

Generatywna sztuczna inteligencja z dużymi modelami tekstowymi

Parameter-Efficient Fine-Tuning (PEFT)

Michał Żarnecki



Parameter-Efficient Fine-Tuning (PEFT)

PEFT (Parameter-Efficient Fine-Tuning)

PEFT to ogólne podejście mające na celu poprawę efektywności procesu fine-tuningu modeli. Zamiast trenować wszystkie parametry modelu, w PEFT modyfikuje się tylko wybrane podzbiory parametrów, co pozwala na oszczędności w zasobach obliczeniowych i pamięciowych. PEFT znajduje zastosowanie w różnych technikach, takich jak LoRA, Adaptery czy Prefix Tuning.

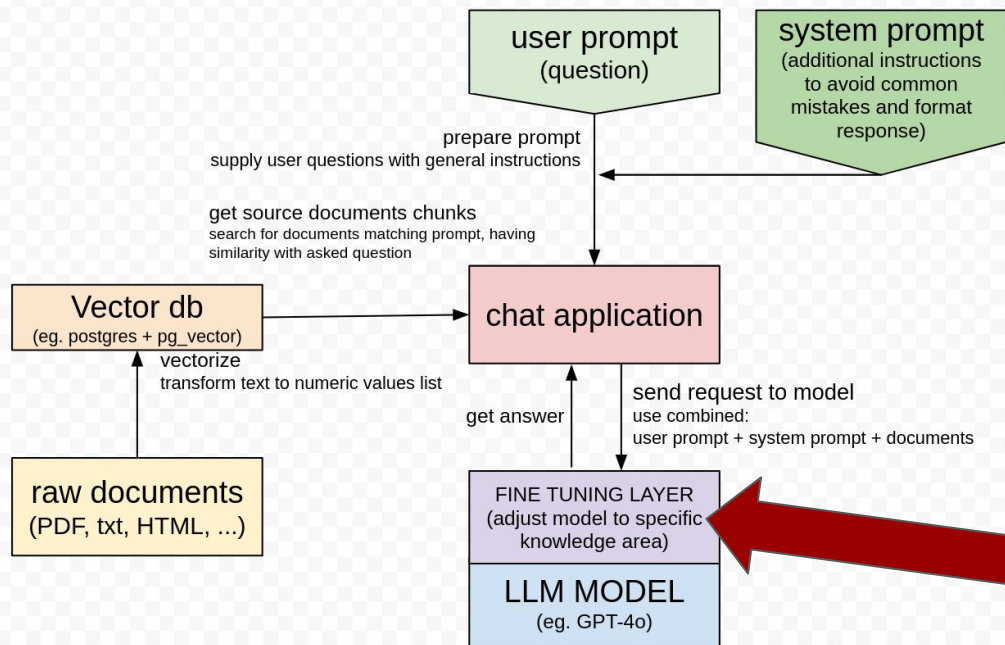
LoRA (Low-Rank Adaptation)

LoRA (Low-Rank Adaptation)

LoRA to jedna z metod PEFT, która wprowadza dodatkowe niskowymiarowe (low-rank) macierze w miejscu pełnych macierzy wag w modelu. Zamiast trenować wszystkie parametry modelu, LoRA optymalizuje tylko te niskowymiarowe macierze, co zmniejsza koszty trenowania i zwiększa efektywność obliczeniową. LoRA pozwala na adaptację dużych modeli do nowych zadań bez potrzeby modyfikowania oryginalnych wag modelu, co jest szczególnie przydatne przy pracy z bardzo dużymi modelami, jak GPT czy BERT.

LLM fine-tuning

AI CHATBOT (LLM + RAG)



Parameter-Efficient Fine-Tuning (PEFT)

<https://www.kaggle.com/code/rzarno/llama3-fine-tuning-1-train>

<https://www.kaggle.com/code/rzarno/llama3-fine-tuning-2-merge-adapter>

<https://www.kaggle.com/code/rzarno/llama3-fine-tuning-3-convert>

<https://www.kaggle.com/code/rzarno/llama3-fine-tuning-4-quantize>

Parameter-Efficient Fine-Tuning (PEFT)

#based on article <https://www.datacamp.com/tutorial/llama3-fine-tuning-locally?c>

```
from transformers import (  
    AutoModelForCausalLM,  
    AutoTokenizer,  
    BitsAndBytesConfig,  
    HfArgumentParser,  
    TrainingArguments,  
    pipeline,  
    logging,  
)  
from peft import (  
    LoraConfig,  
    PeftModel,  
    prepare_model_for_kbit_training,  
    get_peft_model,  
)  
import os, torch, wandb  
from datasets import load_dataset  
from trl import SFTTrainer, setup_chat_format
```

Parameter-Efficient Fine-Tuning (PEFT)

```
from huggingface_hub import login
from kaggle_secrets import UserSecretsClient
user_secrets = UserSecretsClient()

hf_token = user_secrets.get_secret("HuggingFace")

login(token = hf_token)

wb_token = user_secrets.get_secret("WandB")

wandb.login(key=wb_token)
run = wandb.init(
    project='Fine-tune Llama 3 8B on Medical Dataset',
    job_type="training",
    anonymous="allow"
)
```

Parameter-Efficient Fine-Tuning (PEFT)

```
base_model = "/kaggle/input/llama-3/transformers/8b-chat-hf/1"  
dataset_path = "/kaggle/input/industry-examples/industry_examples_v2.csv"  
new_model = "llama-3-8b-chat-industry-fine-tuned-guidelines-examples"
```

```
torch_dtype = torch.float16  
attn_implementation = "eager"
```

```
# QLoRA config  
bnb_config = BitsAndBytesConfig(  
    load_in_4bit=True,  
    bnb_4bit_quant_type="nf4",  
    bnb_4bit_compute_dtype=torch_dtype,  
    bnb_4bit_use_double_quant=True,  
)  
  
# Load model  
model = AutoModelForCausalLM.from_pretrained(  
    base_model,  
    quantization_config=bnb_config,  
    device_map="auto",  
    attn_implementation=attn_implementation  
)
```


Parameter-Efficient Fine-Tuning (PEFT)

```
# Load tokenizer
```

```
tokenizer = AutoTokenizer.from_pretrained(base_model)
model, tokenizer = setup_chat_format(model, tokenizer)
```

```
# LoRA config
```

```
peft_config = LoraConfig(
    r=16,
    lora_alpha=32,
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=['up_proj', 'down_proj', 'gate_proj', 'k_proj', 'q_proj', 'v_proj', 'o_proj']
)
model = get_peft_model(model, peft_config)
```

Parameter-Efficient Fine-Tuning (PEFT)

```
import os.path
os.path.isfile(dataset_path)
```

```
#Importing the dataset
dataset = load_dataset('csv', data_files=dataset_path)
dataset = dataset.shuffle(seed=65)

def format_chat_template(row):
    row_json = [{"role": "user", "content": row["guideline"]},
                {"role": "assistant", "content": row["key"]}
    row["text"] = tokenizer.apply_chat_template(row_json, tokenize=False)
    return row

dataset = dataset.map(
    format_chat_template,
    num_proc=4,
)

dataset['train']['text'][3]
```

Parameter-Efficient Fine-Tuning (PEFT)

```
training_arguments = TrainingArguments(  
    output_dir=new_model,  
    per_device_train_batch_size=1,  
    per_device_eval_batch_size=1,  
    gradient_accumulation_steps=2,  
    optim="paged_adamw_32bit",  
    num_train_epochs=1,  
    evaluation_strategy="steps",  
    eval_steps=0.2,  
    logging_steps=1,  
    warmup_steps=10,  
    logging_strategy="steps",  
    learning_rate=2e-4,  
    fp16=False,  
    bf16=False,  
    group_by_length=True,  
    report_to="wandb"  
)
```

```
trainer = SFTTrainer(  
    model=model,  
    train_dataset=dataset["train"],  
    eval_dataset=dataset["train"],  
    peft_config=peft_config,  
    max_seq_length=512,  
    dataset_text_field="text",  
    tokenizer=tokenizer,  
    args=training_arguments,  
    packing=False,  
)
```

Parameter-Efficient Fine-Tuning (PEFT)

```
trainer.train()
```

```
wandb.finish()  
model.config.use_cache = True
```

```
messages = [  
    {  
        "role": "user",  
        "content": "Welches Wirtschaftszweig passt am besten zu dem unten beschriebenen Unternehmen?: Ticos Syst  
    }  
]  
  
prompt = tokenizer.apply_chat_template(messages, tokenize=False,  
                                       add_generation_prompt=True)  
  
inputs = tokenizer(prompt, return_tensors='pt', padding=True,  
                  truncation=True).to("cuda")  
  
outputs = model.generate(**inputs, max_length=150,  
                        num_return_sequences=1)  
  
text = tokenizer.decode(outputs[0], skip_special_tokens=True)  
  
print(text.split("assistant")[1])
```

Parameter-Efficient Fine-Tuning (PEFT)

```
trainer.model.save_pretrained(new_model)
trainer.model.push_to_hub(new_model, use_temp_dir=False)
```

Parameter-Efficient Fine-Tuning (PEFT) - merge adapter to model

```
from huggingface_hub import login
from kaggle_secrets import UserSecretsClient
user_secrets = UserSecretsClient()

hf_token = user_secrets.get_secret("HuggingFace")
login(token = hf_token)
```

```
base_model = "/kaggle/input/llama-3/transformers/8b-chat-hf/1"
new_model = "/kaggle/input/llama3-fine-tuning-1-train/llama-3-8b-chat-industry-fine-tuned-guidelines-examples/"
```

Parameter-Efficient Fine-Tuning (PEFT) - merge adapter to model

```
from transformers import AutoModelForCausalLM, AutoTokenizer, pipeline
from peft import PeftModel
import torch
from trl import setup_chat_format

# Reload tokenizer and model
tokenizer = AutoTokenizer.from_pretrained(base_model)

base_model_reload = AutoModelForCausalLM.from_pretrained(
    base_model,
    return_dict=True,
    low_cpu_mem_usage=True,
    torch_dtype=torch.float16,
    device_map='auto',
    trust_remote_code=True,
)

base_model_reload, tokenizer = setup_chat_format(base_model_reload, tokenizer)

# Merge adapter with base model
model = PeftModel.from_pretrained(
    base_model_reload,
    new_model,
)

model = model.merge_and_unload()
```

Parameter-Efficient Fine-Tuning (PEFT) - merge adapter to model

```
messages = [{"role": "user", "content": "How much is 2+2"}]

prompt = tokenizer.apply_chat_template(messages, tokenize=False, add_generation_prompt=True)
pipe = pipeline(
    "text-generation",
    model=base_model_reload,
    tokenizer=tokenizer,
    torch_dtype=torch.float16,
    device_map="auto",
)

outputs = pipe(prompt, max_new_tokens=120, do_sample=True, temperature=0.05, repetition_penalty=1.5)
print(outputs[0]["generated_text"])
```

```
model.save_pretrained("llama-3-8b-industry-code-with-adapter")
tokenizer.save_pretrained("llama-3-8b-industry-code-with-adapter")
```

```
model.push_to_hub("llama-3-8b-industry-code-with-adapter", use_temp_dir=False)
tokenizer.push_to_hub("llama-3-8b-industry-code-with-adapter", use_temp_dir=False)
```


Kwantyzacja

Kwantyzacja modelu LLM (Large Language Model) to proces optymalizacji dużych modeli językowych poprzez zmniejszenie precyzji numerycznej wag i aktywacji, co prowadzi do mniejszego zużycia pamięci i szybszych obliczeń. W przypadku LLM, takich jak GPT czy BERT, kwantyzacja jest szczególnie przydatna, ponieważ modele te mają miliardy parametrów, co czyni je kosztownymi pod względem pamięci i czasu przetwarzania.

Cel kwantyzacji w LLM:

Kwantyzacja LLM ma na celu zmniejszenie wymagań obliczeniowych, bez znaczącego pogorszenia wydajności modelu w zadaniach takich jak generowanie tekstu, rozumienie języka, tłumaczenie itp. Dla LLM, kwantyzacja umożliwia uruchamianie modeli na urządzeniach z ograniczonymi zasobami lub przyspiesza inferencję na serwerach w chmurze.

- Redukcja precyzji liczb
- Kwantyzacja wag
- Kwantyzacja aktywacji

Parameter-Efficient Fine-Tuning (PEFT) - kwantyzacja

```
%cd /kaggle/working
!git clone --depth=1 https://github.com/ggerganov/llama.cpp.git
%cd /kaggle/working/llama.cpp
!sed -i 's|MK_LDFLAGS += -lcuda|MK_LDFLAGS += -L/usr/local/nvidia/lib64 -lcuda|' Makefile
!LLAMA_CUDA=1 conda run -n base make -j > /dev/null
```

```
%cd /kaggle/working/

!./llama.cpp/llama-quantize /kaggle/input/llama3-fine-tuning-3-convert/llama-3-8b-industry-code.gguf llama-3-8b-industry-code-Q4_K_M.gguf Q4_K_M
```

Parameter-Efficient Fine-Tuning (PEFT) - kwantyzacja

```
from huggingface_hub import login
from kaggle_secrets import UserSecretsClient
from huggingface_hub import HfApi
user_secrets = UserSecretsClient()
hf_token = user_secrets.get_secret("HUGGINGFACE_TOKEN")
login(token = hf_token)

api = HfApi()
api.upload_file(
    path_or_fileobj="/kaggle/working/llama-3-8b-industry-code-Q4_K_M.gguf",
    path_in_repo="llama-3-8b-industry-code-Q4_K_M.gguf",
    repo_id="rzarno/llama-3-8b-industry-code",
    repo_type="model",
)
```

Jupyter notebook:
`transformers_fine_tune.ipynb`

RLHF - Reinforcement Learning on Human Feedback

Reinforcement Learning from Human Feedback (RLHF) to technika stosowana w trenowaniu modeli uczenia maszynowego, gdzie algorytmy uczenia wzmacniającego są doskonalone poprzez wykorzystanie opinii ludzi.

RLHF łączy tradycyjne uczenie wzmacniające z ocenami i wskazówkami dostarczonymi przez ludzi, aby lepiej dostosować modele do oczekiwań użytkowników.

Kroki:

1. **Generowanie odpowiedzi przez model:** Model (np. LLM, czyli Large Language Model) generuje różne odpowiedzi na zapytania użytkownika.
2. **Ocena przez człowieka:** Ludzie oceniają generowane odpowiedzi, wskazując, które są bardziej trafne, przydatne lub etyczne.
3. **Zastosowanie uczenia wzmacniającego:** Na podstawie tych ocen, model dostosowuje swoje przyszłe odpowiedzi, przyznając "nagrody" za bardziej pożądane odpowiedzi, a "kary" za te mniej pożądane. Celem jest maksymalizacja oczekiwanej sumy nagród w dłuższym okresie.

RLHF - Reinforcement Learning on Human Feedback

```
def reward_function(response):  
    # Użyj modelu detekcji toksyczności, który zwraca wartość toksyczności w przedziale [0, 1]  
    toxicity_score = toxicity_model.predict(response)  
  
    # Ustaw próg toksyczności. Wszystko poniżej 0.3 jest akceptowalne  
    threshold = 0.3  
  
    if toxicity_score < threshold:  
        # Jeśli odpowiedź jest nietoksyczna, przyznaj pozytywną nagrodę  
        return 1.0  
    else:  
        # Jeśli odpowiedź jest toksyczna, przyznaj ujemną nagrodę  
        return -1.0
```