04.12.23

OPENCAMPUS.sh

Deep Dive into LLMs

FINE-TUNING, PART I

- Quiz
- Data Preparation
- Supervised Fine-Tuning Examples
- Overview on Reinforcement Learning Methods
- Milestone Review: Baseline Model
- Next Milestone: Model Evaluation

QUIZ

https://forms.office.com/e/xB9nFPU7kx

DATA PREPARATION USING THE DATASETS LIBRARY

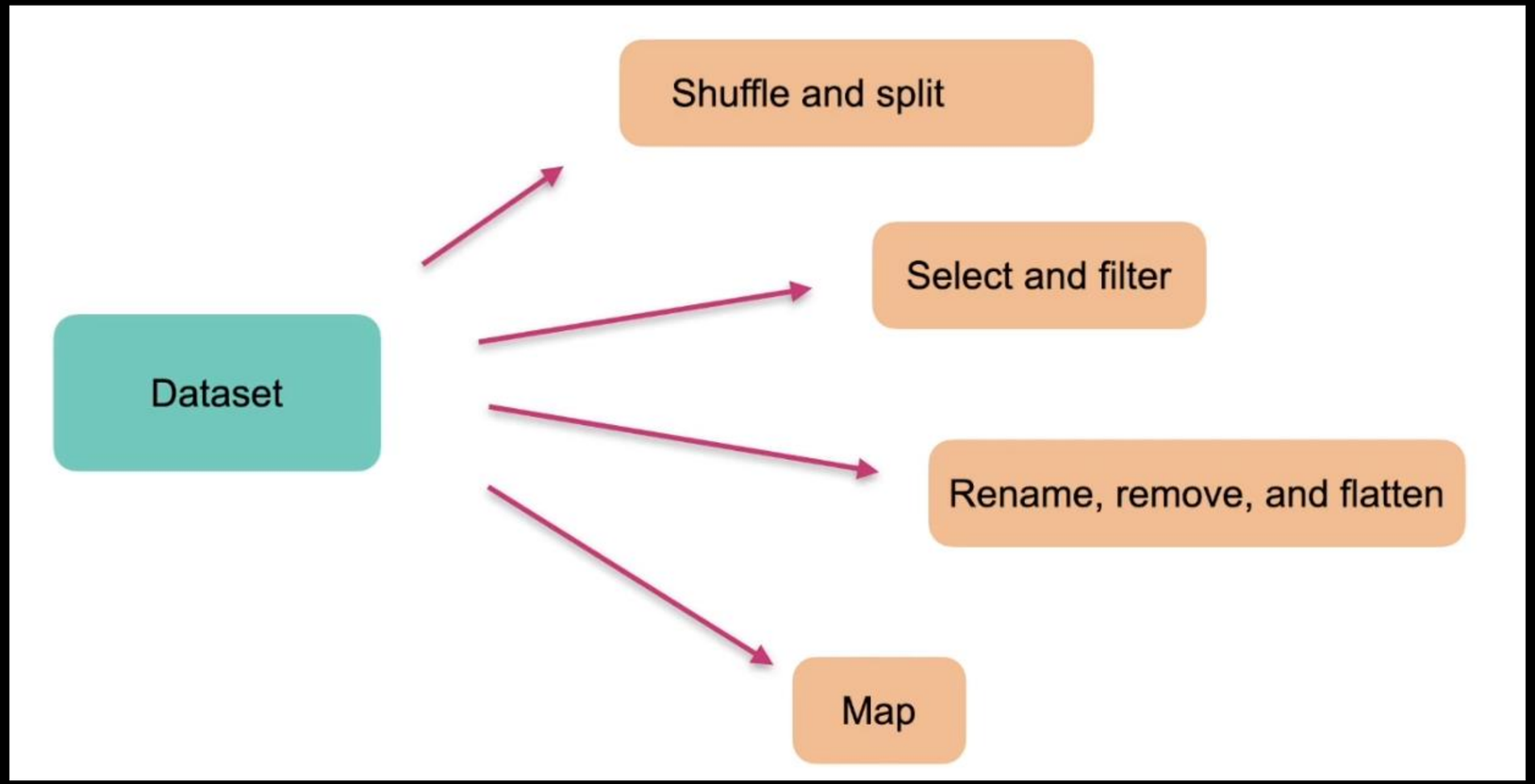
IMPORTING DATA

- **From CSV**
  ```
  load_dataset("csv", data_files="my_file.csv")
  ```

- **From JSON**
  ```
  load_dataset("json", data_files="my_file.jsonl")
  ```

- **From Pandas (Pickle)**
  ```
  load_dataset("pandas", data_files="my_dataframe.pkl")
  Dataset.from_pandas(my_dataframe)
  ```

DATASET METHODS



APACHE ARROWS

- backbone of the Huggingface Datasets library
- an efficient form of Pandas
- optimized for modern CPU and GPU hardware

SAVING MODELS

GitHub, GitLab, Bitbucket, or a similar service using

- git and git LFS (Large File Storage)

Hugging Face Hub using

- huggingface_hub library (based on git and git LFS)
- push_to_hub API

PUSH TO HUB API

```
# authentication
from huggingface_hub import notebook_login
notebook_login()

# saving via callback method
from transformers import PushToHubCallback
callback = PushToHubCallback(
    "bert-finetuned-mrpc", save_strategy="epoch", tokenizer=tokenizer
)
model.fit(train_dataset, epochs=2, callbacks=callback)

# saving manually
model.push_to_hub("bert-finetuned-mrpc, commit="End of training")
```

PREPARATION OF CHAT DATA

- Similar to tokenizers different models expect different input formats
- Correspondingly Hugging Face Tokenizers was extended by chat templates
- A chat template is a dictionary with two keys:
  - “role”: either “system”, “user”, or “assistant”
  - “content”: the string including the content of the message

EXAMPLE WITHOUT GENERATION PROMPT

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("HuggingFaceH4/zephyr-7b-beta")

messages = [
    {"role": "user", "content": "Hi there!"},
    {"role": "assistant", "content": "Nice to meet you!"},
    {"role": "user", "content": "Can I ask a question?"},
]

tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=False)
"""<|im_start|>user
Hi there!<|im_end|>
<|im_start|>assistant
Nice to meet you!<|im_end|>
<|im_start|>user
Can I ask a question?<|im_end|>
"""
```

EXAMPLE WITH GENERATION PROMPT

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("HuggingFaceH4/zephyr-7b-beta")

messages = [
    {"role": "user", "content": "Hi there!"},
    {"role": "assistant", "content": "Nice to meet you!"},
    {"role": "user", "content": "Can I ask a question?"},
]

tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
"""<|im_start|>user
Hi there!<|im_end|>
<|im_start|>assistant
Nice to meet you!<|im_end|>
<|im_start|>user
Can I ask a question?<|im_end|>
<|im_start|>assistant
"""
```

DATA DISTILLATION

- Subsampling according to predefined classes
- Removal of duplicates/ similar training data
- …

MODEL DISTILLATION

- Knowledge transfer from a large, complex model ("teacher" model) to a smaller, simpler model ("student" model)

- The student model is trained to imitate the teacher model, e.g., by using a distillation loss function, combining the standard training loss (like cross-entropy with the true labels) with a term that measures the similarity between the outputs of the student and teacher models.

IMPLEMENTATION OF A CUSTOM LOSS FUNCTION

- Optimizing the training to focus on specific content

- For example:
  - Number of token occurrences
  - Diversity of tokens
  - Certain patterns

SUPERVISED FINE-TUNING

DISCUSSION QUESTION

- When using the Hugging Face Transformers library, the model head for a classification task outputs the raw logits.

  That is, you have to manually apply the softmax to get, e.g., the probabilities for the predicted tokens.
  Why do you think this is?

TRAINING A CAUSAL LANGUAGE MODEL FROM SCRATCH

- Any plain text dataset with a tokenizer can be used for training
- Common metrics are:
  - Perplexity
  - Cross Entropy

TYPICAL PREPROCESSING

- **Concatenate all examples and then split them into chunks of equal size (see “Fine-Tuning a Masked language Model”).**
- **Remove the last chunk if it is shorter than max length to avoid padding issues**
- **After chunking** `DataCollatorForLanguageModeling()` **can be used with argument** `mlm=False`

CONTEXT WINDOW DEPENDENCE

Large context windows:

- **putting all sequences together in one line, including a special EOS**

Short context windows:

- **splitting all sequences and removing the last that is shorter**

TRAINING THE MODEL

```
num_train_steps = len(tf_train_dataset)
optimizer, schedule = create_optimizer(
    init_lr=5e-5,
    num_warmup_steps=1_000,
    num_train_steps=num_train_steps,
    weight_decay_rate=0.01,
)
```

FINE-TUNING A MASKED LANGUAGE MODEL

- Main purpose: Domain adaption
- Unsupervised Training on a large corpus from the more special domain
- Typically, training for 1 epoch

PREPROCESSING

- **A common step is to concatenate all examples (using a special EOS token between the actual samples) and then split the whole corpus into chunks of equal size.**
- **The size of the chunks depend on the amount of GPU memory available and on the model's maximum context size.**
  (As, e.g., given by `tokenizer.model_max_length()`**).**
- **After chunking** `DataCollatorForLanguageModeling()` **can be used to create randomly selected** `[MASKED]` **tokens**
- **For, e.g., whole word masking you have to implement your own data collator.**

TRAINING THE MODEL

```
num_train_steps = len(tf_train_dataset)

optimizer, schedule = create_optimizer(
    init_lr=2e-5,
    num_warmup_steps=1_000,
    num_train_steps=num_train_steps,
    weight_decay_rate=0.01,
)
```

TRANSLATION

A sequence-to-sequence task, similar to:

- Summarization
- Style transfer (e.g., formal to casual or Shakespearean English to modern English)
- Generative question answering (Generates answers to questions, given a context)

Used metric is typically the BLUE score (or sacreBLUE)

PREPROCESSING

- **Tokenization of input *and* targets using a Python context manager:**

```
en_sentence = split_datasets["train"][1]["translation"]["en"]
fr_sentence = split_datasets["train"][1]["translation"]["fr"]

inputs = tokenizer(en_sentence)
with tokenizer.as_target_tokenizer():
    targets = tokenizer(fr_sentence)
```

- **Using** `DataCollatorForSeq2Seq()`:
  - **Padding the encoder input sequence as usual**
  - **Padding the labels with “-100”**
  - **Setting the decoder input sequence equal to the labels but shifted one position to the left**

TRAINING THE MODEL

```
num_epochs = 3
num_train_steps = len(tf_train_dataset) * num_epochs

optimizer, schedule = create_optimizer(
    init_lr=5e-5,
    num_warmup_steps=0,
    num_train_steps=num_train_steps,
    weight_decay_rate=0.01,
)
```

SUMMARIZATION

- Preprocessing is equal to translation
- Seems to need fine-tuning for quite a few epochs (8 in the example)
- Used metric is typically the ROUGE score

TRAINING THE MODEL

```
num_train_epochs = 8
num_train_steps = len(tf_train_dataset) * num_train_epochs

optimizer, schedule = create_optimizer(
    init_lr=5.6e-5,
    num_warmup_steps=0,
    num_train_steps=num_train_steps,
    weight_decay_rate=0.01,
)
```

QUESTION ANSWERING

- Encoder models are good for extractive question answering

- Alternative is generative question answering using:
  - a decoder model or
  - an encoder-decoder model (more similar to text summarization)

- Common metrics are:
  - Exact Match
  - F1-Score

PREPROCESSING

- For training only one correct answer is defined.
  (For evaluation multiple corrects answers are provided.)
- Long contexts are typically split with about 50% overlap,
  the questions remain equal for all splits.
- Labels for the splits have to be provided via a custom function; thereafter the `DefaultDataCollator()` is used.

TRAINING THE MODEL

```
num_train_epochs = 3
num_train_steps = len(tf_train_dataset) * num_train_epochs

optimizer, schedule = create_optimizer(
    init_lr=2e-5,
    num_warmup_steps=0,
    num_train_steps=num_train_steps,
    weight_decay_rate=0.01,
)
```

REINFORCEMENT LEARNING METHODS

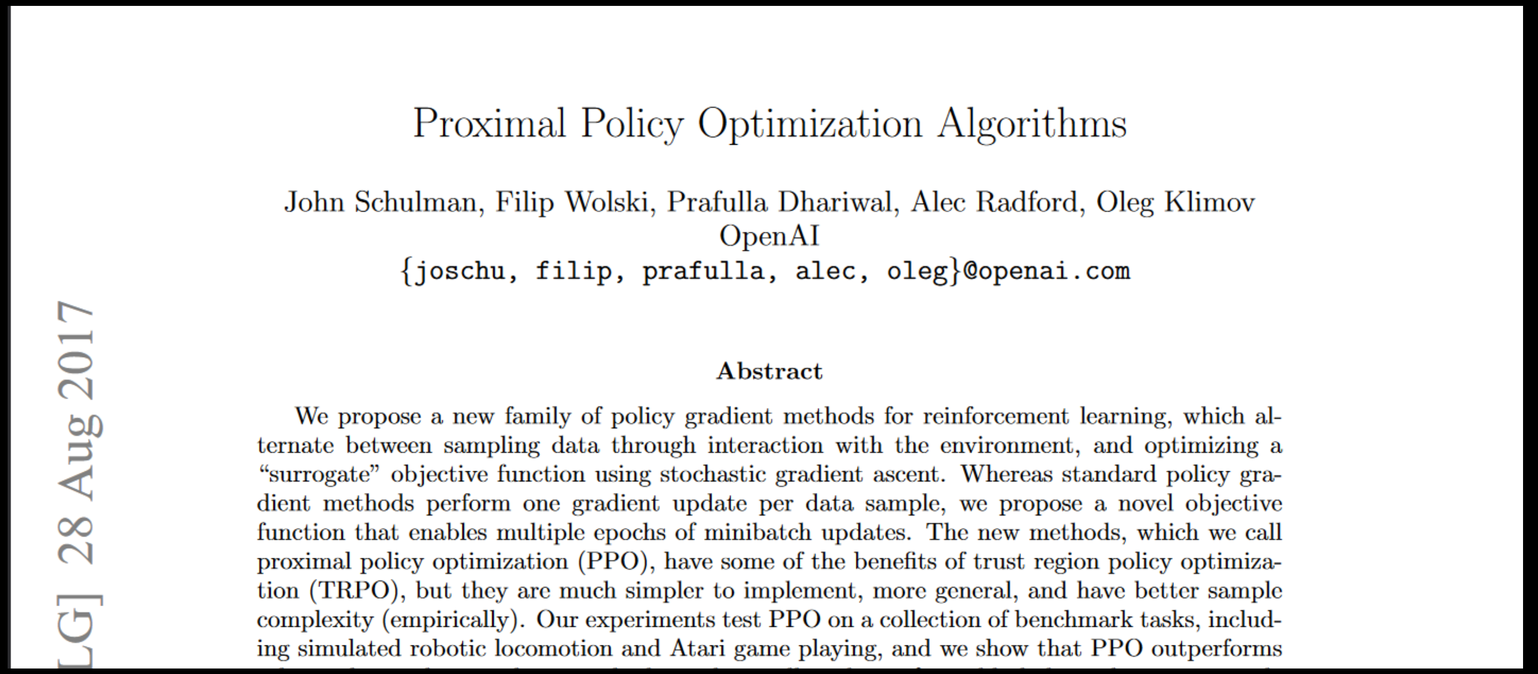
LIBRARY TRL (TRANSFORMERS REINFORCEMENT LEARNING)

Comes with helpers for the following parts typically needed in RLHF:

1. **a supervised fine-tuning (SFT) step**
2. **the process of annotating data with preference labels**
3. **training a reward model on the preference data**
4. **and the RL optimization step**

Rasul, K., Belkada, Y., & von Werra, L. (2023, August 8). Fine-tune Llama 2 with DPO [Blog]. Retrieved November 13, 2023, from Hugging Face website: https://huggingface.co/blog/dpo-trl

PPO (RLHF/RLAIF)



Proximal Policy Optimization Algorithms

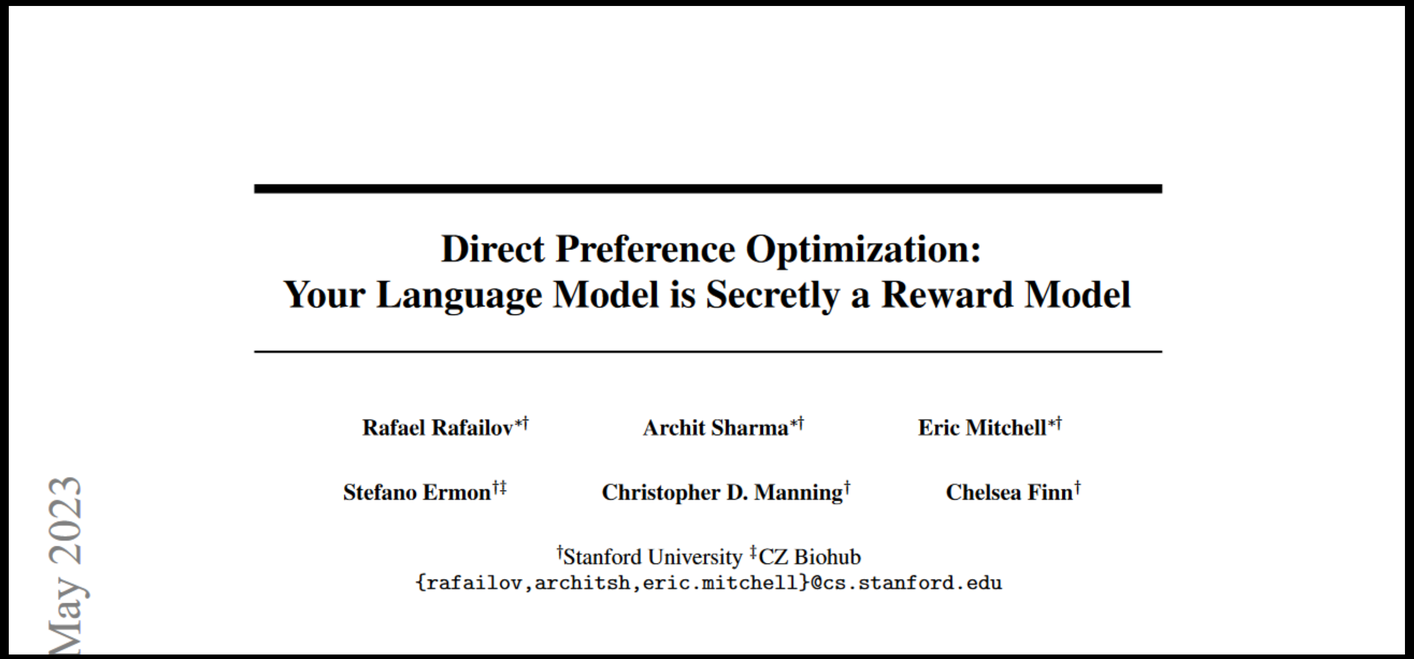
John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

Abstract

We propose a new family of policy gradient methods for reinforcement learning, which alternate between sampling data through interaction with the environment, and optimizing a "surrogate" objective function using stochastic gradient ascent. Whereas standard policy gradient methods perform one gradient update per data sample, we propose a novel objective function that enables multiple epochs of minibatch updates. The new methods, which we call proximal policy optimization (PPO), have some of the benefits of trust region policy optimization (TRPO), but they are much simpler to implement, more general, and have better sample complexity (empirically). Our experiments test PPO on a collection of benchmark tasks, including simulated robotic locomotion and Atari game playing, and we show that PPO outperforms

- Most popular optimization approach in RL
- Very compatible with regular machine learning optimizers
- Based on two simultaneously trained neural nets:
  - one for the policy (which decides which token is predicted next)
  - one for the value function (predicting the expected return of a state)
- LLMs typically need an additional neural network for the reward model
- Still a complex training procedure

DPO



Direct Preference Optimization:
Your Language Model is Secretly a Reward Model

Rafael Rafailov*† Archit Sharma*† Eric Mitchell*†

Stefano Ermon†‡ Christopher D. Manning† Chelsea Finn†

†Stanford University ‡CZ Biohub
{rafailov,architsh,eric.mitchell}@cs.stanford.edu

May 2023

- PPO is focused on the stable and efficient learning of policies in an RL framework
- DPO is directly optimizing models based on human preferences, without the explicit need for learning a reward model.
- This makes DPO particularly suitable for tasks where human preference alignment is crucial.

MILESTONE REVIEW: BASELINE MODEL

NEXT MILESTONE

PROJECT MILESTONES

30.10.	Form Groups
06.11.	Literature Review I
13.11.	Literature Review II
20.11.	Dataset Characteristics I
27.11.	Dataset Characteristics II
04.12.	Baseline Model
11.12.	Model Evaluation
18.12.	Model Definition
08.01.	Project Presentations

MODEL EVALUATION

Clearly specify which metrics you'll use to evaluate the model performance, and why you've chosen these metrics.

TASKS UNTIL NEXT WEEK

- Work through week 2 and week 3 of the course Generative AI with Large Language Models.
- Specify which metrics you'll use to evaluate the model performance, and why you've chosen these metrics. Document it in the corresponding section of your project repository.