LLM API Overview

Here I compare the APIs for GPT-3.5 through Open-AI and GPT-2 through Huggingface. I also already have approval for LLAMA2, however, the computations take around 10 minutes for a single prompt on my laptop. I need to move it to e.g. Google Colab. Hence, some of the tests I did are missing so far. However, it seems like the Hugging Face implementation of LLAMA2 supports the same API endpoints as the one for GPT-2, at least the ones we are interested in so far.

Questions:

- Can we make the output deterministic?
- Can we get the loss?
- Can we get the logits?

For the Open-AI API, we need to have an `OPENAI_API_KEY` environment variable set. Also, there has been a recent change in the API when it comes to text generation. To me, it seems, that the legacy API is the only way to get the log probabilities. This is risky, as the legacy API might be deprecated at some point. According to Open AI, this feature will be added to the new API "in the next few weeks".

Speed comparison: Open-AI GPT-3.5 takes seconds to evaluate and does not require a download (but costs credits), Hugging Face GPT-2 takes a couple of seconds to evaluate and is around a gigabyte large, locally. LLAMA2 takes around 10 minutes to evaluate and is around 10 gigabytes large, locally. I am already using the MPS speed-up for M1 Macs.

I am currently waiting for approval for the LLAMA2 model such that I can use it through the `transformers` API. I have not checked yet whether all necessary features that are available for GPT-2 are also available for LLAMA2. This is for the smallest models of each type.

At the end of the document are some more bullet points and links to resources I found worth saving.

Define a test string and specify a model to use for the test. This is the running example. `gpt-2` is the smallest GPT-2 model on Hugging Face.

```
# Define a test string and specify a model to use for the test.
s_string = "Finn writes code"
s_model = 'gpt2'
s_model_openai = 'gpt-3.5-turbo-instruct'
```

```
import numpy as np
import pandas as pd
from transformers import GPT2Tokenizer, AutoModelForCausalLM

# define tokenizer, model and tokenize inputs
tokenizer = GPT2Tokenizer.from_pretrained(s_model)
tokenizer.pad_token_id = tokenizer.eos_token_id
model = AutoModelForCausalLM.from_pretrained(s_model)

inputs = tokenizer([s_string], return_tensors="pt")
```

Can we make the output deterministic?

- for Open-AI: Kind of, by setting the `seed` and the `temperature` to 0.0. We need to keep an eye on the `fingerprint` of the model (model version at Open-AI) to make sure that the model does not change. This is only "almost" deterministic.
- For Hugging Face GPT-2: Yes, by setting `top_k` to 1
- For LLAMA2: It has the `top_k` parameter, suggesting that it can be deterministic

Can we get the loss?

Open-AI GPT3.5

I have not found that option yet.

Hugging Face-GPT2

We can make a forward pass for inputs and obtain the loss.

```
outputs2 = model(**inputs, labels=inputs["input_ids"])
print(outputs2.loss)
```

```
tensor(10.1251, grad_fn=<NllLossBackward0>)
```

Hugging Face-LLAMA2

Yes.

```
# Calculate logits and loss from the model using unpadded input
loss1 = model(
    input_prompt_tokenized.input_ids,
    attention_mask=input_prompt_tokenized.attention_mask,
    labels=input_prompt_tokenized.input_ids
).loss
```

Can we get the logits?

Open-AI GPT3.5

We can also get the top n log probs per token.

```
from openai import OpenAI
client = OpenAI()

# Have to have an OPENAI_API_KEY environment variable set
response = client.completions.create(
  model=s_model_openai,
  prompt=s_string,
  logprobs = 1,
  temperature = 0,
  seed = 0
)
```

```
print("OPEN-AI MESSAGE:\n")
print(response.choices[0].text)
```

```
OPEN-AI MESSAGE:



Finn is a programmer who writes code for a living. He spends most
```

```
# max log probs
pd.DataFrame({'logprobs': response.choices[0].logprobs.token_logprobs,
'token': response.choices[0].logprobs.tokens})
```

	logprobs	token
0	-1.670775	\n\n
1	-2.232528	F
2	-0.017235	inn
3	-1.111077	is
4	-0.727976	a
5	-1.421415	programmer
6	-0.313894	who
7	-0.859753	writes
8	-0.122264	code
9	-0.930351	for
10	-0.784002	a
11	-0.008029	living
12	-0.002123	.
13	-0.038453	He
14	-0.416001	spends
15	-0.480581	most

Hugging Face-GPT2

Read the message in the table, top to bottom.

```
# Example 1: Print the scores for each token generated with Greedy Search
outputs = model.generate(**inputs,
max_new_tokens=5,
top_k = 1,
return_dict_in_generate=True,
output_scores=True)

transition_scores = model.compute_transition_scores(
    outputs.sequences,
    outputs.scores,
    normalize_logits=True
)
# input_length is the length of the input prompt for decoder-only models,
# like the GPT family, and 1 for
```

```
# encoder-decoder models, like BART or T5.
input_length = 1 if model.config.is_encoder_decoder else inputs.input_ids.shape[1]
generated_tokens = outputs.sequences[:, input_length:]
```

```
import pandas as pd
import numpy as np

# Iterate over the generated tokens and transition scores
df = pd.DataFrame({'token': generated_tokens[0].numpy(),
'trans_scores': transition_scores[0].numpy()})
df['token_str'] = df['token'].apply(lambda x: tokenizer.decode(x))
df['trans_prob'] = df['trans_scores'].apply(lambda x: np.exp(x))
df = df[['token', 'token_str', 'trans_scores', 'trans_prob']]
df
```

	token	token_str	trans_scores	trans_prob
0	284	to	-1.336478	0.262770
1	787	make	-3.182834	0.041468
2	340	it	-1.969614	0.139511
3	4577	easier	-2.133453	0.118428
4	284	to	-0.470468	0.624710

Hugging Face-LLAMA2

Yes.

```
logits1 = model(
    input_prompt_tokenized.input_ids,
    attention_mask=input_prompt_tokenized.attention_mask,
    labels=input_prompt_tokenized.input_ids
).logits
```

Can we pass an input embedding instead of a text?

Open-AI GPT3.5

I have not found that option yet.

Hugging Face-GPT2

```
# create embedding. Could be any embedding with right dimension.
embeds = model.get_input_embeddings()(tokenizer.encode(s_string, return_tensors="pt"))

# Pass as input to the model and generate text.
sample = model.generate(inputs_embeds=embeds,
                        max_new_tokens= 5,
                        do_sample=True,
                        top_k=1,
                        num_return_sequences= 3,
                        output_scores=True)

for i, token in enumerate(sample):
    print(f"{i}: {tokenizer.decode(token, skip_special_tokens=True)}")
```

```
0:  to make it easier to
1:  to make it easier to
2:  to make it easier to
```

Notice how the output is the same, as we set it to be deterministic.

Hugging Face-LLAMA2

It seems like there is also the input parameter `inputs_embeds` for LLAMA2. See here and here.

```
# INPUT = ...embedding of a sequence, ensuring that there are no pad tokens
output_sequences = LLaMA.generate(
    inputs_embeds=INPUT.to(device)
    pad_token_id=tokenizer.pad_token_id,
    # ... generation parameters, top_p top_k etc.
)
```

Notes

- e.g. "\n\n" and " \n\n" are different
- Have to use legacy API to get the logs (risky?)

- Seed parameter (almost) always the same output for the same settings, even more "almost" with temperature=0.0?
- "We're also launching a feature to return the log probabilities for the most likely output tokens generated by GPT-4 Turbo and GPT-3.5 Turbo in the next few weeks, which will be useful for building features such as autocomplete in a search experience."
- Also save the `system_fingerprint`, to keep track of the state of the model. If the model itself gets updated the same seed might yield different results.
- Can set the seed, but not the fingerprint
- Can run LLAMA2 locally, at least the smallest version?
- With OpenAI API, we cannot access the first input layer directly, have to go through prompts
- "inputs_embeds (torch.FloatTensor of shape (batch_size, sequence_length, hidden_size), optional) — Optionally, instead of passing input_ids you can choose to directly pass an embedded representation. This is useful if you want more control over how to convert input_ids indices into associated vectors than the model's internal embedding lookup matrix."
- For GPT-2 we can make the output deterministic
- We can give in an input embedding instead of a tokenized phrase `model(inputs_embeds=embeds)`
- Thread on logit scores and their different variants
  - Transition scores: "transition_scores contains scores for the tokens that were selected at generation time. You can set normalize_logits=True to ensure they are normalized at a token level (i.e. to ensure the sum of probabilities for all vocabulary at a given generation step is 1)."
- An exploration of GPT-2's embedding weights