Course Conclusion

Congratulations! You've successfully completed the Generative AI Advanced Fine-Tuning for LLMs course. Throughout this course, you have explored and applied cutting-edge techniques in instruction tuning, reward modeling, and reinforcement learning from human feedback (RLHF) using Hugging Face.

At this point, you know that:

- Instruction-tuning involves training models with expert-curated data sets.
- For instruction-tuning, the model requires instructions and answers.
- Instruction tuning helps perform a wide variety of tasks by interpreting and executing instructions more effectively.
- Instruction tuning uses three components: instructions, input, and output.
- Instruction masking focuses on the loss calculation of specific tokens.
    - Load a data set using CodeAlpaca 20k data set
    - Format the data set using the formating_prompts_func function
- For creating a formatted data set, use two code blocks for generating instructions with and without responses.
- To create a model, fine-tune Facebook's opt-350m model:
    - Define the collator using DataCollatorForCompletionOnlyLM to prepare data batches for training language models
    - Define the trainer by creating the SFTTrainer object
    - Generate a text pipeline from the transformers library
    - Evaluate the model's text generation using the BLEU score
- The reward model takes prompt as an input and response as an output regarding reward or score.
- Reward modeling helps in quantifying quality responses, guiding model optimization, incorporating reward preferences, and ensuring consistency and reliability of the responses.
- You learned to use the scoring function, enter two inputs, and select the factual and contextual response in the response evaluation process. Based on the generated responses, you'll get scores or rewards for the response.
- The scoring function takes the query and appends the chatbot's responses.

- A data set synthetic-instruct-gptj-pairwise from HuggingFace trains and evaluates instruction-following models.
- Defining the preprocess_function() helps format the keys and tokenize the data for the reward trainer.
- TrainingArguments class from the transformers library defines the training arguments.
- Reward trainers orchestrate the process, save, and evaluate the model using the trainer.train() method.
- The tokenizing process generates scores and compares the output of two functions to achieve the desired win rate.
- Reward model training is an advanced technique that trains a model to identify desired outputs generated by another model and assign scores to the outcome based on its relevance and accuracy.
- Training the scoring function helps generate rewards effectively.
- Generating reward model loss, the encoder model generates responses as contextual embeddings.
- Using the Bradley–Terry reward loss model, you can understand the reward loss model by generating the cost or loss function.
- Direct Preference Optimization (or DPO) is a reinforcement learning technique designed to fine-tune models based on human preferences more directly and efficiently than traditional methods.
- DPO involves collecting data on human preferences by showing users different outputs from the model and asking them to choose the better one.
- DPO involves three models: the reward function, which uses an encoder model, the target decoder, and the reference model.
- In DPO, you can convert a complex problem into a simpler objective function that is more straightforward to optimize.
- The partition function plays a crucial role in normalizing probabilities in custom probability functions.
- The reward function provides human feedback for the inserted query.
- Rollouts help queries and responses to review the sampling process.
- The expected rewards understand how an agent performs in the language model using an empirical formula.

- RLHF uses response distribution as an input query to fine-tune the pretrained LLMs.
- Pretrained reward model evaluates and generates a reward for the query plus response.
- Update the model parameters θ for each response generated by the agents.
- PPO provides feedback on the quality of actions taken by the policy.
- The scores from the sentiment analysis pipeline evaluate the quality of generated responses.
- Pipe_outputs list generates scores for the responses.
- The LengthSampler varies text lengths for data processing, enhances model robustness, and simulates realistic training conditions.
- Two main steps to fine-tuning a language model with DPO include:
    - Data collection
    - Optimization
- To fine-tune a language model with DPO and Hugging Face:
    - Step 1: Data preprocessing
        - Reformat
        - Define and apply the process function
        - Create the training and evaluation sets
    - Step 2: Create and configure the model and tokenizer
    - Step 3: Define training arguments and DPO trainer
    - Step 4: Plot the model's training loss
    - Step 5: Load the model
    - Step 6: Inferencing
- DPO leverages a closed-form optimal policy as a function of the reward to reformulate the problem
- Reward policy:

$$\pi_r(Y|X) = \frac{\pi_{ref}(Y|X)\exp\left(\frac{1}{\beta}r(X,Y)\right)}{Z(X)}$$

- Subtracting the reward model for two samples eliminates the need for the partition function

$$r(X,Y_w) - r(X,Y_l) = \beta \ln\left(\frac{\pi_r(Y_w|X)}{\pi_{ref}(Y_w|X)}\right) - \beta \ln\left(\frac{\pi_r(Y_l|X)}{\pi_{ref}(Y_l|X)}\right)$$

- Loss function:

$$-\sigma\left(\beta \ln\left(\frac{\pi_r(Y_w|X)}{\pi_{ref}(Y_w|X)}\right) - \beta \ln\left(\frac{\pi_r(Y_l|X)}{\pi_{ref}(Y_l|X)}\right)\right)$$

- The sample query questions may provide various random responses based on the probability distribution.
- The transformer model generates probabilities for different words using the softmax function.
- Selecting words at various timestamps, change the probabilities for those words.
- The generation parameters, such as temperature, top-k sampling, beam search, top-p sampling, repetition penalty, and max and min tokens, help change the sequences generated using LLMs.
- The transformer model generates probabilities for different words using the softmax function.
- Objective functions coordinate algorithms and data to reveal patterns, trends, and insights to produce accurate predictions.
- They measure the difference between an ML model's predicted outcomes and the actual target values.
- The Kullback–Leibler, or KL, divergence measures the difference between two probability distributions, the desired and the arbitrary policy.
- The optimal solution scales the reference model to the reward function, with the beta parameter controlling the constant.
- Following the policy distribution, the language model generates responses based on the inserted query.

- You can consider the relationship “Y follows the policy given X” to clarify the relationship between the policy and the language model as a function of omega.
- Rollouts are how the model generates different responses for each query.
- The rollout libraries, such as Hugging Face, differ from the reinforcement learning.
- The policy gradient method maximizes the objective function, and PPO helps to achieve this maximization.
- To optimize the policy, derive the sample response, estimate the reward, and extend the dataset.
- You can calculate the log derivative by identifying a policy that maximizes the objective function by simplifying the expression and converting it into analytical distributions.
- Use a toy gradient ascent example using stochastic gradient ascent or SAG to maximize the objective function compared to a standard optimization problem with maximum likelihood.
- A positive update occurs when a reward is positive, and a negative update occurs when a reward is negative.
- To train the model, regularly evaluate the model using human feedback, use the moderate beta value, and increase the temperature.