```
!pip install datasets
```

```
!pip install transformers[sentencepiece]
```

```
!pip install evaluate
```

```
!pip install Trainer
```

```
!pip install --upgrade protobuf
```

```
from transformers import AutoTokenizer, AutoModelForCausalLM, AutoModelWithLMHead

tokenizer = AutoTokenizer.from_pretrained("microsoft/DialoGPT-small", num_labels=2, gradier

model = AutoModelWithLMHead.from_pretrained("microsoft/DialoGPT-small")
```

```
from datasets import load_dataset
import torch

dataset = load_dataset("csv", data_files="output_file.csv")
```

```
dataset = dataset["train"].train_test_split(test_size=.2)
dataset
```

```
    DatasetDict({
        train: Dataset({
            features: ['question', 'answer'],
            num_rows: 1128
        })
        test: Dataset({
            features: ['question', 'answer'],
            num_rows: 283
        })
    })
```

```
# No padding token in tokenizer
# This is the token GPT2 uses from a quick google
tokenizer.pad_token = tokenizer.eos_token

def tokenize_function(data):
  return tokenizer(data["question"], padding="max_length", truncation=True, max_length=128)

tokenized_datasets = dataset.map(tokenize_function, batched=True)
```

1m 19s completed at 10:10 PM

```
tokenized_datasets
```

```
DatasetDict({
    train: Dataset({
        features: ['question', 'answer', 'input_ids', 'attention_mask'],
        num_rows: 1128
    })
    test: Dataset({
        features: ['question', 'answer', 'input_ids', 'attention_mask'],
        num_rows: 283
    })
})
```

```
tokenized_datasets["train"]
```

```
Dataset({
    features: ['question', 'answer', 'input_ids', 'attention_mask'],
    num_rows: 1128
})
```

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    "test_trainer",
    evaluation_strategy="steps",
    num_train_epochs=2,
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    learning_rate= 5e-05,
    warmup_steps=500,
    weight_decay=0.01,
    logging_dir='./logs',
    load_best_model_at_end=True,
    logging_steps=400,
    save_steps=400,
    gradient_accumulation_steps=2,
    fp16=True,
)
```

```
import numpy as np

import evaluate

metric = evaluate.load("accuracy")
```

```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)


from transformers import Trainer, DataCollatorForLanguageModeling
data_collator = DataCollatorForLanguageModeling(tokenizer, mlm=False)


trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets["train"],
    eval_dataset=tokenized_datasets["test"],
    compute_metrics=compute_metrics,
    data_collator=data_collator,
)


trainer.train()


device = torch.device("cuda")
model.cuda()


model.to(device)


for step in range(5):
    # encode the new user input, add the eos_token and return a tensor in Pytorch
    new_user_input_ids = tokenizer.encode(input(">> User:") + tokenizer.eos_token, return_t

    # append the new user input tokens to the chat history
    bot_input_ids = torch.cat([chat_history_ids, new_user_input_ids], dim=-1) if step > 0 e

    # generated a response while limiting the total chat history to 1000 tokens,
    chat_history_ids = model.generate(bot_input_ids, max_length=1000, pad_token_id=tokenize

    # pretty print last ouput tokens from bot
    print("DialoGPT: {}".format(tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1
```

It was fun while it lasted...

Colab paid products - Cancel contracts here