Fine-tuning Large Language Models

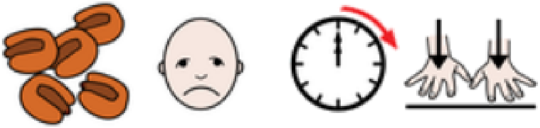
Pablo Ariel Duboue, PhD

PyData Vancouver Meetup, 2023-06-08

Introduction

This talk is being held on the unceded territories of the xʷməθkʷəy̓əm (Musqueam), Sḵwx̱wú7mesh (Squamish), and Sel’íl’witulh (Tsleil-Waututh) Nations. The speaker thanks them for having cared for the land and seeks to work with them in partnerships that might arise.

Teaser



“popcorn/NOUN”, “sad/ADJECTIVE”, “future/VERB”, “to be/VERB”

- An Open Source LLM with 3B paramters (5.3Gb on disk) returns:
  Here is an example of how to simulate an AAC communicator using the AACSim library: python import aactl; import aactl.simulation ...
- Same model, fine-tuned (Δweights 2.6Mb on disk):
  The popcorn will be sad
- Same model, different training data (same size on disk):
  I'll be sad when the popcorn's gone

This Talk

- Transformers
- LLMs
- The task and data
- **Fine-tuning Details**
  - Installation
  - Fine-tuning Run
  - Post-run

Slides at
`https://textualization.com/talks/20230608pydata_finetuning_llms.pdf`

My Background

- Been around doing research in NLP/ML for the last 25 years
  I've seen things you people wouldn't believe... Attack ships on fire off the shoulder of Orion...
- Corporate research scientist for 6 years
  - Helped build the IBM Jeopardy! Watson system
- About 50 peer-reviewed papers and patents
- Have a one person company (textualization.com) in town
  - Consulting mostly for startups
- Wrote a book on Feature Engineering: https://artoffeatureengineering.com/
  - published in 2019 by Cambdrige University press

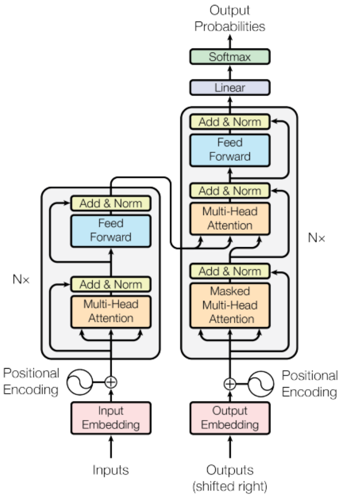
Part I

Transformers

Transformers

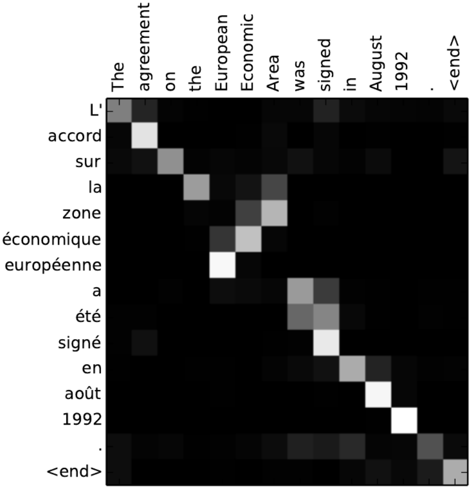
1. Transformer Architecture
2. Sequence-to-Sequence Origin
3. CausalLM vs MaskedLM
4. Q, K, V Matrices
5. CausalLM vs MaskedLM
6. Self-Attention vs. Cross-Attention
7. LoRA

Transformer Architecture



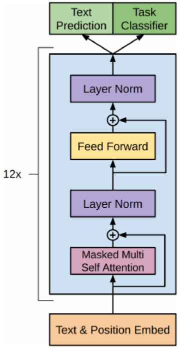
Vaswani et al. (2017) - fig. 1

Sequence-to-Sequence Origin

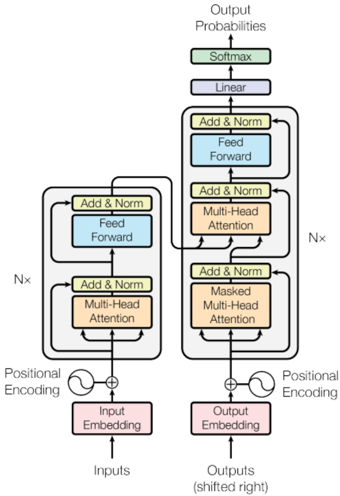


(Bahdanau et al., 2015, fig. 3a)

CausalLM vs MaskedLM



Original GPT



MaskedLM

Q, K, V Matrices

- "Scaled-dot product attention"
- Input:
  - queries and keys, of dimensionality d_k
  - values, of dimensionality d_v
- If multiple queries are packed into a matrix Q, and the key-values are packed into matrices K and V then:
  - $\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}}) * V$

Self-Attention vs. Cross-Attention

- Cross-attention:
  - The decoder attends to the output of the encoder
- In transformers, the encoder and the decoder use only the attention mechanism
  - The decoder can do cross-attention as before
    - Queries: previous decoder layer
    - Keys and values: output of the decoder
  - The encoder, however, cannot do cross-attention and it attends to itself
    - Self-attention
    - Queries, keys and values all come the previous layer of the decoder
    - The output for a token can incorporate information from any other token

Note

Most LLMs are GPT-based and use only self-attention

LoRA

- Upon training a transformers, the matrices Q, K, V tend to be full of zeros
- It is therefore possible to expresses them as the multiplication of two smaller matrices
- This is called "low-rank" approximation and described in the paper *https://arxiv.org/abs/2106.09685*

$$\vec{v} \otimes \vec{w} = \begin{bmatrix}1\\2\\3\end{bmatrix}\begin{bmatrix}4 & 5\end{bmatrix} = \begin{bmatrix}1\cdot4 & 1\cdot5\\ 2\cdot4 & 2\cdot5\\ 3\cdot4 & 3\cdot5\end{bmatrix} = \begin{bmatrix}4&5\\8&10\\12&15\end{bmatrix}$$

From *The Tensor Product, Demystified*

Part II

LLMs

LLMs

1. LLMs vs LMs
2. Emergence
3. Training from Scratch
4. The Pile
5. INCITE RedPajama Models
6. Other Open Source Models
7. Fine-Tuning
8. Prompt Engineering
9. Prompt-Tuning

LLMs vs LMs

- A language model (LM) tells you the probability of new words given already seen words.
  - The most famous example is the cellphone autocorrect functionality
    - "How does ChatGPT knows XYZ?"... the same way your phone knows that after your first name most probably your last name follows
    - With better models and much more training data, they work much better
- Something happens when the language model gets **really big**

Emergence

- *More Is Different*, by P.W. Anderson, Science (1972)
  - A *large* language model all of a sudden is **more** than just a *language model*
- Emergent behaviour
- Reasoning capabilities
- *Are they a mirage?*
  - Are Emergent Abilities of Large Language Models a Mirage? by Schaeffer, Miranda, Koyejo at arXiv (2023)

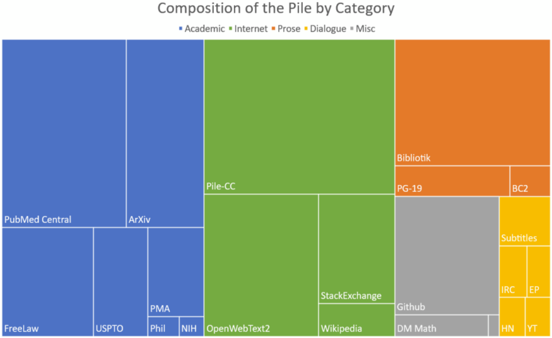
Training from Scratch

- Training from scratch (aka building *Foundational Models*) requires resources available to few organizations
- This is why fine-tuning foundational models puts LLMs models in the hands of small organizations and individuals

The Pile

- Many Open Source LLMs are trained on The Pile gathered by Eleuther.AI:
- From `https://pile.eleuther.ai/` and their arXiv paper

  > *The Pile is a 825 GiB diverse, open source language modelling data set that consists of 22 smaller, high-quality datasets combined together.*



INCITE RedPajama Models

> *The training of the first collection of RedPajama-INCITE models is performed on **3,072 V100 GPUs** provided as part of the INCITE compute grant on Summit supercomputer at the Oak Ridge Leadership Computing Facility (OLCF). This grant was awarded to AAI CERC lab at Université de Montréal, LAION and EleutherAI in fall 2022 for their collaborative project on Scalable Foundation Models for Transferrable Generalist AI.*

- 800B Tokens
- RedPajama is a clean-room, fully open-source implementation of the LLaMa dataset
- Apache v2 Licensed

Other Open Source Models

- EuletherAI GPT NeoX (48G GPU)
- OpenChatKit (instructional training)
    - OIG by LAION of Stable Diffusion fame
- EuletherAI Pythia models
- BLOOM models
    - Not open source (use restrictions): `https://huggingface.co/spaces/bigscience/license`
- Dolly models, by databricks
- Falcon LLM

Fine-Tuning

- This talk focuses on continuing to train the model with additional data
- This training is done for a small amount of time, with a small learning rate to avoid overtaking the existing parameters
  - Catastrophic forgetting
- Moreover, the training is done over Δweights in the form of low-rank matrices

Prompt Engineering

- An alternative way to access the latent knowledge in the weights is to make the initial context as informative as possible
  - This might include a few examples from training data
    - Called *exemplars*
- Using large, complex prompts is the solution of choice when using large commercial models through APIs
  - Due to OpenAI decision to stop LoRA fine-tuning their models after GPT3

Please Note

Open Source models have a smaller context, which limits their potential for prompt engineering

Prompt-Tuning

- Aside from tuning the weights of the model, it is possible to tune the binary numbers representing the input context
  - While we think of the input as a sequence of tokens, the model see tensors
  - These input tensors can be changed (no longer *meaning* vocabulary tokens)
- This process has not led to great gains and it is harder to understand that the other two

Part III

Task and Data

Task: AAC

1. What is AAC
2. 24 Pull Requests
3. Meet an AAC communicator
4. SimpleNLG
5. Data by Sampling
6. Fixing data by hand
7. Writers not Annotators
8. Training Prompt
9. Final instructions

What is AAC

From ChatGPT:

> *An AAC (Augmentative and Alternative Communication) communicator is a tool or strategy designed to help individuals with speech or language impairments to communicate. It can use symbols or pictures to facilitate communication.*

24 Pull Requests

- 24 Pull Requests is a challenge to make 24 Free Software contributions on GitHub in the 24 days before Christmas
- Participating helps give back to the community, as professional software development increasingly relies on Open Source software
- The challenge allows exploration of different languages, platforms, or projects, and encourages learning curiosity
- Contributing to Open Source projects with little attention can promote further community service through Free Software
- Planning daily Pull Requests and finding projects to contribute to is enjoyable for those who love exploration

Meet an AAC Communicator

- https://github.com/vidma/aac-speech-android
  - by Vidmantas Zemleris at EPFL
  - Bilingual French/English
  - SimpleNLG (discussed next) French port by Universite de Montreal
- Demo

SimpleNLG

- In a rule-based NLG pipeline, a surface realiser containts the linguistic rules of grammar (about morphology and syntax) to convert abstract representations of sentences into actual text

```
PhraseSpec p = nlg.createClause();
NPPhraseSpec subject1 = nlg.createNounPhrase("Mary");
NPPhraseSpec subject2 = nlg.createNounPhrase("your", "
    giraffe");
CoordinatedPhraseElement subj = nlg.createdCoordinatedPhrase
    (subj1,subj2);
subj.setFeature(Feature.CONJUNCTION, "or");
p.setSubject(subj);
p.setVerb("chase");
p.setObject("the␣monkey");
```

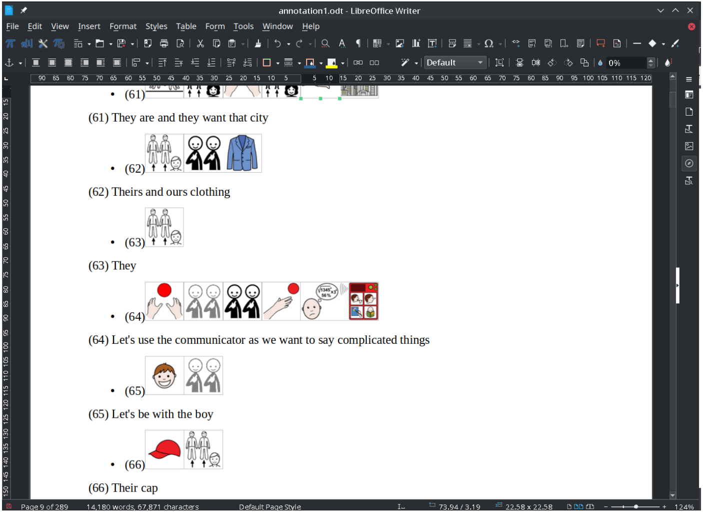
Output

Mary or your giraffe chase the monkey.

Data by Sampling

- While the Open Source AAC Communicator provides some quick sketch of output, for training data we need also **real input data**
- Acquiring that data is not easy without access to AAC users
  - Also, as with any language, most of the things people say are the same
    - "need food", "need to use the toilet", etc
- Instead, a hierarchical sampler was programmed as follows:
  - A poisson distribution with mean=3 was used for the length
  - The position of the buttons in the Open Source AAC was used to determine their likelihood
    - Buttons in first page doubly likely
  - Verbs were also more likely after a word was entered until a first verb was entered

Fixing the Data



Writers not Annotators

- Generative AI is different from traditional machine learning
- The "data" we are using is written text
  - Quality text produces better results
- While in ML we could use crowdworkers and put up with low quality annotations
  - Size trumps quality
- in Generative AI we are need **writers** rather than annotators
  - Quality if paramount

Takeaway

Be prepared to spend much more time and money in data acquisition

Training Prompt

- Prompt:

```
<human>: Simulate an AAC communicator given the following
    icon input:

* { I ; NOUN ; je.png }
* { be ; VERB ; etre.png }
* { behind ; ADJECTIVE ; 5443_1.png.64x64_q85.png }

<bot>: I am behind
```

- The use of “<human>:” and “<bot>:” to distinguish the turns is an OpenChatKit convention.
  - The chat-tuned RedPajama expects it

Final Instructions


```
{"text":"<human>:␣Simulate␣an␣AAC␣communicator␣given␣the␣
    following␣icon␣input:␣\n*␣{␣I␣;␣NOUN␣;␣je.png␣}␣␣\n*␣{␣
    be␣;␣VERB␣;␣etre.png␣}␣␣\n*␣{␣behind␣;␣ADJECTIVE␣;␣5443
    _1.png.64x64_q85.png␣}␣\n<bot>:␣I␣am␣behind\n"}
{"text":"<human>:␣Simulate␣an␣AAC␣communicator␣given␣the␣
    following␣icon␣input:␣\n*␣{␣staff␣;␣NOUN␣;␣7116_1.png.64
    x64_q85.png␣}␣␣\n*␣{␣that␣;␣NOUN␣;␣that_one.png␣}␣\n<bot
    >:␣That␣staff\n"}
{"text":"<human>:␣Simulate␣an␣AAC␣communicator␣given␣the␣
    following␣icon␣input:␣\n*␣{␣you␣;␣NOUN␣;␣tu.png␣}␣\n*␣{␣
    hog␣;␣NOUN␣;␣2327_2.png.64x64_q85.png␣}␣\n<bot>:␣You␣are
    ␣a␣pig\n"}
{"text":"<human>:␣Simulate␣an␣AAC␣communicator␣given␣the␣
    following␣icon␣input:␣\n*␣{␣so␣do␣i␣;␣NOUN␣;␣11591_1.png
    .64x64_q85.png␣}␣\n<bot>:␣So␣do␣I\n"}
```


- This data is in JSONL format, each line is a valid JSON document

Part IV

Fine-tuning Details

1. Installation
2. Fine-tuning run
3. Post-run

Installation

1. Hardware Requirements
2. LoRA Branch
3. OpenChatKit Directory Structure
4. Python Version
5. pip install vs. conda
6. Current `requirements.txt`
7. Preparing the Data
8. WandB woes
9. Bonus: host memory off-load

Hardware Requirements

- For these experiments:
  - No GPU (otherwise at least 24G GPU, most likelye 48G of VRAM)
  - 50G of disk (model itself is 11G, venv is 5G)
  - 24G of CPU RAM
- If you have a small GPU that gets in the way set:
  - export CUDA_VISIBLE_DEVICES=''
- Or in the python code:

```
import os
os.environ["CUDA_VISIBLE_DEVICES"]=""
```

LoRA Branch

- `git clone https://github.com/togethercomputer/OpenChatKit`
- `cd OpenChatKit`
- `checkout low-rank`

OpenChatKit Directory Structure

```
OpenChatKit
├── data
│   ├── OIG
│   ├── OIG-chip2
│   ├── OIG-moderation
│   └── wikipedia-3sentence-level-retrieval-index
├── docs
├── inference
├── outputs
│   ├── redpajama-incite-chat-3b-aac-lowrank
│   └── redpajama-incite-chat-3b-aac-toddler-lowrank
├── pretrained
│   ├── GPT-NeoX-20B
│   ├── Pythia-6.9B-deduped
│   └── RedPajama-3B
│       └── togethercomputer_RedPajama-INCITE-Chat-3B-v1
├── retrieval
├── tools
└── training
    ├── comm
    ├── data_parallel
    ├── lora
    ├── modules
    ├── optimizer
    ├── pipeline_parallel
    ├── tasks
    │   └── data_loaders
    └── utils

28 directories
```

Python Version

- These experiments use python 3.10.10
- The packages being used are rather old
- Newer versions of the interpreter might not work

pip install vs. conda

- Current installation instructions for OpenChatKit use conda
  - And an add-on for fast package installation, mamba
- Using conda requires about 50G of additional disk space
- I prefer the pip install route but conda *might* have more optimized binaries

Current requirements.txt

```
pip install torch==1.13.1 faiss-gpu==1.7.2 pyarrow==8.0.0
pip install accelerate==0.17.1
pip install transformers
pip install bitsandbytes
pip install scipy
pip install datasets
pip install peft
```

- Final `requirements.txt` is 46 lines in total

Preparing the Data

- The scripts with instructions in jsonl format and a "text" entry in the json object can be loaded from any place in the file system
- For ease of organization, the OpenChatKit directory structure assumes they will be in the data/ folder, in a subfolder with the name of the source
- For these experiments, I put the jsonl directly in data
- Aside from the training data, the parameter models have to be put in torch format:

```
python pretrained/RedPajama-3B/prepare.py
```

- It creates the folder `pretrained/RedPajama-3B/togethercomputer_RedPajama-INCITE-C` (5.3G on disk)

WandB woes

- Sometimes there is a paid service "Weights-and-Biases" that is activated by default
- It can be deactivated by
  `export WANDB_DISABLED=true`

Bonus: host memory off-load

- Next version of the Transformers library supports using CPU RAM if the GPU RAM is not enough
- Need to install it from HEAD
  - `pip install git+https://github.com/huggingface/transformers.git`
- The parameter "`use_offload`" becomes available on model construction

Fine-tuning run

1. Run Parameters
2. Fine-tuning Script (1)
3. Fine-tuning Script (2)
4. Fine-tuning Script (3)
5. Fine-tuning Script (4)
6. Fine-tuning Script (5)
7. Tracking the Run

Run Parameters

- Rank was set very low, to 2
  - Default was 16
  - Rationale: this task is very close to the base LLM task
- Learning Rate
  - These experiments use the default learning rate in the script 2e-4
  - The regular fine-tuning for all weights use a smaller 1e-5
- The training was run for 200 steps with a batch size of 4
  - Total of 6 epochs according to the logs
- These parameters risk catastrophic forgetting
  - Reasonable choice as we don't expect to use the model for anything else

Fine-tuning Script (1)

```python
import os
import json
os.environ["CUDA_VISIBLE_DEVICES"]=""
import torch
import transformers
import torch.nn as nn
import bitsandbytes as bnb
from datasets import Dataset
from peft import LoraConfig, get_peft_model
from transformers import AutoTokenizer, AutoConfig,
    AutoModelForCausalLM

# this script should take around 14GB VRAM

MODEL_NAME='redpajama-incite-chat-3b-aac-lowrank'

# read datasets
with open('data/aac-504.jsonl', 'r') as fp:
    data = [json.loads(x) for x in fp.readlines()]
```

Fine-tuning Script (2)

```python
model = AutoModelForCausalLM.from_pretrained(
    "togethercomputer/RedPajama-INCITE-Chat-3B-v1",
    device_map='sequential',
)
tokenizer = AutoTokenizer.from_pretrained("togethercomputer/RedPajama-INCITE-Chat-3B-v1")
tokenizer.pad_token = tokenizer.eos_token

for param in model.parameters():
  param.requires_grad = False  # freeze the model - train adapters later
  if param.ndim == 1:
    # cast the small parameters (e.g. layernorm) to fp32 for stability
    param.data = param.data.to(torch.float32)

model.gradient_checkpointing_enable()  # reduce number of stored activations
model.enable_input_require_grads()
```

Fine-tuning Script (3)

```
config = LoraConfig(
    r=2,
    lora_alpha=32,
    target_modules=["query_key_value", "xxx"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)

model = get_peft_model(model, config)
print_trainable_parameters(model) # skipped in this
    presentation

## Training

data = Dataset.from_list(data)
data = data.map(lambda samples: tokenizer(samples['text'
    ]), batched=True)
```

Fine-tuning Script (4)

```
trainer = transformers.Trainer(
    model=model,
    train_dataset=data,
    args=transformers.TrainingArguments(
        per_device_train_batch_size=4,
        gradient_accumulation_steps=4,
        warmup_steps=100,
        max_steps=200,
        learning_rate=2e-4,
        fp16=False,
        logging_steps=1,
        output_dir='outputs',
    ),
    data_collator=transformers.
        DataCollatorForLanguageModeling(tokenizer, mlm=
        False)
)
```

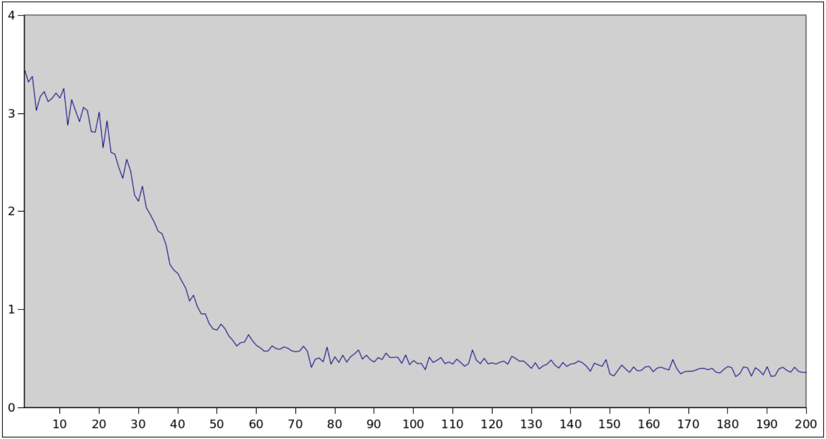
Fine-tuning Script (5)

```
model.config.use_cache = False  # silence the warnings. Please re-enable for inference!
trainer.train()

# save the trained adapter to disk
model.save_pretrained(f"outputs/{MODEL_NAME}")
```

Tracking the Run

- Run used about 18.3g of RAM and ran at 1002% CPU utilization (10 cores, it had more available)
- It ran between 3 to 5 hours, depending on load in the host



Post-run

1. Inference Script (1)
2. Inference Script (2)
3. Inference Script (3)
4. Inference Script (4)
5. Inference Script (5)
6. Merging LoRA back
7. Bonus: redpajama.cpp
8. Bonus: style adaptation using OpenAI
9. Bonus: tortoise-tts

Inference Script (1)

```
import torch
from peft import PeftModel, PeftConfig
from transformers import AutoModelForCausalLM,
    AutoTokenizer

peft_model_path ='outputs/redpajama-incite-chat-3b-aac-
    lowrank'

config = PeftConfig.from_pretrained(peft_model_path)
model = AutoModelForCausalLM.from_pretrained(config.
    base_model_name_or_path, return_dict=True,
    load_in_8bit=True, device_map='auto')
model.config.use_cache = True
tokenizer = AutoTokenizer.from_pretrained(config.
    base_model_name_or_path)

# Load the Lora model
model = PeftModel.from_pretrained(model, peft_model_path
    )
```

Inference Script (2)

```
batch = tokenizer("<human>:␣Simulate␣an␣AAC␣communicator
    ␣given␣the␣following␣icon␣input:␣\n*␣{␣to␣go␣;␣VERB␣;
    ␣2432_2.png.64x64_q85.png␣}␣␣\n*␣{␣broken␣;␣ADJECTIVE
    ␣;␣4736_1.png.64x64_q85.png␣}␣␣\n*␣{␣air␣hostesses␣;␣
    NOUN␣;␣12062_1.png.64x64_q85.png␣}␣␣\n*␣{␣bird␣;␣NOUN
    ␣;␣2490_2.png.64x64_q85.png␣}␣\n<bot>:",
    return_tensors='pt')

with torch.cuda.amp.autocast():
    output_tokens = model.generate(**batch,
        max_new_tokens=50)

print('\n\n', tokenizer.decode(output_tokens[0],
    skip_special_tokens=False))
```

Merging LoRA back

- Sometimes we want a full model rather than a base model plus LoRA delta weights
  - To use redpajama.cpp, for instance
  - Or to do RLHF
- To merge back the LoRA delta weights into the base model do and save it, do:

```
model = model.merge_and_unload()
model.save_pretrained(full_model_path)
```

- Output folder goes from 2.6Mb for LoRA to 11G for full weights

Bonus: redpajama.cpp

```
git clone https://github.com/togethercomputer/redpajama.cpp
cd redpajama.cpp; mkdir models/pythia
make redpajama-chat quantize-gptneox
python
./examples/redpajama/scripts/convert_gptneox_to_ggml.py
/path/to/full/<your-model> models/pythia
```

- Might need to edit the script and point to the tokenizer directly

```
python ./examples/redpajama/scripts/quantize-gptneox.py
models/pythia/ggml-<your-model>.bin
./redpajama-chat -m models/pythia/ggml-<your-model>-q4_0.bin
```

- Uses 2Gb of RAM and all cores

Bonus: style adaptation

- Style adaptation using OpenAI GPT4, prompt:

```
Rewrite the following outputs for an AAC Communicator to
    make them warmer, more familiar and the type of things a
     4 year old would say:

(397) Does not want to be beautiful
(399) You, and my kitchen utensil
(400) The rod can do it
(401) On Sunday, it has it
(402) He wants it
(403) The cause will be wanted
```

- Example output:

- (397) Doesn't wanna be pretty
- (399) You and my kitchen toy
- (400) The stick can do it!
- (401) On Sunday, we got it!
- (402) He really wants it!
- (403) We'll need it for sure!

Bonus: tortoise-tts

- https://github.com/neonbjb/tortoise-tts
- Record at least five 10 second quality voice clips
- Use a custom voice for the communicator

Closing Remarks

- Issues with Open Souce models
  - Windows size
  - Dealing with numbers
  - Poor programming capabilities
- But they are getting so much better even some Googlers fear them
- Just taught a seminar in Neural Network Architectures
  - Lectures in YouTube: https://www.youtube.com/@pabloduboue
- Connecting Prompt Engineering and old-school NLP: see my webinar at https://www.youtube.com/@DataUmbrella
- Learn DS reading group

Connecting

- Twitter: @pabloduboue
- GitHub: DrDub
- Projects: http://wiki.duboue.net/A_Dollar_Worth_of_Ideas

- https://tellandshow.org/: community-owned machine learning
- https://textualization.com/gptwhitepaper/
- https://artoffeatureengineering.com/