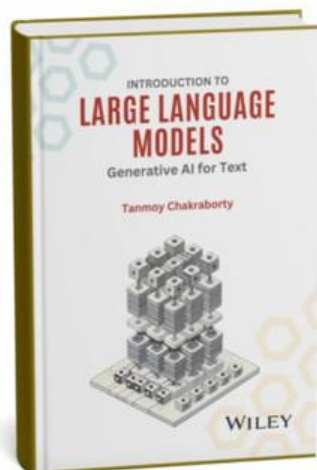


Introduction to Large Language Models (LLMs)
Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi
Lecture 25

Alignment of Language Models-II

Hello everyone, welcome back to the course on LLMs. So we were discussing alignments of large language models. And in the last class, we discussed how, you know, human generated feedback can be injected to this model for further refining things. So, specifically we discussed that we have a policy model. The policy model is the LLM that you want to refine, that you want to fine tune.



Reference Book
INTRODUCTION TO
LARGE LANGUAGE
MODELS

Book Chapter #08
Fine-Tuning & Alignment of LLMs
Section 8.4 (Alignment Methods)

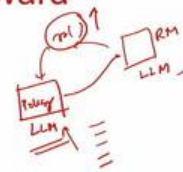
And then this policy model will generate certain output. This output will be scored by a reward model. This reward model itself is another LLM. So, it will produce a reward and based on this reward, we will further refine this policy model. We also discussed that only reward maximization is not enough because what happens if we only maximize the reward.

What would end up happening is that the policy model will start hacking the reward model, meaning that it will start producing such responses for which the reward model will

produce higher reward values. But the responses may not be realistic. For example, the policy model will start producing lot of emoticons, it will start producing sentences which are verbose, lengthy and so on and so forth, not to the point and so on and so forth, which you do not want. Therefore, to address this reward hacking loophole, what you do? you have another component in the objective function. which you want to minimize and what is this component? This component is basically the divergence scale divergence between the old policy and the updated policy, the old LLM and the updated LLM.

Combining the Objective: Regularized Reward Maximization

- Maximize the reward
- Minimize the KL divergence
- Add a scaling factor β & combine



So, you do not want the updated LLM to be very far from the starting LLM. So, we discussed maximizing the reward which is the expected reward that you obtain given a policy. So, the policy model is π_θ . So, this is parameterized by θ . You essentially sample, given a prompt x , this policy model will generate a y and for this y , you have this reward.

So, for example, x is the prompt that the prompt is where is Taj Mahal located and let us say the y is the response. Let us say the response is Taj Mahal is located in Uttar Pradesh or Taj Mahal is located in France and so on and so forth and based on that you give high reward or low reward. And this second term, this term is essentially the KL divergence between the updated policy model and the reference policy model. So, π_θ and π_{ref} .

So, reference LLM reference policy model is the one from which you started your reward maximization process.

These two components will be combined. So, you want to maximize reward and minimize the scale divergence and there is this beta scaling factor. Now, this lambda essentially is responsible for scaling these two components. If you want to give more weightage to scale divergence, you increase the value of lambda and so on and so forth. So this we discussed.

So now the question is or given this regularized reward maximization why this is regularized reward because this component you can think of this as a regularizer. This is your objective and this is your regularizer. So, regularizer essentially tries to balance both these things. So, this regularized reward maximization will be used to further refine the policy model. Policy model is the LLM again.

The policy model is the LLM that you want to rectify, you want to refine. So, how do you do in practice? So, given a prompt, let us say the prompt is write a poem. You have this policy model π_θ . The policy model will generate n number of poems. For each poem, you calculate the reward using reward model and the KL divergence right. So for every response, you have the reward and the KL divergence, reward and the KL divergence, right. And since this is expected, right, what you do, I mean, it's not possible to calculate expectations. So what you do, you essentially sample, you sample several sub responses from the policy and you make an average of this, right? And this is going to be your total reward, regularized reward function, which you want to maximize, right? Now, in order to maximize this, you know, so through this maximization process, you will essentially learn this θ . which is our main objective.

Combining the Objective: Regularized Reward Maximization

- ✓ Maximize the reward

$$\mathbb{E}_{\pi_{\theta}(y|x)} [r(x,y)] \uparrow$$
- ✓ Minimize the KL divergence

$$\mathbb{E}_{\pi_{\theta}(y|x)} \left[\log \frac{\pi_{\theta}(y|x)}{\pi_{\text{ref}}(y|x)} \right] \times \lambda \downarrow$$
- Add a scaling factor β & combine

$$\mathbb{E}_{\pi_{\theta}(y|x)} \left[r(x,y) - \beta \log \frac{\pi_{\theta}(y|x)}{\pi_{\text{ref}}(y|x)} \right]$$

Introduction to LLMs
NPTEL
LCS
Tanmay Chakraborty

Your objective is to basically define the parameters in the policy model. So when you have this kind of picture diagram, What is the way to essentially do this kind of maximization? Let's say this is some average, this is some loss or sorry, in this case, this is essentially the reward maximization objective, right? You calculate the partial derivative of this, you do a normal gradient descent, right? With respect to all the parameters present in the system, right? By taking the derivative, partial derivative of the objective function with respect to parameters. The problem here is you cannot use the traditional, the vanilla version of gradient descent here. Why? Think about it. The reason is this part.

Okay. This is highly non-differentiable because here what you are doing, you are generating multiple responses. You are basically sampling multiple responses. You are generating multiple responses and this is non-differentiable. highly non-differentiable.

Okay. This is not one sample. This is multiple samples, right? So standard gradient descent algorithm won't work here. So we need some other ways to essentially address these issues, right? So let us see. So this is something that we'll discuss today.

Regularized Reward

$$E_{\pi_{\theta}(y|x)} \left[r(x, y) - \beta \log \frac{\pi_{\theta}(y|x)}{\pi_{ref}(y|x)} \right] \equiv E_{\pi_{\theta}(y|x)} r_s(x, y)$$

$$\text{where } r_s(x, y) = r(x, y) - \beta \log \frac{\pi_{\theta}(y|x)}{\pi_{ref}(y|x)}$$

- $r_s(x, y)$ is the regularized reward
- Maximizing the regularized reward ensures
 - High reward outputs as decided by the reward model
 - Outputs that have reasonable probability under the reference model



So our regularized reward looks like this. So this is our reward. This is the KL divergence, and this is the expectation. right. Let us denote this term by R_{sxy} right.

So essentially you are maximizing this R_{sxy} . Again maximizing the regularized reward ensures that high reward outputs as decided by the reward model the output will have high reward because of this term and outputs that have reasonable probability under the reference model okay that will take that will be taken care of by this factor. So under the reference model it should not be very different okay. So now let's look at how to maximize this. So we'll use some algorithm called, and this is a very straightforward algorithm in reinforcement learning.

How to maximize – The REINFORCE algorithm?

- Compute the gradient of the objective.
- Train using Adam/Adagrad optimization algorithms

$$\begin{aligned}\nabla_{\theta} E_{\pi_{\theta}(y|x)} r_s(x, y) &= \nabla_{\theta} \sum_{y \in \mathcal{Y}} \pi_{\theta}(y|x) \underbrace{r_s(x, y)}_{\text{fixed}} \\ &= \sum_{y \in \mathcal{Y}} \nabla_{\theta} \pi_{\theta}(y|x) r_s(x, y)\end{aligned}$$



Introduction to LLMs

NPTEL



LCS

Tanmoy Chakraborty

If you know RL, you may have heard of this term, this algorithm called reinforced algorithm, right? So let's again re-look at the objective function. So the objective function that I mentioned in this slide is this one expectation of this one okay. Now so this one I need to maximize. so I need to take the gradient of this respect to theta so gradient of this. And this is expectation.

So essentially it means, what does it mean? It means that you generate a sample output, right? From this policy model, which belongs to all possible sample outputs that the model can generate. Once a sample output is Y and the probability of this sample output is this one, right? And this is the report. Okay so if we do not want to use this expectation you can unfold it in this way. So given sufficient number of samples which are part of the output set you take one of each of the samples one by one and this is my probability and this is my reward. Now think about it this reward is fixed with respect to theta.

Theta is the parameter of your policy and this is my reward model and this reward model generates R of S right and so when I take the gradient of this with respect to Theta this is constant right. when I move this gradient when I move the derivative inside this will be the part okay and this will be separated this is not a part of this gradient okay. So now think about it again so this is exactly the one that we just derived okay $\nabla_{\theta} \pi_{\theta}(y|x)$ given X reward X, Y okay. So now exact computation of this gradient is intractable because you do not know the output space, you do not know this output space Y because the LLM can

generate infinite number of responses right. So can we approximate it? In order to approximate this I need to rewrite this expression in terms of some sort of expectation okay.

How to maximize – The REINFORCE algorithm?

- Compute the gradient of the objective.
- Train using Adam/Adagrad optimization algorithms

$$\begin{aligned} \nabla_{\theta} E_{\pi_{\theta}(y|x)} r_s(x, y) &= \nabla_{\theta} \sum_{y \in Y} \pi_{\theta}(y|x) \underbrace{r_s(x, y)}_{\text{fixed}} \\ &= \sum_{y \in Y} \left(\nabla_{\theta} \pi_{\theta}(y|x) \right) r_s(x, y) \end{aligned}$$

Handwritten notes: $\pi_{\theta}(y|x)$ is boxed and labeled "Policy", $r_s(x, y)$ is boxed and labeled "RM".

So I need to rewrite this in terms of expectation which would look like this. So what would be the exact expectation formulation that we will discuss okay. So this is the desired form that we wanted to, uh, that the, that you want to want to produce from, uh, from this equation. Right. Okay.

Don't worry about all these equations. Okay. This is very straightforward. I will explain these equations one by one. So, so what I will do here, I will do some sort of log derivative tricks and there's a very old trick.

Okay. Nothing fancy. What does it say? It says that you take the derivative of this. What is this term? Why this term from where this is coming? I will explain later. Let us assume that there is a term this log of pi theta y given x and you are taking the derivative of this.

So this will be 1 by this one times derivative of derivative of this one log of z right 1 by z and then dz dx okay. So now you move this here okay so if you move this here then you will have this term which is this and this times this this times this here okay. Now why this is important because Why I derived this? Now look at the expression that I had earlier. Okay. This is this one.

Okay. Y given capital Y , right? Φ theta, right? And then we also had... derivative with a derivative of this π theta y given x and r is xy right.

So this is the expression okay derivative of right. So what I will do? and then this r term okay I will replace this by this okay you replace this here okay. So now if you look at this very very carefully this is actually an expectation. This is probability this is sample right expectation. So what is this expectation? This is this expectation.

So if you move this part separately if you keep this part separate if you only focus on this this is nothing but expectation and you can write in this way right. so expectation of... So essentially you sample a Y from the policy model, then the reward of that Y and then the derivative of the log of the reward.

This is the expression. So I wanted to get an expression like this from here that I got. Okay. So now we have this expectation, right? So let's see. So again let's look at the same figure prompt policy model all these responses are generated right.

You have reward model reward model gives a reward with these are the rewards right and then but then we have, so we need essentially two terms, the reward and the derivative of the policy, right? Derivative of the policy model, you know, given this prompt and this Y . So this is this part, right? given x , x is the prompt, 0.2 given x and so on and so forth, right. Then what you do, you take the average, right. And why this is called Monte Carlo, this is again a fancy term.

Whenever we use sampling as a replacement of expectation, we use Monte Carlo term, okay. Then we average and then we, you know, back propagate. So but, you know, the game is not over yet, you know, a long way to go by the way. So this is just started. If you think of this response right This response poem one for example this is y okay and this response constitutes multiple words okay $a_1 a_2 \dots a_t$ multiple tokens right.

So this is the response and i am calculating the reward of the entire response, right? The entire response. This is very important to remember. I'm not calculating the reward for every token in the response. This is the entire response. So it means that I need to wait till the end of this response generation, right? And then I'll be able to calculate this reward,

right? So once the response generation is complete, I give it to the reward model and the reward model will generate the response value, right? So nevertheless so if you look at this term again right $\pi_\theta(y|x) \log \pi_\theta(y|x)$ is essentially a $1 \times 2 \times \dots \times a_t$.

Expanding the Gradient

- Let $y = (a_1, \dots, a_t)$ be the tokens of y .
- $$r_s(x, y) \nabla_\theta \log \pi_\theta(y|x) = r_s(x, y) \nabla_\theta \sum_{t=1}^T \log \pi_\theta(a_t | s_t) \quad s_t = (x, a_0, \dots, a_{t-1})$$

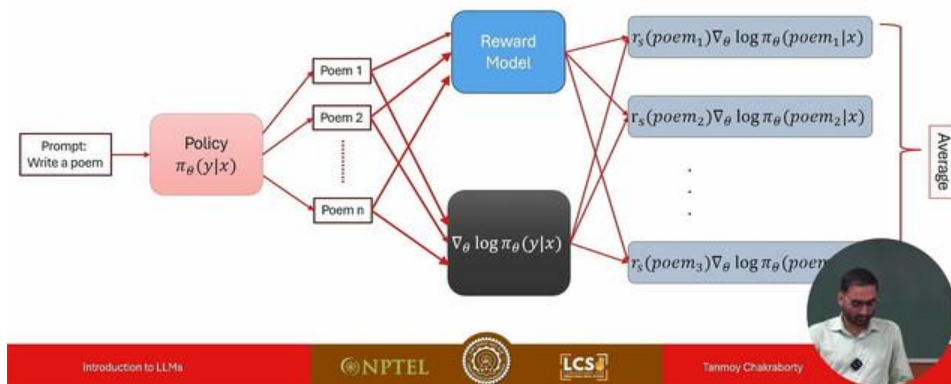
$$= r_s(x, y) \sum_{t=1}^T \nabla_\theta \log \pi_\theta(a_t | s_t)$$

$$= \sum_{t=1}^T r_s(x, y) \nabla_\theta \log \pi_\theta(a_t | s_t)$$

So what you can do, you can right. So you can actually think of it as a cumulative, you know, cumulative gradient, right, for every token. So this is essentially log of, let's say, Taj Mahal, is in UP. So log of the policy Taj given the state and what is the state? The state is essentially the prompt and the response generated so far. So this is this plus log of π_θ Mahal, given so this is s_2 right this is s_1 this is s_2 .

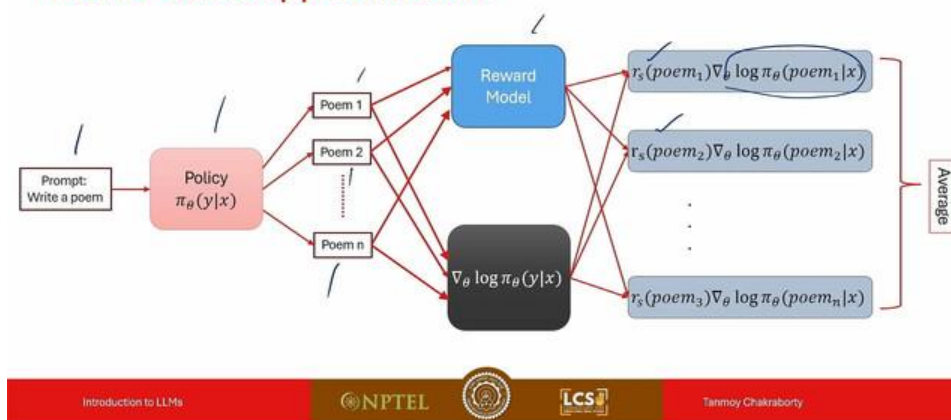
What is s_2 by the way, s_2 is the prompt where is Taj Mahal located prompt and Taj right together and plus log of right is s_3 , what is s_3 ? Prompt Taj Mahal right and so on and so forth. So this is this right and we move this gradient inside right, move this gradient inside, you can also move this reward inside, right? So this is my compact version. Again, if you look at the reward part carefully, there is no T involved meaning that this is independent of every token in Y . This is only that this reward is calculated for the entire response. Okay.

Monte Carlo Approximation



This is written here for every token. You use the same reward that is calculated for the entire sequence. And now this is the implementation of this reinforced algorithm. This is the prompt, where is Taj Mahal, assistant colon, Taj Mahal is in Agra, right. So the policy model, essentially the policy model generates this, right? And you get reward, this reward value R, right? For every token.

Monte Carlo Approximation



Now for every token, now you calculate this log probability, right? Every token you can calculate the log probability, states are different by the way, okay? And then you take the sum of all these rewards, right? And then you back forward it, okay? But here is the problem. The problem is in order to find the contribution of the word Taj here, right? I'll

have to wait till this number one. Number two is, let's say the model produces that the Taj Mahal is in France, right? Which is wrong. Therefore a negative reward will be generated. So that negative reward will also impact the word Taj because the same reward will be used for all the tokens separately.

Right? But this should not have happened. Why? Because the word Taj is correct. Right. So we should not use the same reward for all the tokens present in this response. Okay.

So how to address this? Now what I mentioned is basically written here. The reward at token Taj depends on the tokens generated in the future. If the model had generated Taj Mahal is in Paris, the reward would be negative. The probability of generating Taj would be decreased, right? If the model had generated Taj Mahal is in Agra, reward would be positive and the probability of generating Taj would be increased, right? This variance in the reward leads to unstable training. Obviously, right? So to address this, what we'll do, we'll take the average reward.

Problems with REINFORCE

- The reward at token "Taj" depends on the tokens generated in the future
- If the model had generated "Taj Mahal is in Paris"
 - The reward would be negative
 - The probability of generating Taj would be decreased
- If the model had generated "Taj Mahal is in Agra"
 - The reward would be positive
 - The probability of generating Taj would be increased
- This variance in the reward leads to unstable training.
- To reduce variance – take the average reward over all likely sequences (under the policy) that generate "Taj" for the first token.
- This is called the Q – function

This is very important. The average reward over all likely sequences that generate Taj for the first token. What does it mean? It means that you look at all the responses generated by the policy model which starts with the word Taj. Taj Mahal is in Paris, Taj Mahal is in UP, Taj Mahal is in, you know, Lucknow, Taj Mahal is in West Bengal, whatever responses that the model generates given the prompt, of course. Right? And then you calculate some sort of reward, right, some sort of cumulative reward for the word touch.

And this is called Q function. Okay. You know, some of you may know reinforcement learning. This is essentially a very popular method in reinforcement learning. So as you see here, I use this Q function okay.

Now here instead of using R I use Q function which is essentially a function of the state and the current action right, what is the action? The action is a token that you generate. So here you use this Q function. Now in this case, the state is S1 and the token that you generated Mahal right here, the state is S2 and the token you generate is E and so on and so forth. Okay. Now, what is this Q function? So the Q function intuitively, this is average reward of all likely, you know, the all likely sequences that start with Taj.

Right? So you look at all sequences, again, hypothetically, you look at all sequences, given the prompt, you look at all sequences, starting with touch, you calculate the reward of all the sequences one by one, right? And then you take the average of this. And this will give you some ideas about the, you know, the likelihood of the of generating the token touch. Okay so let us again discuss Q function you know mathematically. So Q function I wrote this thing in the slide so that you read and you remember. The Q function for a state action pair is the average discounted cumulative reward received at the state after taking is repeated after taking the specific action okay I will explain.

Q-function & Value function

- The Q-function for a state-action pair is the average discounted cumulative reward received at the state after taking the specified action. $s_{k+1} = (s_k, a_k)$

$$Q(s_k, a_k) = \mathbb{E}_{\pi_\theta(a_{k+1}, a_{k+2}, \dots, a_{k+T} | s_k)} \left[r(s_k, a_k) + \underbrace{\gamma r(s_{k+1}, a_{k+1}) + \gamma^2 \dots}_{\text{discount factor}} \right]$$

- The discount factor γ ensures that immediate rewards get higher weight.
- The Value function of a state is the average discounted cumulative reward received after reaching the state.

$$V(s_k) = \mathbb{E}_{\pi_\theta(a_k, a_{k+1}, \dots, a_{k+T} | s_k)} \left[r(s_k, a_k) + \gamma r(s_{k+1}, a_{k+1}) + \gamma^2 \dots \right]$$

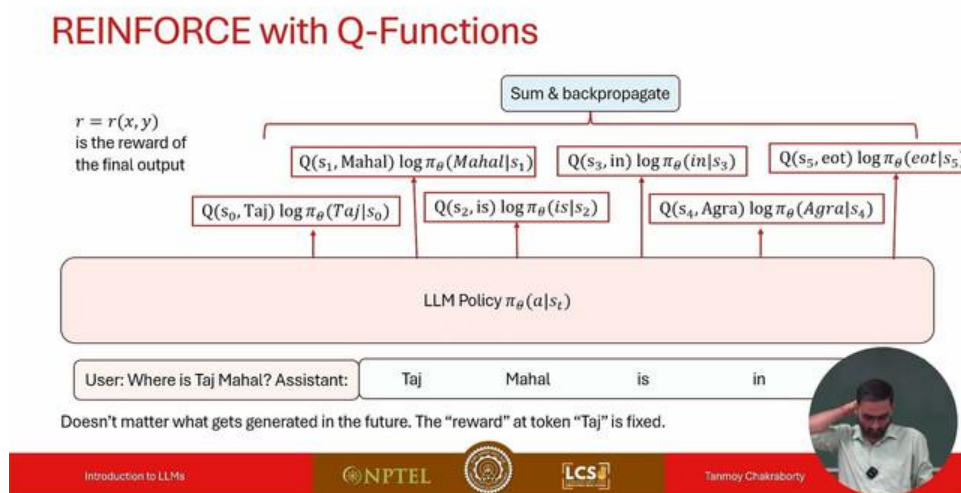
So why average, because I will I will take multiple such, you know, for such actions, and then I'll take the average or some expectation, right? Why this is discounted? This is

discounted, because what it will do, let's say you generate Taj Mahal is located in West Bengal, right? So and you want to calculate the importance of likelihood of the word Taj. So the word Mahal will impact more towards this likelihood. The word is will impact little bit less towards the word Taj, towards this likelihood and so on and so forth. And the word West Bengal will essentially impact least towards the likelihood of Taj. It means as we move on in the sequence, the importance of those words towards the current state action pair will reduce.

And we captured this using this gamma discounted factor. Okay. So gamma is a discount factor. So you see that the Q function of the state action pair is expectation of the reward of the first state action plus gamma of the second state action right plus gamma square r is t plus 2 a t plus 2 right third state action and gamma square gamma ranges between 0 to 1 let us say gamma is 0.9.

So this will be 0.9. This will be 0.81 and so on and so forth. So the importance of this will reduce.

Okay. And so on and so forth. So therefore discounted average, why average? Because we take expectation here. Why cumulative? Because we take the sum of all the rewards. So cumulative reward received at the state.



Okay. So what it is doing? Given the model theta you generate multiple such responses of size let us say size t from the state st. Let's say ST is the current state for the word Taj,

right? And action is the word Taj. And what are these sequences? The sequences can be Taj Mahal is located in West Bengal, Taj Mahal is located in Agra, Taj Mahal is located in Mumbai, and so on and so forth. These are all sequences.

Now for every sequence, you measure this cumulative reward, cumulative discounted reward. and you take the average. So this is Q function. Q function is a function of the state action pair whereas the value function is a function of the state only state. So what is value function? Value function is the average discounted cumulative reward received after reaching that state.

Okay so this is not a function of action and state by the way. So what is the difference between this equation and this equation? Only this part okay. So in order to generate the next state, next token, my state will be s_t plus one, you generate taj, right, which is my a_t . Okay, and you have this prompt. The next state is the so this is my s_t plus one.

From Q-Function to Advantage Function

- For text generation using language models

$$s_{t+1} = (s_t, a_t)$$
- That is, once you have generated the next token, the next state is determined completely.
- Hence, the Q-function for a state-action pair can be written as

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma V(s_{t+1})$$
- To further reduce variance, the advantage function $A(s_t, a_t)$ is used instead of Q-function

$$\begin{aligned} A(s_t, a_t) &= Q(s_t, a_t) - V(s_t) \\ &= r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t) \end{aligned}$$
- Intuitively, advantage function captures contribution of the action over an average action at the same state.

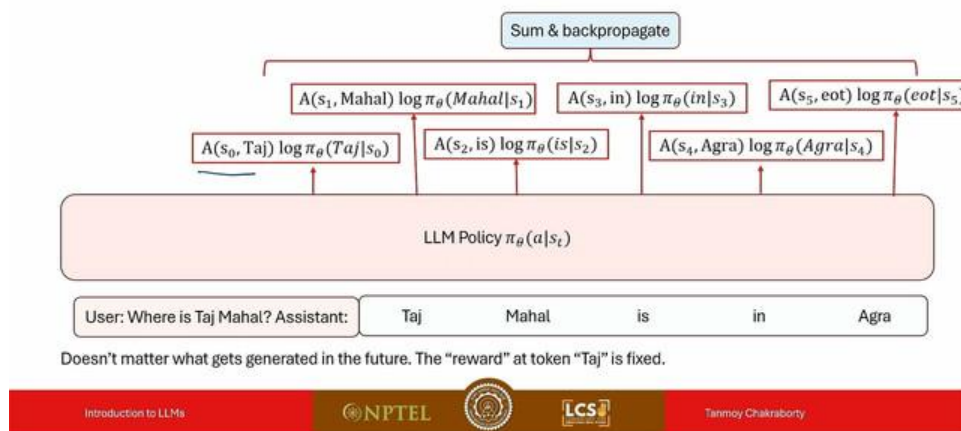


Okay, s_t was this one till this part. Okay. So Q function if you think of it carefully, if you look at these two equations carefully Q s_t at is the reward right, the reward is s_t at and this would be t plus 1, this would be t plus 1, okay, this would be t plus 1, why? Because v t plus 1 will start from t plus 1, okay, this t , this is also t , t plus 1 will start from t plus 1, right and what is, so to calculate this q , I need the reward at t and then the value in the next state, okay. So if you make some adjustment now you can use this Q function here in the

reward part okay but it turned out empirically that this Q function is not enough I need some other function which is called the advantage function. What is the advantage function? Advantage function is basically Q function minus the value function for stability for better stability okay.

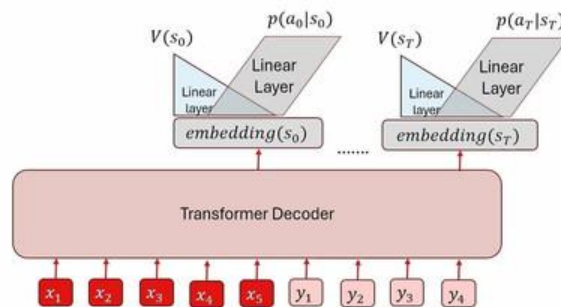
So advantage function captures the contribution of the action over an average action at the same state. It is very straight forward okay. If you look at this very carefully okay it is essentially the contribution of the action over all possible actions that you can think of at that particular state. So once this advantage function is calculated right now you replace this part which was earlier Q function before that it was reward right then it was Q function now this is advantage function. So the advantage function you replace all these rewards by advantage function Okay and then the remaining part is straight forward, the same process.

REINFORCE with Advantage Functions



So now this is called reinforced with advantage function. Now how do we implement this thing in neural network? So I need to learn this value function. It's a value. So what, what I can do on top of every embedding of the token, right? We have all these tokens now corresponding to every token, we can add a linear layer. Now this linear layer will essentially serve this linear layer will learn let's say this is parameterized by five and this will learn the value function.

Implementing the Value Function



Right now how do you learn this value function by the way because we need some sort of objective function right. So what is my objective function? So given the input x we sample a response y right from the policy model I have this linear layer which will which is parameters by ϕ which is responsible for calculating the value. Right and what is my object what is my ground truth value? The ground truth is essentially the reward. What is reward? So given this so I have this reward model right so given this response right I can calculate the reward the cumulative reward right discounted cumulative reward which is this one right the entire response to the entire response is known to me okay and this is called the reward to go and then my task would be to essentially move this value towards this RT right. So I will minimize this mean square error.


So minimize over all token, the, the value and the reward difference. Right. And through this, I mean, when we back prop, I will also update the linear layer, which is responsible for value value function. Okay. So this is my vanilla policy gradient method.

So we sample given a prompt. So we first sample a batch of prompts, right? B. Now given a prompt, you sample outputs, Y , right? For each Y , which is essentially A_1, A_2 dot dot A_T , you calculate the reward RT for every sequence, for every token A_T , right? Then you calculate the cumulative discounted reward RT right which is this one then you compute the value and advantage function A_T right that you already discussed right? So I will get

Q from our, I will get V from the linear layer. From there I can put that one test function and for every token, then what we'll do, we applied the reinforce method, right? and with this advantage function, this is my reinforced method, right? This is my reinforced method with advantage function. And we basically use to essentially train the value function, right? The linear layer that I mentioned, the linear layer. I'll calculate the MAC, the MAC, and this will help me get the parameters fine-tuned in this linear layer, okay? That's it, but there are still some problems, okay? So what are the problems? The problem is if you look at the reward carefully this part carefully what we are doing for every updated policy model I am generating this corresponding response okay isn't it? If you look at this algorithm here for every updated policy model.

From Q-Function to Advantage Function

- For text generation using language models

$$s_{t+1} = (s_t, a_t)$$

- That is, once you have generated the next token, the next state is determined completely.
- Hence, the Q-function for a state-action pair can be written as

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma V(s_{t+1})$$
- To further reduce variance, the advantage function $A(s_t, a_t)$ is used instead of Q-function

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

$$= r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)$$
- Intuitively, advantage function captures contribution of the action over an average action at the same state.



Vanilla Policy Gradient

- Repeat until convergence
 - Sample a batch of prompts B
 - For each prompt, sample one-or more outputs $y \in Y$
 - For each output $y = (a_1, \dots, a_T)$
 - Compute the reward r_t at each token a_t
 - Compute cumulative discounted reward R_t for each token
 - Compute the value & advantage function A_t for each token
 - Apply few gradient updates using REINFORCE with the advantage values computed above
 - Apply few gradient updates to train the value function by minimizing the MSE.

Credit: <https://spinningup.openai.com/en/latest/algo/trpo.html>

Introduction to LLMs

NPTEL



LCS

Tanmay Chakraborty



Problems

- Sampling from the policy after every update can be challenging.
- Solution: Sample from an older fixed policy instead

$$\begin{aligned}
 \mathbb{E}_{\pi_{\theta}(y|x)} r_s(x, y) &= \sum_{y \in Y} \pi_{\theta}(y|x) r_s(x, y) \times \left(\frac{\pi_{\theta_{\text{old}}}(y|x)}{\pi_{\theta}(y|x)} \right) \\
 &= \sum \pi_{\theta_{\text{old}}}(y|x) \left[\frac{\pi_{\theta}(y|x)}{\pi_{\theta_{\text{old}}}(y|x)} \right] r_s(x, y) \\
 &= \mathbb{E}_{\pi_{\theta_{\text{old}}}(y|x)} \underbrace{\left[\frac{\pi_{\theta}(y|x)}{\pi_{\theta_{\text{old}}}(y|x)} \right]}_{\text{importance weight}} r_s(x, y)
 \end{aligned}$$

Introduction to LLMs

NPTEL



LCS

Tanmay Chakraborty



So theta gets updated for every updated policy model. I need to generate a response from the response. I will calculate all these things, reward, advantage function and stuff like that. Now after every update, if you start generating things that would be time taking, right? So can we do this thing beforehand? Yes. So this is the idea.

The idea is so this is my reward function right what I will do? So first let me let us you know expand this expectation in terms of summation right. So this will be my probability and this is the reward, what I will do let us use an old model which is theta k. Theta k is the old model right. So I will use a term here which is the old model and the old model

essentially this will cancel out So I add this term okay and then I move this term here I move this term here and this term here. So π_{θ_k} will come here, π_{θ} will go here and then the reward right.

Problems

- Sampling from the policy after every update can be challenging.
- Solution: Sample from an older fixed policy instead

$$\begin{aligned}
 \mathbb{E}_{\pi_{\theta}(y|x)} r_s(x, y) &= \sum_{y \in \mathcal{Y}} \pi_{\theta}(y|x) r_s(x, y) \times \left(\frac{\pi_{\theta_k}(y|x)}{\pi_{\theta}(y|x)} \right) \\
 &= \sum_{y \in \mathcal{Y}} \pi_{\theta_k}(y|x) \left[\frac{\pi_{\theta}(y|x)}{\pi_{\theta_k}(y|x)} \right] r_s(x, y) \\
 &= \mathbb{E}_{\pi_{\theta_k}(y|x)} \left[\underbrace{\frac{\pi_{\theta}(y|x)}{\pi_{\theta_k}(y|x)}}_{\text{importance weight}} r_s(x, y) \right]
 \end{aligned}$$

θ_k



So now I am doing this expectation calculating the expectation with respect to the old model. So it means that given a prompt from the old model I will generate responses. Okay and I will measure the probability of those responses from the new model and from the old model. So I don't need to generate this responses from the new model I will use the old model generate responses calculate the probability with respect to the old model which is the denominator with respect to the new model right and then I have the reward. So this can be addressed okay and this component is called importance weight okay.

So this is the reinforced algorithm with importance weight. have been modifying this algorithm right again and again. So now along with the advantage function I have the importance weights. We are almost done. So in PPO, now this is reinforced algorithm, in PPO proximal policy optimization it turned out that you know these important functions, this importance weight, these important weights can change drastically.

If you do this thing for a specific batch of prompts and some fixed output, you will see a major change in terms of importance in the importance weight values. So we can bound it. We can bound it between one minus epsilon and one plus epsilon. What does it mean? It's

called PPO clip. What does it mean? It means that if the advantage function is positive, then I will take the mean of this and this.

So I clip. I don't allow this importance weight to be greater than this one. If the advantage function is negative, then I will take the maths of this and this. Okay. So I will not allow that the importance weight to essentially go below $1 - \epsilon$.

It's a trick essentially to make this thing stable. Okay. So what is my current algorithm? This is PPO clip, batch of prompts. We have for each prompt, we have samples. For every sample, we calculate the reward, right? For every token in the sample, we calculate the cumulative discounted reward.

The PPO-CLIP Algorithm

- For $k = 1$ to K
 - Sample a batch of prompts B
 - For each prompt, sample one-or more outputs from $\pi_{\theta_k}(y|x)$
 - For each output $y = (a_1, \dots, a_T)$
 - Compute the reward r_t at each token a_t
 - Compute cumulative discounted reward R_t for each token
 - Compute the value & advantage function A_t for each token
 - Apply few gradient updates using REINFORCE PPO-CLIP with the advantage values computed above
 - Apply few gradient updates to train the value function by minimizing the MSE.

Credit: <https://spinningup.openai.com/en/latest/algorithms/ppo.html>

Introduction to LLMs

NPTEL



LCS

Tanmoy Chakraborty



We calculate the advantage function, and then we use PPO clip. okay with the advantage values computed above and we use MSE for training the linear layer for value function okay. So things to remember log derivative trick is very important right. Log probability of the tokens should be weighted by the advantage function. Importance weights should be used, right? And importance weights should be clipped in PPO.

Things to Remember

- The log-derivative trick should be used to compute gradient in REINFORCE
- The log-probability of the tokens should be weighed by the advantage function to reduce variance
- Importance weights should be used to allow sampling from a fixed policy
- The importance weights should be clipped to prevent large gradient updates.



Okay, that's it. So this is about aligning the model. aligning the policy model with human feedback. Reward model is used to generate the reward. Reward model is trained with human feedback and reward model will help update the policy model.

Okay. With this I stop here and then the next lecture will be taken. Thank you.