LLaMA-Adapter

Title: LLaMA-Adapter: Efficient Fine-tuning of Language Models with Zero-init Attention
Venue: ICML 2023

Reviewer: HyeongJun Do
Last Updated: 05.06.2024
Refence

> Paper Link: https://arxiv.org/abs/2303.16199
> Github: https://github.com/OpenGVLab/LLaMA-Adapter/tree/main/alpaca_finetuning_v1

---

1. Introduction

- **대규모 언어 모델**(LLMs): 최근 LLMs는 언어 이해 및 생성 능력에서 큰 진전을 보였음.
- **문제점**: 기존의 LL됨. 이 메커니즘은 zero gating을 사용하여 학습 초기에는 기존 모델의 성능을 유지하고 점진적으로 새로운 신호를 통합함.

수식 2: Zero-initialized Attention

$$S_l = Q_l K_l^T / \sqrt{C}$$

여기서 Q_l , K_l는 쿼리와 키 매트릭스.

수식 3: Gating Factor 적용

$$S_g = \left[\text{softmax}\left(S_l^K\right) \cdot g_l; \text{softmax}\left(S_l^{M+1}\right)\right]$$

Figure 2: Zero-initialized attention 메커니즘 다이어그램

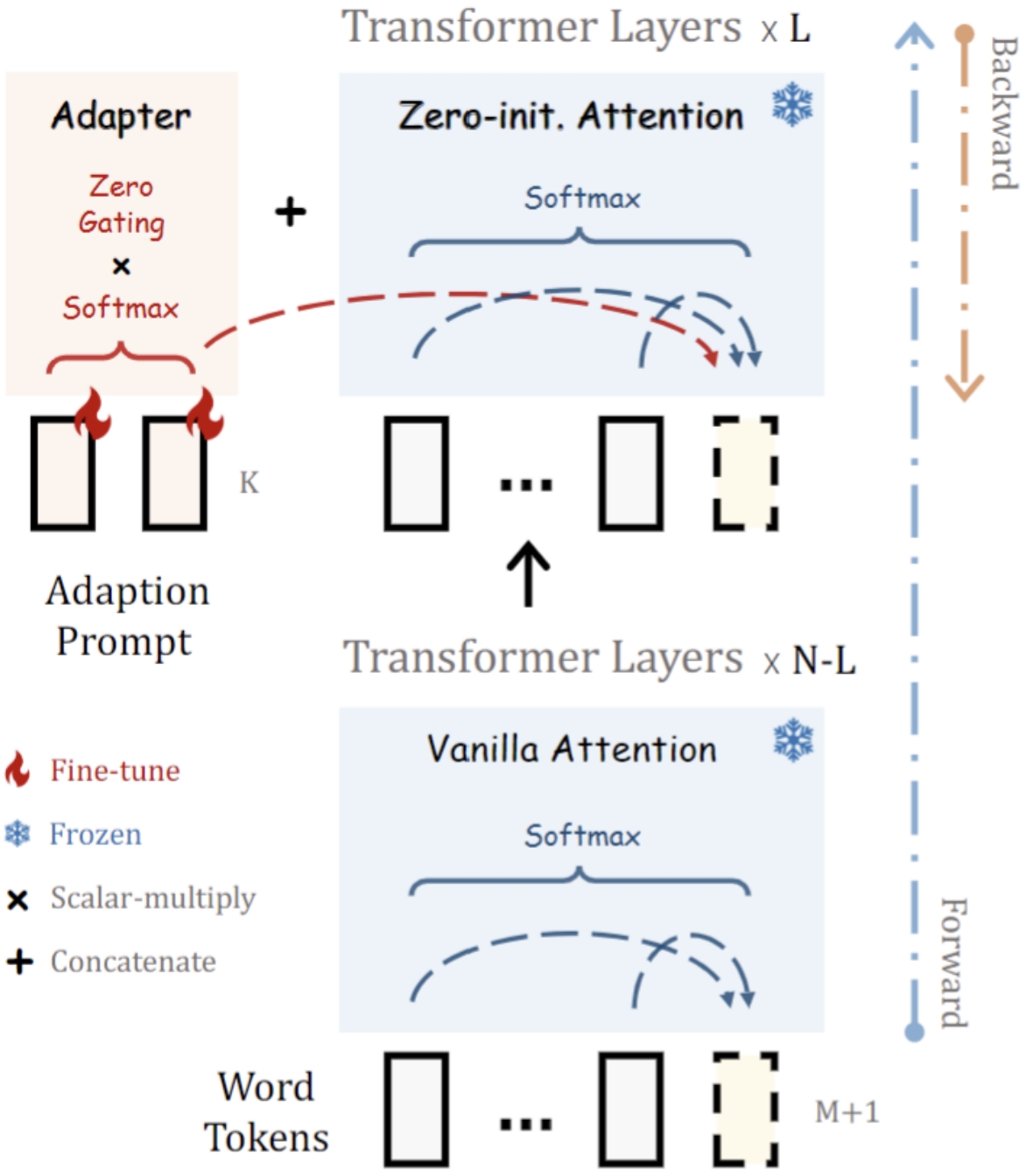

Figure 2: **Details of LLaMA-Adapter.** We insert lightweight adapters with learnable prompts into L out of N transformer layers of LLaMA. To progressively learn the instructional knowledge, we adopt zero-initialized attention with gating mechanisms for stable training in early stages.

- 초기화된 주의 메커니즘과 zero gating이 적용된 모습을 다이어그램으로 설명.

3.3 Multi-modal Extension

- **다중 모달 학습**: 이미지 기반 학습 적용.
- **확장 가능성**: 다양한 모달리티와의 통합.

LLaMA-Adapter는 다중 모달 학습으로 확장 가능하며, 텍스트와 이미지의 조합을 처리 가능. 이를 위해 이미지 기반 프롬프트를 추가하여, 모델이 이미지 조건부 언어 생성 작업을 수행할 수 있음. 이 방법은 ScienceQA와 COCO Caption 벤치마크에서 우수한 성능을 보였으며, 이는 다중 모달 학습의 가능성을 증명함.

**Fig임. 이미지 조건부 언어 생성 작업에서 ScienceQA와 COCO Caption 벤치마크를 사용하여 평가되었으며, 기존의 이미지 기반 모델들과 비교하여 경쟁력 있는 성능을 입증함.

Table 2: Multi modality 작업 성능 비교 표

Table 2: **Question Answering Accuracy (%) on ScienceQA's [41] test set.** We report GPT-3 [4], ChatGPT [2], and GPT-4 [45] for zero-shot inference. CoT denotes to utilize additional chain of thought for question answering. T denotes the single-modal model with text-only input.

Model	Tuned Params	Avg	NAT	SOC	LAN	TXT	IMG	NO	G1-6	G7-12
Random Choice [41]	-	39.83	40.28	46.13	29.25	47.45	40.08	33.66	39.35	40.67
Human [41]	-	88.40	90.23	84.97	87.48	89.60	87.50	88.10	91.59	82.42
MCAN [65]	95M	54.54	56.08	46.23	58.09	59.43	51.17	55.40	51.65	59.72
VisualBERT [33, 34]	111M	61.87	59.33	69.18	61.18	62.71	62.17	58.54	62.96	59.92
UnifiedQA [27]	223M	70.12	68.16	69.18	74.91	63.78	61.38	77.84	72.98	65.00
UnifiedQA$_{CoT}$	223M	74.11	71.00	76.04	78.91	66.42	66.53	81.81	77.06	68.82
GPT-3 [4]	0M	74.04	75.04	66.59	78.00	74.24	65.74	79.58	76.36	69.87
GPT-3$_{CoT}$	0M	75.17	75.44	70.87	78.09	74.68	67.43	79.93	78.23	69.68
ChatGPT$_{CoT}$ [2]	0M	78.31	78.82	70.98	83.18	77.37	67.92	86.13	80.72	74.03
GPT-4$_{CoT}$ [45]	0M	83.99	85.48	72.44	90.27	82.65	71.49	92.89	86.66	79.04
MM-COT$_T$ [74]	223M	70.53	71.09	70.75	69.18	71.16	65.84	71.57	71.00	69.68
MM-COT	223M	84.91	87.52	77.17	85.82	87.88	82.90	86.83	84.65	85.37
LLaMA-Adapter$_T$	**1.2M**	**78.31**	**79.00**	**73.79**	**80.55**	**78.30**	**70.35**	**83.14**	**79.77**	**75.68**
LLaMA-Adapter	**1.8M**	**85.19**	**84.37**	**88.30**	**84.36**	**83.72**	**80.32**	**86.90**	**85.83**	**84.05**

- 모델, 파라미터 수, ScienceQA 및 COCO Caption 벤치마크 성능 지표를 비교한 표.

4.3 Generalization to Other Models

- **일반화 성능**: 다른 사전 학습 모델에의 적용.
- **적용 사례**: ViT, RoBERTa.

Zero-initialized attention 메커니즘은 ViT, RoBERTa와 같은 다른 사전 학습 모델에도 적용 가능. 이들 모델의 전통적인 비전 및 언어 작업에서 우수한 성능을 보였으며, 제안된 메커니즘의 일반화 능력을 입증.

Table 3: 다른 모델에의 일반화 성능 비교 표

Table 3: **Ablation on Inserted Layers** of LLaMA's transformer.

Layers	Params	Val Acc (%)
10	0.97	55.95
20	1.37	73.36
30	1.79	**83.85**

Table 4: **Ablation on Zero-initialized Attention.** Blue highlights the gain.

Setting	Val Acc (%)
Rand-Init Attention	40.77
Zero-Init Attention	**83.85**
Gain	+43.08

- ViT, RoBERTa 등의 모델에 대해 zero-initialized attention 메커니즘 적용 후 성능을 비교한 표.

---

5. Conclusion

- **연구 요약**: LLaMA-Adapter의 주요 기여.
- **향후 연구 방향**: 추가적인 연구 가능성.

LLaMA-Adapter는 효율적인 미세 조정 방법으로, 적은 파라미터로도 높은 성능을 유지할 수 있는 방법을 제안함. Zero-initialized attention 메커니즘을 통해 기존 지식을 보존하면서 새로운 지시 신호를 통합하는 혁신적인 방법을 소개. 향후 연구에서는 더 다양한 모델과 작업에 대한 적용 가능성을 탐구할 예정.

---

6. Code

zero-initialized attention 메커니즘

```
import torch
import torch.nn as nn
from transformers import LlamaModel

class LLaMAAdapterV1(nn.Module):
    def __init__(self, llama_model, prompt_length, adapter_dim):
        super(LLaMAAdapterV1, self).__init__()
        self.llama = llama_model
        self.prompt_embeddings = nn.Parameter(torch.randn(prompt_length,
llama_model.config.hidden_size))
        self.adapter = nn.Linear(llama_model.config.hidden_size,
adapter_dim)
        self.zero_gating = nn.Parameter(torch.zeros(adapter_dim))

    def forward(self, input_ids, attention_mask=None):
        # LLaMA 모델의 출력
        hidden_states = self.llama(input_ids,
attention_mask=attention_mask).last_hidden_state

        # Adaption Prompt 추가
```

```
        prompts = self.prompt_embeddings.expand(hidden_states.size(0), -1,
-1)
        hidden_states = torch.cat([prompts, hidden_states], dim=1)

        # Zero-initialized Attention 적용
        attention_scores = torch.matmul(hidden_states,
hidden_states.transpose(-1, -2)) / self.llama.config.hidden_size ** 0.5
        attention_probs = torch.nn.functional.softmax(attention_scores,
dim=-1)
        gated_attention = attention_probs * self.zero_gating
        hidden_states = torch.matmul(gated_attention, hidden_states)

        return hidden_states
```

- **prompt_embeddings**: 학습 가능한 프롬프트 임베딩을 초기화
- **adapter**: LLaMA 모델의 출력 차원을 어댑터 차원으로 변환
- **zero_gating**: 제로 초기화된 게이팅 파라미터를 정의
- **forward 메서드**: 입력 토큰에 대한 LLaMA 모델의 출력을 얻고, 프롬프트 임베딩을 추가하여 zero-initialized attention 메커니즘을 적용

Finetuning(Training)

```
import torch
from transformers import Trainer, TrainingArguments, LlamaTokenizer,
LlamaForSequenceClassification
from models_llama_adapter import LLaMAAdapterV1

# 사전 학습된 LLaMA 모델 로드
llama_model = LlamaForSequenceClassification.from_pretrained('llama-base')
tokenizer = LlamaTokenizer.from_pretrained('llama-base')

# LLaMA-Adapter V1 초기화
llama_adapter = LLaMAAdapterV1(llama_model, prompt_length=10,
adapter_dim=512)

# 학습 설정
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=3,
    per_device_train_batch_size=8,
    save_steps=10_000,
    save_total_limit=2,
)

# 트레이너 초기화
trainer = Trainer(
    model=llama_adapter,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=eval_dataset,
)

# 학습 시작
```

```
trainer.train()
```

- **TrainingArguments**: 학습 설정을 정의 (출력 디렉토리, 에포크 수, 배치 크기 등)
- **Trainer**: 모델, 학습 설정, 데이터셋을 사용하여 트레이너를 초기화
- **trainer.train()**: 모델 학습을 시작

Evaluation

```
import torch
from transformers import Trainer, TrainingArguments, LlamaTokenizer,
LlamaForSequenceClassification
from models_llama_adapter import LLaMAAdapterV1

# 사전 학습된 LLaMA 모델 로드
llama_model = LlamaForSequenceClassification.from_pretrained('llama-base')
tokenizer = LlamaTokenizer.from_pretrained('llama-base')

# LLaMA-Adapter V1 초기화
llama_adapter = LLaMAAdapterV1(llama_model, prompt_length=10,
adapter_dim=512)

# 평가 설정
training_args = TrainingArguments(
    output_dir='./results',
    per_device_eval_batch_size=8,
)

# 트레이너 초기화
trainer = Trainer(
    model=llama_adapter,
    args=training_args,
    eval_dataset=eval_dataset,
)

# 평가 시작
results = trainer.evaluate()
print(results)
```

- **Trainer**: 평가 설정과 데이터셋을 사용하여 트레이너를 초기화
- **trainer.evaluate()**: 모델 평가를 수행하고 결과를 출력