

Attention is All You Need

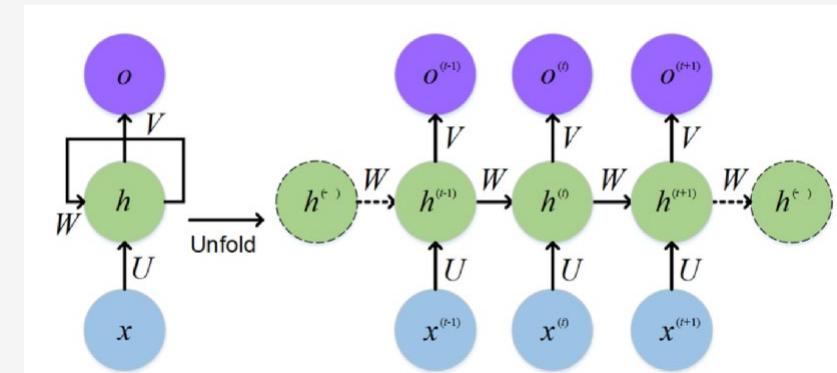
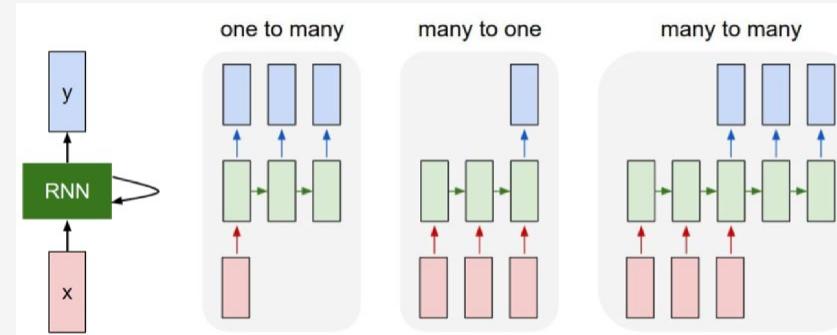
[Transformer]

목차

1. 배경 - 3p
2. 전체구조 - 2p
3. 상세 아키텍처 - 6p

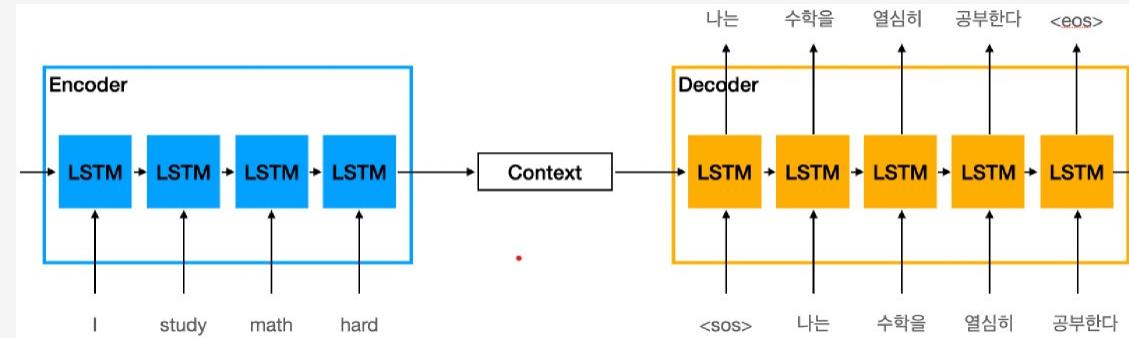
1.1. 순환 신경망 (Recurrent Neural Network, RNN)

- 은닉층(Hidden Layer)의 노드들이 순차적으로 연결된 구조를 가져, 가변적인 길이의 시퀀스 데이터를 처리
- 응용모델 : LSTM, GRU
- 입출력길이가 서로 다른 경우(예: 번역)를 처리하기 어려움
- 장기 의존성 문제(정보 손실)



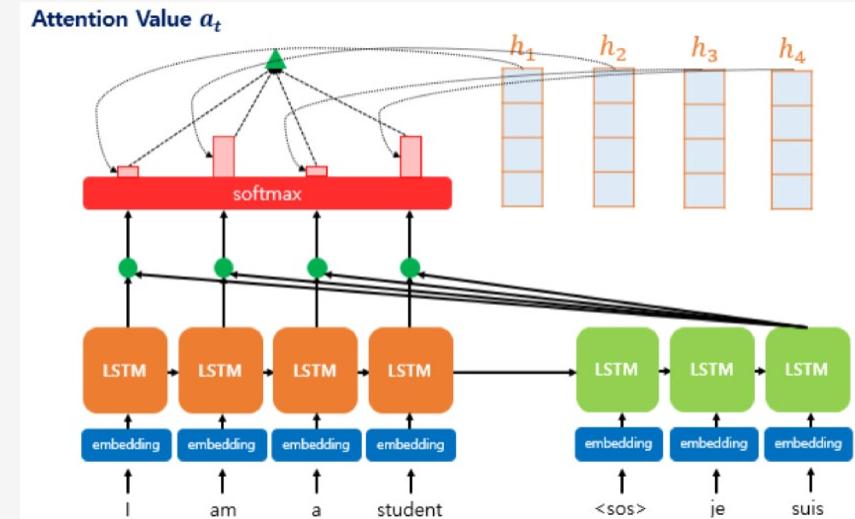
1.2. Seq2Seq(Sequence-to-Sequence)

- 인코더 : 입력 문장 전체를 하나의 **문맥 벡터(Context Vector)**로 요약
- 디코더 : 코더가 생성한 문맥 벡터를 받아서, 출력 문장을 단어 하나씩 생성
- 입출력 길이가 다른 문제를 해결 : 인코더가 입력 길이와 상관없이 정보를 압축하고, 디코더가 출력 길이에 맞춰 자유롭게 문장을 생성하므로
- 정보 병목 현상 : 디코더의 입력값인 **문맥 벡터** 한 곳에 모든 정보가 담겨 정보가 뭉개지는 현상
- RNN 계열 모델 기반으로 장기 의존성 문제(정보 손실)여전



1.3. Attention Mechanism

- 디코더가 출력 단어를 예측하는 매 시점(time-step)마다, 인코더의 전체 입력 시퀀스를 다시 한번씩 훑어 가장 관련이 깊은 입력 단어에 더 집중(Attention)하여 그 정보를 활용
(Q, K, V)
- 정보 병목 현상 해결: 디코더가 매번 동적인 *문맥 벡터* 생성 → 하나의 문맥 벡터에 모든 정보를 담을 필요가 없음.
- RNN계열 모델에 적용할 경우 장기의존성문제 해결 불가



▪ Attention score

$$e_{ij} = a(s_{i-1}, h_j)$$

▪ Attention distribution

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

▪ Attention output

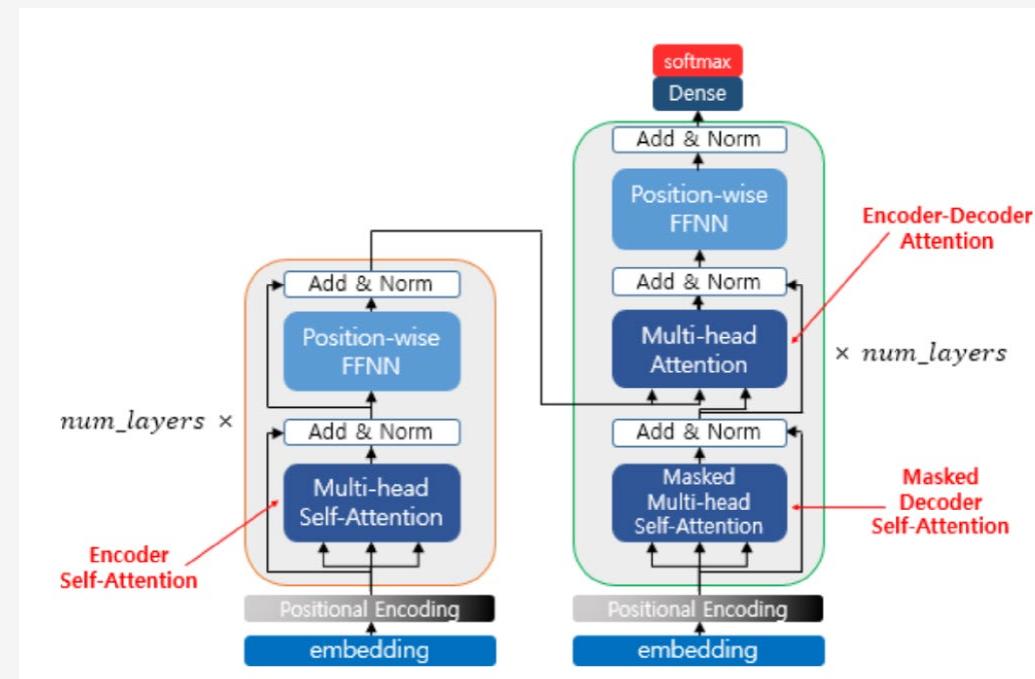
$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

▪ Decoder hidden state

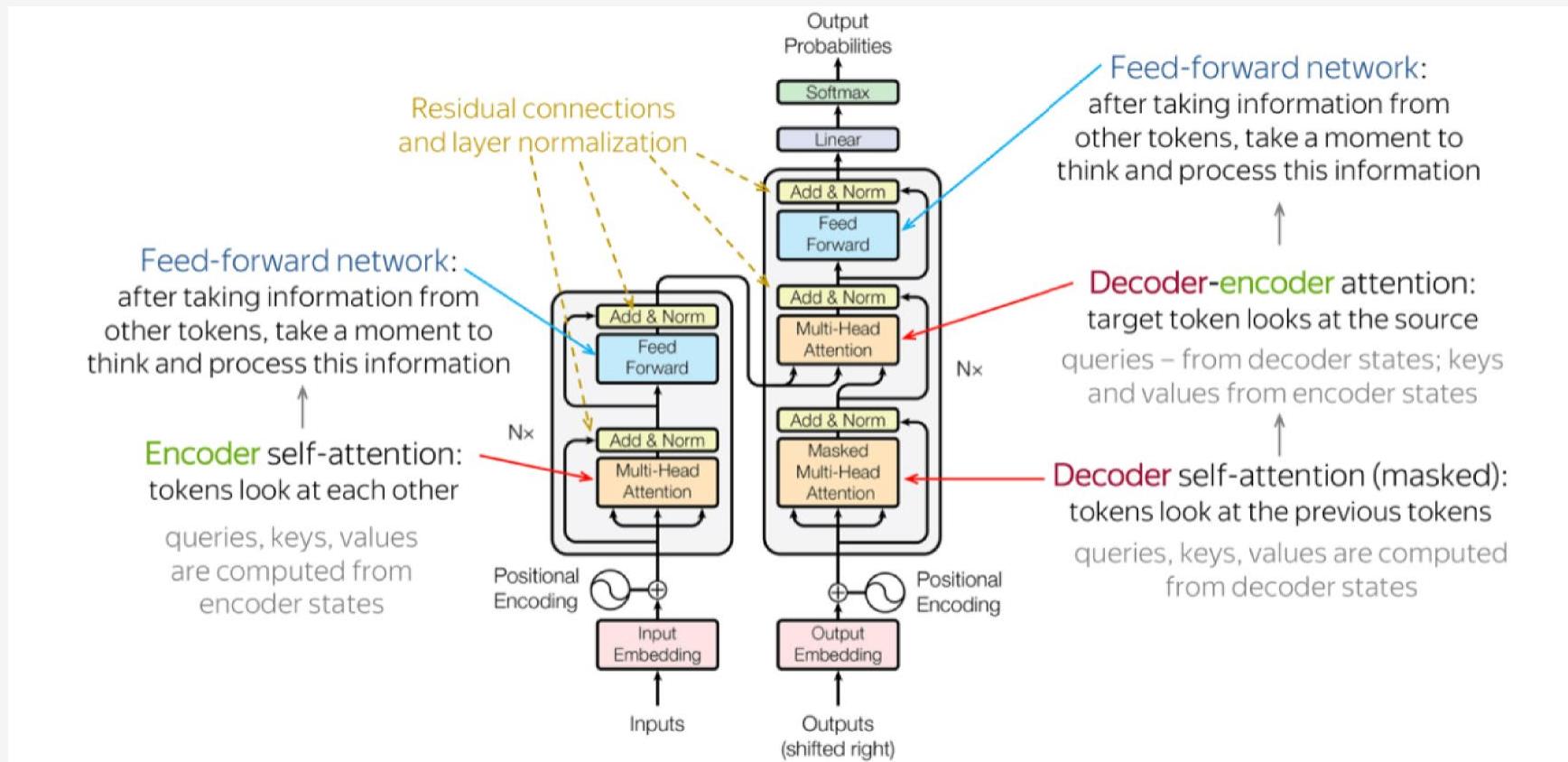
$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

2.1. 모델 아키텍처(Transformer)

- Recurrent(RNN) 나 Convolution(CNN) 없이 Attention으로만 이루어진 모델
- 각각 6개로 이루어진 Encoder-Decoder 구조
- RNN모델의 고질병 장기의존성문제를 해결 : 모든 입/출력 토큰이 서로의 관계를 계산
- 순차적계산 구조를 포기하여 대규모 병렬연산 가능 (시간복잡도 : $O(n) \rightarrow O(1)$)



2.1. 모델 아키텍처(Transformer)



3.1. Positional Encoding

```

class Embeddings(nn.Module):
    def __init__(self, vocab_num, d_model):
        super(Embeddings, self).__init__()
        self.emb = nn.Embedding(vocab_num, d_model)
        self.d_model = d_model
    def forward(self, x):
        return self.emb(x) * math.sqrt(self.d_model)

class PositionalEncoding(nn.Module):
    def __init__(self, max_seq_len, d_model, dropout=0.1):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        pe = torch.zeros(max_seq_len, d_model)

        position = torch.arange(0, max_seq_len).unsqueeze(1)
        base = torch.ones(d_model//2).fill_(10000)
        pow_term = torch.arange(0, d_model, 2) / torch.tensor(d_model, dtype=torch.float32)
        div_term = torch.pow(base, pow_term)

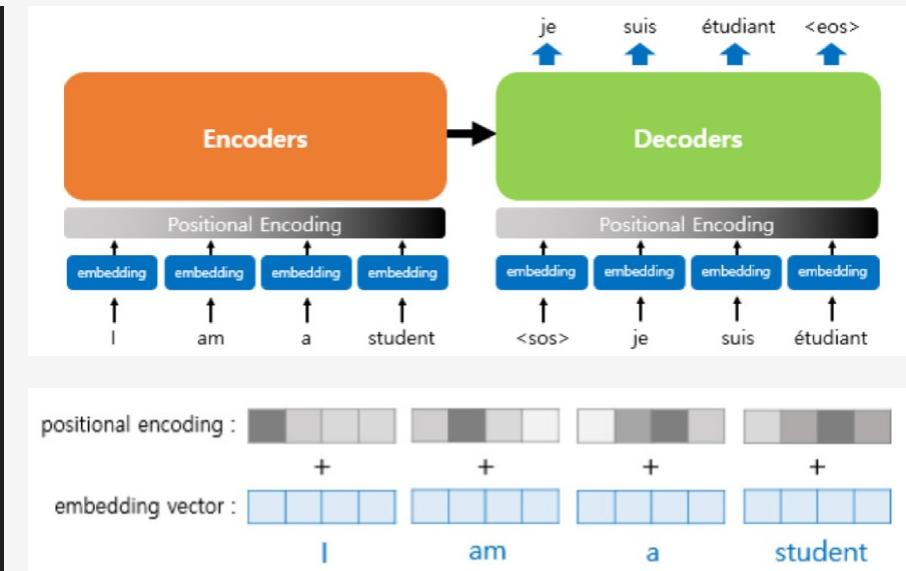
        pe[:, 0::2] = torch.sin(position / div_term)
        pe[:, 1::2] = torch.cos(position / div_term)

        pe = pe.unsqueeze(0)

        # pe를 학습되지 않는 변수로 등록
        self.register_buffer('positional_encoding', pe)

    def forward(self, x):
        x = x + Variable(self.positional_encoding[:, :x.size(1)], requires_grad=False)
        return self.dropout(x)

```



왕자는 공주를 구했다

공주는 왕자를 구했다

→ 위치정보가 없으면 어텐션을 거쳐도 차이가 거의 없다.

3.2. Self-Attention

```

class SelfAttention(nn.Module):
    def __init__(self):
        super(SelfAttention, self).__init__()
        self.matmul = torch.matmul
        self.softmax = torch.softmax

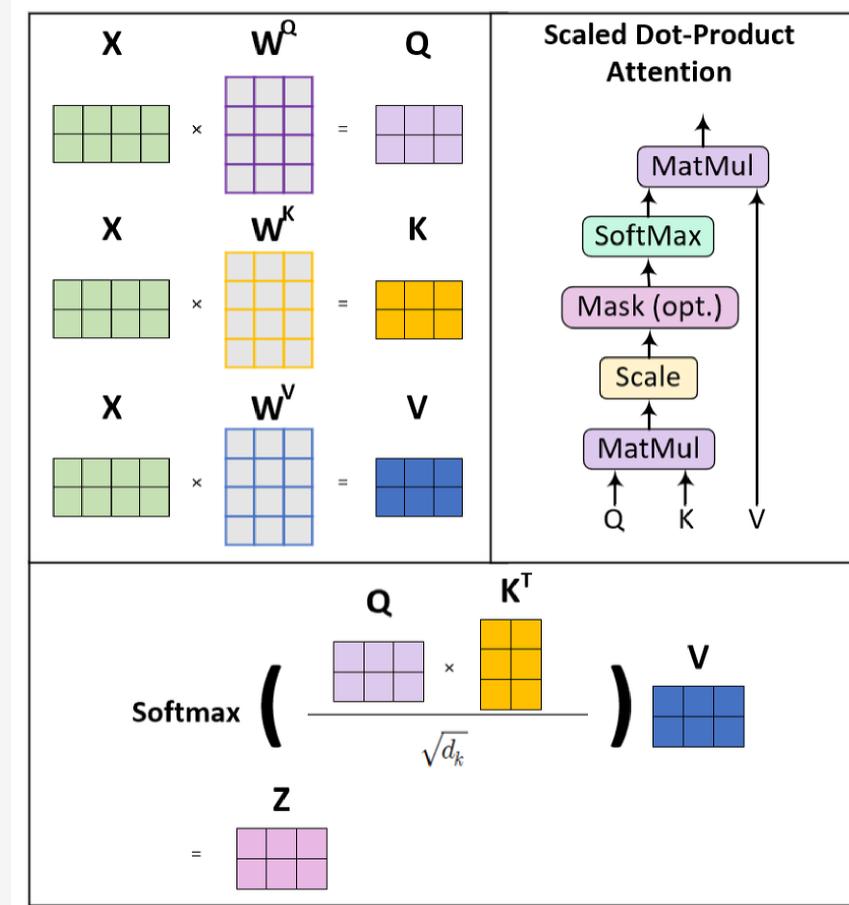
    def forward(self, query, key, value, mask=None):
        key_transpose = torch.transpose(key, -2, -1)
        matmul_result = self.matmul(query, key_transpose)
        d_k = key.size()[-1]
        attention_score = matmul_result / math.sqrt(d_k)

        if mask is not None:
            attention_score = attention_score.masked_fill(mask == 0, -1e20)

        softmax_attention_score = self.softmax(attention_score, dim=-1)
        result = self.matmul(softmax_attention_score, value)

        return result, softmax_attention_score

```



3. 상세 아키텍처(3/7)

9/38

3.3. Multi Head Attention

```
class MultiHeadAttention(nn.Module):
    def __init__(self, head_num=8, d_model = 512, dropout = 0.1):
        super(MultiheadAttention, self).__init__()

        self.head_num = head_num
        self.d_model = d_model
        self.d_k = self.d_v = d_model // head_num

        self.w_q = nn.Linear(d_model,d_model)
        self.w_k = nn.Linear(d_model,d_model)
        self.w_v = nn.Linear(d_model,d_model)
        self.w_o = nn.Linear(d_model,d_model)

        self.self_attention = SelfAttention()
        self.dropout = nn.Dropout(p=dropout)

    def forward(self, query, key, value, mask = None):
        if mask is not None:
            # Same mask applied to all h heads.
            mask = mask.unsqueeze(1)

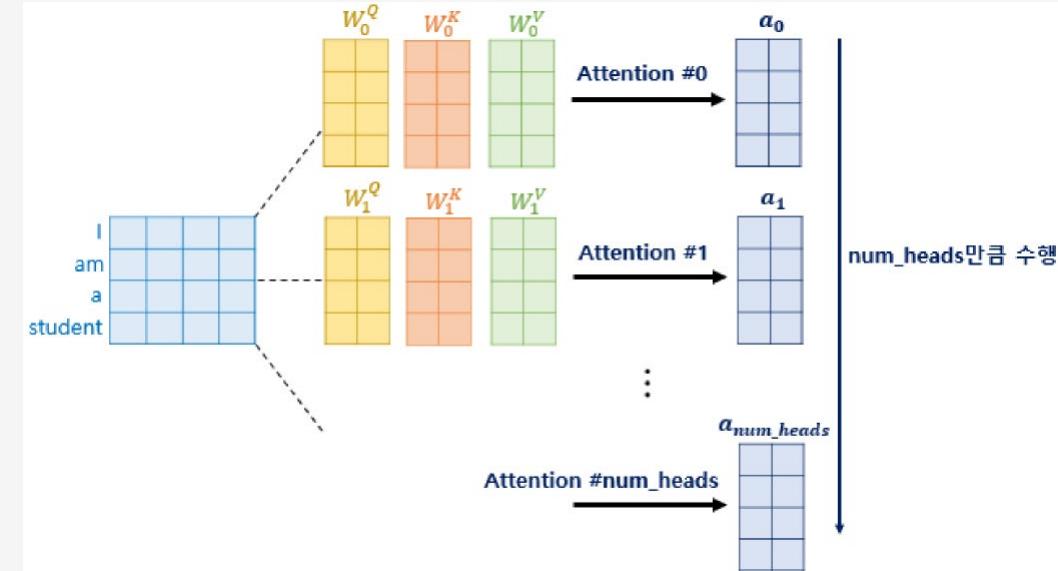
        batche_num = query.size(0)

        query = self.w_q(query).view(batche_num, -1, self.head_num, self.d_k).transpose(1, 2)
        key = self.w_k(key).view(batche_num, -1, self.head_num, self.d_k).transpose(1, 2)
        value = self.w_v(value).view(batche_num, -1, self.head_num, self.d_k).transpose(1, 2)

        attention_result, attention_score = self.self_attention(query, key, value, mask)

        attention_result = attention_result.transpose(1,2).contiguous().view(batche_num, -1, self.head_num * self.d_k)

        return self.w_o(attention_result)
```



3.4. Encoder

```
class Encoder(nn.Module):
    def __init__(self, d_model, head_num, dropout):
        super(Encoder, self).__init__()
        self.multi_head_attention = MultiHeadAttention(d_model=d_model, head_num=head_num)
        self.residual_1 = ResidualConnection(d_model, dropout=dropout)

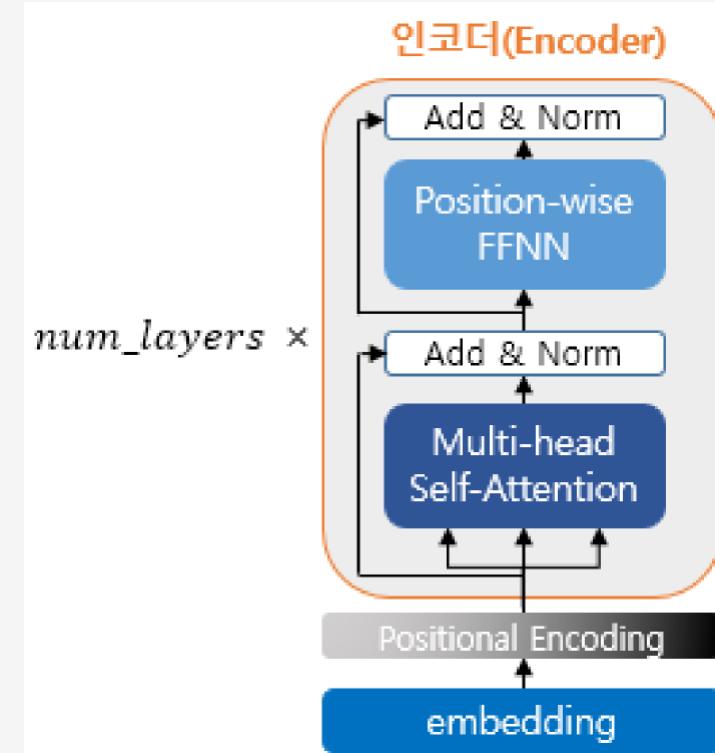
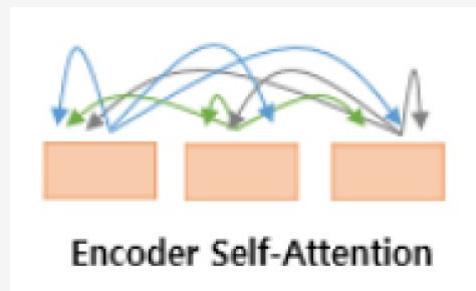
        self.feed_forward = FeedForward(d_model)
        self.residual_2 = ResidualConnection(d_model, dropout=dropout)

    def forward(self, input, mask):
        x = self.residual_1(input, lambda x: self.multi_head_attention(x, x, x, mask))
        x = self.residual_2(x, lambda x: self.feed_forward(x))

        return x
```

[인코더 어텐션]

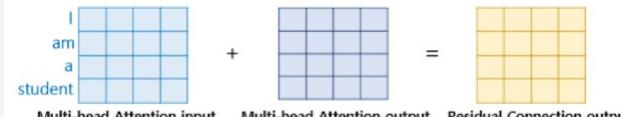
- 인코더에서 모든 벡터가 서로 어텐션



3.5. 인코더와 디코더에서 쓰이는 여러 아키텍처

(1) 잔차연결(Add)

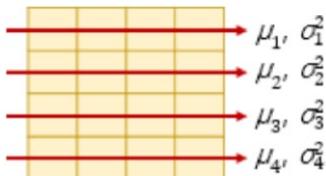
$$H(x) = x + \text{Multi - head Attention}(x)$$



- Multi Head Attention의 출력이 잔차이므로 잔차를 Attention 연산이전 원본벡터와 선형 합

→ 원본 정보 보존 / 기울기 소실문제 완화

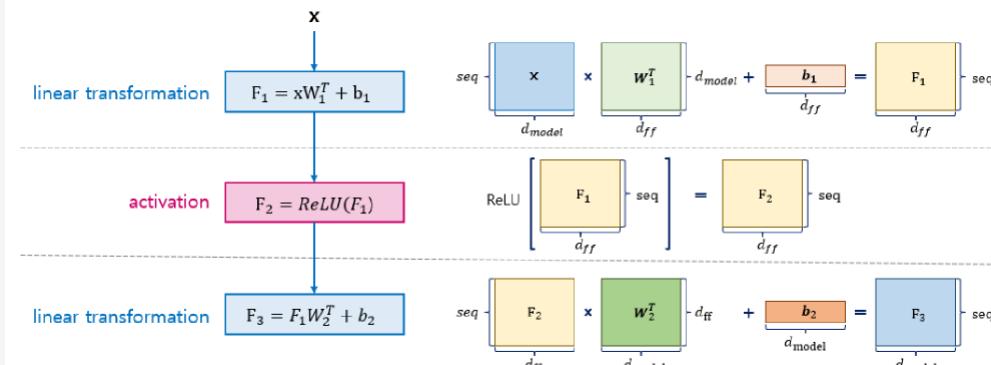
(2) 층 정규화(Norm)



Residual Connection output
(seq_len, d_model)

- 토큰의 벡터값을 표준화
→ 분포를 일정하게하여 학습을 안정화

(3) Feed Forward



- 비선형성부여
→ 선형성을 가지는 각종 *합* 어텐션 레이어 출력에 활성화함수(ReLU)를 통과시켜 비선형성부여
- 토큰의 표현을 풍부하고 깊이있게 가공
→ 차원을 일시적으로 확장하여 더 넓은 특징 추출

3.6. 잔차 연결(Residual Connection)과 트랜스포머

잔차 연결 vs 스kip 연결

스킵 연결 (Skip Connection): 더 넓은 개념. 정보가 여러 계층을 건너뛰어 전달되는 모든 구조를 지칭합니다.

잔차 연결 (Residual Connection): 스kip 연결의 가장 성공적인 구현 방식. 입력 x 가 함수 $F(x)$ 의 결과에 더해지는 $F(x) + x$ 구조입니다. 네트워크가 입력과의 차이(Residual)만 학습하도록 유도합니다.

모든 잔차 연결은 스kip 연결이지만,
모든 스kip 연결이 잔차 연결인 것은 아니다.

기울기 소실(Vanishing Gradient) 문제

잔차 학습이 없을 때, 네트워크가 깊어지면 역전파 과정에서 기울기가 점차 작아져 0에 수렴하고 학습이 어려워집니다.

1. 일반적인 심층 신경망 (Without Residual)

출력 $H^l = F(H^{l-1})$. 역전파 시 기울기는 계속해서 곱해집니다.

$$\frac{\partial L}{\partial W^l} = (\frac{\partial L}{\partial H^l}) * \dots * (\frac{\partial H^2}{\partial H^1}) * (\frac{\partial H^1}{\partial W^1})$$

만약 $|\frac{\partial H}{\partial H}| < 1$ 이면, 기울기는 0으로 소실!

2. 잔차 블록 (With Residual)

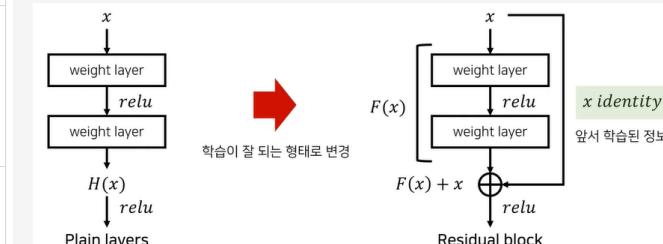
출력 $H^l = F(H^{l-1}) + H^{l-1}$. 항등 매핑(Identity)이 추가됩니다.

$$\frac{\partial H^l}{\partial W^l} = \frac{\partial}{\partial H^l} [F(H^{l-1}) + H^{l-1}] = (\dots) + 1$$

'1'이 더해져 기울기 흐름이 유지되어 소실을 방지!

트랜스포머 vs 다른 모델의 잔차 연결

구분	ResNet 등 일반 모델	트랜스포머
주요 목적	네트워크 깊이의 한계 극복 및 기울기 소실 해결. 수백 개의 층을 쌓아도 학습이 가능하게 함.	정보 보존 및 흐름 제어. 복잡한 연산 중 원본 정보가 소실되지 않도록 보존하는 데 중점.
구조적 역할	주로 Convolution 블록 전체가 끝난 이후에 적용됨.	어텐션, FFNN 등 각 서브 레이어(Sub-layer) 직후에 바로 적용됨.
데일리터 흐름	$x \rightarrow \text{Conv Block} \rightarrow F(x) \rightarrow F(x) + x$	$x \rightarrow \text{Sub-layer}(x) \rightarrow \text{LayerNorm}(\text{Sub-layer}(x) + x)$



핵심 차이 요약

ResNet의 잔차 연결은 '더 깊은 모델'을 만드는 것이 핵심이라면,

트랜스포머의 잔차 연결은 복잡한 변환 속에서 '입력 정보의 원형을 보존'하는 역할이 매우 중요합니다. 어텐션이 재조합한 정보와 원본 정보를 모두 활용하여 안정적인 학습을 가능하게 합니다.

- (1) 일반 레이어 오차역전파시 로컬 그래디언트 중 하나라도 0에 근접하면 최종 그래디언트가 0에 수렴 한다.
- (2) 잔차 연결 레이어 오차역전파시 +1이 더해져 로컬 그래디언트가 0에 근접하는 것을 방지하기 때문에 최종 그래디언트가 0에 수렴하는 것을 막는다.

(1) 일반 레이어

순전파 (Forward Pass):

- $x_1 = F_1(x_0)$
- $x_2 = F_2(x_1)$
- $x_3 = F_3(x_2)$

역전파 (Backward Pass - Chain Rule):

$$\frac{\partial L}{\partial x_0} = \frac{\partial L}{\partial x_3} \times \frac{\partial F_3}{\partial x_2} \times \frac{\partial F_2}{\partial x_1} \times \frac{\partial F_1}{\partial x_0} = -0.125 \times (-0.000001) = 0.000000125$$

-0.000001
-0.5 -0.5 -0.5

(2) 잔차 연결 레이어

순전파 (Forward Pass):

- $x_1 = F_1(x_0) + x_0$
- $x_2 = F_2(x_1) + x_1$
- $x_3 = F_3(x_2) + x_2$

역전파 (Backward Pass - Chain Rule):

$$\frac{\partial L}{\partial x_0} = \frac{\partial L}{\partial x_3} \times (\frac{\partial F_3}{\partial x_2} + 1) \times (\frac{\partial F_2}{\partial x_1} + 1) \times (\frac{\partial F_1}{\partial x_0} + 1) = -0.125 * 0.999999 = -0.124999875$$

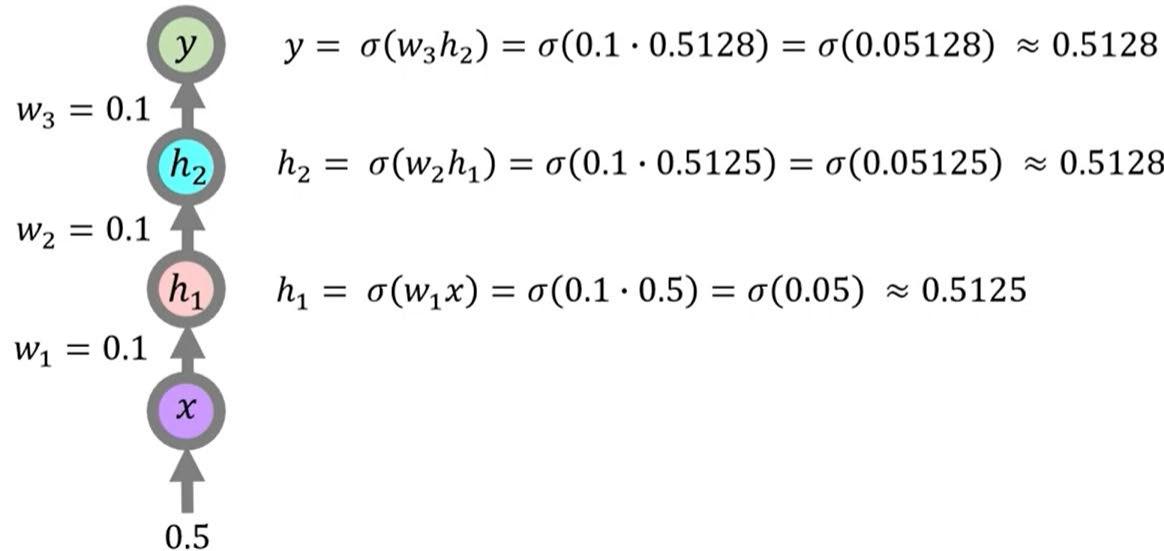
0.999999
-0.5 0.5 0.5

3.6. 잔차 연결(Residual Connection) - 순전파

시나리오1: 그냥 신경망

(활성화함수는 시그모이드)

$$\text{loss} = (y_{true} - y)^2 = (1 - 0.5128)^2 \approx 0.2372$$

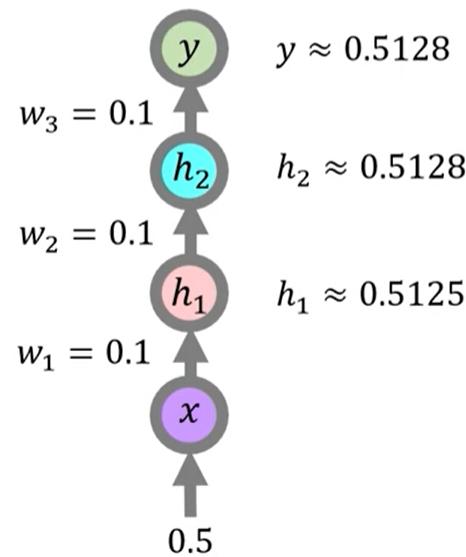


3.6. 잔차 연결(Residual Connection) - 역전파

시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial (w_3 h_2)} \cdot \frac{\partial (w_3 h_2)}{\partial w_3} = -0.9744 \cdot 0.2498 \cdot 0.5128 = -0.1248$$



$$\frac{\partial L}{\partial y} = 2(y - y_{true}) = 2(0.5128 - 1) = -0.9744$$

$$\frac{\partial y}{\partial (w_3 h_2)} = \sigma'(w_3 h_2) = y(1 - y) = 0.5128(1 - 0.5128) \approx 0.2498$$

$$\frac{\partial (w_3 h_2)}{\partial w_3} = h_2 \approx 0.5128$$

3.6. 잔차 연결(Residual Connection) - 역전파

시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$

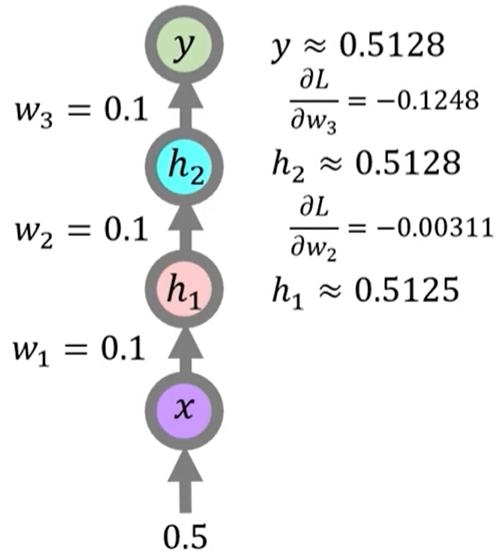
$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial(w_3 h_2)} \cdot \frac{\partial(w_3 h_2)}{\partial h_2} \cdot \frac{\partial h_2}{\partial(w_2 h_1)} \cdot \frac{\partial(w_2 h_1)}{\partial h_1} \cdot \frac{\partial h_1}{\partial(w_1 x)} \cdot \frac{\partial(w_1 x)}{\partial w_1}$$

$$= -0.9744 \cdot 0.2498 \cdot 0.1 \cdot 0.2498 \cdot \frac{\partial(w_2 h_1)}{\partial h_1} \cdot \frac{\partial h_1}{\partial(w_1 x)} \cdot \frac{\partial(w_1 x)}{\partial w_1}$$

$$= -0.00608 \cdot 0.1 \cdot \frac{\partial h_1}{\partial(w_1 x)} \cdot \frac{\partial(w_1 x)}{\partial w_1}$$

$$= -0.00608 \cdot 0.1 \cdot 0.2498 \cdot 0.5$$

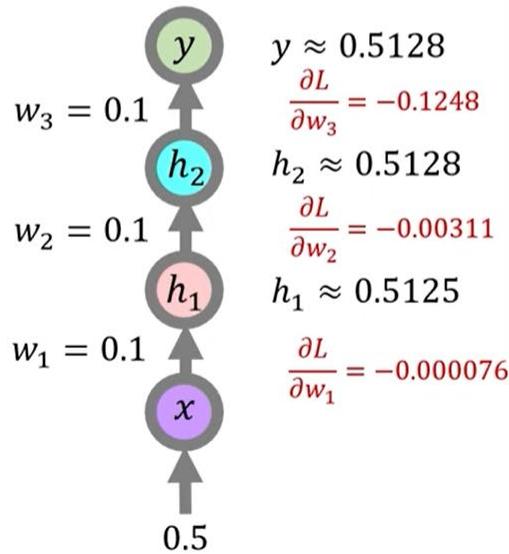
$$= -0.000076$$



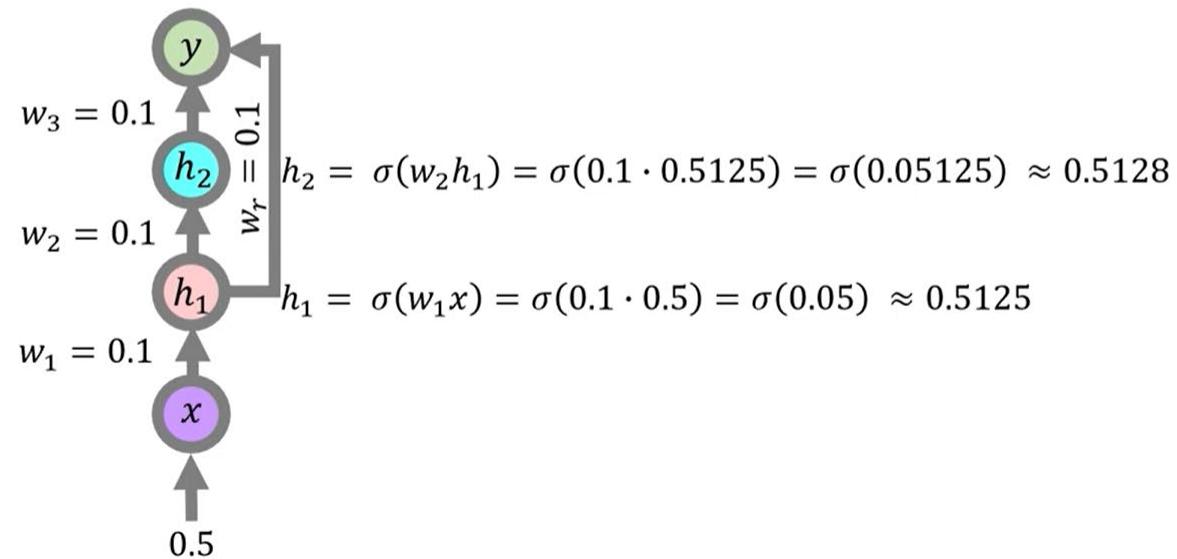
3.6. 잔차 연결(Residual Connection) - 순전파

시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



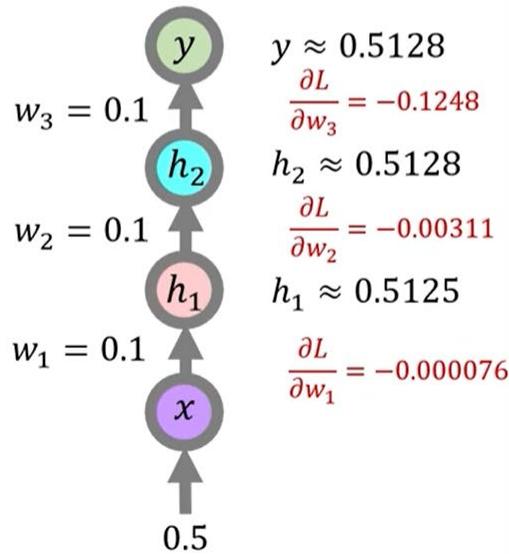
시나리오2: 잔차 신경망
(활성화함수는 시그모이드)



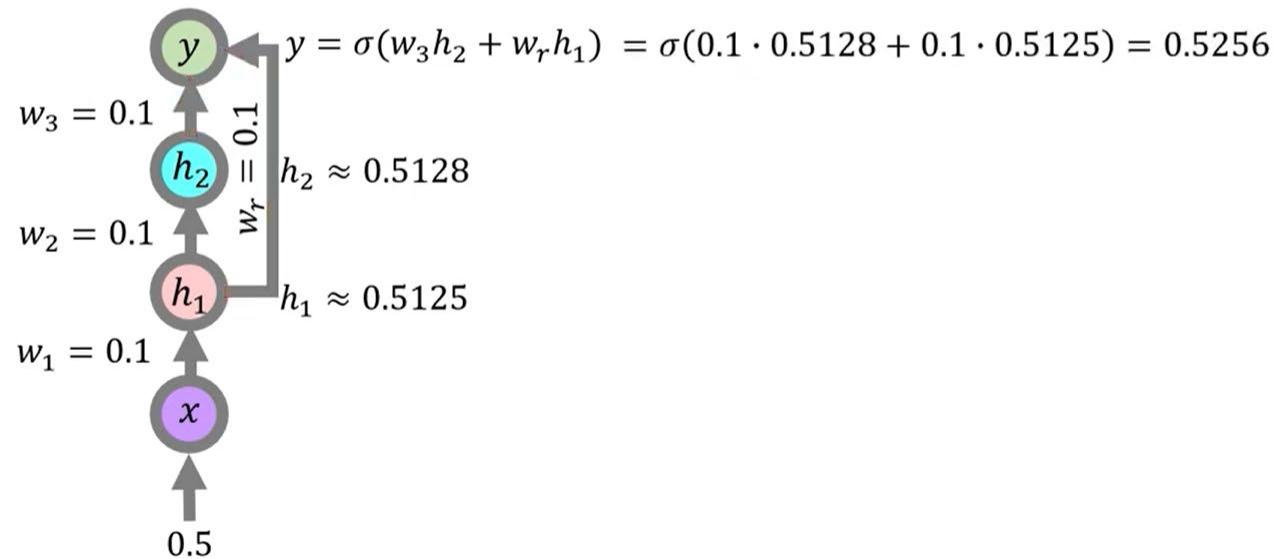
3.6. 잔차 연결(Residual Connection) - 순전파

시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



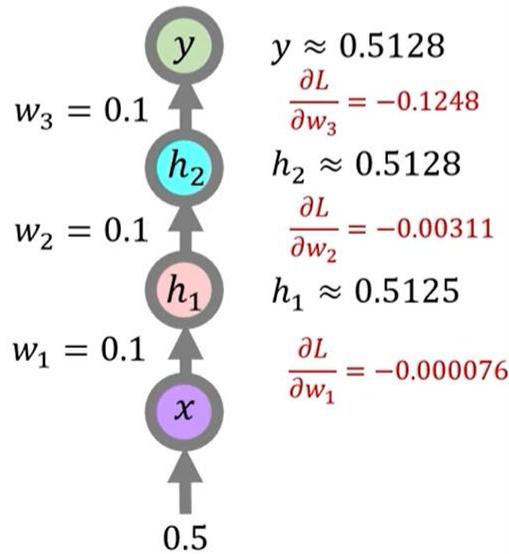
시나리오2: 잔차 신경망
(활성화함수는 시그모이드)



3.6. 잔차 연결(Residual Connection) - 순전파

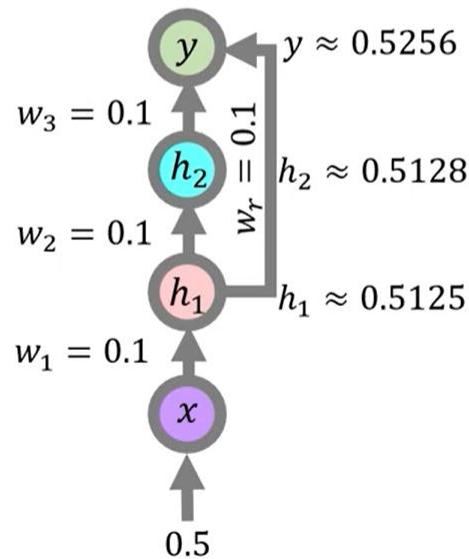
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

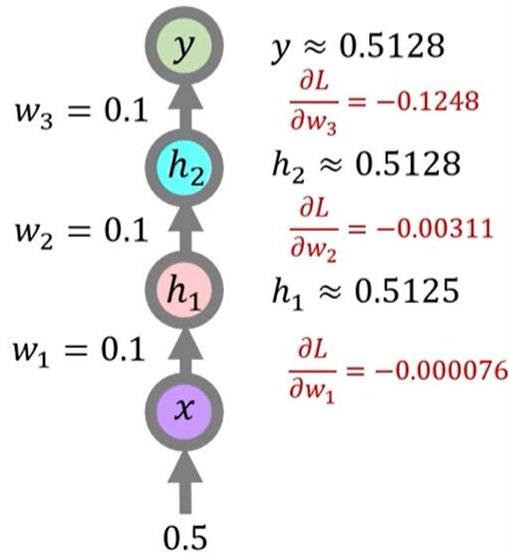
$$\text{loss} = (y_{true} - y)^2 = (1 - 0.5256)^2 \approx 0.2247$$



3.6. 잔차 연결(Residual Connection) - 역전파

시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

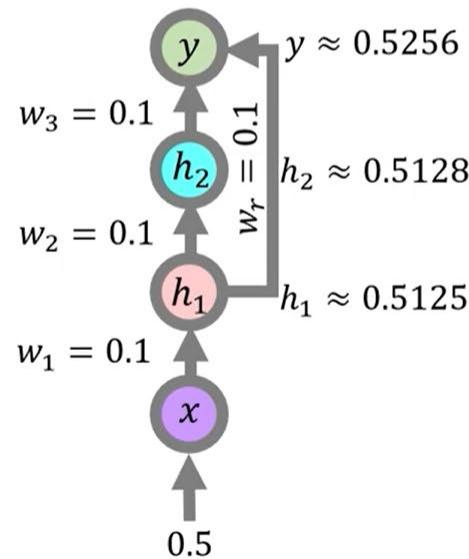
$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2247$$

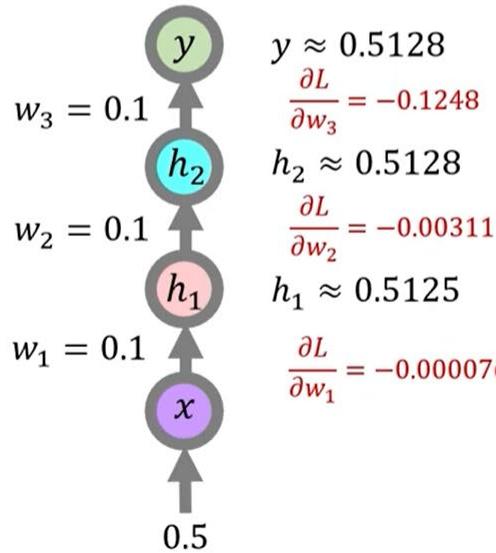
$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial (w_3 h_2 + w_r h_1)} \cdot \frac{\partial (w_3 h_2 + w_r h_1)}{\partial w_3}$$



3.6. 잔차 연결(Residual Connection) - 역전파

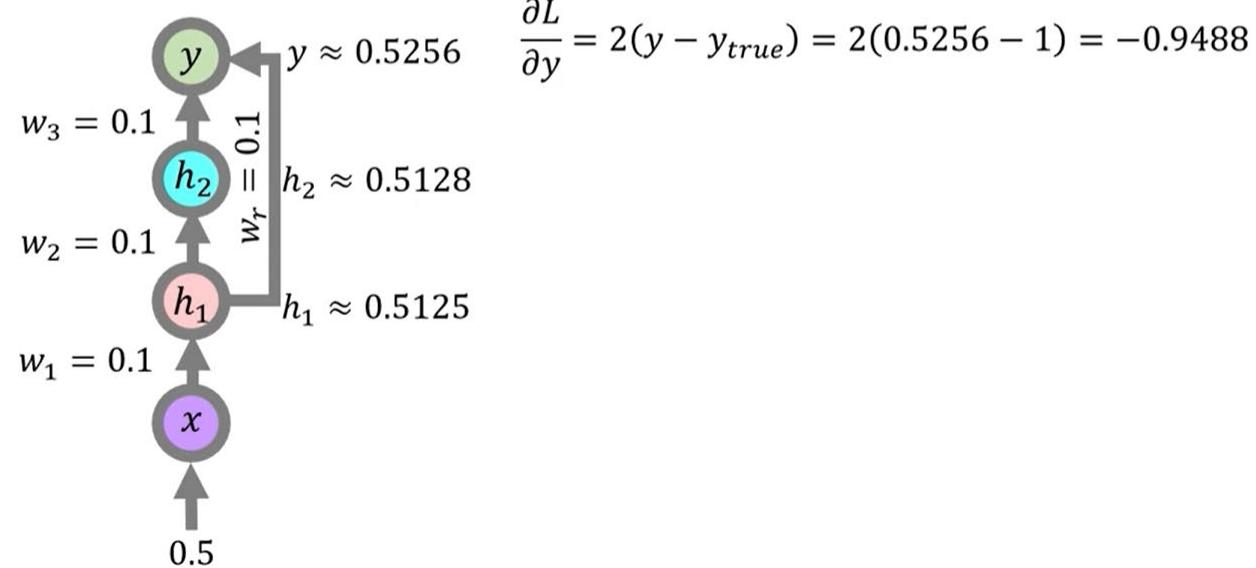
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

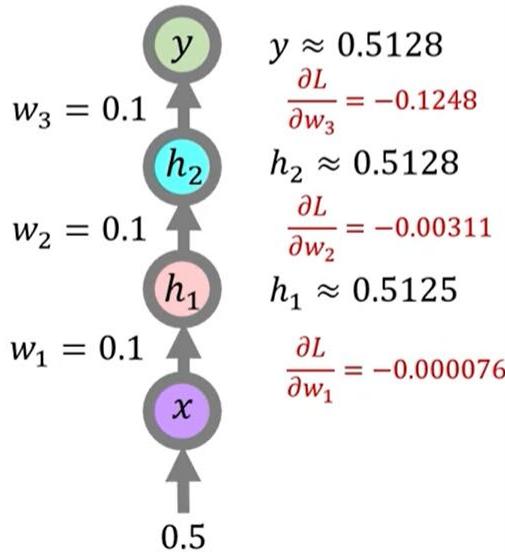
$$\text{loss} \approx 0.2247$$



3.6. 잔차 연결(Residual Connection) - 역전파

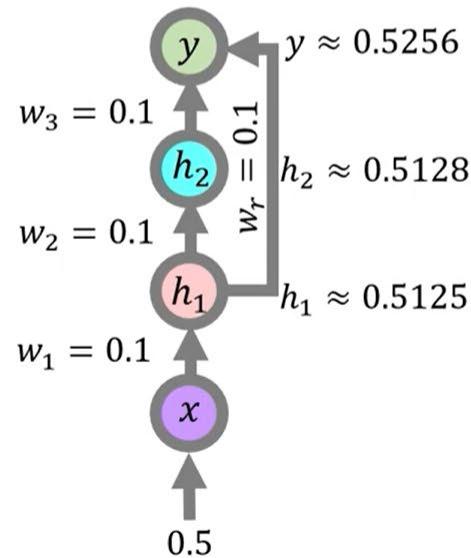
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2247$$



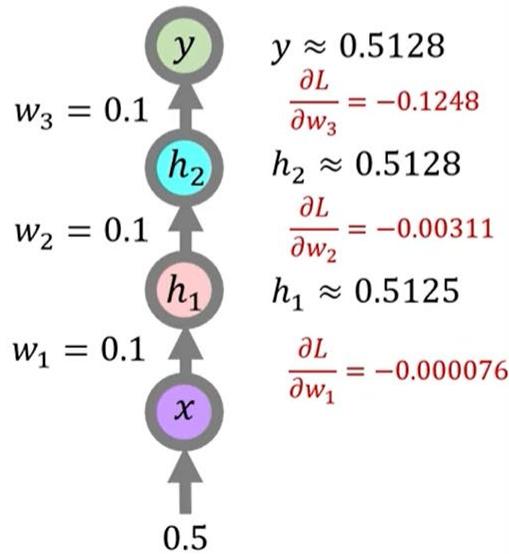
$$\frac{\partial L}{\partial w_3} = -0.9488 \cdot \frac{\partial y}{\partial (w_3 h_2 + w_r h_1)} \cdot \frac{\partial (w_3 h_2 + w_r h_1)}{\partial w_3}$$

$$\begin{aligned} \frac{\partial y}{\partial (w_3 h_2 + w_r h_1)} &= \frac{\partial (\sigma(w_3 h_2 + w_r h_1))}{\partial (w_3 h_2 + w_r h_1)} \\ &= y(1 - y) = 0.5256(1 - 0.5256) \\ &\approx 0.2493 \end{aligned}$$

3.6. 잔차 연결(Residual Connection) - 역전파

시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

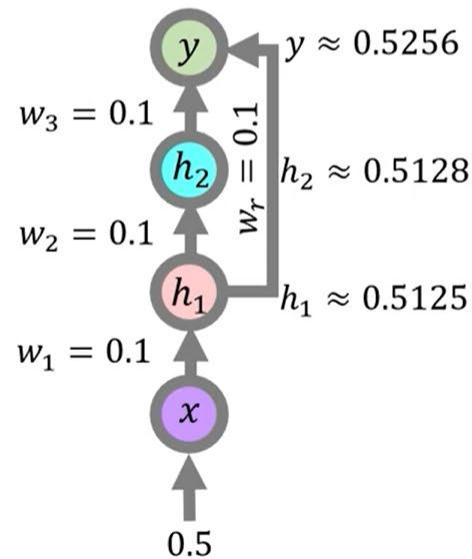
$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2247$$

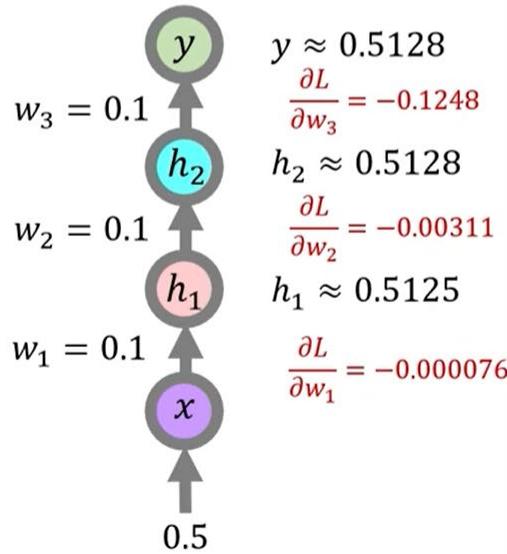
$$\frac{\partial L}{\partial w_3} = -0.9488 \cdot 0.2493 \cdot \frac{\partial(w_3 h_2 + w_r h_1)}{\partial w_3}$$



3.6. 잔차 연결(Residual Connection) - 역전파

시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

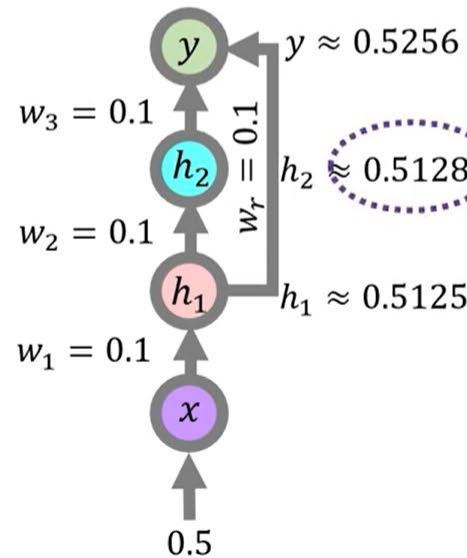
$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2247$$

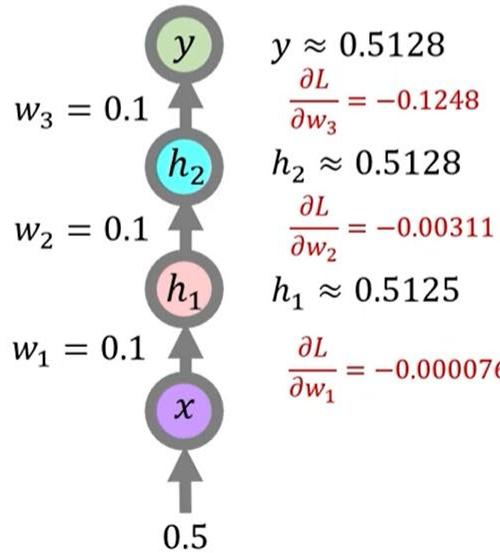
$$\frac{\partial L}{\partial w_3} = -0.9488 \cdot 0.2493 \cdot 0.5128$$



3.6. 잔차 연결(Residual Connection) - 역전파

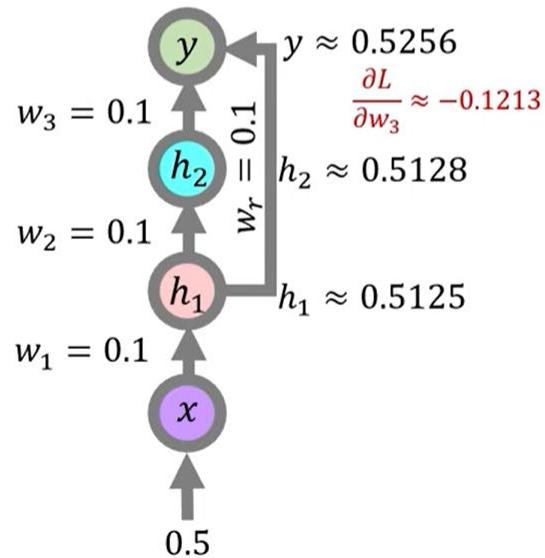
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

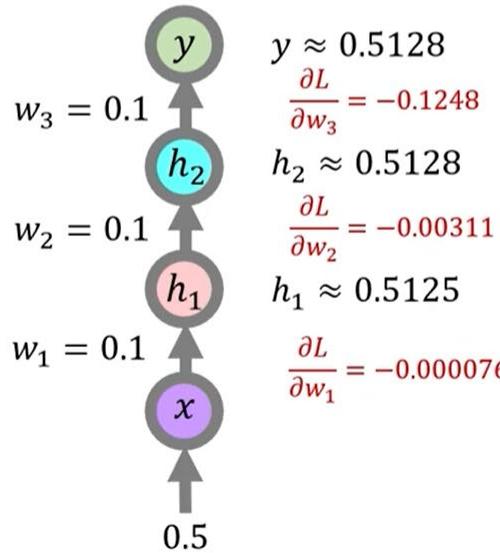
$$\text{loss} \approx 0.2247 \quad \frac{\partial L}{\partial w_2} = \boxed{\frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial(w_3 h_2 + w_r h_1)} \cdot \frac{\partial(w_3 h_2 + w_r h_1)}{\partial h_2} \cdot \frac{\partial h_2}{\partial w_2 h_1} \cdot \frac{\partial w_2 h_1}{\partial w_2}}$$



3.6. 잔차 연결(Residual Connection) - 역전파

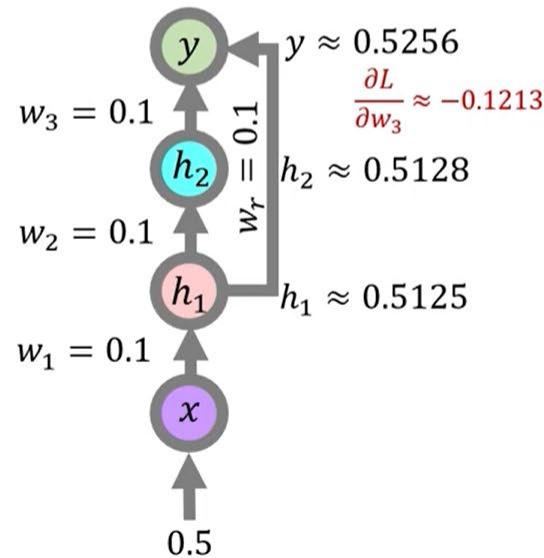
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

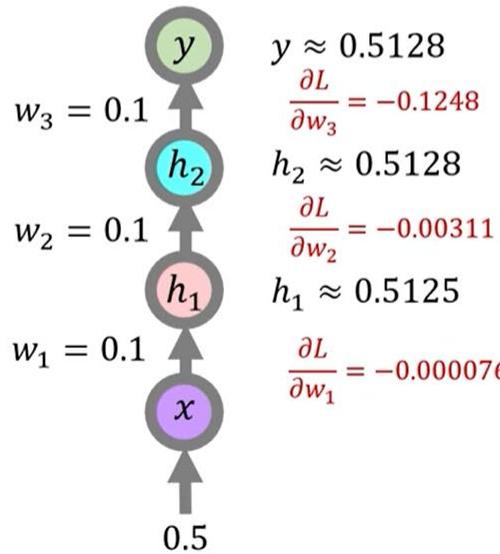
$$\text{loss} \approx 0.2247 \quad \frac{\partial L}{\partial w_2} = -0.9488 \cdot 0.2493 \cdot \frac{\partial(w_3 h_2 + w_1 h_1)}{\partial w_2} \cdot \frac{\partial h_2}{\partial w_2 h_1} \cdot \frac{\partial w_2 h_1}{\partial w_2}$$



3.6. 잔차 연결(Residual Connection) - 역전파

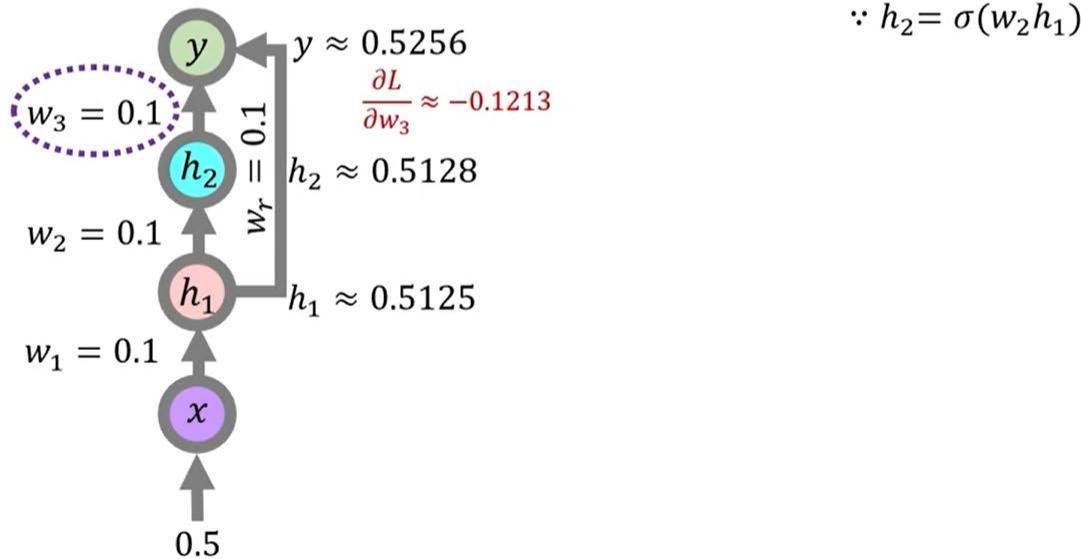
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2247 \quad \frac{\partial L}{\partial w_2} = -0.9488 \cdot 0.2493 \cdot 0.1 \cdot h_2(1 - h_2) \cdot \frac{\partial w_2 h_1}{\partial w_2}$$

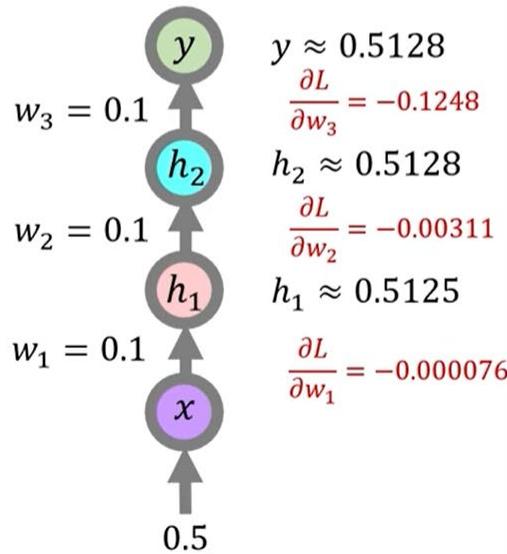


$$\because h_2 = \sigma(w_2 h_1)$$

3.6. 잔차 연결(Residual Connection) - 역전파

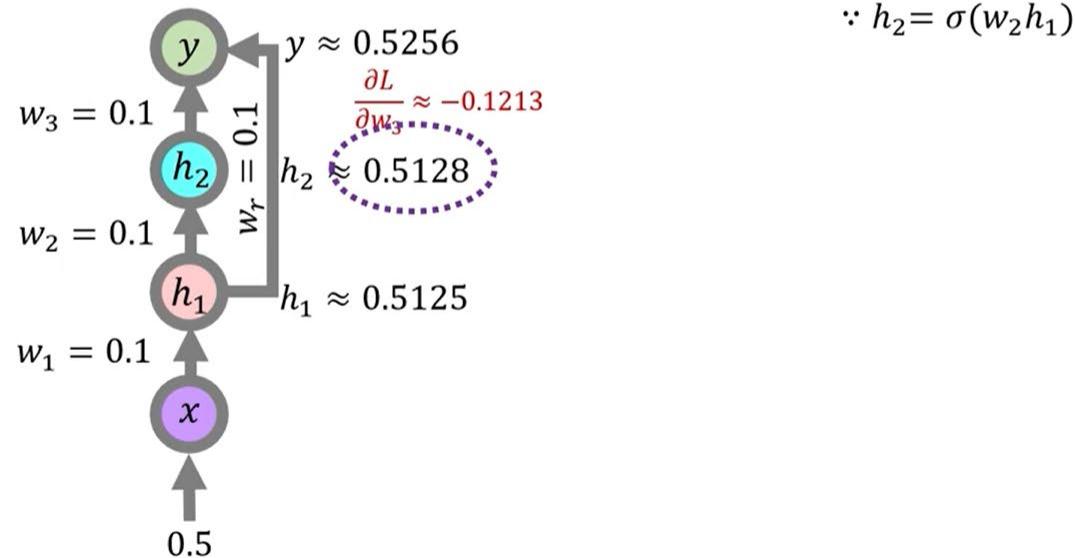
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

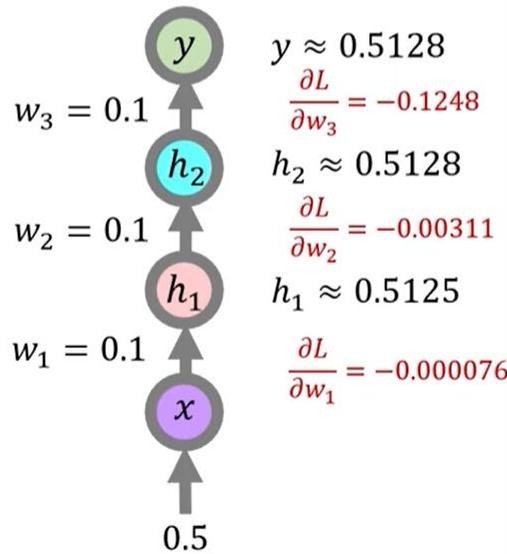
$$\text{loss} \approx 0.2247 \quad \frac{\partial L}{\partial w_2} = -0.9488 \cdot 0.2493 \cdot 0.1 \cdot 0.2498 \cdot \frac{\partial w_2 h_1}{\partial w_2}$$



3.6. 잔차 연결(Residual Connection) - 역전파

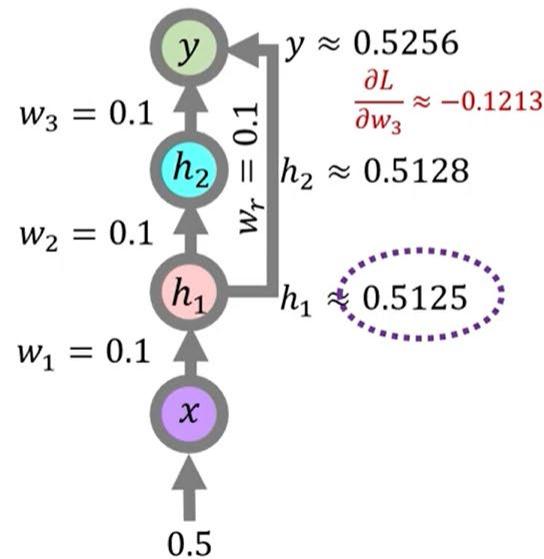
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

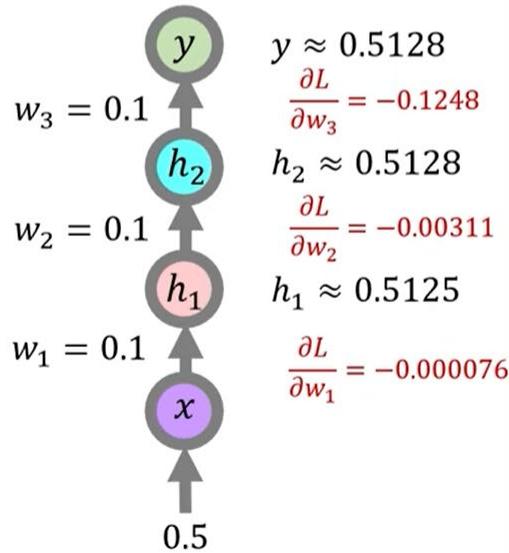
$$\text{loss} \approx 0.2247 \quad \frac{\partial L}{\partial w_2} = -0.9488 \cdot 0.2493 \cdot 0.1 \cdot 0.2498 \cdot 0.5125$$



3.6. 잔차 연결(Residual Connection) - 역전파

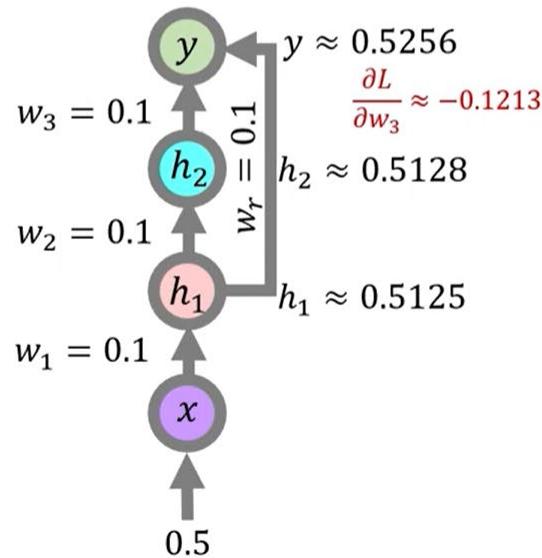
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

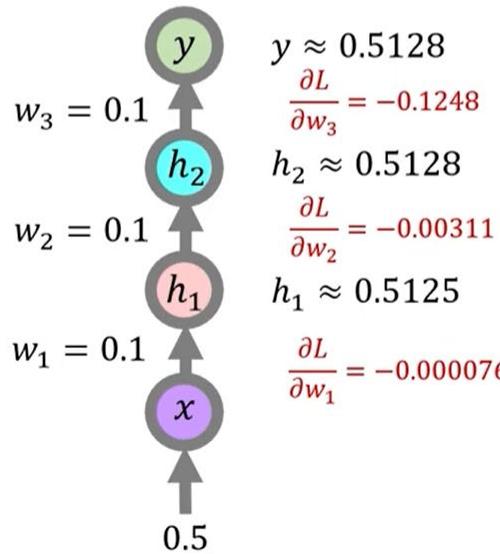
$$\text{loss} \approx 0.2247 \quad \frac{\partial L}{\partial w_2} = -0.9488 \cdot 0.2493 \cdot 0.1 \cdot 0.2498 \cdot 0.5125 = \textcolor{red}{-0.0030}$$



3.6. 잔차 연결(Residual Connection) - 역전파

시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

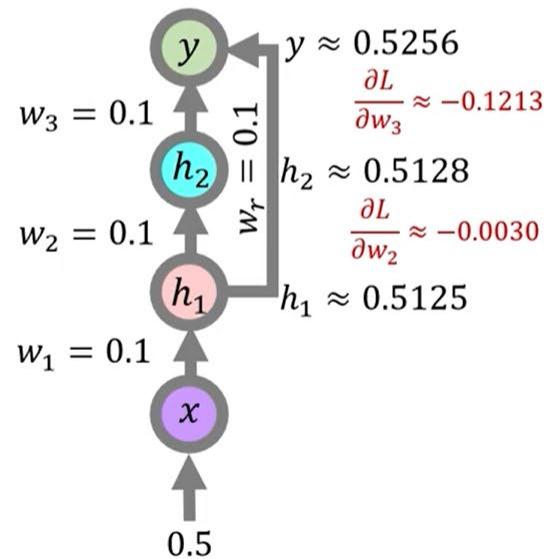
$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2247$$

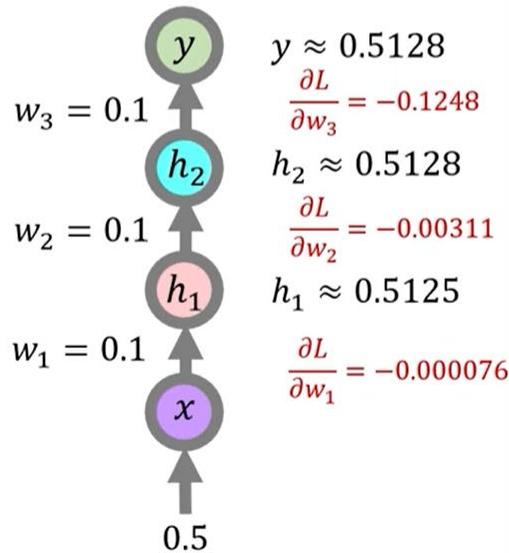
$$\frac{\partial L}{\partial w_1} = \boxed{\frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial (w_3 h_2 + w_r h_1)} \cdot \frac{\partial (w_3 h_2 + w_r h_1)}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1 x} \cdot \frac{\partial w_1 x}{\partial w_1}}$$



3.6. 잔차 연결(Residual Connection) - 역전파

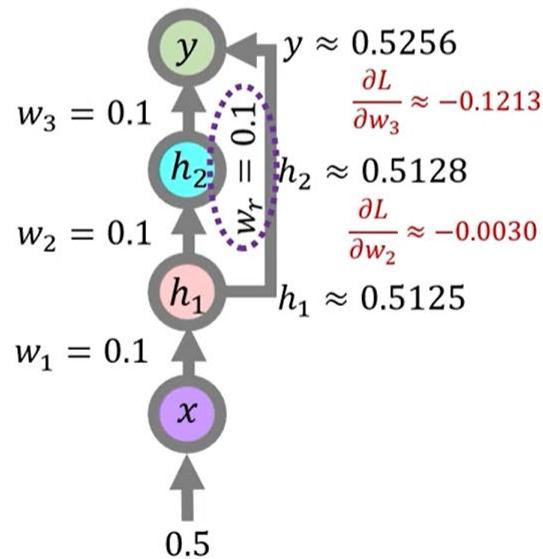
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

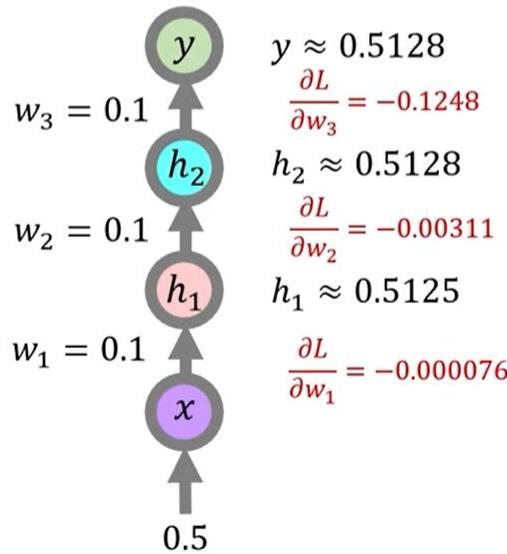
$$\text{loss} \approx 0.2247 \quad \frac{\partial L}{\partial w_1} = -0.9488 \cdot 0.2493 \cdot 0.1 \cdot \frac{\partial h_1}{\partial w_1 x} \cdot \frac{\partial w_1 x}{\partial w_1}$$



3.6. 잔차 연결(Residual Connection) - 역전파

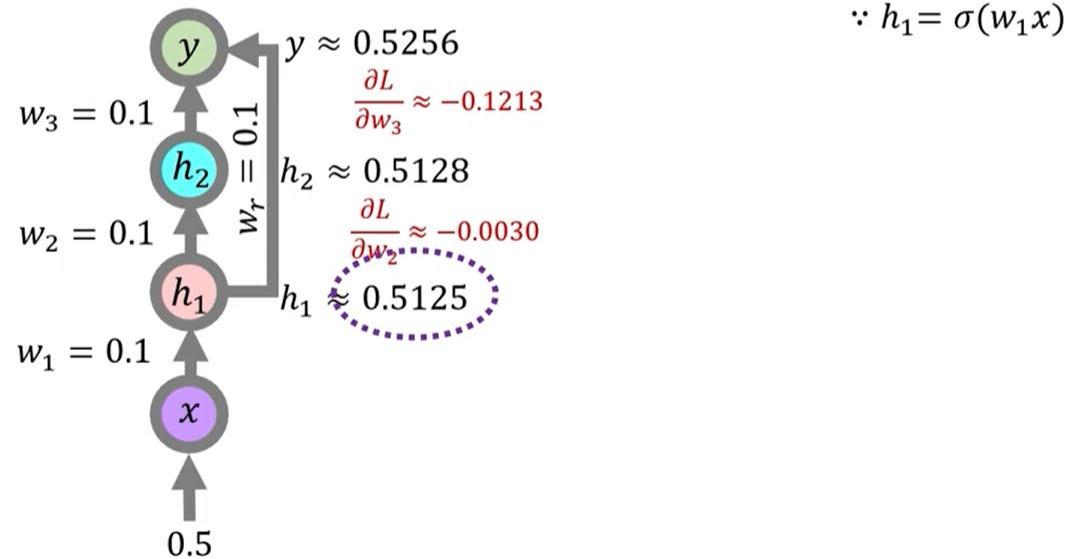
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

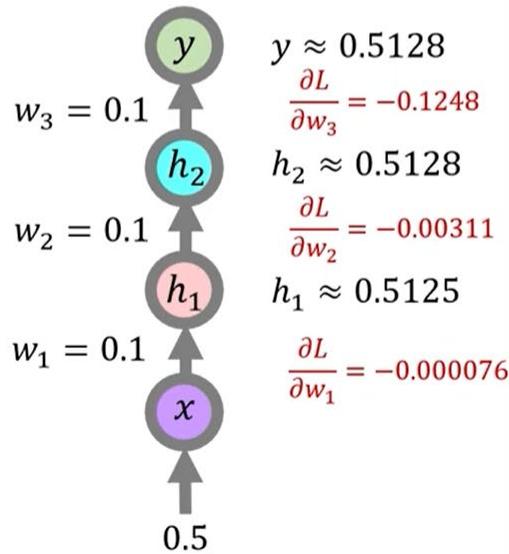
$$\text{loss} \approx 0.2247 \quad \frac{\partial L}{\partial w_1} = -0.9488 \cdot 0.2493 \cdot 0.1 \cdot 0.2498 \cdot \frac{\partial w_1 x}{\partial w_1}$$



3.6. 잔차 연결(Residual Connection) - 역전파

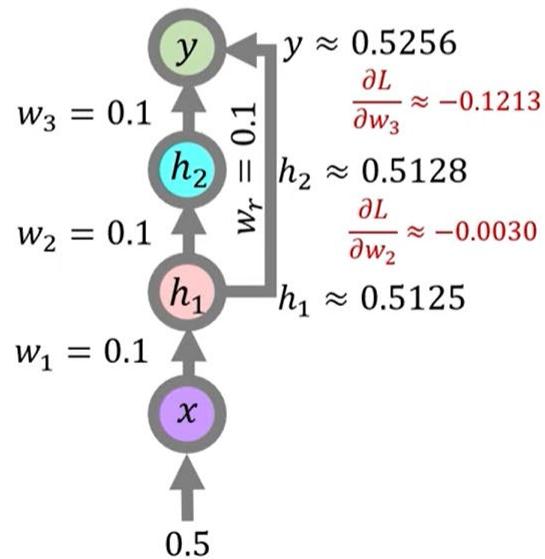
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

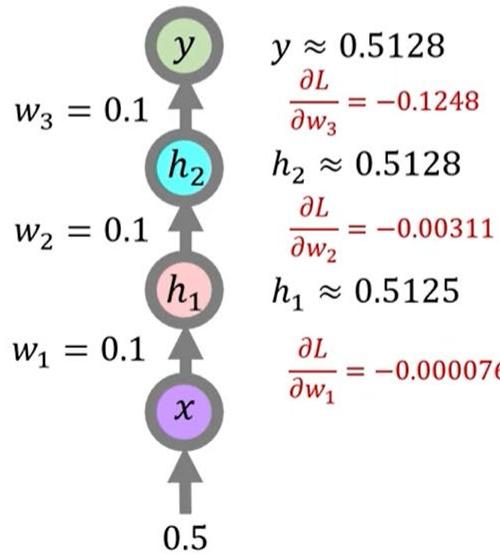
$$\text{loss} \approx 0.2247 \quad \frac{\partial L}{\partial w_1} = -0.9488 \cdot 0.2493 \cdot 0.1 \cdot 0.2498 \cdot 0.5 = \color{red}{-0.00295}$$



3.6. 잔차 연결(Residual Connection) - 역전파

시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

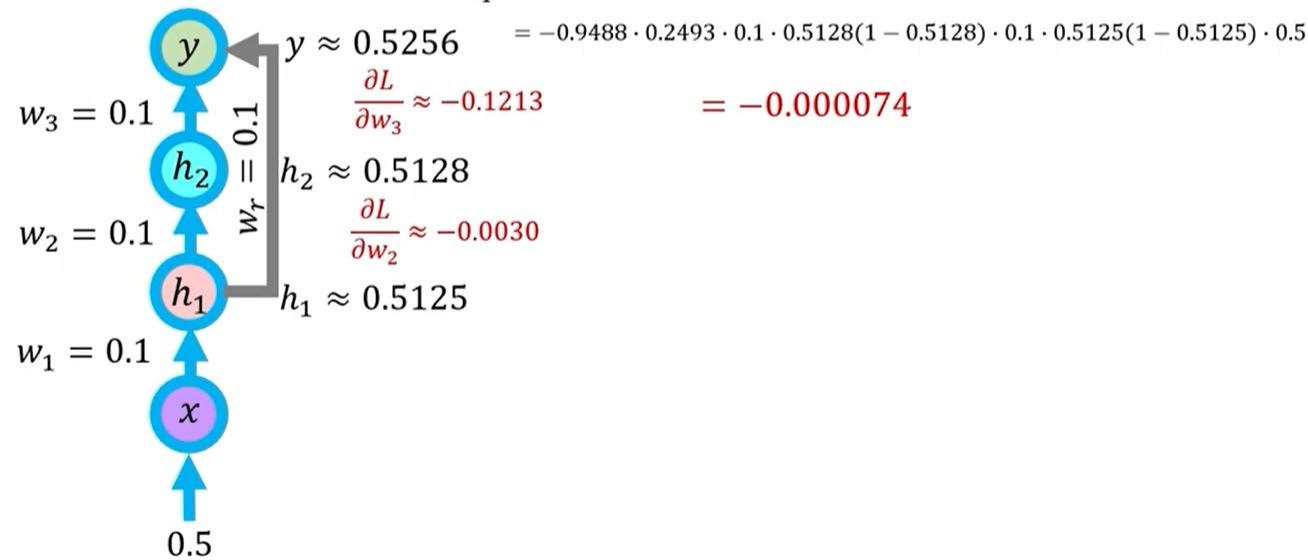
$$\text{loss} \approx 0.2372$$



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2247 \quad \frac{\partial L}{\partial w_1} = -0.00295$$

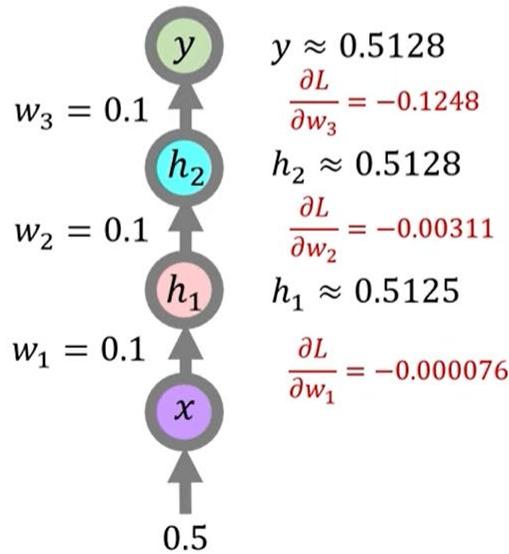
$$\frac{\partial L}{\partial w_1} = -0.9488 \cdot 0.2493 \cdot w_3 \cdot h_2(1 - h_2) \cdot w_2 \cdot h_1(1 - h_1) \cdot x$$



3.6. 잔차 연결(Residual Connection) - 역전파

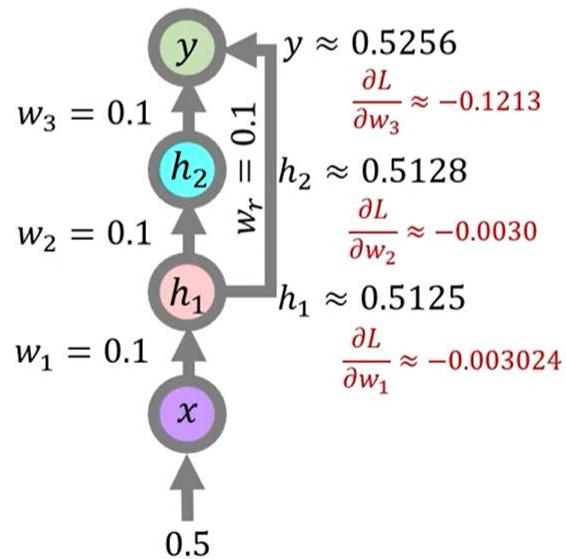
시나리오1: 그냥 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2372$$



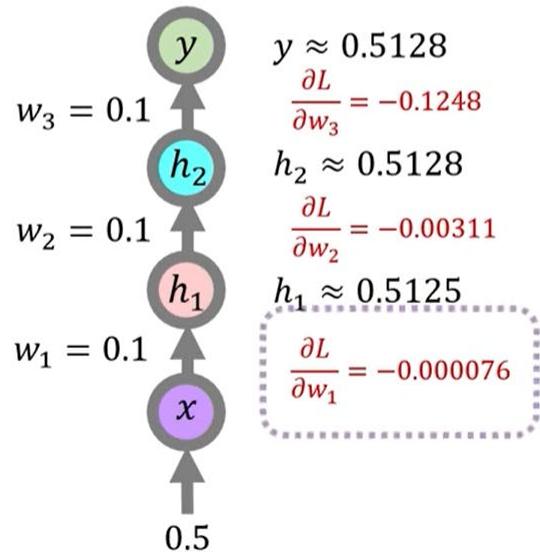
시나리오2: 잔차 신경망
(활성화함수는 시그모이드)

$$\text{loss} \approx 0.2247 \quad \frac{\partial L}{\partial w_1} = -0.00295 - 0.000074 = 0.003024$$

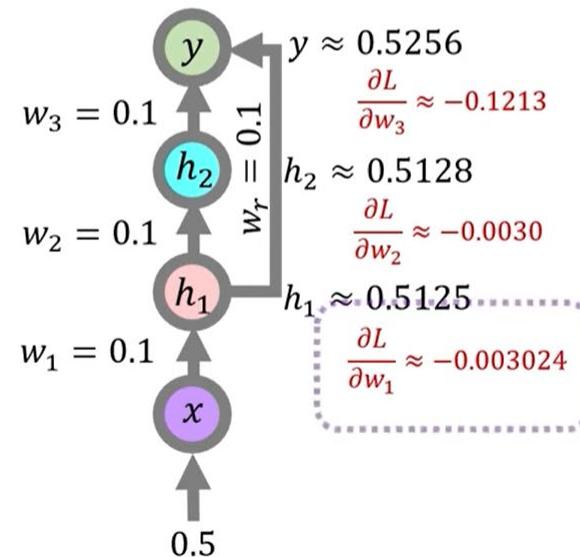


3.6. 잔차 연결(Residual Connection) - 비교

시나리오1: 그냥 신경망
(활성화함수는 시그모이드)



시나리오2: 잔차 신경망
(활성화함수는 시그모이드)



3.7. Decoder

```

class Decoder(nn.Module):
    def __init__(self, d_model, head_num, dropout):
        super(Decoder, self).__init__()
        self.masked_multi_head_attention = MultiHeadAttention(d_model=d_model, head_num=head_num)
        self.residual_1 = ResidualConnection(d_model, dropout=dropout)

        self.encoder_decoder_attention = MultiHeadAttention(d_model=d_model, head_num=head_num)
        self.residual_2 = ResidualConnection(d_model, dropout=dropout)

        self.feed_forward = FeedForward(d_model)
        self.residual_3 = ResidualConnection(d_model, dropout=dropout)

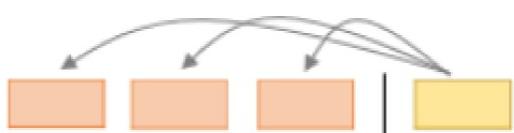
    def forward(self, target, encoder_output, target_mask, encoder_mask):
        # target, x, target_mask, input_mask
        x = self.residual_1(target, lambda x: self.masked_multi_head_attention(x, x, x, target_mask))
        x = self.residual_2(x, lambda x: self.encoder_decoder_attention(x, encoder_output, encoder_output, encoder_mask))
        x = self.residual_3(x, self.feed_forward)

        return x

```

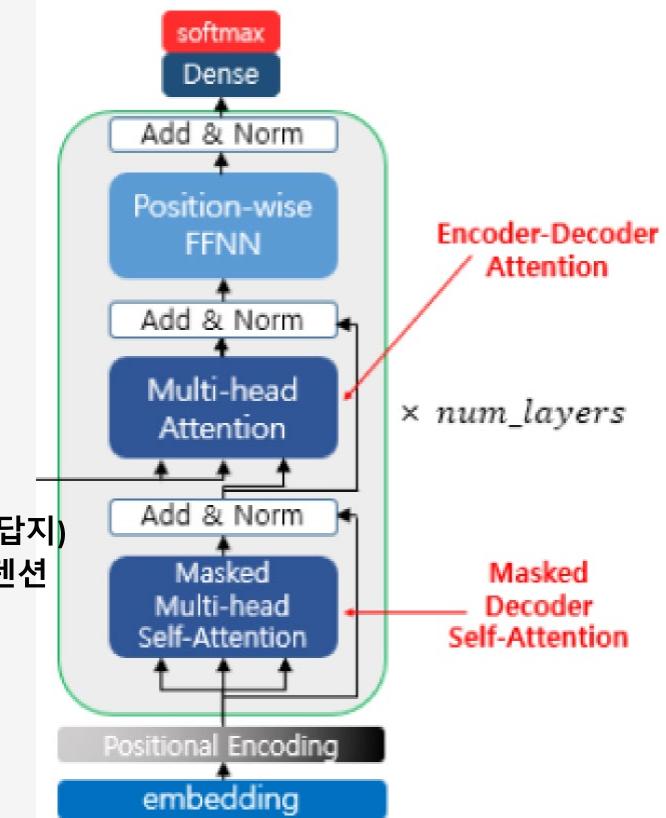
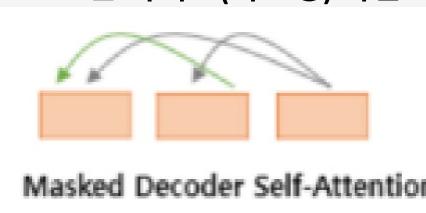
[인코더-디코더 어텐션]

- 디코더와 모든 인코더의 벡터를 어텐션



[마스크드 디코더 어텐션]

- 디코더에서 생성토큰의 뒷부분(정답지)는 가리고(마스킹) 이전 벡터와 어텐션



References

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, “Attention is all you need,” in Proc. Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 5998–6008.
- [2] W. Docs, “Attention is all you need - 트랜스포머 (한글 설명),” WikiDocs, 2019. [Online]. Available: <https://wikidocs.net/31379> [Accessed: Sep. 25, 2025].
- [3] 신박Ai, “[Deep Learning 101] ResNet 잔차신경망이란? (feat.기울기소실문제),” YouTube, Jul. 27, 2024. [Online]. Available: https://www.youtube.com/watch?v=Yf_-bj2Zaq4 [Accessed: Sep. 29, 2025].

감사합니다