AMRITA
VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

श्रद्धावान् लभते ज्ञानम्

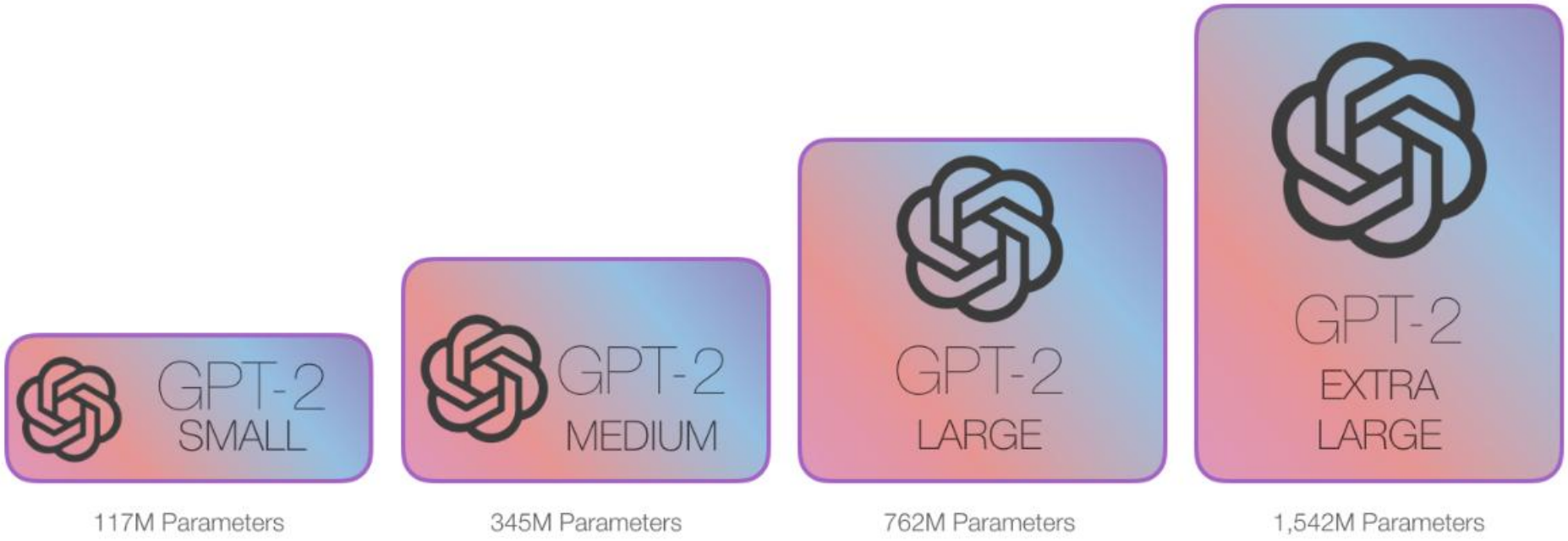
GPT

Generative Pretrained Transformer (GPT)- Decoder Only model

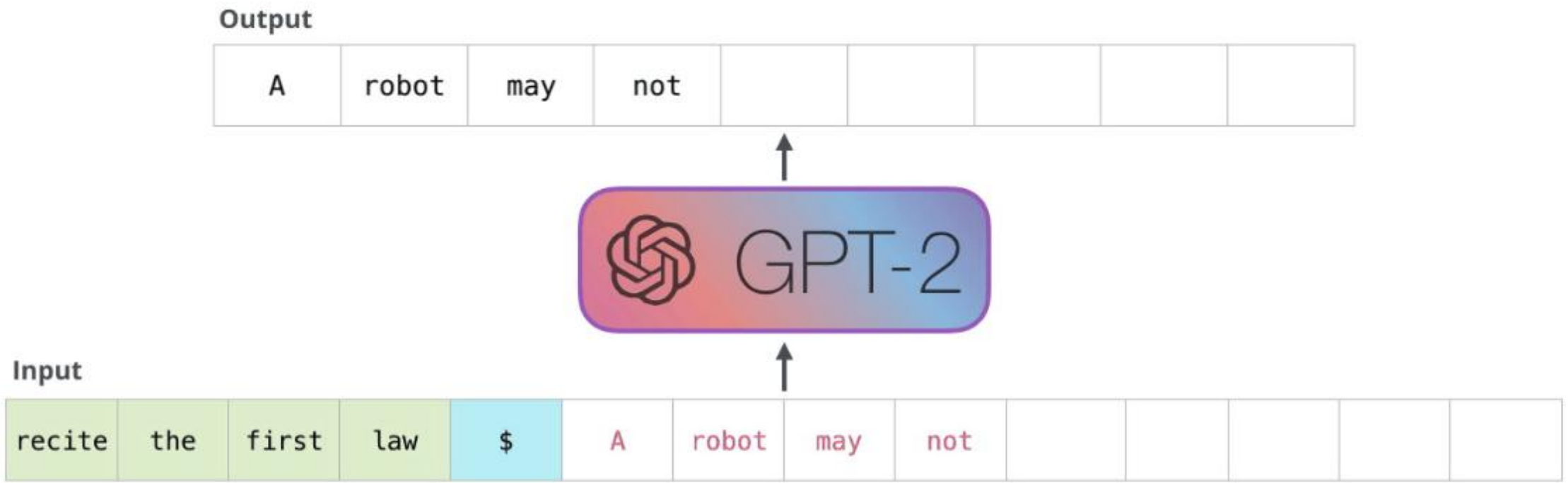
Courtesy:, analytics vidya,fast.ai, coursera: AndrewNG, http://jalammar.github.io/, ai4 bharat

GPT-2

GPT-2 was trained on a massive 40GB dataset called WebText that the OpenAI researchers crawled from the internet as part of the research effort. The smallest variant of the trained GPT-2, takes up 500MBs of storage to store all of its parameters. The largest GPT-2 variant is 13 times the size so it could take up more than 6.5 GBs of storage space.



The GPT2, and some later models like TransformerXL and XLNet are auto-regressive in nature. BERT is not. That is a trade off. In losing auto-regression, BERT gained the ability to incorporate the context on both sides of a word to gain better results. XLNet brings back autoregression while finding an alternative way to incorporate the context on both sides.



Language Modelling

Language modelling

Let $\mathcal{V}$ be a vocabulary of language (i.e., collection of all unique words in the language)

We can think of a sentence as a sequence $X_1, X_2, \cdots, X_n$, where $X_i \in \mathcal{V}$

For example, if $\mathcal{V} = \{an, apple, ate, I\}$, some possible sentences (not necessarily grammatically correct) are

a. An apple ate I

b. I ate an apple

c. I ate apple

d. an apple

e. ....

Intuitively, some of these sentences are more probable than others.

Language Modelling

If we naively assume that the words in a sequence are independent of each other then

$$P(x_1, x_2, \cdots, x_T) = \prod_{i=1}^{T} P(x_i)$$

$\prod_{i=1}^{T} P(x_i|x_1, \cdots, x_{i-1})$: Current word $\underbrace{x_i}$ depends on previous words $\underbrace{x_1, \cdots, x_{i-1}}$

How do we estimate these conditional probabilities?

One solution: use **autoregressive models** where the conditional probabilities are given by parameterized functions with a fixed number of parameters (like transformers).

Language Modelling

Intuitively, we mean that give a very very large corpus, we expect some of these sentences to appear more frequently than others (hence, more probable)

We are now looking for a function which takes a sequence as input and assigns a probability to each sequence

$$f : (X_1, X_2, \cdots X_n) \rightarrow [0, 1]$$

Such a function is called a language model.

$$P(x_1, x_2, \cdots, x_T) = P(x_1)P(x_2|x_1)P(x_3|x_2, x_1) \cdots P(x_T|x_{T-1}, \cdots, x_1)$$

$$= \prod_{i=1}^{T} P(x_i|x_1, \cdots, x_{i-1})$$

How do we enable a model to understand language?

Simple Idea: Teach it the task of predicting the next token in a sequence..

You have tons of sequences available on the web which you can use as training data

Chandrayaan-3

Article Talk

From Wikipedia, the free encyclopedia

Chandrayaan-3 (/ ˌtʃʌndrəˈjɑːn/ *CHUN-drə-YAHN*) is the third mission in the Chandrayaan programme, a series of lunar-exploration missions [redacted] by the Indian Space Research Organisation (ISRO).[7] Launched on 14 July 2023, the mission consists of a [redacted] named *Vikram* and a lunar rover named *Pragyan*, similar to those launched aboard Chandrayaan-2 in 2019.

Chandrayaan-3 was [redacted] from Satish Dhawan Space Centre on 14 July 2023. The spacecraft [redacted] lunar orbit on 5 August, and the lander [redacted] down near the Lunar south pole[8] on 23 August at 18:03 IST (12:33 UTC), making India the fourth country to [redacted] land on the Moon, and the first to do so near the lunar south [redacted]. On 3 September the lander hopped and repositioned itself 30–40 cm (12–16 in) from its landing site.[13] [redacted] the completion of its mission objectives, it was hoped that the lander and rover would [redacted] for extra tasks, on 22 September 2023,

BECAUSE he is the [redacted] in conceiving and implementing India's digital building blocks such as UPI, Aadhaar, eKYC and FASTag. [redacted] also helped the government develop the IT infrastructure to [redacted] GST and the Ayushman Bharat Yojana. The cerebral [redacted] is the government's go-to person for tech intervention, irrespective of the party in power

BECAUSE he was [redacted] in the launch of the Beckn protocol, now being used in sectors such as e-commerce, mobility and health. He was [redacted] the government-backed Open Network for Digital [redacted] aimed at helping small Indian retailers fight back against e-com giants such as [redacted] and Flipkart. He is also supporting IIT Madras's AI4Bharat, an [redacted] AI initiative, in 22 Indian languages

Decoder only Model — Causal Language Modelling

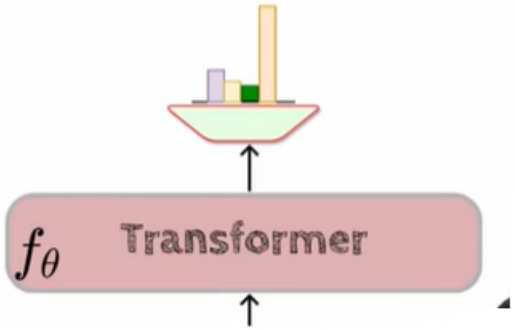
$$P(x_1, x_2, \cdots, x_T) = \prod_{i=1}^{T} P(x_i | x_1, \cdots, x_{i-1})$$

$$= P(x_1)P(x_2|x_1)P(x_3|x_2, x_1) \cdots P(x_T|x_{T-1}, \cdots, x_1)$$

We are looking for f_θ such that

$$P(x_i|x_1, \cdots, x_{i-1}) = f_\theta(x_i|x_1, \cdots, x_{i-1})$$

Can f_θ be a transformer?



Decoder only Model — Causal Language Modelling

Now we can create a stack (n) of modified decoder layers (called transformer block in the paper)

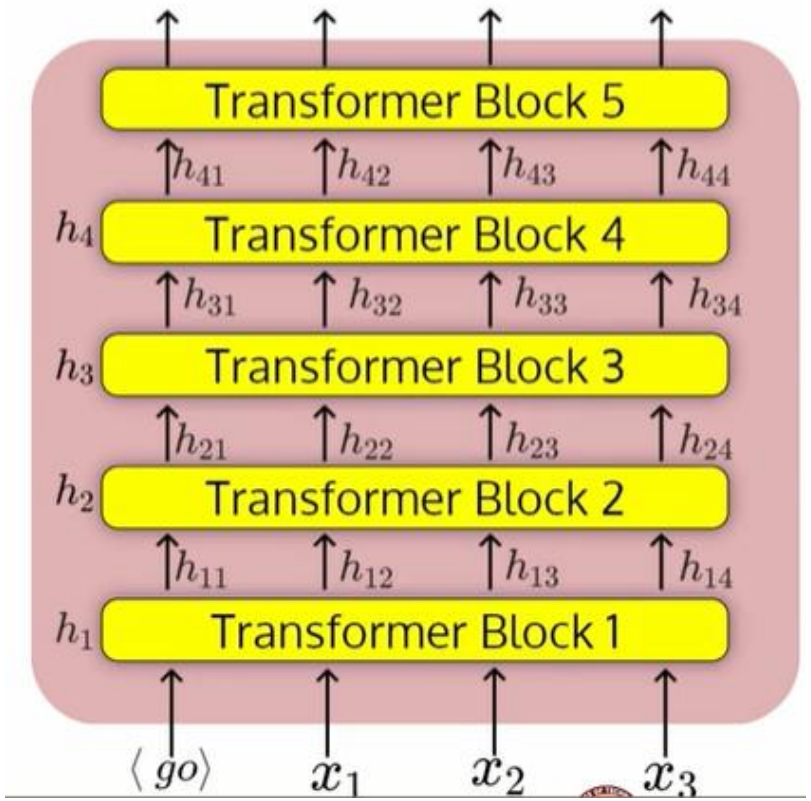
Let, X denote the input sequence

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

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

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

$$P(x_i) = softmax(h_n[i]W_v)$$



- Data that is trained on? **Pretraining** **GPT-2**
- Architecture — How many dimensions? How many attention heads? Compute total number of parameters?

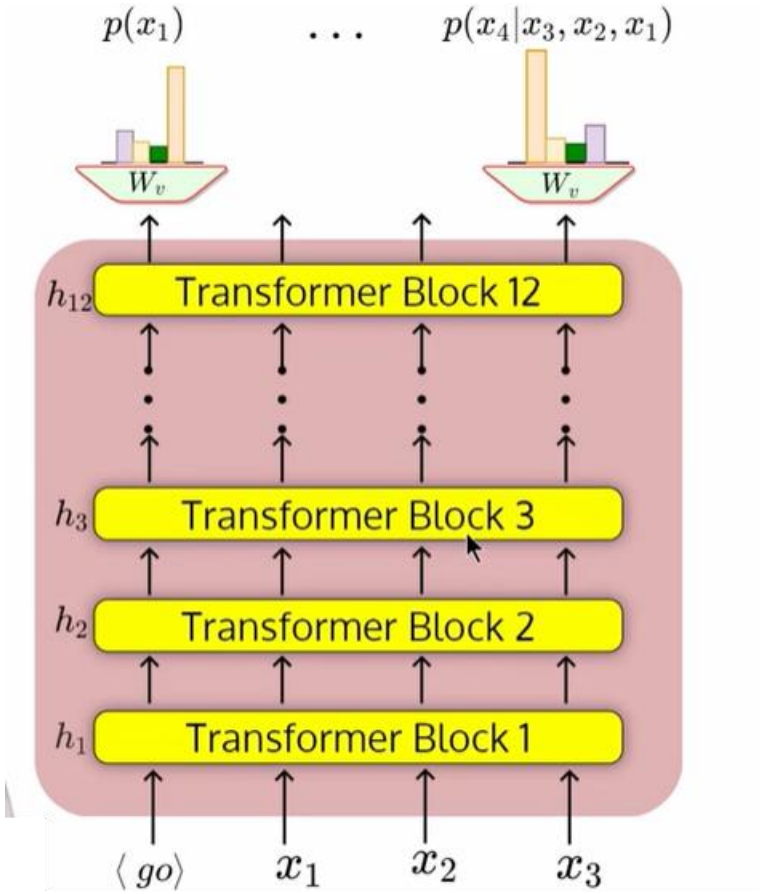
BookCorpus

- Which contains 7000 unique books, 74 Million sentences and approximately 1 Billion words across 16 genres.
- Also it uses long-range contiguous text (i.e., no shuffling of sentences or paragraphs).
- The sequence length 'T' is 512 tokens or context size and it is contiguous.
- It has a tokenizer which is Byte Pair Encoding.
- Vocabulary size of 40478 unqiue words.
- Embedding dimension is 768.

GPT-2

- Contains 12 decoder layers (transformer blocks)
- Context size : 512 → 512 inputs/Tokens at one go (T)
- Attention heads : 12
- Feed Forward Network layer size : 768 x 4 times = 3072
- **Activation :** Gaussian Error Linear Unit (GELU)

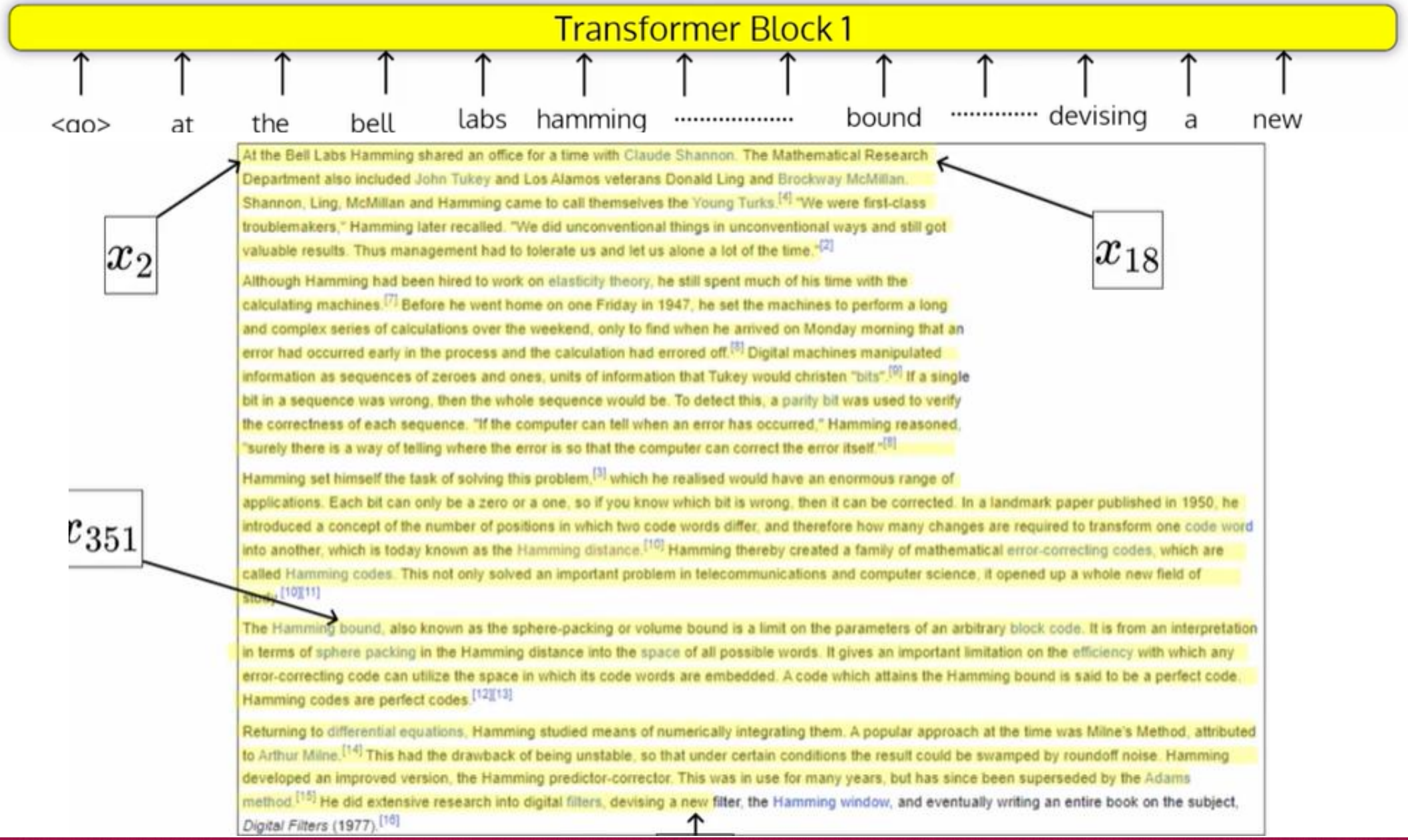
Drop out , layer normalisation , residual connection for faster convergence



Model dimension size(dmodel) is 768 (64x12)

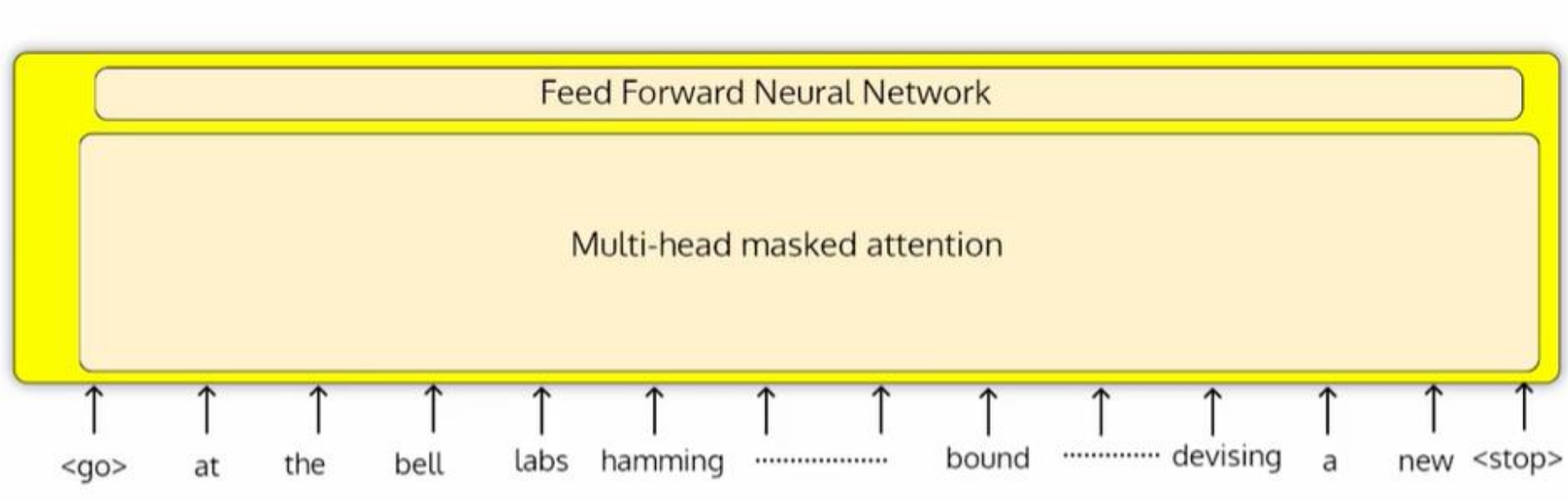
dmodel/#heads = 768/12 = 64

512 No of tokens each of 768 dimension

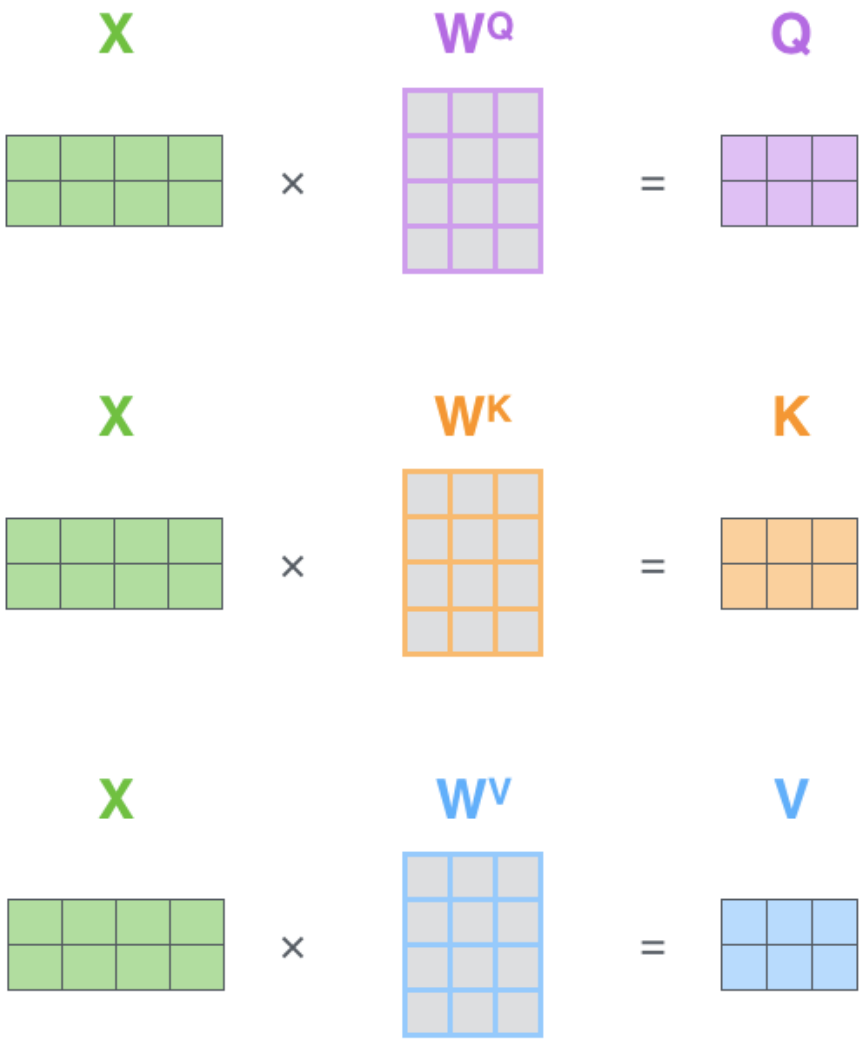

Transformer Block 1
<go>
at
the
bell
labs
hamming
bound
devising
a
new
x_2
x_{18}
x_{351}
At the Bell Labs Hamming shared an office for a time with Claude Shannon. The Mathematical Research Department also included John Tukey and Los Alamos veterans Donald Ling and Brockway McMillan. Shannon, Ling, McMillan 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]

Masked Multi head Attention

Let's look at how multi-head masked attention looks like. The first head will take all these 512 tokens and it will generate Q, K, V vectors and do this entire operation until MatMul and again gives 512 vectors. The size of each vector will be dmodel/#heads = 768/12 = 64. Hence it produces each of individual head output and then concatenate to get the 768 output size.



Matrix Calculation of Self-Attention – Recap Transfomers



$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

X512x768 . Y768x64--- 512x64

“Multi-headed” attention

1) This is our input sentence*

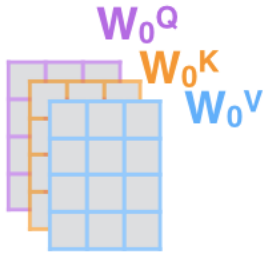
2) We embed each word*

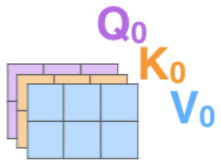
3) Split into 8 heads. We multiply X or R with weight matrices

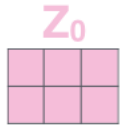
4) Calculate attention using the resulting Q/K/V matrices

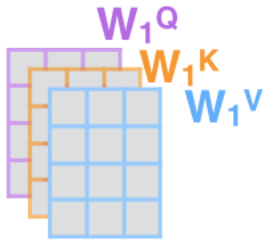
5) Concatenate the resulting Z matrices, then multiply with weight matrix W^O to produce the output of the layer

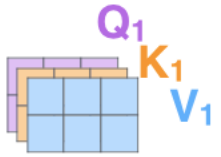
Thinking Machines

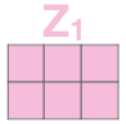




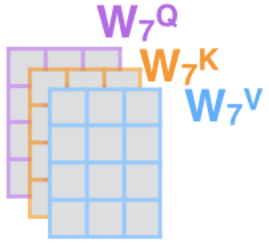


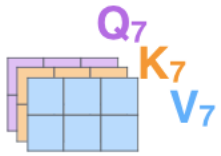


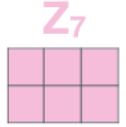


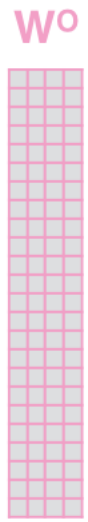


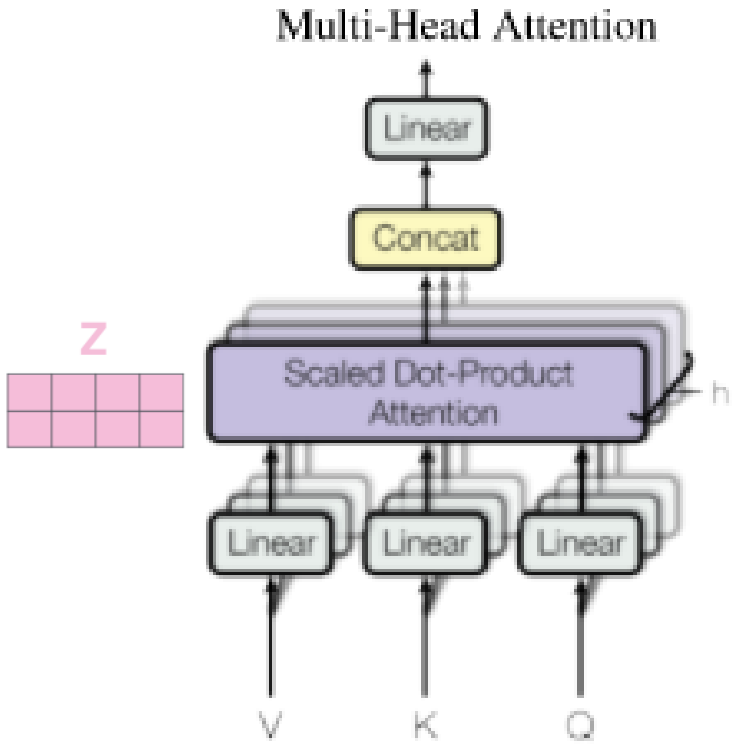
* In all encoders other than #0, we don’t need embedding. We start directly with the output of the encoder right below this one





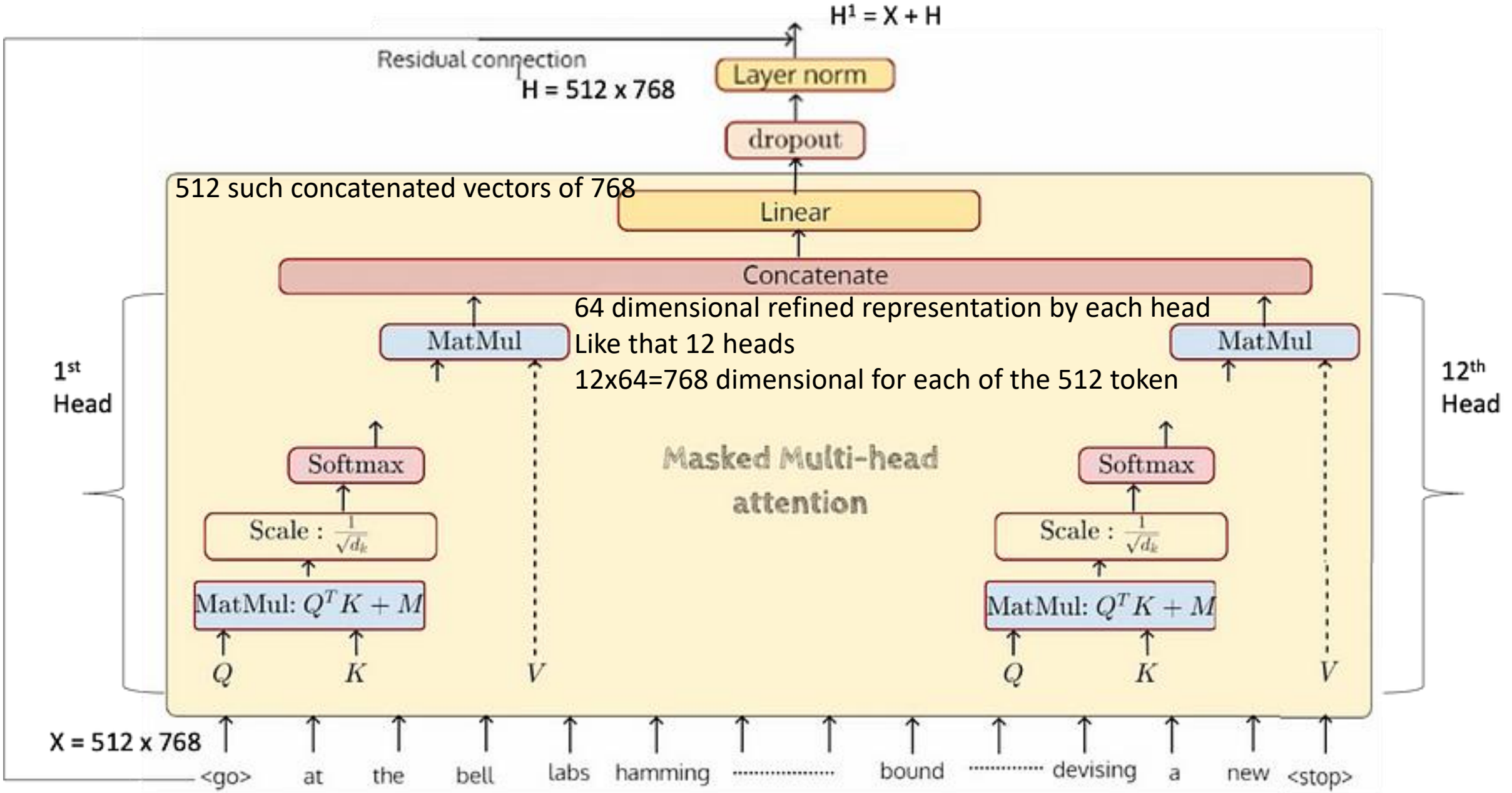


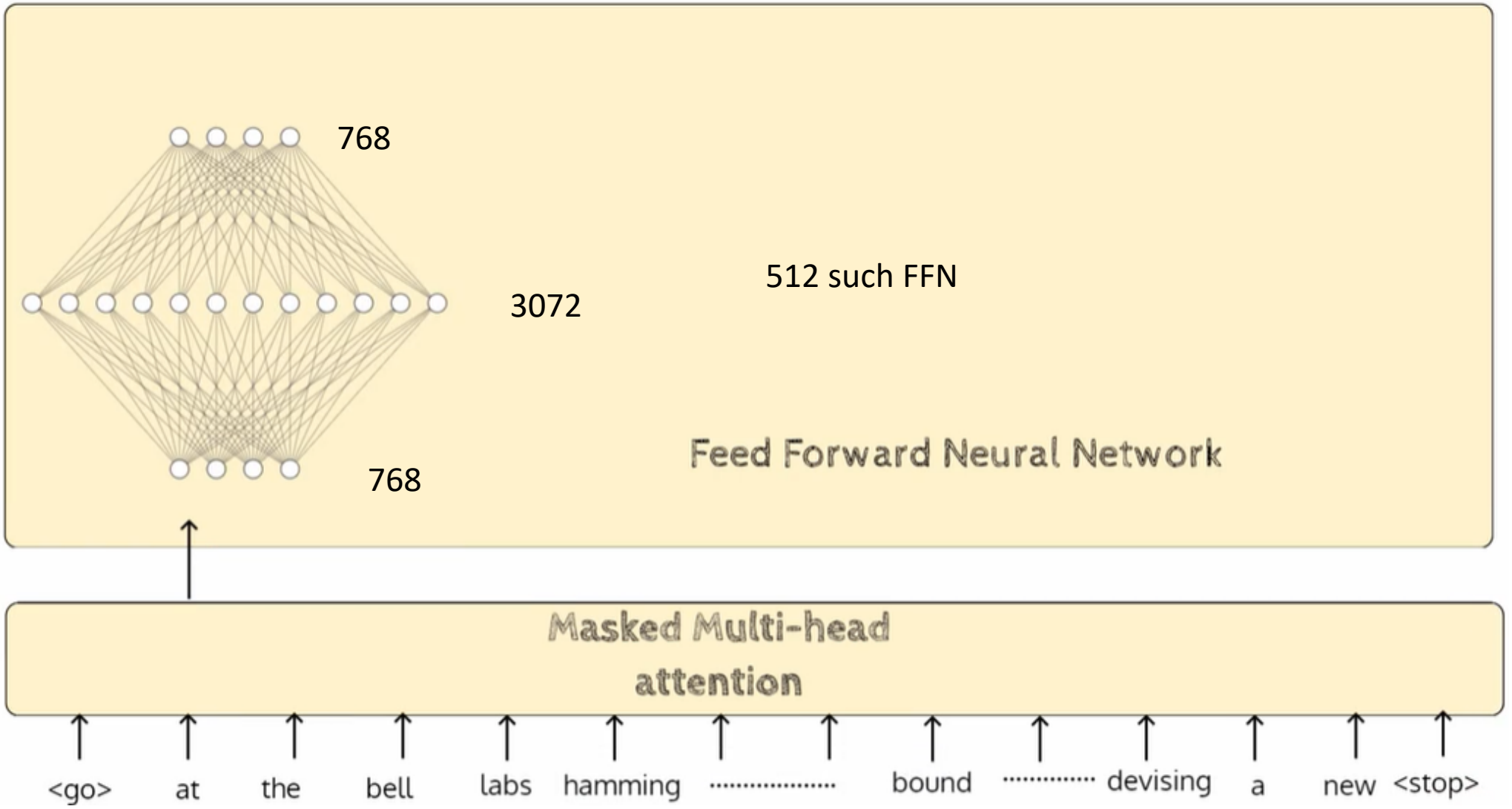




multiple “representation subspaces”

multiple sets of Query/Key/Value weight matrices


$H^1 = X + H$
Residual connection
H = 512 x 768
Layer norm
dropout
512 such concatenated vectors of 768
Linear
Concatenate
64 dimensional refined representation by each head
Like that 12 heads
12x64=768 dimensional for each of the 512 token
MatMul
MatMul
1st Head
12th Head
Softmax
Scale : $\frac{1}{\sqrt{d_k}}$
MatMul: $Q^T K + M$
Masked Multi-head attention
Softmax
Scale : $\frac{1}{\sqrt{d_k}}$
MatMul: $Q^T K + M$
Q
K
V
Q
K
V
X = 512 x 768
<go>
at
the
bell
labs
hamming
.....
bound
.....
devising
a
new
<stop>


768
3072
768
512 such FFN
Feed Forward Neural Network
Masked Multi-head attention
<go>
at
the
bell
labs
hamming
bound
devising
a
new
<stop>

summary

No of inputs will be same as No of outputs and will be same across each layer. So we had a 512 x 768 input and it went to into the masked attention and gave 512 x 768 output. Each of those 768 output went into the linear transformation which kind of completes one cycle of masked multi head attention.At this point of time we got all 512 new representations each being of size 768.

On now all the 512 tokens of 768 size will pass into the FFN and produce 3072 intermediate vector size before the final 768 size vector is again produced. So likewise this will be done for all 512 tokens.

No: of parameters (Embedding)

token Embeddings: $|\mathcal{V}| \times$ embedding_dim

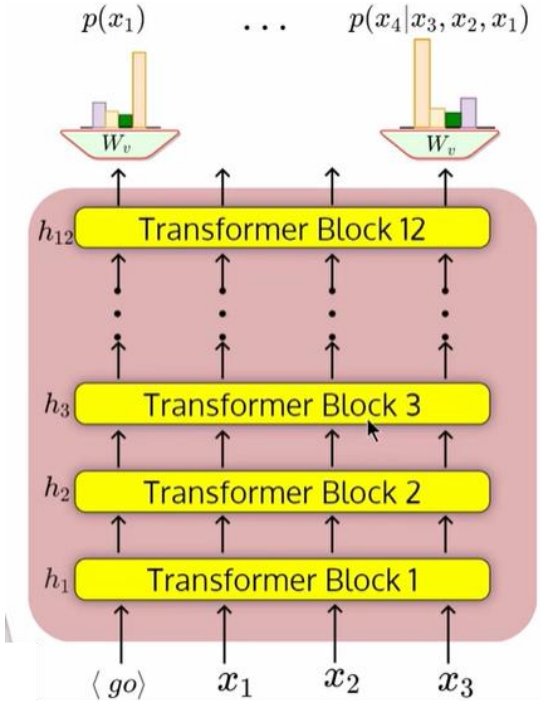
$$40478 \times 768 = 31 \times 10^6 = 31M$$

Position Embeddings : context length $\times$ embedding_dim

$$512 \times 768 = 0.3 \times 10^6 = 0.3M$$

Total: $31.3M$

- Taking a large piece of contiguous text which is 512 tokens
- Feeding it to the transformer layer and every layer we are again producing 512 outputs till the last layer
- At this last layer we are applying softmax to convert this into 512 probabilities which are P(x1), P(x2/x1), P(x3/x1,x2)…….P(x512/x1,x2,x3…x511)
- All of these predictions are happening at parallel and a mask is ensuring that we cannot see any of the future words or which are not allowed to see



Token embedding : 40478 (vocabulary size) * 768 (embedding dimension) = 31,021,184 parameters. These parameters represent the weights of the embedding layer, which are learned during the training process. They are responsible for mapping tokens from the discrete space of the vocabulary into continuous vector representations that the model can process and learn from.

No of Parameters (Attention blocks)

64x12=768

Attention parameters per block

$$W_Q = W_K = W_v = (768 \times 64)$$

Per attention head

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

For 12 heads

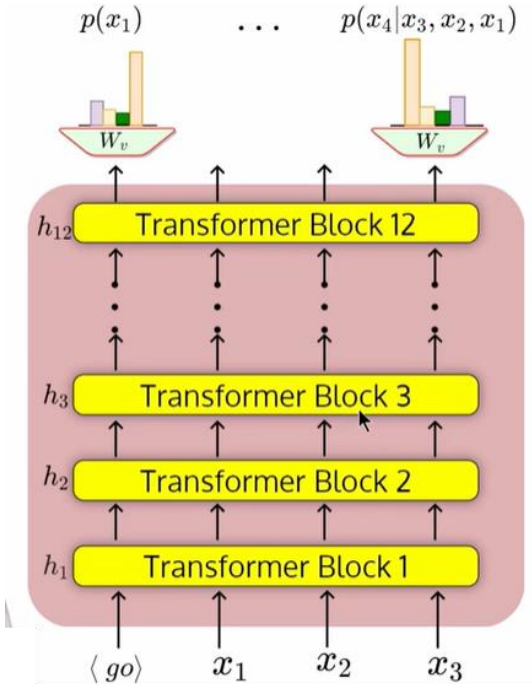
$$12 \times 147 \times 10^3 \approx 1.7M$$

For a Linear layer:

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

For all 12 blocks

$$12 \times 2.3 = 27.6M$$



For attention parameters we have 3 matrices Wq, Wk and Wv and each of them takes 768 dimensions input and converts into 64 dimension output so matrix will be 768 x 64 — this is per attention head and we have 3 such matrices so hence 3 x (768 x 64) ~ 147 x 10^3. So we have 12 such heads hence the parameters will be 12 x 147 x 10^3 ~ 1.7M. Then a linear layer is considered where we have 768 dimension which is a concatenated output (64 x 12) which is multiplied by Wo (768 x 768 dimension) gives the 768 vector — 768 x 768 ~ 0.6M. So finally for all 12 blocks it is ~27.6M parameters exist for masked multihead self attention summed up across all layers.

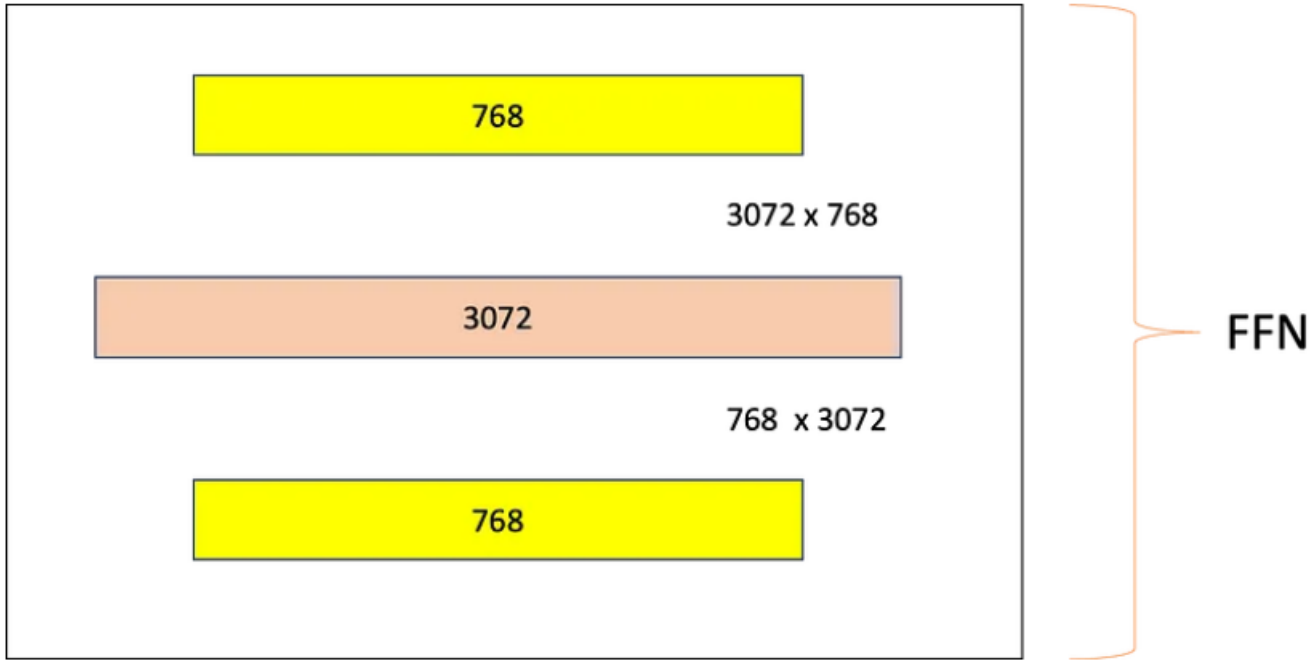
No of parameters (FFN)

FFN parameters per block

$$2 \times (768 \times 3072) + 3072 + 768$$
$$= 4.7 \times 10^6 = 4.7M$$

For all 12 blocks

$$12 \times 4.7 = 56.4M$$



Bias of 3072 and 768 respectively along with 2 x (768 x 3072) will give finally the FFN parameters which summed up to 4.7M for one block. Hence when done for all 12 blocks it is 12 x 4.7 ~ 56.4M

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

Thus, GPT-1 has around 117 million parameters.

Pre-training

Pre-training is about when there is no supervision or explicit supervision and we get automatic supervision from large unlabeled corpus where every next token is the label that we need to predict.

Batch size : 64

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

Optimizer : Adam with cosine learning rate scheduler

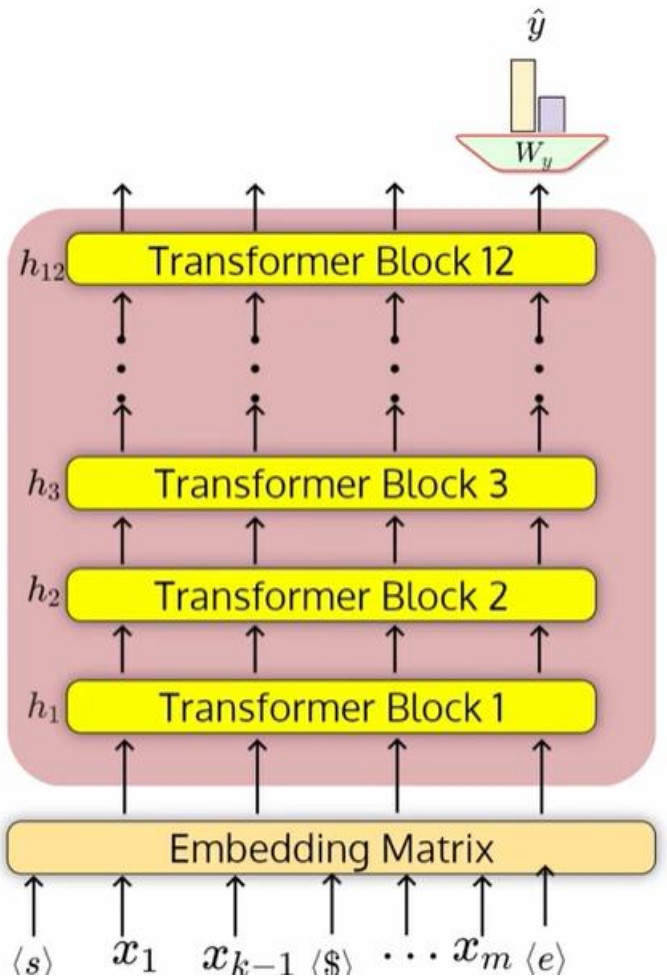
Strategy : Teacher forcing (instead of auto-regressive training) for qucker and stable convergence

Then we can minimize the following objective

$$\mathcal{L} = -\sum_{i=1}^{T-1} \log \hat{y}_i[x_{i+1}]$$

Now our objective is to predict the label of the input sequence

$$\hat{y} = P(y|x_1, \cdots, x_m) = softmax(W_y h_l^m)$$



Fine tuning (Textual Entailment)

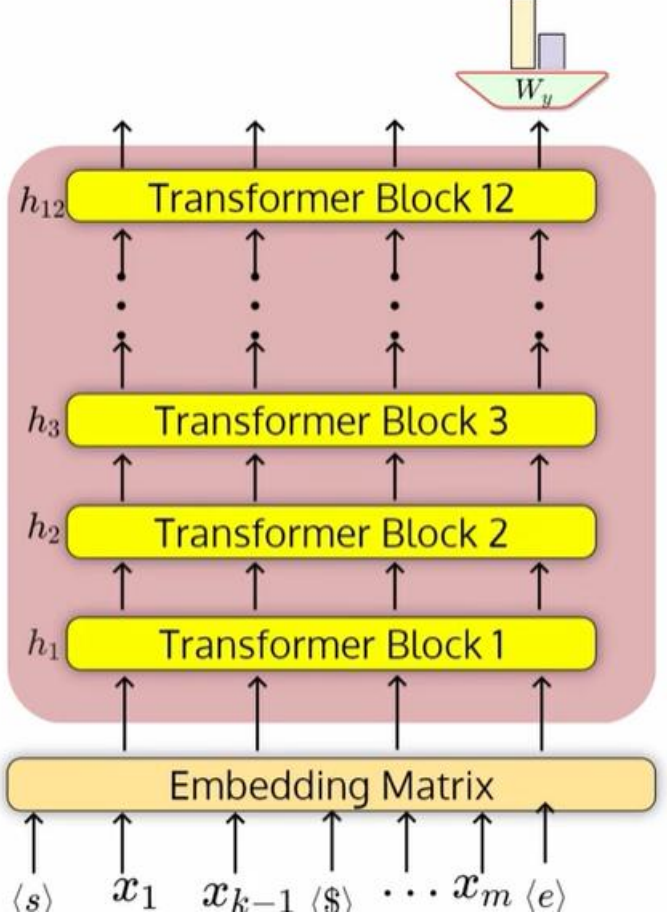
Textual Entailement/Contradiction

Text: A soccer game with multiple males playing

Hypothesis: Some men are playing a sport

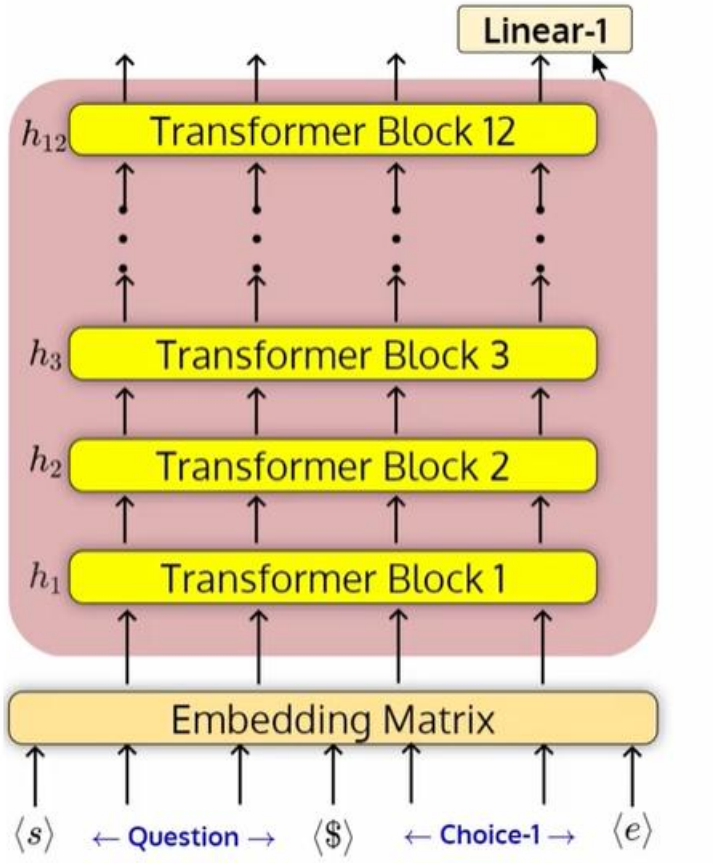
Entailment: True

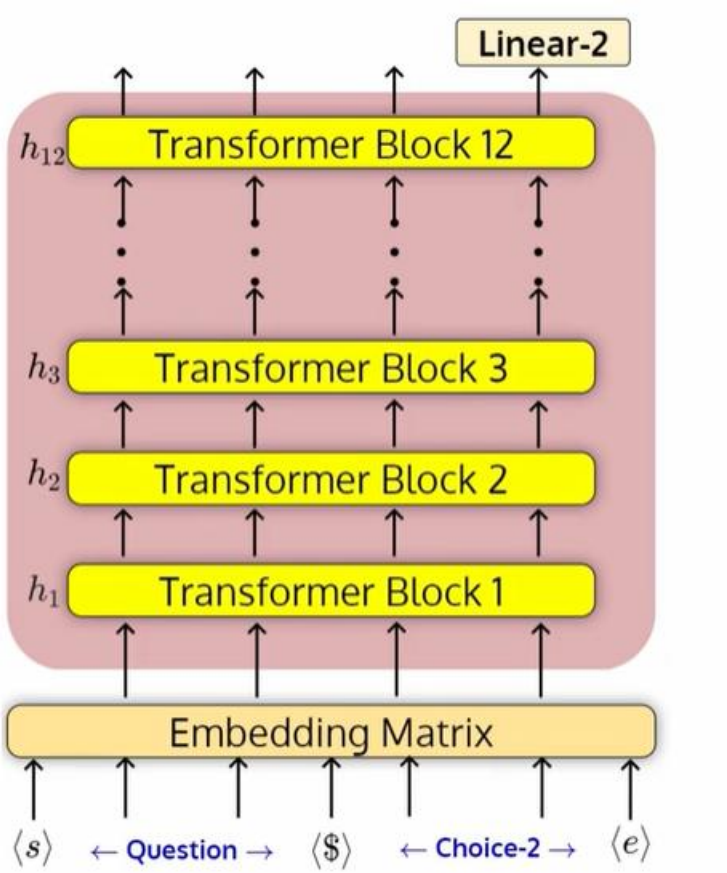
In this case, we need to use a delimiter token ⟨$⟩ to differentiate the text from the hypothesis.



Textual entailment is a natural language processing (NLP) task that involves determining the logical relationship between two text fragments, typically referred to as the "premise" and the "hypothesis." The task is to determine whether the meaning of the hypothesis can be inferred (or entailed) from the premise.

Multiple choice (Fine tuning)





Question: Which of the following animals is an amphibian?

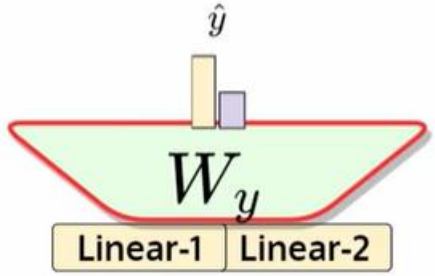
Choice: Frog

Choice: Fish

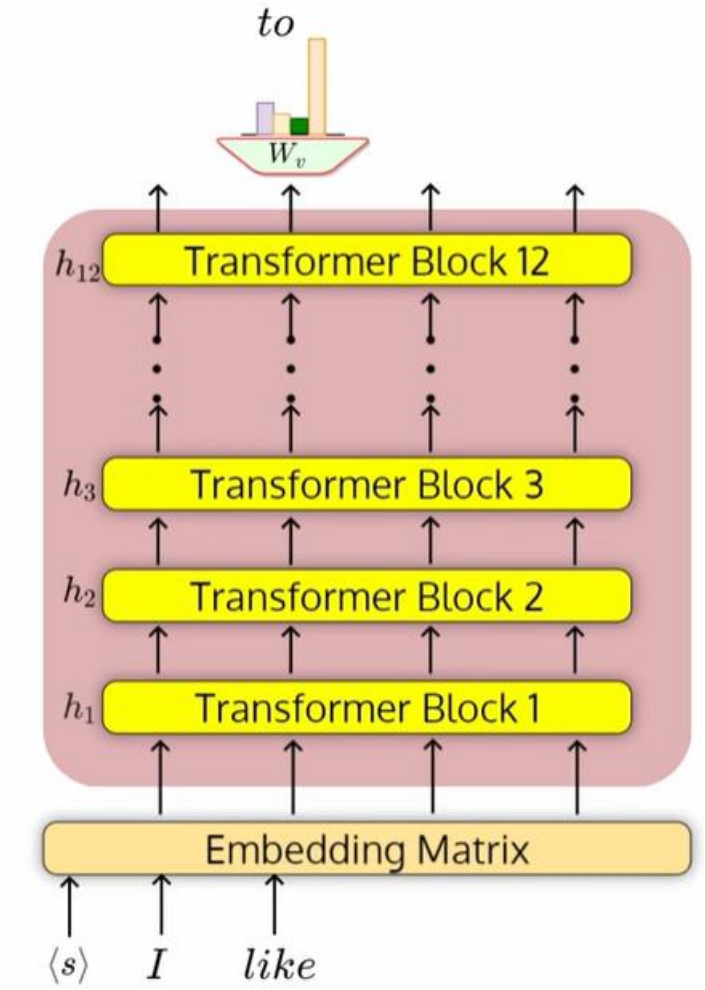
Feed in the question along with the choice-2

Repeat this for all choices

Normalize via softmax



Fine tuning (Text Generation)



Input:

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 autregressive mode

Stoping Criteria:

Sequence length: 5

or outputing a token: ‹e›

Output: I like to think that

Does it produce the same output sequence for the given prompt?

What is our wishlist

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

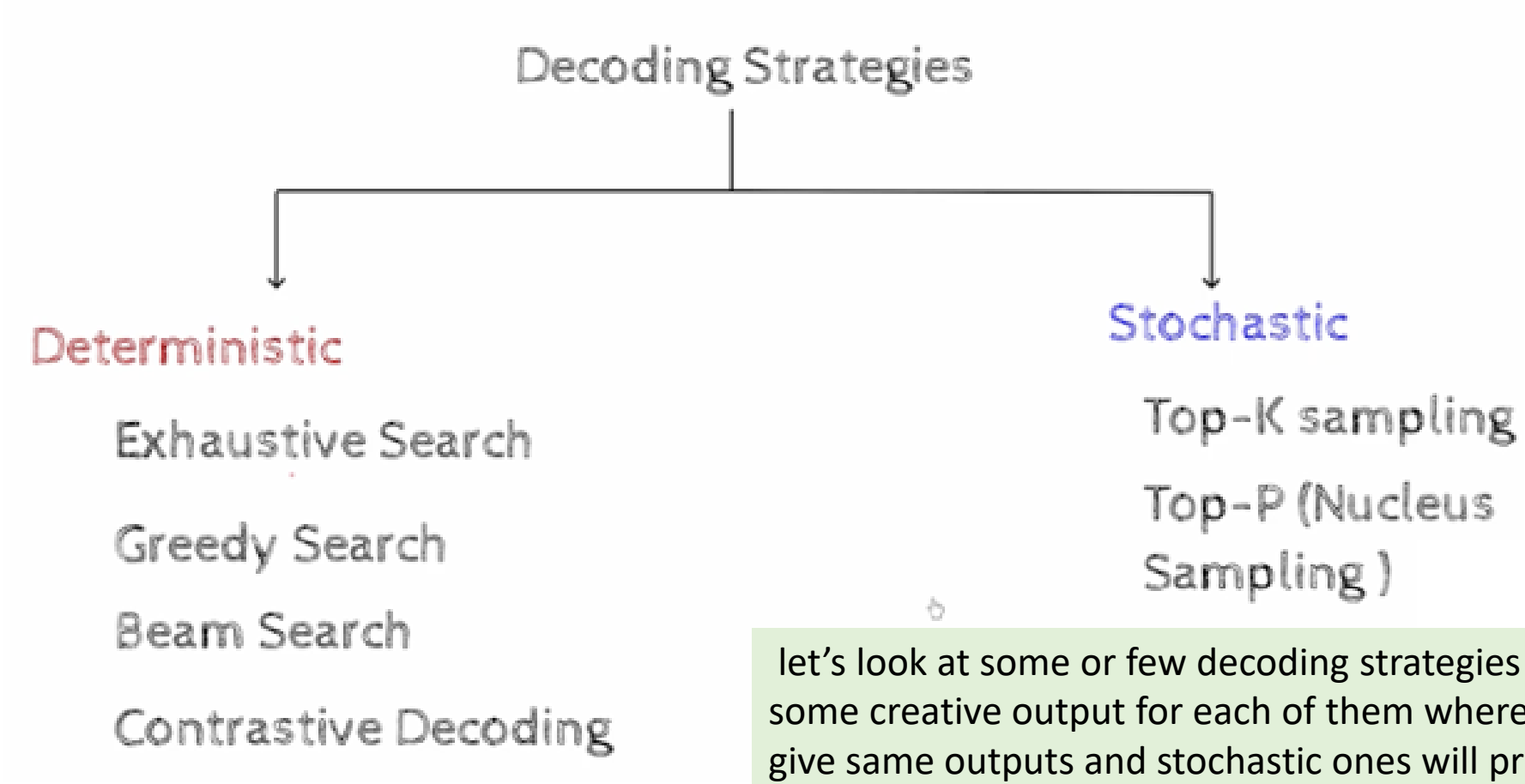
I like to read a book

I like to buy a hot beverage

I like a person who cares about others

Accelarate the generation of tokens

How to generate such creative text



let's look at some or few decoding strategies where we have some creative output for each of them where deterministic will give same outputs and stochastic ones will produce different outputs.

Exhaustive Search:

Suppose that we want to generate a sequence of 5 words with the vocabulary { cold, coffee, I , like , water, <stop>}

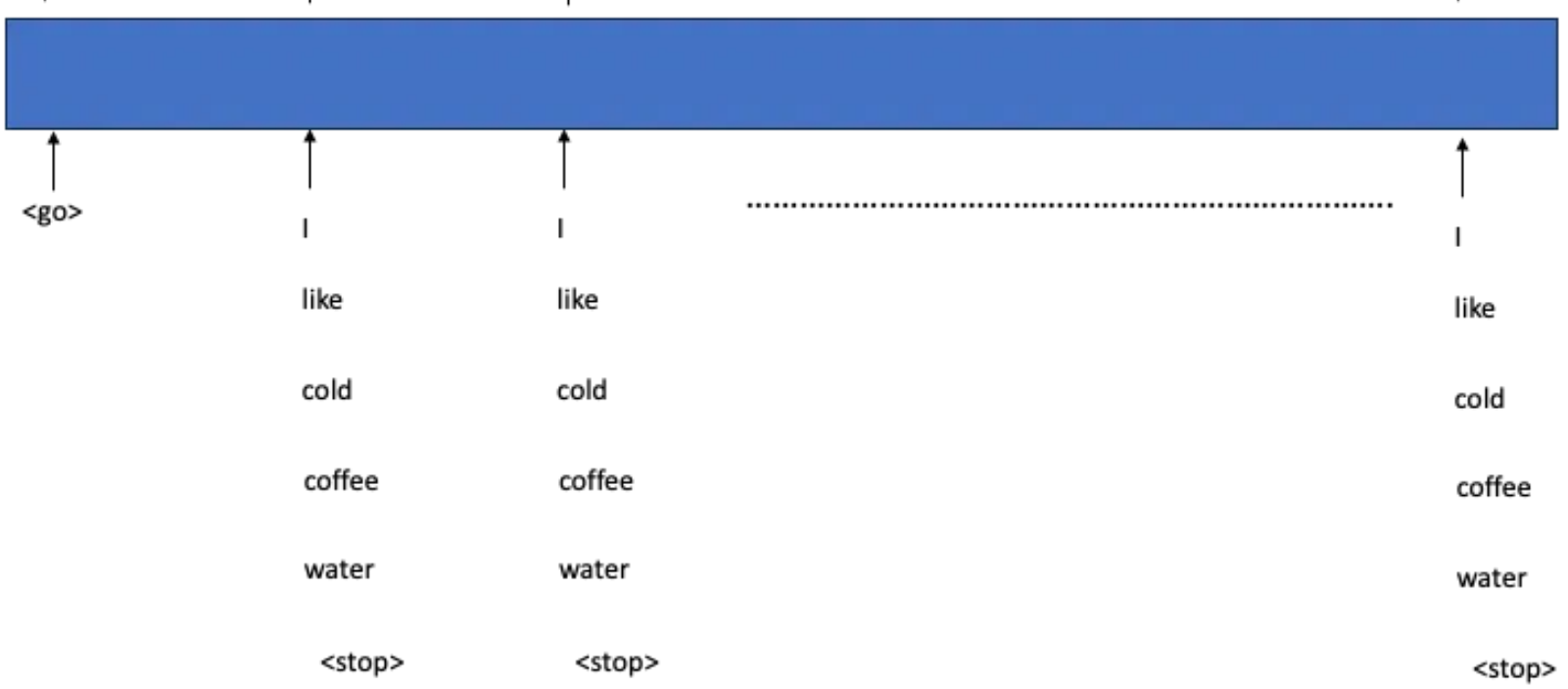
Exhaustive search for all possible sequences with the associated probabilities and output the sequence with the highest probability.

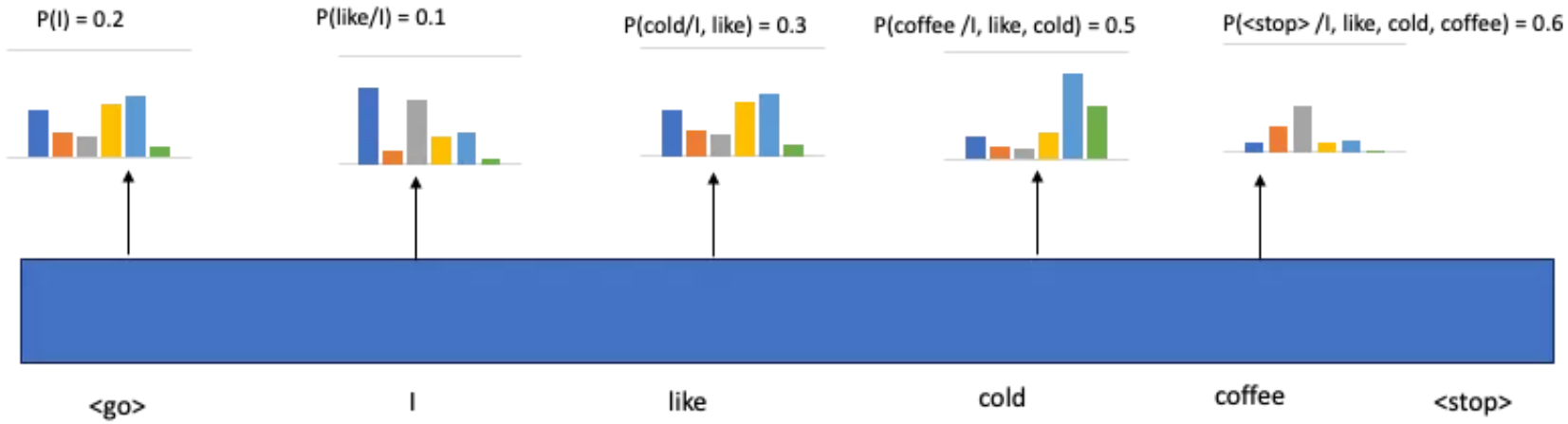
- *I like cold water*
- *I like cold coffee*
- *coffee like cold coffee*
- *I like I like*
- *coffee coffee coffee coffee*

So for each of the sentence output the probability will be

$$P(x_1, x_2, \cdots, x_T) = P(x_1)P(x_2|x_1)P(x_3|x_2, x_1)\cdots P(x_T|x_{T-1}, \cdots, x_1)$$

$$= \prod_{i=1}^{T} P(x_i|x_1, \cdots, x_{i-1})$$


<go>
I
like
cold
coffee
water
<stop>
I
like
cold
coffee
water
<stop>
I
like
cold
coffee
water
<stop>


P(I) = 0.2
P(like/I) = 0.1
P(cold/I, like) = 0.3
P(coffee /I, like, cold) = 0.5
P(<stop> /I, like, cold, coffee) = 0.6
<go>
I
like
cold
coffee
<stop>

Exhaustive Search:

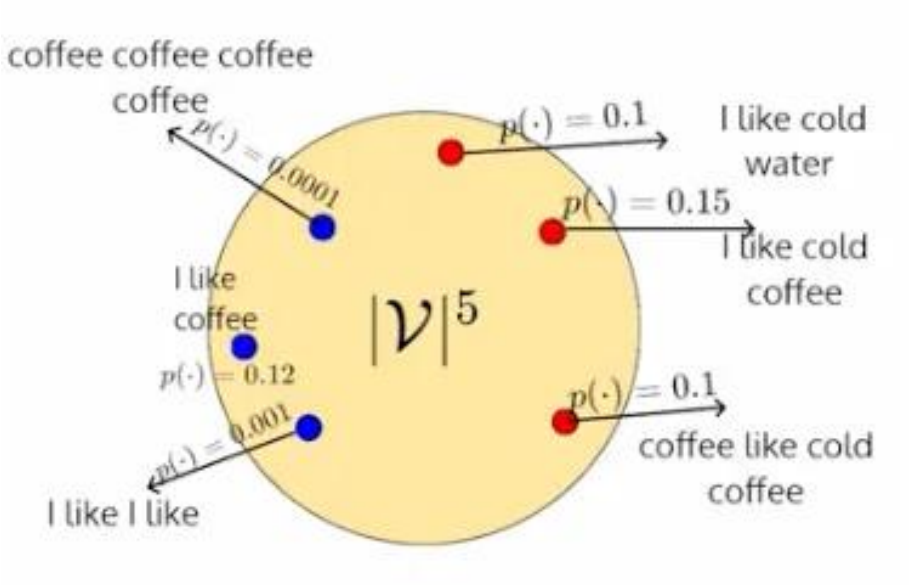
The total probability for the above sequence will be equal to

$$P(I) * P(like/I)*P(cold/I,like)*P(coffee/I,like,cold)$$

Similarly other combinations of sequences also will follow in the same pattern as above and gives us the output with maximum probabilty —

this probability calculation is done with all the tokens at every time step.

So based on the above exhaustive search let us assume these are the probabilities in the search space



Exhaustive Search:

of these 9 possibilities whichever has the maximum probability it gives that output at timestep =2. If we had timestep = 3 then we will have 27 sequences with probabilities and we take the maximum score with all those 27 sequences.

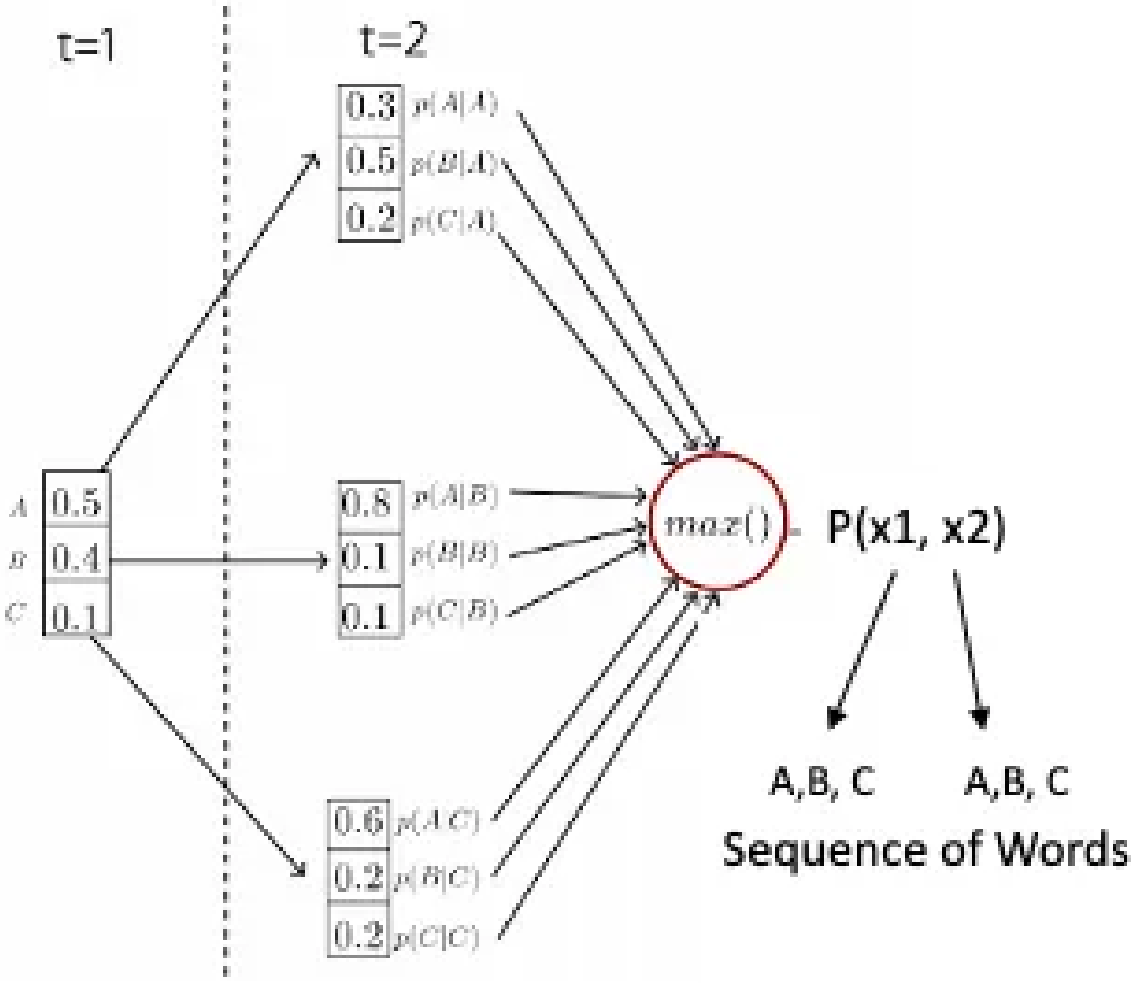
Illustration

Exhaustive search for a sequence of length 2 with the vocabulary of size 3

At time step-1, the decoder outputs probability for all 3 tokens

At time step-2, we need to run the decoder three times independently conditioned on all the three predictions from the previous time step

At time step-3 we will have to run the decoders 9 times



Exhaustive Search:

Assuming the sequence has the highest probability among all $|v|^5$ sequences — in this case above if “I like cold coffee ” sequence is generated as the highest probability then the outcome will be highlighted

- *I like cold water*
- *I like cold coffee*
- *coffee like cold coffee*
- *I like I like*
- *coffee coffee coffee coffee*

Exhaustive Search:

With this exhaustive search , no matter how many times we calculate this — we are going to get the same answer for the given same input, there is no creative output we can see. This falls under deterministic strategy.

Greedy Search:

With greedy search — at each time step we always output the token with the highest probability (Greedy)

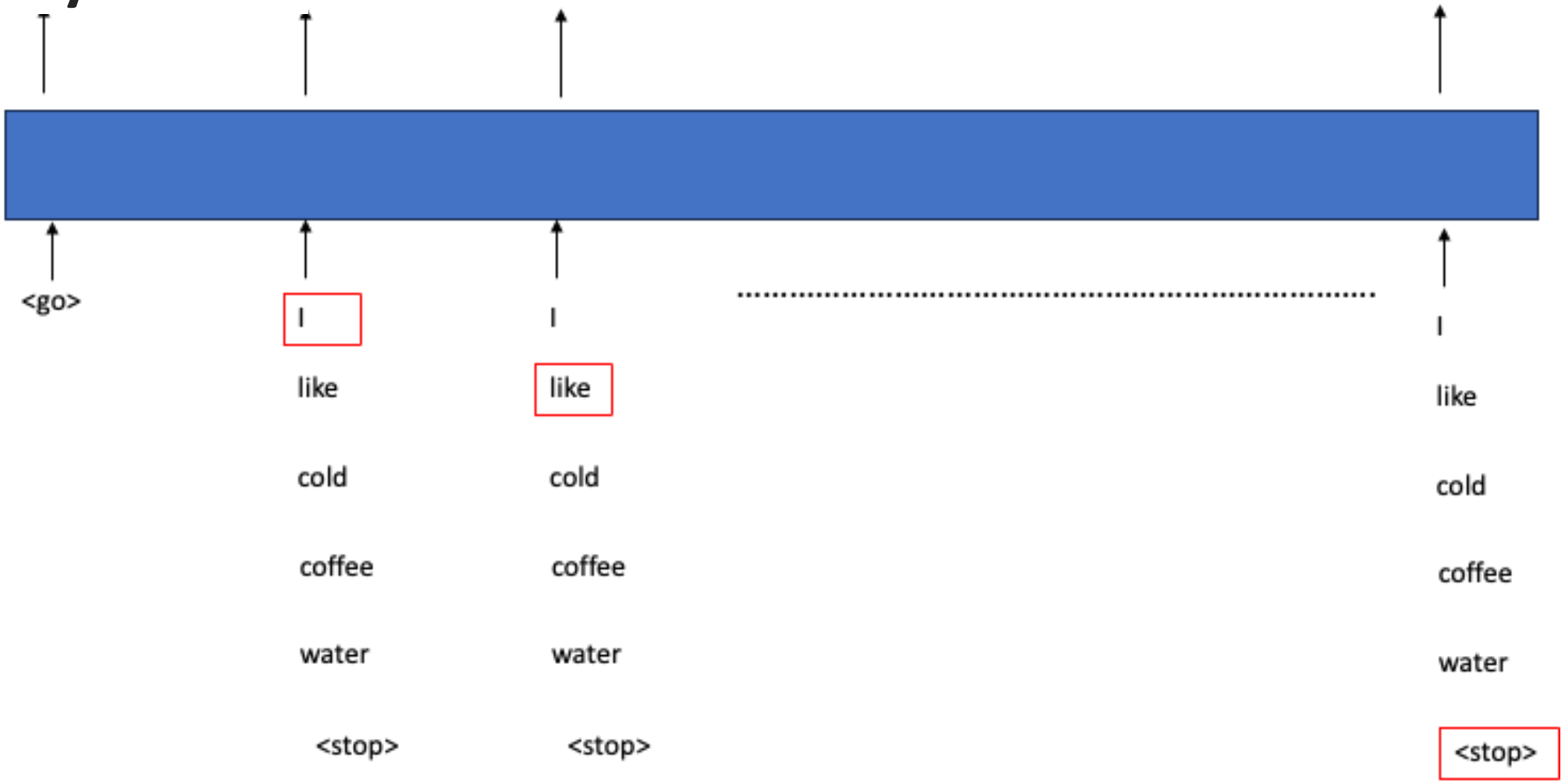
p(w2 = like|w1=I) = 0.35
p(w3= cold | w1,w2) = 0.45
p(w4 = coffee |w1,w2,w3) = 0.35
p(w5 = stop | w1, w2, s3, s4) = 0.5

Then the probability of the generated sequence is

p(w5,w1,w2,w3,w4) = 0.5*0.35*0.45*0.35*0.5 = 0.011

	time steps				
	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>

Greedy Search:



Greedy Search:

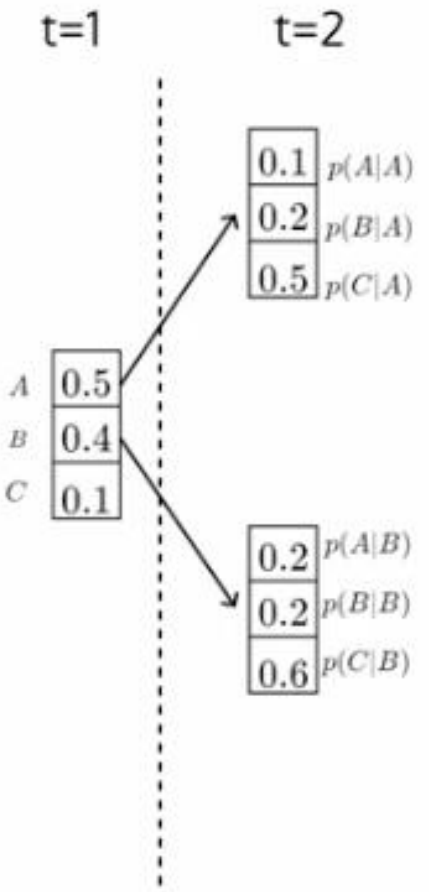
Some limitations!

- Is this the most likely sequence?
- What if we want to get a variety of sequence of the same length?
- If the starting token is the word “I”, then it will always end up producing the same sequence: *I like cold coffee*.
- ***What if we picked the second most probable token in the first time step?***
  - Then the conditional distribution in the subsequent time steps will change. Then the probability of the generated sequence is
  - p(w5,w1,w2,w3,w4) = 0.25*0.55*0.65*0.8*0.5 = 0.035

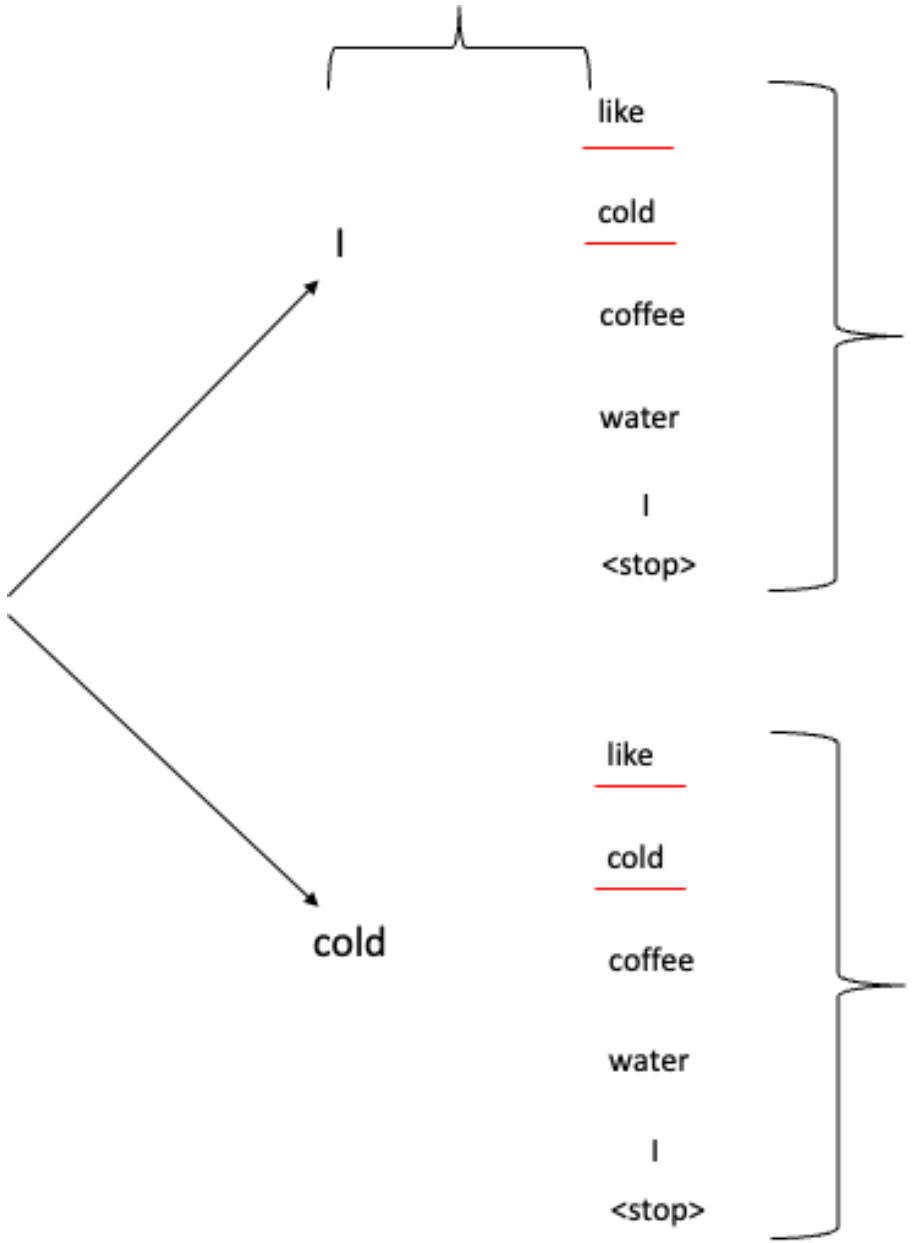
Beam search:

Instead of considering probability for all the tokens at every time step (as in exhaustive search) , consider only top-k tokens

Suppose (k=2), at timestep = 1 we have two tokens with probability 0.5 and 0.4 and in time step 2 we will be having 6 such sequences.



Beam search:

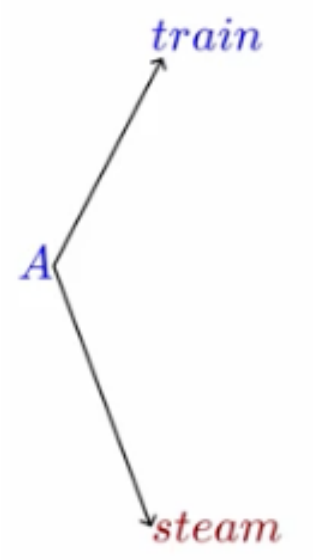


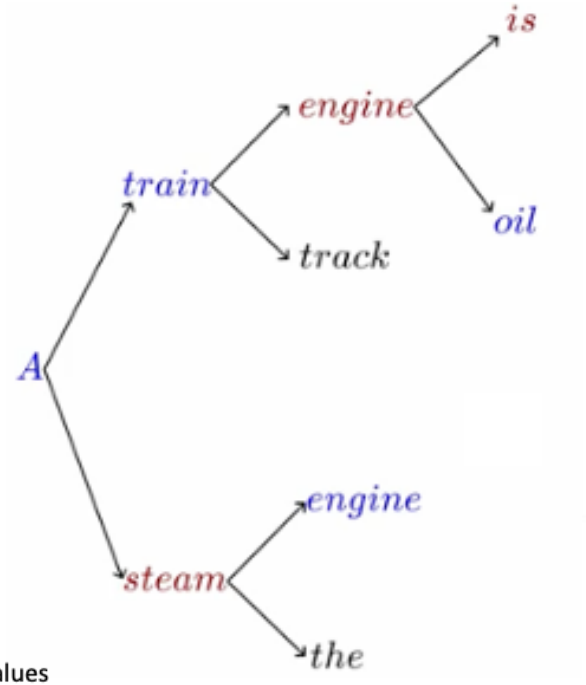
Assuming the top 2 sequences with high probabilities and we will be having 12 such sequences at timestep = 2

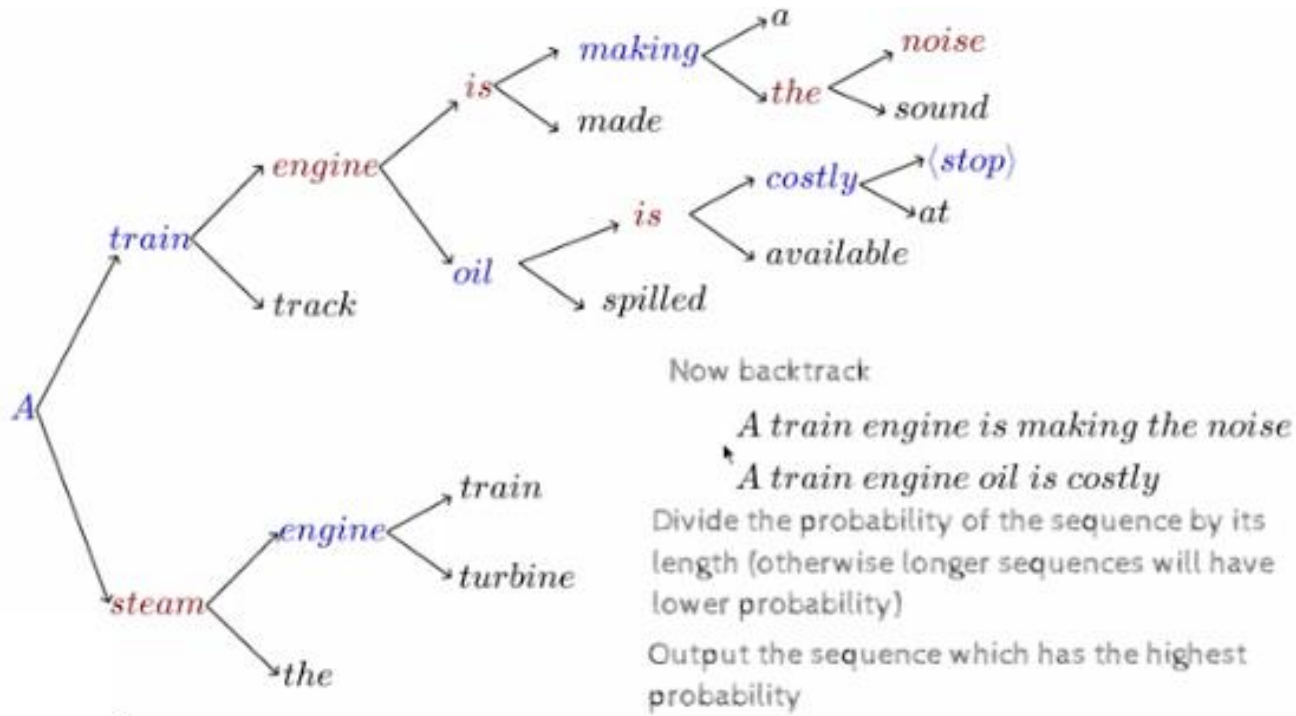
Suppose (k=2), at timestep = 2 we have two tokens with probability for e.g., ***I , cold*** and we will be having 12 such sequences.

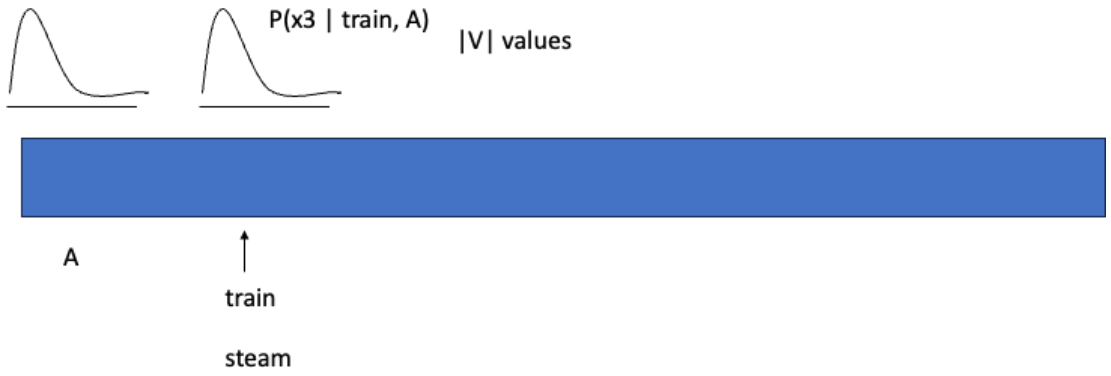
Now we have to choose the tokens that maximize the probability of the sequence. It requires k x |v| computations at each timestep. At second timestep we had 2 x 6=12 computations are then ranked and we select the highest probability sequence.

K=2 and tokens vocabulary is |v|.









The parameter k is called beam size. It is an approximation to exhaustive search. If k = 1 then it is equal to greedy search. If k > 1 then we are doing beam search and if k = V then we are doing exhaustive search.

Limitations : Beam Search

- Both the greedy search and the beam search are prone to be degenerative i.e., they can be repetitive without any creativity.
- 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.

Basically we need some surprises with creative answers or outputs — hence there we need some sampling based strategies without the greedy or beam search.

Sampling Strategies — Top –K (Stochastic)

Sample a token from the top-k tokens. Say k = 2

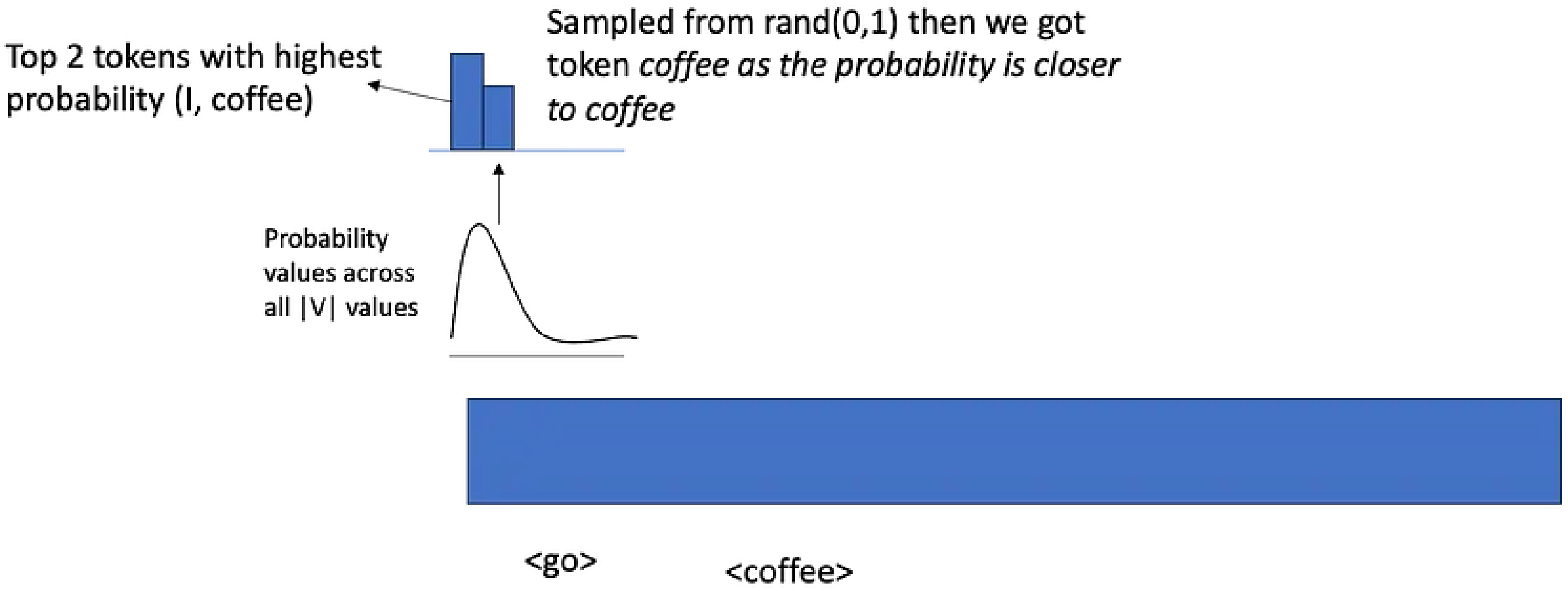
	time steps				
	1	2	3	4	5
cold	0.1	0.1	0.65	0.04	0.1
<stop>	0.15	0.15	0.05	0.01	0.5
coffee	0.25	0.05	0.05	0.1	0.2
I	0.4	0.05	0.05	0.03	0.1
like	0.05	0.55	0.09	0.02	0.05
water	0.05	0.1	0.11	0.8	0.05

The probability of top-k tokens will be normalized relatively as , P(I) = 0.61 ~ (0.25/ (0.25+0.4)), P(Coffee) = 0.39 ~ 0.4/(0.25+0.4) before sampling a token.

Here at every time step, consider top — k tokens from the probability distribution.

Let us assume and create a random number generator that predicts betwen 0 and 1 — rand(0,1). Suppose if the number obtained is ~0.7 then coffee will be the word or token that becomes as input at timestep 2, if again the random number generated is ~0.2 then at timestep 2 the word or token “I” will be the input.

Sampling Strategies — Top –K (Stochastic)



Sampling Strategies — Top –K (Stochastic)

	time steps				
	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	0.09	0.02	0.05
water	0.05	0.1	0.11	0.8	0.05

The generated sequence using top-K sampling for top 2 words are

like < stop>

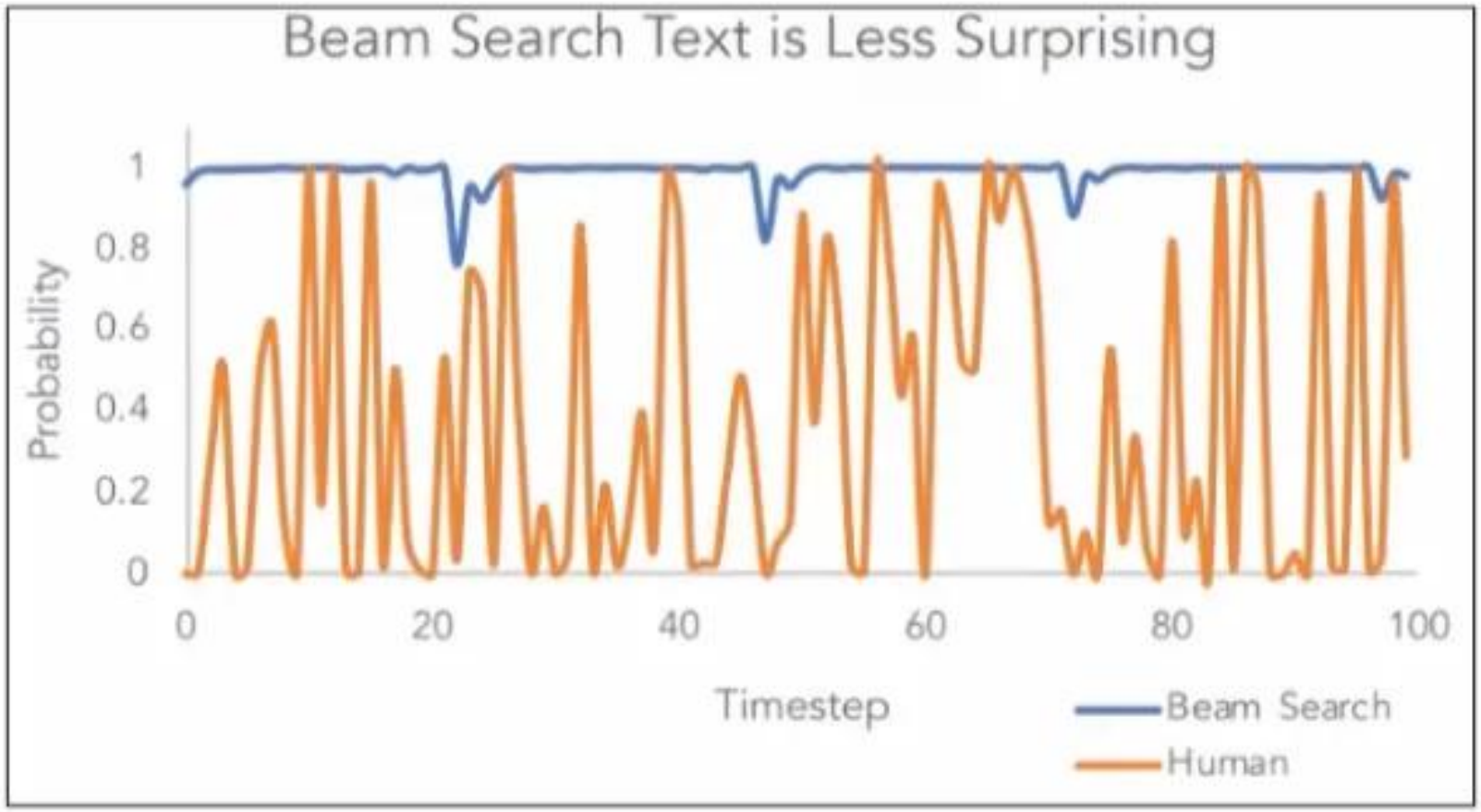
The normalized probabilities will be 0.15/(0.55+0.15)~0.23 and 0.55/(0.55+0.15) ~0.77 respectively for both the **like** and **<stop>**.

Sampling Strategies — Top –K (Stochastic)

	time steps				
	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	0.09	0.02	0.05
water	0.05	0.1	0.11	0.8	0.05
	coffee	like	cold	coffee	<stop>

now we run the rand function to generate the number from 0 to 1 — assuming if value is 0.9 the ***<stop>*** will be the output then the outcome process will stop there. Next time when the random generator output is 0.5 then we will have “like” as the outcome. So by doing this random generation we will have different outputs. May be for the first **“I” , “<stop>”** is generated — for all other scenarios the outcomes can vary as shown

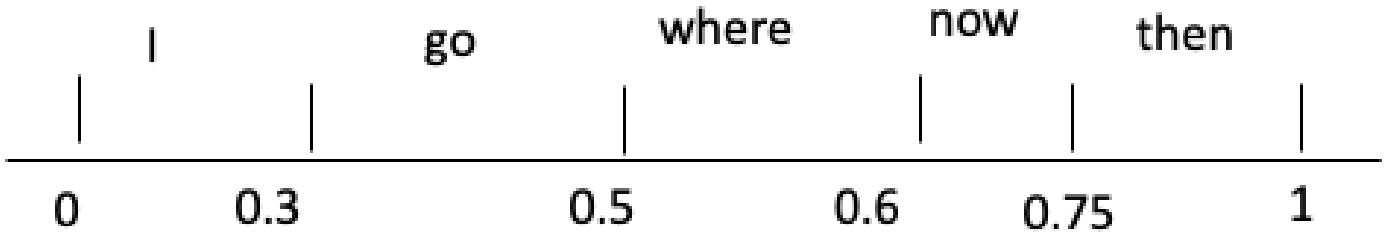
Sampling Strategies — Top –K (Stochastic)



If we look at beam search, it produces output with very high probability and hence we do not see any surprises — but If humans are asked to fill the sentences, we are going to have different and random outcomes with very less probability as 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.

Sampling Strategies — Top –K (Stochastic)

If we look at beam search, it produces output with very high probability and hence we do not see any surprises — but If humans are asked to fill the sentences, we are going to have different and random outcomes with very less probability as 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.
Suppose if we have top 5 words from the 40K vocabulary (I, go, where, now, then) and with probabilities as (0.3, 0.2, 0.1, 0.1,0.3) respectively.

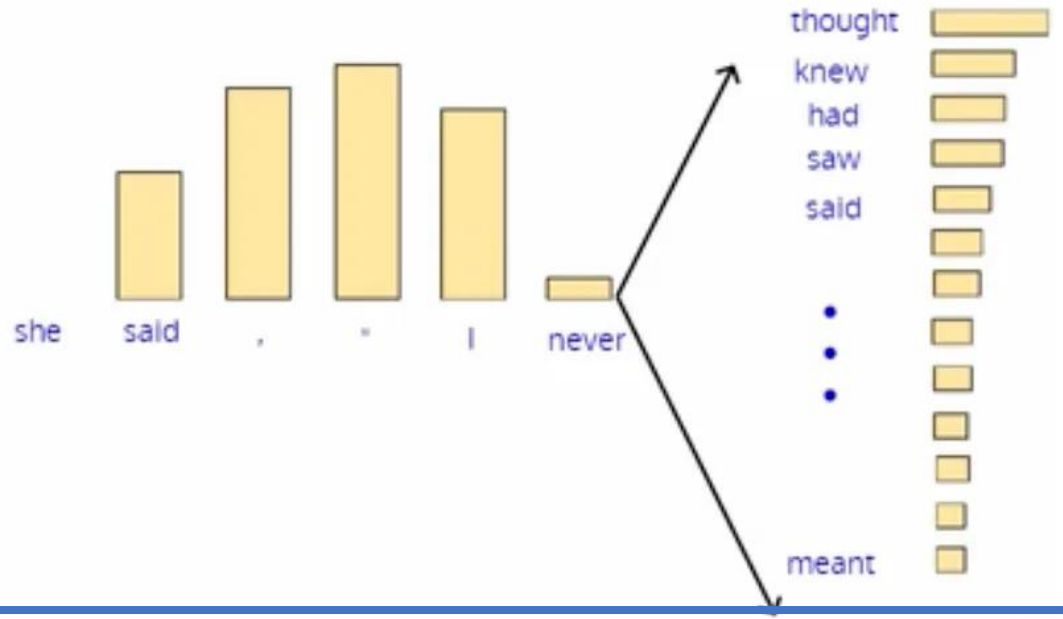


If the random generator generates any number b/w 0 and 1 and based on that value we will be selecting or sample the word or token to select the high probability values. We have to remember that here we are not randomly picking the sample from 40K vocabulary but what we are doing is already we have top 5 words from the 40K vocabulary and from this subset of top 5 words or samples we are creating the sequences **— here it is random but it is a controlled random selection of sequences.**

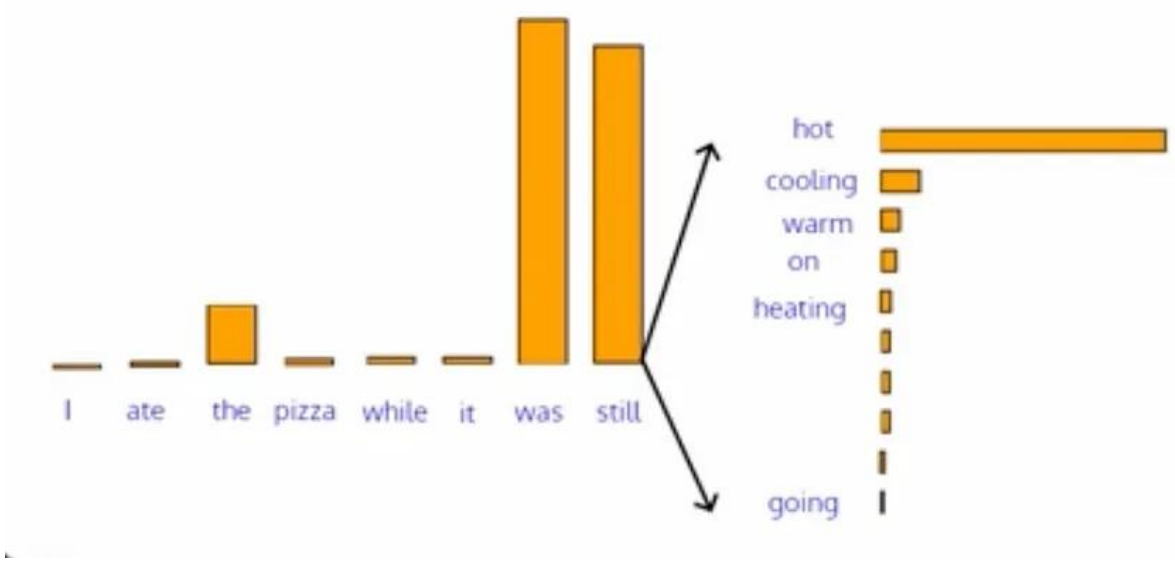
Sampling Strategies — Top -P

What should be the optimal value of k be?
let us take 2 examples which is flat and peaky distribution respectively.
Example-1: (flat distribution)

More Creative



Less Creative



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)
For a peaked distribution, using the same value of k = 5, we might end up creating some meaningless sentences.

Depending on the distribution type the value of k will vary — if we have a peaky distribution having a high value of k will not help as compared to a flat distribution where we can afford to have a little high value of k.

Solution 1: Vary Temperature

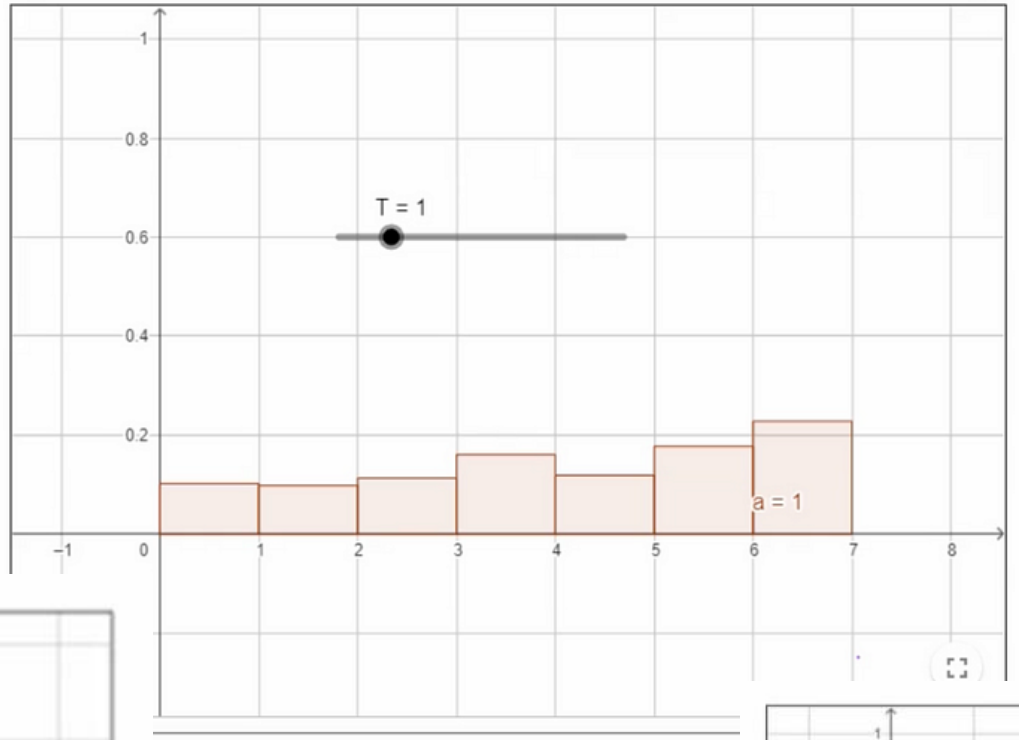
- **Temperature:** Controls the randomness of the output. Lower values make the output more deterministic and repetitive, while higher values increase creativity and variation.

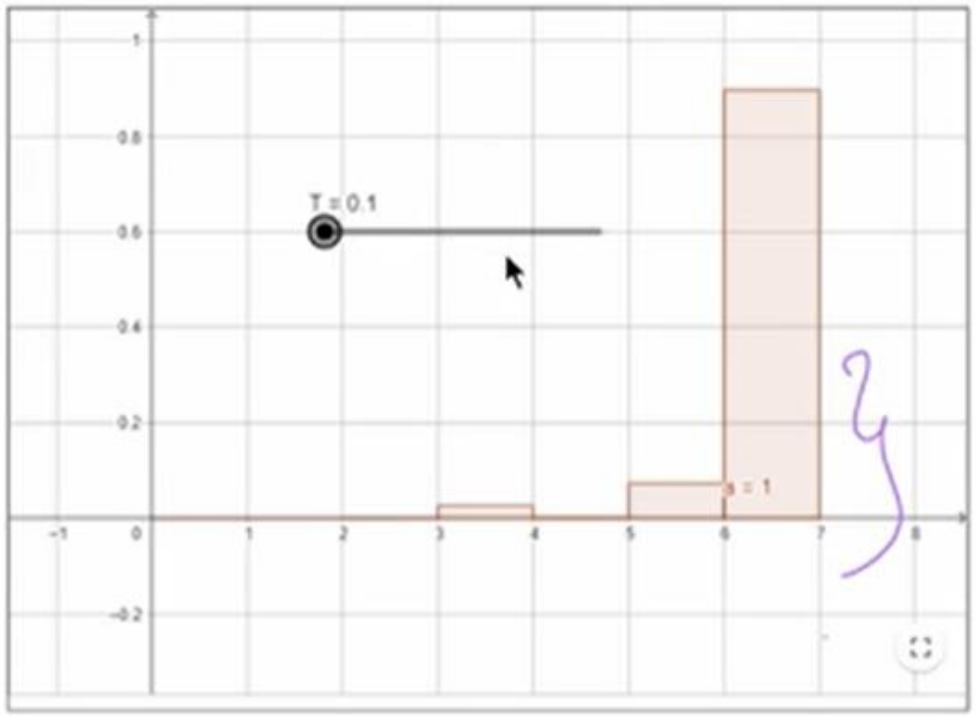
$$P(x = u_l | x_{1:i-1}) = \frac{exp(\frac{u_l}{T})}{\sum_{l'} exp(\frac{u_{l'}}{T})}$$

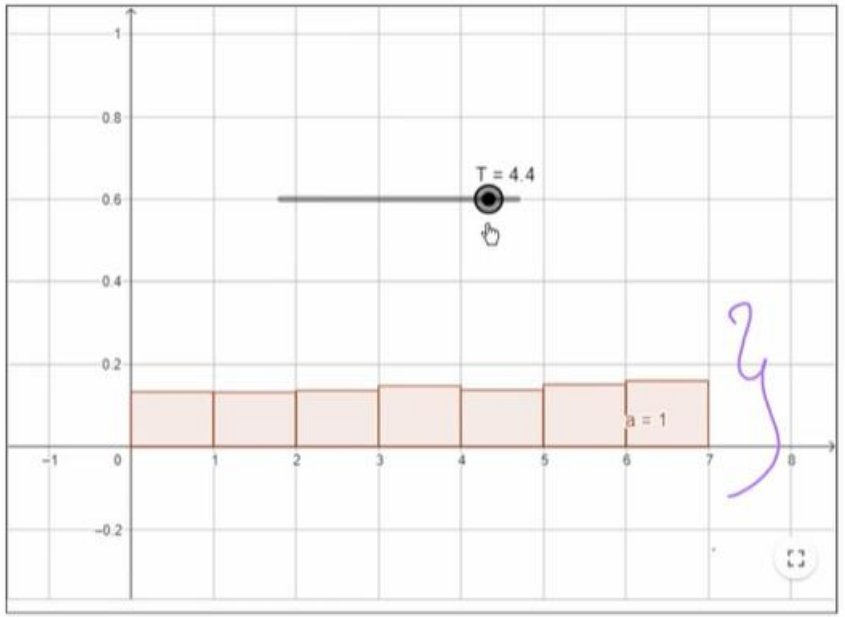
- Low temperature = skewed distribution = less creativity
- high temperature = flatter distribution = more creativity

When Temperature is lower, exp() term gets amplified and softmax becomes sharper and hence very peaky, confident distribution and vice versa

If we decrease the T value — we get the peaky distribution.





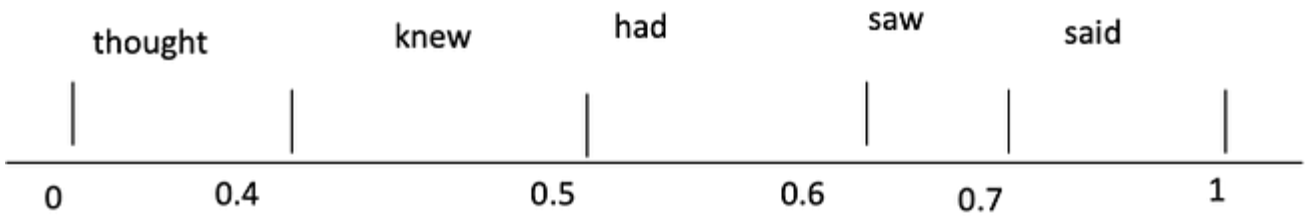


Solution — 2: Top — P (Nucleus) sampling

- Sort the probabilities in descending order
- Set a value for the parameters p, 0 < p < 1
- Sum the probabilities of tokens starting form 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.

Suppose we set p = 0.6 as threshold,

For example -1 distribution: The model will sample from the tokens (thought, knew, had, saw, said)



This is a wrap for all the decoding strategies for decoder only models i.e., GPT where we looked at deterministic and stochastic — this stochastic strategies are making sure that even though transformer has a deterministic computation output but at the end we are adding a sampling function which will make sure that we are sampling different tokens every time and hence generating different sequences.

Namah Shivaya