Conversational AI Chatbot with Transformers in Python

Learn how to use Huggingface transformers library to generate conversational responses with the pretrained DialoGPT model in Python.

Abdou Rockikz · 13 min read · Updated Jan 2022 · 5.9K · Machine Learning · Natural Language Processing

Chatbots have gained a lot of popularity in recent years. As the interest grows in using chatbots for business, researchers also did a great job on advancing conversational AI chatbots.

In this tutorial, we'll use the Huggingface transformers library to employ the pre-trained DialoGPT model for conversational response generation.

DialoGPT is a large-scale tunable neural conversational response generation model trained on 147M conversations extracted from Reddit. The good thing is that you can fine-tune it with your dataset to achieve better performance than training from scratch.

This tutorial is about text generation in chatbots and not regular text. If you want open-ended generation, see this tutorial where I show you how to use GPT-2 and GPT-J models to generate impressive text.

Alright, to get started, let's install `transformers`:

```
$ pip3 install transformers
```

Open up a new Python file or notebook and do the following:

```
from transformers import AutoModelForCausalLM, AutoTokenizer
import torch

# model_name = "microsoft/DialoGPT-large"
model_name = "microsoft/DialoGPT-medium"
# model_name = "microsoft/DialoGPT-small"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)
```

There are three versions of DialoGPT; small, medium, and large. Of course, the larger, the better, but if you run this on your machine, I think small or medium fits your memory with no problems. I tried loading the large model, which takes about 5GB of my RAM. You can also use Google Colab to try out the large one.

Generating Responses with Greedy Search

In this section, we'll be using the greedy search algorithm to generate responses. We select the chatbot response with the highest probability of choosing on each time step.

Let's make code for chatting with our AI using greedy search:

```
# chatting 5 times with greedy search
for step in range(5):
    # take user input
    text = input(">> You:")
    # encode the input and add end of string token
    input_ids = tokenizer.encode(text + tokenizer.eos_token, return_tensors="pt")
    # concatenate new user input with chat history (if there is)
```


Tags
Machine Learning
Ethical Hacking
General Python Tutorials
Web Scraping
Computer Vision
Python Standard Library
Application Programming Interfaces
Database
Finance
Packet Manipulation Using Scapy
Natural Language Processing
Healthcare
Web Programming
PDF File Handling
Python for Multimedia

```
    bot_input_ids = torch.cat([chat_history_ids, input_ids], dim=-1) if step > 0 else input_ids
    # generate a bot response
    chat_history_ids = model.generate(
        bot_input_ids,
        max_length=1000,
        pad_token_id=tokenizer.eos_token_id,
    )
    #print the output
    output = tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip_special_tokens=True)
    print(f"DialoGPT: {output}")
```

Let's explain the core of this code:

- We first take input from the user for chatting.
- We encode the text to `input_ids` using the DialoGPT tokenizer, we also append the end of the string token and return it as a Pytorch tensor.
- If this is the first time chatting with the bot, we directly feed `input_ids` to our model for a generation. Otherwise, we append the chat history using concatenation with the help of `torch.cat()` method.
- After that, we use the `model.generate()` method for generating the chatbot response.
- Lastly, as the returned output is a tokenized sequence too, we decode the sequence using `tokenizer.decode()` and set `skip_special_tokens` to `True` to make sure we don't see any annoying special tokens such as `<|endoftext|>`. Also, since the model returns the whole sequence, we skip the previous chat history and print only the newly generated chatbot answer.

Below is a sample discussion with the bot:

```
>> You:How can you be rich so quickly?
DialoGPT: I'm not rich, I'm just a rich man.
>> You:Ok, you don't make sense, how old are you ?
DialoGPT: I'm not a rich man, I'm just a rich man.
>> You:How can you be happy ?
DialoGPT: I'm not happy, I'm just a rich man.
>> You:But they say money buys happiness!
DialoGPT: Money buys happiness, but not happiness itself.
>> You:You don't make sense again!
DialoGPT: I'm not happy, I'm just a rich man.
```

You see the model repeats a lot of responses, as these are the highest probability, and it is choosing it every time.

By default, `model.generate()` uses greedy search algorithm when no other parameters are set. In the following sections, we'll be adding some arguments to this method to see if we can improve the generation.

Learn also: *How to Train BERT from Scratch using Transformers in Python.*

Generating Responses with Beam Search

Beam search allows us to reduce the risk of missing high probability sequences by keeping the most likely `num_beams` of hypotheses at each time step and then taking the sequences that have the overall highest probability, below code will generate chatbot responses with beam search:

```
# chatting 5 times with beam search
for step in range(5):
    # take user input
    text = input(">> You:")
    # encode the input and add end of string token
    input_ids = tokenizer.encode(text + tokenizer.eos_token, return_tensors="pt")
    # concatenate new user input with chat history (if there is)
    bot_input_ids = torch.cat([chat_history_ids, input_ids], dim=-1) if step > 0 else input_ids
    # generate a bot response
    chat_history_ids = model.generate(
        bot_input_ids,
        max_length=1000,
        num_beams=3,
        early_stopping=True,
        pad_token_id=tokenizer.eos_token_id
    )
    #print the output
    output = tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip_special_tokens=True)
    print(f"DialoGPT: {output}")
```

When setting `num_beams` to `3` in `model.generate()` method, then we're going to select three words at each time step and develop them to find the highest overall probability of the sequence, setting `num_beams` to 1 is the same as greedy search.

Below is a sample discussion with the chatbot using beam search:

```
>> You:How can I be rich?
DialoGPT: You can't.
>> You:Why not?
DialoGPT: Because you can't.
>> You:Then how am I supposed to live well?
DialoGPT: You can't.
>> You:So basically, you want me to be a poor guy?
DialoGPT: No, I want you to be a rich guy.
>> You:Ok how to be rich, then?
DialoGPT: I don't know.
```


New Tutorials

- Recommender Systems using Association Rules Mining in Python
- Customer Churn Prediction: A Complete Guide in Python
- How to Convert Pandas Dataframes to HTML Tables in Python
- Credit Card Fraud Detection in Python
- How to Inject Code into HTTP Responses in the Network in Python

Popular Tutorials

- How to Convert Speech to Text in Python
- How to Encrypt and Decrypt Files in Python
- How to Read Emails in Python
- How to Transfer Files in the Network using Sockets in Python
- How to Make a Keylogger in Python

Generating Responses with Sampling

We used beam and greedy search in previous sections to generate the highest probability sequence. Now that's great for tasks such as machine translation or text summarization where the output is predictable. However, it is not the best option for an open-ended generation as in chatbots.

For a better generation, we need to introduce some randomness where we sample from a wide range of candidate sequences based on probabilities:

```
# chatting 5 times with sampling
for step in range(5):
    # take user input
    text = input(">> You:")
    # encode the input and add end of string token
    input_ids = tokenizer.encode(text + tokenizer.eos_token, return_tensors="pt")
    # concatenate new user input with chat history (if there is)
    bot_input_ids = torch.cat([chat_history_ids, input_ids], dim=-1) if step > 0 else input_ids
    # generate a bot response
    chat_history_ids = model.generate(
        bot_input_ids,
        max_length=1000,
        do_sample=True,
        top_k=0,
        pad_token_id=tokenizer.eos_token_id
    )
    #print the output
    output = tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip_special_tokens=True)
    print(f"DialoGPT: {output}")
```

This time, we set `do_sample` to `True` for sampling, and we set `top_k` to `0` indicating that we're selecting all possible probabilities, we'll later discuss `top_k` parameter.

Here is a chat with these parameters:

```
>> You:Hello, how can I be rich?
DialoGPT: You can't
>> You:Really? then how others get rich?
DialoGPT: They work as money artists
>> You:How to be a money artist?
DialoGPT: I don't know, I don't have enough knowledge on artists
>> You:Alright, thank you!
DialoGPT: No problem, if you need tips pm me.
>> You:Great, you're my savior!
DialoGPT: Thank you for helping a stranger out like me!
```

There are some improvements. However, sampling on an exhaustive list of sequences with low probabilities can lead to random generation (like you see in the last sentence).

To improve it furthermore, we can:

- Lower the sampling `temperature` that helps us decrease the likelihood of picking low probability words and increase the likelihood of picking high probability words.
- Use Top-k sampling instead of picking all probable occurrences. This will help us to discard low probability words from getting picked.

```
# chatting 5 times with Top K sampling & tweaking temperature
for step in range(5):
    # take user input
    text = input(">> You:")
    # encode the input and add end of string token
    input_ids = tokenizer.encode(text + tokenizer.eos_token, return_tensors="pt")
    # concatenate new user input with chat history (if there is)
    bot_input_ids = torch.cat([chat_history_ids, input_ids], dim=-1) if step > 0 else input_ids
    # generate a bot response
    chat_history_ids = model.generate(
        bot_input_ids,
        max_length=1000,
        do_sample=True,
        top_k=100,
        temperature=0.75,
        pad_token_id=tokenizer.eos_token_id
    )
    #print the output
    output = tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip_special_tokens=True)
    print(f"DialoGPT: {output}")
```

Now, we set `top_k` to `100` to sample from the top `100` words sorted descendingly by probability. We also set `temperature` to `0.75` (default is `1.0`) to give a higher chance of picking high probability words, setting the temperature to `0.0` is the same as greedy search; setting it to infinity is the same as completely random.

Here is a discussion with these parameters:

```
>> You:How can I get rich?
DialoGPT: I'd also like to know this.
>> You:So you don't know it?
DialoGPT: I know it, but it doesn't matter.
>> You:Really? money matters, man!
DialoGPT: Well, money is money and I gotta get my money.
>> You:Yeah how to get them?
DialoGPT: Well, that's a question. I'd like to know more, just for curiosity's sake, to know what other people have said about how I sho
>> You:Some robots like you told me you have to buy gold, silver or stocks.
DialoGPT: I think my wallet is empty.
```

As you can see, it is much better now; feel free to tweak `temperature` and `top_k` parameters and see if it can improve it.

Related: *How to Paraphrase Text using Transformers in Python.*

Nucleus Sampling

Nucleus sampling or Top-p sampling chooses from the smallest possible words whose cumulative probability exceeds the parameter `p` we set.

Below is an example using Top-p sampling:

```
# chatting 5 times with nucleus sampling & tweaking temperature
for step in range(5):
    # take user input
    text = input(">> You:")
    # encode the input and add end of string token
    input_ids = tokenizer.encode(text + tokenizer.eos_token, return_tensors="pt")
    # concatenate new user input with chat history (if there is)
    bot_input_ids = torch.cat([chat_history_ids, input_ids], dim=-1) if step > 0 else input_ids
    # generate a bot response
    chat_history_ids = model.generate(
        bot_input_ids,
        max_length=1000,
        do_sample=True,
        top_p=0.95,
        top_k=0,
        temperature=0.75,
        pad_token_id=tokenizer.eos_token_id
    )
    #print the output
    output = tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0], skip_special_tokens=True)
    print(f"DialoGPT: {output}")
```

We set `top_k` to `0` to disable Top-k sampling, but you can use both methods, which works better. Here is a chat:

```
>> You:How can I be rich fast?
DialoGPT: Inflation and free money
>> You:lol, but that takes time, I need fast money.
DialoGPT: Your post makes no sense. You just need a lot of money to be rich fast
>> You:Ok how to get a lot of money?
DialoGPT: Buy gold and then trade it for money.
>> You:Hmm, I got your point, thanks.
DialoGPT: No problem. You have to have money to buy gold, but its only good for trading it.
>> You:Yeah, I understand, thank you so much.
DialoGPT: Ok, let me know if you need help
```

Now the chatbot clearly makes sense in a lot of cases.

Now let's add some code to generate more than one chatbot response, and then we choose which response to include in the next input:

```
# chatting 5 times with nucleus & top-k sampling & tweaking temperature & multiple
# sentences
for step in range(5):
    # take user input
    text = input(">> You:")
    # encode the input and add end of string token
    input_ids = tokenizer.encode(text + tokenizer.eos_token, return_tensors="pt")
    # concatenate new user input with chat history (if there is)
    bot_input_ids = torch.cat([chat_history_ids, input_ids], dim=-1) if step > 0 else input_ids
    # generate a bot response
    chat_history_ids_list = model.generate(
        bot_input_ids,
        max_length=1000,
        do_sample=True,
        top_p=0.95,
        top_k=50,
        temperature=0.75,
        num_return_sequences=5,
```

```
        pad_token_id=tokenizer.eos_token_id
    )
    #print the outputs
    for i in range(len(chat_history_ids_list)):
      output = tokenizer.decode(chat_history_ids_list[i][bot_input_ids.shape[-1]:], skip_special_tokens=True)
      print(f"DialoGPT {i}: {output}")
    choice_index = int(input("Choose the response you want for the next input: "))
    chat_history_ids = torch.unsqueeze(chat_history_ids_list[choice_index], dim=0)
```

I have set `num_return_sequences` to `5` to return five sentences at a time, we have to choose the one included in the following sequence. Here is how it went:

```
>> You:Hello, friend.
DialoGPT 0: Hey there!
DialoGPT 1: I am your friend, and I am also your friend.
DialoGPT 2: Hello. :
DialoGPT 3: Hiya! How are you?
DialoGPT 4: Hello. I like your username.
Choose the response you want for the next input: 3
>> You:Fine, thanks, what about you?
DialoGPT 0: Good. Not much to do.
DialoGPT 1: Not much, you?
DialoGPT 2: Fine, thanks.
DialoGPT 3: I'm okay, thank you for asking.
DialoGPT 4: I'm fine too.
Choose the response you want for the next input: 2
>> You:Can you tell me how can I get rich quickly?
DialoGPT 0: Yes, I can!
DialoGPT 1: Yes, just look up how to get rich on Google.
DialoGPT 2: Just keep doing your thing. You'll get rich.
DialoGPT 3: No, I can't.
DialoGPT 4: no, i have no money
Choose the response you want for the next input: 1
>> You:Yeah, but I thought you have an answer...
DialoGPT 0: I'm getting married in the morning.
DialoGPT 1: How do you get rich?
DialoGPT 2: I think there are many good answers, but I could send you one, if you want.
DialoGPT 3: I will, thanks!
DialoGPT 4: I thought you have an answer.
Choose the response you want for the next input: 2
>> You:Yes, please send me.
DialoGPT 0: You can send me link.
DialoGPT 1: I am very interested
DialoGPT 2: I sent you a PM
DialoGPT 3: I'll send you a PM
DialoGPT 4: I am always interested in new ideas.
Choose the response you want for the next input: 2
```

Conclusion

And there you go. I hope this tutorial helped you out on how to generate text on DialoGPT and similar models. For more information on generating text, I highly recommend you read the How to generate text with Transformers guide.

I'll leave you tweaking the parameters to see if you can make the bot performs better.

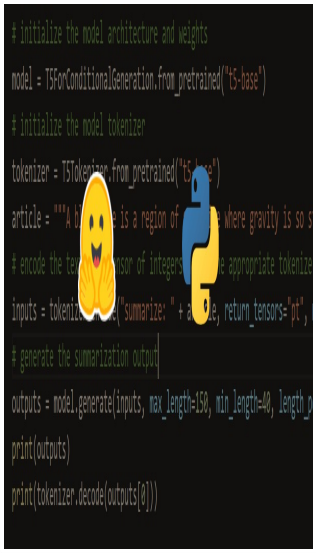
Also, a great and exciting challenge for you is combining this with text-to-speech and speech-to-text tutorials to build a virtual assistant like Alexa, Siri, and Cortana!

Learn also: *How to Fine Tune BERT for Text Classification using Transformers in Python.*

Happy learning ♥

VIEW FULL CODE | VIEW ON SKILLSHARE

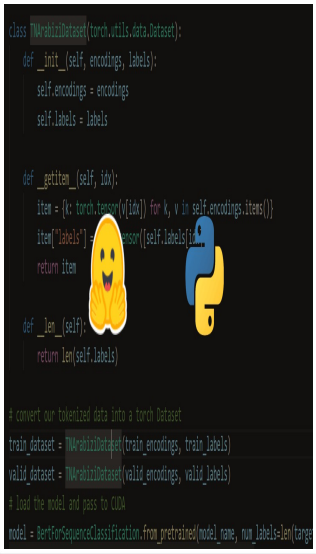
Read Also



How to Perform Text Summarization using Transformers in Python

Learn how to use Huggingface transformers and PyTorch libraries to summarize long text, using pipeline API and T5 transformer model in Python.

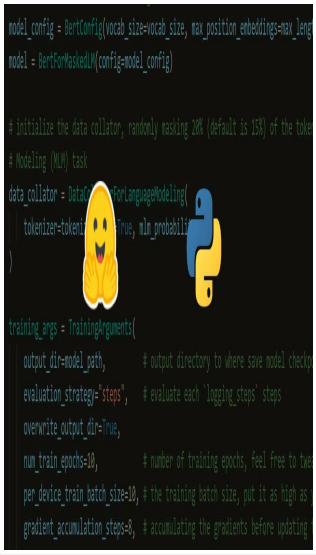
VISIT →



How to Fine Tune BERT for Text Classification using Transformers in Python

Learn how to use HuggingFace transformers library to fine tune BERT and other transformer models for text classification task in Python.

VISIT →



How to Train BERT from Scratch using Transformers in Python

Learn how you can pretrain BERT and other transformers on the Masked Language Modeling (MLM) task on your custom dataset using Huggingface Transformers library in Python

VISIT →

Comment panel

Aravind 11 months ago

REPLY

Thanks Abdou...Hope you continue with other variations in chatbot development. :)

Abdou Rockikz 11 months ago

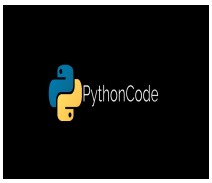
REPLY

Thanks Aravind! This is currently the state of the art chatbot among researchers, I may make another tutorial for fine tuning with other chatbot datasets soon.

jamie kidd 7 months ago

REPLY

Hi
I'm having trouble installing torch in pycharm



Abdou Rockikz 6 months ago

REPLY

Hello Jamie,
I do not recommend using PyCharm on machine learning and deep learning projects (at least that's my opinion). I would go for simply VS code or Jupiter notebooks. So you'll have to install torch via pip and not the PyCharm package manager.
Hope this helps.

Mouse 3 months ago

REPLY

Hi, really nice article!

I managed to get it working, but my IDE is throwing an error for 1 line, and when I try and modify the code (I'm trying to create a function instead of the for loop) it breaks because of this 1 line.

>> bot_input_ids = torch.cat([chat_history_ids, input_ids], dim=-1) if step > 0 else input_ids

I'm not sure where chat_history_ids is coming from? They seem to be created later with: chat_history_ids = model.generate(..)

So this line feels like a chicken and egg scenario? Is that what the for loop is handling with: if step > 0 else input_ids ?

IDE complains: Undefined name 'chat_history_ids'

I'm a bit confused about where the chat history is stored and how chat_history_ids is being called by bot_input_ids before it's been created (if that makes sense)

Even though when I run the code 'as is' it seems to run fine; I keep running into trouble with that line when trying to make my own tweaks (I'm trying to write it as a function)

I'm still learning so apologie

Abdou Rockikz 3 months ago

REPLY

Hey there!
First of all, thanks for the kind words! Appreciate your support.
You're right, chat_history_ids does not exist at the beginning, and "torch.cat([chat_history_ids, input_ids], dim=-1)" statement won't be called because the step is 0 and therefore bot_input_ids will be identical to input_ids since it's the first time we chat with the bot there isn't any chat history. (Python is dynamically typed and these kinds of errors won't raise, only some IDEs throws some warnings).
If the step is above 1, then we concatenate the chat history with the new user input, and so on. Hope this makes sense.

Mouse 3 months ago

REPLY

Hey Abdou!

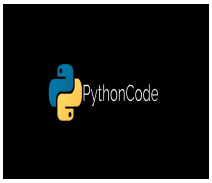
Thanks for the reply, you have indeed confirmed what I suspected about my IDE.
It seems like a silly question in hindsight, but I really wasn't sure if I was missing something.

This has been a great Python learning exercise for me, and I actually managed to get two chat bots talking to each other; one using your code and DialoGPT and the other using the Chatterbot Python library. (Just for fun and out of curiosity)

Some of the conversations they had were quite interesting and varied, and some were hilarious. I guess this is a consequence of training an AI on Reddit data!

I still have to prompt the first bot with human input first though; but after that I can pipe the bots replies to each other indefinitely or until Spacy crashes chatterbot.

It did get me thinking about how I'd go about getting the first bot to start the conversation, rather than me.
It was at this point that I fell down a deep philosophical black hole regarding how humans actually do this, and how to av



Abdou Rockikz 2 months ago

REPLY

Oh, It seems the character length stopped you from resuming sharing your idea, sorry for that!

A great project you did there! hmmm... I can't think of a way of starting the conversation automatically, and I guess human input is necessary.

Your email address will not be published.

Enter your name

Enter Email

Write a comment...

I'm not a robot

reCAPTCHA
Privacy - Terms

Subscribe for our newsletter

COMMENT