TEAM
BLIP4FASHION

FASHION CAPTIONING With BLIP Finetuning

방선유 배주원 손소현 장윤서 차수빈

INDEX

INTRODUCTION

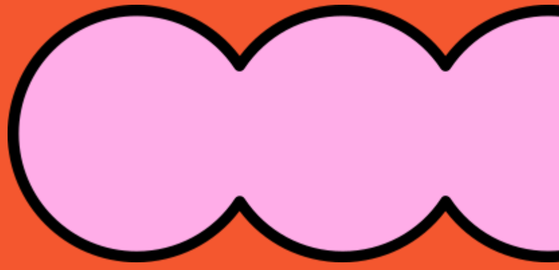


Image Captioning

❖ 이미지 캡셔닝(Image Captioning)은 이미지의 내용을 단어로 설명하는 task

❖ 컴퓨터 비전과 자연어 처리 둘 다에 대한 종합적인 이해가 필요 [멀티모달]

❖ 목표 : 주어진 이미지에 대한 높은 수준의 이해와 텍스트 생성 능력을 모델에 부여

Output : silhouette of a girl on the beach at sunset

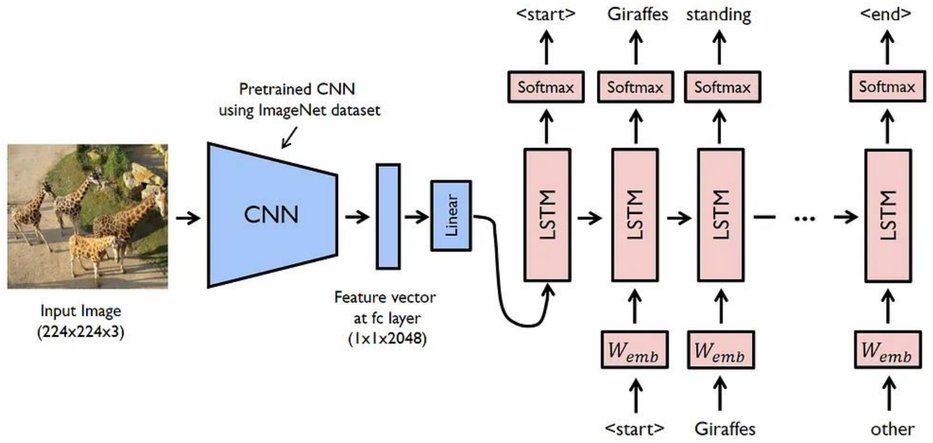
Output : ice cream in a glass bowl

1. Image Captioning

❖ 대부분의 이미지 캡셔닝 구조는 '이미지 인코더'와 '텍스트 디코더'가 결합된 형태

❖ 단계

1. 이미지 입력
2. 이미지 특성 추출
3. 문맥 학습
4. 자연어 설명 생성
5. 평가 및 최적화



2. 주제 선정 동기

- **이미지 캡셔닝이라는 큰 주제는 정한 상태에서 세부 분야를 정해야했음**
- **e-commerce 분야를 살펴보다가 구할 수 있었던 데이터가 패션이미지 + 캡션 데이터였음**
  - 온라인 패션 플랫폼 제품 자동 생성 / 패션 트렌드분석 등 활용도가 많다고 생각
- **But 일반적인 이미지 캡셔닝과 패션 이미지 캡셔닝은 본질적으로 비슷한 작업이지만 차이가 있음**
  - 전체적인 내용 설명 vs 특정한 패션 아이템에 대한 특징 (디자인, 색상, 재질 등)
  - 패션에 특화된 정보가 중요
  - 패션 도메인에 대한 특화된 어휘 학습을 목표로

Related Works

Image Captioning Models

- **이미지 인코더 + 텍스트 디코더 구조**
- **input :** 이미지 / output : 텍스트
- **train data**
  - [이미지 + 해당 이미지에 대한 캡션] pair

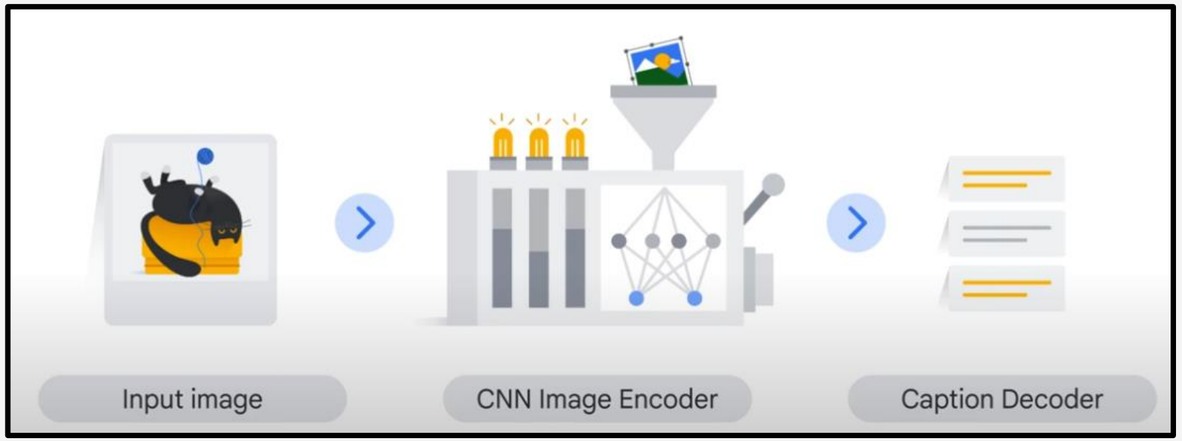
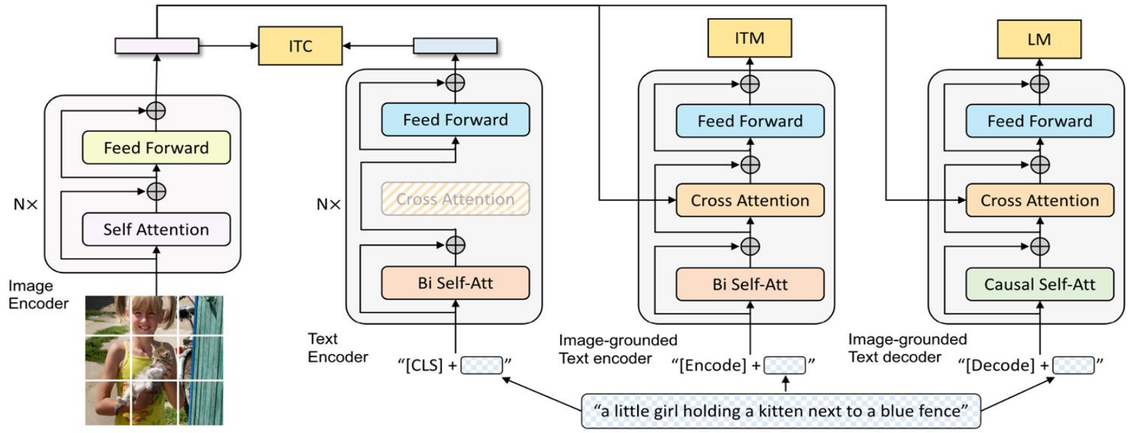


Image Captioning Models

- **대표적인 모델로는 BLIP, mPLUG, ExpansionNet 등이 있음**
  - 그중에서 가장 널리 쓰이는 BLIP에 대한 논문 리뷰를 진행하고, 파인튜닝할 모델로 선정
  - BLIP : 이미지와 텍스트 간의 상호작용 모델
  - 이미지 캡셔닝 뿐만 아니라 이미지-텍스트 매칭, 이미지 기반 질의응답 등 다양한 task에서 활용 가능
- **평가 지표**
  - 이미지 분류에서의 Accuracy, 이미지 생성에서의 FID 처럼 대표적인 평가지표의 부재
  - BLEU, METEOR, ROUGE, CIDEr, SPICE를 종합하여 평가

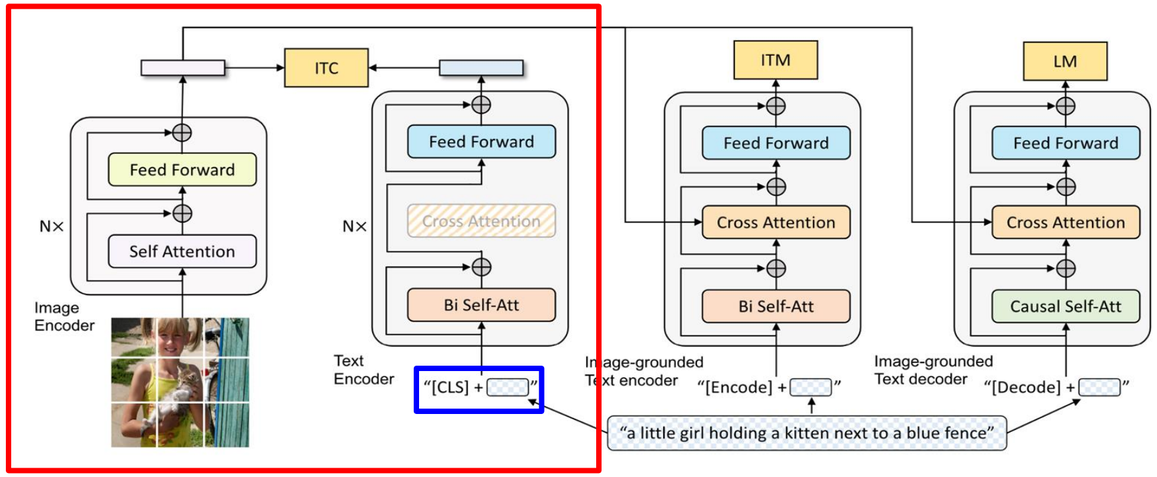
1. BLIP

- 이해와 생성 능력을 갖춘 통합 모델을 사전 학습하기 위해 멀티모달 인코더-디코더 혼합(MED) 모델을 제안
- BLIP의 MED 모델 아키텍처는 다음과 같은 세 가지 기능을 수행할 수 있음
  - Unimodal Encoder
  - Image-grounded Text Encoder
  - Image-grounded Text Decoder



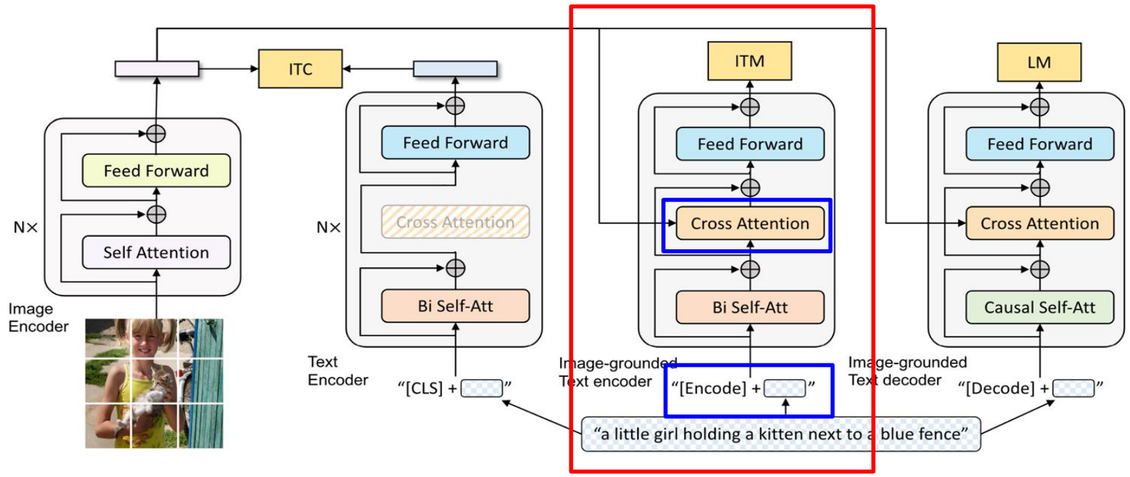
1. BLIP

- **Unimodal Encoder**
  - Image Encoder는 이미지를 패치로 분할하고 시퀀스로 인코딩하는 ViT를 사용
  - Text Encoder는 BERT 활용, 문장 요약을 위해 text 입력의 시작 부분에 [CLS] 토큰 추가



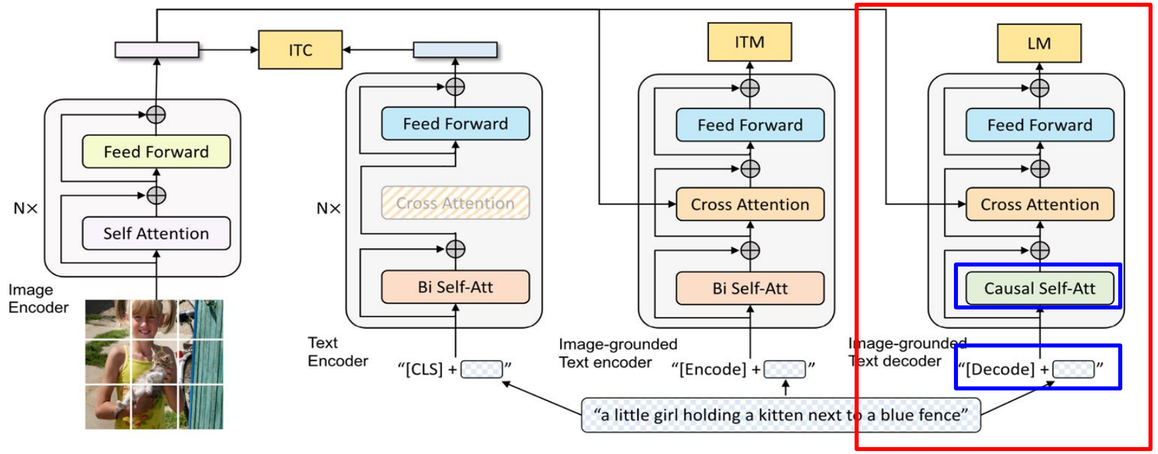
1. BLIP

- **Image-grounded Text Encoder**
  - Text Encoder에 cross-attention 층을 추가하여 시각 정보를 주입
  - 텍스트 끝에 task-specific [Encode] 토큰을 추가하고, 이 토큰의 출력 임베딩을 이미지-텍스트 쌍의 multi-modal 표현으로 사용



1. BLIP

- **Image-grounded Text Decoder**
  - Text Encoder의 self-attention 층을 causal self-attention 층으로 대체
  - [Decode] 토큰을 사용하여 시퀀스의 시작을 알리며, end-of-sequence 토큰을 사용하여 끝을 알림



DATASET

1. 데이터셋 소개

- COCO Captions
  - Microsoft에서 개발
  - 330000개 이상의 이미지와 80개 이상의 객체 카테고리 포함
  - 각 이미지에 객체의 위치와 label이 주석으로 달려있음
- FACAD
  - 다양한 계절, 연령대, 의류/액세서리 카테고리, 포즈 등의 패션 이미지를 포함
  - 패션 아이템에 대한 상세하고 섬세한 캡션을 제공
  - FACAD의 캡션은 평균 21단어로 COCO Captions의 10.4단어보다 길고, 더 풍부한 표현을 사용함("pearly", "so-simple yet so-chic", "retro flair" 등)

2. 데이터셋 구축 방법

1. FACAD 데이터셋 불러오기
2. 사물을 정면에서 바라보고 있는 이미지 데이터 추출
    - 0번 id, 색상이 여러개인 경우 첫 번째 이미지 활용
1. 이미지 파일 압축, annotation 파일 전처리
2. 데이터 개수 나누기
    a. train: 10000
    b. val: 1000
    c. test: 2000

```
[ ] visualize_samples(output_dir, 'train', num_samples = 5)
```

croc embossed leather in a pale pink hue softens the utilitarian aesthetic of a classic cap toe combat boot grounded by a toothy lug sole

2. 데이터셋 구축 방법

5. 이후 모델링 단계에서 BLIP 모델에 넣기 위해 COCO Captions format으로 맞춤

- images, annotations, info, licenses
- json 파일로 저장


```
{
    "images": [
        {
            "id": 1,
            "file_name": "image_1.jpg"
        },
```



```
"annotations": [
    {
        "id": 1,
        "image_id": 1,
        "caption": "faceted stone add sparkling glamour to gleaming hoop earring that are destined to make a statement",
        "attributes": [
            "faceted",
            "stone",
            "glamour",
            "hoop",
            "earring",
            "statement"
        ],
        "category": 21
    },
```

Modeling & Experiments

Setup General

- **모델: BLIP w/ ViT-L(blip-image-captioning-large)**
  - 이미지 size: 384 × 384
  - max caption length: 50
  - train:validation:test = 10000:2000:1000
  - fine-tuning 시 PEFT 방식 적용
- **batch_size: 16**
- **optimizer: AdamW**
  - initial learning rate: $2e^{-5}$
  - lr scheduler: cosine_schedule_with_warmup
    - warmup 비율: 10%
    - weight decay: 0.05
- **num_epochs: 10**

Setup ; PEFT+LoRA 설정

❖ LoRA(Lo-Rank Adaptation)

- ➢ 대규모 모델 파인 튜닝 시 행렬의 랭크를 줄이는 방법
- ➢ 기존 모델의 가중치 행렬을 저랭크 행렬로 분해하여 적은 수의 파라미터만을 학습하는 방식
- ➢ 메모리와 계산 비용을 줄이면서도 성능을 유지함

```
[ ]  ## PEFT + LoRA 설정

     ## LoRA Configuration
     lora_config = LoraConfig(
         r = 16, # LoRA의 rank
         lora_alpha = 32, # LoRA의 alpha
         lora_dropout = 0.05, # LoRA의 dropout 비율
         target_modules = target_modules,
         bias = "none"
     )

     # LoRA 적용 모델
     peft_model = get_peft_model(model, lora_config)
```

Modeling Overview

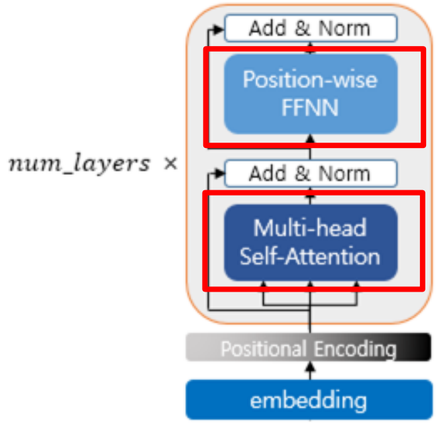
❖ Trials

#	fine-tuned layers	semantic metrics
1	Decoder only	X
2	Decoder only	O
3	Decoder + Encoder의 마지막 6개 layer	X
4	Decoder + Encoder의 마지막 6개 layer	O

Modeling Overview

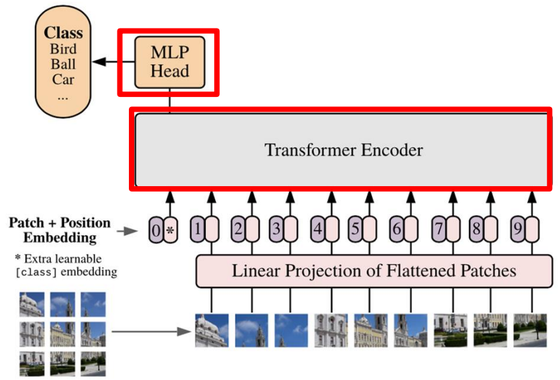
❖ Layers

➢ Decoder only



(모든 layer에 대해)
- attention layer의 query/key/value
- dense layers (Feed-Forward Neural Networks)

➢ Decoder + Encoder



(마지막 6개 layer에 대해)
- attention layer의 query/key/value
- dense layers (Feed-Forward Neural Networks)

Recap: Evaluation Metrics

- **BLEU-4**
  - 생성된 캡션과 기준 캡션(= 참고 캡션) 간의 4-gram 일치 정도를 측정
  - 0 ~ 1의 범위를 가지며, 점수가 높을수록 good
    → 점수가 높을수록 생성된 텍스트가 참고 텍스트와 더 잘 일치한다는 것을 의미
- **METEOR**
  - 단어의 형태소 일치, 어휘적 의미, 동의어 등을 고려하여 번역 품질을 평가하는 지표
  - 0 ~ 1의 범위를 가지며, 점수가 높을수록 good
- **ROUGE**
  - 텍스트 요약의 평가를 위해 개발된 지표
  - 0 ~ 1의 범위를 가지며, 점수가 높을수록 good
    → 점수가 높을수록 요약이 원본 텍스트의 중요한 부분을 잘 포함하고 있음을 의미
- **CIDEr**
  - 다수의 기준 캡션과 생성된 캡션 사이의 유사도를 n-gram으로 계산. 특히, 인간이 작성한 캡션과의 합의(consensus)를 기반으로 평가
  - 0 ~ 10의 범위를 가지며, 점수가 높을수록 good
    → 점수가 높을수록 사람이 작성한 캡션과 유사함을 의미
- **SPICE**
  - 이미지 캡션의 의미론적 구조(객체, 관계 등)를 그래프로 표현하고 이를 비교하여 평가
  - 0 ~ 1의 범위를 가지며, 점수가 높을수록 good
    → 점수가 높을수록 이미지 캡셔닝의 의미적 정확성이 높음을 의미

Modeling 1

❖ Trial 1

- ➢ Decoder만 fine-tuning
  - attention layer의 query/key/value
  - dense layers
- ➢ 평가 지표

	BLEU-4	METEOR	ROUGE	CIDEr	SPICE
scores	0.0193	0.0872	0.1812	0.2492	0.0863

❖ Trial 3

- ➢ Encoder의 마지막 6개 layer 또한 fine-tuning
  - attention layer의 query/key/value
  - dense layers
- ➢ 평가 지표

	BLEU-4	METEOR	ROUGE	CIDEr	SPICE
scores	0.0202	0.0915	0.1848	0.2750	0.0930

Modeling 2

❖ Background

➢ 패션 캡셔닝은 아이템의 **내재적 속성**을 정확히 설명해야 한다는 독특한 과제를 가지고 있음

➢ 패션 캡셔닝의 정확도를 높이기 위해 두 가지 semantic metric을 도입

- attribute-level semantic(ALS): 이미지 속성들을 더 많이 포함한 문장을 생성하도록 유도
- sentence-level semantic(SLS): 패션 아이템의 카테고리를 더 정확하게 설명하도록 생성된 문장을 유도
- 기존 수식을 구현 상의 편의성을 위해 조금 변경하여 활용

Modeling 2

❖ Background

➢ attribute-level semantic(ALS)

- 가능한 올바른 속성(attribute)을 많이 생성하도록 모델을 유도
  - attributes: 명사(NOUN), 형용사(ADJ)
- 생성된 caption이 데이터의 속성을 얼마나 잘 포함하고 있는지 평가
  - 생성된 문장과 참조 문장에서 "공통" attributes의 비율을 계산
  - reference_caption에 적절한 속성이 없는 경우 보상을 0으로 설정

```python
def extract_attributes(caption):
    doc = nlp(caption)
    attributes = [token.text for token in doc if token.pos_ in ['NOUN', 'ADJ']]
    return attributes
```

```python
def compute_als(generated_caption, reference_attributes):
    generated_attributes = extract_attributes(generated_caption)

    # 패딩된 속성 제거(공백)
    reference_attributes = [attr for attr in reference_attributes if attr != ""]

    common_attributes = set(generated_attributes) & set(reference_attributes)
    return len(common_attributes) / len(reference_attributes) if reference_attributes else 0
```

Modeling 2

❖ Background

➢ sentence-level semantic(SLS)

- 패션 아이템의 카테고리를 더 정확하게 설명하도록 생성된 문장을 유도
- 생성된 caption이 category를 반영하고 있는지 평가
  - 포함되면 1, 아니면 0

```
{
    "1": "backpack",
    "2": "bag",
    "3": "belt",
    "4": "blazer",
    "5": "blouse",
    "6": "bodysuit",
    "7": "boot",
    "8": "bottom",
    "9": "bra",
    "10": "bracelet",
```

```
[ ]  def load_category_wordmap(file_path):
         with open(file_path, 'r') as f:
             return json.load(f)

     category_wordmap = load_category_wordmap('/content/drive/MyDrive/Euron/6th-project/final/data/cates_wordmap.json')
```

```
[ ]  def get_category_from_caption(caption):
         caption_words = set(caption.lower().split())
         for cat_id, cat_name in category_wordmap.items():
             if cat_name.lower() in caption_words:
                 return int(cat_id)
         return None
```

```
[ ]  def compute_sls(generated_caption, reference_category):
         predicted_category = get_category_from_caption(generated_caption)
         return 1.0 if predicted_category == reference_category else 0.0
```

Modeling 2

❖ Background

➢ computing gradient(update rule)

- 앞에서 정의한 두 보상을 합하여 활용:
- reward의 목표: 최대화 ⇔ loss의 목표: 최소화

$$r = r_{\text{ALS}} + r_{\text{SLS}}$$
$$L_r = -\mathbf{E}_{Y' \sim p_\theta}[r(Y')]$$

- 위의 두 보상 최적화 정책은 미분이 불가능함
  - MLE training 동안 강화 학습 과정으로 보완
- Monte Carlo Estimate를 활용하여 p_θ로부터 샘플링 된 H개의 sample로 근사

$$\nabla_\theta L_r(\theta) \simeq \frac{1}{H}\sum_{j=1}^{H}[(r_j(Y_j') - b)\nabla_\theta \log p_\theta(Y_j')]$$

- $b = \frac{1}{H}\sum_{j=1}^{H} r(Y_j')$
- $Y_j' \sim p_\theta$: 모델 p_θ로부터 샘플링 된 j번째 문장
- $r_j(Y_j')$: 그에 해당하는 보상

Modeling 2

- ❖ Background
  - ➢ computing gradient(update rule)
    - gradient descent를 2번 적용

```
## === Training ===
model.train()
epoch_train_loss = 0

for images, captions, attributes, categories in tqdm(train_loader, desc=f"Training Epoch {epoch+1}/{num_epochs}"):
    images = images.to(device)
    inputs = processor(images=images, text=captions, return_tensors="pt", padding=True).to(device)

    outputs = model(**inputs, labels=inputs.input_ids)
    mle_loss = outputs.loss

    generated_ids = model.generate(pixel_values=inputs.pixel_values, max_length=50)
    generated_texts = processor.batch_decode(generated_ids, skip_special_tokens=True)
    als_rewards, sls_rewards = compute_rewards(generated_texts, attributes, categories)

    # MLE 손실 및 gradient 계산
    optimizer.zero_grad()
    mle_loss.backward(retain_graph=True)
```

```
    # REINFORCE를 통한 reward gradient 계산
    log_probs = torch.nn.functional.log_softmax(outputs.logits, dim=-1)
    input_ids_expanded = inputs.input_ids.unsqueeze(-1)
    gathered_log_probs = log_probs.gather(2, input_ids_expanded).squeeze(-1)

    batch_size, seq_length = gathered_log_probs.size()
    reward = torch.tensor(als_rewards, device=device).unsqueeze(1).expand(batch_size, seq_length)

    baseline = reward.mean()
    reinforce_loss = -torch.mean((reward - baseline) * gathered_log_probs)
    reinforce_loss.backward()

    optimizer.step()
    scheduler.step()

    total_loss = 0.5 * mle_loss + 0.5 * reinforce_loss
    epoch_train_loss += total_loss.item()
```

Modeling 2

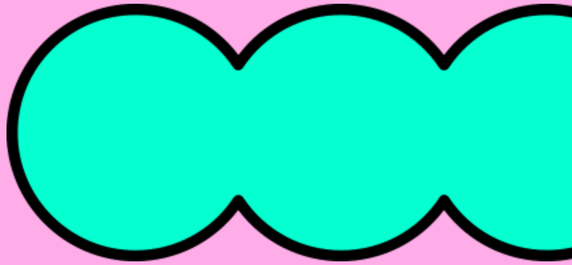
- Trial 2
  - Decoder만 fine-tuning
    - attention layer의 query/key/value
    - dense layers
  - 평가 지표

	BLEU-4	METEOR	ROUGE	CIDEr	SPICE
scores	0.0172	0.0855	0.1789	0.2347	0.0870

- Trial 4
  - Encoder의 마지막 6개 layer 또한 fine-tuning
    - attention layer의 query/key/value
    - dense layers
  - 평가 지표

	BLEU-4	METEOR	ROUGE	CIDEr	SPICE
scores	0.0180	0.0876	0.1782	0.2442	0.0905

CONCLUSION



1. Evaluation

❖ Test scores

Trials	BLEU-4	METEOR	ROUGE	CIDEr	SPICE
1	0.0171	0.0873	0.1816	0.2448	0.0909
2	0.0168	0.0864	0.1797	0.2355	0.0889
3	**0.0200**	**0.0888**	**0.1820**	**0.2592**	**0.0916**
4	0.0190	0.0883	0.1818	0.2504	0.0912

➢ Trial 1 vs 3: Decoder만 fine-tuning하는 것보다 Encoder도 일부 fine-tuning을 하는 것이 더 효과적임

➢ Trial 3 vs 4: RL metric을 활용하지 않는 경우가 일반화된 성능은 더 높게 나타남

2. Inference(Human Evaluation)

❖ Trial 1 vs 3

➢ Encoder도 일부 fine-tuning을 실행한 경우 더 풍부한 caption이 생성됨

Trial 1: a sporty tee with a sporty side stripe and a logo graphic at the chest for a sporty look

Trial 3: a classic logo graphic and a classic tape sleeve make this t tee a classic staple of the adidas brands iconic adidas brand

2. Inference(Human Evaluation)

❖ Trial 3 vs 4

➢ RL Metric을 추가하는 경우 fashion item의 속성 및 카테고리를 더 잘 반영

Trial 3: a classic logo graphic and a classic tape sleeve make this t tee a classic staple of the adidas brands iconic adidas brand

Trial 4: a t shirt with a logo and a logo embroidered on the chest and a white stripe down the chest

기대효과

- 패션 이미지를 이해하고, 이에 대한 적절한 캡션을 자동으로 생성하는 모델을 구축함으로써 패션 산업에 새로운 기술적 도약을 도모할 수 있다.
- 이미지 캡션을 자동으로 생성하고, 이를 검색 엔진에 최적화(SEO)하여 검색 엔진에서 높은 순위를 차지하도록 할 수 있다.
- 번역 모델과 결합하여, 다국어 이미지 캡셔닝을 통해 전 세계적으로 다양한 언어로 패션 카탈로그를 제공할 수 있다.
- 시각장애인을 위한 대체 텍스트 생성에 활용할 수 있다.

FASHION CAPTIONING WITH BLIP FINETUNING

THANK YOU

QUESTIONS
AND
OPINIONS