ONECLICK AI

Vit

연준모

목차

0. 들어가기에 앞서

Transformer

오늘 하는 내용들은 전부 다 트랜스포머 아키텍쳐 기반의 모델

어떻게 트랜스포머의 구조를 원하는 방식대로 사용하는지 알아보자

1. Vit

Vit

지금까지, 모든 이미지 처리 모델에선 Conv를 사용하였다

하지만, 텍스트 처리를 목적으로 만든 Transformer에선 이미지를 처리할 수 없다
따라서, Transfermer 아키텍쳐에 어거지로 Conv를 박아둔게 Vit다

놀랍게도, Vit는 기존의 이미지처리 모델을 전부 이기고 가장 강한 모델로 자리잡고 있다.

Vit

단어만 취급하는 Transformer. 어떻게 CNN을 박았을까?

이미지 패치 (Image Patches)

ViT는 이미지를 **단어(Word)**처럼 간주하는 대신, 이미지를 여러 개의 작은 **패치(Patch)**로 분할한다

분할 : 원본 이미지를 $N \times N$ 크기의 겹치지 않는 패치들로 잘라낸다 예 : 16×16 픽셀

패치 임베딩 (Patch Embedding) : 각 패치의 픽셀 값들을 일렬로 펼친(Flattened) 다음, 학습 가능한 **선형 변환 (Linear Projection)**을 통해 트랜스포머의 입력 차원(d_{model})을 갖는 벡터(Embedding)로 변환한다.
이는 텍스트에서 토큰을 임베딩 벡터로 변환하는 과정과 정확히 같습니다.

Patch 자르기 → Flatten → Linear Layer (Dense)

결과 : $H \times W \times C$ 차원의 이미지는 L개의 '패치 토큰' 시퀀스(길이 L)로 변환되어
트랜스포머의 인코더 입력이 된다.

Vit

클래스 토큰 (Class Token) 추가

문장 전체의 의미를 파악하여 번역하는 것처럼, ViT는 이미지 전체의 정보를 요약하기 위해 특별한 토큰을 시퀀스 맨 앞에 추가한다

[CLS] 토큰 : 이 토큰은 BERT 모델에서 유래했으며, 다른 모든 패치 토큰들과 함께 셀프 어텐션을 거쳐 이미지 전체의 문맥(Context) 정보를 압축한다.

최종적으로 트랜스포머 인코더의 출력을 이 **[CLS] 토큰 위치의 벡터**만 가져와 분류를 위한 Feed Forward Network(헤드)에 통과시킵니다.

Vit

위치 정보 (Positional Embedding) 추가

트랜스포머는 순환(Recurrence) 구조를 없앴기 때문에, 순서 정보가 사라진다다.
따라서 각 패치의 **위치 정보**를 담은 벡터를 패치 임베딩에 더해준다.

이미지에서 패치의 **위치(Spatial Information)**는 매우 중요하다.
ViT에서는 학습 가능한 **1D 위치 임베딩(Learnable Positional Encoding)**을 사용하며, 이는 토큰 임베딩 벡터와 합쳐져 최종 입력 벡터를 완성한다

$$\text{Input Vector} = \text{Patch Embedding} + \text{Positional Embedding}$$

Vit 아키텍쳐 구조

트랜스포머의 인코더 블록을 그대로 사용한다

MHA : 각 패치 토큰이 이미지 내의 **다른 모든 패치 토큰**과 얼마나 관련이 있는지 동시에 파악한다.
이를 통해 CNN의 커널이 국소적인 영역만 보는 것과 달리,
이미지 전체를 한 번에 보고 전역적인 관계를 학습한다.
(예: '강아지 얼굴' 패치는 '꼬리' 패치에 더 높은 가중치를 부여)

FFN : MHA를 통과하며 풍부해진 문맥 정보를 비선형적으로 한 번 더 변환하여 표현력을 높인다.

Add & Norm : 깊은 모델의 학습을 안정화하고 정보 소실(Vanishing Gradient)을 방지하는
잔차 연결(Residual Connection)과 계층 정규화가 필수적으로 적용된다.

Vit 아키텍쳐 구조

트랜스포머의 인코더 블록을 그대로 사용한다

CNN의 특성	ViT에서 구현하는 방식	트랜스포머 원리
지역적 특징 추출 (Local Feature)	**패치 임베딩**을 통해 인접한 픽셀 정보를 하나의 토큰 벡터에 압축	[cite_start]선형 변환($x \cdot W$) [cite: 115]
계층적 특징 학습 (Hierarchical)	여러 개의 인코더 레이어를 쌓아 올리며, **깊은 층**으로 갈수록 더 추상적이고 복잡한 **전역적(Global)** 특징을 학습	[cite_start]**스택 구조** [cite: 40, 281]
전역적 문맥 파악 (Global Context)	**Multi-Head Self-Attention**을 통해 한 패치가 이미지의 모든 다른 패치와 직접 관계를 계산	[cite_start]O(1)의 직접 연결 경로 [cite: 384]

2. Vit 수식으로 열어보기

1. 이미지 토큰화 및 임베딩

패치 추출 및 선형 임베딩

원본 이미지 $\mathbf{x} \in R^{H \times W \times C}$ (높이 H, 너비 W, 채널 C)는 크기 $P \times P$의 L개 패치로 분할된다.
여기서 패치의 개수 L은 다음과 같다.

$$L = \frac{H \cdot W}{P^2}$$

1. 각 패치는 $\mathbf{x_i} \in R^{P^2 \cdot C}$ 차원의 벡터로 펼쳐진(flattened) 후
2. 학습 가능한 임베딩 가중치 행렬 $\mathbf{E} \in R^{(P^2 \cdot C) \times D}$를 통해 트랜스포머의 모델 차원 D를 갖는 벡터로 변환

D는 트랜스포머 논문에서 d_{model}에 해당.

$$\mathbf{z_0^i} = \mathbf{x_i E} \quad \text{for } i = 1, \ldots, L$$

2. Vit 수식으로 열어보기

1. 이미지 토큰화 및 임베딩

클래스 토큰 및 위치 임베딩

1. 이미지 전체를 대표하는 학습 가능한 CLS 토큰 $\mathbf{z_0^0}$을 시퀀스 맨 앞에 추가
2. 각 패치(토큰)의 공간적 위치 정보를 제공하기 위해 학습 가능한 위치 임베딩 $\mathbf{E_{pos}} \in R^{(L+1)\times D}$가 더해진다

최종 입력 임베딩 $\mathbf{z_0}$는 다음과 같다.

$$\mathbf{z_0} = [\mathbf{z_0^0}; \mathbf{z_0^1}; \dots; \mathbf{z_0^L}] + \mathbf{E_{pos}}$$

$\mathbf{z_0} \in R^{(L+1)\times D}$는 이제 트랜스포머 인코더의 첫 번째 레이어로 들어갈 준비가 된 **패치 토큰의 시퀀스**이다.

2. Vit 수식으로 열어보기

2. MHA

MHA는 각 패치 토큰이 시퀀스 내의 **다른 모든** 패치 토큰들과 얼마나 관련 있는지 계산하여 **전역적인 문맥 정보**를 추출한다.

1. Q, K, V 벡터의 생성

입력 시퀀스 $\mathbf{z} \in R^{(L+\mathbb{1})\times D}$는 학습 가능한 가중치 행렬 $\mathbf{W^Q}, \mathbf{W^K}, \mathbf{W^V} \in R^{D\times D}$를 통해 Query ($\mathbf{Q}$), Key ($\mathbf{K}$), Value ($\mathbf{V}$) 행렬로 변환된다.

$$\mathbf{Q} = \mathbf{z}\mathbf{W^Q}$$
$$\mathbf{K} = \mathbf{z}\mathbf{W^K}$$
$$\mathbf{V} = \mathbf{z}\mathbf{W^V}$$

2. Vit 수식으로 열어보기

2. MHA

MHA는 각 패치 토큰이 시퀀스 내의 **다른 모든** 패치 토큰들과 얼마나 관련 있는지 계산하여 **전역적인 문맥 정보**를 추출한다.

2 Scaled Dot-Product Attention (SDPA)

MHA는 h개의 병렬적인 어텐션 헤드(Head)로 구성되며, 각 헤드 i는 저차원 d_k를 사용한다. 여기서 $d_k = D/h$이다. 각 헤드에서 Scaled Dot-Product Attention은 다음 식을 따른다.

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\mathbf{T}}}{\sqrt{d_k}}\right)\mathbf{V}$$

2. Vit 수식으로 열어보기

2. MHA

MHA는 각 패치 토큰이 시퀀스 내의 **다른 모든** 패치 토큰들과 얼마나 관련 있는지 계산하여 **전역적인 문맥 정보**를 추출한다.

+ 장거리 의존성 파악 메커니즘

1. **유사도 계산 ($\mathbf{QK^T}$):**

Query 행렬 $\mathbf{Q}$와 Key 행렬 $\mathbf{K}$의 내적은 시퀀스 내의 모든 토큰 쌍 간의 유사도 점수 행렬을 생성한다.
만약 패치 i**의 쿼리 $\mathbf{q_i}$가 패치 j**의 키 $\mathbf{k_j}$와 내적될 때 높은 점수를 얻는다면, 이는 두 패치가 서로 문맥적으로 관련이 깊다는 것을 의미한다.
패치 i와 j가 이미지 상에서 아무리 멀리 떨어져 있더라도 (즉, 장거리 의존성), 이 내적을 통해 **단 한 번의 행렬 연산**으로 관계가 직접 연결된다.
CNN에서는 먼 곳의 관계를 파악하려면 여러 층을 거쳐야만 한다. 이에 비해 매우 효율적이다.

2. Vit 수식으로 열어보기

2. MHA

MHA는 각 패치 토큰이 시퀀스 내의 **다른 모든** 패치 토큰들과 얼마나 관련 있는지
계산하여 **전역적인 문맥 정보**를 추출한다.

+ 장거리 의존성 파악 메커니즘

2. **스케일링 ($\frac{1}{\sqrt{d_k}}$):**

Key 벡터의 차원 d_k가 커질 때 내적 값이 너무 커져 Softmax의 기울기(gradient)가 소실되는 현상(Vanishing Gradient)을 방지하기 위해 사용된다.

3. **가중합 ($\mathbf{softmax}(\dots)\mathbf{V}$):**

Softmax를 통과한 어텐션 가중치(Attention Weights)는 $\mathbf{V}$ 행렬의 각 행(각 패치의 '본질적' 정보)을 **가중 평균**한다.
즉, 패치 i의 새로운 출력 벡터 $\mathbf{z_i}$는 이미지 내에서 자신과 관련 깊다고 판단된 다른 모든 패치들의 정보($\mathbf{v_j}$)를 가져와 융합한 결과이다.

2. MHA

MHA는 각 패치 토큰이 시퀀스 내의 **다른 모든** 패치 토큰들과 얼마나 관련 있는지 계산하여 **전역적인 문맥 정보**를 추출한다.

3 다중 헤드 (Multi-Head)의 효과

각 헤드 i는 서로 다른 가중치 행렬 $\mathbf{W_i^Q}, \mathbf{W_i^K}, \mathbf{W_i^V}$를 사용하므로, h개의 헤드는 각기 다른 "관점" 또는 "관계 유형"에 주목한다.

- **헤드 1 :** 색상이나 질감 관계에 주목 (예: '하늘' 패치와 '구름' 패치).
- **헤드 2 :** 기하학적/문법적 관계에 주목 (예: '강아지 몸통' 패치와 '다리' 패치).

h개의 헤드 출력 $\text{head}_i \in R^{(L+\mathbb{1})\times d_k}$는 모두 이어 붙여지고 (Concatenation), 최종 가중치 행렬 $\mathbf{W^O} \in R^{D\times D}$를 통해 원래 차원으로 투영된다.

$$\text{MultiHead}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Concat}(\text{head}_1,\dots,\text{head}_h)\mathbf{W^O}$$

$$\text{where head}_i = \text{Attention}\left(\mathbf{zW_i^Q}, \mathbf{zW_i^K}, \mathbf{zW_i^V}\right)$$

3. 학습 안정화

MHA의 출력을 받아 트랜스포머 인코더의 하나의 레이어를 완성해 보자

1. Add & Norm

모든 서브 레이어(MHA, FFN)의 출력은 입력과 더해지는 **잔차 연결(Residual Connection)**과 **계층 정규화(Layer Normalization)**를 거친다.

1. **잔차 연결 (Add):**

$$\mathbf{z}' = \mathbf{z} + \text{MultiHead}(\mathbf{z})$$

이는 입력 $\mathbf{z}$의 원천 정보가 깊은 네트워크를 통과하면서 소실되는 것을 방지한다 (Vanishing Gradient 방지).

2. **계층 정규화 (Norm):**

$$\mathbf{z}'' = \text{LayerNorm}(\mathbf{z}')$$

이는 배치 크기가 가변적인 NLP/Vision 환경에서 학습을 안정화시키기 위해 각 토큰 벡터의 특성(D) 차원을 기준으로 정규화를 수행한다.

3. 학습 안정화

MHA의 출력을 받아 트랜스포머 인코더의 하나의 레이어를 완성해 보자

2. FFN

정규화된 출력 $\mathbf{z}''$는 FFN을 통과하여 비선형적으로 정보를 정제.
두 개의 Dense Layer로 구성된다

$$\mathrm{FFN}(\mathbf{z}'') = \mathrm{ReLU}(\mathbf{z}''\mathbf{W_1} + \mathbf{b_1})\mathbf{W_2} + \mathbf{b_2}$$

1. **확장 (Expand) :** 입력 차원 D가 4D로 확장된다 (예: $512 \rightarrow 2048$).
2. **축소 (Contract) :** 확장된 차원이 다시 D로 축소된다.

FFN 역시 **잔차 연결**과 **계층 정규화**를 거치며, L개의 토큰 각각에 대해 **독립적으로** 적용된다.

2. Vit 수식으로 열어보기

4. 최종 출력

K개의 인코더 레이어를 거친 후, 최종 출력 $\mathbf{z_K} \in R^{(L+\mathbb{1})\times D}$에서 CLS 토큰 위치의 벡터($\mathbf{z_K^0}$)만 추출

$\mathbf{z_K^0}$ 는 최종 분류를 위한 MLP Head를 통과하여 이미지 클래스에 대한 확률 분포를 출력

2. Vit 수식으로 열어보기

+ 왜 Vit가 CNN보다 우월할까

전역적 문맥 파악:
MHA를 통해 이미지의 모든 영역 간의 관계를 한 번에 직접 파악하여, CNN의 지역적 필터링 제약을 제거한다.
병렬화 (Speed):

모든 계산이 행렬 곱셈 기반, GPU의 병렬 처리 능력에 최적화되어, CNN보다 훨씬 빠른 학습이 가능하다.

확장성 (Scaling Law):
모델 크기와 데이터 크기를 늘릴수록 성능이 지속적으로 향상되는 트랜스포머의 근본적인 장점을 계승했다.

감사합니다