LLaMA: Open and Efficient Foundation Language Models.

생성일	@2023년 11월 13일 오전 9:20
태그	NLP deep learning 논문리뷰

Abstract

- 1조의 vaca size, publicly available dataset, 65B(gpt-3:175B)의 모델 사이즈로 state-of-art의 성능을 달성함
- 모든 모델 공개함
    - 이후에 해당 코드 참조하여 다양한 community code와 응용이 나옴.
    - LLaMA2 공식버젼 (commercial use 가능함)
        - https://huggingface.co/meta-llama/Llama-2-70b-hf
    - pytorch-lightning 구현
        - https://github.com/Lightning-AI/lit-llama

Introduction

- 이전의 연구들에 의해 모델을 Language model을 scaling up하면 small sample이나 약간의 자연어 형태의 prompt 명령으로 새로운 task를 수행할 수 있음을 발견함
- 이후 계속해서 모델을 scaling up하여 성능향상을 보는 연구들이 나옴
- 하지만 Hoffmann et al. (2022)이 오히려 smaller model에서 더많은 data로 best performance 얻을 수 있음을 보임
    - 해당 논문에서 LLM 모델들이 under-trained 되었으며 그 원인이 **model size와 함께 torken의 양 (vocabulary size)를 함께 늘리지 않았기 때문**이라고 주장한다.
    - 저자들은 model size가 두배 늘어날 때, torken 개수도 두배 늘어나야한다고 한다.

- 사실 model size가 줄어들면 **inference budget**이 주는 게 이점이다.
    - 그래서 **더 작은 model size에서 voca size를 늘려서** inference budget 관점에서 optimum을 찾아보겠다가 focus

Pre-training data

- publicly available dataset만 이용하였음
- 데이터셋마다 특성 고려하여 filtering과 deduplication 작업 진행함
- 다양한 common crawl dataset을 이용하는 것이 성능향상에 더 좋았음
    - 때문에 English CommonCrawl, C4를 같이 사용함

- Common
    - English CommonCrawl [67%]
    - C4 [15%]

- Code
    - Github [4.5%]

- 전문 지식
    - Wikipedia [4.5%]
    - Gutenberg and Books3 [4.5%]
    - ArXiv [2.5%]
    - Stack Exchange [2%]

- tokenizer로는 ‘BPE(byte- pair encoding)’을 사용함
    - 결과적으로 전체 말뭉치에서 1.4T(1.4조)의 토큰이 생성됨

Architecture

- 기본적으로 transformer 구조에서 아래의 모델들에서 영감받은 부분들을 추가함
    - [GPT3, PaLM, GPTNeo]

- **Pre-normalization [GPT3]**

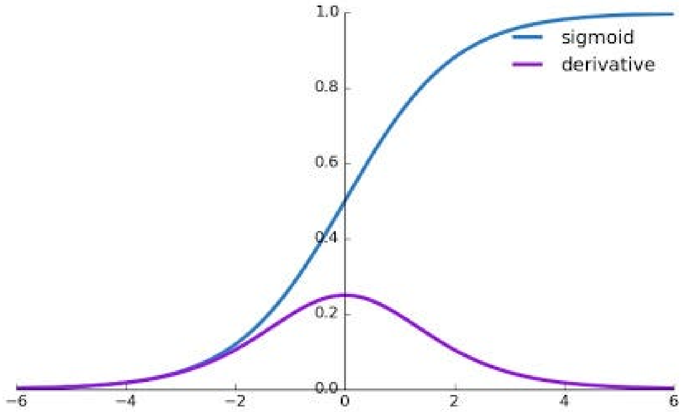
- learning의 안전성을 위해서 **RMSNorm**을 이용

$$RMSNorm:$$
$$a = Wx$$
$$RMS(a) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}$$
$$\overline{a_i} = \frac{a_i}{RMS(a)}$$

▼ **RMSNorm?**

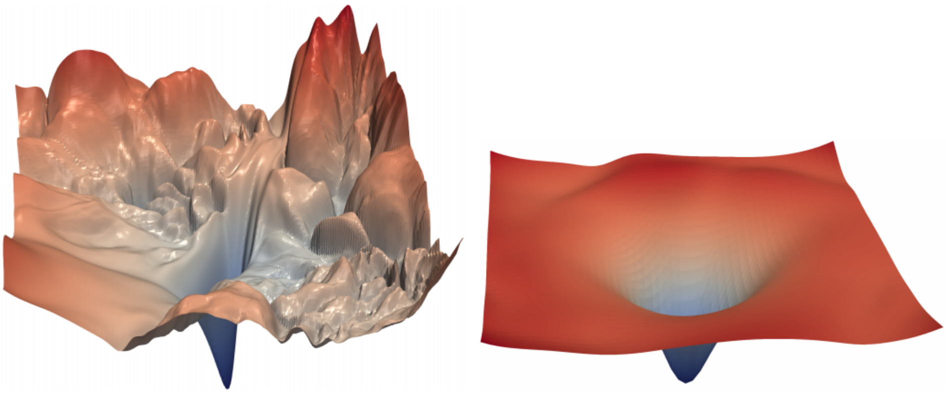
▼ batchnorm?

- 먼저 batchnorm에 대해 알아보면 input value를 안전화 시키기위해
  - 배치단위로 **(x-mean) / std의 변환**을 하고 (0 centered & unit variance)
  - 이는 인풋값을 sigmoid의 **미분값이 최대가 되는 구간에 인풋을 집중**시키는 효과를 볼 수 있음
  - **Affine 변환을 추가적**으로 시행함 (x_normW + b)
    - **이유는 0 근처에서 sigmoid가 선형에 가까워서 Non-linearity를 잃게 되므로**



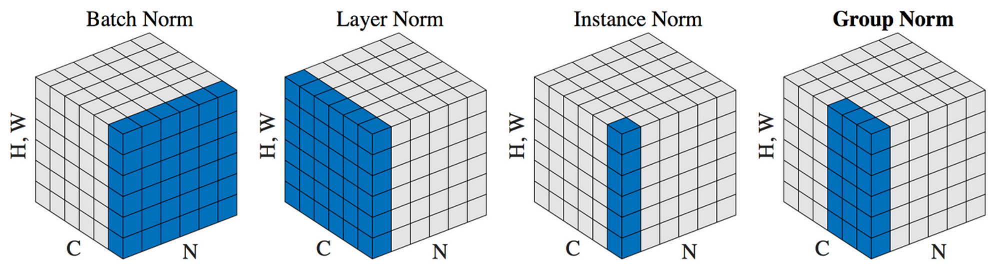
- 실험적으로 batchnorm은 **아래의 효과**가 있는 것으로 관찰됨
  - 필요한 **regularization**의 강도를 감소시킴
    - (batchnorm 사용만으로도 어느정도 일반화 성능을 가져가므로)
  - 더 큰 learning rate을 사용할 수 있게 됨 → 결과적으로 **빠른 학습**
- 과거에는 ‘internal covariant shifting’을 해소해줌으로서 효과를 보인다고 했으나

- 현재는 **optimize landscape를 smoothing함**으로써 얻는 이득으로 밝혀짐
  - (residual connection 또한 같은 원리로 효과를 나타냄)

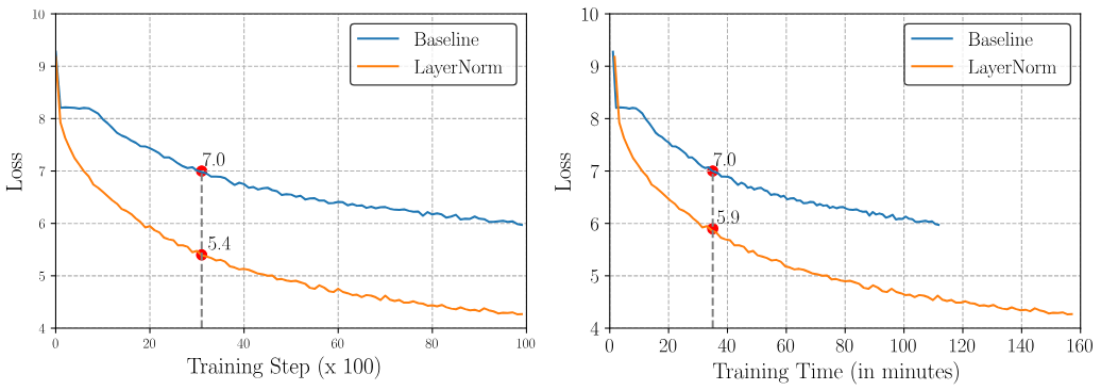


(a) without skip connections (b) with skip connections

- batchnorm에는 normalization 방법(평균, 분산을 구하는 axis)에 따라 다양한 변주가 존재함



- https://github.com/bzhangGo/rmsnorm: 자세한 내용 확인 가능
- layer normalization을 단순화한 normalization 방법론
  - root mean square 값을 기준으로 한 neuron에 들어가는 인풋의합을 제안하는 방식으로 작동
  - layer normalization 대비 계산 복잡도가 감소 → 결과적으로 **속도에서 이득**



- transformer를 구성하는 sub-layer의 인풋을 모두 normalizing함
    - (output에 normalizing 하는 대신에)

```
# lit-lamma 구현체 예시
## att input과 mlp input에 RMSNorm을 해준다.
class Block(nn.Module):
    def __init__(self, config: LLaMAConfig) -> None:
        super().__init__()
        self.rms_1 = RMSNorm(config.n_embd)
        self.attn = CausalSelfAttention(config)
        self.rms_2 = RMSNorm(config.n_embd)
        self.mlp = MLP(config)

    def forward(
        self,
        x: torch.Tensor,
        rope: RoPECache,
        mask: MaskCache,
        max_seq_length: int,
        input_pos: Optional[torch.Tensor] = None,
        kv_cache: Optional[KVCache] = None,
    ) -> Tuple[torch.Tensor, Optional[KVCache]]:
        h, new_kv_cache = self.attn(self.rms_1(x), rope, mask, max_seq_length, input_pos, kv_cache)
        x = x + h
        x = x + self.mlp(self.rms_2(x))
        return x, new_kv_cache
```

- **SwiGLU activation function [PaLM]**
    - ReLU를 대신하여 Shazeer (2020)에 의해 소개된 **SwiGLU**라는 활성 함수를 이용함
    - 성능향상에 도움이 되었다고 함
  - **SwiGLU?**
    - 기존에 존재하던 swish 활성함수와 GLU 활성 함수를 결합하여 만든 컨셉

$$SwiGLU(x) = swish_{\beta}(xW + b) \otimes (xV + c)$$

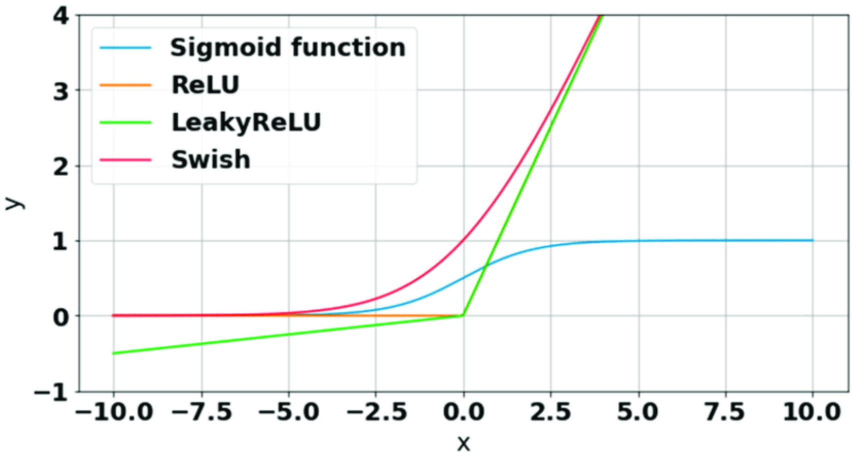
    - SwiGLU는 GLU에서 non-linear ftn로 Sigmoid 대신 Swish 함수를 적용한 변주이다.

- Shazeer (2020)에 의해 실험적으로 두 가지를 결합한 활성함수를 이용하는 것이 언어모델에서 가장 좋은 성능을 보여줌이 확인되었다.
  - Note) ReLU 대비 trainable param 때문에 capa가 늘어서 좋은게 아닌가 할 수 있지만 ReLU의 dimension을 추가해서 compute- equivalent하게 실험하더라도 성능향상이 있었다고 함

▼ **Swish 활성함수**

$$swish(x) = x\sigma(\beta x)$$
$$where\ \beta\ is\ trainable\ param$$

- ReLU 보다 smoother된 형태의 활성함수로 일반적으로 더 깊은 신경망일수록 ReLU 대비 더 잘 작동함



- **smoother한 특성**으로 인해 더 **optimize가 잘되고 convergence가 빠르다.**
  - B parameter가 커짐에 따라 indicator ftn에 가까워진다.
  - 반대의 경우 linear ftn에 가까워진다
  - 즉, **linear ftn과 indicator ftn의 interpolation**으로 해석될 수 있다
    - (linear ftn 덕분에 smoother하게 되서 gradient기반의 최적화에 이점을 가지게 된다.)
  - ref) Shazeer (2020)

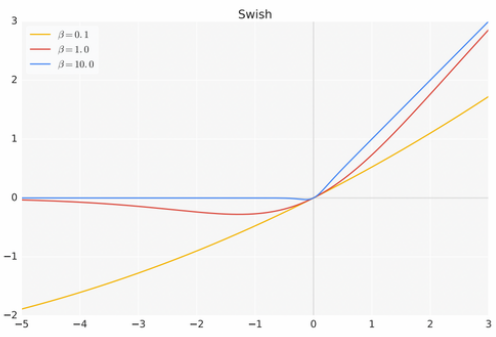


Figure 4: The Swish activation function.

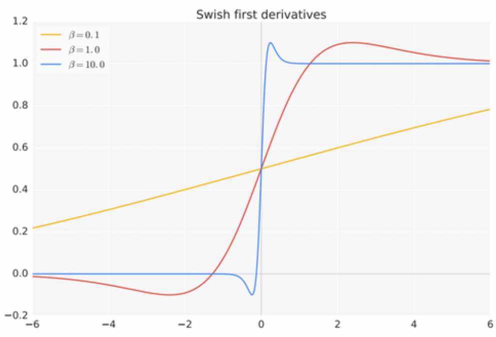


Figure 5: First derivatives of Swish.

- 여기서 재밌는 포인트는 **x<0의 구간**에서 볼 수 있는 **non-monotonic한 구간**이 **성능에 중요한 영향**을 끼친다는 사실이다. 이는 아래 그림에서 간접적으로 확인할 수 있다.
- 아래 그림에서 처럼 B=1의 **Swish**를 이용해서 학습된 모델에서 대부분의 활성함수의 **입력값이 [-5, 0]의 구간에 몰려있는 것**을 알 수 있다.

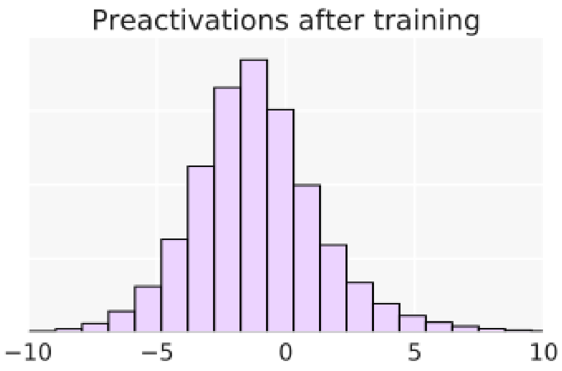


Figure 6: Preactivation distribution after training of Swish with $\beta = 1$ on ResNet-32.

▼ **GLU(Gated Linear Unit) 활성함수**

$$GLU(x) = \sigma(xW + b) \otimes (xV + c)$$

- Microsoft 연구팀에 의해서 제안된 활성함수
- trainable한 param으로 구성된 Linear function과 sigmoid로 이루어짐
- NLP분야에서 seq의 **long-term dependency를 잘 학습하기 위해서는 'gating mechanism'**이 중요함이 익히 알려져 있음(ex. LSTM)
    - 이러한 부분에서 착안하여 단순화된 GLU 활성함수를 고안함
- 이때 위의 식에서 sigmoid를 대신하여 **다양한 Non-linear를 적용함**으로서 **다양한 변주**가 생길 수 있음

- **Rotary Embeddings [GPTNeo]**

  ▼ **GPTNeo?**

  - EleutherAI라는 비영리 연구 단체에 의해 만들어진 GPT의 open source 버젼
  - 현재는 GPTNeoX라는 후속 버젼이 나와있고, 20 billion parameter size
  - 동일 사이즈의 GPT-3의 성능보다는 앞선다고 주장함

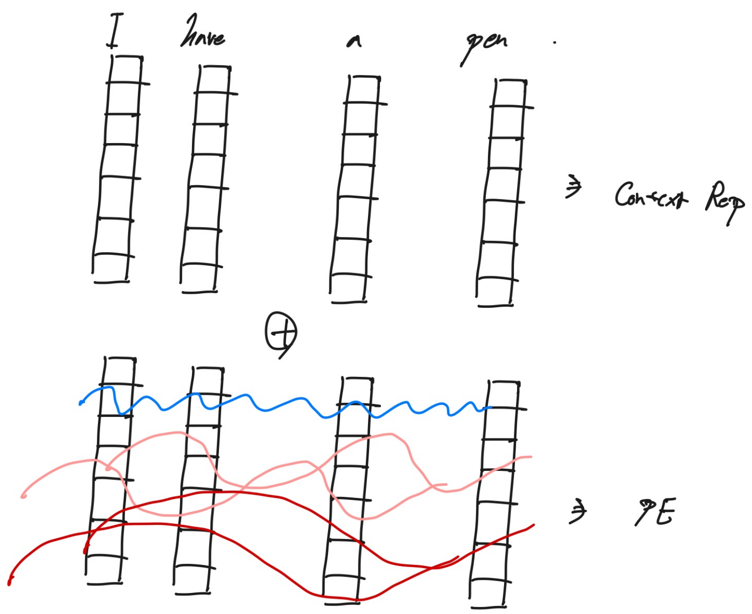
  - 언어모델의 대세는 **self-attention architecture**이지만, 이 구조의 문제는 position-agnostic한 구조이기 때문에 **word의 문장내 위치의 정보는 encoding할 수 가 없다**.
  - 때문에 주로 토큰의 relative(간혹 absolute)한 위치를 인코딩한 **"positional encoding" 값을** 따로 구하여(learned/not learned) context **representation**에 **더해주는 방식**으로 많이 연구가 진행됨
    - 다만 '**linear** self-attention architecture'으로 구해지는 **representation에** encoding 값을 **더하는 방식**은 적절하지 않음을 지적함

  - 따라서 **context representation에 rotation matrix**를 **곱하는 방식**을 제안함

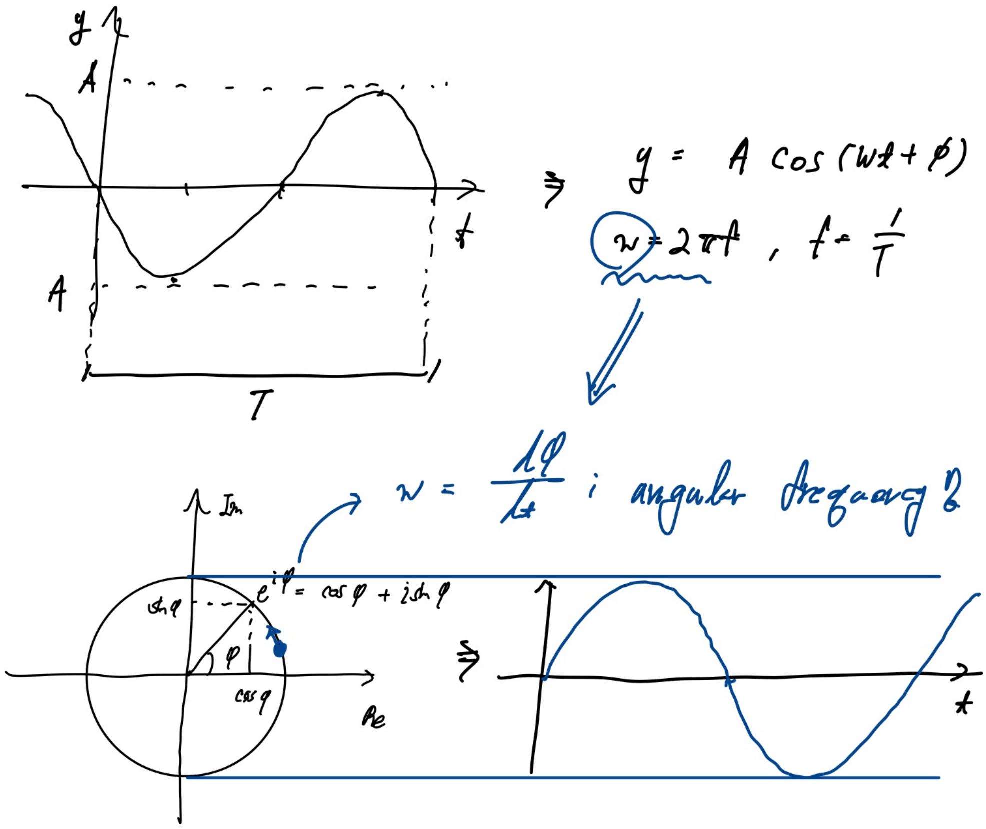
  ▼ **Absolute position embedding**

  - absolute position encoding하는 방식중 하나로는 **'attention is all you need'**에서 제시된 정현파를 이용하는 방식이 있음

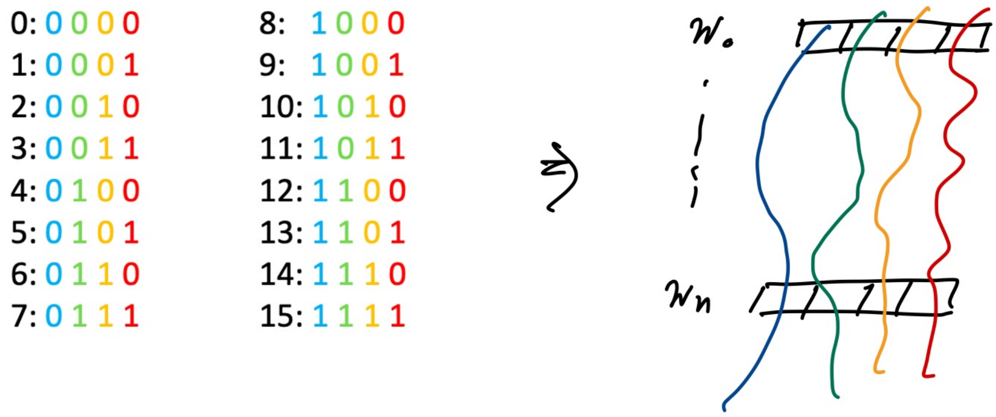
$$p_{t,2i} = sin(t/10000^{2i/d_{model}})$$
$$p_{t,2i+1} = cos(t/10000^{2i/d_{model}})$$



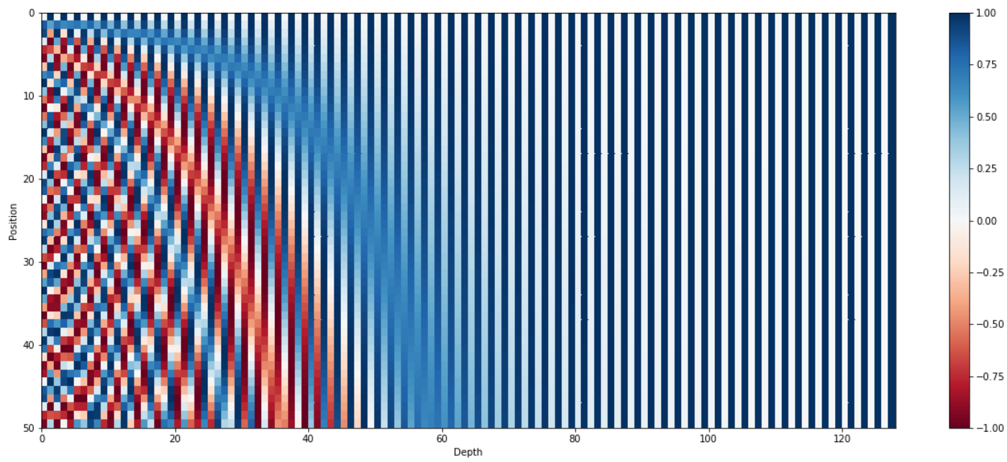
- 저자들이 encoding 함수에 필요한 조건은 다음과 같았다.
  - 각 time-step에 대해 unique한 encoding 값을 출력해야한다.
  - 어떠한 문장에서도 어떤 두 time-step에 대한 임베딩 간의 거리가 같다.
    - $dist(t_i, t_j) = dist(s_i, s_j)$
  - 학습 중에 본 길이 보다 긴 문장도 처리 가능해야했다.(learned embedding이 힘든 부분)
  - 결정적인 함수 이어야 했다. (rather than stochastic)
- 위에 제시된 **정현파를 이용**한 함수를 고안함으로써 위의 **조건을 모두 만족**하는 encoding 방법을 찾아냄
  - ▼ 정현파??



- 이 방법론을 직관적으로 이해하자면 아래와 같은 **숫자를 이진법으로 나타내는 것**과 유사하다.



- 이진법 표현에서 각 자리수는 각각 **특정한 주기**를 가지는 것을 볼 수 있다.
- 이것을 discrete에서 **continuous**하게 **확장시킨 개념이 정현파를 이용한 방식이다.**
- 정현파의 **주파수를 각 depth(embedding dim에서의 index)에 따라 다르게 함**으로서 이를 구현 **이진법 표현과 비슷하게 적용**한 것이다.



▼ **Relative position embedding**

- 일반적인 **Query, Key, Value**를 이용한 attention setting에서 Absolute PE와 Relative PE는 아래와 같은 형태로 **일반화** 할 수 있음

$\forall$ m-th, n-th position,

$$\begin{cases} q_m = f_q(x_m, m) \\ k_n = f_k(x_n, n) \\ v_n = f_v(x_n, n) \end{cases}$$

① Absolute

$$\begin{cases} f_q(x_i, i) = W_q(x_i + p_i) \\ f_k(x_i, i) = W_k(x_i + p_i) \\ f_v(x_i, i) = W_v(x_i + p_i) \end{cases}$$

② Relative

$$\begin{cases} f_q(x_m) = W_q x_m \\ f_k(x_n) = W_k(x_n + \tilde{p}_r^k) \\ f_v(x_n) = W_v(x_n + \tilde{p}_r^v) \end{cases}$$

$\Rightarrow$ $\tilde{p}_r^k, \tilde{p}_r^v$ is trainable

'relative' PE

- 여기서 $q^T k$를 활용하여 구해지는 **attention weight에 집중**한다면 **value에 관한 텀은 무시**될 수 있음, 따라서 $q^T k$**를 decompose하고** 필요없는 term을 줄이면 아래와 같은 형태가 된다.

$\forall$ $q_m = W_q(x_m + p_m)$, $k_n = W_k(x_n + p_n)$

$$q_m^T k_n = (x_m + p_m)^T W_q^T W_k (x_n + p_n)$$

$$= (x_m^T W_q^T + p_m^T W_q^T)(W_k x_n + W_k p_n)$$

$$= x_m^T W_q^T W_k x_n + x_m^T W_q^T W_k p_n + p_m^T W_q^T W_k x_n \; (+ p_m^T W_q^T W_k p_n)$$

do not need

↳ replace p_n and p_m to p_{m-n}

$$\Rightarrow \therefore q_m^T k_n = x_m^T W_q^T W_k x_n + x_m^T W_q^T W_k \tilde{p}_{m-n} + \tilde{p}_{m-n}^T W_q^T W_k x_n$$

↳ relative PE for (m-th − n-th)

$\Rightarrow$ trainable

- 저자들은 **Query, Key, Value setting**에서 1) **relative PE**이고 2) 더하기는 방식이 아니라 context rep를 **rotation하는 방식**을 제한함

- 결론부터 말하자면 저자들이 제안하는 방식은 **단어의 벡터표현을 문장내 위치에 따라 회전변환을 시켜주는 방식**이다. 이를 표현한 그림이 아래의 그림이다.

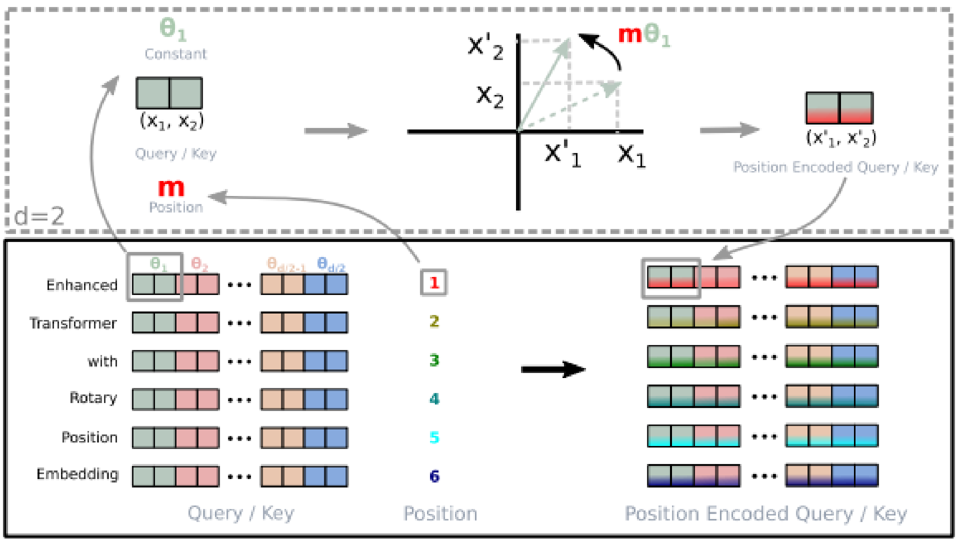


Figure 1: Implementation of Rotary Position Embedding(RoPE).

- 저자들은 아래와 같이 $q_m^T k_n$를 아래와 같은 context rep와 relative position을 함수, $g()$를 찾는 문제로 바꾸었다.

$$\langle f_q(x_m, m), f_k(x_n, n)\rangle = g(x_m, x_n, m-n)$$

- positional encoding을 포함한 **q, k 함수를** $f(x, m) = (Wx)e^{im\theta}$**의 꼴로 정의**하면 **내적 값**은 아래와 같이 되고 **(m-n)으로 표현되는 상대적인 위치는 각도**, **context rep의 내적값은 길이**가 된다.

$$\begin{aligned} g(x_m, x_n, m-n) &= (W_q x_m)(W_k x_n)^* e^{i(m-n)\theta} \\ &= qk^* e^{i(m-n)\theta} \end{aligned}$$

- 편의를 위해 이를 이차원에서 확인해보면 아래와 같이 볼 수 있다.

$$\begin{cases} f_q(x_m, m) = (W_q x_m) e^{im\theta} \\ \text{and} \\ f_k(x_n, n) = (W_k x_n) e^{in\theta}, \quad \theta \text{ is non-zero constant} \end{cases}$$

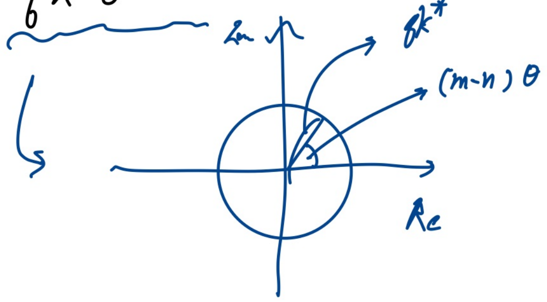
$$\Rightarrow g(x_m, x_n, m-n)$$

$$= \langle f_q(x_m, m), f_k(x_n, n) \rangle$$

$$= W_q x_m e^{im\theta} \cdot W_k^* x_n^* e^{-in\theta}$$

$$= (W_q x_m W_k^* x_n^*) e^{(m-n)i\theta}$$

$$\Rightarrow q k^* e^{(m-n)i\theta}$$



- 최종적으로 **q, k를 행렬로 표현**하면 아래와 같이 **입력 X를 affine변환하고** $m\theta$**만큼 회전시킨 벡터**가 된다.

$$x = [x_1, x_2]$$

$$f(x, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W_{00} & W_{01} \\ W_{10} & W_{11} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

given constant

position

⇒ weight of query + key

- 한계점
  - 현재까지 알려진바에 의하면 **context 길어질수록 perplexity가 높아지는 문제**가 있음

▼ **perplexity?**

- test set의 내에서 n개의 단어로 구성된 문장의 확률(=$P(w_1, w_2, ..., w_n)$)의 확률의 역수를 normalizing한 값

$$PP(W) = \sqrt[n]{\frac{1}{P(w_1, w_2, ..., w_n)}}$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{1}{P(w_i|w_{i-1}, ..., w_1)}}$$

◦ 이후 seq length에 따라 dynamic하게 RPoE scaling하는 방식이 한 커뮤니티(Reddit) 유저에 의해 제안됨 (링크)