Introduction to Large Language Models (LLMs)

Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti

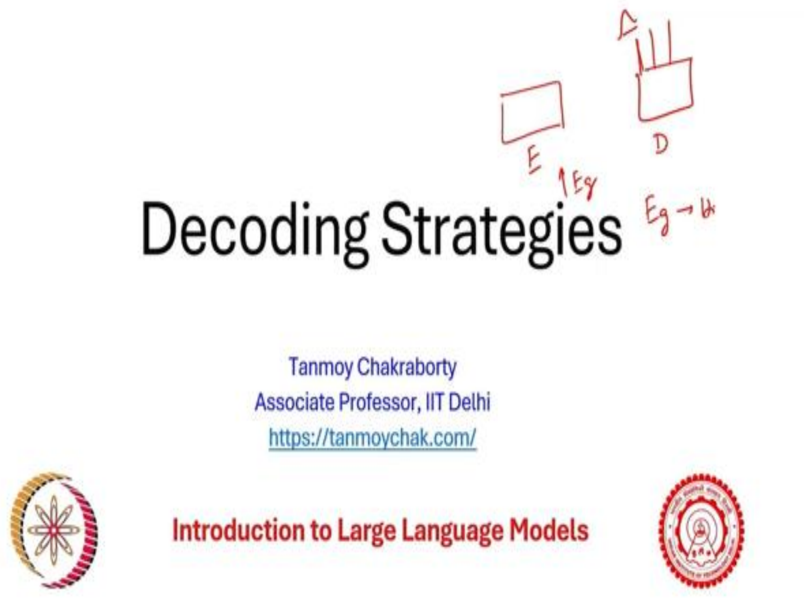
Department of Computer Science & Engineering

Indian Institute of Technology, Delhi

Lecture 13

Decoding Strategies

Hello everyone. So welcome back. So in today's lecture, we will discuss different types of decoding strategies. Okay.



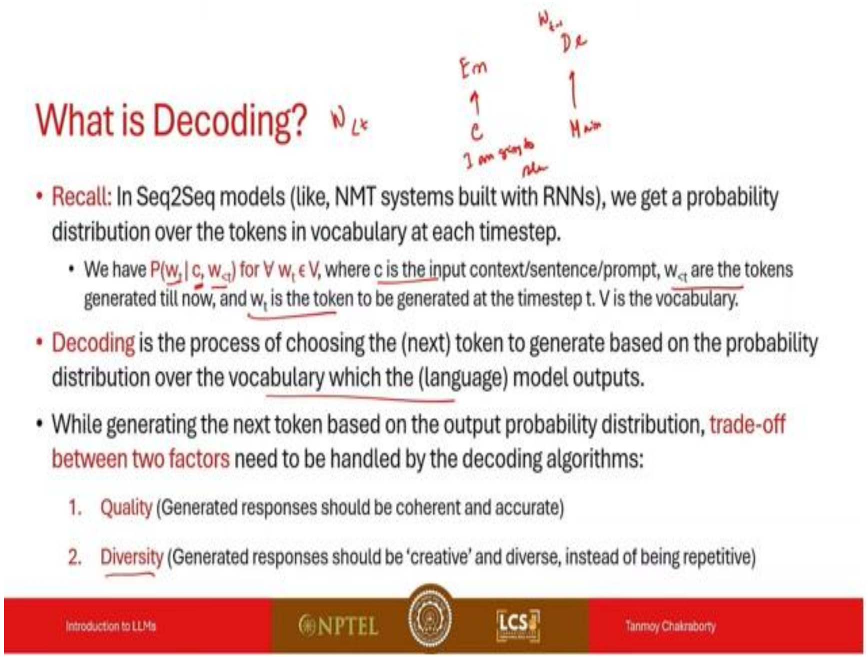
Decoding Strategies

Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

Introduction to Large Language Models

So if you remember, you know, we have this encoder decoder model, right? So encoder decoder model, let's say the application is machine translation, let's say English to Hindi. input is English and output is Hindi.

So at every state of the LSTM or let us say transformer, we will discuss later, we will produce a distribution and from the distribution the task is to choose the word which you want to essentially place at this position. There are different British strategies, different you know top case strategies that people came up with and today we will discuss some of the popular strategies.



What is Decoding?

- Recall: In Seq2Seq models (like, NMT systems built with RNNs), we get a probability distribution over the tokens in vocabulary at each timestep.
  - We have $P(w_t \mid c, w_{<t})$ for $\forall w_t \in V$, where c is the input context/sentence/prompt, $w_{<t}$ are the tokens generated till now, and w_t is the token to be generated at the timestep t. V is the vocabulary.
- Decoding is the process of choosing the (next) token to generate based on the probability distribution over the vocabulary which the (language) model outputs.
- While generating the next token based on the output probability distribution, trade-off between two factors need to be handled by the decoding algorithms:
  1. Quality (Generated responses should be coherent and accurate)
  2. Diversity (Generated responses should be 'creative' and diverse, instead of being repetitive)

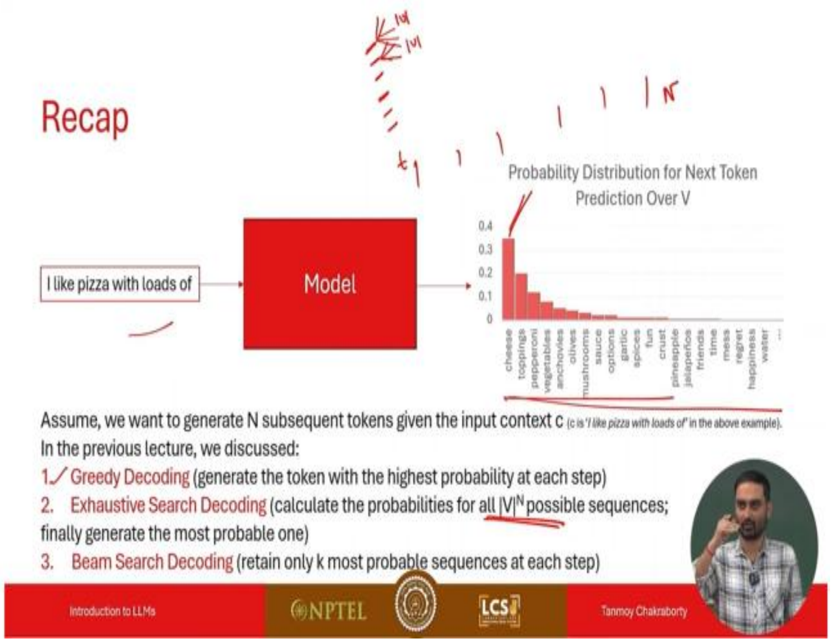
Let's look at the decoding strategy. So in a as I mentioned in a sequence to sequence model where you know you are given a prompt or whatever an input right and the input is C okay and that's the input to your encoder and to your decoder.

you have already generated things till t-1, right. This input is I am going to school and the output you have already generated mah, right mah, okay. Then what's going to be my next word. So, given the input c and the tokens I have generated so far W sub less than t means whatever I have generated till t minus 1 okay. What is going to be my word at the tth time okay.

C is the input W sub less than t are the tokens generated till now and we are interested in the current token. So decoding is a process of choosing the next token to generate based on the probability distribution of what a vocabulary, right, of that language. So now, you know, if we purely do this based on greedy, let's say I choose the best probable word as my next token, that will have certain disadvantage. What would be the disadvantage? The disadvantage would be this would mostly be deterministic. and this would be not only deterministic but also you know sometimes the quality may also be good, because let's say the application is question answering.

So, for every question, you always want the same answer to be generated by the model the quality would be good, but the problem would be diversity right. You won't be able to produce words, which are creative, which are diverse, right because you will end up producing only single word every time. It will not produce anything, any new output tokens given a specific input right. Now, of course, for machine translation, diversity is not important, innovation is not important. but let's say your input is that you know write a poem about something right.

So you do not want the model to produce the same response every time okay. You want the model to produce diverse responses. So to make a balance between quality and diversity we need some strategy.



Recap

Assume, we want to generate N subsequent tokens given the input context c (c is 'I like pizza with loads of' in the above example).

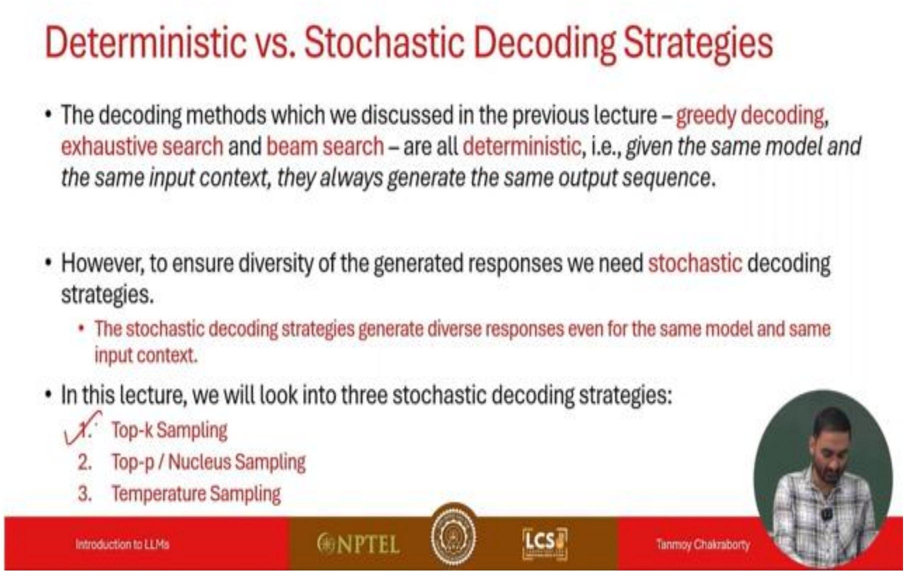
In the previous lecture, we discussed:

1. Greedy Decoding (generate the token with the highest probability at each step)
2. Exhaustive Search Decoding (calculate the probabilities for all $|V|^N$ possible sequences; finally generate the most probable one)
3. Beam Search Decoding (retain only k most probable sequences at each step)

So we have discussed greedy decoding, where as I mentioned, so given the input, when it generates the outputs, let's say the input is, I like pizza with lots of, and this is the probability distribution over tokens. And you see that the word cheese has the maximum probability.

So in greedy decoding, you essentially choose the word cheese, right? For exhaustive decoding, what you do? For exhaustive decoding, let's say at state time, you produce all possible words that can come here, meaning the vocabulary words. And for every vocabulary word, you see what would be the next vocabulary word. Again, a branch of size mod V. So at every position, you have mod V options. Right and how many words are there, how many words you want to output? you want to output n number of words, right.

So you have more V to the power n possible sequences which is not possible. Beam search we discussed, in beam search what we do? we choose you know a specific value of the beam size which is K for example and then at every time you look at K most probable sequence and essentially you keep on pruning those which don't come within this K probable sequence and you move on right. So greedy, Exhaustive, beam search, three of them are deterministic, right? There is no stochasticity, you know, associated with them because, you know, greedy, you choose the best one, which will always be the same. Exhaustive, you have all possible options. And beam search also, you look at K, K beams, right?

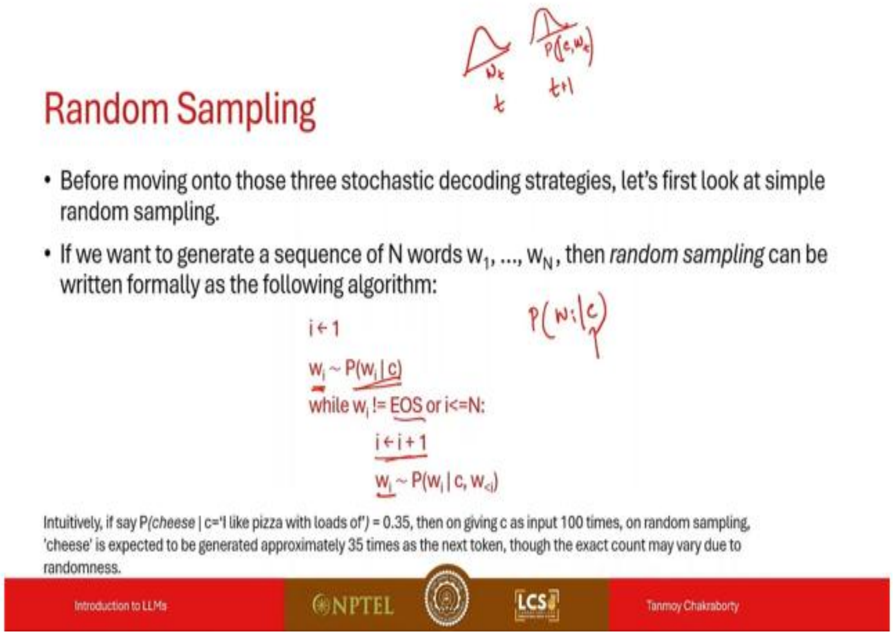


Deterministic vs. Stochastic Decoding Strategies

- The decoding methods which we discussed in the previous lecture – greedy decoding, exhaustive search and beam search – are all deterministic, i.e., *given the same model and the same input context, they always generate the same output sequence.*
- However, to ensure diversity of the generated responses we need stochastic decoding strategies.
  - The stochastic decoding strategies generate diverse responses even for the same model and same input context.
- In this lecture, we will look into three stochastic decoding strategies:
  1. Top-k Sampling
  2. Top-p / Nucleus Sampling
  3. Temperature Sampling

So deterministic versus stochastic strategies.

If you use greedy or exhaustive or beam search, you would end up producing accurate results, but the response will not be diverse. So we will discuss three strategies today, top K, top P and temperature sampling.



w_t t $P(c, w_t)$ $t+1$

Random Sampling

- Before moving onto those three stochastic decoding strategies, let's first look at simple random sampling.
- If we want to generate a sequence of N words $w_1, \dots, w_N$, then *random sampling* can be written formally as the following algorithm:

$P(w_i | c)$

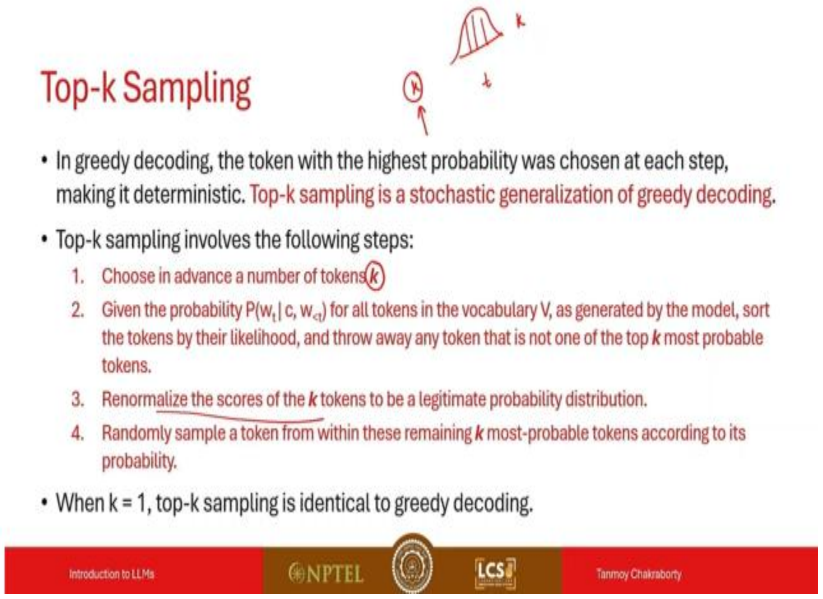
```
i ← 1
w_i ~ P(w_i | c)
while w_i != EOS or i<=N:
    i ← i + 1
    w_i ~ P(w_i | c, w_<i)
```

Intuitively, if say P(*cheese* | c='I like pizza with loads of') = 0.35, then on giving c as input 100 times, on random sampling, 'cheese' is expected to be generated approximately 35 times as the next token, though the exact count may vary due to randomness.

Introduction to LLMs NPTEL LCS Tanmoy Chakraborty

Okay, so in random sampling what we do, I mean the worst case, right. So let us say, so we have the distribution P of WI given C, C is the input. What we do? We randomly sample a word from the distribution okay and that's going to be my next word.

Again next so at t you sample from the distribution this would be wt at t plus 1 your input is c and till time t you have a distribution and you choose one randomly and you keep on going till you generate the end of sentence or you know the in number of tokens that you want to generate are all exhausted okay. It is purely random and this is not a good strategy at all.



Top-k Sampling

- In greedy decoding, the token with the highest probability was chosen at each step, making it deterministic. Top-k sampling is a stochastic generalization of greedy decoding.
- Top-k sampling involves the following steps:
  1. Choose in advance a number of tokens k
  2. Given the probability $P(w_t | c, w_{<t})$ for all tokens in the vocabulary V, as generated by the model, sort the tokens by their likelihood, and throw away any token that is not one of the top ***k*** most probable tokens.
  3. Renormalize the scores of the ***k*** tokens to be a legitimate probability distribution.
  4. Randomly sample a token from within these remaining ***k*** most-probable tokens according to its probability.
- When k = 1, top-k sampling is identical to greedy decoding.

Introduction to LLMs NPTEL LCS Tanmoy Chakraborty

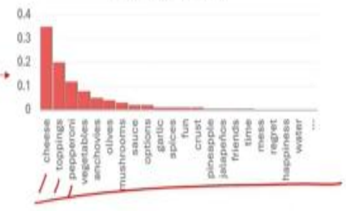
In case of top k sampling what you do? you first this is a variation of random sampling right but with a bit difference here and there. You choose the number of tokens value that you want to generate, right? And let's say the number is K. That means you will now play with this K tokens, right? What do you do? At tth time, right, you have the distribution.

You choose top K highly probable tokens, right? And then you renormalize the probabilities of this k tokens right and then now in the probability you have only k tokens right not mod V number of tokens and from there you choose things random right. So you are essentially doing random sampling within the top k tokens not mod V tokens okay.

Top-k Sampling: Working Example

I like pizza with loads of → Model →

Probability Distribution for Next Token Prediction Over V

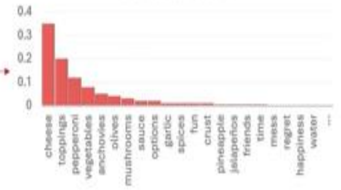


- Let's take **k=3.**
- Step-1: Sort the tokens by their likelihood, and throw away any token that is not one of the top ***k*** most probable tokens.
  - In our example, top-3 tokens with P(w|c) are: **cheese (0.35), toppings (0.20), pepperoni (0.12)**

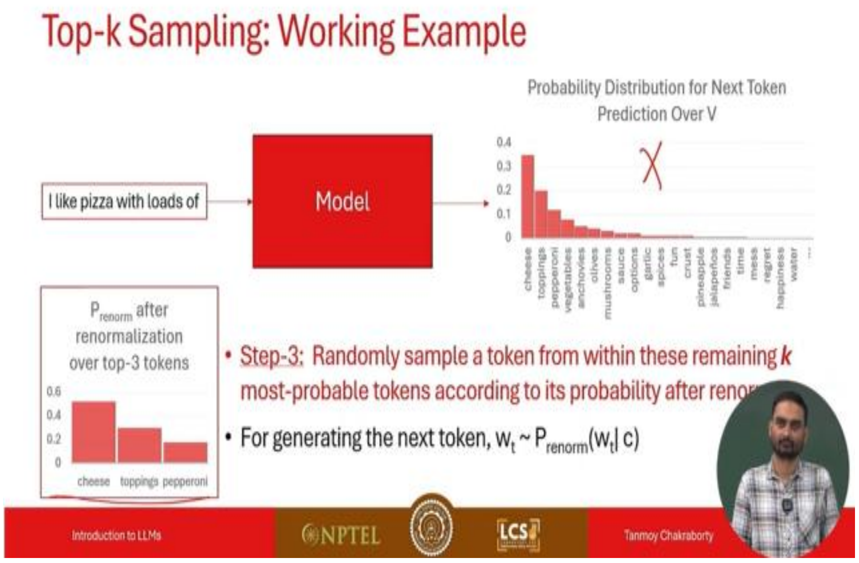
Top-k Sampling: Working Example

I like pizza with loads of → Model →

Probability Distribution for Next Token Prediction Over V



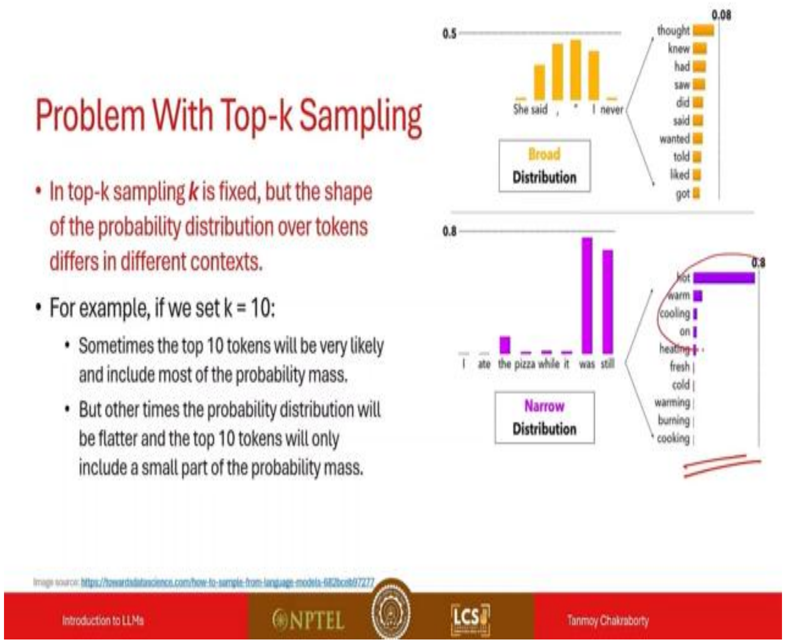
- Step-2: Renormalize the scores of the ***k*** tokens to be a legitimate probability distribution.
- After renormalization, $P_{renorm}(w|c) = \frac{P(w|c)}{\sum_{x \in top-k} P(x|c)}$.
  - Renormalized probabilities: **cheese (0.522), toppings (0.299), pepperoni (0.179)**



Top-k Sampling: Working Example

- Step-3: Randomly sample a token from within these remaining *k* most-probable tokens according to its probability after reno
- For generating the next token, $w_t \sim P_{renorm}(w_t | c)$

Let's look at example. Let's say K is 3, right? So this is my total, the entire probability distribution, right? You choose top 3, cheese, toppings and pepperoni, right? And this is their probabilities and now this is your candidate probability. These are your candidate probabilities now here so now you renormalize this right so if you renormalize then for cheese this is going to be 0.35 plus 0.35 plus 0.2 plus 1 2 right for toppings this would be I mean you can do in this way or you can do in a in a short max way right. which would be exponential of e to the power of this right. For topping this would be 0.2 by 0.35 plus 0.2 plus 0.12 and so on and so forth. So once you do this denormalization right, this is your resultant probability distribution okay. Now you choose things randomly from this distribution not from this, right. So you are restricted to k number of tokens now but you are doing random sampling. So this is top k.



Problem With Top-k Sampling

- In top-k sampling ***k*** is fixed, but the shape of the probability distribution over tokens differs in different contexts.
- For example, if we set k = 10:
  - Sometimes the top 10 tokens will be very likely and include most of the probability mass.
  - But other times the probability distribution will be flatter and the top 10 tokens will only include a small part of the probability mass.

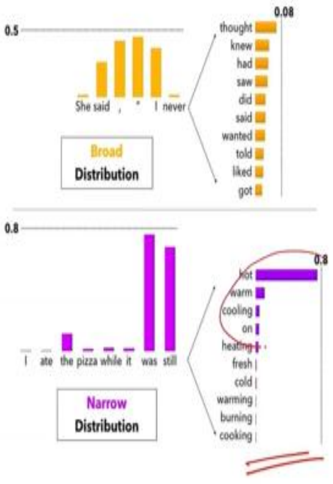
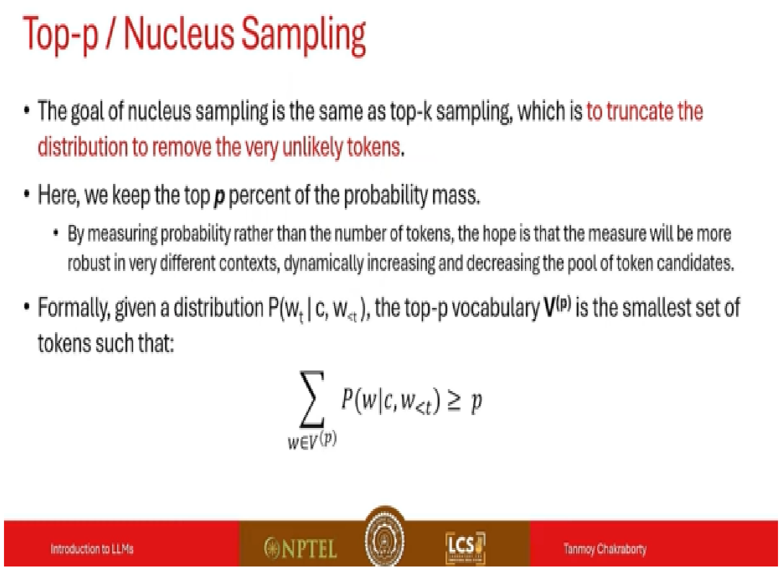


Image source: https://towardsdatascience.com/how-to-sample-from-language-models-682bceb97277



Top-p / Nucleus Sampling

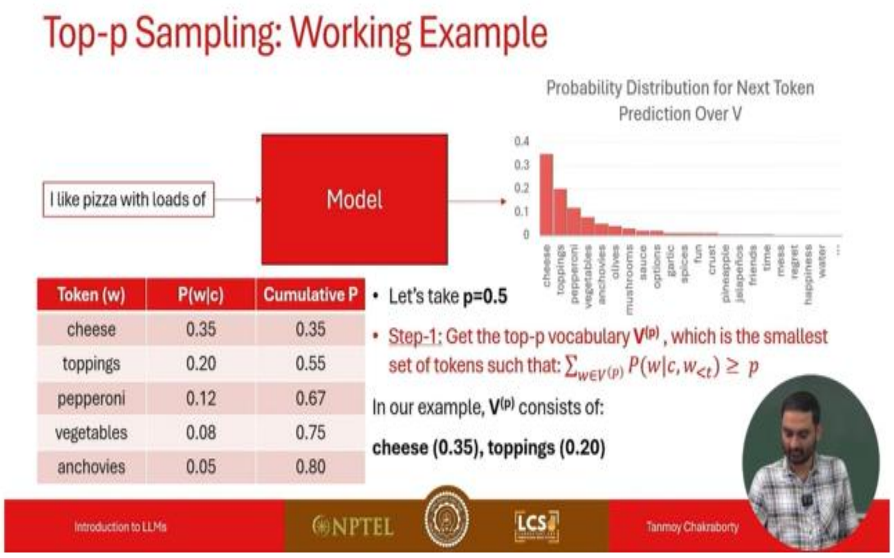
- The goal of nucleus sampling is the same as top-k sampling, which is to truncate the distribution to remove the very unlikely tokens.
- Here, we keep the top ***p*** percent of the probability mass.
  - By measuring probability rather than the number of tokens, the hope is that the measure will be more robust in very different contexts, dynamically increasing and decreasing the pool of token candidates.
- Formally, given a distribution $P(w_t \mid c, w_{<t})$, the top-p vocabulary $\mathbf{V}^{(p)}$ is the smallest set of tokens such that:

$$\sum_{w \in V^{(p)}} P(w \mid c, w_{<t}) \geq p$$

So the problem with top k sampling is that let's say you have a skewed distribution like this. Okay where top k words will encompass, the probabilities of the top k words will encompass the entire probability mass, right. You see in this case if you take this and this and this and this, this will be more or less the entire probability, right.

Whereas and if you have the distribution like this then top k makes sense. Whereas if you have a kind of uniform distribution like this, if you fix the value of k as 3 or 4, right, you will, let's say k is 3, so you would end up choosing only 3 words for your candidates, right, and the remaining words are also equally likely because they also have similar probabilities. So in this case, top k doesn't make any sense, okay. So instead of fixing K which is the number of tokens that you want to play with, if we can choose the certain probability and you say that look my cumulative probability threshold is 0.6 and I will start from highly probable word, then the next one, the next one and I will keep on adding you know the top probable top probabilities and keep on choosing those corresponding words until and unless the threshold is met okay.

So now we have a threshold on the probability not the number of tokens okay and this is exactly top k sampling or nuclear sampling okay. So you choose top k vocabulary right which is the smallest set of tokens such that this satisfies, smallest set this is important.



Top-p Sampling: Working Example

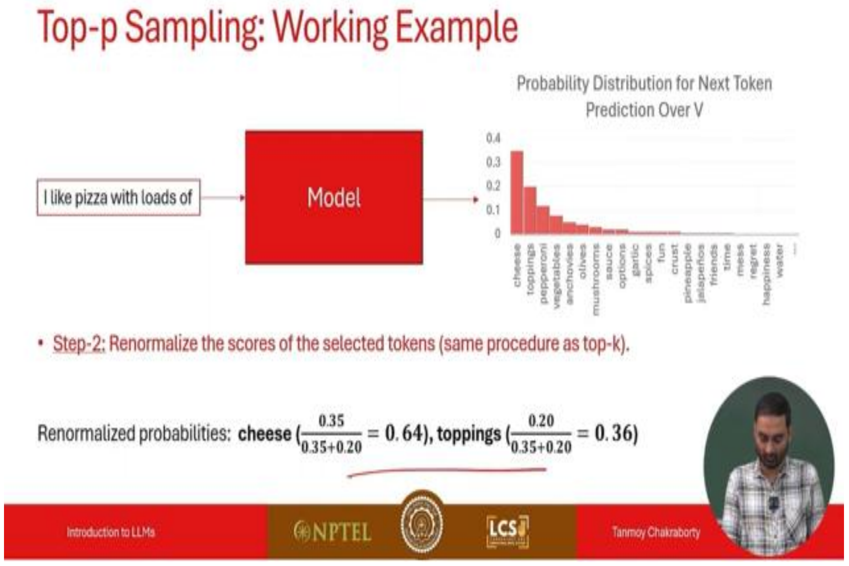
Token (w)	P(w|c)	Cumulative P
cheese	0.35	0.35
toppings	0.20	0.55
pepperoni	0.12	0.67
vegetables	0.08	0.75
anchovies	0.05	0.80

- Let's take **p=0.5**
- Step-1: Get the top-p vocabulary $\mathbf{V^{(p)}}$, which is the smallest set of tokens such that: $\sum_{w\in V^{(p)}} P(w|c, w_{<t}) \geq p$

In our example, $\mathbf{V^{(p)}}$ consists of:

cheese (0.35), toppings (0.20)

Introduction to LLMs NPTEL LCS Tanmoy Chakraborty



Top-p Sampling: Working Example

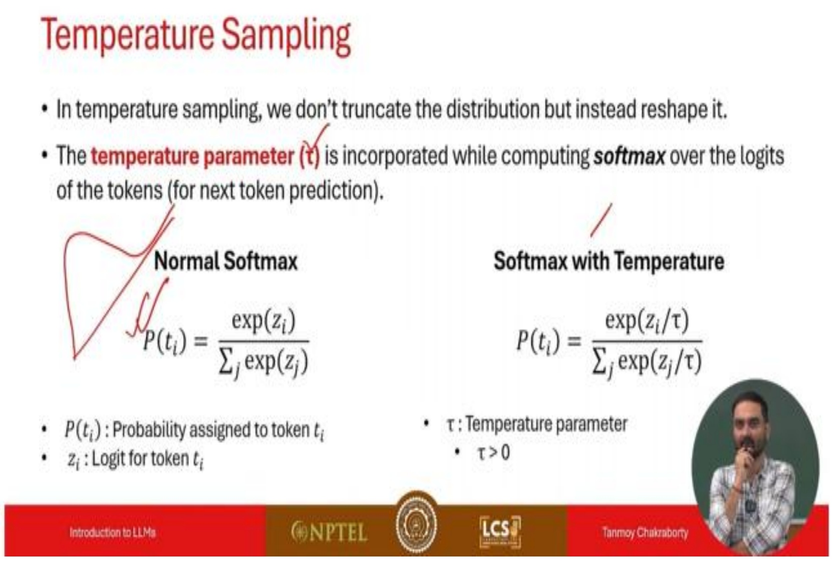
- Step-2: Renormalize the scores of the selected tokens (same procedure as top-k).

Renormalized probabilities: **cheese** ($\frac{0.35}{0.35+0.20} = 0.64$), **toppings** ($\frac{0.20}{0.35+0.20} = 0.36$)

Let's take an example, let's say this is my actual distribution right and let's say some of the, let's take some of the top probabilities right and these are the top probabilities. If we choose cheese the cumulative probability would be 0.35 if you choose toppings after that this would be 0.35 plus 0.2, 0.55 and so on and so forth right. So let's say the P is 0.5 okay.

It means the threshold is 0.5. It means that if I choose this and this, the sum of the probability would cross 0.5. I will essentially choose 0.35 and 0.2, right. That would basically exhaust my total probability budget and then I do the same. I will take these two words, probabilities, renormalize them and then do random sampling, right. So in this case cheese and toppings would be enough. If we do random sampling, if we do renormalization this is the renormalized probability and then we do random sampling okay. So the advantage here is that, let's say you have a uniform distribution, let's say the distribution, the probabilities look like this, 0.2, 0.2, 0.1, 0.1, right, 0.05, 0.05 and so on and so forth, right. So in that case, if the threshold is 0.5, you will end up choosing this, this, this, and maybe this because it is greater than 0.5, if it is greater than 0.5 then you choose this, this, this and this.

If it is skewed like this then you end up choosing only two, right. So this is the beauty of top case sampling.



Temperature Sampling

- In temperature sampling, we don't truncate the distribution but instead reshape it.
- The **temperature parameter (τ)** is incorporated while computing ***softmax*** over the logits of the tokens (for next token prediction).

Normal Softmax

$$P(t_i) = \frac{\exp(z_i)}{\sum_j \exp(z_j)}$$

- $P(t_i)$: Probability assigned to token t_i
- z_i : Logit for token t_i

Softmax with Temperature

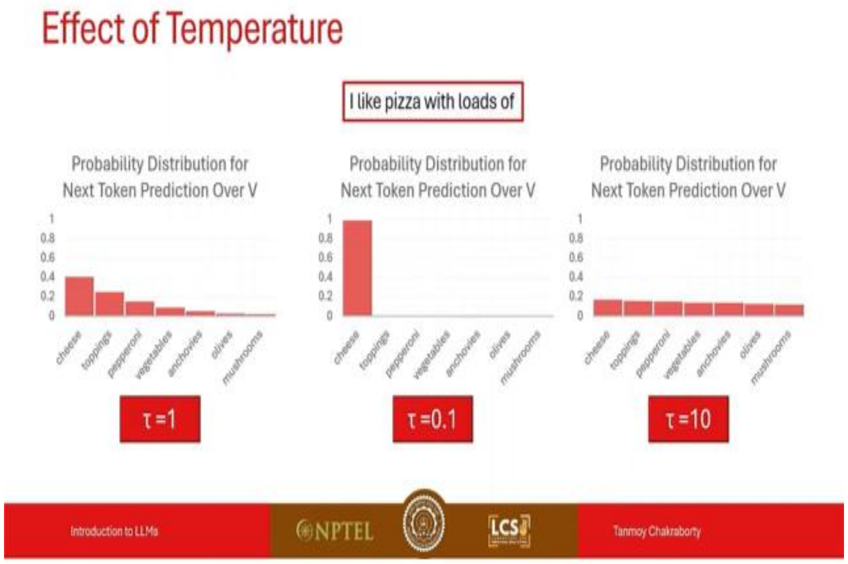
$$P(t_i) = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)}$$

- τ : Temperature parameter
  - $\tau > 0$

Introduction to LLMs NPTEL LCS Tanmoy Chakraborty

The remaining one is temperature sampling and this is very simple. Temperature sampling, we do not truncate the entire distribution. It's not like from the distribution I choose top 3 or top 4, whatever, top 3.

Here I use a new hyperparameter which is the temperature parameter called tau, right? And the hyperparameter, so this is my short max probability based on which I create the distribution and this is my short max in temperature. Right you see the temperature here the actual logit by the tau parameter okay.



Effect of Temperature

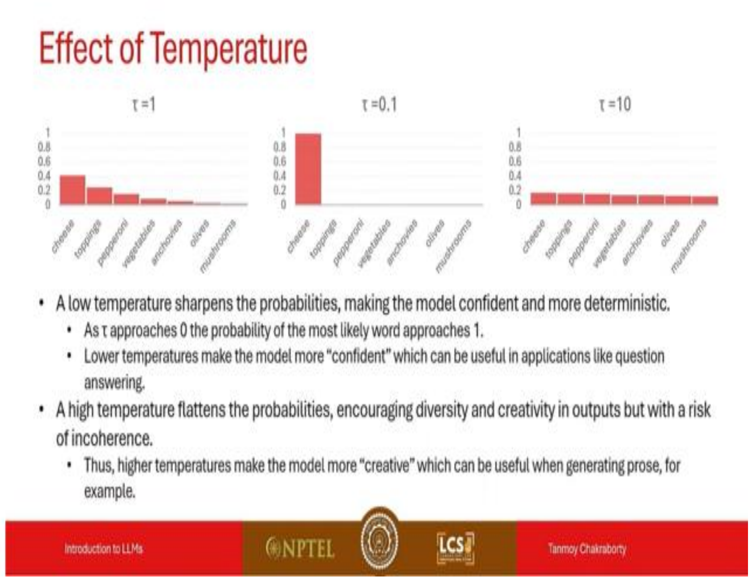
I like pizza with loads of

Introduction to LLMs NPTEL LCS Tanmoy Chakraborty

Now if tau is 1 then this is same as you know the random short max probability if tau is low 0.1 for example you would end up think about it tau is low that means this would be high right versus if tau is high then this would be low, the exponent would be low. So if tau is low you are kind of promoting more skewedness, right.

So if tau is low you would see skewed distribution and then you can choose the top one or top two whatever. If tau is high it will be more of an uniform distribution. So you can play with this temperature parameter tau and if you want to be deterministic then you always choose the lower value of tau. If you want to be more stochastic, probabilistic, right, more variation, more diversity, right, you should choose high value of tau, right, depending upon your application.

This is a very important hyperparameter.



Effect of Temperature

- A low temperature sharpens the probabilities, making the model confident and more deterministic.
  - As τ approaches 0 the probability of the most likely word approaches 1.
  - Lower temperatures make the model more "confident" which can be useful in applications like question answering.
- A high temperature flattens the probabilities, encouraging diversity and creativity in outputs but with a risk of incoherence.
  - Thus, higher temperatures make the model more "creative" which can be useful when generating prose, for example.

Introduction to LLMs NPTEL LCS Tanmoy Chakraborty

Okay so this is written here. If you want to be creative your tau should be higher. If you want to be deterministic tau should be lower and this is tau equals to 1 is the regular shortness probability.

That's it. So this is about decoding. Now this part is complete. In the next lecture we will move to the next part. Thank you.