Introduction to Sampling

Alex van Vorstenbosch

2025-02-01

Google Colab

You can make these exercises using the general google colab notebook. Please take a few minutes to read through the notebook to familiarize yourself with the environment.

1 Exercise: Getting to know the generate_response function

In order to make interaction with the LLMs easier, I have written a helper function called `generate_response`. Run `help(generate_response)` to see the documentation of the function.

Experiment with the various parameters of LLM response:

- What happens when you adjust temperature to 0.4? And what happens when you adjust temperature to 0?
- What happens when you adjust max_tokens?

Answer:
Adjusting the temperature determines how much randomness is added to the model's output. A temperature of 0 will result in the model always choosing the most likely token, while a temperature of 1 will result in the model choosing tokens according to their probabilities. Thus a temperature below 1 will make the model more deterministic, while a temperature above 1 will make the model more random. 'llama-cpp' does not support temperatures above 1, so the maximum value is 1.

Adjusting the max_tokens determines how long the model response is allowed to be before we cut if off.

2 Exercise: Iterative prompt development on a movie review

As we have access to the sampling parameters of the LLM, we have a lot more control compared to the online interfaces such as ChatGPT and DeepSeek. We can set the temperature for our chat completion to 0, which ensures that the same prompt will gave the same result every time.

For each of the tasks below, finetune your prompt untill you are happy with the result

2.1 Load the review

It can be found under `./Day 1/exercises/data/movie-review-1.txt`

2.2 Task: Provide a synopsis of the review

2.3 Task: Make sure the synopsis of the review is at most 3 sentences long

2.4 Task: Extract the movie title, genre, year, director and actors from the review. Place this information at the top of the response.

2.5 Task: Extract a list of tips and tops of the movie, using only keywords

2.6 Task: Make the model return these fields in a nicely structured (and valid) .json format

For this you can use the flag `json_mode = True` in the `generate_response` function.

2.7 Task: Review number 2

Now apply the prompt you've crafted to the second movie review:
`./Day 1/exercises/data/movie-review-2.txt`
Does it work?

3 Exercise: accessing the sampling backbone - logits

While we cannot really look into the inner workings of the models, it can be informative to look at the outputs of the model. The logits are the raw outputs of the model, before they are transformed into probabilities. These are accessible to the user, and allow us to look into the sampling options of the model.

For this exercise you need to reload your model adding `logits_all=True` to the function call:

```
# Load you llm model
llm = Llama.from_pretrained(
    # Huggingface repo name
    repo_id="bartowski/Meta-Llama-3.1-8B-Instruct-GGUF",
    # select the quant file within the repo you want '*' is a wildcard selector
    filename="*Q6_K.gguf",
    n_gpu_layers=-1,
    n_ctx=32518, # this is 50 A4 pages of context window!
    verbose=False,
    logits_all=True
)
```

Use the `1.3 Clear GPU VRAM` chunk to clear your VRAM. So you can load the model again.

3.1 The quick Brown Fox

Using the low-level API for llama-cpp, we are going to look at what possible completions the model generates for the phrase: `The quick brown fox`. Use the code below to look at the completions. What do they look like? What happens when you adjust the temperature in block 2 where the probabilities are calculated? What happens when you change the temperature to 0.1? And what happens when you change the temperature to 2?

```
# ---------------------------
# 1. Generate a single token as completion
# ---------------------------
prompt = "The quick brown fox"

#tokenize() expects bytes, not raw strings
prompt_tokens = llm.tokenize(prompt.encode("utf-8"))

# Generate a single token as completion
response = llm(
    prompt=prompt,
    max_tokens=1,
    temperature = 0,
    echo=True
)

# ---------------------------
# 2. Retrieve and process the logits for the last token generated
```

```
# ----------------------------
logits = np.array(llm.eval_logits)  # shape: (vocab_size,)
last_token_logits = logits[-1]      # shape: (vocab_size,)

# Compute probabilities using softmax:
t = 0.8 #         sampling temperature
exp_logits = np.exp(last_token_logits/t)
probs = exp_logits / np.sum(exp_logits)

# ----------------------------
# 3. retrieve the tokens
# ----------------------------
size_vocab = llm.n_vocab()
token_indices = range(size_vocab)

tokens = []
for token_id in token_indices:
    try:
        token_str = llm.detokenize([token_id]).decode("utf-8")  # Convert to readable text
    except Exception:
        token_str = f"<{token_id}>"  # If decoding fails, use token ID as fallback
    tokens.append((token_id, token_str))

# ----------------------------
# 4. Return the N most likely continuations
# ----------------------------
n = 10
highest_indices = np.argsort(probs)[-1:-n-1:-1]
for i in range(0, n):  # Last n elements in the sorted highest indices
    index = highest_indices[i]
    token_str = tokens[index][1]
    prob = probs[index]
    print(f"index:{index} - token:{token_str} - prob:{prob}")
```

Answer:
index:35308 - token: jumps - prob:0.91480221985911
index:27096 - token: jumped - prob:0.047910475067383315
index:198 - token: - prob:0.006527466874359068
index:11 - token:, - prob:0.006012544180474189
index:374 - token: is - prob:0.005523037904949924
index:320 - token: (- prob:0.003918785286337486
index:25 - token:: - prob:0.002829161441488689
index:7940 - token: jump - prob:0.00070806648654585
index:574 - token: was - prob:0.0006192475005630696
index:1389 - token: – - prob:0.0005784195307390696

If we set the temperature to 0.1 99.9If we set the temperature to 2, the model will return the max probability token only 3.3

3.2 Chat completions

Now use the chat completion framework to let the model generate responses to a question. Give the model a name in the system prompt, and ask it what it's name is in the user prompt.

```
# specify the system message
system_role = """
<-- your input here -->
"""

# Provide your specific input

prompt = """
<-- your input here -->
"""

messages = [
    {"role": "system", "content": system_role},
    {"role": "user", "content": prompt}
]

# ---------------------------
# 1. Load your model and set up the prompt
# ---------------------------
# Use a short prompt so that we get the full logits distribution for each generated token.
# Note: tokenize() expects bytes.
prompt_tokens = llm.tokenize(prompt.encode("utf-8"))

# Generate several tokens; here we generate 1 tokens.
response = llm.create_chat_completion(
    messages=messages,
    max_tokens=1,
    temperature = 0)
```

Visualize the output probabilities in a bar-chart using the code provided below. Plotting all tokens is very slow, so we filter on a reasonable threshold value.

```
# ---------------------------
# 1. Filter and sort tokens based on probability
# ---------------------------
# Set a probability threshold. Tokens with probability below this will be skipped.
threshold = 0.0001

# Find indices where probability is at least the threshold.
filtered_indices = np.where(probs >= threshold)[0]
top_indices = filtered_indices
top_probs = probs[top_indices]

# ---------------------------
# 2. Retrieve token strings for the filtered tokens
# ---------------------------
tokens = []
for token_id in top_indices:
    try:
        token_str = llm.detokenize([token_id]).decode("utf-8")  # Convert to readable text
    except Exception:
```

```
        token_str = f"<{token_id}>"
    tokens.append(token_str)

# ---------------------------
# 3. Plot the results
# ---------------------------
fig, ax = plt.subplots(figsize=(15, 8))
# We use the filtered token indices (or their order in the new list) for plotting.
x_positions = np.arange(len(top_indices))
ax.bar(x_positions, top_probs, edgecolor='black')
ax.set_xlabel("Token (filtered and sorted)", fontsize=14)
ax.set_ylabel("Probability", fontsize=14)
ax.set_title("Next-Token Probability Distribution", fontsize=16)

# Annotate the top 10 tokens with the highest probabilities.
# Find the indices in the filtered array that correspond to the top 10 probabilities.
top10_order = np.argsort(top_probs)[-10:]
for idx in top10_order:
    token_str = tokens[idx]
    prob = top_probs[idx]
    ax.text(x_positions[idx], prob + 0.002, token_str, ha='center', fontsize=10, fontweight='bold')

plt.tight_layout()
plt.show()
```

Answer:

In this case I told the model it was called Tim. This is clearly picked up by the model.

3.3 random number generator

Again use the chat completions framework:

- Specify in the system prompt that the model should generate a random number between 0 and 9
- Ask the model to generate a random number in the user prompt

Visualize the output probabilities in a bar-chart using the code provided above. Is this what you would expect? Can you get the distribution to change?

Answer:
