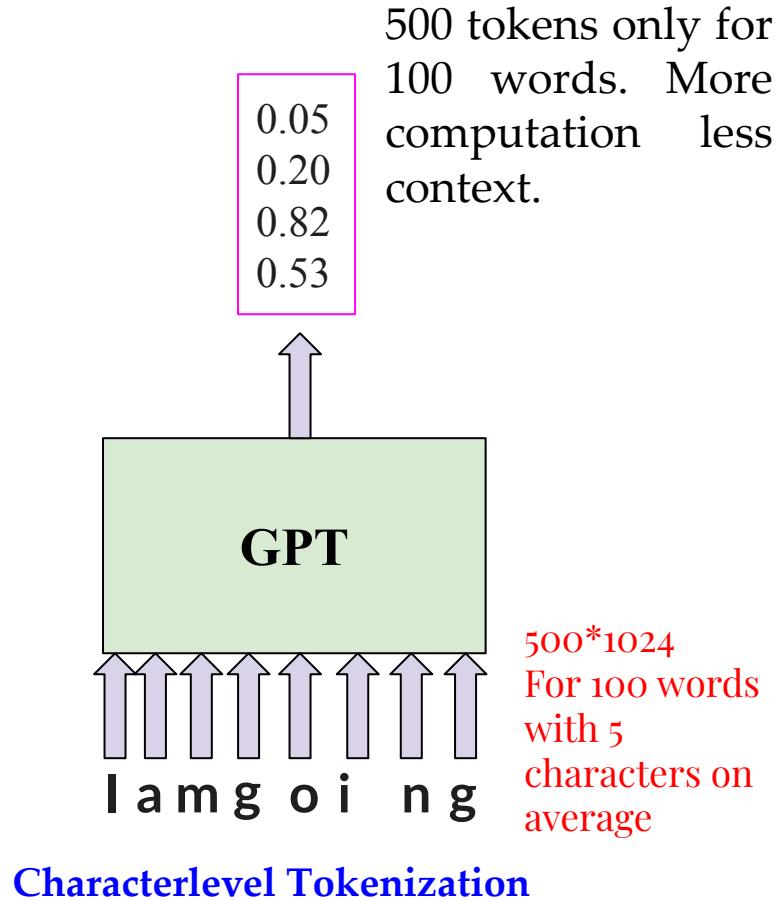
Challenges in building the vocabulary

- What should be the size of vocabulary?
 - Larger the size, larger the size of embedding matrix and greater the computation of softmax probabilities. What is the optimal size?
- Out-of-vocabulary
 - If we limit the size of the vocabulary (say, 250K to 50K) , then we need a mechanism to handle out-of-vocabulary (OOV) words. How do we handle them?
- Handling misspelled words in corpus
 - Often, the corpus is built by scraping the web. There are chances of typo/spelling errors. Such erroneous words should not be considered as unique words.
- Open Vocabulary problem
 - A new word can be constructed (say,in agglutinative languages) by combining the existing words . The vocabulary, in principle, is infinite (that is, names,numbers,..) which makes a task like machine translation challenge.

Motivation for Sub-word tokenization

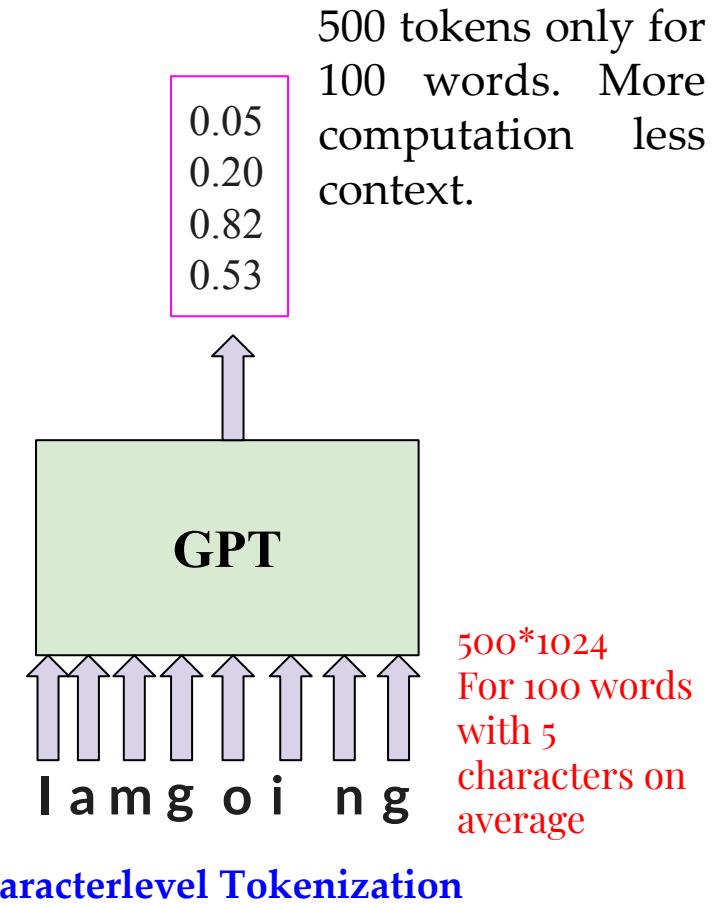
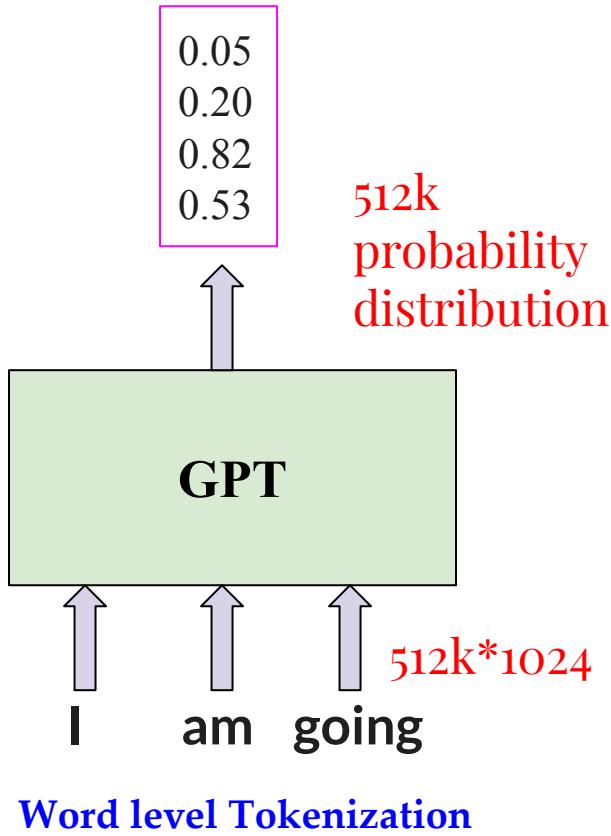


Word level Tokenization

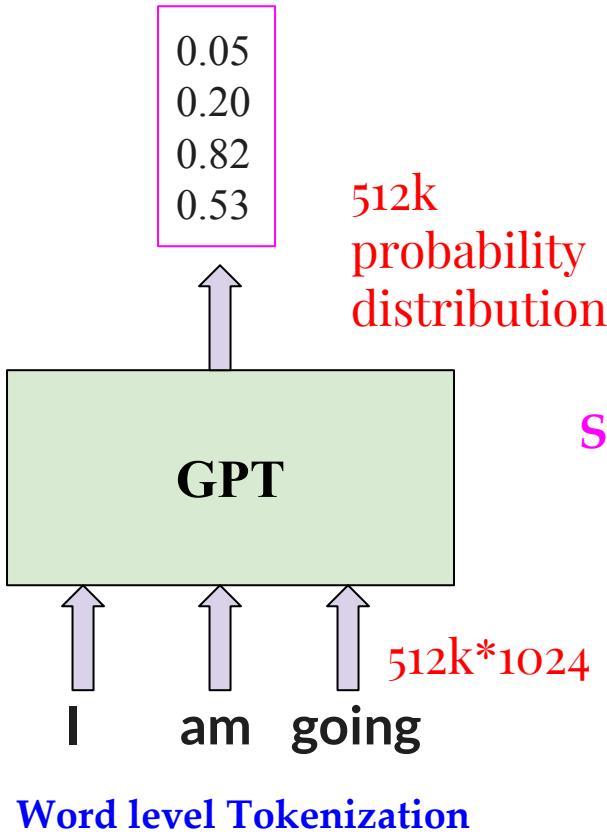


Characterlevel Tokenization

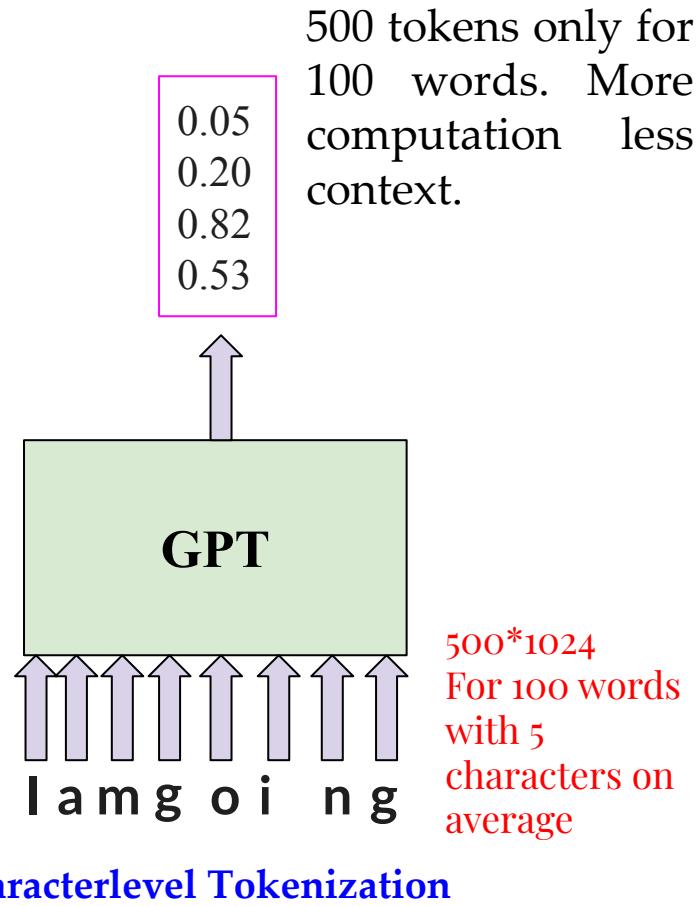
Motivation for Sub-word tokenization



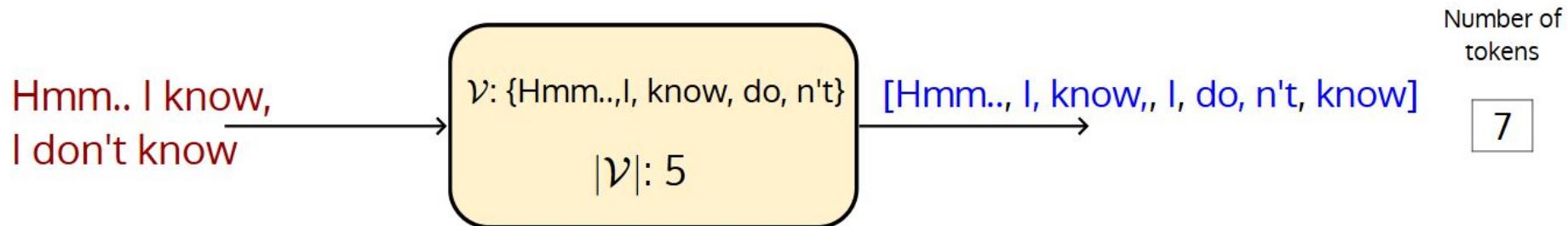
Motivation for Sub-word tokenization



Sub-word tokenization
may be a better
approach.

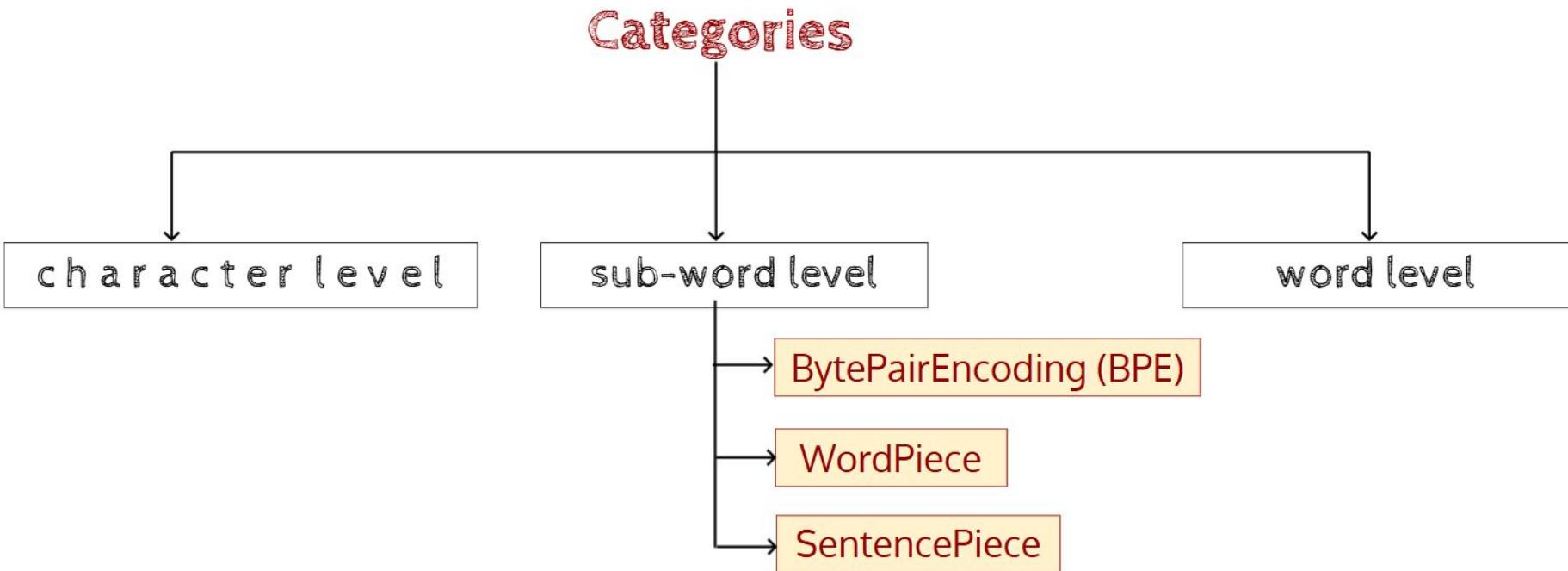


Sub-word level tokenization



- The size of the vocabulary is carefully built based on the frequency of occurrence of subwords
- A middle ground between character level and word level tokenizers
- The most frequently occurring words are preserved as is and rare words are split into subword units.
- The size of the vocabulary is moderate.
- For ex, the subword level tokenizer breaks "don't" into "do" and "n't" by learning that "n't" occurs frequently in a corpus.
- we have a representation for "do" and a representation for "n't"
- Therefore, subword level tokenizers are preferred for LLMs.

Sub-word tokenizer Categories

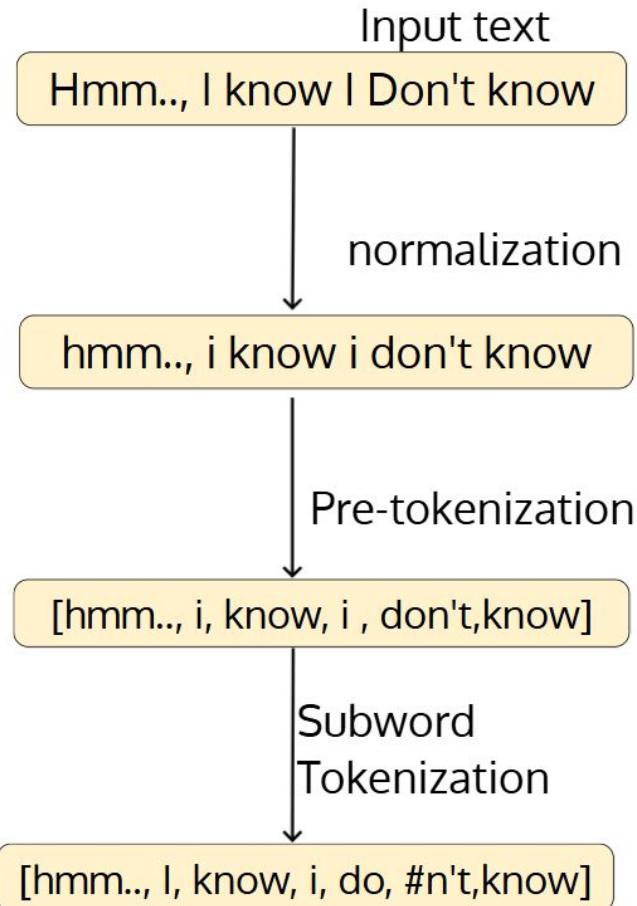


Sub-word tokenizer wishlist

- Moderate-sized Vocabulary
- Efficiently handle unknown words during inference
- Be language agnostic

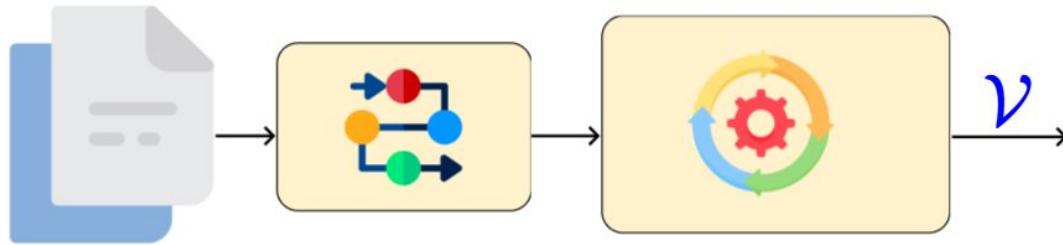
Byte-Pair Encoding

- First the text is normalized, which involves operations such as treating cases, removing accents, eliminating multiple whitespaces, handling HTML tags, etc.
- Splitting the text by whitespace was traditionally called tokenization. However, when it is used with a sub-word tokenization algorithm, it is called pre-tokenization.
- Learn the vocabulary (called training) using these words.

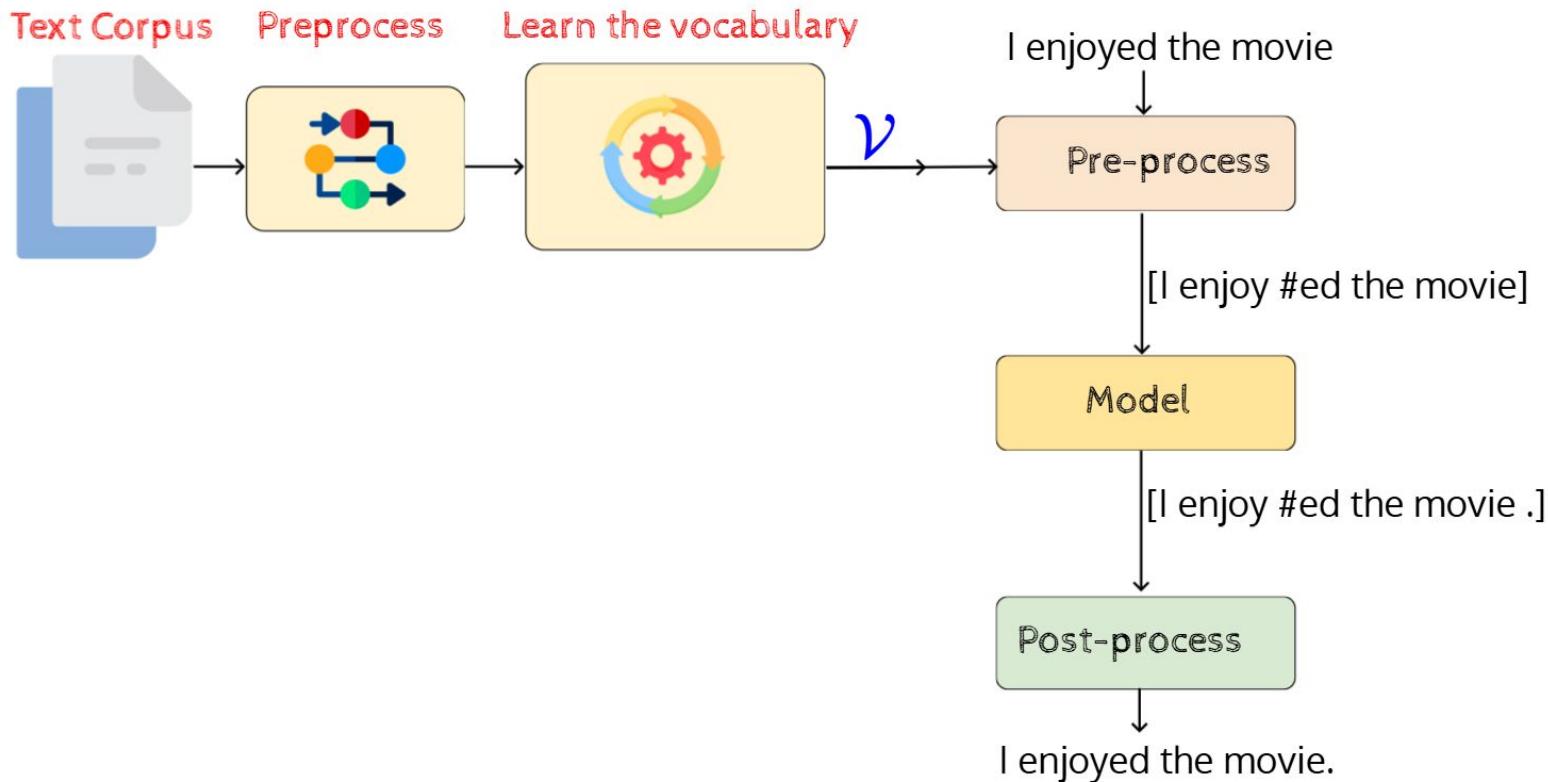


Byte-Pair Encoding

Text Corpus Preprocess Learn the vocabulary



Byte-Pair Encoding



Byte-Pair Encoding

BPE Algorithm steps:

- Start with a dictionary that contains words and their count
- Append a special symbol </w> at the end of each word in the dictionary
- Set required number of merges (a hyperparameter)
- Get the frequency count for a pair of characters
- Merge pairs with maximum occurrence
- Initialize the character-frequency table (a base vocabulary)

Byte-Pair Encoding: example

Corpus: knowing the name of something is different from knowing something.
knowing something about everything isn't bad.

Byte-Pair Encoding: example

Word
'knowing </w>'
'the </w>'
'name </w>'
'of </w>'
'something </w>'
'is </w>'
'different </w>'
'from </w>'
'something.</w>'
'about </w>'
'everything </w>'
"isn't </w>'
'bad </w>'

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

</w> identifies the word boundary.

Byte-Pair Encoding: example

Word
'knowing </w>'
'the </w>'
'name </w>'
'of </w>'
'something </w>'
'is </w>'
'different </w>'
'from </w>'
'something.</w>'
'about </w>'
'everything </w>'
"isn't </w>'
'bad </w>'

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Objective:

Find most frequently occurring byte-pair

</w> identifies the word boundary.

Byte-Pair Encoding: example

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Word	Frequency
'knowing </w>'	3
'the </w>'	1
'name </w>'	1
'of </w>'	1
'something </w>'	2
'is </w>'	1
'different </w>'	1
'from </w>'	1
'something.</w>'	1
'about </w>'	1
'everything </w>'	1
"isn't </w>'	1
'bad </w>'	1

- Let's count the word frequencies first.
- We can count character frequencies from the table

Byte-Pair Encoding: example

Word Count

Word	Frequency
'knowing </w>'	3
'the </w>	1
'name </w>	1
'of </w>'	1
'something </w>	2
'is </w>'	1
'different </w>'	1
'from </w>'	1
'something.</w>'	1
'about </w>'	1
'everything </w>'	1
"isn't </w>'	1
'bad </w>'	1

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Initial Token Count

Character	Frequency
'k'	3

- We could infer that "k" has occurred three times by counting the frequency of occurrence of words having the character "k" in it.
- In this corpus, the only word that contains "k" is the word "knowing" and it has occurred three times.

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Byte-Pair Encoding: example

Word Count: 13

Word	Frequency
'knowing </w>'	3
'the </w>	1
'name </w>	1
'of </w>'	1
'something </w>	2
'is </w>'	1
'different </w>'	1
'from </w>'	1
'something.</w>'	1
'about </w>'	1
'everything </w>'	1
"isn't </w>'	1
'bad </w>'	1

Initial Token Count

Character	Frequency
'k'	3

Character	Frequency
'k'	3
'n'	13

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Byte-Pair Encoding: example

Word Count: 13

Word	Frequency
'knowing </w>'	3 $2 * 3 = 6$
'the </w>'	1
[name] </w>	1 $1 * 1 = 1$
'of </w>'	1
'something'	2 $1 * 2 = 2$
'is </w>'	1
'different'	1 $1 * 1 = 1$
'from </w>'	1
'something.'	1 $1 * 1 = 1$
'about </w>'	1
'everything </w>'	1 $1 * 1 = 1$
"isn't </w>'	1 $1 * 1 = 1$
'bad </w>'	1

Initial Token Count

Character	Frequency
'k'	3

Character	Frequency
'k'	3
'n'	13

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Byte-Pair Encoding: example

Word Count: 13

Word	Frequency
'knowing </w>'	3 $2 * 3 = 6$
'the </w>'	1
'name </w>'	1 $1 * 1 = 1$
'of </w>'	1
'something </w>'	2 $1 * 2 = 2$
'is </w>'	1
'different </w>'	1 $1 * 1 = 1$
'from </w>'	1
'something. </w>'	1 $1 * 1 = 1$
'about </w>'	1
'everything </w>'	1 $1 * 1 = 1$
"isn't </w>'	1 $1 * 1 = 1$
'bad </w>'	1

Initial Token Count

Character	Frequency
'k'	3

Character	Frequency
'k'	3
'n'	13

Character	Frequency
'k'	3
'n'	13
'o'	9

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Byte-Pair Encoding: example

Word Count: 13

Word	Frequency
'knowing </w>'	3 $2 * 3 = 6$
'the </w>'	1
[name] </w>	1 $1 * 1 = 1$
'of </w>'	1
'something'	2 $1 * 2 = 2$
'is </w>'	1
'different'	1 $1 * 1 = 1$
'from </w>'	1
'something.'	1 $1 * 1 = 1$
'about </w>'	1
'everything </w>'	1 $1 * 1 = 1$
"isn't </w>'	1 $1 * 1 = 1$
'bad </w>'	1

Initial Token Count

Character	Frequency
'k'	3

Character	Frequency
'k'	3
'n'	13

Character	Frequency
'k'	3
'n'	13
'o'	9

Character	Frequency
'k'	3
'n'	13
'o'	9
:	:
'</w>'	$3+1+1+1+2+\dots+1=16$

Byte-Pair Encoding: example

Word Count: 13

Word	Frequency
'knowing </w>'	3 $2 * 3 = 6$
'the </w>'	1
'name </w>'	1 $1 * 1 = 1$
'of </w>'	1
'something </w>'	2 $1 * 2 = 2$
'is </w>'	1
'different </w>'	1 $1 * 1 = 1$
'from </w>'	1
'something. </w>'	1 $1 * 1 = 1$
'about </w>'	1
'everything </w>'	1 $1 * 1 = 1$
"isn't </w>'	1 $1 * 1 = 1$
'bad </w>'	1

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Initial Token Count

Character	Frequency
'k'	3

Character	Frequency
'k'	3
'n'	13

Character	Frequency
'k'	3
'n'	13
'o'	9

Character	Frequency
'k'	3
'n'	13
'o'	9
:	:
'</w>'	16
:	:
'''	1

Character	Frequency
'k'	3
'n'	13
'o'	9
:	:
'</w>'	16
:	:
'''	1

Initial Vocab size: 22

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Byte-Pair Encoding: example

Word count

Word	Frequency
'knowing </w>'	3
'the </w>'	1
'name </w>'	1
'of </w>'	1
'something </w>'	2
'is </w>'	1
'different </w>'	1
'from </w>'	1
'something. </w>'	1
'about </w>'	1
'everything </w>'	1
"isn't </w>'	1
'bad </w>'	1

Byte-Pair count

Word	Frequency
('k', 'n')	3

Character	Frequency
'k'	3
'n'	13
'o'	9
:	:
'</w>'	16
:	:
'''	1

Initial Vocab size: 22

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Byte-Pair Encoding: example

Word count

Word	Frequency
'k n owing </w>'	3
'the </w>'	1
'n a m e </w>'	1
'o f </w>'	1
's o m e t h i n g </w>'	2
'i s </w>'	1
'd i f f e r e n t </w>'	1
'f r o m </w>'	1
's o m e t h i n g . </w>'	1
'a b o u t </w>'	1
'e v e r y t h i n g </w>'	1
"i s n ' t </w>"	1
'b a d </w>'	1

Byte-Pair count

Word	Frequency
('k', 'n')	3
('n', 'o')	3

Character	Frequency
'k'	3
'n'	13
'o'	9
:	:
'</w>'	16
:	:
'''	1

Initial Vocab size: 22

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Byte-Pair Encoding: example

Word count

Word	Frequency
'k n <ins>ow</ins> ing </w>'	3
'the </w>	1
'name </w>	1
'of </w>'	1
'something </w>	2
'is </w>'	1
'different </w>'	1
'from </w>'	1
'something.</w>'	1
'about </w>'	1
'everything </w>'	1
"isn't </w>'	1
'bad </w>'	1

Byte-Pair count

Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3

Character	Frequency
'k'	3
'n'	13
'o'	9
:	:
'</w>'	16
:	:
'''	1

Initial Vocab size: 22

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Byte-Pair Encoding: example

Word count

Word	Frequency
'k n o w i n g </w>'	3
't h e </w>'	1
'n a m e </w>'	1
'o f </w>'	1
's o m e t h i n g </w>'	2
'i s </w>'	1
'd i f f e r e n t </w>'	1
'f r o m </w>'	1
's o m e t h i n g . </w>'	1
'a b o u t </w>'	1
'e v e r y t h i n g </w>'	1
"i s n ' t </w>'	1
'b a d </w>'	1

Byte-Pair count

Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3
('w', 'i')	3
('i', 'n')	$3 + 2 + 1 + 1 = 7$

Character	Frequency
'k'	3
'n'	13
'o'	9
:	:
'</w>'	16
:	:
'''	1

Initial Vocab size: 22

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Byte-Pair Encoding: example

Word count

Word	Frequency
'knowing </w>'	3
'the </w>'	1
'name </w>'	1
'of </w>'	1
'something </w>'	2
'is </w>'	1
'different </w>'	1
'from </w>'	1
'something. </w>'	1
'about </w>'	1
'everything </w>'	1
"isn't </w>'	1
'bad </w>'	1

Byte-Pair count

Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3
('w', 'i')	3
('i', 'n')	7
('n', 'g')	7
('g', '</w>')	6
('t', 'h')	5
('h', 'e')	1
('e', '</w>')	2
⋮	⋮
('a', 'd')	1
('d', '</w>')	1

Character	Frequency
'k'	3
'n'	13
'o'	9
⋮	⋮
'</w>'	16
⋮	⋮
'''	1

Initial Vocab size: 22

Corpus: knowing the name of something is different from knowing something. knowing something about everything isn't bad.

Byte-Pair Encoding: example

Word count

Word	Frequency
'knowing </w>'	3
'the </w>'	1
'name </w>'	1
'of </w>'	1
'something </w>'	2
'is </w>'	1
'different </w>'	1
'from </w>'	1
'something. </w>'	1
'about </w>'	1
'everything </w>'	1
"isn't </w>'	1
'bad </w>'	1

Byte-Pair count

Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3
('w', 'i')	3
('i', 'n')	7
('n', 'g')	7
('g', '</w>')	6
('t', 'h')	5
('h', 'e')	1
('e', '</w>')	2
:	:
('a', 'd')	1
('d', '</w>')	1

Merge the most commonly occurring pair : (i,n)→in

Character	Frequency
'k'	3
'n'	13
'o'	9
:	:
'</w>'	16
:	:
'''	1

Initial Vocab size: 22

"in"	7
------	---

Byte-Pair Encoding: example

Word count

Word	Frequency
'knowing </w>'	3
'the </w>'	1
'name </w>'	1
'of </w>'	1
'something </w>'	2
'is </w>'	1
'different </w>'	1
'from </w>'	1
'something. </w>'	1
'about </w>'	1
'everything </w>'	1
"isn't </w>'	1
'bad </w>'	1

Byte-Pair count

Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3
('w', 'i')	3
('l', 'n')	7
('n', 'g')	7
('g', '</w>')	6
('t', 'h')	5
('h', 'e')	1
('e', '</w>')	2
:	:
('a', 'd')	1
('d', '</w>')	1

Updated Vocabulary

Character	Frequency
'k'	3
'n'	13 - 7 = 6
'o'	9
't'	10 - 7 = 3
'</w>'	16
:	:
""	1
"in"	7

Added new token

Merge the most commonly occurring pair

Byte-Pair Encoding: example

Word count

Word	Frequency
'knowing </w>'	3
'the </w>'	1
'name </w>'	1
'of </w>'	1
'something </w>'	2
'is </w>'	1
'different </w>'	1
'from </w>'	1
'something. </w>'	1
'about </w>'	1
'everything </w>'	1
"isn't </w>'	1
'bad </w>'	1

Byte-Pair count

Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3
('w', 'i')	3
('l', 'n')	7
('n', 'g')	7
('g', '</w>')	6
('t', 'h')	5
('h', 'e')	1
('e', '</w>')	2
:	:
('a', 'd')	1
('d', '</w>')	1

Updated Vocabulary

Character	Frequency
'k'	3
'n'	13 - 7 = 6
'o'	9
'l'	10 - 7 = 3
'</w>'	16
:	:
""	1
"in"	7

Added new token

Merge the most commonly occurring pair → Update the token count

Byte-Pair Encoding: example

Word count

Word	Frequency
'k n o w <i>in</i> g </w>'	3
't h e </w>	1
'n a m e </w>	1
'o f </w>'	1
's o m e t h <i>in</i> g </w>'	2
'i s </w>'	1
'd i f f e r e n t </w>'	1
'f r o m </w>'	1
's o m e t h <i>in</i> g . </w>'	1
'a b o u t </w>'	1
'e v e r y t h <i>in</i> g </w>'	1
"i s n ' t </w>"	1
'b a d </w>'	1

Byte-Pair count

Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3
('w', 'i')	3
('i', 'n')	7
('n', 'g')	7
('g', '</w>')	6
('t', 'h')	5
('h', 'e')	1
('e', '</w>')	2
:	:
('a', 'd')	1
('d', '</w>')	1

Updated vocabulary

Character	Frequency
'k'	3
'n'	6
'o'	9
'i'	3
'</w>'	16
:	:
'''	1
'g': 7	7
"in"	7

Now treat “in” as a single token and update its count

In Byte-pair, replace ‘i’ and ‘n’ with ‘in’.

Byte-Pair Encoding: example

Word count

Word	Frequency
'k now <i>ing</i> </w>'	3
't he </w>	1
'n a m e </w>	1
'o f </w>'	1
's o m e t h <i>ing</i> </w>'	2
'i s </w>'	1
'd i f f e r e n t </w>'	1
'f r o m </w>'	1
's o m e t h <i>ing</i> . </w>'	1
'a b o u t </w>'	1
'e v e r y t h <i>ing</i> </w>'	1
"i s n ' t </w>'	1
'b a d </w>'	1

Byte-Pair count

Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3
('w', 'i')	3
('w', 'in')	3
('in', 'g')	7
('g', '</w>')	6
('t', 'h')	5
('h', 'e')	1
('e', '</w>')	2
:	:
('a', 'd')	1
('d', '</w>')	1

Updated vocabulary

Character	Frequency
'k'	3
'n'	6
'o'	9
'i'	3
'</w>'	16
:	:
""	1
'g': 7	7
"in"	7

the new byte pairs are (w,in):3,(in,g):7, (h,in):4

Byte-Pair Encoding: example

Word count

Word	Frequency
'k n o w i n g </w>'	3
't h e </w>'	1
'n a m e </w>'	1
'o f </w>'	1
's o m e t h i n g </w>'	2
'i s </w>'	1
'd i f f e r e n t </w>'	1
'f r o m </w>'	1
's o m e t h i n g . </w>'	1
'a b o u t </w>'	1
'e v e r y t h i n g </w>'	1
"i s n ' t </w>'	1
'b a d </w>'	1

Byte-Pair count

Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3
('w', 'in')	3
('h', 'in')	4
('in', 'g')	7
('g', '</w>')	6
('t', 'h')	5
('h', 'e')	1
('e', '</w>')	2
:	:
('a', 'd')	1
('d', '</w>')	1

Updated vocabulary

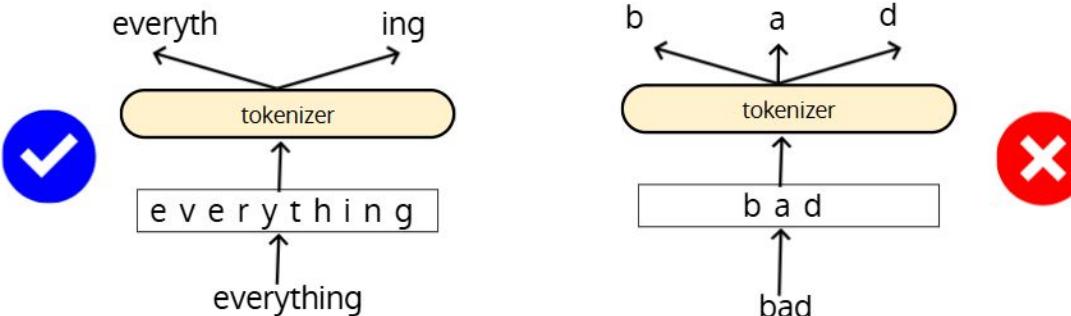
Character	Frequency
'k'	3
'n'	6
'o'	9
'l'	3
'</w>'	16
:	:
'''	1
'g': 7	7
"in"	7
"ing"	7

the new byte pairs are (w,in):3,(in,g):7, (h,in):4. Of all these pairs, merge most frequently occurring byte-pairs which turns out to be "ing" Now, treat "ing" as a single token and repeat the steps

Byte-Pair Encoding: example

After 45 merges

- The final vocabulary contains initial vocabulary and all the merges (in order). The rare words are broken down into two or more subwords
- At test time, the input word is split into a sequence of characters, and the characters are merged into a larger and known symbols

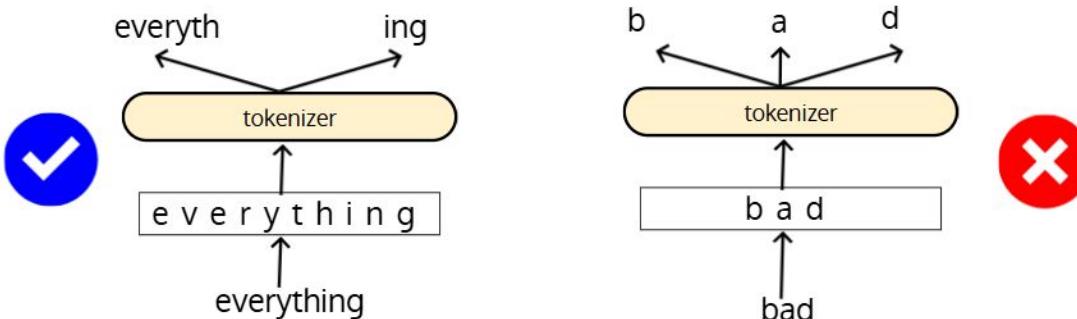


Tokens
'k'
'n'
'o'
'i'
'</w>'
:
'in'
'ing'
:
'knowing </w>'
'the </w>'
'name </w>'
'of </w>'
'something </w>'
'is </w>'
'different </w>'
'from </w>'
'something.</w>'
'about </w>'
'everyth'
'isn't </w>'
'bad </w>'

Byte-Pair Encoding: example

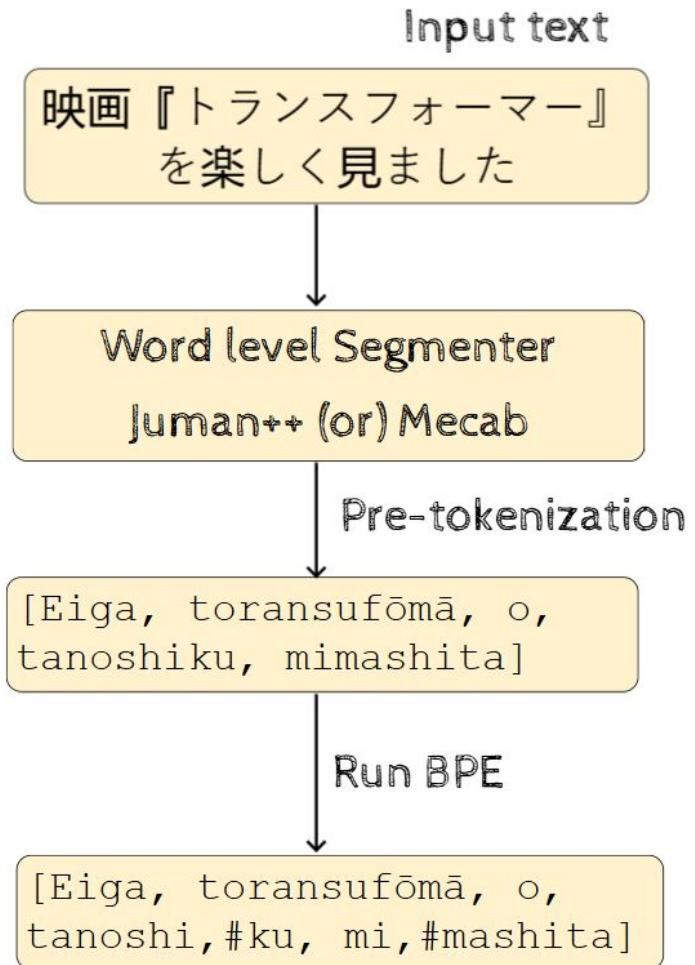
After 45 merges

- The algorithm offers a way to adjust the size of vocabulary as a function of number of merges.
- For a larger corpus, we often end up with vocabulary of size smaller than considering individual words as tokens.



Tokens
'k'
'n'
'o'
'i'
'</W>'
:
'in'
'ing'
:
'knowing </W>'
'the </W>'
'name </W>'
'of </W>'
'something </W>'
'is </W>'
'different </W>'
'from </W>'
'something.</W>'
'about </W>'
'everyth'
'isn't </W>'
'bad </W>'

BPE for non-segmented language



BPE example

$V = \{a, b, c, \dots, z, lo, he\}$

Tokenize the text "hello lol"

[h e l l o], [l o l]

search for the byte-pair 'lo', if present merge

Yes. Therefore, Merge

[h e l lo], [lo l]

search for the byte-pair 'he', if present merge

Yes. Therefore, Merge

[he l lo], [lo l]

return the text after merge

he #l #lo, lo #l

Word Piece Tokenizer

- In BPE we merged **a pair** of tokens which has the highest frequency of occurrence.
- What if there are more than one pair that is occurring with the same frequency, for example ('i','n') and ('n','g')?
- Take the frequency of occurrence of individual tokens in the pair into an account

$$score = \frac{count(\alpha, \beta)}{count(\alpha)count(\beta)}$$

Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3
('w', 'i')	3
('i', 'n')	7
('n', 'g')	7
('g', '.')	1
('t', 'h')	5
('h', 'e')	1
('e', '</w>')	2
:	:
('a', 'd')	1

Word Piece Tokenizer

- Now we can select a pair of tokens where the individual tokens are less frequent in the vocabulary
- The WordPiece algorithm uses this score to merge the pairs.

$$score = \frac{count(\alpha, \beta)}{count(\alpha)count(\beta)}$$

Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3
('w', 'i')	3
('i', 'n')	7
('n', 'g')	7
('g', '.')	1
('t', 'h')	5
('h', 'e')	1
('e', '</w>')	2
:	:
('a', 'd')	1

Word Piece Tokenizer

Word count

Word	Frequency
'knowing'	3
'the'	1
'name'	1
'of'	1
'something'	2
'is'	1
'different'	1
'from'	1
'something.'	1
'about'	1
'everything'	1
"isn't"	1
'bad'	1

Initial Vocab Size :22

Character	Frequency
'k'	3
'#n'	13
'#o'	9
:	:
't'	16
'#h'	5
'''	1
:	:

Word Piece Tokenizer

Now, merging is based on the score of each byte pair

$$score = \frac{count(\alpha, \beta)}{count(\alpha)count(\beta)}$$

Word	Frequency
'knowing'	3
'the'	1
'name'	1
'of'	1
'something'	2
'is'	1
'different'	1
'from'	1
'something.'	1
'about'	1
'everything'	1
"isn't"	1
'bad'	1

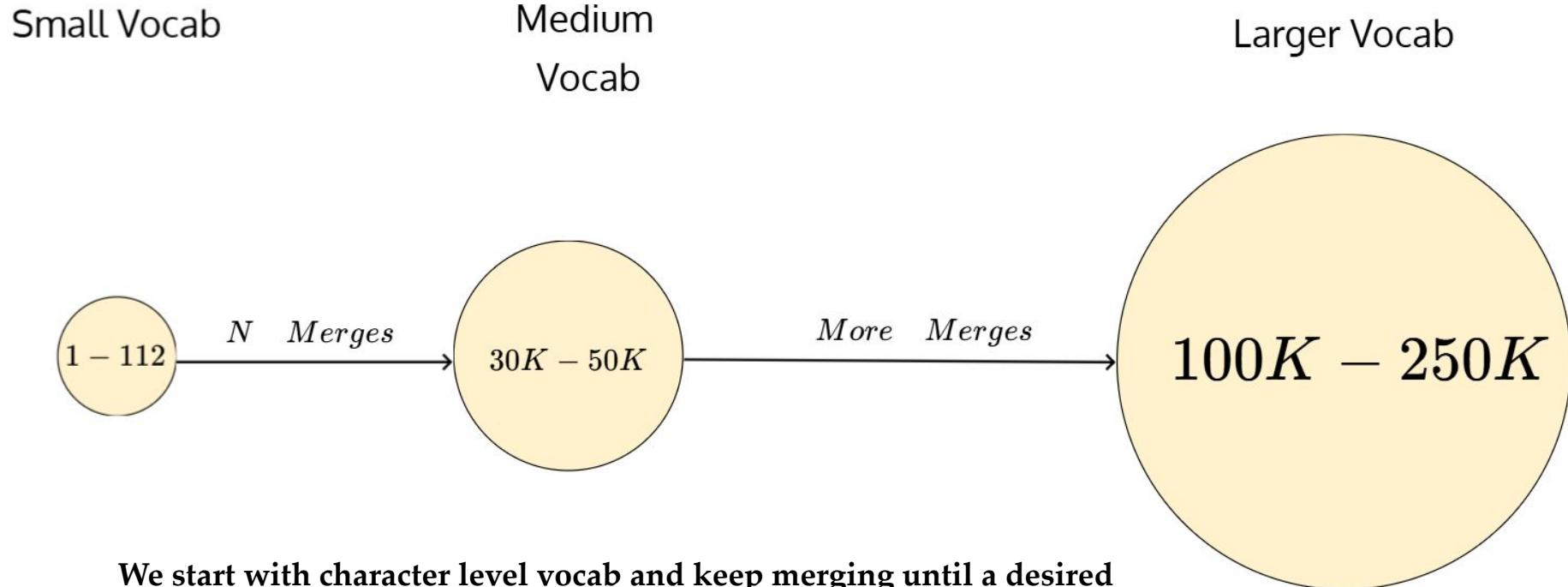
Word	Frequency
('k', 'n')	3
('n', 'o')	3
('o', 'w')	3
('w', 'i')	3
('i', 'n')	7
('n', 'g')	7
('g', '.')	1
('t', 'h')	5
('h', 'e')	1
('e', '</w>')	2
:	:
('a', 'd')	1

Freq of 1st element	Freq of 2nd element	score
'k':3	'n':13	0.076
'n':13	'o':9	0.02
'o':9	'w':3	0.111
'i':10	'n':13	0.05
'n':13	'g':7	0.076
't':8	'h':5	0.125
'a':3	'd':2	0.16

We merge tokens with less count

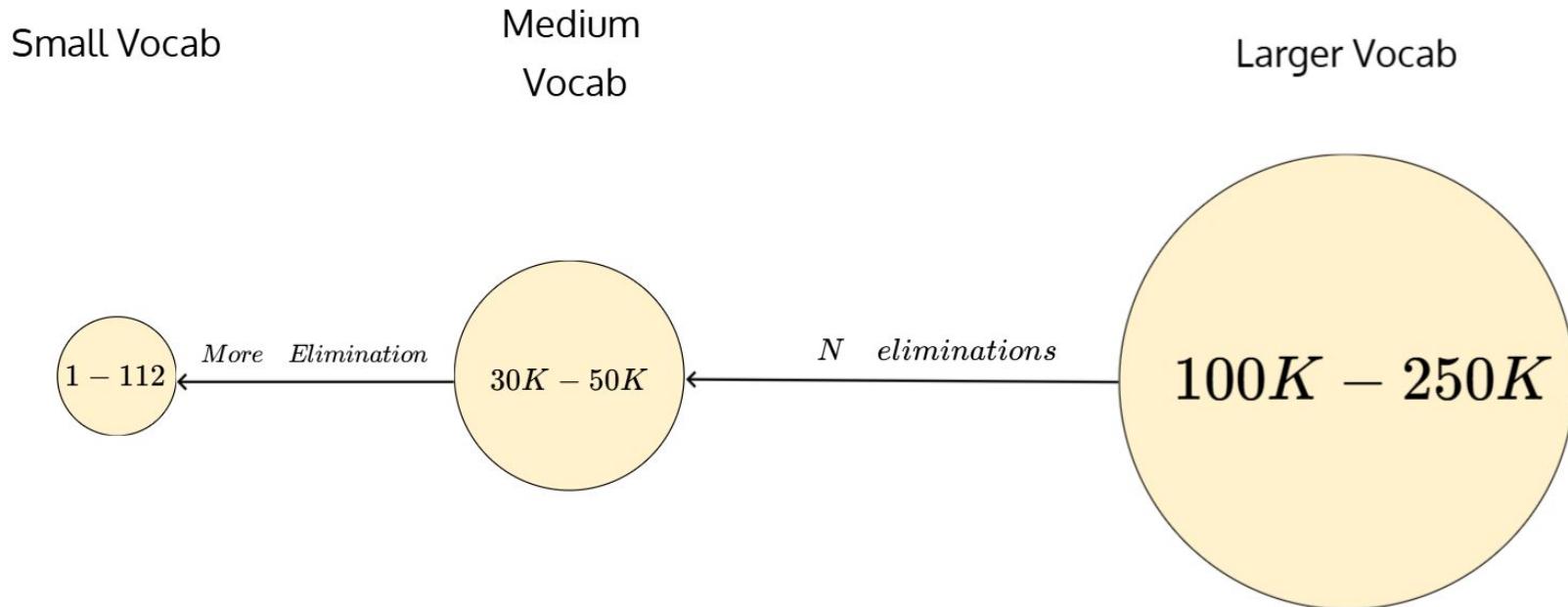
Merge the pair with highest score

Tokenization



Tokenization

Well, we can do the reverse as well.



We start with word level vocab and and keep eliminating words until a desired vocabulary size is reached.

SentencePiece Tokenizer

A given word can have numerous subwords.

For instance, the word "X=hello"

can be segmented in multiple ways (by BPE) even with the **same vocabulary**

$V=\{h,e,l,l,o,he,el,lo,ll,hell\}$

$x1=he, ll, o$, $x2=h, el, lo$, $x3=he, l, lo$, $x4=hell, o$

however, following the merge rule, BPE outputs: he, l, lo

On the other hand, if $V=\{h,e,l,l,o,el,he,lo,ll,hell\}$

then BPE outputs: h, el, lo

SentencePiece Tokenizer

A given word can have numerous subwords.

For instance, the word "X=hello"

can be segmented in multiple ways (by BPE) even with the **same vocabulary**

$V=\{h,e,l,l,o,he,el,lo,ll,hell\}$

$x1=he, ll, o$, $x2=h, el, lo$, $x3=he, l, lo$, $x4=hell, o$

however, following the merge rule, BPE outputs: he, l, lo

On the other hand, if $V=\{h,e,l,l,o,el,he,lo,ll,hell\}$

then BPE outputs: h, el, lo

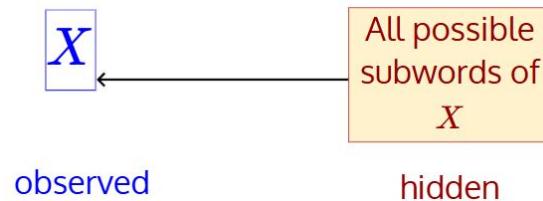
Therefore, we say BPE is **greedy** and **deterministic** (we can use BPE-Dropout to make it **stochastic**)

SentencePiece Tokenizer

- The probabilistic approach is to find the subword sequence

$$x \in \{ x_1, x_2, \dots, x_k \}$$

- that maximizes the likelihood of the word X .



- The word X in *sentencepiece* means a sequence of characters or words (without spaces).
- Therefore, it can be applied to languages (like Chinese and Japanese) that do not use any word delimiter in sentence.

SentencePiece Tokenizer

Let $X = \text{"knowing"}$ and a few segmentation candidates be

$$S(X) = \{k, now, ing, know, ing, knowing\}$$

Given the unigram language model we can calculate the probabilities of the segments as follows:

$$\begin{aligned} p(\mathbf{x}_1 = k, now, ing) &= p(k)p(now)p(ing) \\ &= \frac{3}{16} \times \frac{3}{16} \times \frac{7}{16} = \frac{63}{4096} \end{aligned}$$

$$p(\mathbf{x}_2 = know, ing) = p(know)p(ing) = \frac{21}{256} = \frac{336}{4096}$$

$$p(\mathbf{x}_3 = knowing) = p(knowing) = \frac{3}{16} = \frac{768}{4096}$$

$$\boxed{\mathbf{x}^* = \arg \max_{\mathbf{x} \in S(X)} P(\mathbf{x}) = \mathbf{x}_3}$$

Word	Frequency
'knowing'	3
'the'	1
'name'	1
'of'	1
'something'	2
'is'	1
'different'	1
'from'	1
'something.'	1
'about'	1
'everything'	1
"isn't"	1
'bad'	1

SentencePiece Tokenizer

Let $X = \text{"knowing"}$ and a few segmentation candidates be

$$S(X) = \{k, now, ing, know, ing, knowing\}$$

Given the unigram language model we can calculate the probabilities of the segments as follows:

$$\begin{aligned} p(\mathbf{x}_1 = k, now, ing) &= p(k)p(now)p(ing) \\ &= \frac{3}{16} \times \frac{3}{16} \times \frac{7}{16} = \frac{63}{4096} \end{aligned}$$

$$\begin{aligned} p(\mathbf{x}_2 = know, ing) &= p(know)p(ing) = \frac{21}{256} = \frac{336}{4096} \\ p(\mathbf{x}_3 = knowing) &= p(knowing) = \frac{3}{16} = \frac{768}{4096} \end{aligned}$$

$$\boxed{\mathbf{x}^* = \arg \max_{\mathbf{x} \in S(X)} P(\mathbf{x}) = \mathbf{x}_3}$$

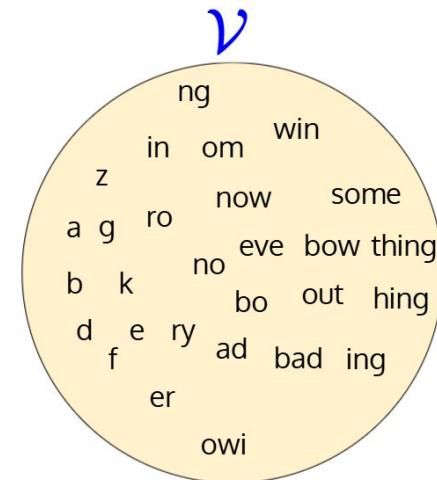
Word	Frequency
'knowing'	3
'the'	1
'name'	1
'of'	1
'something'	2
'is'	1
'different'	1
'from'	1
'something.'	1
'about'	1
'everything'	1
"isn't"	1
'bad'	1

In practice, we use Viterbi decoding to find \mathbf{x}^* instead of enumerating all possible segments.

SentencePiece Tokenizer- Algorithm

Set the desired vocabulary size

1. Construct a reasonably large seed vocabulary using BPE or Extended Suffix Array algorithm.
2. **E-Step:**
Estimate the probability for every token in the given vocabulary using frequency counts in the training corpus
3. **M-Step:**
Use Viterbi algorithm to segment the corpus and return optimal segments that maximizes the (log) likelihood.
4. Compute the likelihood for each new subword from optimal segments
5. Shrink the vocabulary size by removing top $x\%$ of subwords that have the smallest likelihood.
6. Repeat step 2 to 5 until desired vocabulary size is reached



SentencePiece Tokenizer- Algorithm

Let us consider segmenting the word "whereby" using Viterbi decoding:

whereby
 ^
 |
 w

Iterate over every position in the given word and output the segment which has highest likelihood

Token	$\log(p(x))$
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

SentencePiece Tokenizer- Algorithm

Let us consider segmenting the word "whereby" using Viterbi decoding:

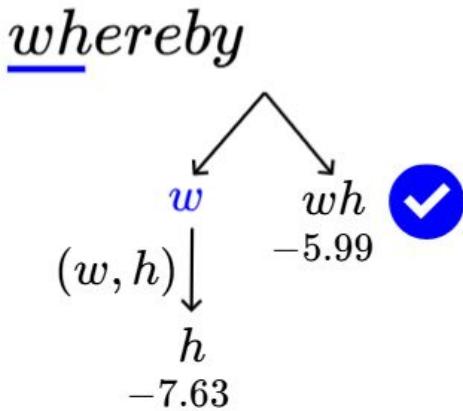
whereby
 ^
 |
 w

At this position, the possible segmentations of the slice "wh" are (w,h) and (wh)

Token	$\log(p(x))$
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

SentencePiece Tokenizer- Algorithm

Let us consider segmenting the word "whereby" using Viterbi decoding:



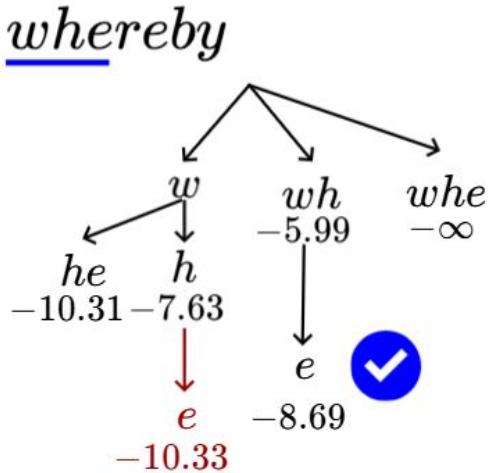
At this position, the possible segmentations of the slice "wh" are (w,h) and (wh)

Compute the log-likelihood for both and output the best one.

Token	$\log(p(x))$
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

SentencePiece Tokenizer- Algorithm

Let us consider segmenting the word "whereby" using Viterbi decoding:

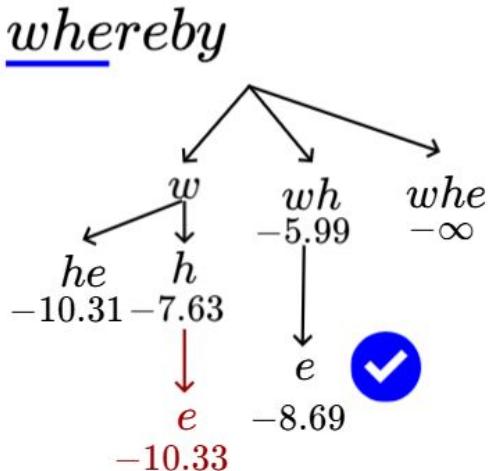


- We do not need to compute the likelihood of (w,h,e) as we already ruled out (w,h) to (wh).
- We display it for completeness w h e whe $-\infty$ Of these, (wh,e) is the best segmentation that maximizes the likelihood.

Token	$\log(p(x))$
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

SentencePiece Tokenizer- Algorithm

Let us consider segmenting the word "whereby" using Viterbi decoding:

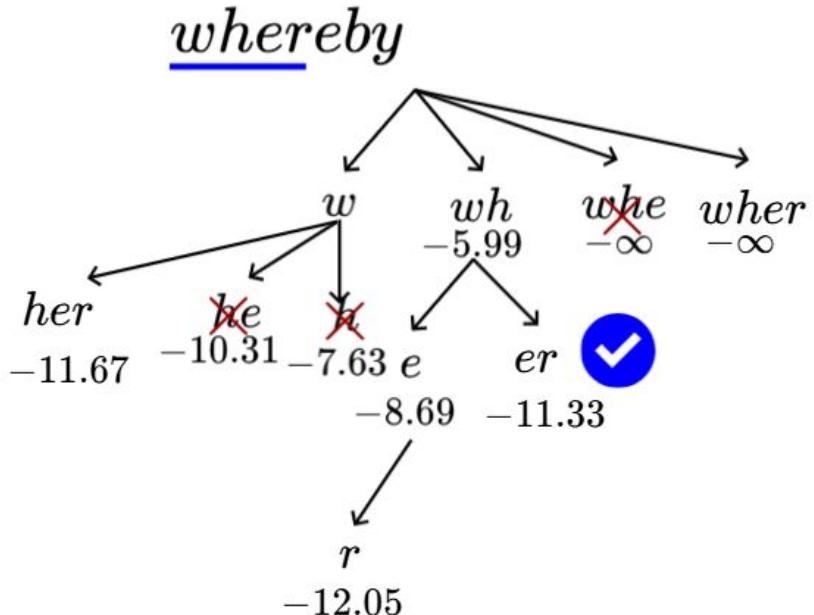


- We do not need to compute the likelihood of (w,h,e) as we already ruled out (w,h) to (wh).
- We display it for completeness w h e whe $-\infty$ Of these, (wh,e) is the best segmentation that maximizes the likelihood.

Token	$\log(p(x))$
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

SentencePiece Tokenizer- Algorithm

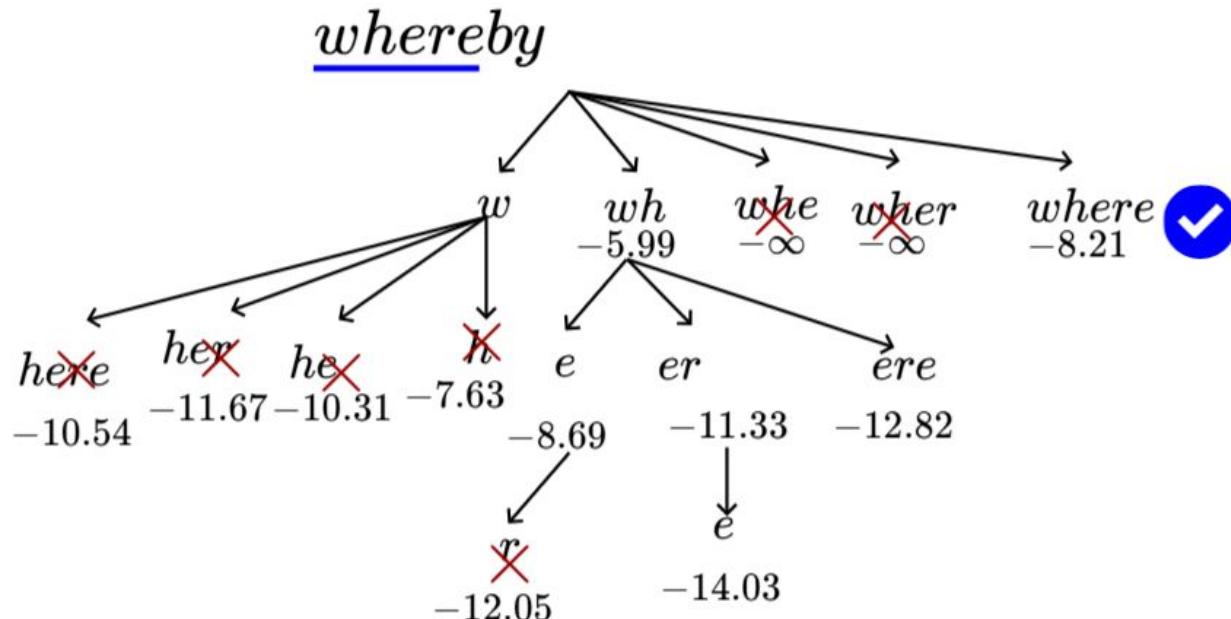
Let us consider segmenting the word "whereby" using Viterbi decoding:



Token	$\log(p(x))$
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

SentencePiece Tokenizer- Algorithm

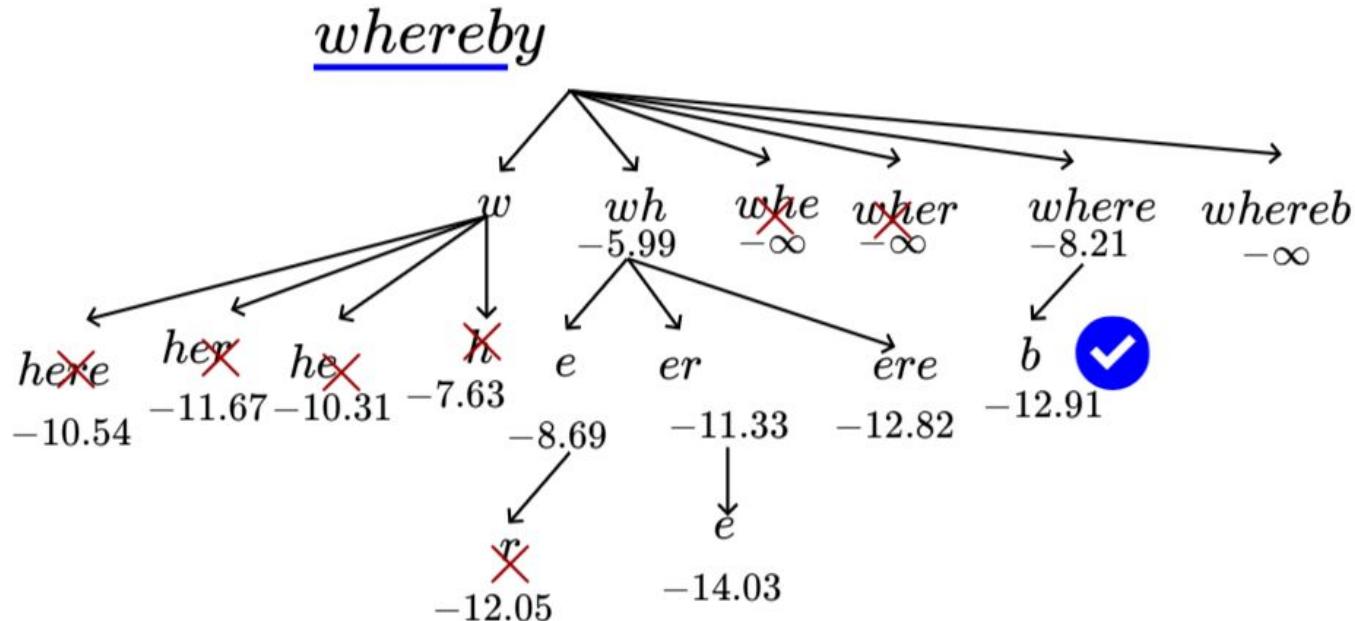
Let us consider segmenting the word "whereby" using Viterbi decoding:



Token	$\log(p(x))$
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

SentencePiece Tokenizer- Algorithm

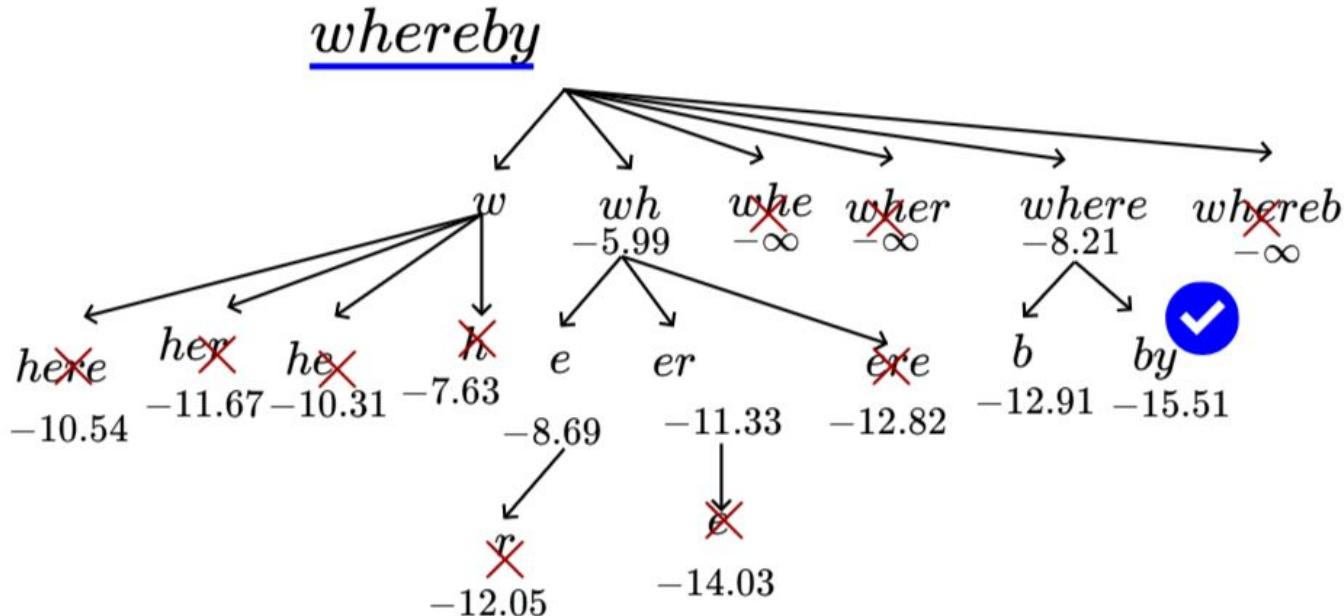
Let us consider segmenting the word "whereby" using Viterbi decoding:



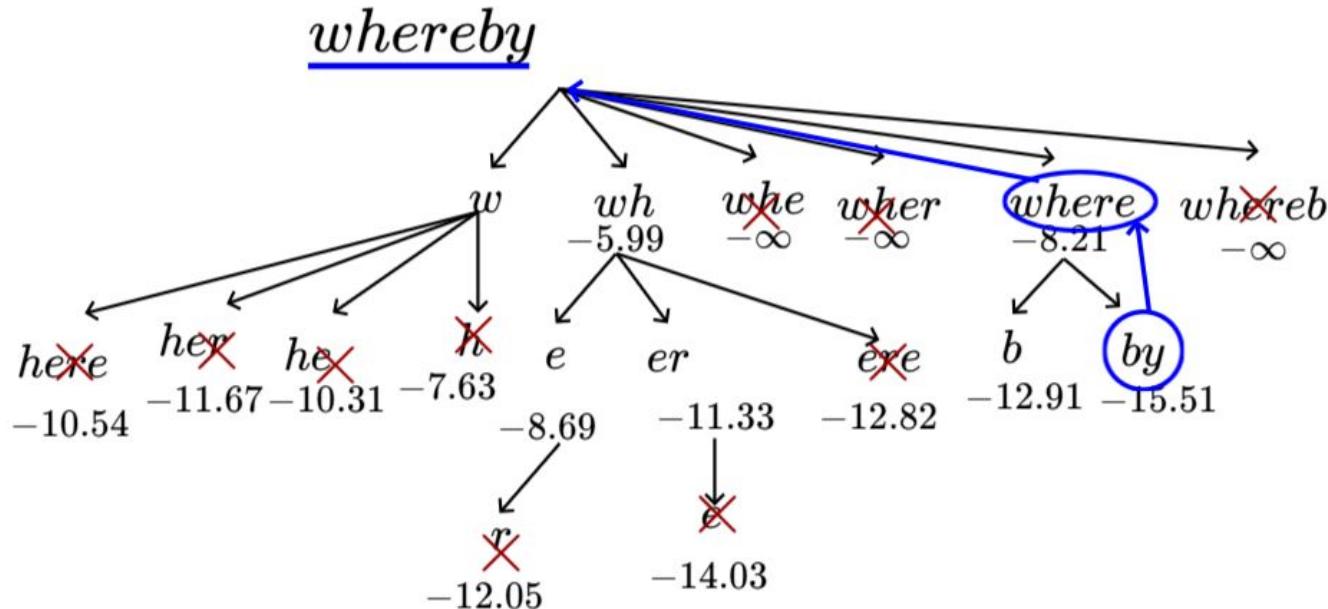
Token	$\log(p(x))$
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

SentencePiece Tokenizer- Algorithm

Let us consider segmenting the word "whereby" using Viterbi decoding:



SentencePiece Tokenizer- Backtrack



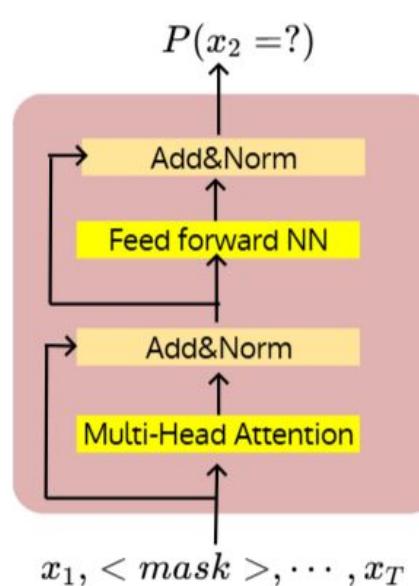
Token	$\log(p(x))$
b	-4.7
e	-2.7
h	-3.34
r	-3.36
w	-4.29
wh	-5.99
er	-5.34
where	-8.21
by	-7.34
he	-6.02
ere	-6.83
here	-7.84
her	-7.38
re	-6.13

The best segmentation of the word "whereby" that maximizes the likelihood is "where,by"

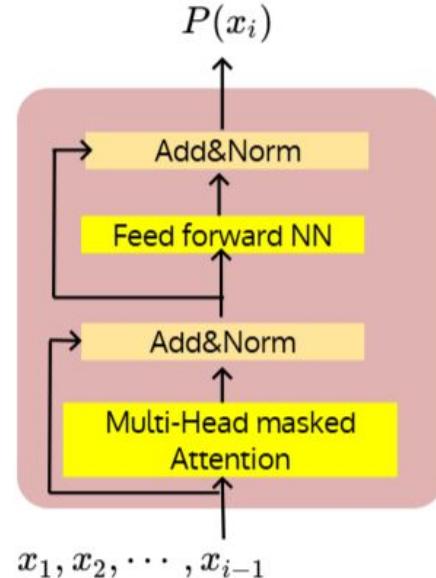
BART

Encoder-Decoder model: BART

- So far, we have learned two approaches with two different objectives for language modelling
- **BERT** is good at comprehension tasks like questions answering because of its bidirectional nature
- **GPT** is good at text generation because of its unidirectional nature.



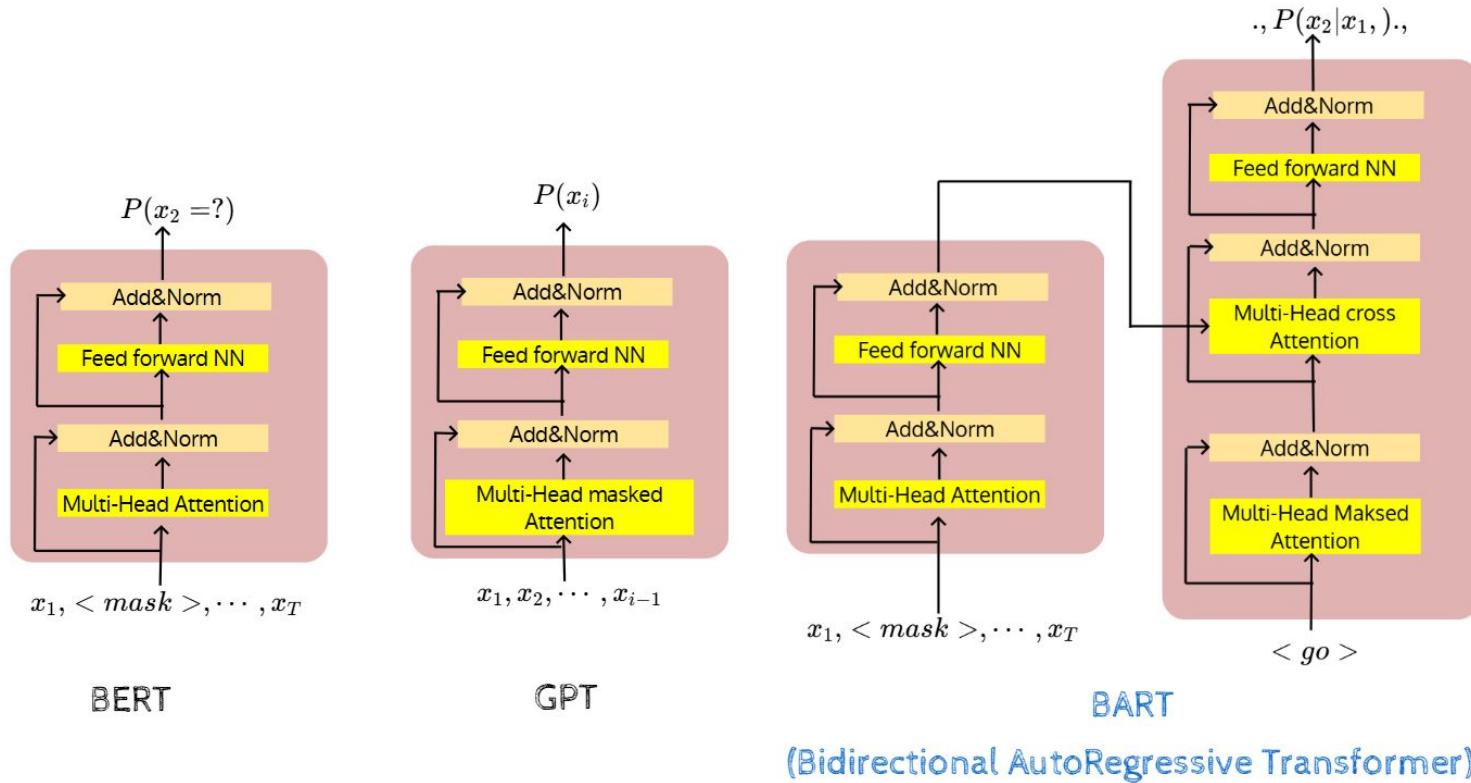
BERT and its variations



GPT and its variations

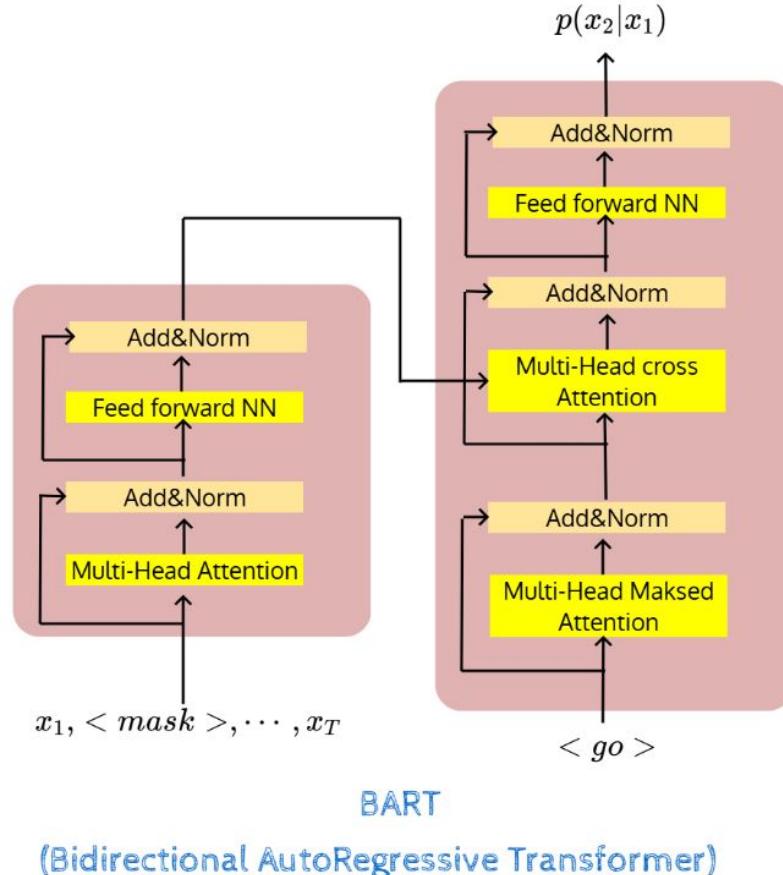
Why don't we take the best of both worlds?

Encoder-Decoder model: BART



Encoder-Decoder model: BART

- A vanilla transformer architecture with GELU activation function.
- The decoder predicts the entire sequence and computes the loss (as in the case of GPT) over the entire sequence.
- The input sequence is corrupted in multiple ways: random masking, token deletion, document rotation and so on
- The encoder takes in the corrupted sequence
- One can think of this as a denoising-autoencoder.

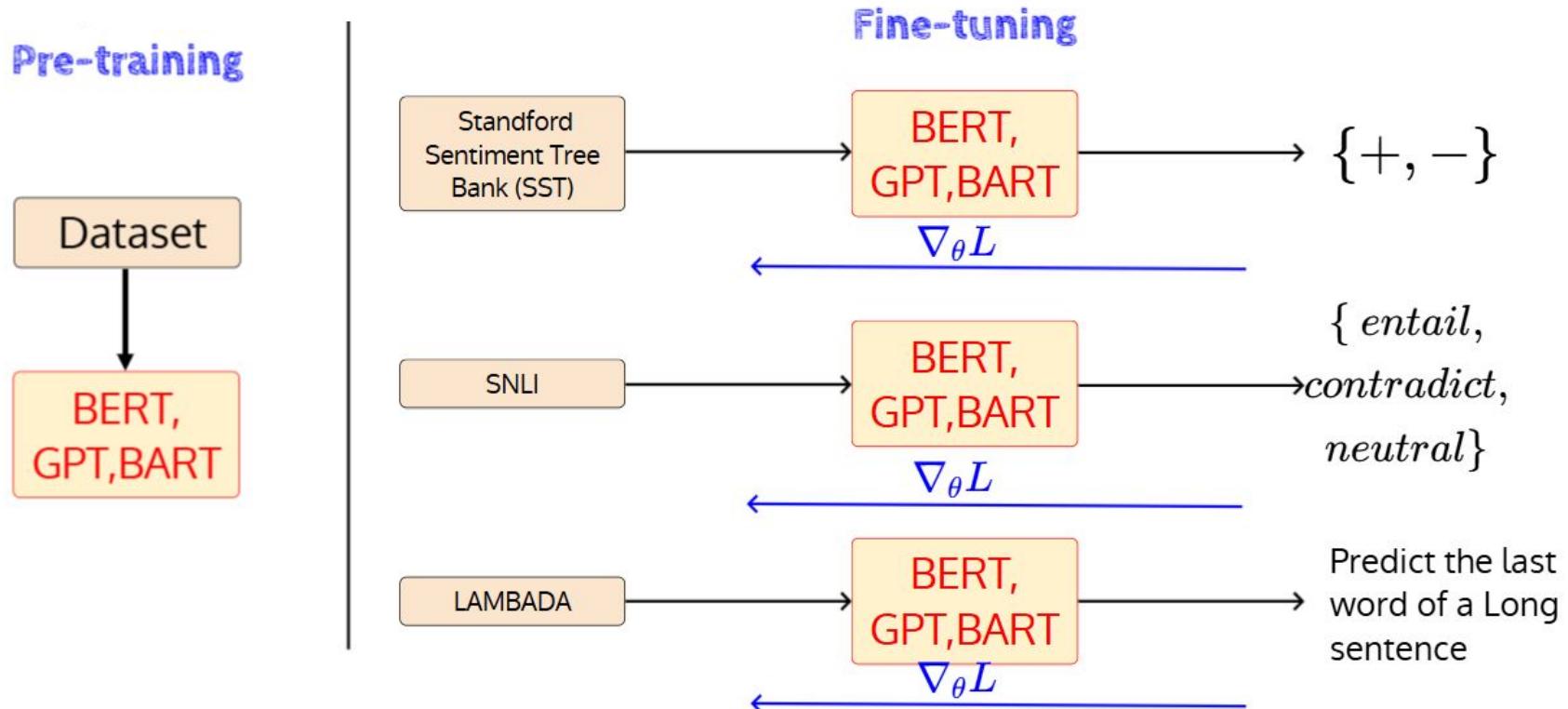


Prompting LLM

Pre-training and Fine tuning of LLMs

- Using any of these pre-trained models for downstream tasks requires us to independently fine-tune the parameters of the model for each task (objective might differ for each downstream task)
- That is, we make a copy of the pre-trained model for each task and fine-tune it on the dataset specific to that task

Pre-training and Fine tuning of LLMs



Pre-training and Fine tuning of LLMs

- However, fine-tuning large models is costly, as it still requires thousands of samples to obtain good performance in a downstream task.
- Some tasks may not have enough labelled samples
- Moreover, this is not how humans adapt to different tasks once they understand the language.
- We can give them a book to read and "**prompt**" them to summarize it or find an answer for a specific question.
- That is, we do not need "**supervised fine-tuning**" at all (for most of the tasks)
- In a nutshell, we want a single model that learns to do **multiple tasks** with zero to a few examples (instead of thousands)!

Pre-training and Fine tuning of LLMs

- Can we adapt a pre-trained language model for downstream tasks without any explicit supervision (called zero-shot transfer)?
 - Zero-shot (0 data point for fine tuning)
 - Few-shot (10-50 data points for fine tuning)
 - Full fine tuning (large data points, say 10000 data points for fine tuning)

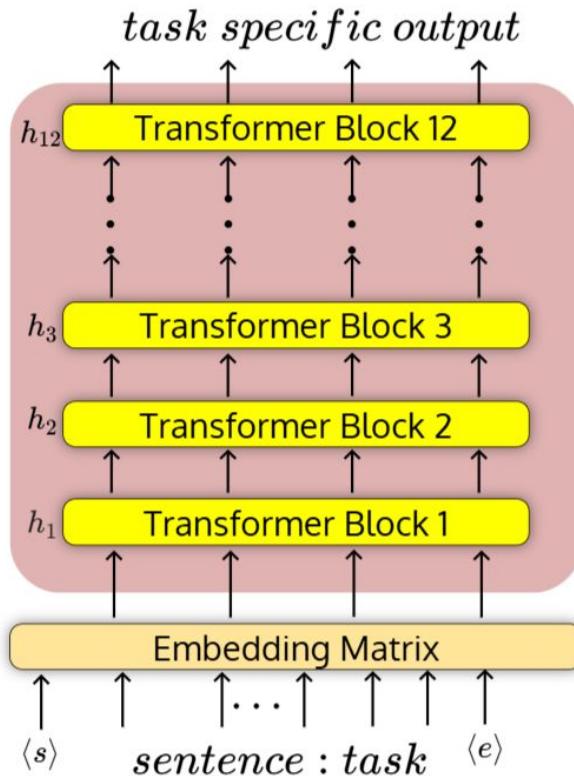
Pre-training and Fine tuning of LLMs

- Can we adapt a pre-trained language model for downstream tasks without any explicit supervision (called zero-shot transfer)?
- Yes, with a simple tweak to the input text!
- Note that, the prompts (or instructions) are words. Therefore, we just need to change the single task (LM) formulation

$$P(\text{output} | \text{input})$$

To multi task

$$P(\text{output} | \text{input}, \text{task})$$

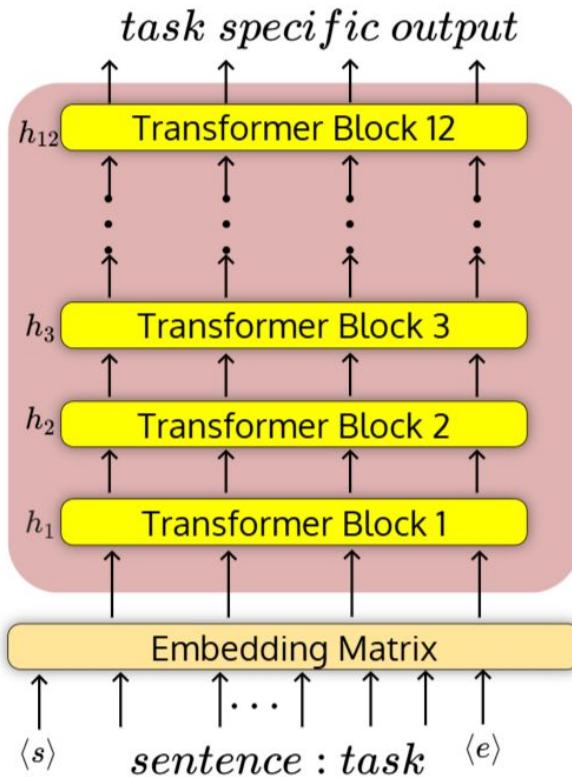


Pre-training and Fine tuning of LLMs

For multi task

$$P(\text{output} | \text{input}, \text{task})$$

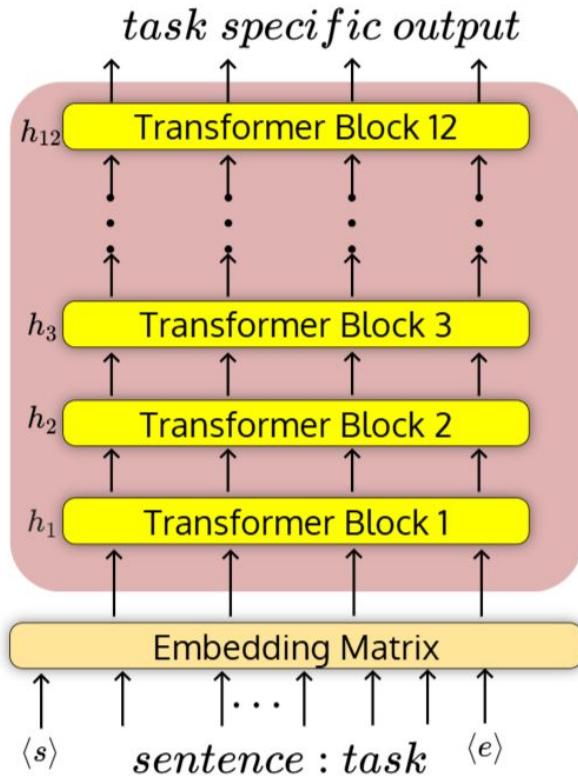
- Where the **task** is just an instruction in plain words that is prepended (appended) to the input sequence during inference
- Surprisingly, this induces a model to output a task specific response for the same input.



Pre-training and Fine tuning of LLMs

For example,

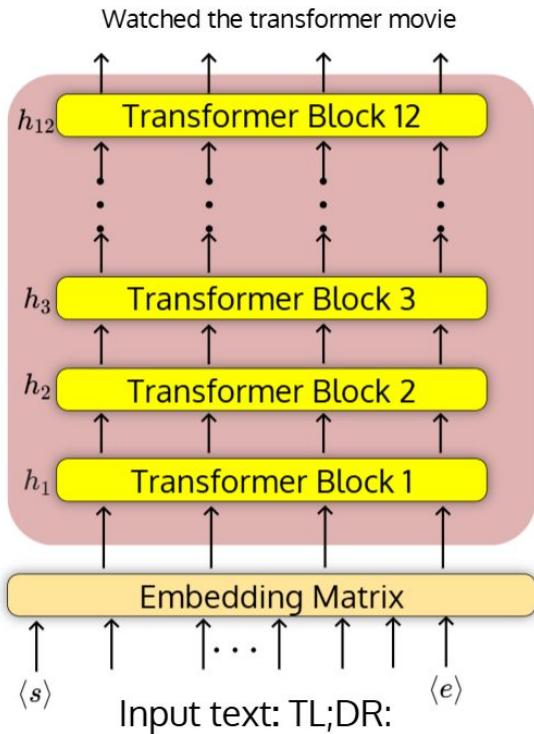
- **Input text:** I enjoyed watching the movie transformer along with my



Pre-training and Fine tuning of LLMs

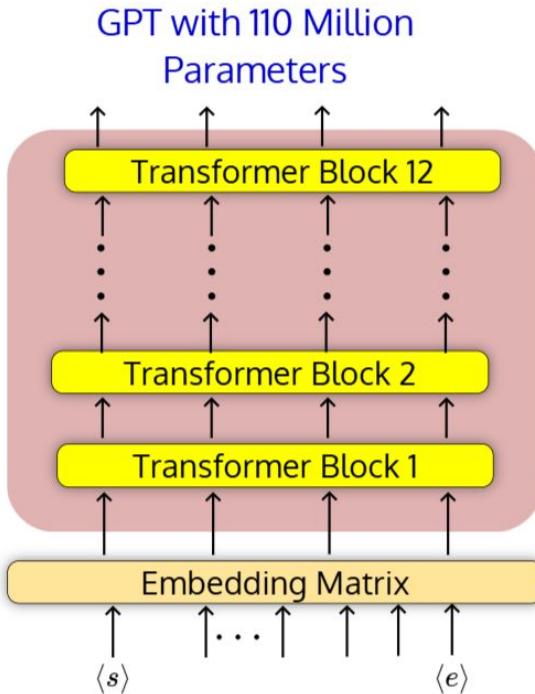
For example,

- **Input text:** I enjoyed watching the movie transformer along with my
- **Task:** summarize (or TL;DR:)



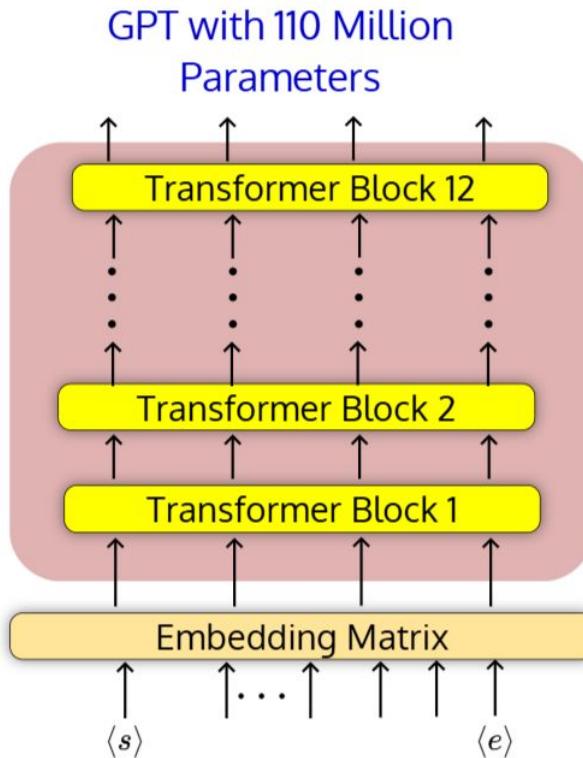
Zero shot transfer

To get a good performance from zero/few shot/s, we need to scale up both the model size and the data size.

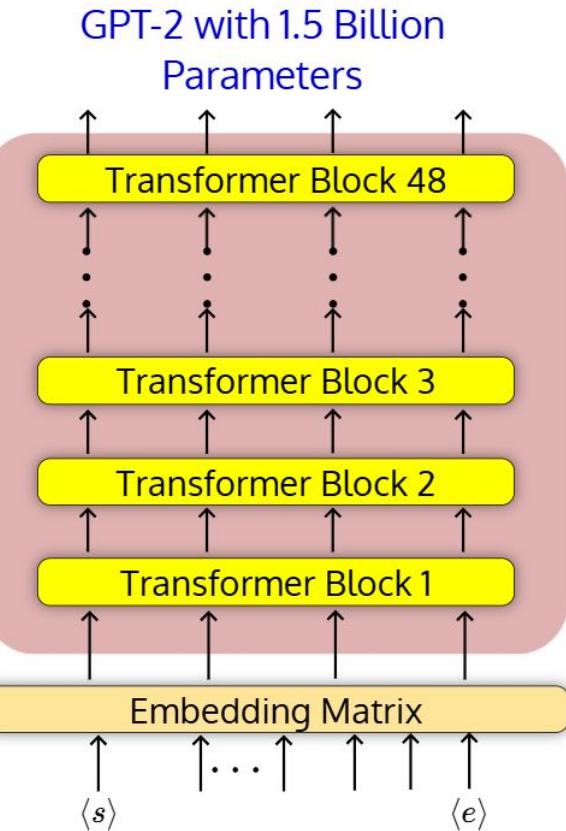


GPT-2

To get a good performance from zero/few shot/s, we need to scale up both the model size and the data size.



Layers: $4X$
Parameters: $10X$



GPT-2

- A new dataset called [WebText](#) was created by crawling outbound links (with at least 3 karma) from Reddit, excluding links to Wikipedia.
- This is to ensure the diversity and quality of data
- What about translation tasks?
- Data of size 40 GB (about 8 million webpages)
- Uses [Byte-Level BPE](#) tokenizer
- Vocabulary with 50,257 tokens

"I'm not the cleverest man in the world, but like they say in French: **Je ne suis pas un imbecile** [**I'm not a fool**].

In a now-deleted post from Aug. 16, Soheil Eid, Tory candidate in the riding of Joliette, wrote in French: "**Mentez mentez, il en restera toujours quelque chose**," which translates as, "**Lie lie and something will always remain**."

"I hate the word '**perfume**','" Burr says. 'It's somewhat better in French: '**parfum**'.'

If listened carefully at 29:55, a conversation can be heard between two guys in French: "**-Comment on fait pour aller de l'autre côté? -Quel autre côté?**", which means "**- How do you get to the other side? - What side?**".

If this sounds like a bit of a stretch, consider this question in French: **As-tu aller au cinéma?**, or **Did you go to the movies?**, which literally translates as Have-you to go to movies/theater?

"Brevet Sans Garantie Du Gouvernement", translated to English: "**Patented without government warranty**".

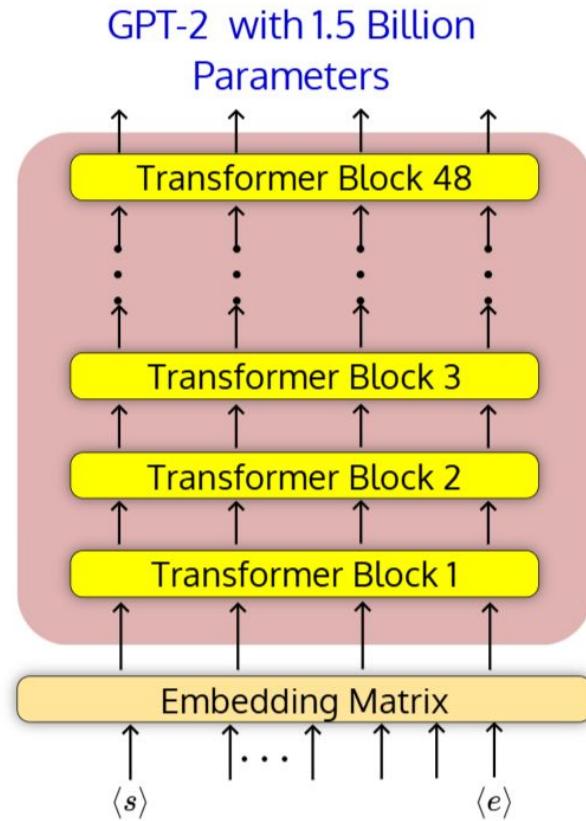
naturally occurring demonstrations of English to French and French to English translation found throughout the *WebText* training set

GPT-2

- GPT with 4 variations in architecture

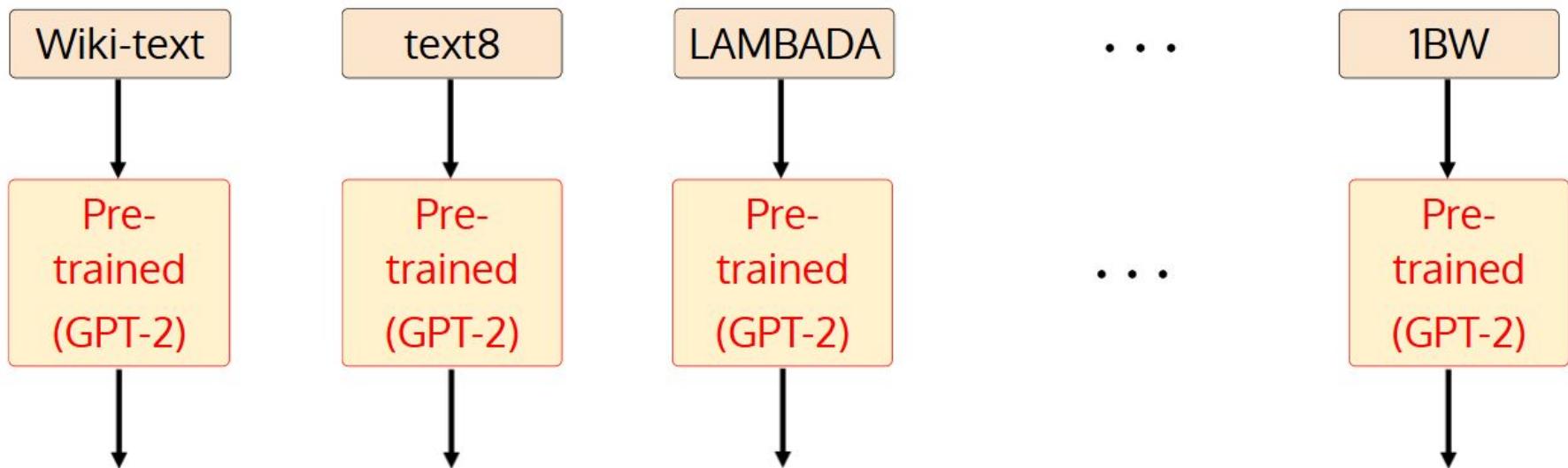
Parameters	Layers	d_model
117 M	12	768
345 M	24	1024
762 M	36	1280
1542	48	1600

- Context window size: 1024
- Batch size: 512
- **Training objective :** Causal Language Modeling (CLM)



GPT-2: Performance on Zero-shot domain transfer

- Let's take the pre-trained (using WebText) GPT-2 large (1.5 B) model
- Measure the performance of the **pre-trained model** on various **benchmarking** datasets across domains such as LM, translation,summarization, reading comprehension..



GPT-2: Performance on Zero-shot domain transfer

Language Modelling

Wiki-text

Pre-trained
(GPT-2)

Measure
Perplexity

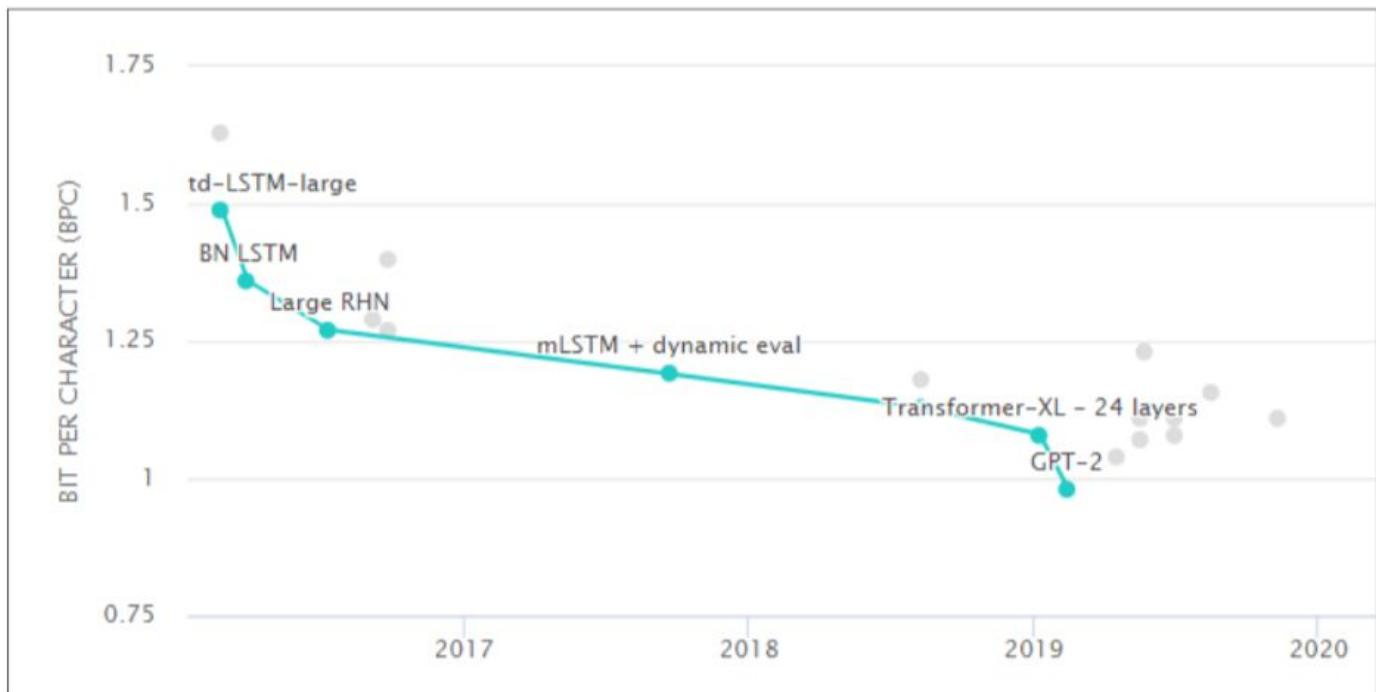


source: Paperswithcode

The model performance increases significantly upon fine-tuning

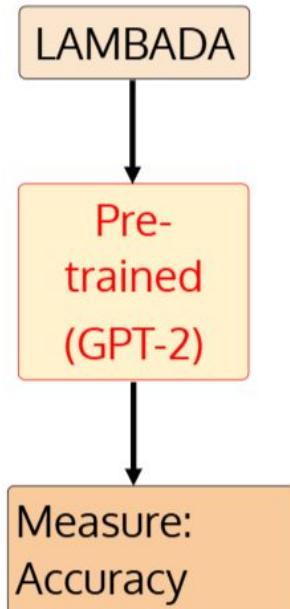
GPT-2: Performance on Zero-shot domain transfer

Character level language modelling



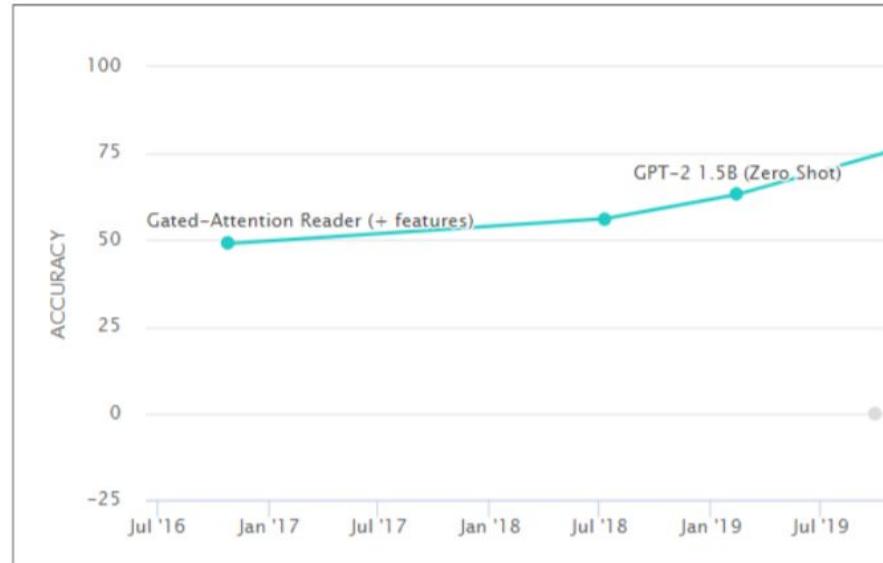
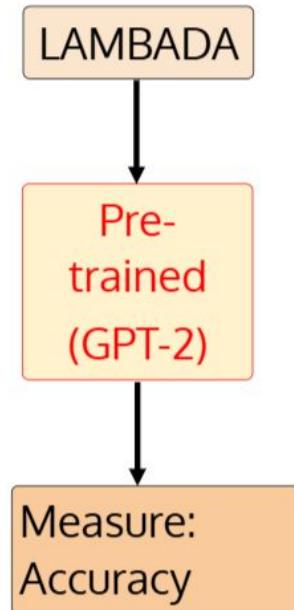
GPT-2: Performance on Zero-shot domain transfer

Open-ended cloze task which consists of about 10,000 passages from BooksCorpus where a missing target word is to be predicted in the last sentence of each passage



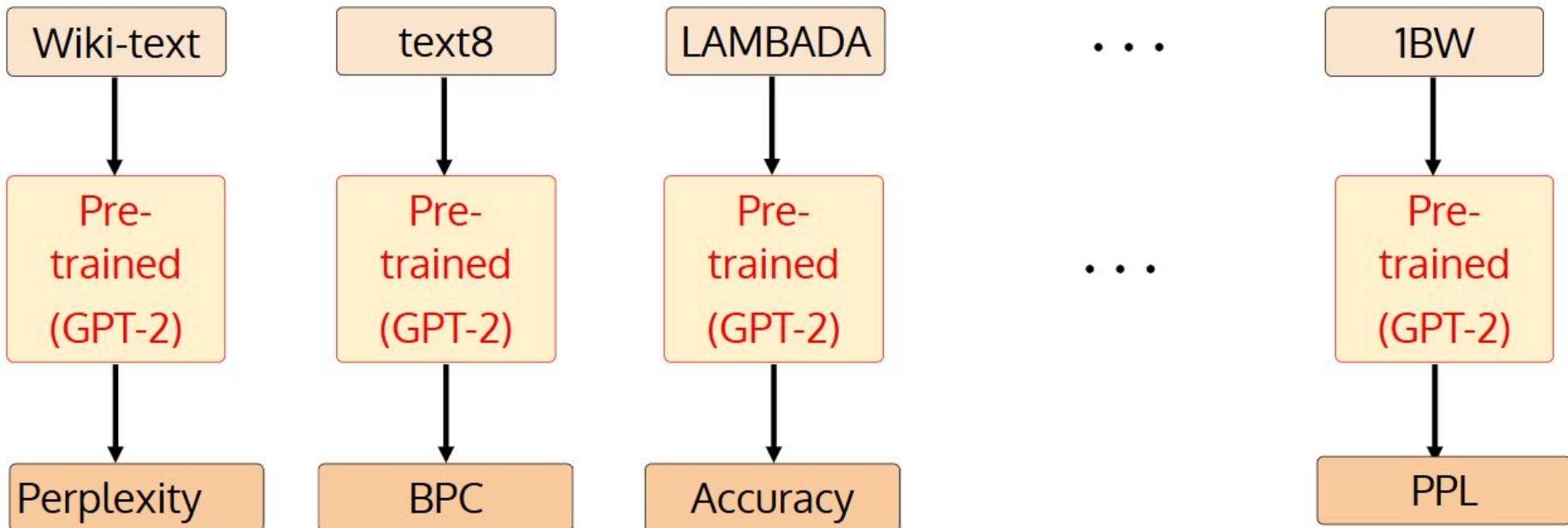
GPT-2: Performance on Zero-shot domain transfer

Open-ended cloze task which consists of about 10,000 passages from BooksCorpus where a missing target word is to be predicted in the last sentence of each passage.



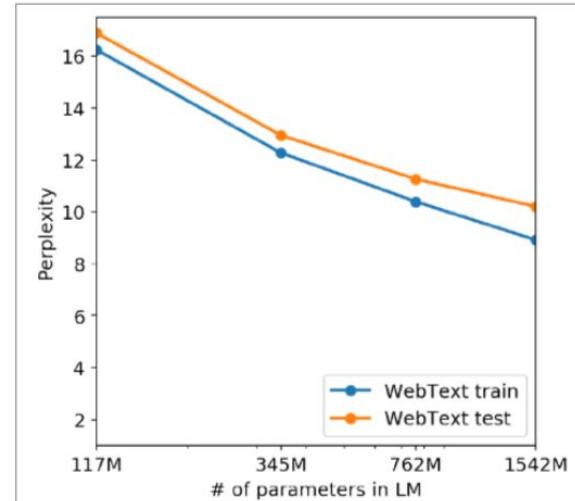
GPT-2: Performance on Zero-shot domain transfer

It outperforms other language models trained on domains like Wiki and books, without using these domain-specific training datasets



Promising direction

- However, the performance of the model on tasks like question answering, reading comprehension, summarization, and translation is far from SOTA models specifically fine-tuned for these tasks.
- Moreover, the 1.5 Billions parameters model still underfits the training set!
- From the graph on the left, we can observe the trend that scaling the model and datasize further might give us some promising results on prompting.
- This work established that "Large Language models are unsupervised multitask learners.



Choices that affect LLM Performance

Choices that affect performance

Let's compare the performance of GPT (110M) and BERT on various tasks using full fine tuning approach

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	Average
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
BERT _{BASE}	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	92.7	94.9	60.5	86.5	89.3	70.1	82.1

Why is one Model performing better than the other (in all or some tasks)?

Choices that affect performance

1. Is it because of the size of the data set?

Books (0.7B) Vs BookCorpus (1B)

&

WiKiPedia (2.5B)

Choices that affect performance

1. Is it because of the size of the data set?
2. Is it because of the pre-training objective?

Books (0.7B) Vs BookCorpus (1B)

&

WiKiPedia (2.5B)

CLM Vs MLM Vs ____?

Choices that affect performance

1. Is it because of the size of the data set?
2. Is it because of the pre-training objective?

Books (0.7B) Vs BookCorpus (1B)

&

CLM Vs MLM Vs ____?

WiKiPedia (2.5B)

3. Is it because of the size of the model (number of parameters)?

GPT (117M) Vs BERT Large (336M)

Choices that affect performance

1. Is it because of the size of the data set?
2. Is it because of the pre-training objective?

Books (0.7B) Vs BookCorpus (1B)

&

WiKiPedia (2.5B)

CLM Vs MLM Vs ____?

3. Is it because of the size of the model (number of parameters)?
4. Is it because of training a model longer?

GPT (117M) Vs BERT Large (336M)

_____ Vs _____
T times 4T times

Choices that affect performance

1. Is it because of the size of the data set?
4. Is it because of training a model longer?

Books (0.7B) Vs BookCorpus (1B)

&

WiKiPedia (2.5B)

_____ Vs _____

T times

4T times

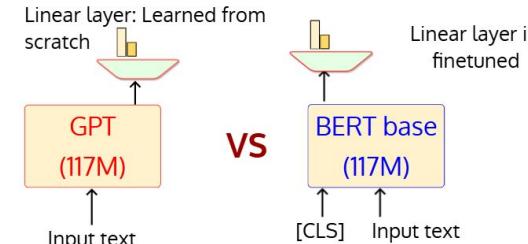
3. Is it because of the size of the model (number of parameters)?

GPT (117M) Vs BERT Large (336M)

2. Is it because of the pre-training objective?

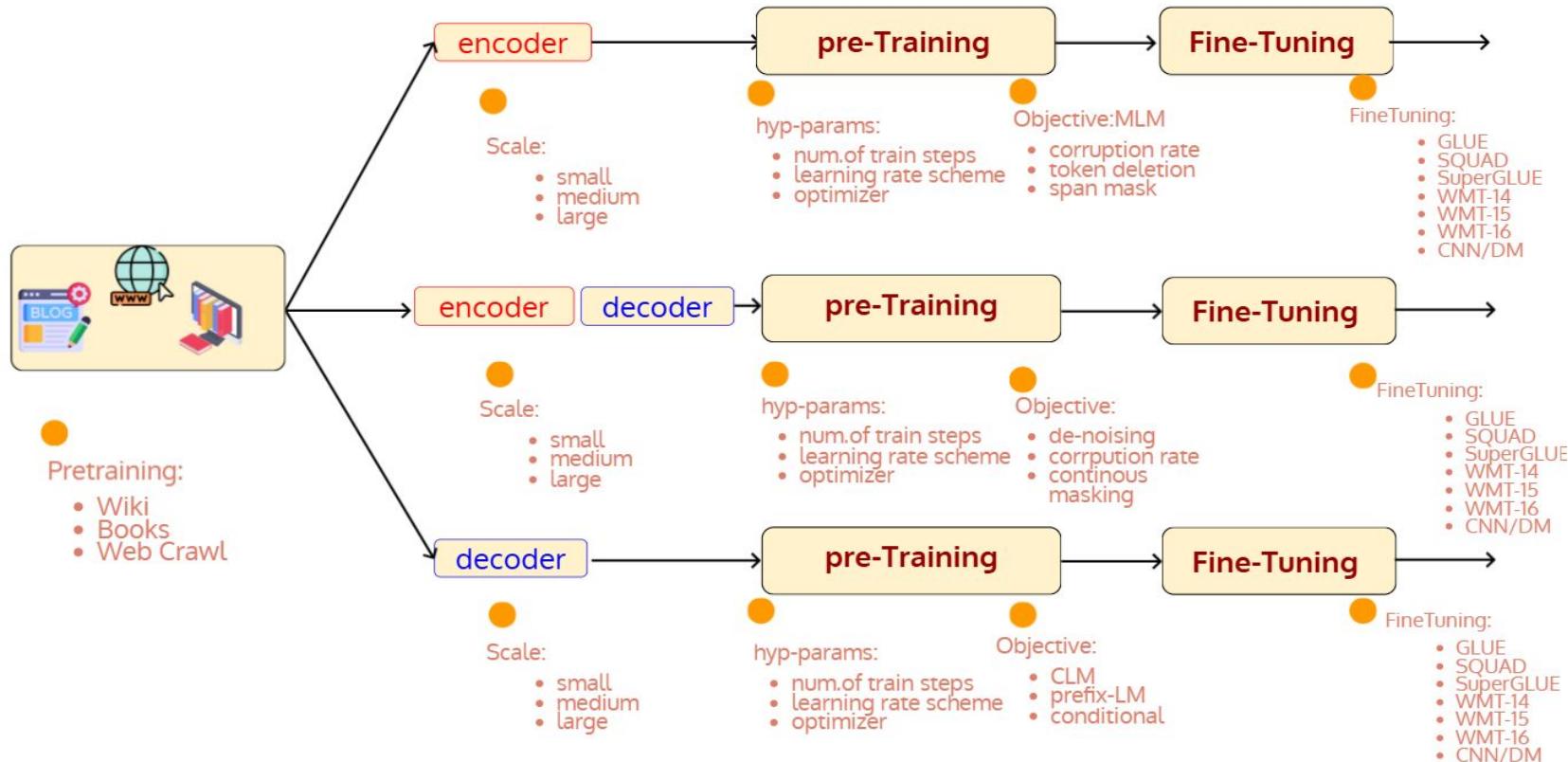
CLM Vs MLM Vs ____?

5. Is it because of the way one fine-tunes the model for downstream tasks?



Choices that affect performance

To answer these questions, we need to conduct extensive experiments by keeping one aspect of the pipeline constant and vary the other.

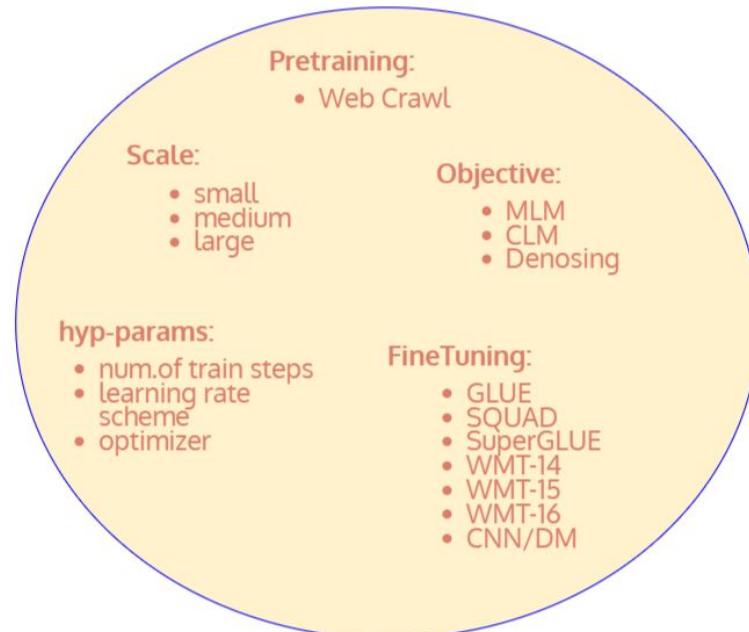


Choices that affect performance: Wishlist

- Can we have the **same objective** for both pre-training and fine-tuning stages?
- Can we have an architecture that uses both CLM and MLM like objectives?
- Can we use a pre-training dataset which is as large as possible (to understand the effect of size of unlabelled data)?

T5

- Compare a variety of different approaches on a diverse set of tasks while keeping as many factors fixed as possible.

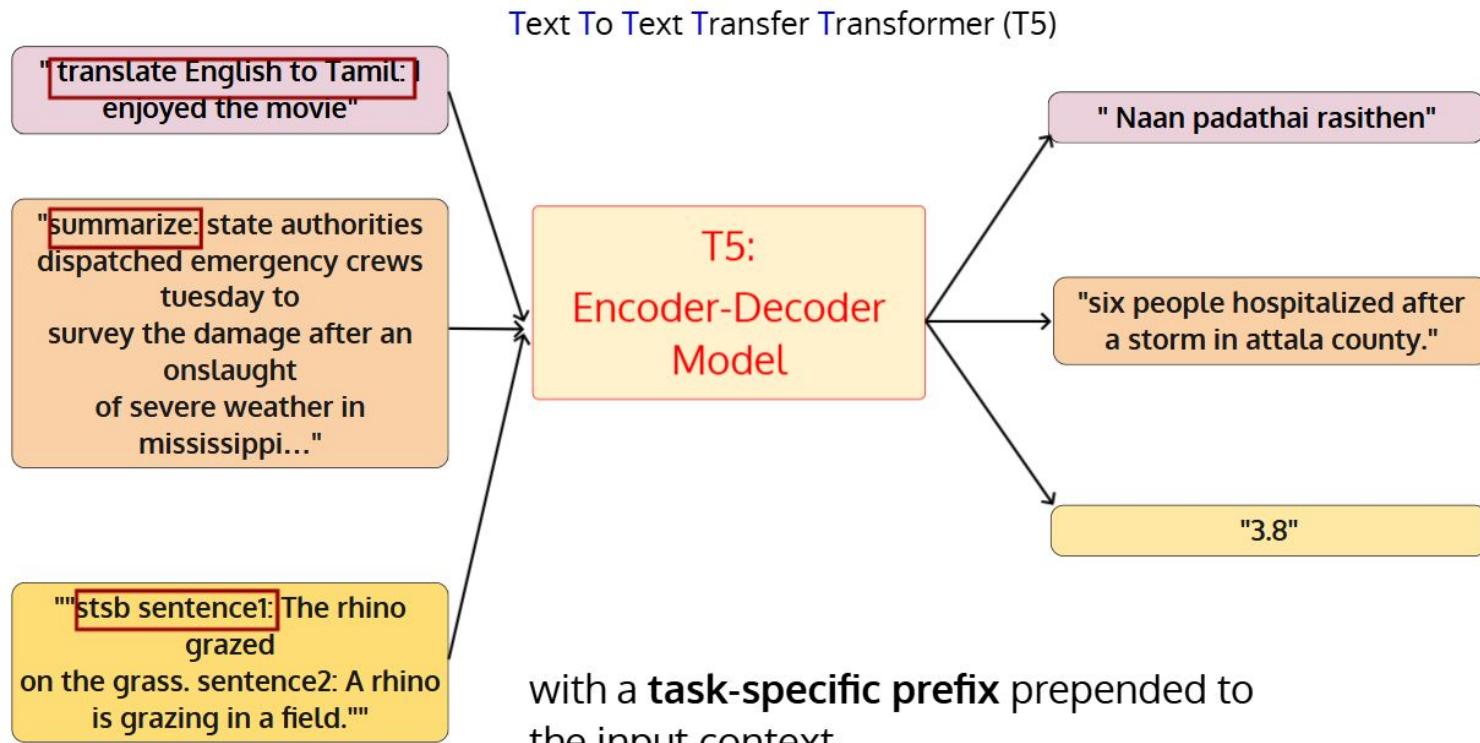


T5: Idea

- Formulate all NLP problems as "Text-in" and "Text-out".

T5: Idea

- Formulate all NLP problems as "Text-in" and "Text-out".



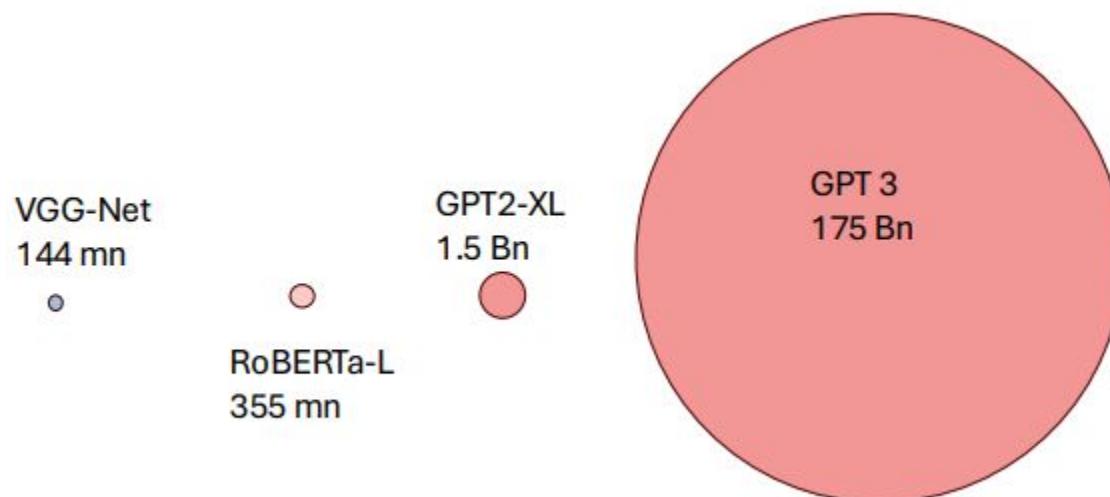
T5: Details

- Formulate all NLP problems as "Text-in" and "Text-out".
- Colossal Clean Crawled Corpus ([C4](#))
- Language: [Mainly](#) English
- Tokenizer: SentencePiece
- Vocab Size: 32,000
- Number of tokens: 156 Billion (About 52 times bigger than the dataset used for pertaining BERT)
- Dataset: 750 GB, 300M documents

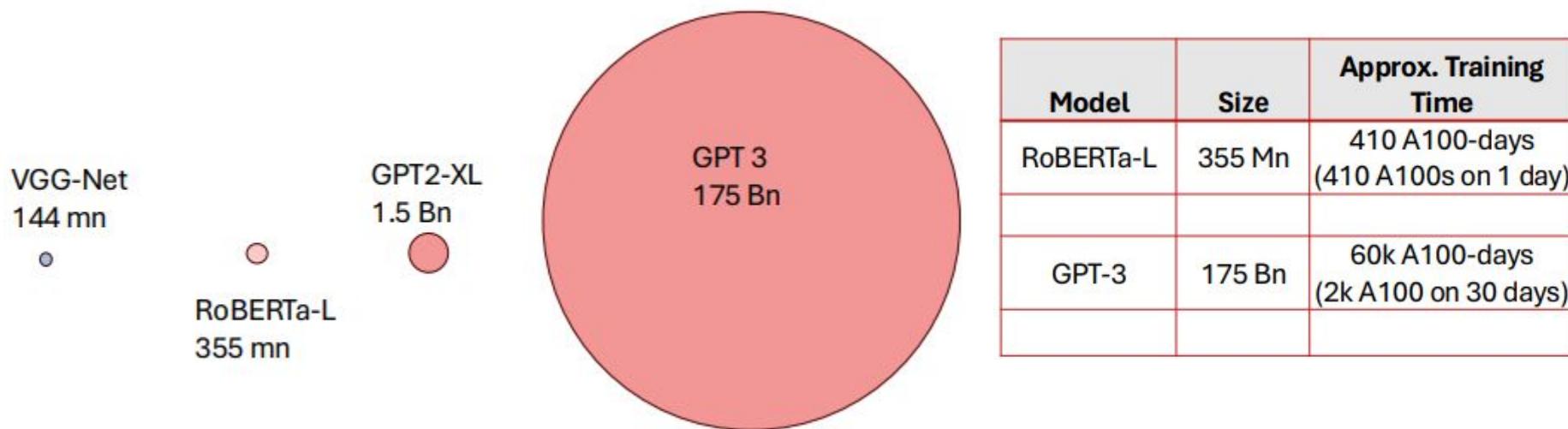
Mixture of Experts (MoE)

<https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mixture-of-experts>

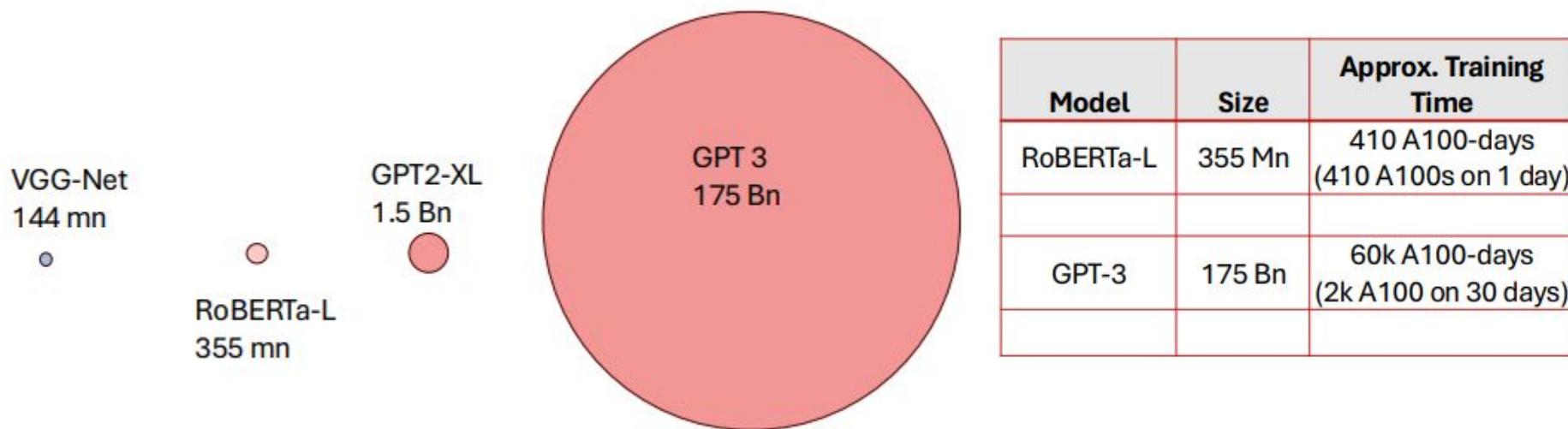
Motivation



Motivation

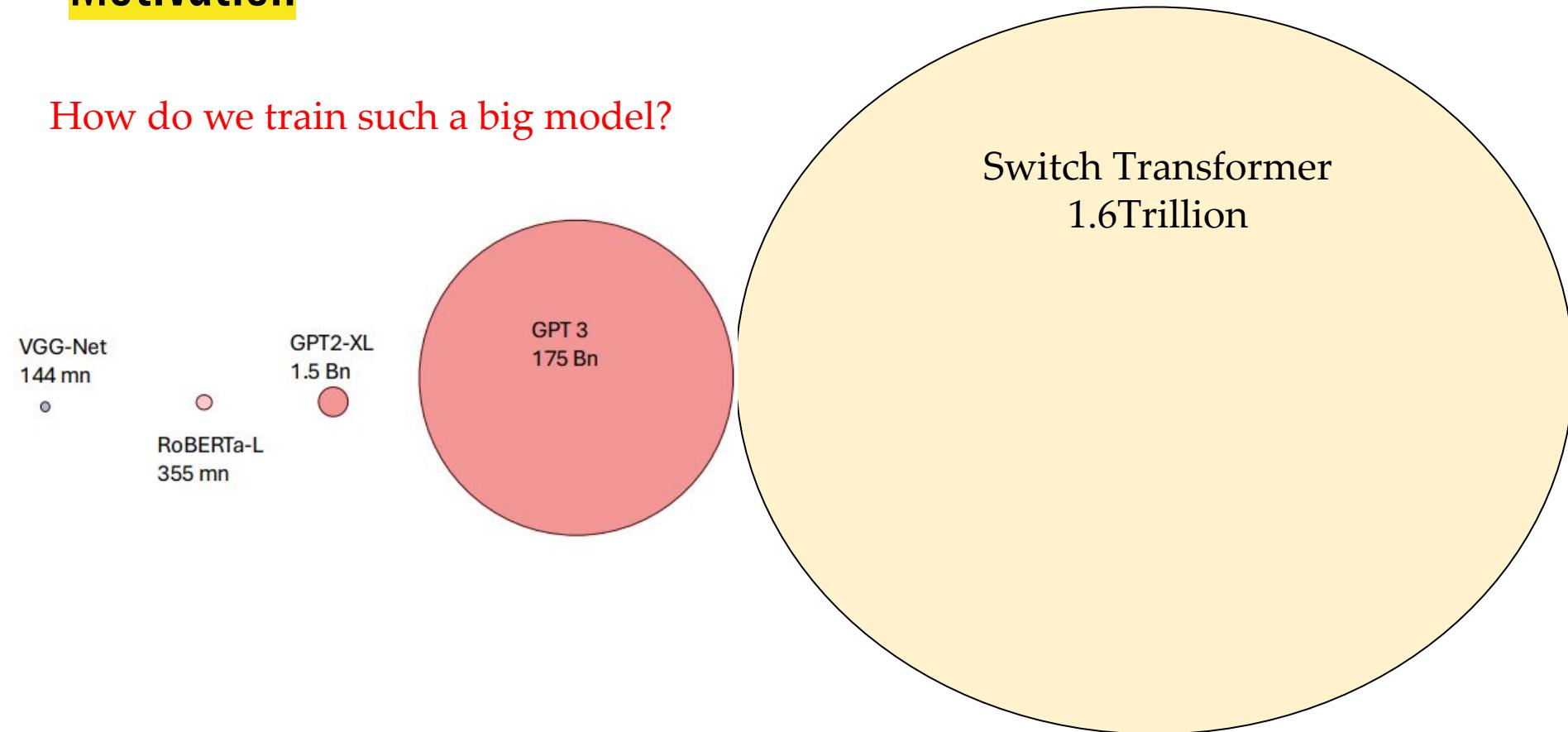


Motivation



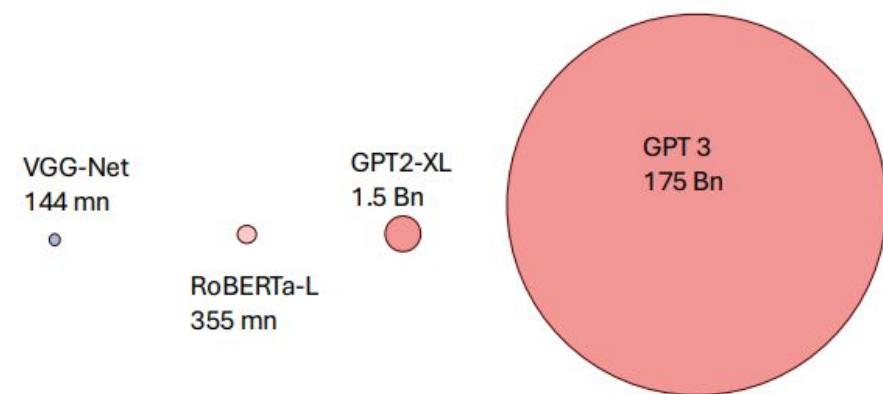
Motivation

How do we train such a big model?



Motivation

How do we train such a big model?

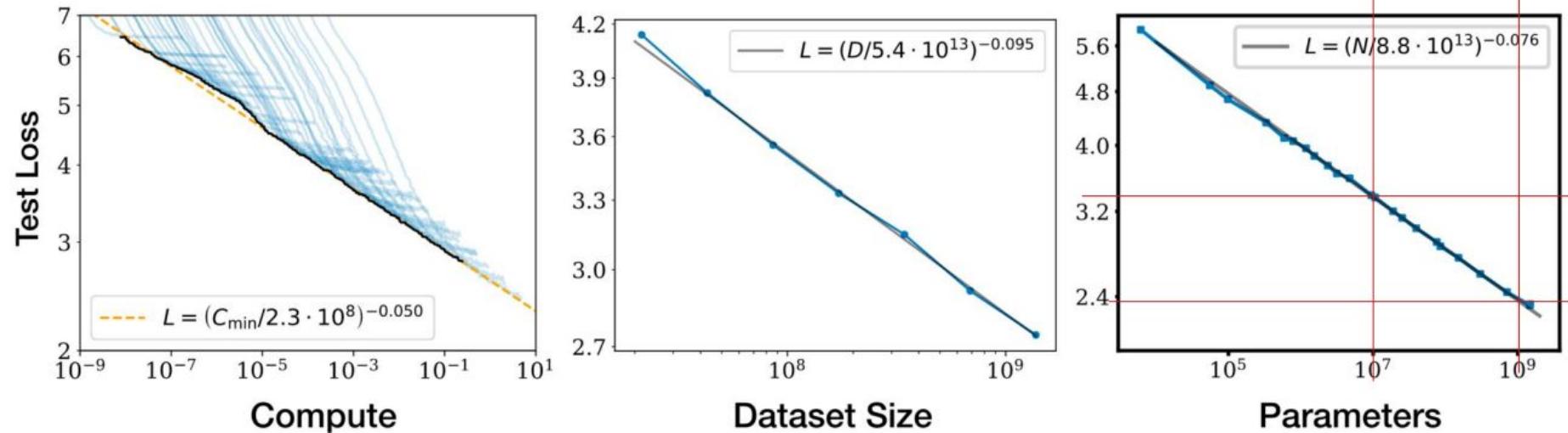


Switch Transformer
1.6Trillion

Model	Size	Approx. Training Time
RoBERTa-L	355 Mn	410 A100-days (410 A100s on 1 day)
GPT-3	175 Bn	60k A100-days (2k A100 in 1 Month)
	1.6T	540k A100-days (2k A100 in 9 Months!)

Why care about model size?

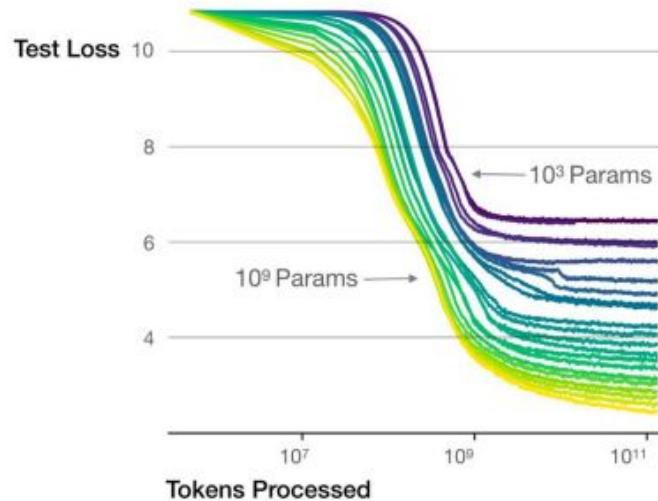
Neural Scaling laws: Performance improve smoothly as we increase the compute, dataset size, or the model size



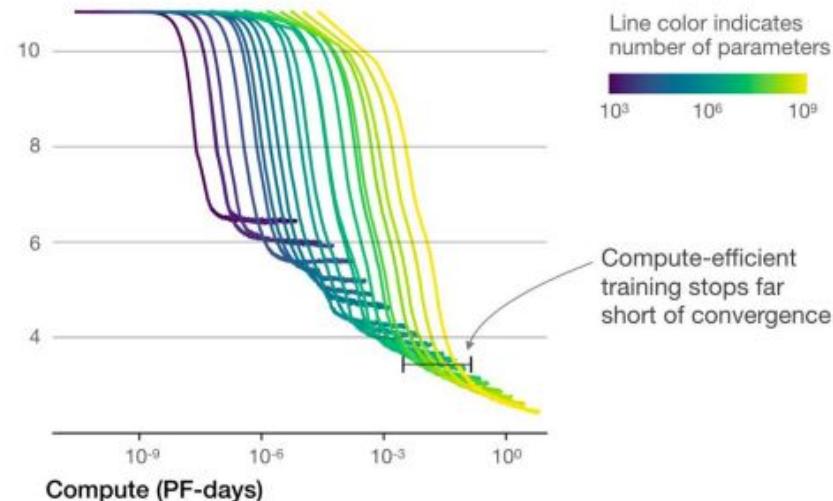
Why care about model size?

Neural Scaling laws: Large models are more sample efficient -- given a fixed computing budget, training a larger model for fewer steps is better than training a smaller model for more steps.

Larger models require **fewer samples** to reach the same performance

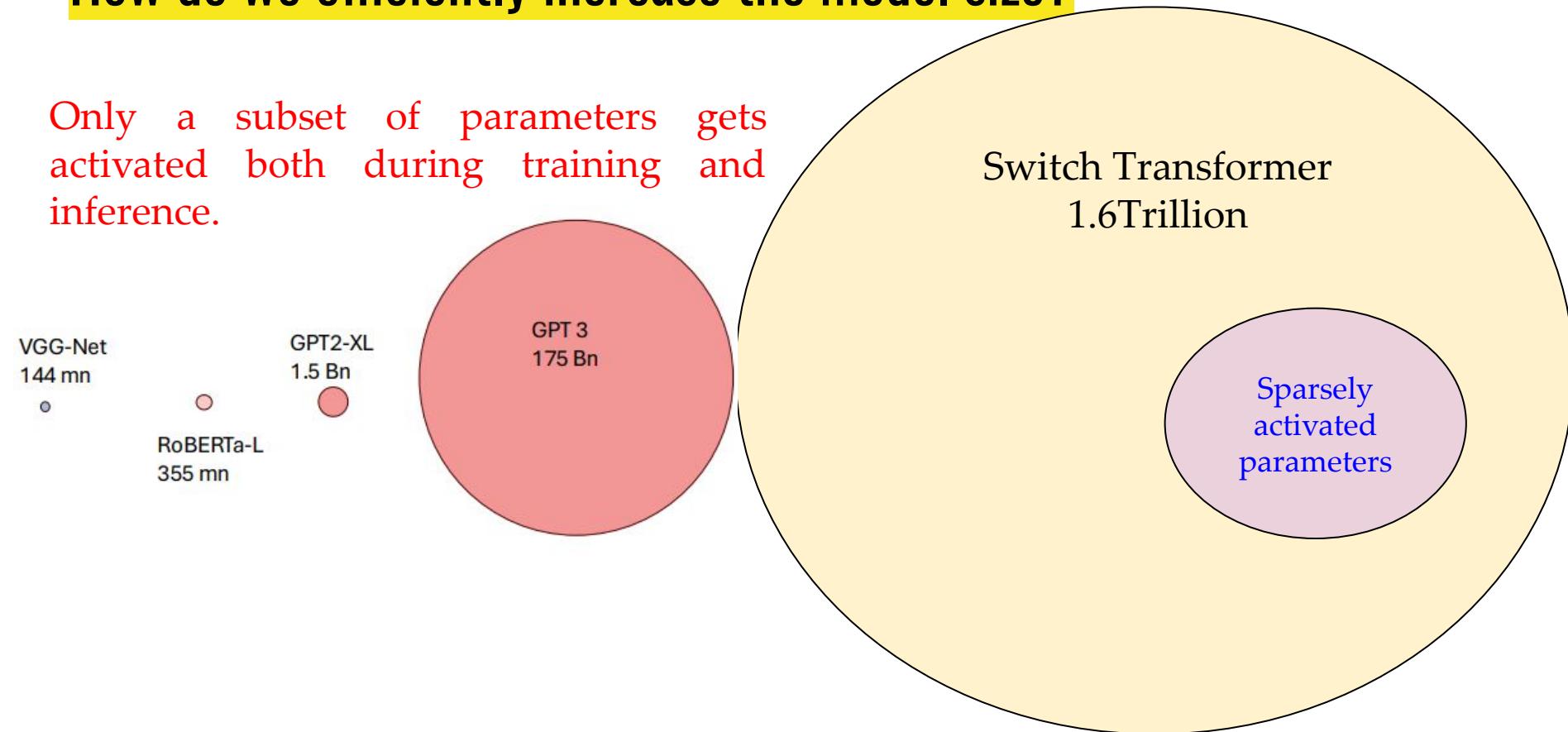


The optimal model size grows smoothly with the loss target and compute budget



How do we efficiently increase the model size?

Only a subset of parameters gets activated both during training and inference.



Mixture of Experts (MoE)

- Mixture of Experts (MoE) is a technique that uses many different sub-models (or “experts”) to improve the quality of LLMs.
- Two main components define a MoE:
 - **Experts** - Each FFNN layer now has a set of “experts” of which a subset can be chosen. These “experts” are typically FFNNs themselves.
 - **Router or gate network** - Determines which tokens are sent to which experts.
- Each expert is not an entire LLM but a submodel part of an LLM’s architecture.

Mixture of Experts (MoE)

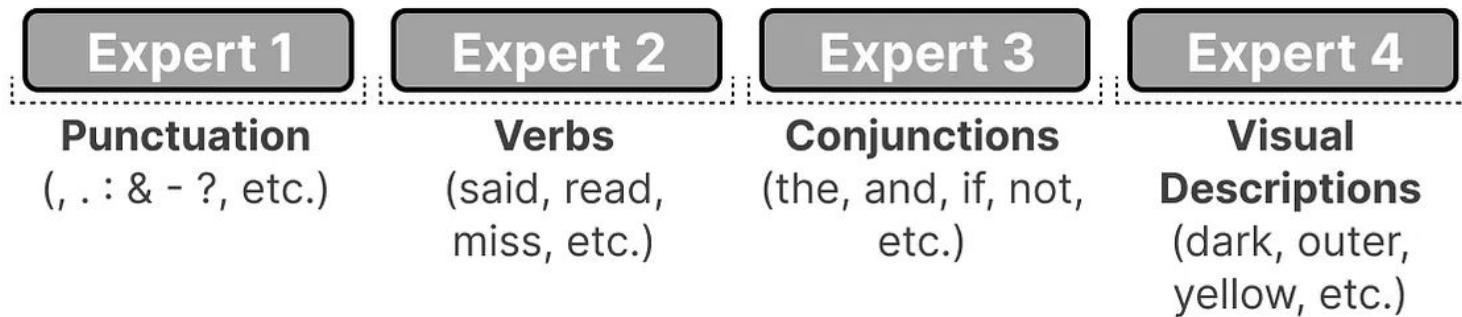
- In each layer of an LLM with an MoE, we find (somewhat specialized) experts:



Mixture of Experts (MoE)

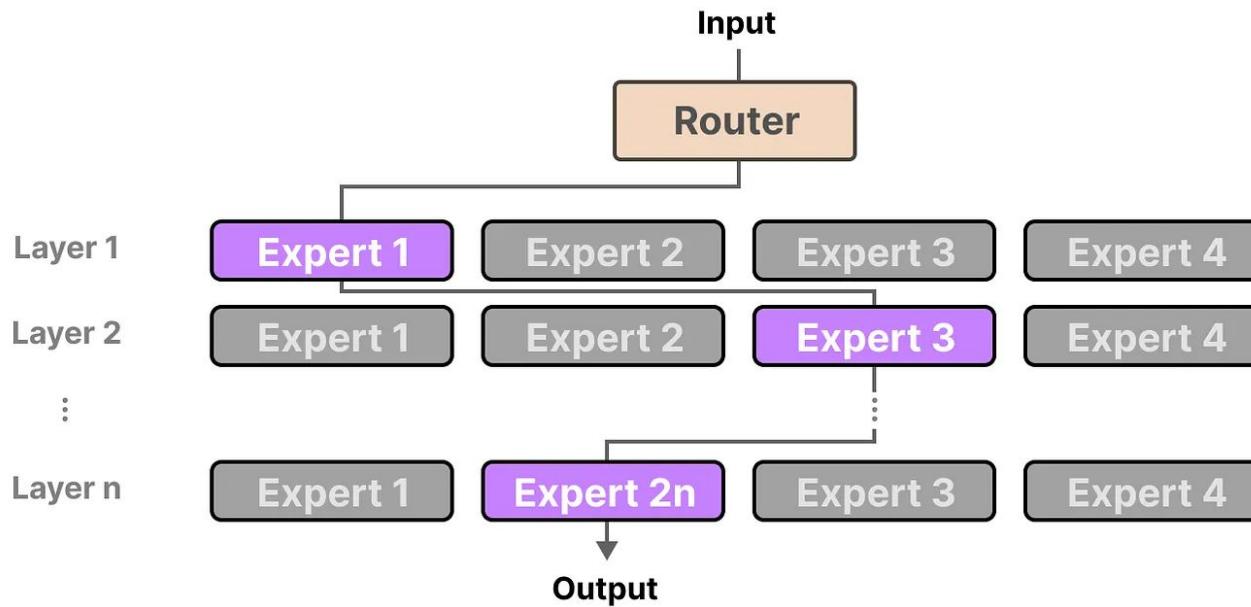
- An “expert” is not specialized in a specific domain like “Psychology” or “Biology”.
- At most, it learns syntactic information on a word level instead:

Layer 1



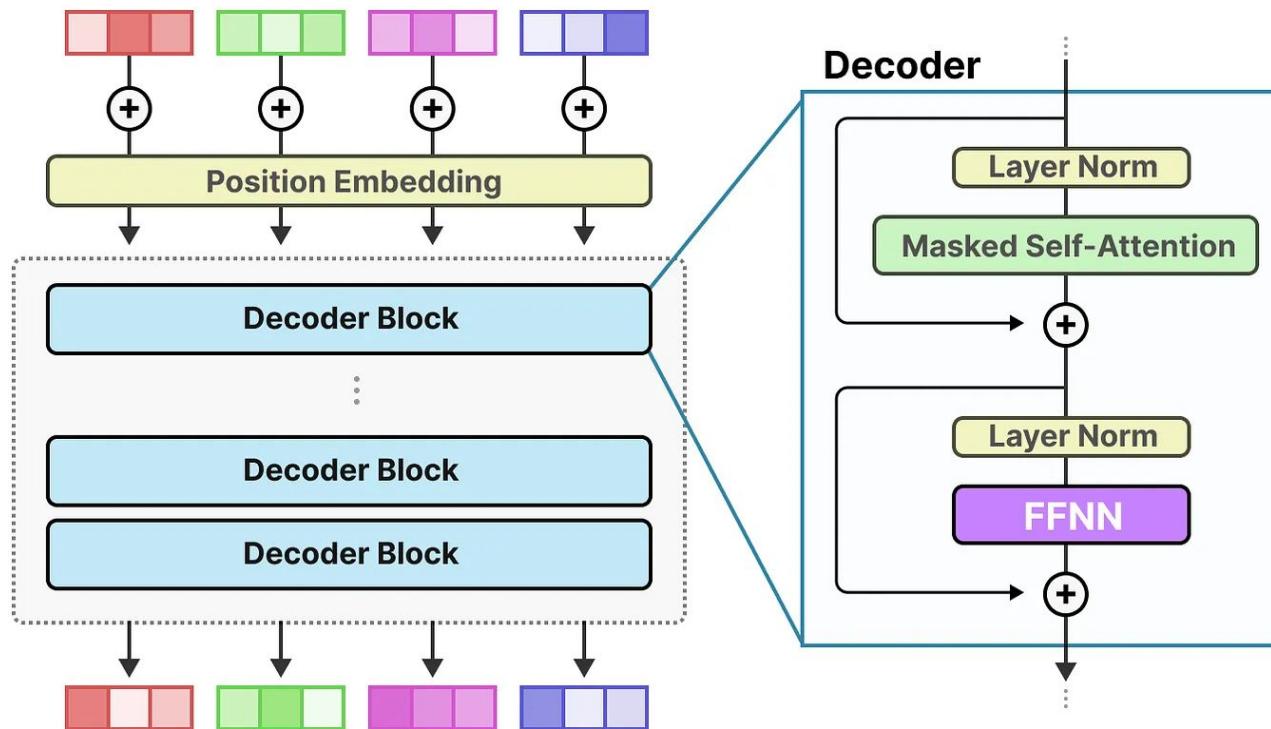
Mixture of Experts (MoE)

- More specifically, their expertise is in handling specific tokens in specific contexts. The *router* (gate network) selects the expert(s) best suited for a given input:



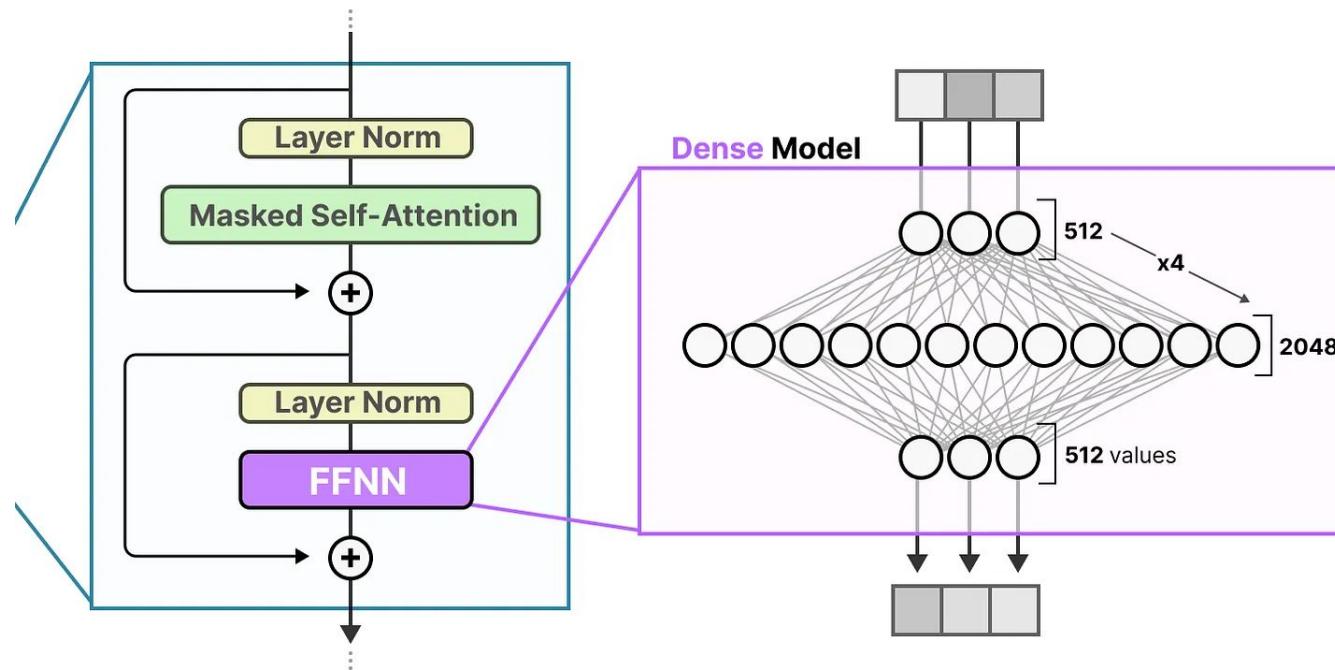
LLM: FFNN

- Mixture of Experts (MoE) all start from a relatively basic functionality of LLMs, namely the Feedforward Neural Network (FFNN).
- Remember that a standard decoder-only Transformer architecture has the FFNN applied after layer normalization:



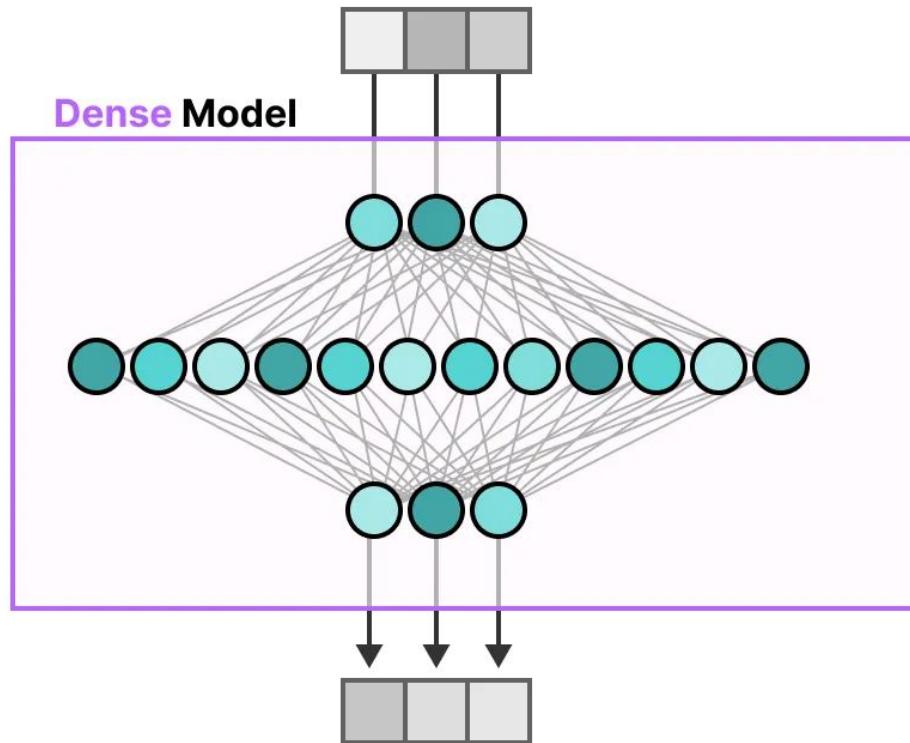
LLM: FFNN

- The FFNN, however, does grow quickly in size. To learn these complex relationships, it typically expands on the input it receives:



LLM: FFNN

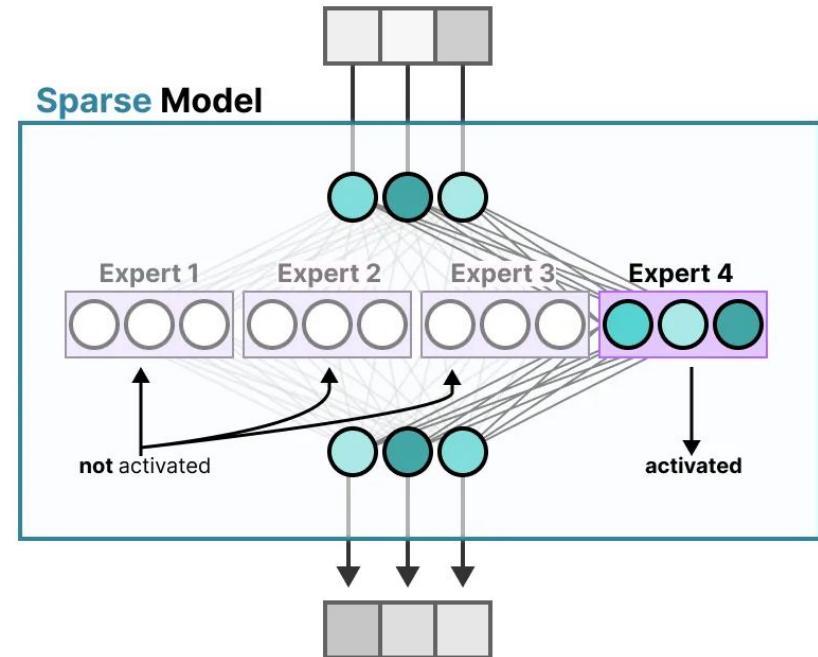
- If we take a closer look at the FFNN (**dense**) model, notice how the input activates all parameters to some degree:



The Experts

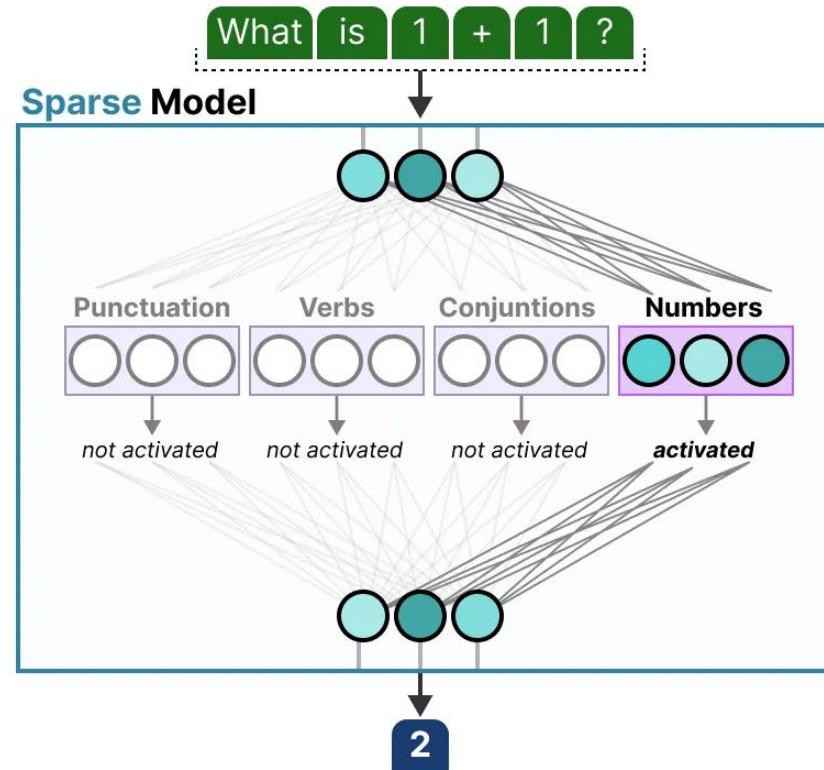
- In contrast, sparse models only activate a portion of their total parameters and are closely related to Mixture of Experts.
- To illustrate, we can chop up our dense model into pieces (so-called **experts**), retrain it, and only activate a subset of experts at a given time:

- The underlying idea is that each expert learns different information during training.
- Then, when running inference, only specific experts are used as they are most relevant for a given task.



The Experts

- When asked a question, we can select the expert best suited for a given task:

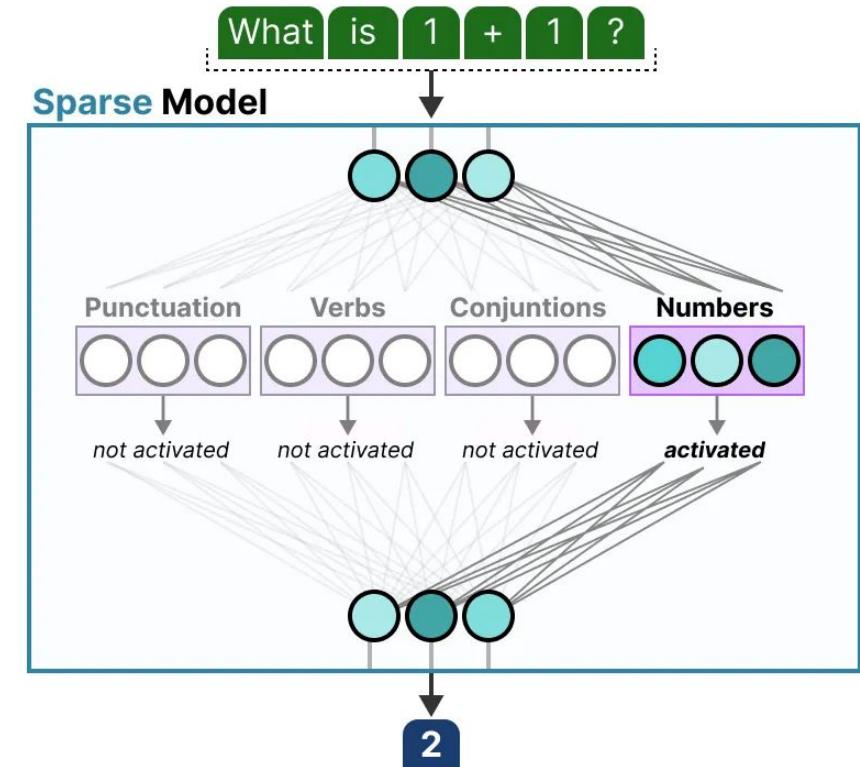


The Experts

- When asked a question, we can select the expert best suited for a given task:

Experts learn more fine-grained information than entire domains

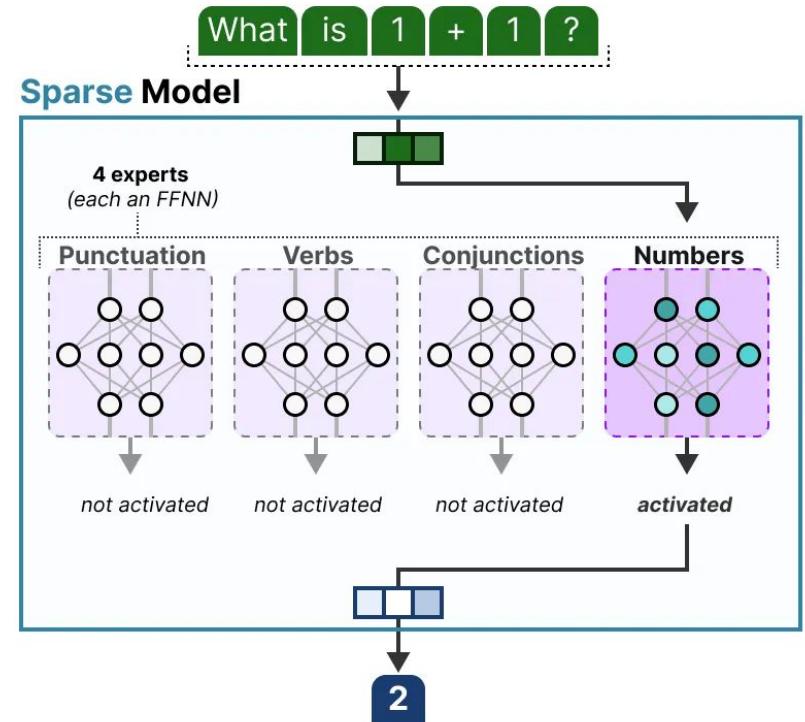
Expert specialization	Expert position	Routed tokens
Punctuation	Layer 2-....).
	Layer 6,:., & , & & ? &-,? ...
Conjunctions and articles	Layer 3	The the the the the the the The...
	Layer 6	a and and and and and and or and ...
Verbs	Layer 1	died falling identified fell closed left posted lost felt left said read miss place struggling falling signed died...
Visual descriptions <i>color, spatial position</i>	Layer 0	her over her know dark upper dark outer center upper blue inner yellow raw mama bright bright over open your dark blue
Counting and numbers <i>written and numerical forms</i>	Layer 1	after 37 19. 6. 27 Seven 25 4, 54 two dead we Some 2012 who we few lower



Expert specialization of an encoder model in the ST-MoE paper.

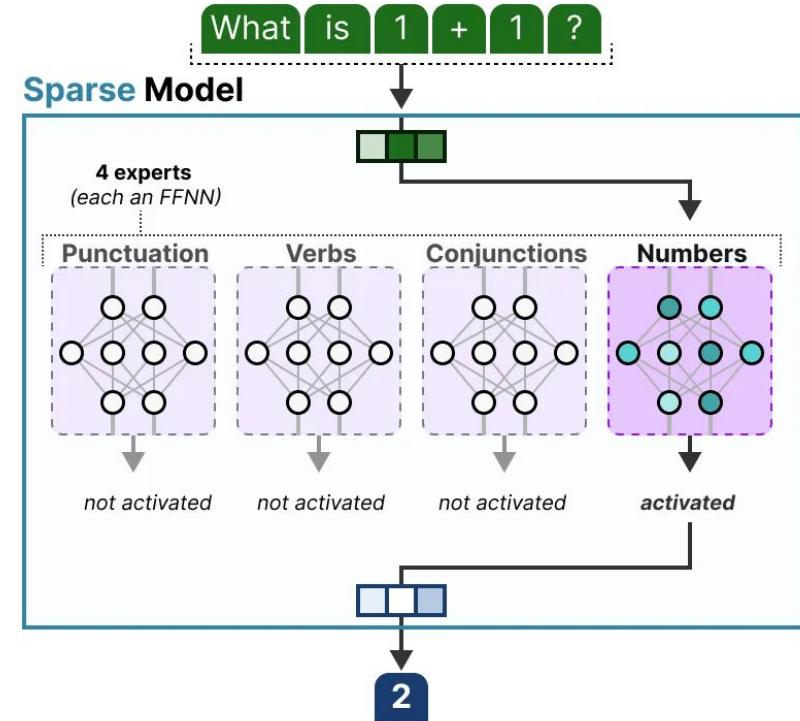
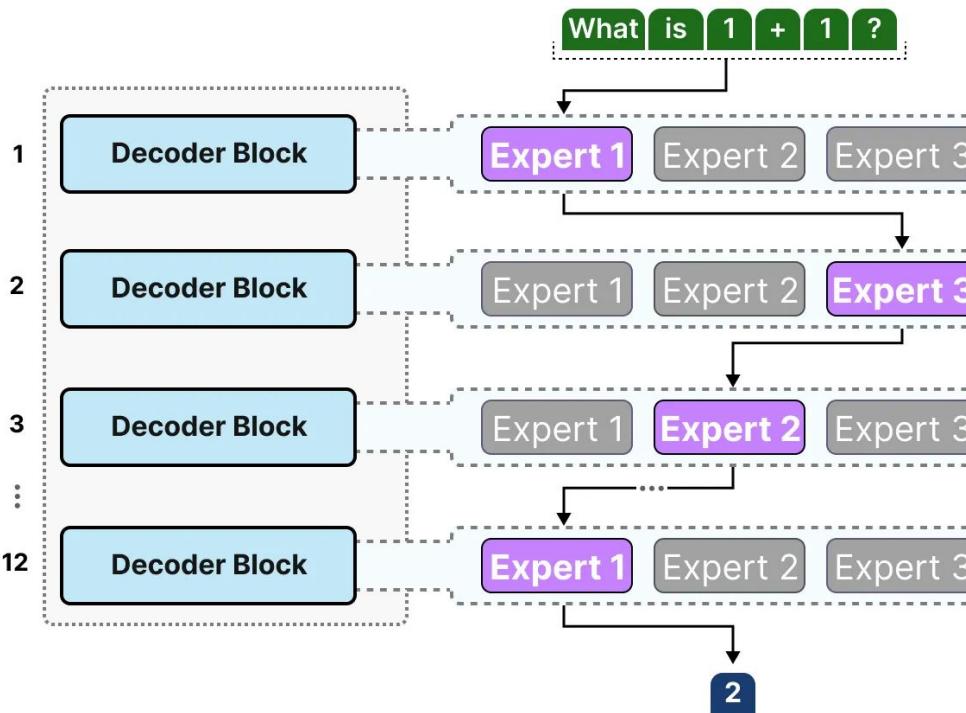
The Architecture of Experts

- Although it's nice to visualize experts as a hidden layer of a dense model cut in pieces, they are often whole FFNNs themselves:



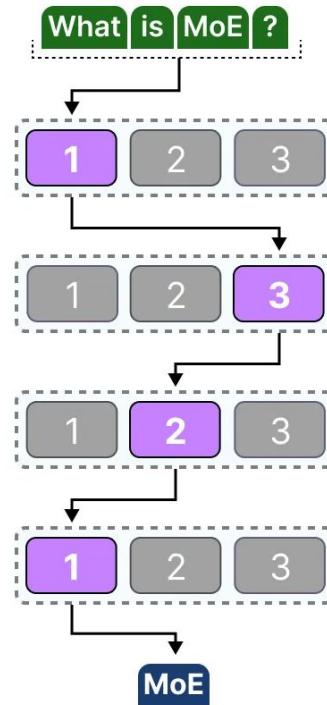
The Architecture of Experts

- Since most LLMs have several decoder blocks, a given text will pass through multiple experts before the text is generated:

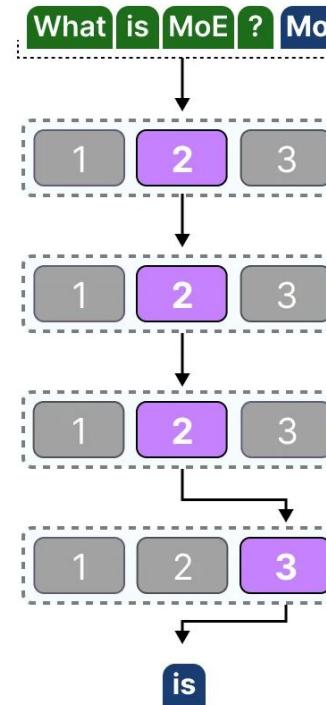


The Architecture of Experts

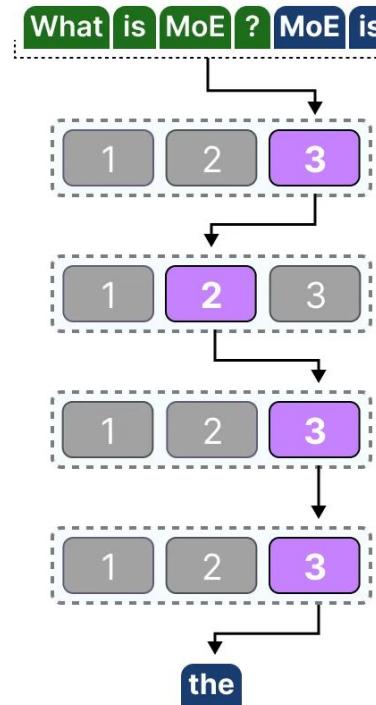
- The chosen experts likely differ between tokens which results in different “paths” being taken:



First pass



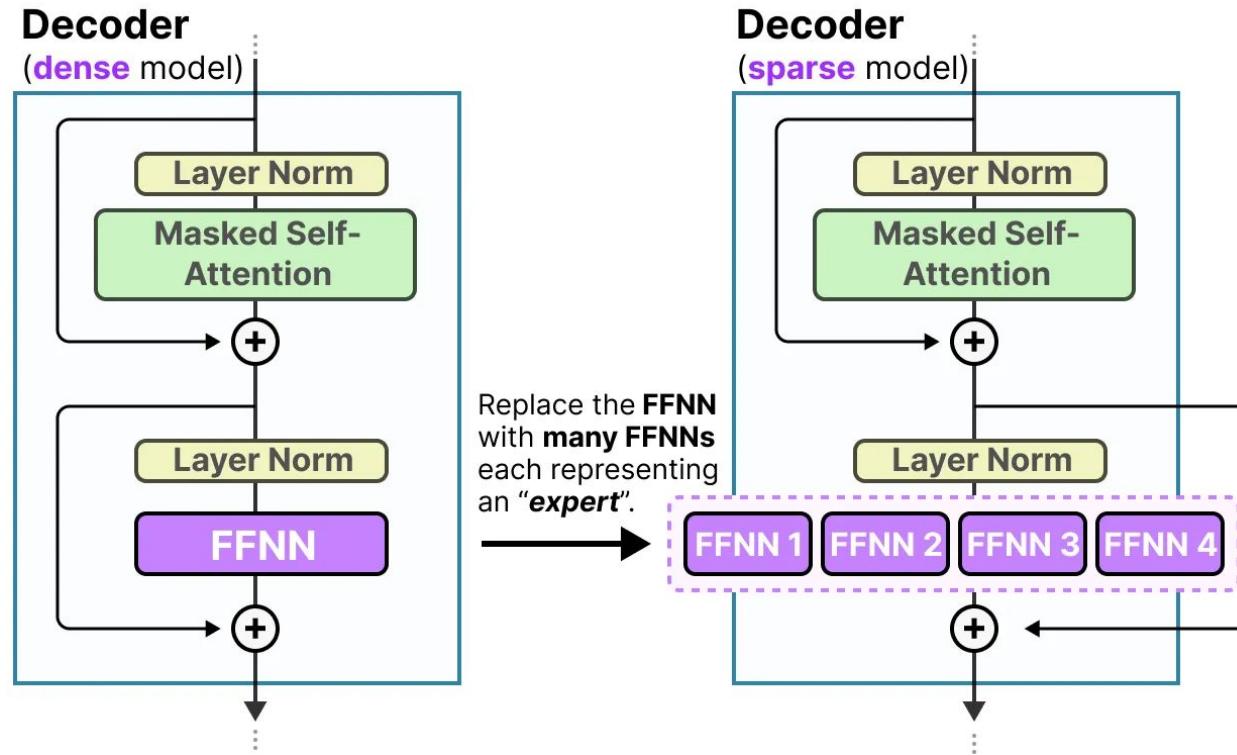
Second pass



Third pass

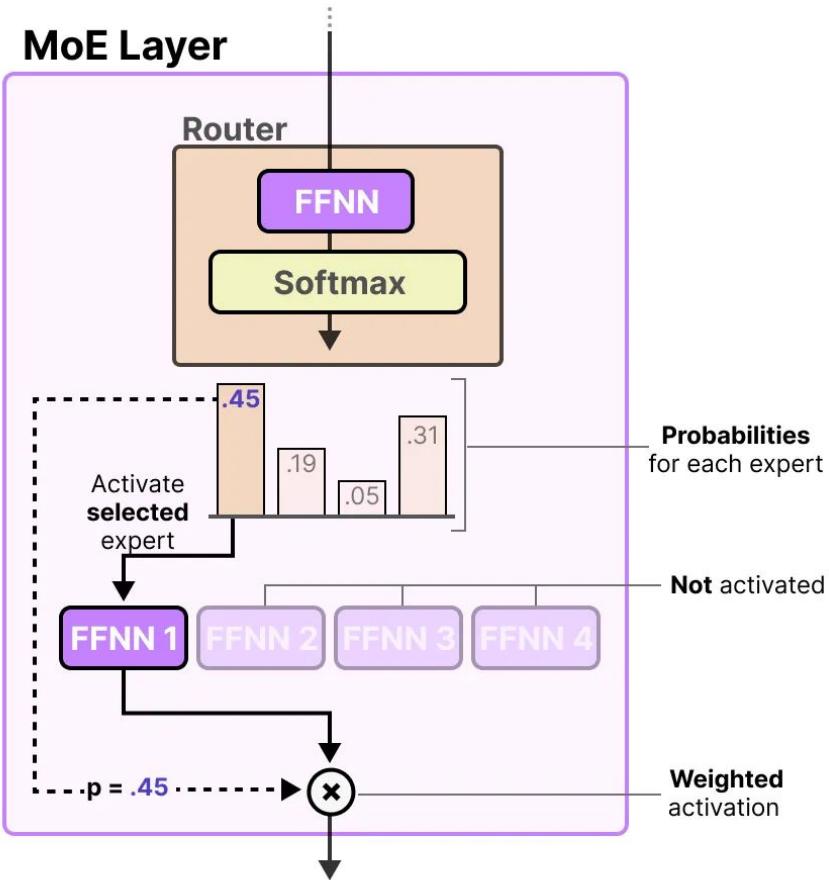
The Architecture of Experts

- If we update our visualization of the decoder block, it would now contain more FFNNs (one for each expert) instead:



The Router Mechanism

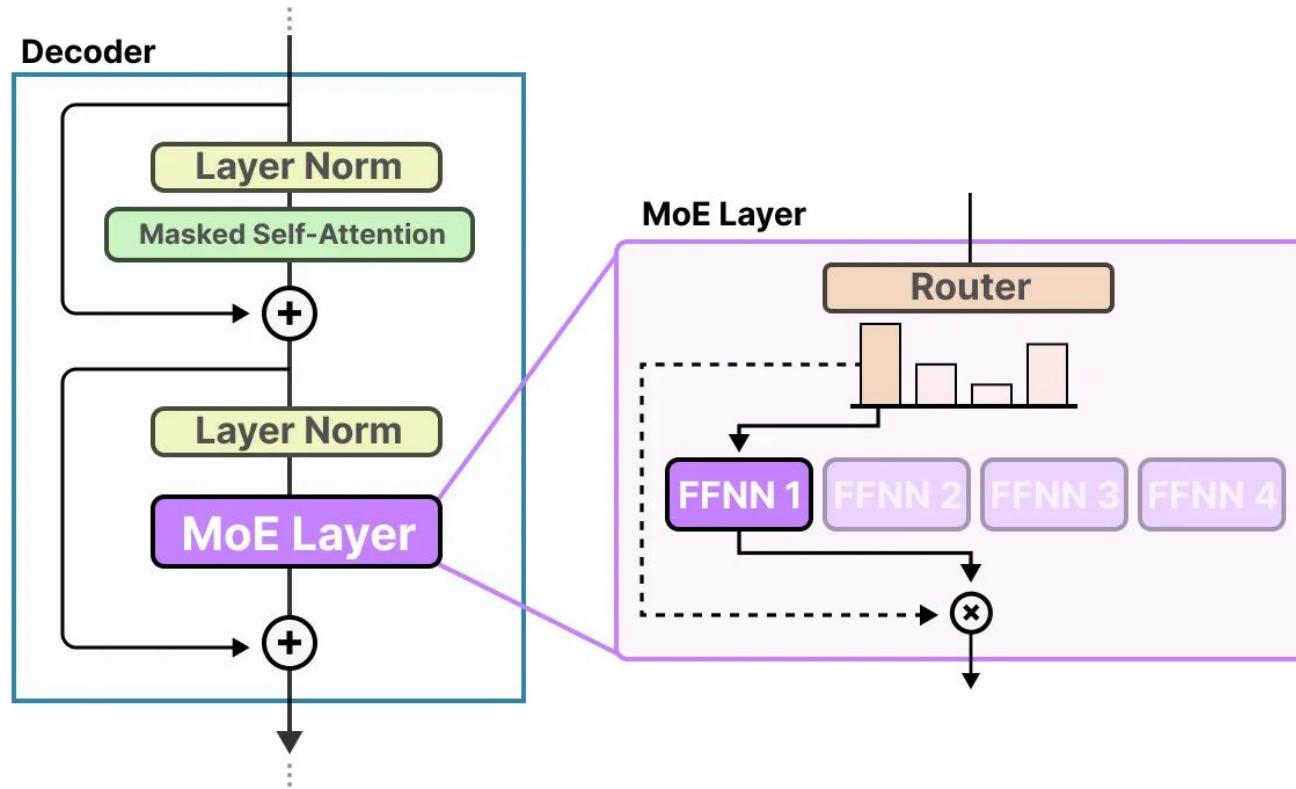
- Now that we have a set of experts, how does the model know which experts to use?
- Just before the experts, a router (also called a gate network) is added which is trained to choose which expert to choose for a given token.
- The **router** (or **gate network**) is also an FFNN and is used to choose the expert based on a particular input. It outputs probabilities which it uses to select the best matching expert:



The expert layer returns the output of the selected expert multiplied by the gate value (selection probabilities).

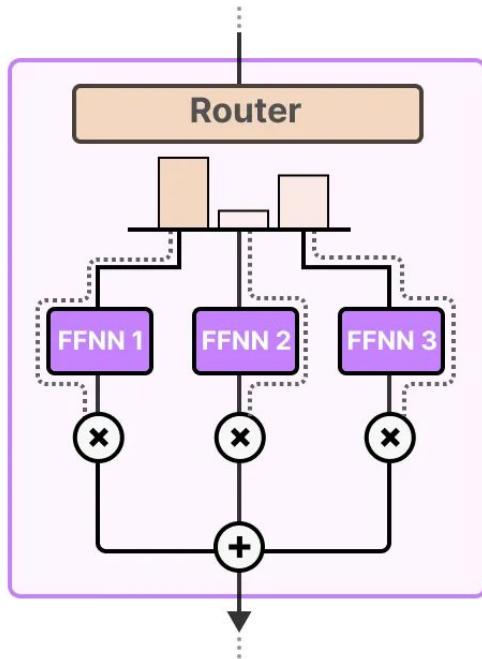
MoE Layer

- The router together with the experts (of which only a few are selected) makes up the MoE Layer:

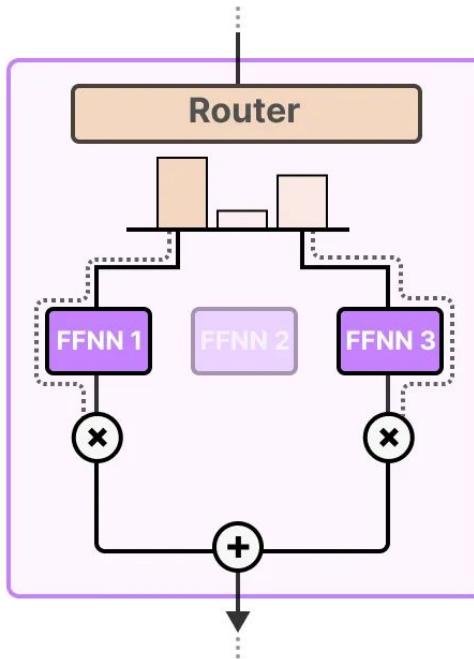


MoE Layer: Dense Vs Sparse

- Both use a router to select experts but a Sparse MoE only selects a few whereas a Dense MoE selects them all but potentially in different distributions.



Dense MoE



Sparse MoE

- For instance, given a set of tokens, a MoE will distribute the tokens across all experts whereas a Sparse MoE will only select a few experts.
- In the current state of LLMs, when you see a “MoE” it will typically be a Sparse MoE as it allows you to use a subset of experts.
- This is computationally cheaper which is an important trait for LLMs.

Selection of Experts

- The gating network is arguably the most important component of any MoE as it not only decides which experts to choose during inference but also training.

- In its most basic form, we multiply the input (x) by the router weight matrix (W):

The diagram illustrates the computation of $H(\mathbf{x})$. It shows three matrices: an *input* matrix with green blocks, a *router weights* matrix with orange blocks, and the resulting *output* matrix with blue blocks. Below the matrices is the equation $H(\mathbf{x}) = \mathbf{x} * \mathbf{W}$, where \mathbf{x} is the input and \mathbf{W} is the router weights matrix.

- Then, we apply a SoftMax on the output to create a probability distribution $G(x)$ per expert:

The diagram shows the SoftMax operation. It starts with the *output* matrix from the previous step, which is then processed by the **Softmax** function to produce a *probability distribution per expert*. This distribution is represented by a bar chart with varying heights. Below the chart is the equation $G(\mathbf{x}) = \text{Softmax}(H(\mathbf{x}))$.

Selection of Experts

- The router uses this probability distribution to choose the best matching expert for a given input.
- Finally, we multiply the output of each router with each selected expert and sum the results.

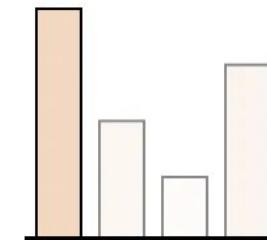
Sparse MoE

output

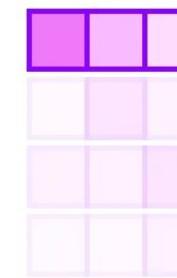


selected
Expert

Router Output



Output per *Expert*

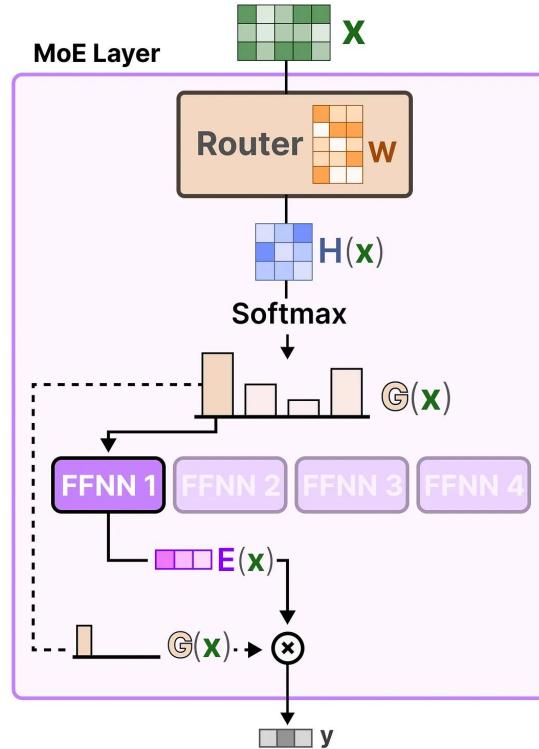


$$y = \sum (G(x) * E(x))$$

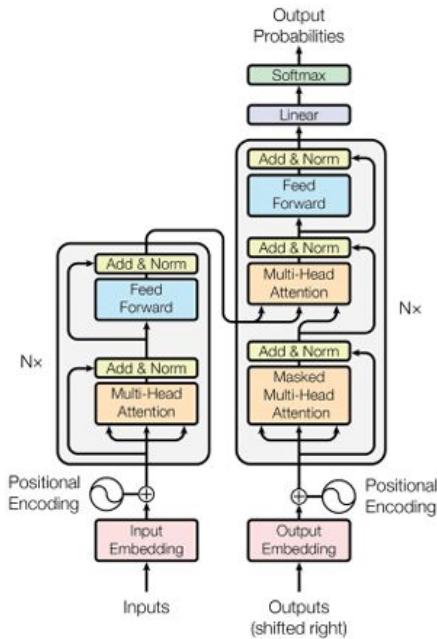
(only **1** expert is chosen in this example)

Selection of Experts

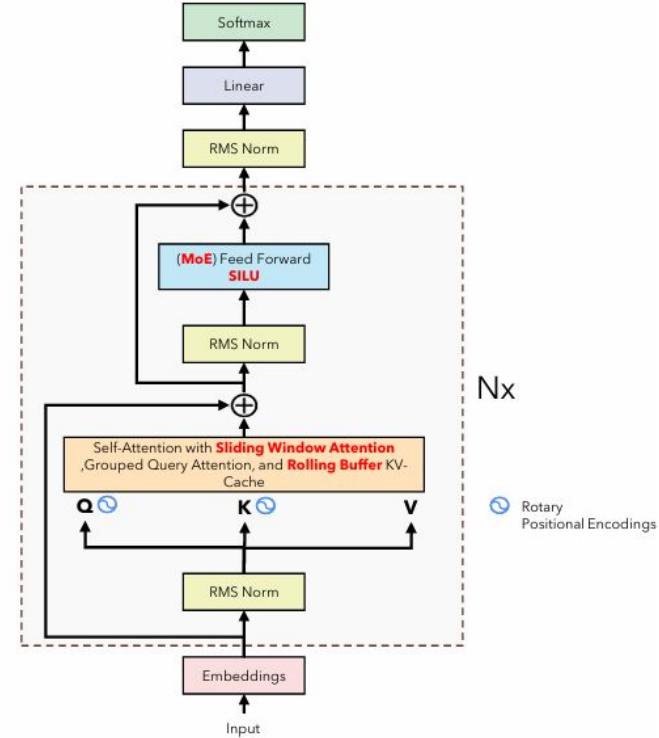
- Let's put everything together and explore how the input flows through the router and experts:



Mistral



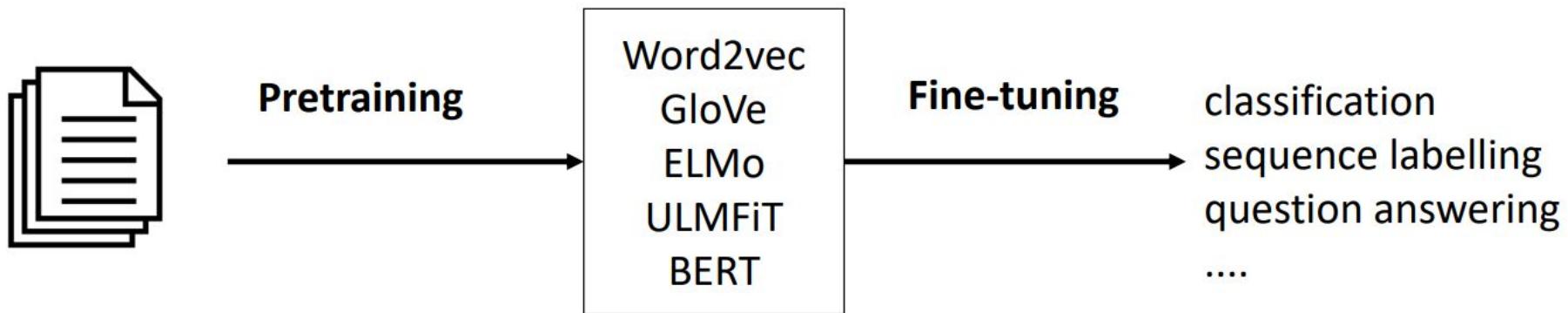
Transformer
("Attention is all you need")



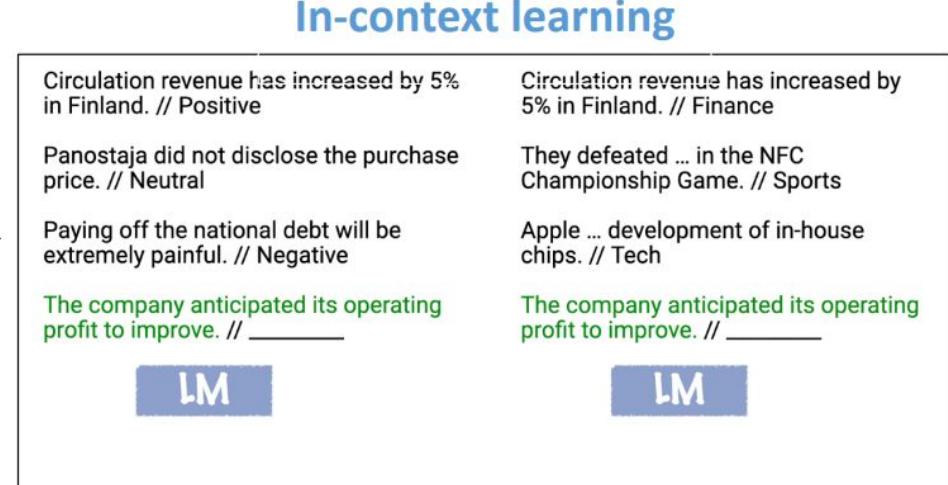
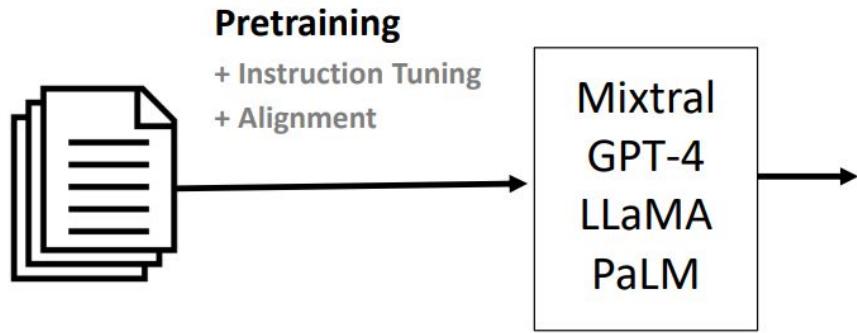
Mistral

PEFT: Parameter Efficient Fine Tuning

Pre-training and Fine Tuning before LLM era



Pre-training and Fine Tuning after LLM era

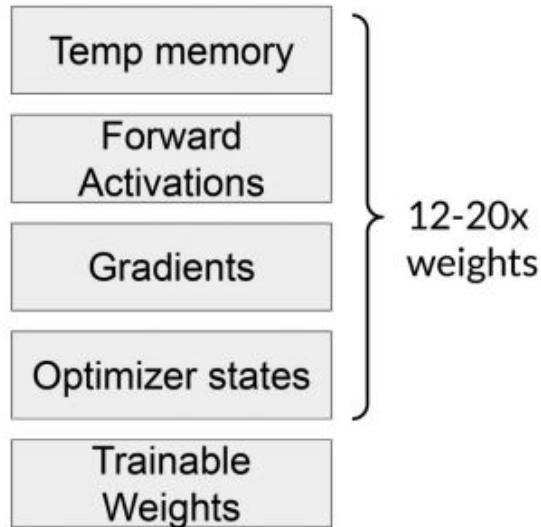


- In-context learning has mostly replaced fine-tuning in large models
- In-context learning is very useful if we don't have direct access to the model, for instance, if we are using the model through an API.

Disadvantage of In-context learning

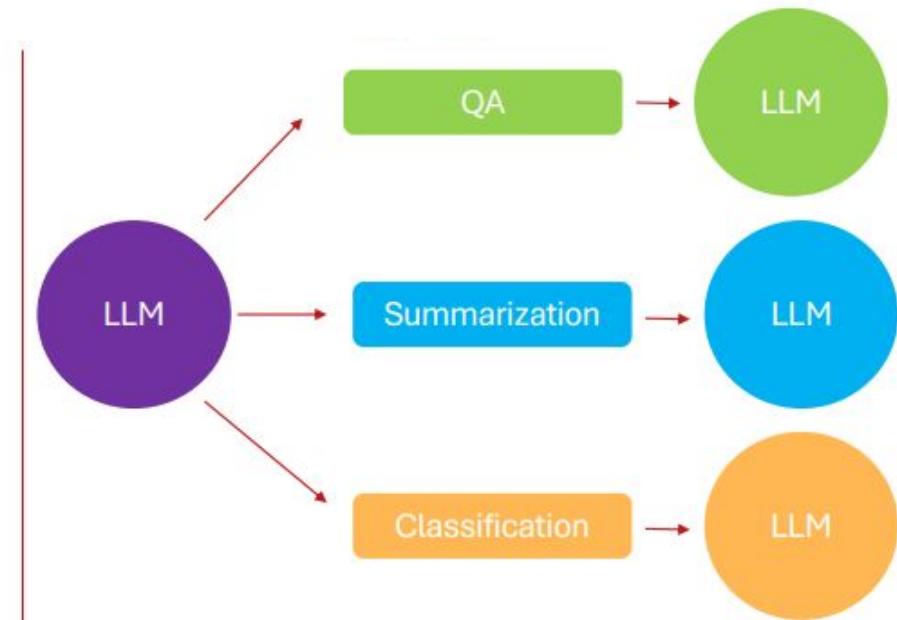
- **Poor Performance:** Prompting generally performs worse than fine-tuning.
- **Sensitivity:** to the wording of the prompt, order of examples, etc
- **Lack of clarity:** regarding what the model learns from the prompt. Even random labels work.
- **Inefficiency:** The prompt needs to be processed every time the model makes a prediction.

Why Full Fine Tuning is challenging?



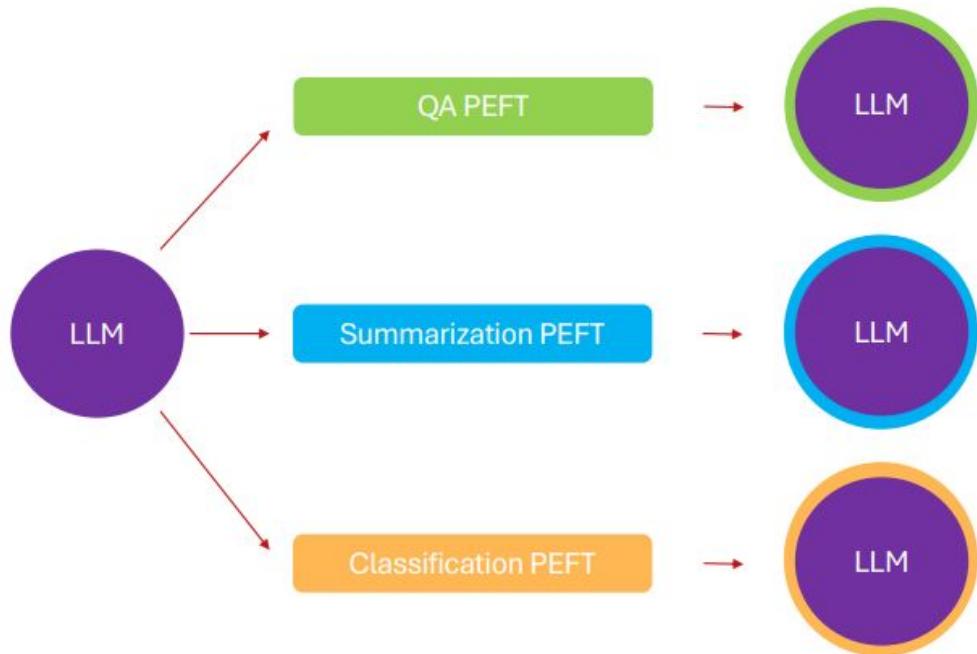
1. Hardware Requirements

3. Huge training data requirement for fine tuning.



2. Storage

PEFT



- **Reduced computational costs :** requires fewer GPUs and GPU time
- **Lower hardware requirements:** works with smaller GPUs & less memory
- **Better modelling performance:** reduces overfitting by preventing catastrophic forgetting
- **Less storage:** majority of weights can be shared across different tasks

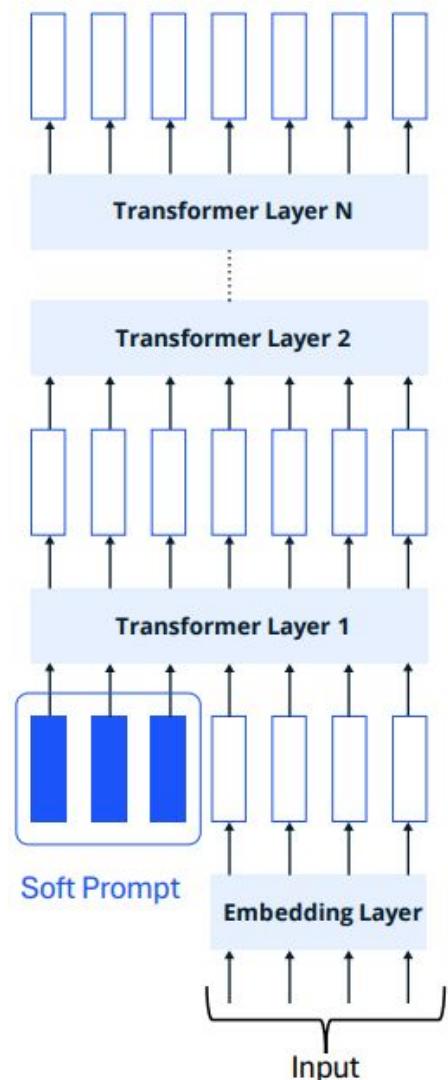
Fewer no. of parameters is trained

PEFT Techniques

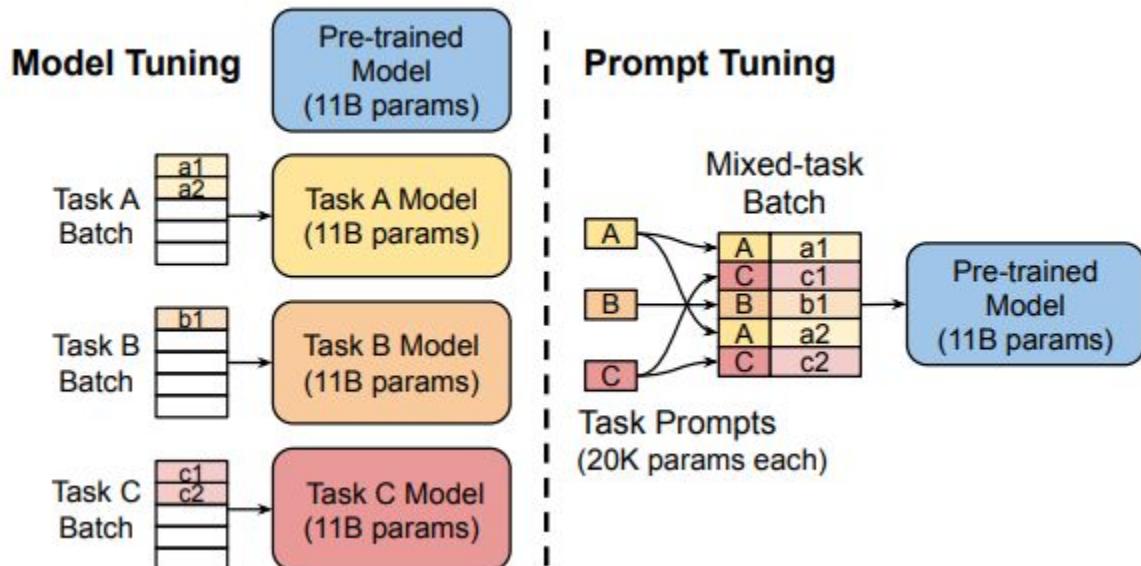
- (Soft) Prompt Tuning
- Prefix Tuning
- Adapters
- Low Rank Adaptation

Soft Prompt Tuning

- prepends a trainable tensor to the model's input embeddings, creating a soft prompt
- for a specific task, only a small task-specific soft prompt needs to be stored
- soft prompt tuning is significantly more parameter-efficient than full-finetuning.



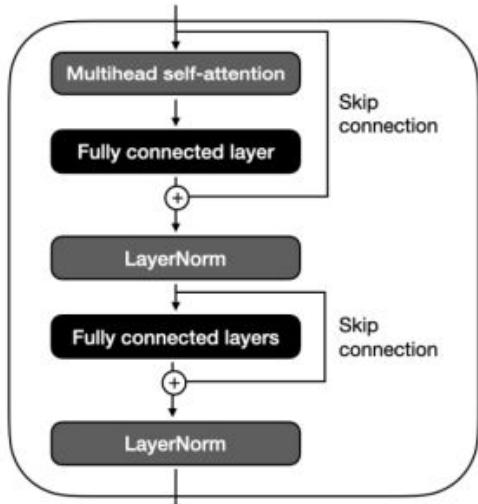
Soft Prompt Tuning



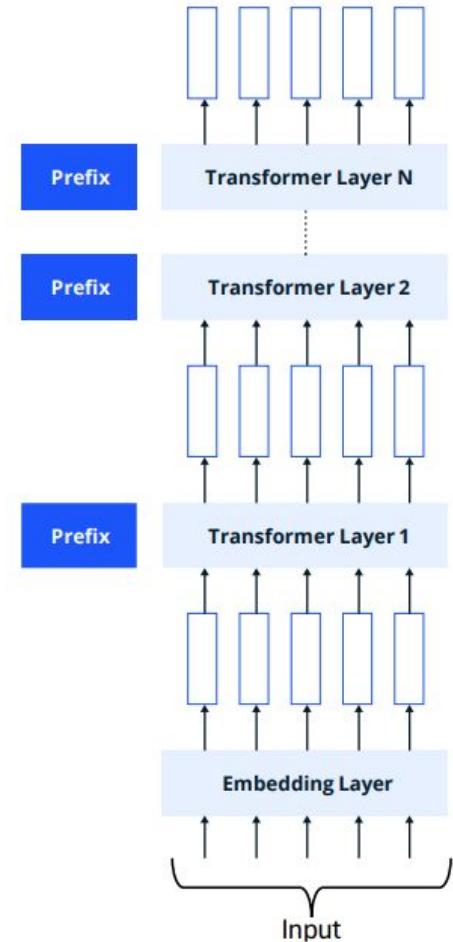
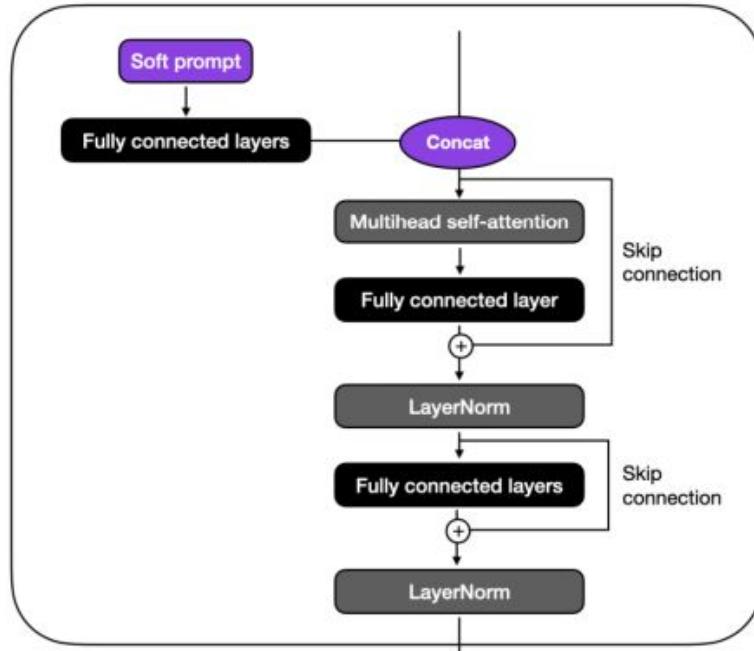
- **Model tuning** requires making a task specific copy of the entire pre-trained model for each downstream task and inference must be performed in separate batches.
- **Prompt tuning** only requires storing a small task-specific prompt for each task, and enables mixed-task inference using the original pretrained model.

Prefix fine tuning

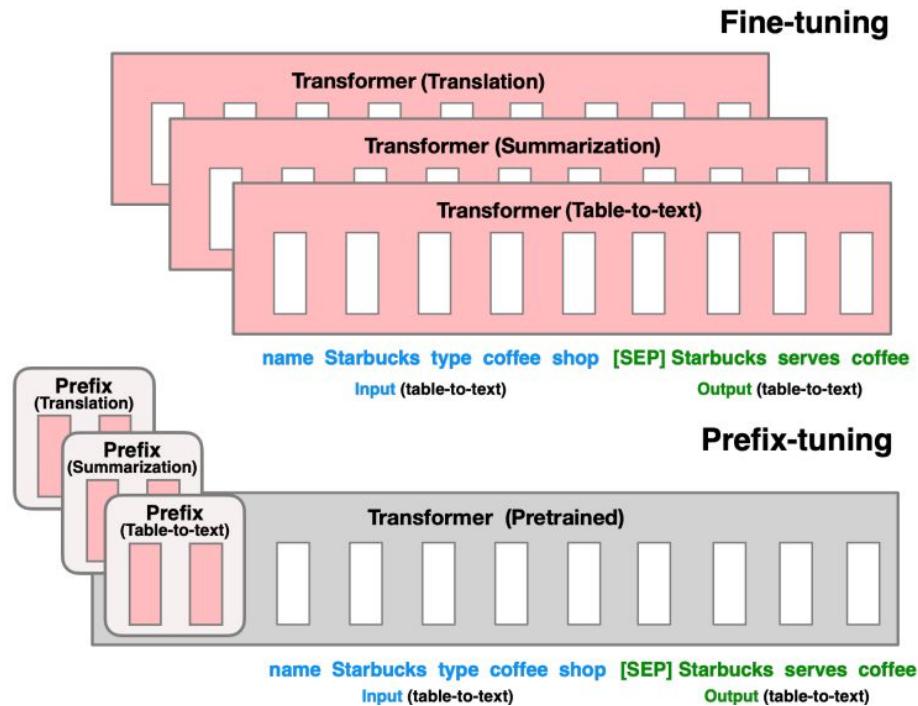
REGULAR TRANSFORMER BLOCK



TRANSFORMER BLOCK WITH PREFIX

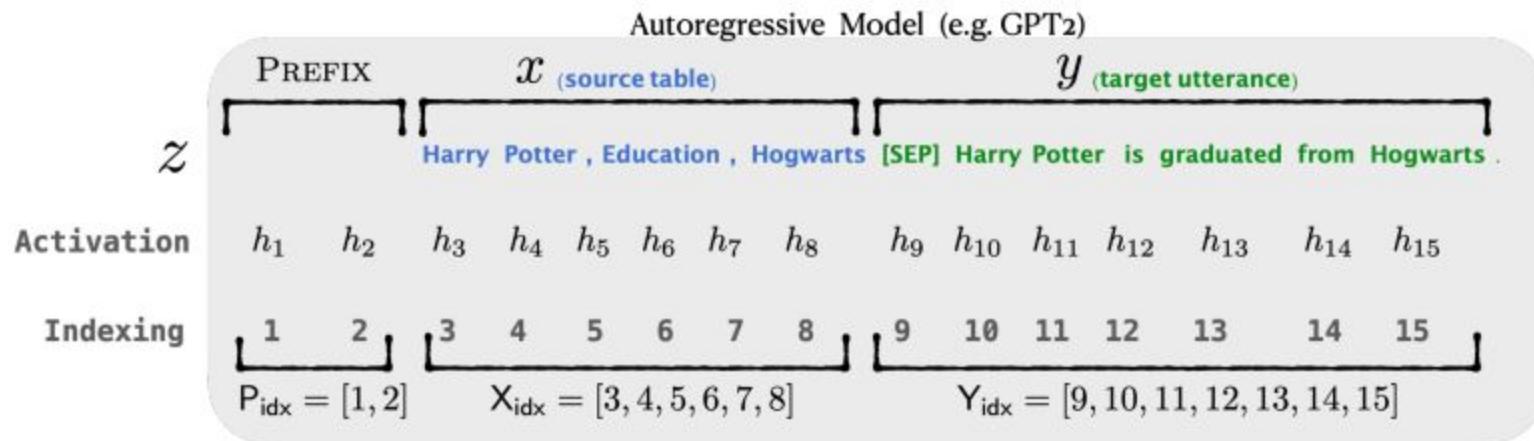


Prefix fine tuning



- Fine-tuning updates all Transformer parameters (the red Transformer box) and requires storing a full model copy for each task.
- Prefix-tuning, which freezes the Transformer parameters and only optimizes the prefix (the red prefix blocks).
- Consequently, we only need to store the prefix for each task, making prefix-tuning modular and space-efficient.

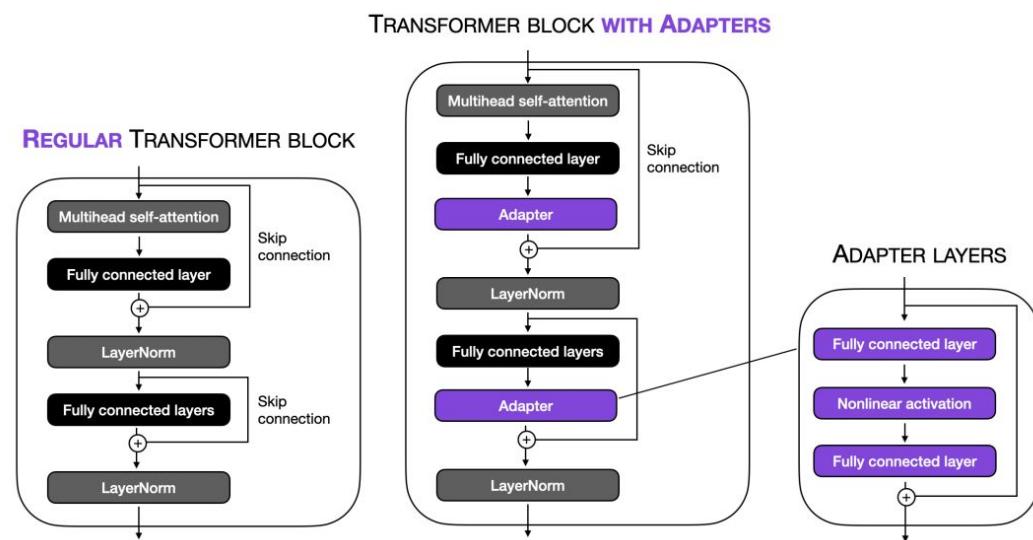
Prefix fine tuning



- An annotated example of prefix-tuning using an autoregressive LM.

Adapters

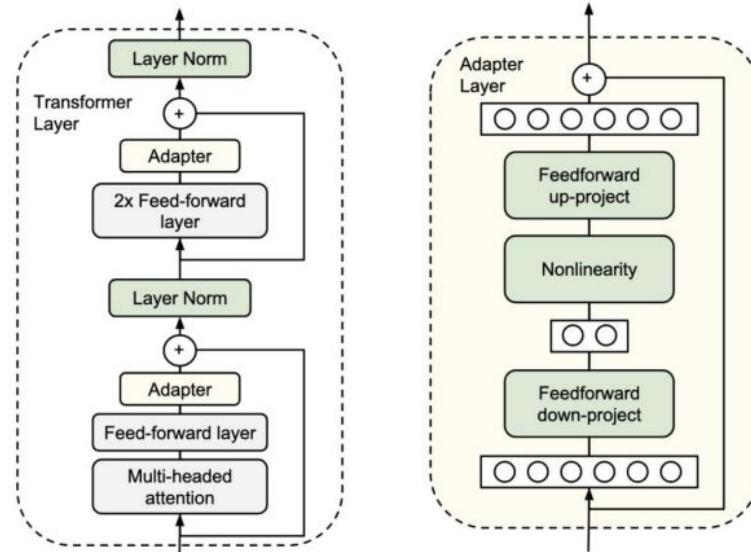
- Introduced a “Adapter” in each transformer block.
-



Adapters

Bottleneck Structure

- significantly reduces the number of parameters
- reduces d -dimensional features into a smaller m dimensional vector
- example: $d=1024$ and $m=24$
 - (1024×1024) requires 1,048,576 parameters
 - $2^* (1024 * 24)$ requires 49,152 parameters
- m determines the number of optimizable parameters and hence poses a parameter vs performance tradeoff.



Adapters-limitation

- **Performance issue:** other techniques with being more efficient showed better performance than Adapters.
- **Architecture rigidness:** Adapter layer, which is made up of bottleneck, is difficult to change.
- **Increased latency:** because Adapters are in between layers of transformer at inference time, it takes more time to give the output.

Low-Rank Adaption (LoRA)

Key Concept:

- Instead of fine-tuning the full weight matrix W , LoRA **freezes the base model** and introduces **low-rank matrices A and B**.
- **Formula:** $W' = W + \Delta W = W + AB$

A: Low-rank matrix of shape $d \times r$

B: Low-rank matrix of shape $r \times k$

$r \ll \min(d, k) \rightarrow$ Significantly fewer parameters

Benefits:

- Reduces trainable parameters
- Lower memory and compute requirements

Low-Rank Adaption (LoRA): Matrix Example

Step 1: Assume we have a simple **weight matrix W** with dimensions 4×3:

$$W = \begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 1 & 0 & 2 \\ 8 & 9 & 10 \end{bmatrix}$$

- Now, during fine-tuning, we want to slightly modify this matrix to adapt the model to a new task.
- Instead of fine-tuning the entire matrix, LoRA introduces a low-rank update.

Low-Rank Adaption (LoRA): Matrix Example

Step 2: LoRA creates two smaller matrices A and B that approximate the update:

$$\Delta \mathbf{W} = \mathbf{A} \times \mathbf{B}$$

Let's assume we choose rank $r = 2$, which is much smaller than the original dimensions.

Matrix A: 4*2

$$A = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \\ 1 & 2 \end{bmatrix}$$

Matrix B: 2*3

$$B = \begin{bmatrix} 0.5 & 1.0 & -0.5 \\ -0.5 & 0.0 & 0.5 \end{bmatrix}$$

Low-Rank Adaption (LoRA): Matrix Example

Step 3: Matrix Multiplication (Low-Rank Update)

$$\Delta W = A \times B$$

$$\Delta W = \begin{bmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 1 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 0.5 & 1.0 & -0.5 \\ -0.5 & 0.0 & 0.5 \end{bmatrix}$$

$$\Delta W = \begin{bmatrix} 0.5 & 1.0 & -0.5 \\ 0.5 & 2.0 & -0.5 \\ -0.5 & 0.0 & 0.5 \\ -0.5 & 1.0 & 0.5 \end{bmatrix}$$

Low-Rank Adaption (LoRA): Matrix Example

Step 4: The Final Adapted Matrix

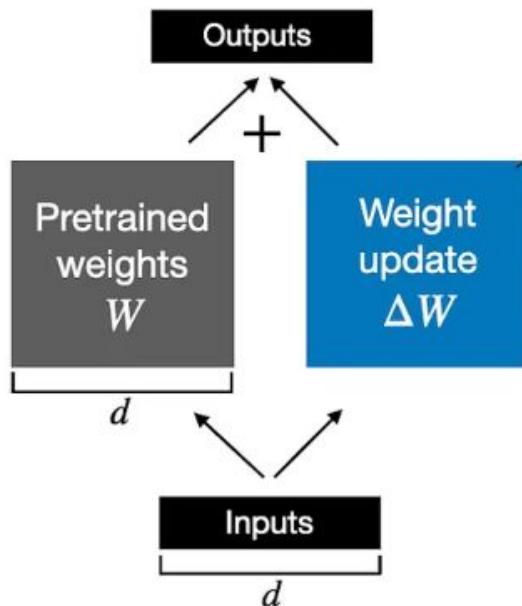
$$W' = W + \Delta W$$

$$W' = \begin{bmatrix} 2 + 0.5 & 4 + 1.0 & 6 - 0.5 \\ 3 + 0.5 & 5 + 2.0 & 7 - 0.5 \\ 1 - 0.5 & 0 + 0.0 & 2 + 0.5 \\ 8 - 0.5 & 9 + 1.0 & 10 + 0.5 \end{bmatrix}$$

$$W' = \begin{bmatrix} 2.5 & 5.0 & 5.5 \\ 3.5 & 7.0 & 6.5 \\ 0.5 & 0.0 & 2.5 \\ 7.5 & 10.0 & 10.5 \end{bmatrix}$$

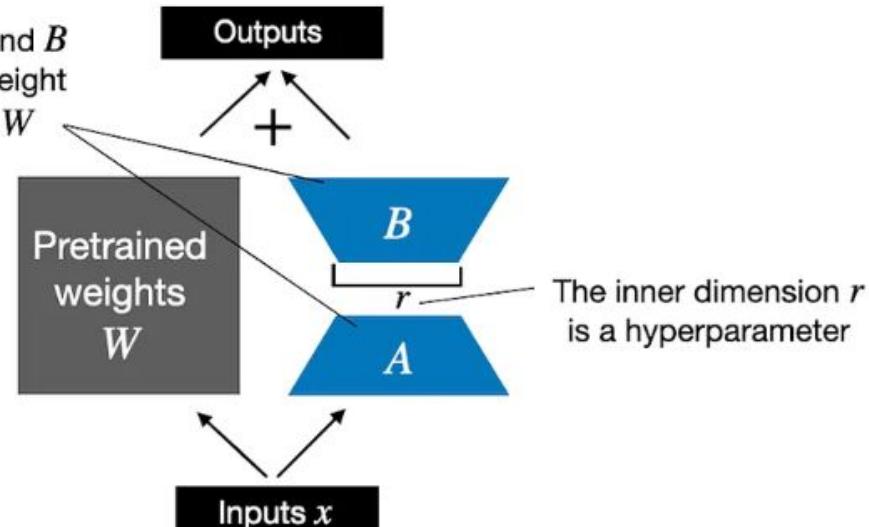
Low-Rank Adaption (LoRA)

Weight update in regular finetuning

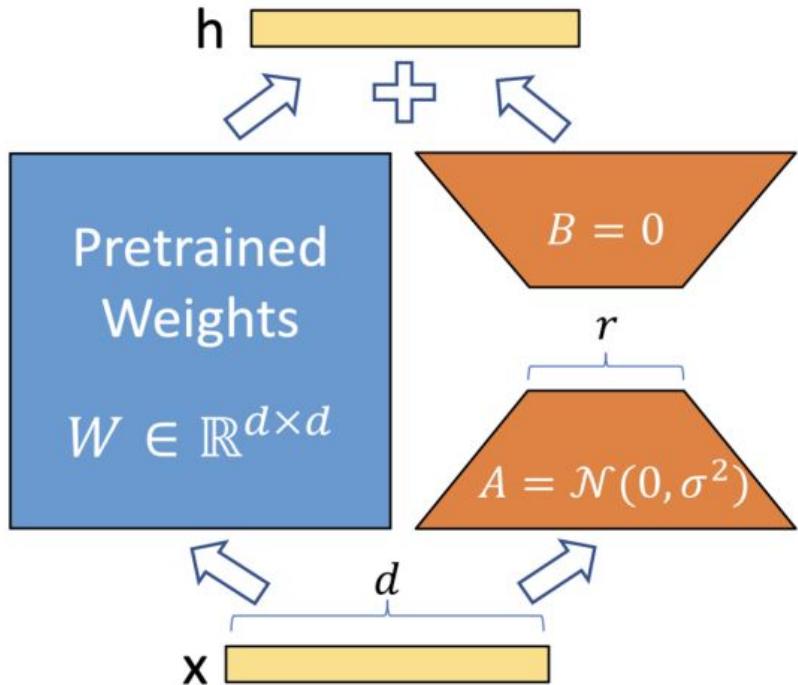


LoRA matrices A and B approximate the weight update matrix ΔW

Weight update in LoRA



Low-Rank Adaption (LoRA)



- By setting **B** to zero, the product $\Delta W = BA$ initially equals zero. This preserves the behaviour of the original model at the start of fine-tuning
- Gaussian distribution helps ensure that the values in **A** are neither too large nor too biased in any direction, which could lead to disproportionate influence on the updates when **B** begins to change.

Low-Rank Adaption (LoRA): Why it works

- **Freezing** the original model weights and adding a low-rank update.
- **Efficiently** fine-tuning fewer parameters while retaining model capabilities.
- **Reducing** memory usage and improving adaptability for large-scale models.

Intrinsic Dimension in LoRA

- To make **Intrinsic Dimension (ID)** clear, let's use a concrete matrix example and demonstrate how **low-rank adaptation** works by identifying **independent directions** (standalone columns) and redundant dependencies.
- Assume we have a simple **weight matrix W** with dimensions 4×3 :

$$W = \begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 1 & 0 & 2 \\ 8 & 9 & 10 \end{bmatrix}$$

Intrinsic Dimension in LoRA

$$W = \begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 1 & 0 & 2 \\ 8 & 9 & 10 \end{bmatrix}$$

To find the **intrinsic dimension**, we need to determine how many **linearly independent directions** exist. Let's inspect the columns:

- Column 1: $c1=[1,4,2,3]^T$
- Column 2: $c2=[2,8,4,6]^T = 2 \times c1 \rightarrow$ Linearly dependent
- Column 3: $c3=[3,12,6,9]^T = 3 \times c1 \rightarrow$ Linearly dependent

Thus:

- The matrix only has **one linearly independent column**.
- The **rank** of this matrix is **1**.
- Even though the matrix has $4 \times 3 = 12$ elements, its **intrinsic dimension** is only **1**.

Intrinsic Dimension in LoRA

What Does This Mean?

- The matrix **looks like it's 4×3** , but it **only has one unique direction**.
- The **second and third columns** are just scaled versions of the first.
- Hence, the matrix can be **compressed** to a low-rank representation without losing information.

Intrinsic Dimension in LoRA

In LoRA:

- The **full model weights** might have millions or billions of parameters, but the **effective fine-tuning** happens in a much lower-dimensional space.
- The **low-rank matrices A and B** represent the meaningful fine-tuning directions.
- The **intrinsic dimension** corresponds to the actual number of unique directions that significantly impact the output.

Quantization: numerical values representation



Quantization: numerical values representation

Sign (1 bit) Exponent (8 bits) Significand / Mantissa (23 bits)

FP32 0 10000000 1001001000011111011011

Sign (1 bit) Exponent (5 bits) Significand / Mantissa (10 bits)

FP16 0 10000 1001001000

(signed) **INT8** 0 1001000

(1 bit) (7 bits)

Quantization: from FP32 to INT8

FP32		
5.47	3.08	-7.59
0	-1.95	-4.57
10.8	3.02	-1.92

INT8		
64	36	-89
0	-23	-54
127	36	-23

FP32		
5.44	3.06	-7.54
0	-1.96	-4.59
10.8	3.06	-1.96

FP32 (original)		
5.47	3.08	-7.59
0	-1.95	-4.57
10.8	3.02	-1.92

FP32 (dequantized)		
5.44	3.06	-7.54
0	-1.96	-4.59
10.8	3.06	-1.96

Quantization error		
.03	.02	.05
0	-.01	-.02
0	-.04	-.04

$$s = \frac{2^{b-1}-1}{\alpha}$$

(scale factor)

$$x_{\text{quantized}} = \text{round}(s \cdot x)$$

(quantization)

$$s = \frac{127}{10.8} = 11.76$$

(scale factor)

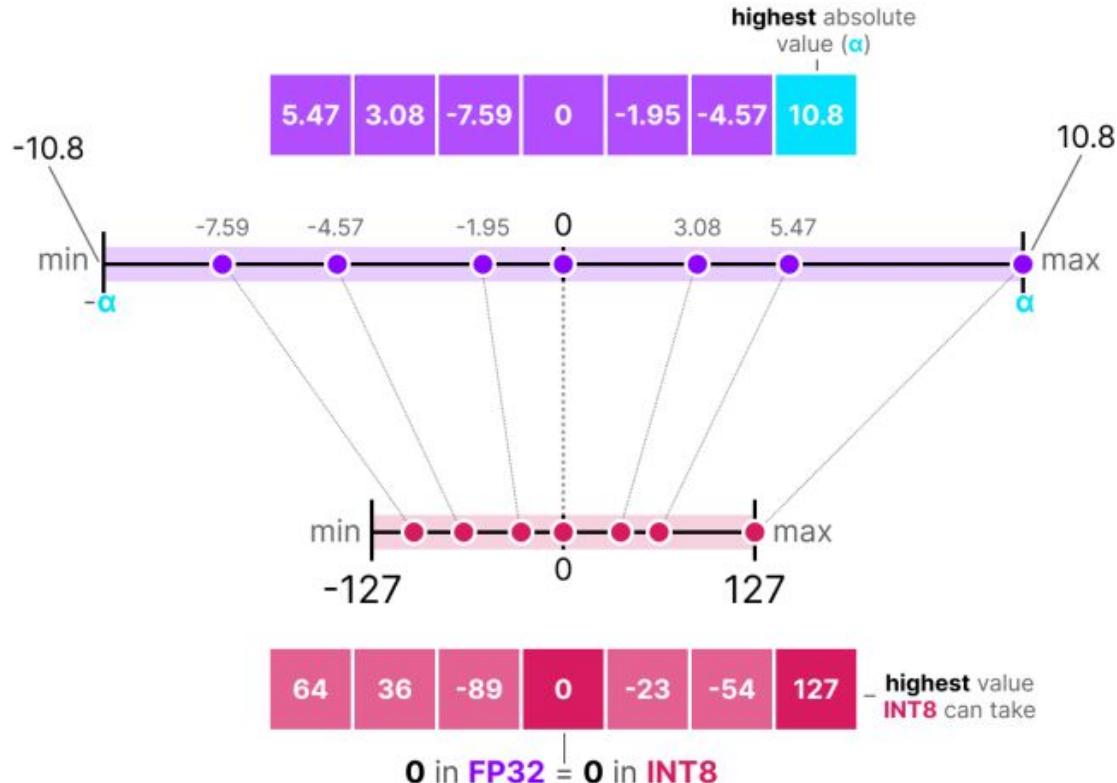
$$x_{\text{quantized}} = \text{round}(11.76 \cdot \boxed{})$$

(quantization)

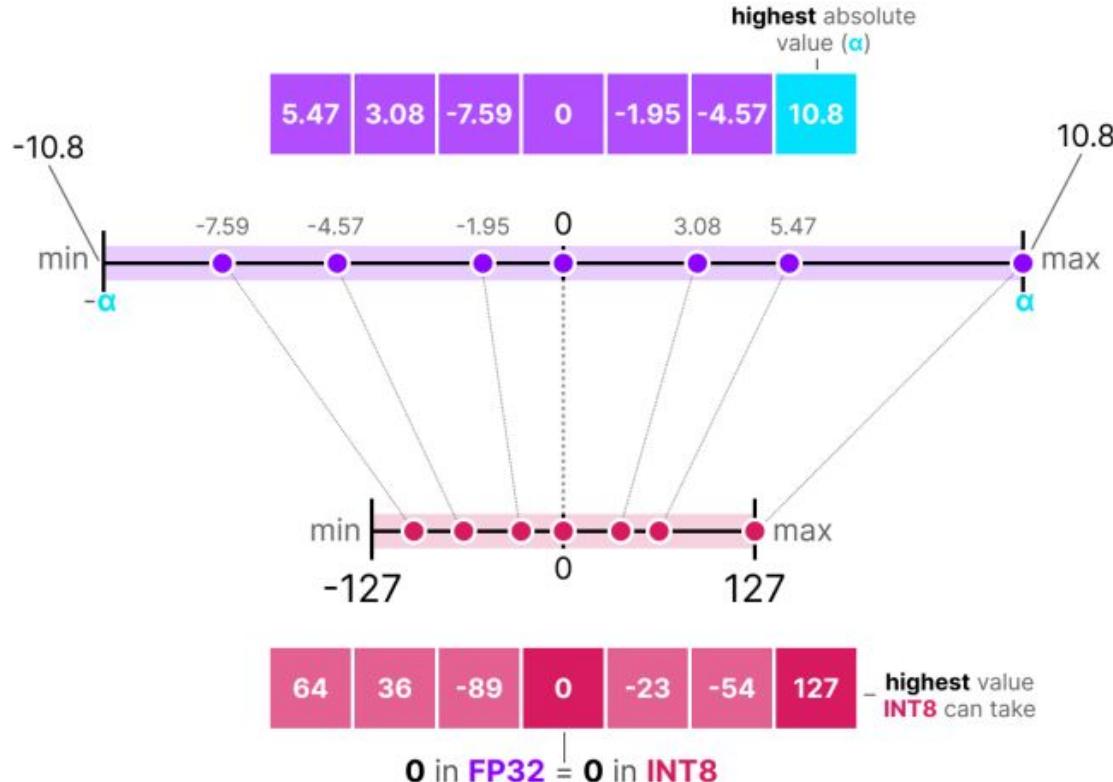
$$x_{\text{dequantized}} = \frac{\boxed{}}{s}$$

(dequantize)

Quantization: from FP32 to INT8



Quantization: from FP32 to INT8



$$s = \frac{2^{b-1}-1}{\alpha}$$

$$x_{\text{quantized}} = \text{round}(s \cdot x)$$

$$s = \frac{127}{10.8} = 11.76$$

$$x_{\text{quantized}} = \text{round}(11.76 \cdot \text{████████})$$

$$x_{\text{dequantized}} = \frac{\text{████████}}{s}$$

Quantization: option

- Post Training Quantization
- Quantization Aware Training (Fine-tuning LLM)

Quantization: example

Let's consider a **weight matrix W** with **FP32 precision**:

$$W = \begin{bmatrix} 1.53 & -2.81 \\ 0.76 & 3.91 \end{bmatrix}$$

Step 1: Define Quantization Range

We want to **quantize this matrix to 4-bit precision**.

- **4-bit precision** $\rightarrow 2^4=16$ discrete values.
- **Range:** $[-8,7]$ (signed 4-bit integers)

Quantization: example

Let's consider a **weight matrix W** with **FP32 precision**:

$$W = \begin{bmatrix} 1.53 & -2.81 \\ 0.76 & 3.91 \end{bmatrix}$$

Step 2: Quantization Formula

We need to **map the floating-point values into the integer range**.

The quantization formula is:
$$Q(W) = \text{round}\left(\frac{W}{S}\right)$$

Quantization: example

Let's consider a **weight matrix W** with **FP32 precision**:

$$W = \begin{bmatrix} 1.53 & -2.81 \\ 0.76 & 3.91 \end{bmatrix}$$

Step 3: Find the scaling factor

To map the matrix values into the **4-bit range**: $S = \frac{\max(|W|)}{7}$

The maximum absolute value: $\max(|W|) = 3.91$

Scaling factor: $S = \frac{3.91}{7} \approx 0.5586$

Quantization: example

Let's consider a **weight matrix W** with **FP32 precision**:

$$W = \begin{bmatrix} 1.53 & -2.81 \\ 0.76 & 3.91 \end{bmatrix}$$

Step 4: Apply Quantization

$$Q(W) = \text{round}\left(\frac{W}{0.5586}\right)$$

$$Q(W) = \begin{bmatrix} \text{round}(1.53/0.5586) & \text{round}(-2.81/0.5586) \\ \text{round}(0.76/0.5586) & \text{round}(3.91/0.5586) \end{bmatrix}$$

$$Q(W) = \begin{bmatrix} 3 & -5 \\ 1 & 7 \end{bmatrix}$$

Quantization: example

Let's consider a **weight matrix W** with **FP32 precision**:

$$W = \begin{bmatrix} 1.53 & -2.81 \\ 0.76 & 3.91 \end{bmatrix}$$

Step 5: De-Quantization

$$\hat{W} = Q(W) \times S$$

$$\hat{W} = \begin{bmatrix} 3 & -5 \\ 1 & 7 \end{bmatrix} \times 0.5586$$

$$\hat{W} = \begin{bmatrix} 1.68 & -2.79 \\ 0.56 & 3.91 \end{bmatrix}$$

The reconstructed matrix is **slightly different** from the original due to the **loss of precision** during quantization.

QLoRA or Quantized LoRA

QLoRA is an **optimization technique** that combines:

- **Quantization** → To reduce the memory footprint of large language models (LLMs).
- **LoRA (Low-Rank Adaptation)** → To fine-tune only a small, low-rank subset of parameters.

The combination of quantization and LoRA allows **efficient fine-tuning** of massive models on **consumer-grade GPUs** with limited memory, without sacrificing much performance.

Why QLoRA?

Fine-tuning large models like **LLaMA 7B**, **13B**, or **65B** is extremely **memory-intensive**:

- These models require **hundreds of GB of VRAM** for full fine-tuning.
- Even **LoRA** reduces the number of trainable parameters but still requires the **full model to be loaded in memory**, which can be expensive.

QLoRA solves this problem by:

- **Quantizing the base model** to reduce its memory footprint.
- **Applying LoRA** on top of the quantized model.
- This allows you to **fine-tune large models on a single GPU**.

How QLoRA works?

QLoRA involves **three key steps**:

Step 1: 4-Bit Quantization of the Base Model

The first step is to **quantize the model** into **4-bit precision** (instead of the default 16-bit or 32-bit precision).

- This reduces the memory footprint by **4x-8x**.
- The quantized model is **frozen during fine-tuning**, meaning the base weights are not updated.
- Only the **LoRA adapters** are trained.

How QLoRA works?

QLoRA involves three key steps:

Step 1: 4-Bit Quantization of the Base Model

Step 2: LoRA Fine-Tuning on Quantized Model

Step 3: Backpropagation with Frozen Quantized Base

QLoRA: Mathematical formulation

Let W be the original weight matrix of the model.

After QLoRA, the updated weight matrix becomes:

$$W' = \text{Quantized}(W) + \Delta W$$

Where:

- $\text{Quantized}(W) \rightarrow$ The frozen, low-memory, 4-bit quantized model.
- $\Delta W = A \times B \rightarrow$ The **low-rank adaptation**.
- Only A and B are trainable.

QLoRA: Benefits

1. Massive Memory Savings:

- **4-bit quantization** reduces the model size by **4x-8x**, making it possible to fine-tune large models on a single GPU.
 - For example: **LLaMA-13B** typically requires **65 GB of VRAM** → With QLoRA, it needs only **15-20 GB**.

2. Cost Efficiency:

- **Full fine-tuning of a 65B model** can cost **thousands of dollars** in cloud compute.
- With QLoRA, you can fine-tune the same model on a **single A100 or even consumer GPUs**.

3. Competitive Performance:

- QLoRA achieves **near full fine-tuning quality** despite using fewer parameters and lower precision.
- In practice, QLoRA fine-tuned LLaMA models have achieved **comparable performance** to fully fine-tuned models on many NLP benchmarks.