

< 이미지 고주파 영역 학습 능력 부족 문제 해결 >

- 이미지에서의 주파수는 화소 밝기의 변화율을 의미한다. (영상에서도 동일하게 적용되는지는 잘 모르겠다.)

고주파 : 밝기가 빨리 바뀌는 영역을 의미

이미지 < ex). 주변 영역과 색 차이 많이 나는 경계 부분, 객체의 윤곽이나 모서리 부분 등

저주파 : 밝기가 거의 바뀌지 않는 영역을 의미

ex). 주변 영역과 색 차이가 거의 없는 배경이나 객체의 내부 등

5 차원 데이터 (x, y, z, θ, ψ)를 별도의 가공 없이 입력 데이터로 사용하면

데이터의 정보량이 부족해서 이미지의 고주파 영역에 대한 학습 능력이 떨어지게 된다.

따라서, Positional encoding으로 입력 데이터를 High Dimension으로 변환해야 한다.

NeRF에서 사용한 Positional encoding은 Transformer에서 사용하는 Positional encoding과 동작 원리는 동일하고, 사용하는 목적만 다르다.

< Positional encoding >

E_0	p_0	E_1	p_1	E_2	p_2	E_3	p_3
0.19		0.70		0.34		0.69	
-0.47	⊕	-0.65	⊕	0.87	⊕	0.79	⊕
-0.77		0.11		-0.39		-0.25	
0.59		0.04		-0.91		0.44	

E_i < Transformer : 한 시퀀스 내의 각각의 단어 벡터를 표현

NeRF : 한 ray 위의 샘플링된 모든 점들의 5D 입력 데이터 (x, y, z, θ, ψ)의 각 원소들을 표현
↳ 학습하기 않음!

P_i < Transformer : 단어 벡터에 대해서 위치 벡터 정보

NeRF : 5D를 고차원으로 변환하기 위해서 대해서 새로운 한 개의 차원 정보

↳ 학습하지 않음!

Transformer에서

Positional encoding을 적용하기 위한 조건 1.

this	is	my	car
0.19	0.01	0.70	0.13
-0.47	0.01	-0.65	0.31
-0.77	0.02	0.11	0.23
0.59	0.02	0.04	0.24

모든 위치 벡터는 시퀀스의 길이나 입력 데이터(단어)가

달라져도 항상 동일한 위치 벡터 정보를 가져야 된다.

- 최소를 0으로, 최대를 1로 하고 그 사이를 데이터 개수로

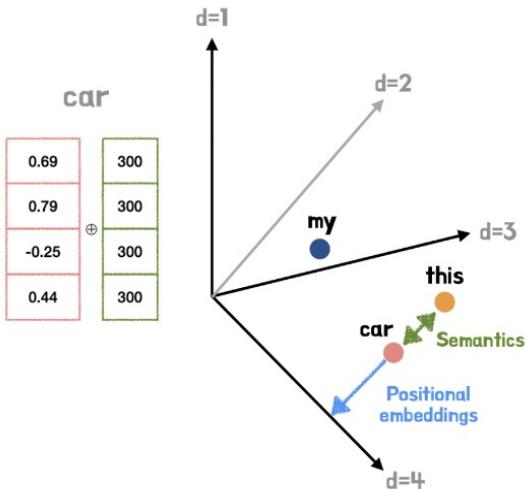
나누어서 위치 벡터를 할당하는 방법을 사용할 수 있다.



that	is	not	[PAD]
0.19	0.01	0.70	0.13
-0.41	0.01	-0.65	0.31
0.12	0.02	0.11	0.23
0.59	0.02	0.04	0.24

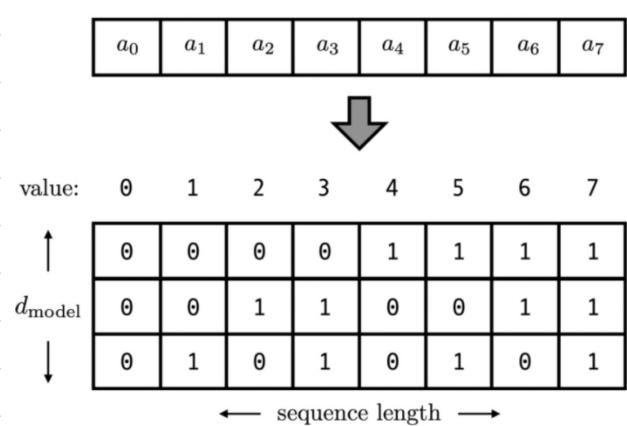
Transformer의
Positional encoding을 적용하기 위한 조건 2.

this	is	my	car
0.19	0.70	0.34	0.69
-0.47	-0.65	0.87	0.79
-0.77	0.11	-0.39	-0.25
0.59	0.04	-0.91	0.44
1	30	100	300
1	30	100	300
1	30	100	300
1	30	100	300



모든 위치 벡터 정보는 값이 너무 크면 입력 데이터 (단어) 간의 의미를 연관 지어서 유추하는 의미 정보 값이 상대적으로 작아지기 때문에 성능이 떨어진다.
- 위치 벡터를 인덱스가 1 증가할 때마다 선형적으로 1씩 증가시킨 방법을 사용할 수 있는 이유

Transformer의
Positional encoding을 적용하기 위한 조건 3.



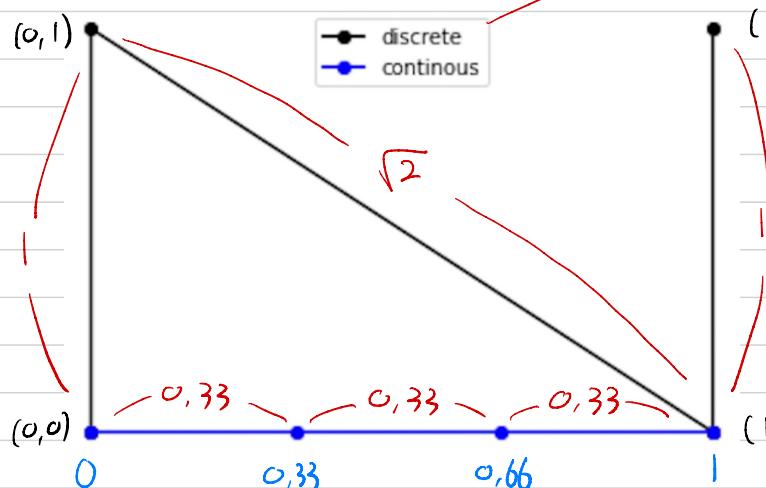
위치 벡터를 이진수로 할당하면 조건 1, 2는 만족하지만
위치 벡터 내부에서 벡터 간의 간격이 동일하다면
토큰들의 위치와는 관계 없이 위치 벡터들의 거리도
동일해야 된다는 조건을 만족하지 못한다.
— 위치 벡터를 이진수로 사용할 수 없는 이유

discrete: 이진수로 위치 벡터 할당

따라서, discrete한 특성이 있음

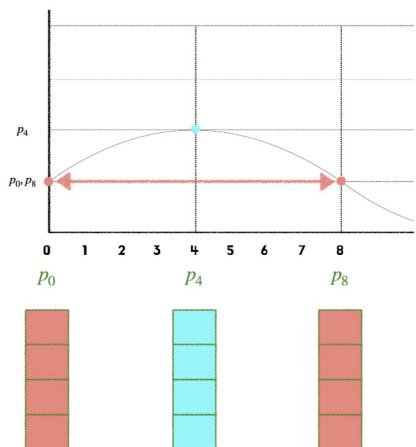
continuous: 최소, 최대를 정해두고 그 사이를
시퀀스의 길이 N 으로 나누어서 (토큰 개수 N)
위치 벡터 할당
따라서, continuous한 특성이 있음

discrete 위치 벡터는 벡터 간 거리가 모두 1만큼
떨어져 있고 실질했지만 실제 거리는 1 또는 $\sqrt{2}$ 가
나오기 때문에 조건 3을 만족하지 못함
반면, continuous 위치 벡터는 다른 토큰들은 만족하지
못하지만 조건 3은 만족할 수 있음

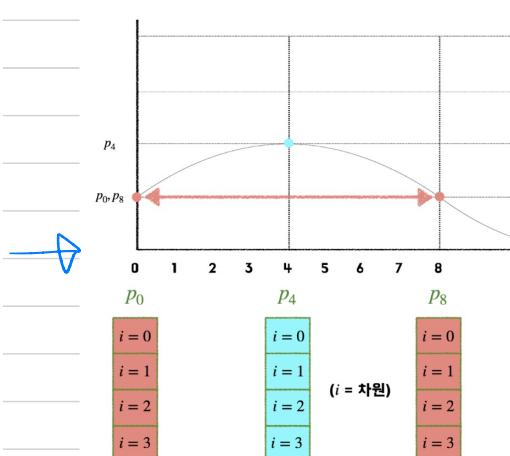


〈sin, cos 함수를 사용함에 따라 해결해야 되는 문제들〉

- Sigmoid도 범위가 정해져 있지만 구간이 작기 때문에 시퀀스가 길어지면 벡터 간 의차가 약해진다.
→ sin, cos은 sigmoid에 비해 구간이 넓어서 문제가 해결됨.
- 정해진 범위를 반복하는 주기 함수이기 때문에 한 주기를 넘어설 때 시퀀스가 길어지면 한 시퀀스 내에서 동일한 값을 가진 위치 벡터가 나오게 된다.
 1. 위치 벡터도 입력 데이터(단어) 벡터와 동일한 차원을 가진 벡터 형식이라는
Positional encoding의 특징 덕분에 위치 벡터의 각 차원마다 주기가 서로 다른 sin, cos 함수를
사용하여 문제 해결 됨
→ 진폭 범위는 $1 \sim 1$ 사이로 사용할 것에 때문
 2. $A \cdot \sin N$ 에서 A 는 진폭에 영향을 주기 때문에 사용하지 않고, N 이 커지면 진동 주기가 빨라져서 한 시퀀스 내에
동일한 값이 나올 수 있으므로 N 을 충분히 작은 값으로 사용해서 진동주기를 늘려 문제 해결
→ Transformer에서는 $1/10000$ 을 사용



〈문제 발생 상황〉



〈문제 해결 방법〉



° 범위가 $-1 \sim 1$ 사이에 존재하는 sin, cos 사용한다.

→ 논문 Attention is all you need.

<Transformer의 Positional encoding 원리>

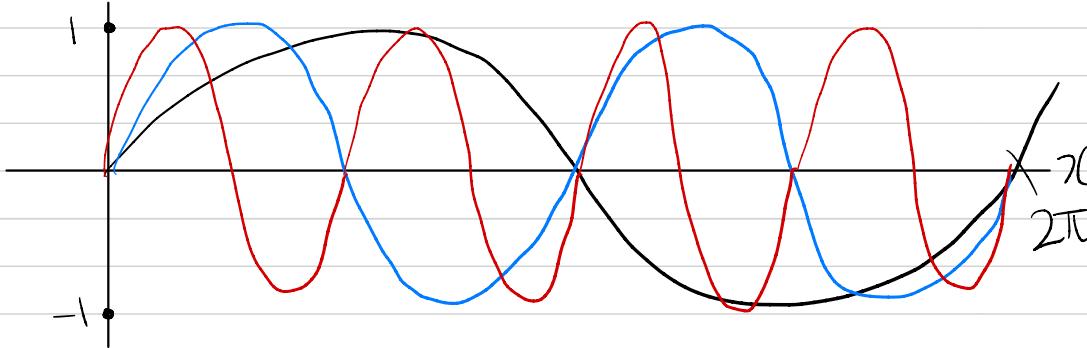
Attention is all you need 논문에서 제시하는 Positional encoding 수식

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

pos : position

i : dimension



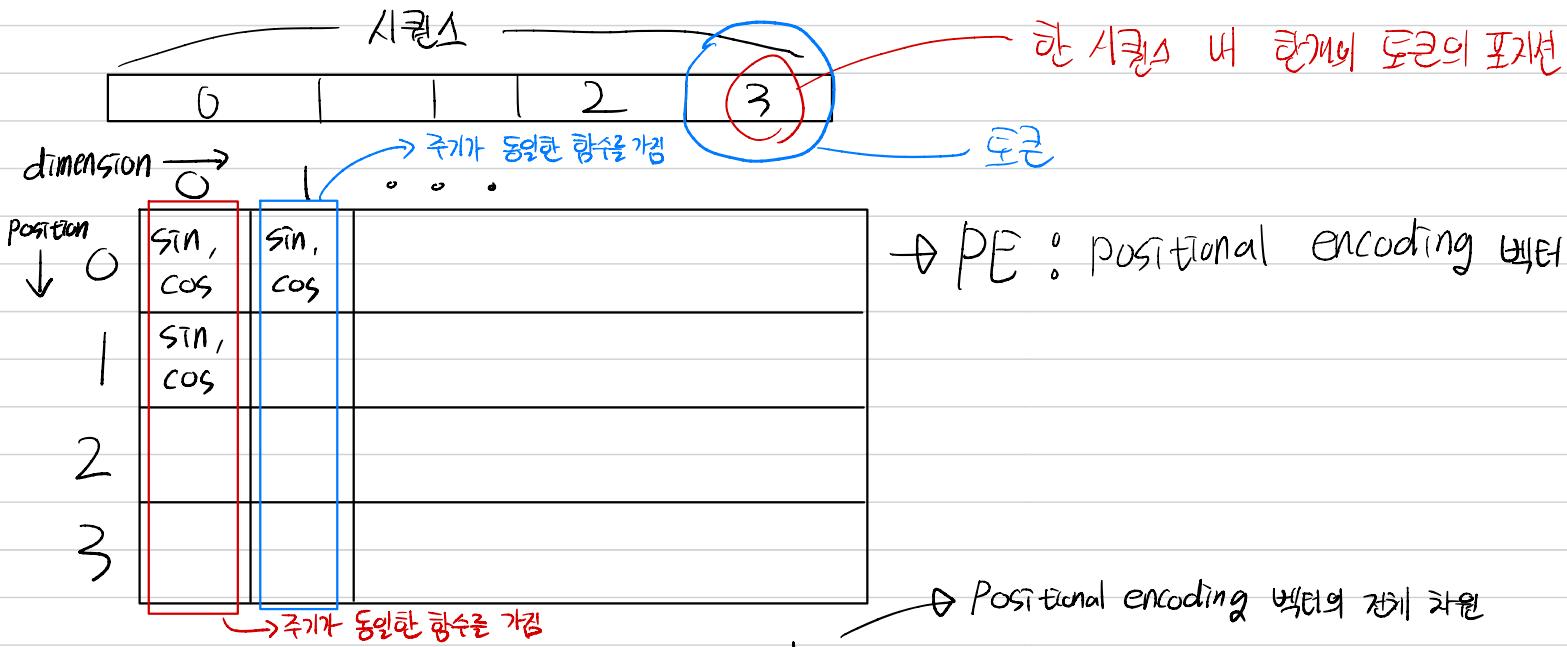
$\sin 4\pi$ 주기 $\frac{\pi}{2}$ ($\frac{2\pi}{4}$)

$\sin 2\pi$ 주기 π ($\frac{2\pi}{2}$)

$\sin \pi$ 주기 2π ($\frac{2\pi}{1}$)

주기가 빠른 sine 함수를 기준으로 그 힘수보다 주기가 느린 모든 함수들이
절반 만큼 이동할 때마다 주기가 빠른 함수와 같은 값을 가진다.

= 반복을 기준으로 위치벡터가 동일한 값을 가진다. → 그래서 수식에서 볼 수 있듯이 $\frac{1}{10000}$ 처럼 매우 작은 수를
사용하여 힘수 주기를 조절하여 문제를 해결한다.



position : 1 , dimension : 2 차원 , d_{model} : 5 차원 일 때 예시

$$\sin(1/10000^{2x2/5}), \cos(1/10000^{2x2/5})$$

$$10000^{\frac{4}{5}} = (10^4)^{\frac{4}{5}} = 10^{\frac{16}{5}} = \sqrt[5]{10^{16}}$$

↳ 5제곱하여 10^{16} 이 나오는 수

$$10000^{\frac{4}{5}} = \sqrt[5]{10^{16}} = 1584.893192, (1584.893192)^5 = 10^{16}$$

$$\sin\left(1/10000^{\frac{2 \times 0/5}{1}}\right), \cos\left(1/10000^{\frac{2 \times 0/5}{1}}\right)$$

$$= \sin\left(1/10000^{\frac{1}{5}}\right), \cos\left(1/10000^{\frac{1}{5}}\right) \Rightarrow \sin(1/6,3), \cos(1/6,3)$$

$$\sin\left(1/10000^{\frac{6}{5}}\right)^{\circ}, \cos\left(1/10000^{\frac{8}{5}}\right)^{\circ} \Rightarrow \sin(1/2511886), \cos(1/2511886)$$

함수 주기를 $\pi/2^{\top}$ 으로 조절하면 시퀀스 길이가 길어졌을 때 주기가 절반 만큼 이동한 지점에서 동일한 위치에서 값을 가질 수 있다.

주기를 꼭 $\pi/2^{\top}$ 간격으로 조절할 필요는 없다.

따라서, 함수 주기를 매우 조금씩만 증가 또는 감소 시켜도 문제를 해결 할 수 있다.

\sin, \cos 함수의 그래프에서

볼 수 있듯이

해당 함수 특성상 주기를 π 를 기준으로 이분했을 때
필연적으로 발생하는 문제다.

<NeRF에서 Positional encoding을 적용하는 과정>

$$\gamma(p) = (\underbrace{\sin(2^0 \pi p)}_{\text{주기 } \frac{1}{\pi}}, \underbrace{\cos(2^0 \pi p)}_{\text{주기 } \frac{1}{\pi}}, \dots, \underbrace{\sin(2^{L-1} \pi p)}_{\text{주기 } \frac{1}{\pi}}, \underbrace{\cos(2^{L-1} \pi p)}_{\text{주기 } \frac{1}{\pi}})$$

$L = \begin{cases} 10, \gamma(x) & X \text{는 샘플링된 점의 좌표 } (x, y, z) \\ 4, \gamma(d) & d \text{는 샘플링된 점의 방향 좌표 } (x', y', z') \end{cases}$

↳ (θ, ψ) 의 각각 좌표

Cartesian Vector

각각 좌표계 → 구면 좌표계

각각 좌표가 (x, y, z) 일 때

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\theta = \arccos \frac{z}{r} \rightarrow \text{역코사인 } \cos^{-1} \text{를 계산}$$

$$\psi = \arctan \frac{y}{x} \rightarrow \text{역탄젠트 } \tan^{-1} \frac{y}{x} \text{ 계산}$$

구면 좌표계 → 각각 좌표계

구면 좌표가 (r, θ, ψ) 일 때

r 은 원점으로부터의 거리

$= (x, y, z, \theta, \psi)$ 에서

원점에서부터 점 (x, y, z) 까지의 거리

$$x = r \sin \theta \cos \psi$$

$$y = r \sin \theta \sin \psi$$

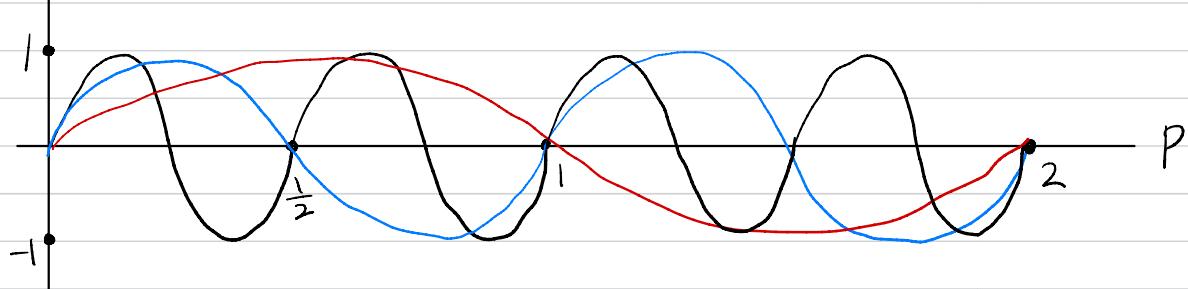
$$z = r \cos \theta$$

$\gamma(p)$, P 는 $X = (x, y, z)$ 와 $d = (x', y', z')$ 가 입력으로 들어간다.

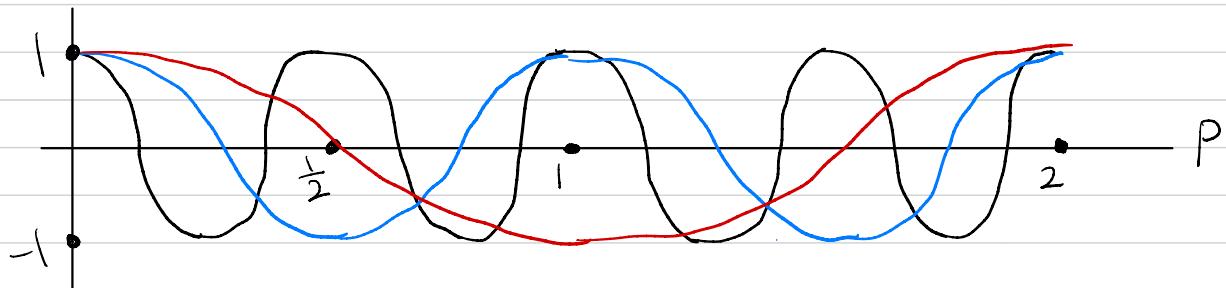
입력으로 들어온 X 와 d 의 원소들은 \sin 과 \cos 을 통과하면서 구간 $[-1, 1]$ 사이로 Normalization 된다.

$$\langle \sin(2\pi p), \cos(2\pi p) \text{ 제곱} \rangle$$

$$(\sin(4\pi)x_p, \cos(4\pi)x_p)$$



\Rightarrow 함수 주기에서 P를
생략하고 그걸 몇 때



⇒ Transformer는 입력 시퀀스 내의 토큰 순서를 표현하기 위해서 Positional encoding을 사용한다. \sin , \cos 함수를 돌면서 입력으로 들어온 토큰들에게 함수값을 할당해주면서 주기를 반복한다.

순서 즉, 위치는 중복되는 것이 불가능하기 때문에 토큰의 위치(순서) 값으로 사용하는 \sin , \cos 함수에 최대한 중복되는 값이 나오지 않도록 주기를 $\frac{1}{10000}$ 처럼 매우 작은 값을 사용하여 흡수 간 주기 차이를 미세하게 조절했다.

→ \sin 과 \cos 으로 이루어진 Positional encoding 함수를 사전에 계산해놓고 끝날 때까지 계속 사용한다.
이전 토큰 개수 만큼 이동한 주기 이후부터 이어서 진행한다.

반면에, NeRF는 단순히 영역 데이터를 $[-1, 1]$ 사이에서 고차원의 데이터로 매핑하는 목적이다.

→ 수식에서 P를 의미함

2π 를 주기로 가진 \sin, \cos 함수에서 입력 원소의 값 만큼 곱한 후에 해당하는 함수 값을 얻는다, 따라서, 같은 입력 원소에 대해서는 중복이 발생할 수 없다.

또한, 서로 다른 퍼셀에 대해서 입력 원소의 값이 서로 동일하여 동일한 함수 값이 나와도 두 퍼셀은 동일하지 않기 때문에 무관하다.

= Transformer에서 한 시퀀스 내의 서로 다른 토큰들에 대해서 동일한 함수 값이 중복된 경우

↳ 확실하지 않음 ▶

∴ 한개 차원의 데이터를 고차원으로 늘려서 데이터가 가리키고 있는 해당 부분을 더 자세하게 확대하는 효과를 준다.
↳ 확장하기 않음!

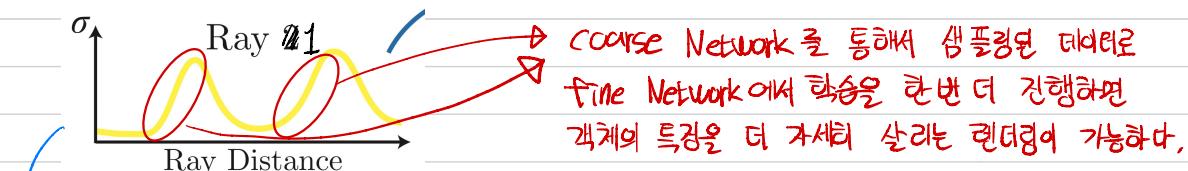
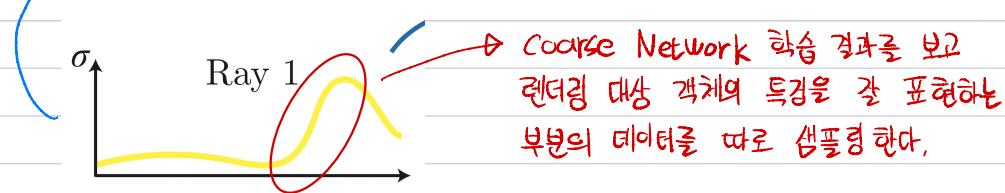
< Hierarchical Volume Sampling >

학습 효율을 높이기 위해서 Hierarchical Volume Sampling 기법을 사용한다.

Network

- coarse Network : 전체 데이터에 대해서 우선적으로 학습을 진행한다,
- fine Network : coarse Network 학습 결과에서 학습 효율을 높일 수 있는 구간들을 샘플링하여, 샘플링된 데이터들로 학습을 한번 더 진행한다.

→ coarse Network 학습



→ fine Network 학습

⇒ 실제 렌더링에 사용할 점들은 coarse Network에서 샘플링된 N_c 개와 fine Network에서 샘플링된 N_f 개의 점들로 구성하여 렌더링에 사용한다.

$$\hat{C}(r) \rightarrow N_c + N_f \text{ 로 렌더링 계산}$$

< 한 개의 ray에서 coarse Network를 통해 r, g, b 값인 $\hat{C}_c(r)$ 을 계산하는 방법 >

$$\hat{C}_c(r) = \sum_{i=1}^{N_c} w_i c_i , \quad w_i = T_i (1 - \exp(-\epsilon_i s_i)) , \quad \hat{w}_i = \frac{w_i}{\sum_{j=1}^{N_c} w_j}$$

N_c 는 ray의 전체 점들 중에서 층화표집으로 샘플링된 점들을 의미함.

→ 잘 모르겠음.

(* N_c 에서 N_f 를 샘플링 한 것으면
 N_f 는 N_c 의 점들을 포함한 상태로 학습을 해야되나?)

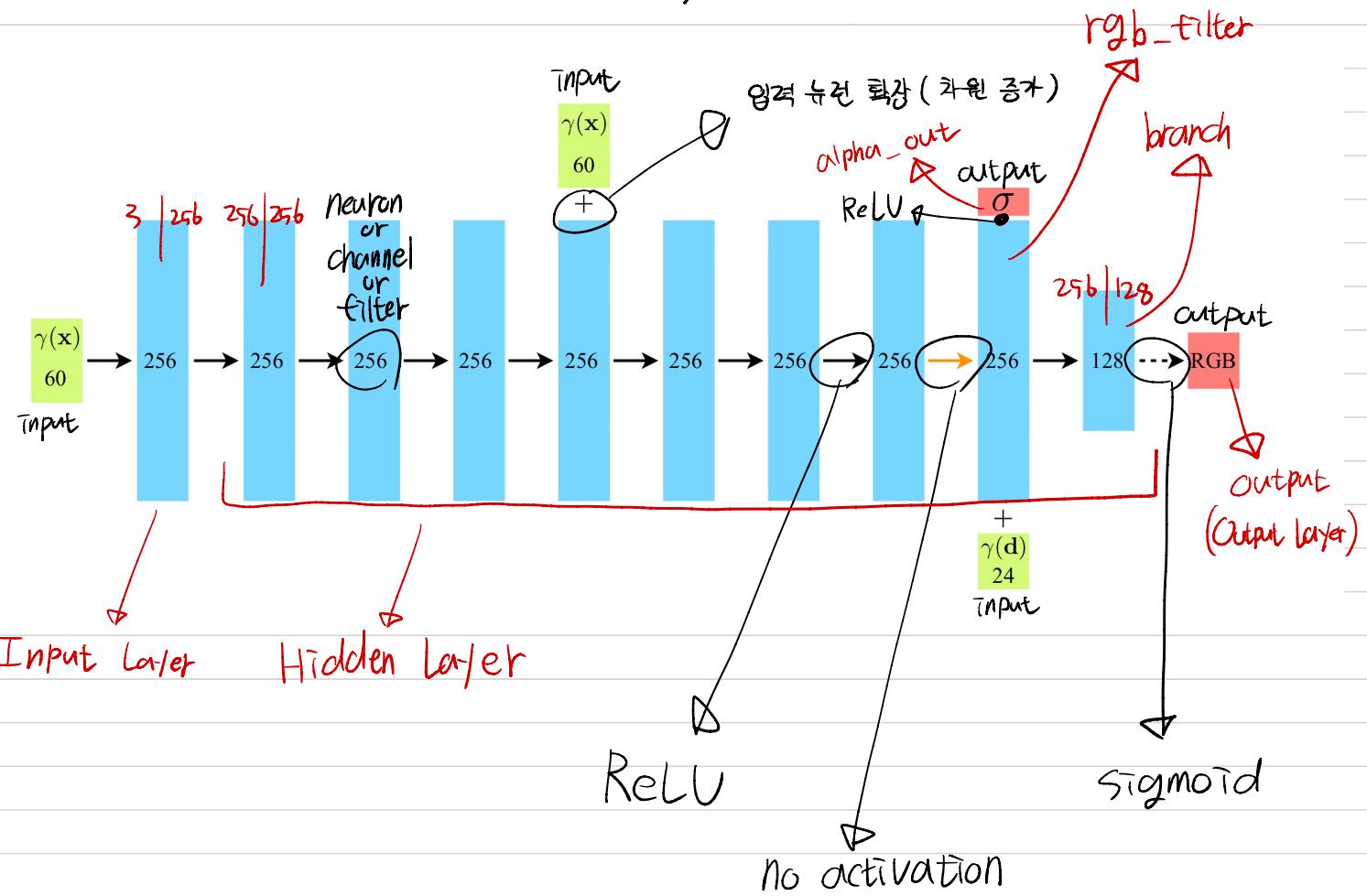
〈NeRF의 Loss function〉

Loss는 MSE 사용한다.

$$\text{Loss} = \sum_{r \in R} \left[\left\| \hat{C}_c(r) - C(r) \right\|_2^2 + \left\| \hat{C}_f(r) - C(r) \right\|_2^2 \right]$$

N_c 보다 N_f 가 더 많이 셱풀링 되어야 학습이 잘 된다.

<NeRF Model Architecture >



네트워크 중간에 들어가는 입력 레이어 $\gamma(x)$, $\gamma(d)$ 는

모델 아키텍처에서 표시되는 지점의 레이어에서 바로 사용되는 것이 아니라

해당 레이어에서 나오는 출력값과 합쳐서 다음 레이어의 입력으로 같이 들어가는 것이다.