Experiment-1: Exploring Optimal Hyperparameter for Language Models

Objective: To investigate the effect of different values of temperature, top-p, and max tokens on language model outputs for various use cases.

Temperature: In the context of LLMs, "temperature" is a Hyperparameter that controls the randomness of predictions made by the model. It determines how the model samples from its probability distribution of possible next words.

How Temperature Works: Probability Distribution: After the model predicts the next word, it assigns a probability to each possible word based on its training data.

Temperature Adjustment:

- A low temperature (e.g., 0.1) makes the model choose the most probable word almost every time, leading to more deterministic and focused responses.
- A high temperature (e.g., 1.5) increases randomness, making the model more likely to explore less probable words, which can result in creative or unexpected responses.

Effects of Different Temperatures:

Low Temperature (e.g., 0.1):

- Produces highly consistent and deterministic outputs.
- Best for factual or logical tasks where precision is crucial.
- Responses might lack creativity or variation.

Medium Temperature (e.g., 0.7):

- Balances randomness and determinism.
- Suitable for most general-purpose conversations.

High Temperature (e.g., 1.5):

- Increases creativity and diversity.
- May produce more surprising or unique responses but risks incoherence or nonsensical outputs.

Practical Use:

- Controlled Output: Use a low temperature when you need precise answers (e.g., coding, technical explanations).
- Creative Output: Use a higher temperature for brainstorming, storytelling, or generating ideas.

By adjusting the temperature, you can fine-tune the behavior of an LLM to match the requirements of your task.

Task

Set Up the Environment

A. **Login to Hugging Face Playground:** Visit Hugging Face Playground.
B. **Choose a Language Model:** Select an appropriate model, e.g., ***gemma***, to run your tests.
C. **Keep Default Settings:** Do not modify settings like max tokens, top-p

Choose Your Prompt: Select a prompt that can generate varied and interesting responses.

- Example Prompts:
  - *"Describe a magical forest and its creatures."*
  - *"Write a motivational message for a student."*
  - *"What would happen if robots learned to paint like humans?"*

Test the Model with Different Temperatures

A. **Set Temperature Values:** Use these values sequentially: 0.2, 0.5, 0.7, 1.0, 1.3.
B. **Generate Responses:** Enter the same prompt for each temperature setting.

Record the Results

Temperature	Response	Creativity	Coherence	Relevance	Observations
0.2		*(1-5)*	*(1-5)*	*(1-5)*	
0.5		*(1-5)*	*(1-5)*	*(1-5)*	
0.7		*(1-5)*	*(1-5)*	*(1-5)*	
1.0		*(1-5)*	*(1-5)*	*(1-5)*	
1.3		*(1-5)*	*(1-5)*	*(1-5)*	

Creativity: How unique and diverse is the response?
Coherence: How logical and grammatically sound is the response?
Relevance: Does the response address the prompt accurately?

A. What happens to creativity, coherence, and relevance as the temp increases?
B. Which temperature setting worked best for your prompt?
C. What trade-offs did you notice (e.g., creativity vs. coherence)?
D. Recommend an ideal temperature setting for specific tasks like storytelling, summarization, or technical explanations.

Top-p (nucleus sampling) selects the smallest set of tokens whose cumulative probability is at least p, generating text only from this set. It balances diversity and relevance by dynamically adapting the token pool.

- **High top-p (e.g., 0.9–1.0)**: Increases creativity and diversity.
- **Low top-p (e.g., 0.7)**: Produces more focused and deterministic outputs.

Example: If the model predicts these token probabilities:

- "cat": 0.5; "dog": 0.3; "mouse": 0.1; "elephant": 0.05; "tiger": 0.05

For top-p = 0.9:

- The set {"cat", "dog", "mouse"} has a cumulative probability of 0.9.
- Sampling happens only from this set, ignoring "elephant" and "tiger".

Task

Fix one value of **temperature** and vary **top-p** to observe changes.
Repeat with different fixed **temperature** values.

- Suggested setup: **Temperature:** 0.2, 0.7, 1.0; **Top-p:** 0.1, 0.3, 0.5, 0.8, 1.0

Temperature	Top-p	Response	Creativity	Coherence	Relevance	Observations
0.2	0.1		*(1-5)*	*(1-5)*	*(1-5)*	
0.2	0.3		*(1-5)*	*(1-5)*	*(1-5)*	
0.2	0.5		*(1-5)*	*(1-5)*	*(1-5)*	
...	...		...	...	...	
1	1		*(1-5)*	*(1-5)*	*(1-5)*	

Max tokens in a language model refers to the maximum number of tokens (units of text) the model can process in one go, including both input and output.

- **Tokens** are chunks of text, like words or punctuation.
- **Max tokens** limit how much text the model can handle at once. For example, if the limit is 1,000 tokens and your input is 200 tokens, the model can generate up to 800 tokens in response.
- The token count includes both input (prompt) and output (generated response).

Max Tokens	Response Length	Input Size	Context Understanding	Performance	Cost
Low	Shorter, less detailed answers	Less room for long prompts	May lose context	Faster responses	Lower cost
High	Longer, detailed answers	More room for longer prompts	Better context retention	Slower responses	Higher cost

Task

Select a prompt to test the effect of max tokens. Example:

Set Max Tokens Values: Suggested values: 100, 300, 500, 1000, 2000 tokens

Evaluate the Output:

- Does the model maintain a logical and clear response as the token limit increases?
- At what token limit does the response lose focus or become too detailed?

Consider any four use cases of your choice. For each use case, determine the optimal values for temperature, top-p, and max tokens, and provide your observations in the table below

Use Case	Temperature	Top-P	Max Tokens	Observations
Code Generation				
Creative Writing				
Chatbot Responses				
Code Comment Generation				
Story Telling				
Technical Explanation				

Write Python script to handle the above tasks.

```
# Example prompt to test the models
prompt = "Write a motivational message for a student"

# List of temperature values to experiment with
temperatures = [0.2, 0.5, 0.7, 1.0, 1.3]

# Choose a model to experiment with (e.g., "GPT-2", "GPT-Neo", "GPT-J")
model_name = "gpt2"  # GPT-2 model

# Generate and print text for various temperatures
generated_texts = generate_text(model_name, prompt, temperatures)

# Display the results
for temp, text in generated_texts:
    print("\n" + "-"*50 + "\n")
    print(f"Temperature = {temp}:")
    print(text)
```