DATA
SCIENCE LAB

Transformer 2

23.03.02 / 8기 조찬형

0. INTRO

Basic Architecture	층별로 살펴보며 복습하는 Transformer Code! (Ref. Attention is all you need(2017)) 이번 세션이 끝나면 여러분의 Task에 맞는 트랜스포머 모델을 만들 수 있도록.
Data Loader	딥러닝 전반에 사용할 수 있는 하이퍼파라미터 튜닝 / 여러가지 기법 코드 설명
Simple Copy Model	응용(1) – Copy Tasks. 작은 단어군에서 나온 인풋을 복제해서 반출하는 형태의 트랜스포머.

0. Preview

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

0. Preview

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

복사-메커니즘과 추론 단계의 페널티를 이용한 Copy-Transformer 기반 문서 생성 요약

전동현○, 강인호
네이버
{donghyeon.jeon, once.ihkang}@navercorp.com

Copy-Transformer model using Copy-Mechanism and Inference Penalty for Document Abstractive Summarization

Donghyeon-Jeon, In-Ho Kang
Naver Corporation

CONTENTS

0. Prelims

Architecture on Paper

논문 재현 + 코드적 모델 이해에 목적을 두고 진행.

Prelims는 함께 배포된 노트북에서 확인.

코드 refer : 1.

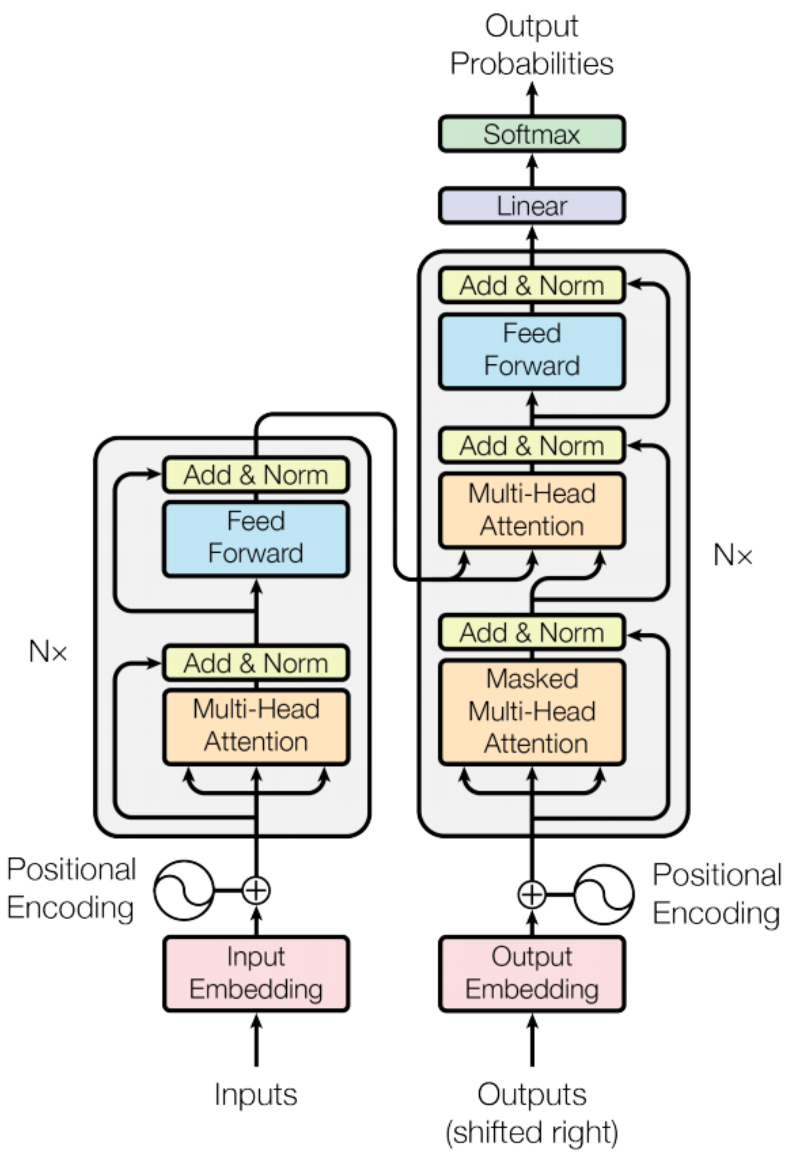


Figure 1: The Transformer - model architecture.

0. Prelims

Architecture on Paper

```python
def is_interactive_notebook():
    return __name__ == "__main__"


def show_example(fn, args=[]):
    if __name__ == "__main__" and RUN_EXAMPLES:
        return fn(*args)


def execute_example(fn, args=[]):
    if __name__ == "__main__" and RUN_EXAMPLES:
        fn(*args)


class DummyOptimizer(torch.optim.Optimizer):
    def __init__(self):
        self.param_groups = [{"lr": 0}]
        None

    def step(self):
        None

    def zero_grad(self, set_to_none=False):
        None


class DummyScheduler:
    def step(self):
        None
```

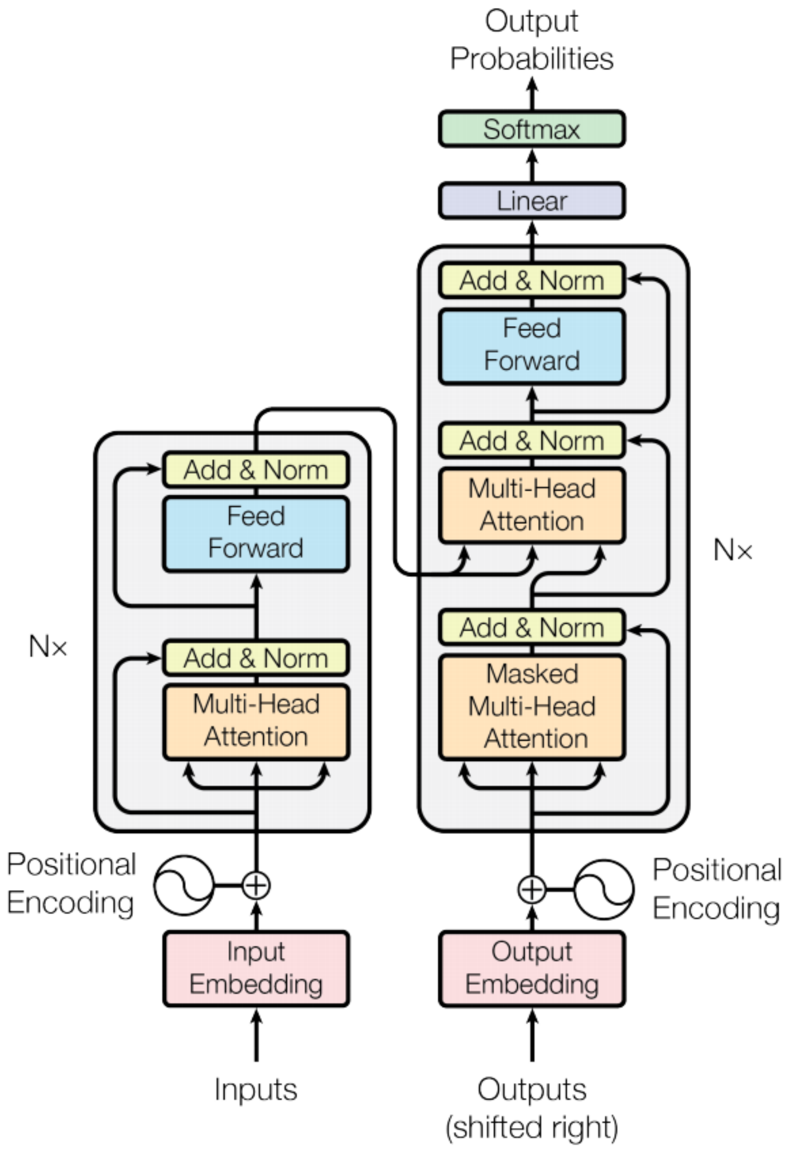
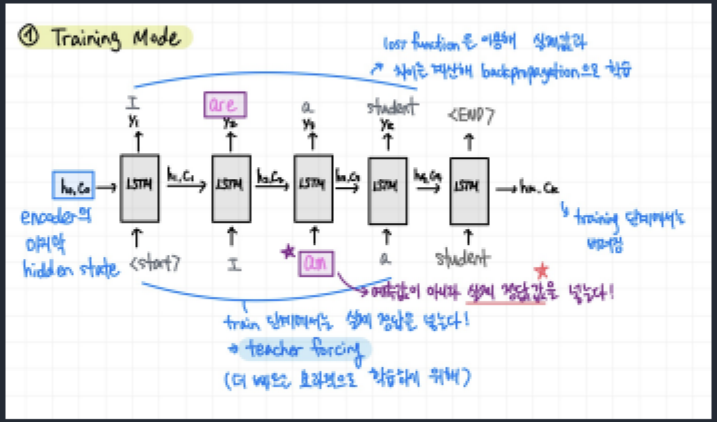


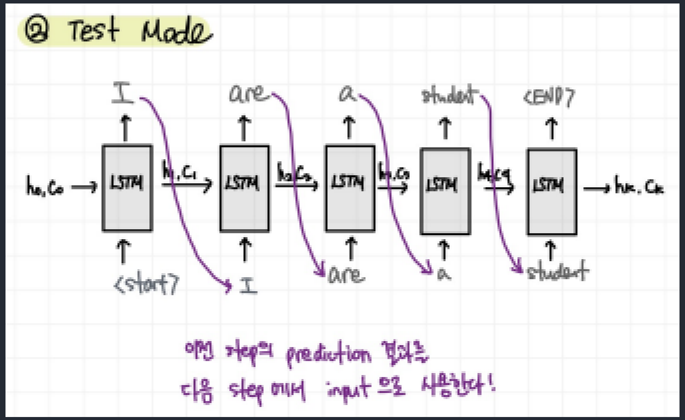
Figure 1: The Transformer - model architecture.

1. Basic Architecture

Architecture on Paper

Encoder, Decoder 의 학습 방법





<Decoder Stage>

- Decoder는 encoder에서 넘겨받은 hidden state로 자신을 초기화한다
- Decoder 는 Train mode, Test mode 작동 방식이 다르다
- Train시에는 실제 정답을 넣는 teacher forcing을 사용(더 빠르게 학습하기 위해), Test시는 이전 step의 prediction 결과를 다음 step에서 input으로 사용

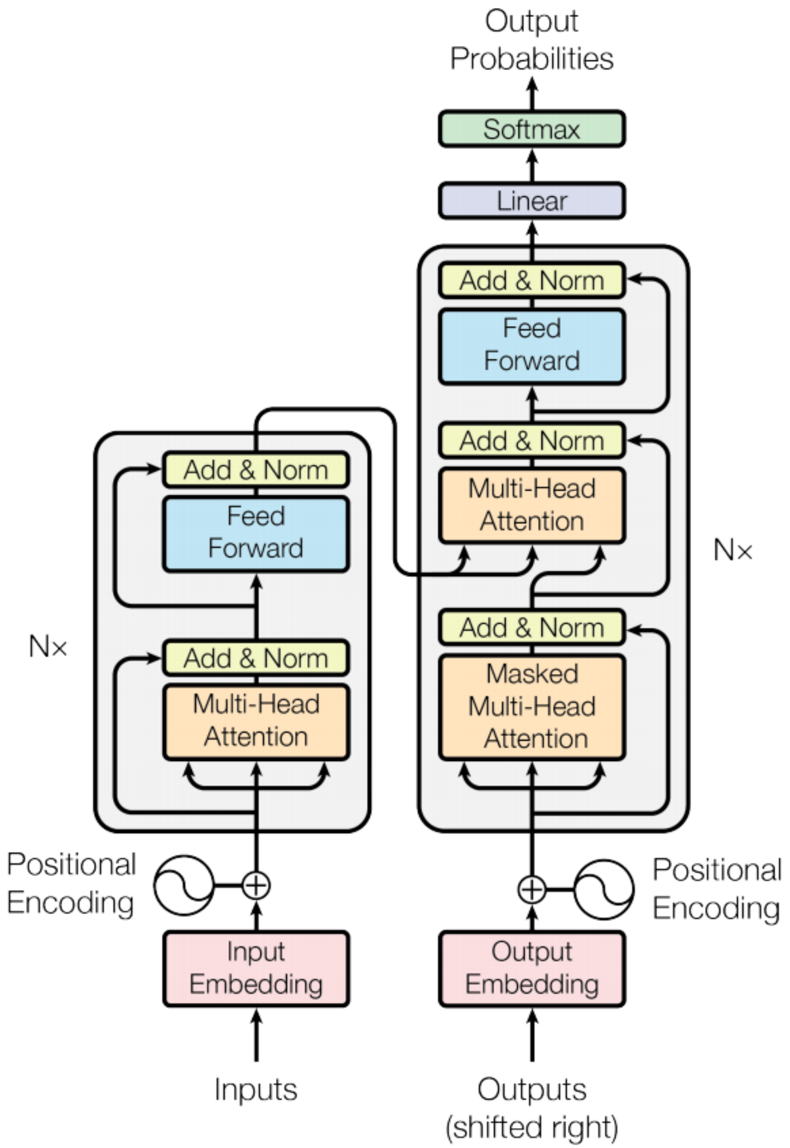


Figure 1: The Transformer - model architecture.

1. Basic Architecture

클래스 생성 및 기본 인코더-디코더 구조 할당

Architecture on Paper

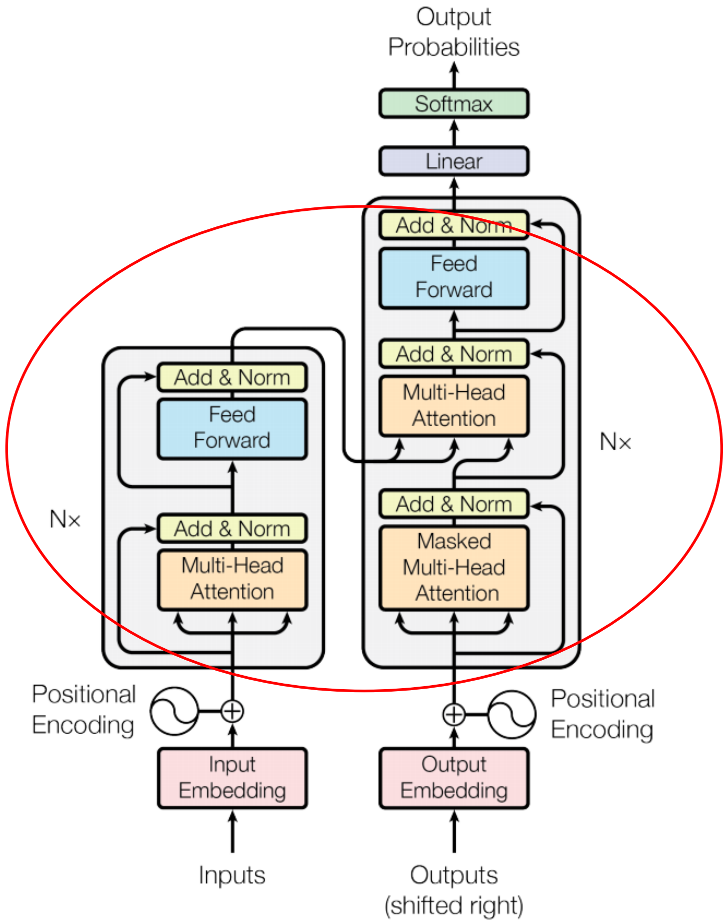


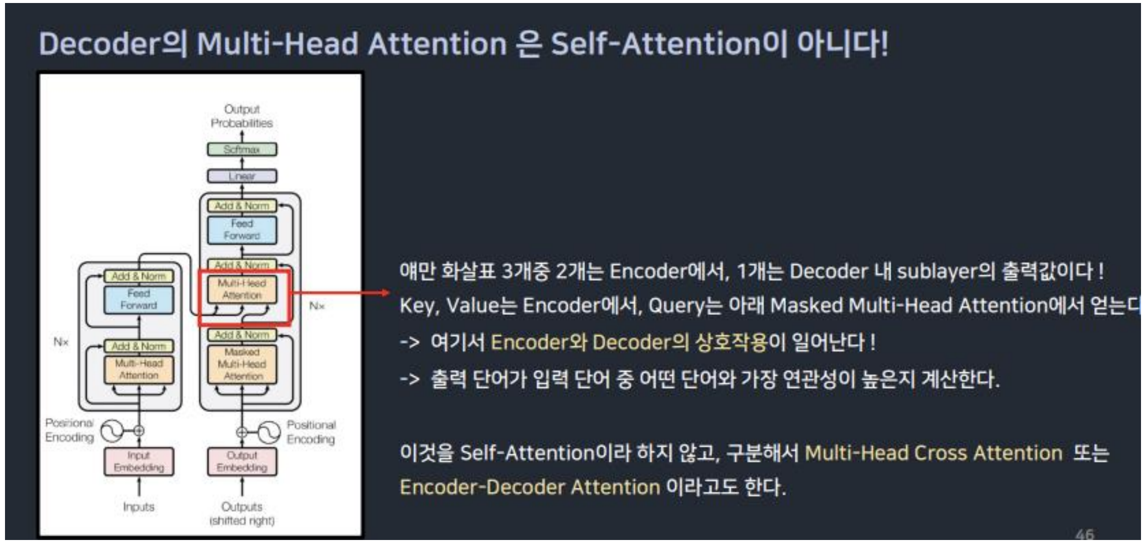
Figure 1: The Transformer - model architecture.

```
[6]  class EncoderDecoder(nn.Module):
         """
         A standard Encoder-Decoder architecture. Base for this and many
         other models.
         """

         def __init__(self, encoder, decoder, src_embed, tgt_embed, generator):
             super(EncoderDecoder, self).__init__()
             self.encoder = encoder
             self.decoder = decoder
             self.src_embed = src_embed
             self.tgt_embed = tgt_embed
             self.generator = generator

         def forward(self, src, tgt, src_mask, tgt_mask):
             "Take in and process masked src and target sequences."
             return self.decode(self.encode(src, src_mask), src_mask, tgt, tgt_mask)

         def encode(self, src, src_mask):
             return self.encoder(self.src_embed(src), src_mask)

         def decode(self, memory, src_mask, tgt, tgt_mask):
             return self.decoder(self.tgt_embed(tgt), memory, src_mask, tgt_mask)
```

1. Basic Architecture

클래스 생성 및 기본 인코더-디코더 구조 할당

Architecture on Paper




```
[6] class EncoderDecoder(nn.Module):
        """
        A standard Encoder-Decoder architecture. Base for this and many
        other models.
        """

        def __init__(self, encoder, decoder, src_embed, tgt_embed, generator):
            super(EncoderDecoder, self).__init__()
            self.encoder = encoder
            self.decoder = decoder
            self.src_embed = src_embed
            self.tgt_embed = tgt_embed
            self.generator = generator

        def forward(self, src, tgt, src_mask, tgt_mask):
            "Take in and process masked src and target sequences."
            return self.decode(self.encode(src, src_mask), src_mask, tgt, tgt_mask)

        def encode(self, src, src_mask):
            return self.encoder(self.src_embed(src), src_mask)

        def decode(self, memory, src_mask, tgt, tgt_mask):
            return self.decoder(self.tgt_embed(tgt), memory, src_mask, tgt_mask)
```

1. Basic Architecture

Generator class

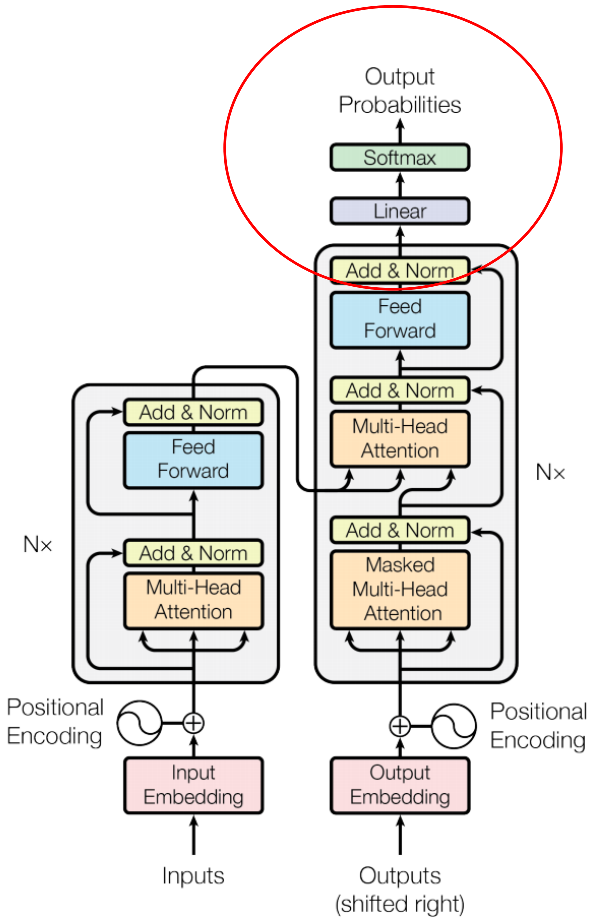


Figure 1: The Transformer - model architecture.

```python
class Generator(nn.Module):
    "Define standard linear + softmax generation step."

    def __init__(self, d_model, vocab):
        super(Generator, self).__init__()
        self.proj = nn.Linear(d_model, vocab)

    def forward(self, x):
        return log_softmax(self.proj(x), dim=-1)
```

1. Basic Architecture

Clone by deepcopy

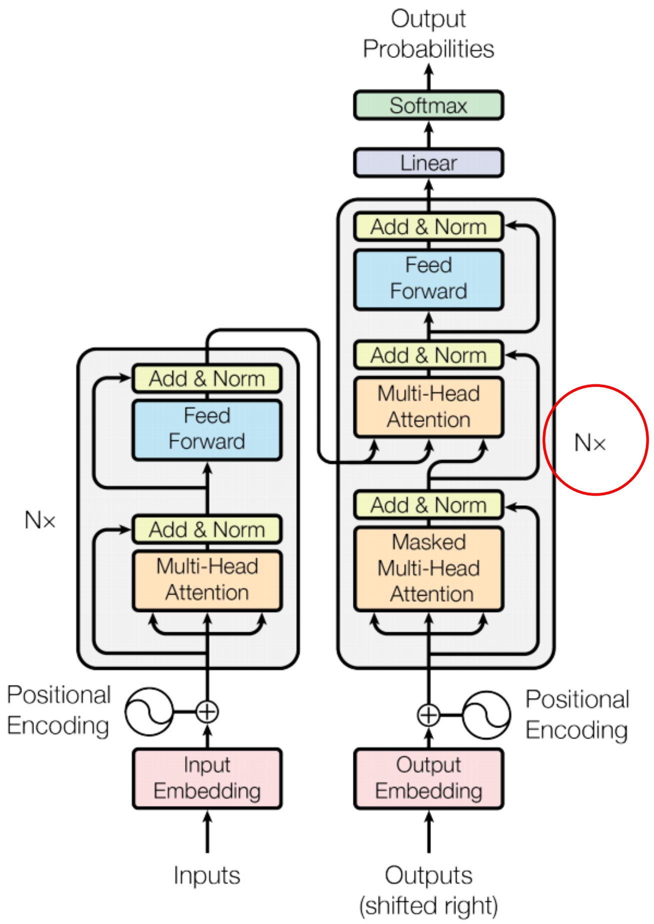


Figure 1: The Transformer - model architecture.

```
def clones(module, N):
    "Produce N identical layers."
    return nn.ModuleList([copy.deepcopy(module) for _ in range(N)])
```

1. Basic Architecture

Encoder code

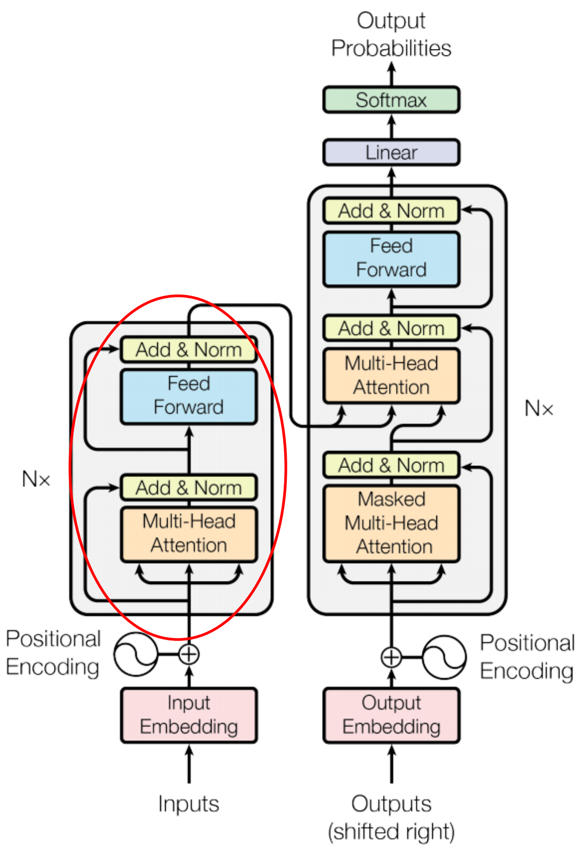


Figure 1: The Transformer - model architecture.

```python
class Encoder(nn.Module):
    "Core encoder is a stack of N layers"

    def __init__(self, layer, N):
        super(Encoder, self).__init__()
        self.layers = clones(layer, N)
        self.norm = LayerNorm(layer.size)

    def forward(self, x, mask):
        "Pass the input (and mask) through each layer in turn."
        for layer in self.layers:
            x = layer(x, mask)
        return self.norm(x)
```

1. Basic Architecture

LayerNorm

Refer: 2번

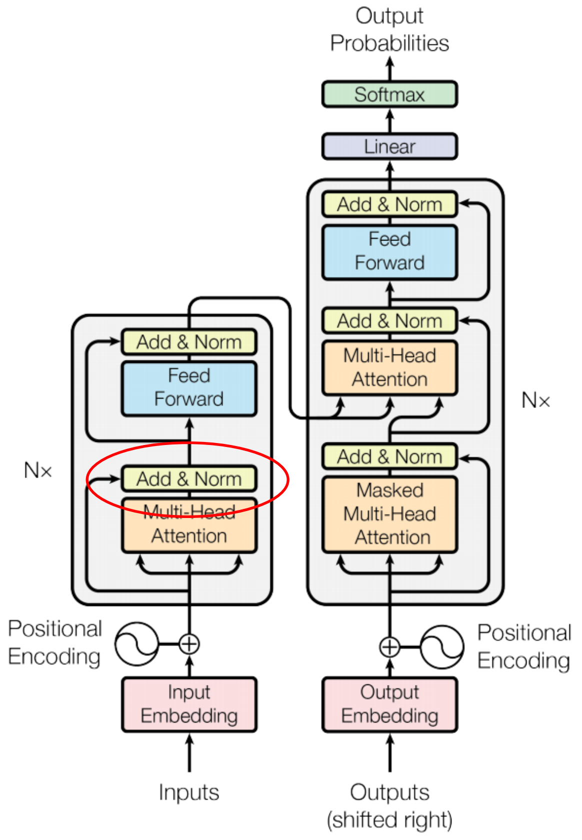
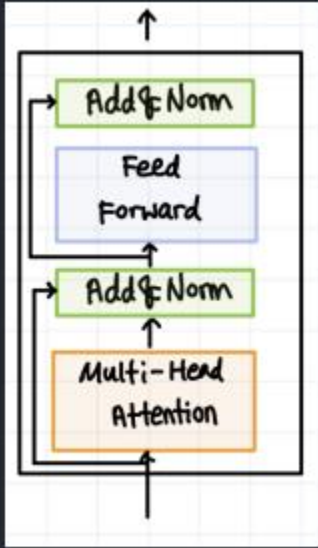


Figure 1: The Transformer - model architecture.

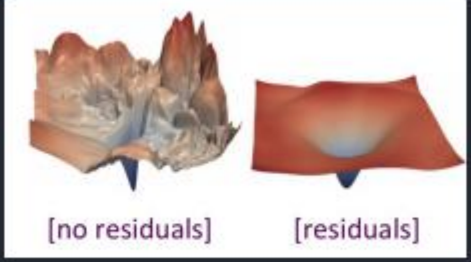
Add & Norm



Add

- Multi-Head Attention 의 입력값과 출력값을 더한다.
- Feed Forward의 입력값과 출력값을 더한다.

-> Residual Connection : $X^l = X^{l-1} + Layer(X^{l-1})$



Norm

- Layer Normalization은 각 layer값이 크게 변화는 것을 방지해 모델이 더 빠르게 학습할 수 있게 한다.

$$output = LayerNorm(x + sublayer(x))$$

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l$$

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}$$

1. Basic Architecture

LayerNorm code

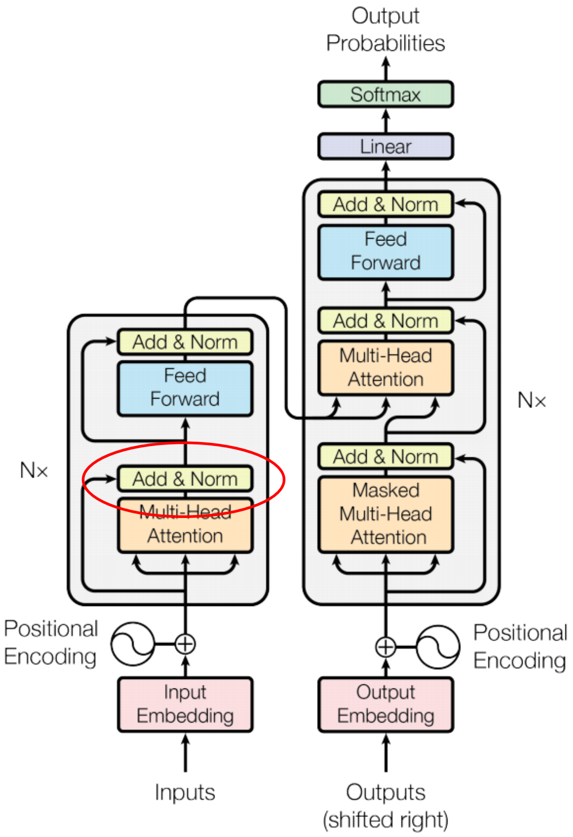


Figure 1: The Transformer - model architecture.

```
class LayerNorm(nn.Module):
    "Construct a layernorm module (See citation for details)."

    def __init__(self, features, eps=1e-6):
        super(LayerNorm, self).__init__()
        self.a_2 = nn.Parameter(torch.ones(features))
        self.b_2 = nn.Parameter(torch.zeros(features))
        self.eps = eps

    def forward(self, x):
        mean = x.mean(-1, keepdim=True)
        std = x.std(-1, keepdim=True)
        return self.a_2 * (x - mean) / (std + self.eps) + self.b_2
```

1. Basic Architecture

Sublayer

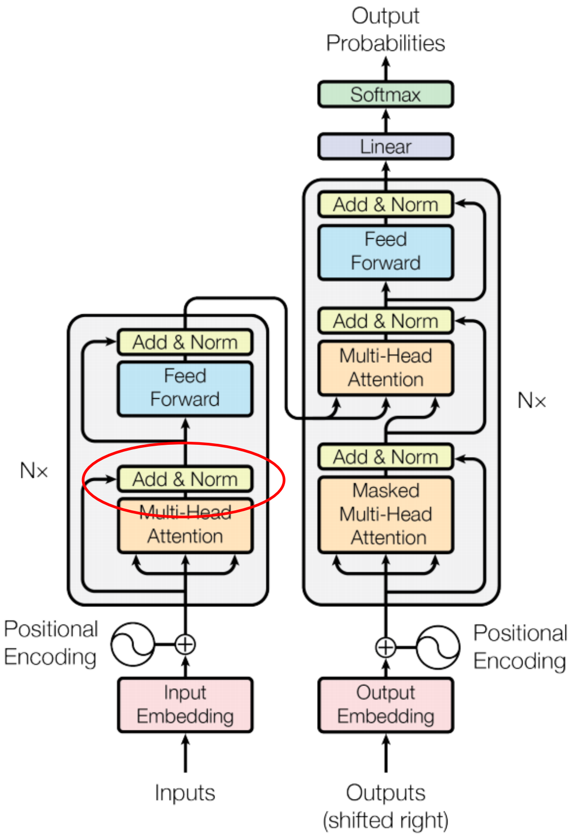
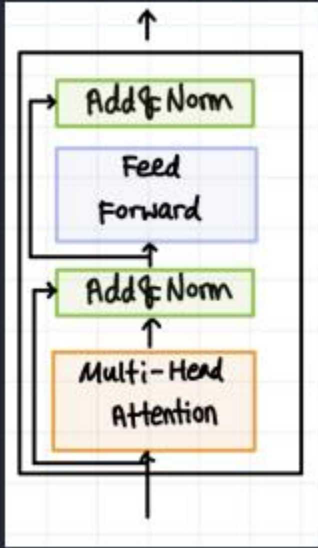


Figure 1: The Transformer - model architecture.

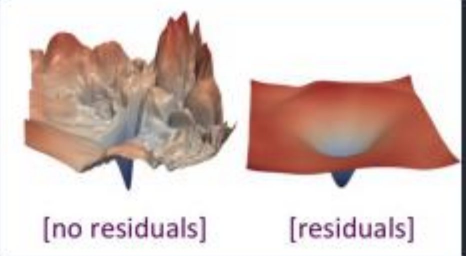
Add & Norm



Add

- Multi-Head Attention 의 입력값과 출력값을 더한다.
- Feed Forward의 입력값과 출력값을 더한다.

-> Residual Connection : $X^l = X^{l-1} + Layer(X^{l-1})$



Norm

- Layer Normalization은 각 layer값이 크게 변화는 것을 방지해 모델이 더 빠르게 학습할 수 있게 한다.

$$output = LayerNorm(x + sublayer(x))$$

1. Basic Architecture

Sublayer connection code

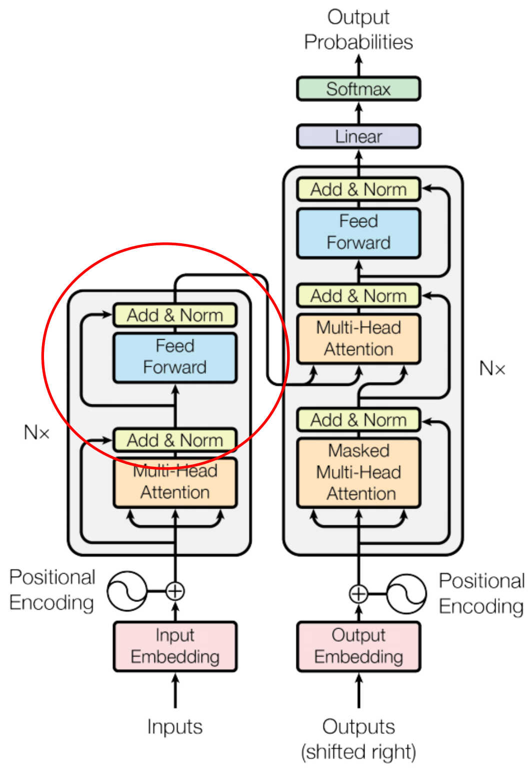
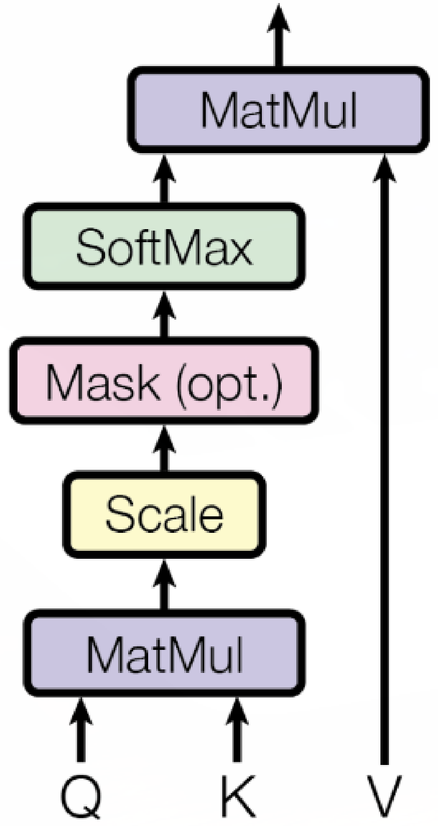


Figure 1: The Transformer - model architecture.

```python
class SublayerConnection(nn.Module):
    """
    A residual connection followed by a layer norm.
    Note for code simplicity the norm is first as opposed to last.
    """

    def __init__(self, size, dropout):
        super(SublayerConnection, self).__init__()
        self.norm = LayerNorm(size)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x, sublayer):
        "Apply residual connection to any sublayer with the same size."
        return x + self.dropout(sublayer(self.norm(x)))
```

1. Basic Architecture

Self Attention

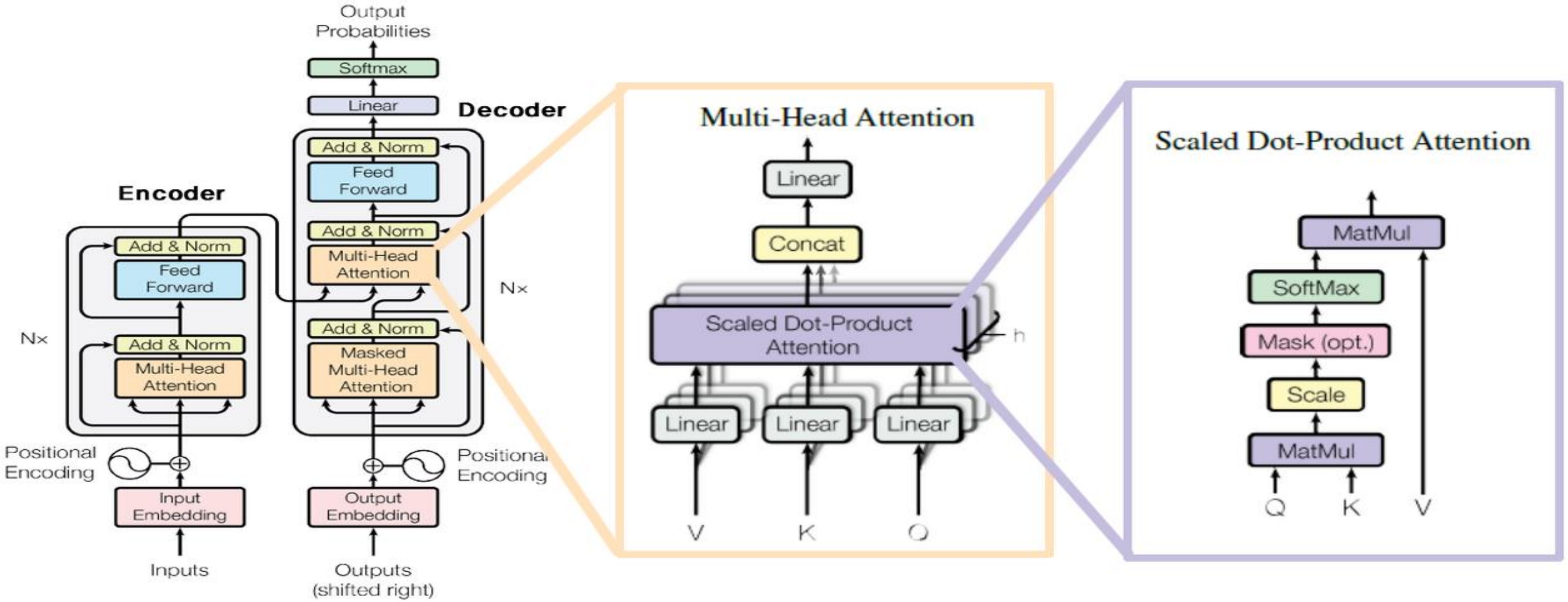


$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

```python
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    d_k = query.size(-1)
    scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(d_k)
    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)
    p_attn = scores.softmax(dim=-1)
    if dropout is not None:
        p_attn = dropout(p_attn)
    return torch.matmul(p_attn, value), p_attn
```

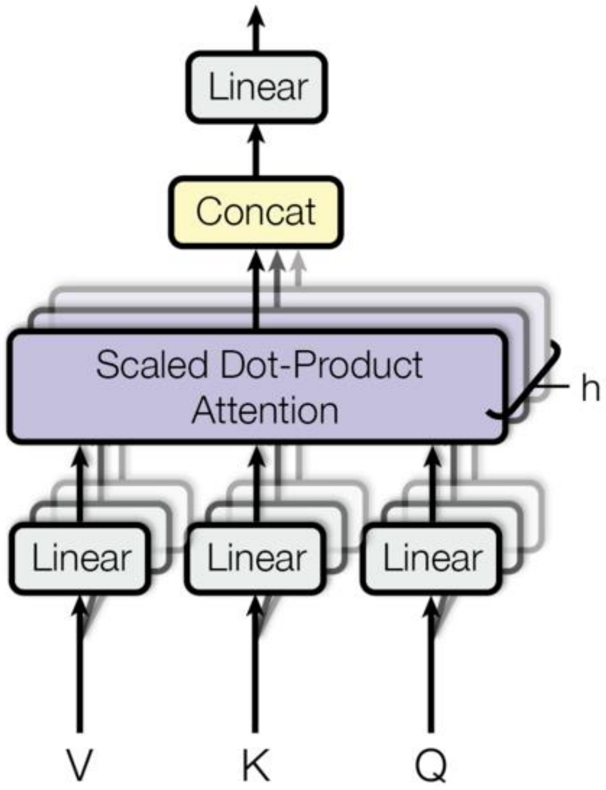
1. Basic Architecture

Architecture on Paper



1. Basic Architecture

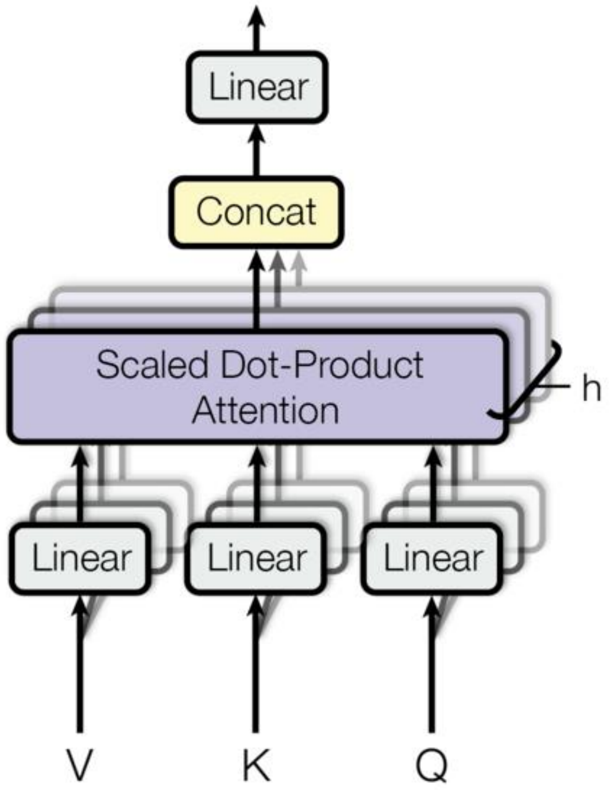
Multi-Head Attention




```python
class MultiHeadedAttention(nn.Module):
    def __init__(self, h, d_model, dropout=0.1):
        "Take in model size and number of heads."
        super(MultiHeadedAttention, self).__init__()
        assert d_model % h == 0
        # We assume d_v always equals d_k
        self.d_k = d_model // h
        self.h = h
        self.linears = clones(nn.Linear(d_model, d_model), 4)
        self.attn = None
        self.dropout = nn.Dropout(p=dropout)

    def forward(self, query, key, value, mask=None):
        "Implements Figure 2"
        if mask is not None:
            # Same mask applied to all h heads.
            mask = mask.unsqueeze(1)
        nbatches = query.size(0)
```

1. Basic Architecture

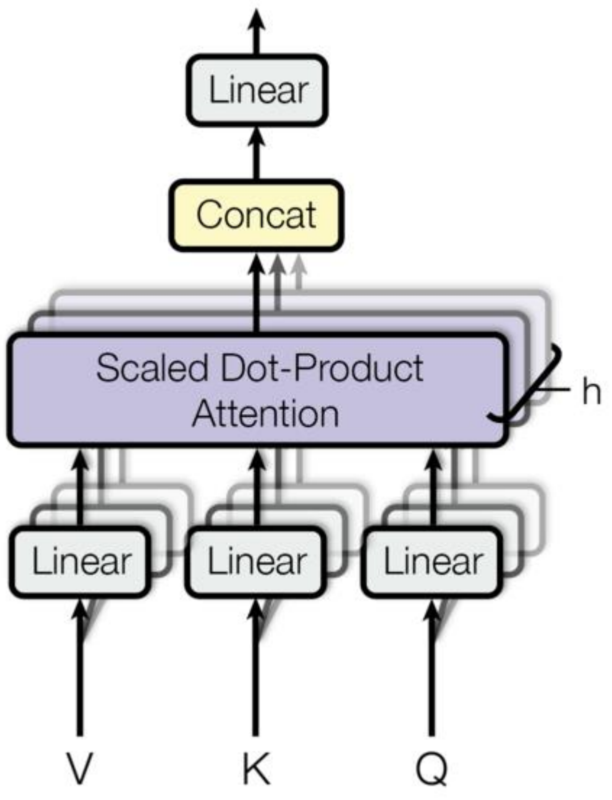
Multi-Head Attention

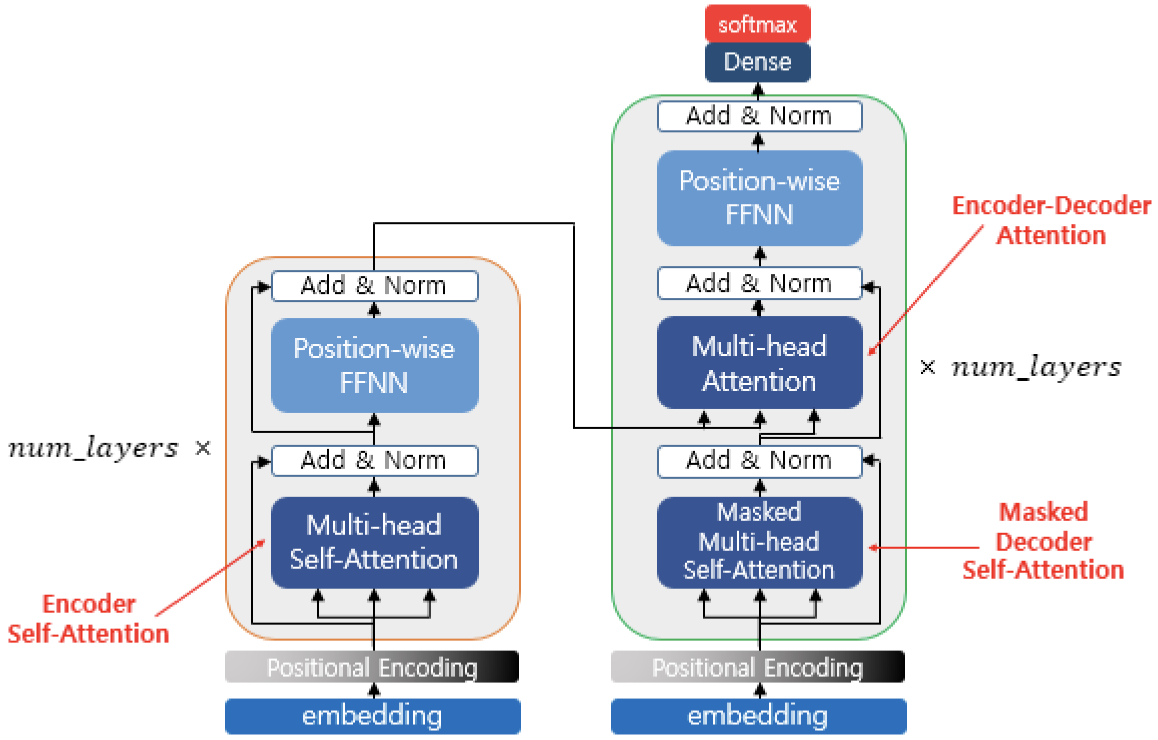



```python
# 1) Do all the linear projections in batch from d_model => h x d_k
query, key, value = [
    lin(x).view(nbatches, -1, self.h, self.d_k).transpose(1, 2)
    for lin, x in zip(self.linears, (query, key, value))
]

# 2) Apply attention on all the projected vectors in batch.
x, self.attn = attention(
    query, key, value, mask=mask, dropout=self.dropout
)

# 3) "Concat" using a view and apply a final linear.
x = (
    x.transpose(1, 2)
    .contiguous()
    .view(nbatches, -1, self.h * self.d_k)
)
del query
del key
del value
return self.linears[-1](x)
```

1. Basic Architecture

Multi-Head Attention





1. Basic Architecture

Encoder code

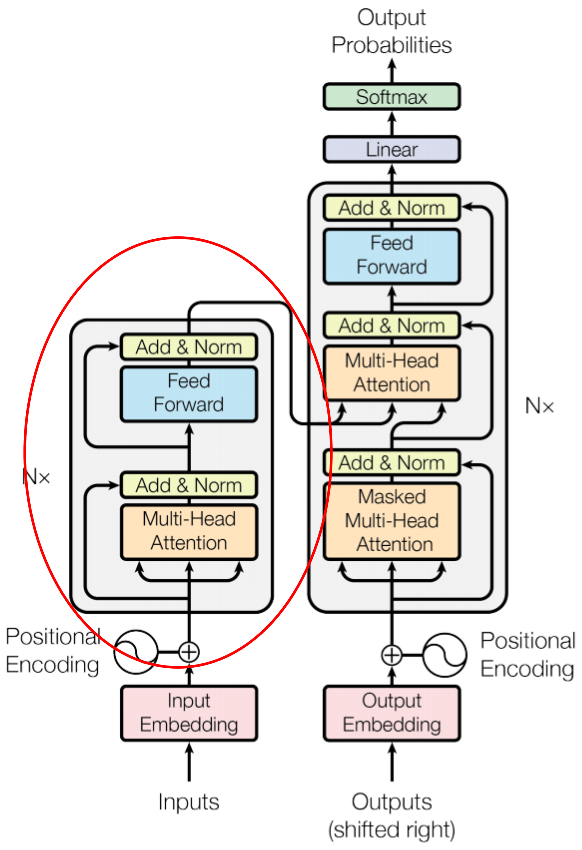


Figure 1: The Transformer - model architecture.

```
class EncoderLayer(nn.Module):
    "Encoder is made up of self-attn and feed forward (defined below)"

    def __init__(self, size, self_attn, feed_forward, dropout):
        super(EncoderLayer, self).__init__()
        self.self_attn = self_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 2)
        self.size = size

    def forward(self, x, mask):
        "Follow Figure 1 (left) for connections."
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, mask))
        return self.sublayer[1](x, self.feed_forward)
```

1. Basic Architecture

Decoder code

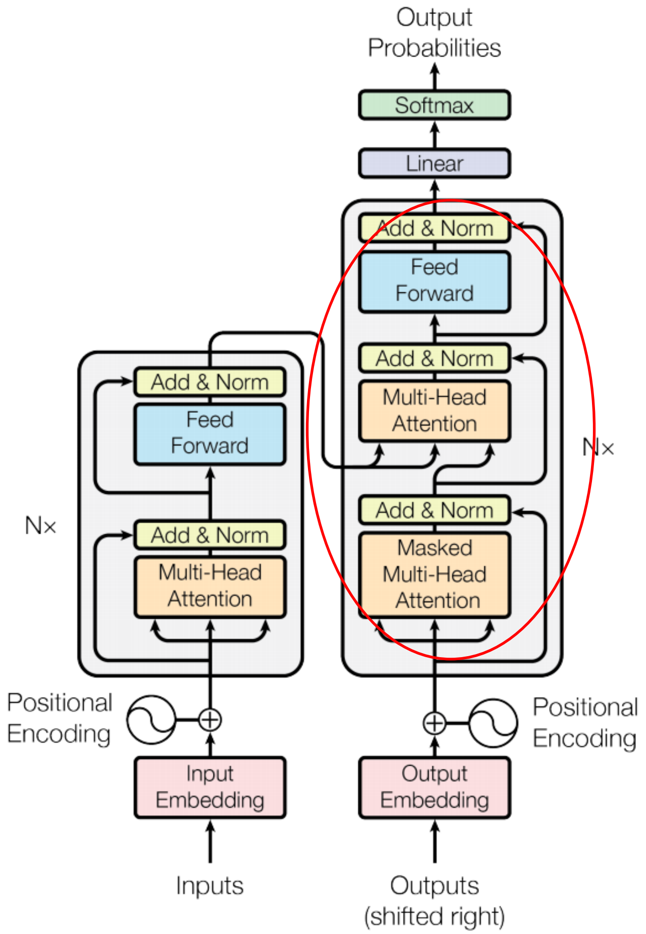


Figure 1: The Transformer - model architecture.

```
class Decoder(nn.Module):
    "Generic N layer decoder with masking."

    def __init__(self, layer, N):
        super(Decoder, self).__init__()
        self.layers = clones(layer, N)
        self.norm = LayerNorm(layer.size)

    def forward(self, x, memory, src_mask, tgt_mask):
        for layer in self.layers:
            x = layer(x, memory, src_mask, tgt_mask)
        return self.norm(x)
```

1. Basic Architecture

Decoder code

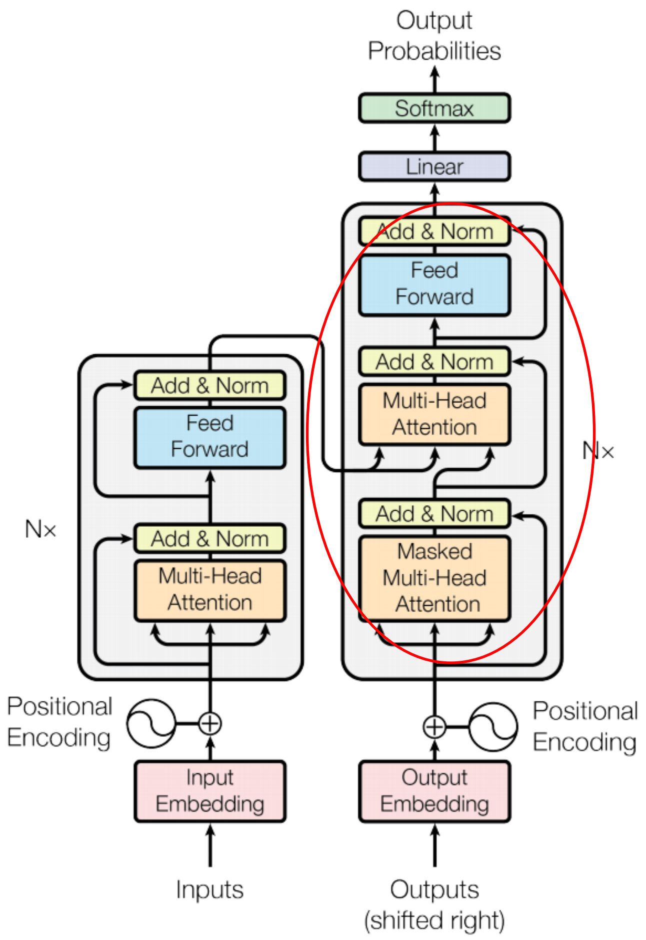
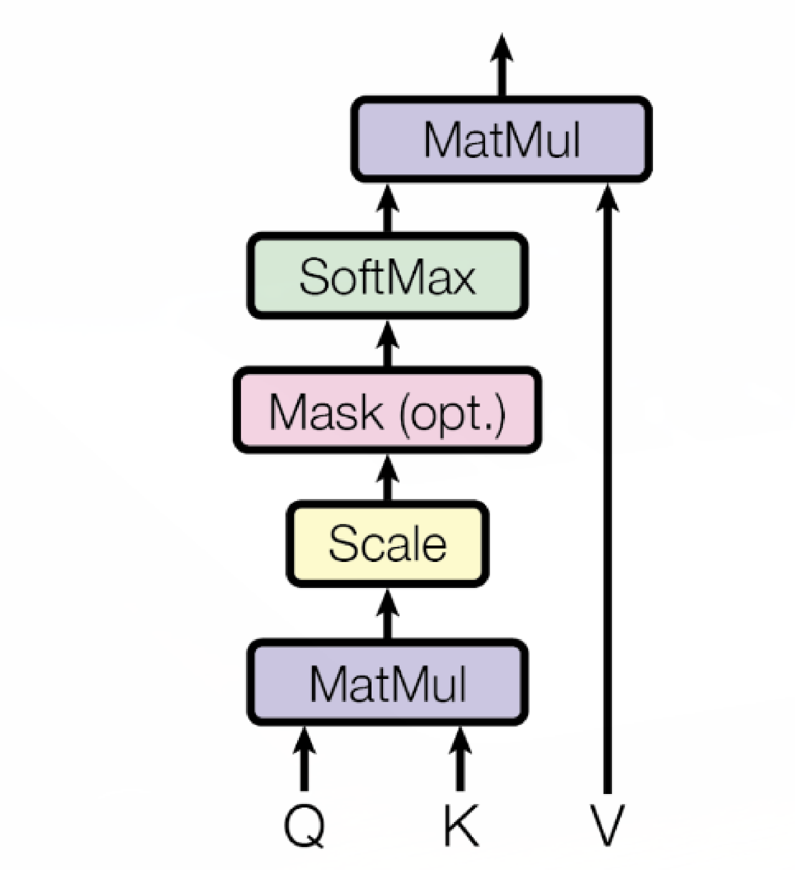


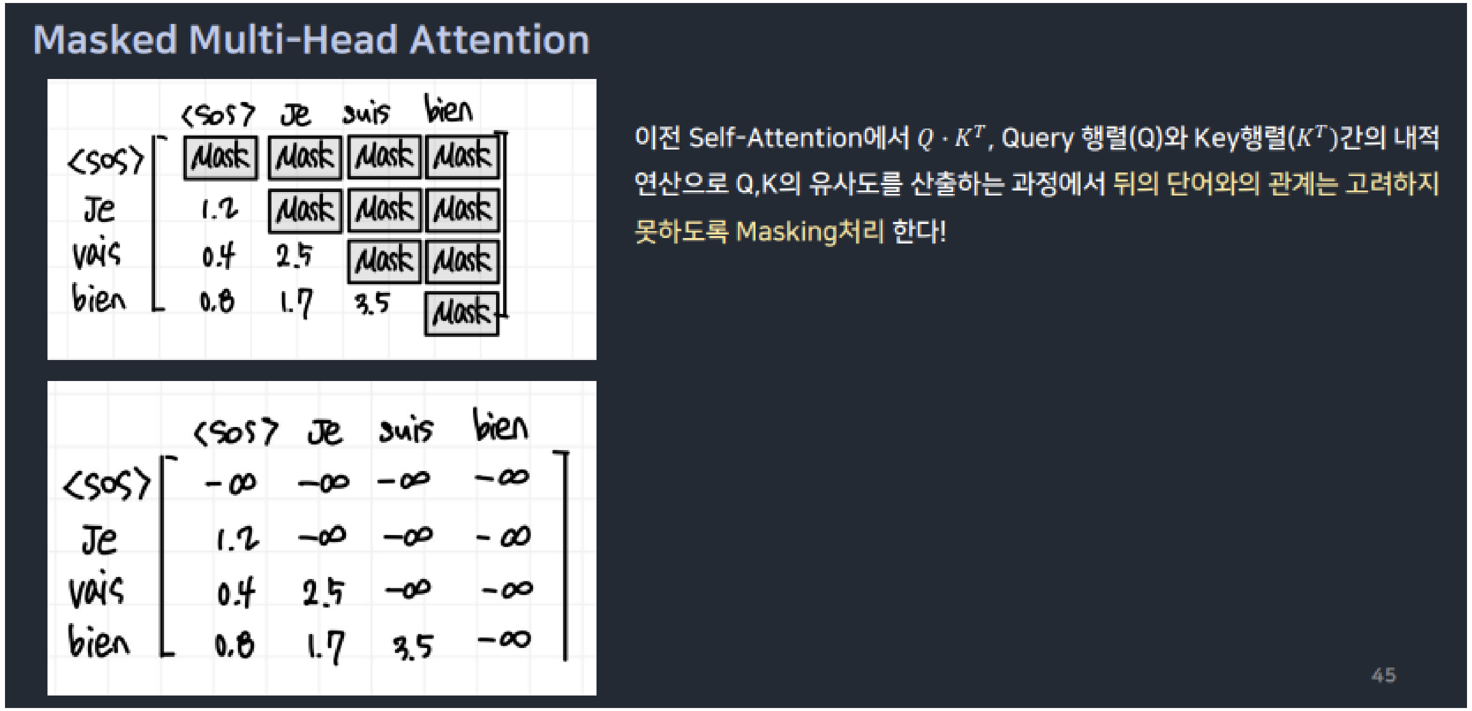
Figure 1: The Transformer - model architecture.

```
class DecoderLayer(nn.Module):
    "Decoder is made of self-attn, src-attn, and feed forward (defined below)"

    def __init__(self, size, self_attn, src_attn, feed_forward, dropout):
        super(DecoderLayer, self).__init__()
        self.size = size
        self.self_attn = self_attn
        self.src_attn = src_attn
        self.feed_forward = feed_forward
        self.sublayer = clones(SublayerConnection(size, dropout), 3)

    def forward(self, x, memory, src_mask, tgt_mask):
        "Follow Figure 1 (right) for connections."
        m = memory
        x = self.sublayer[0](x, lambda x: self.self_attn(x, x, x, tgt_mask))
        x = self.sublayer[1](x, lambda x: self.src_attn(x, m, m, src_mask))
        return self.sublayer[2](x, self.feed_forward)
```

1. Basic Architecture

Subsequent mask





1. Basic Architecture

Mask code

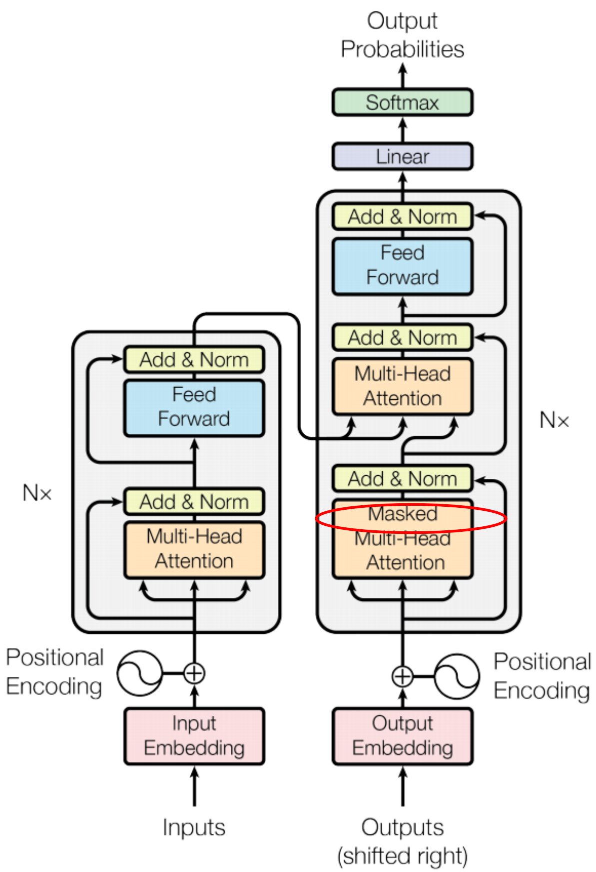


Figure 1: The Transformer - model architecture.

```
def subsequent_mask(size):
    "Mask out subsequent positions."
    attn_shape = (1, size, size)
    subsequent_mask = torch.triu(torch.ones(attn_shape), diagonal=1).type(
        torch.uint8
    )
    return subsequent_mask == 0
```

1. Basic Architecture

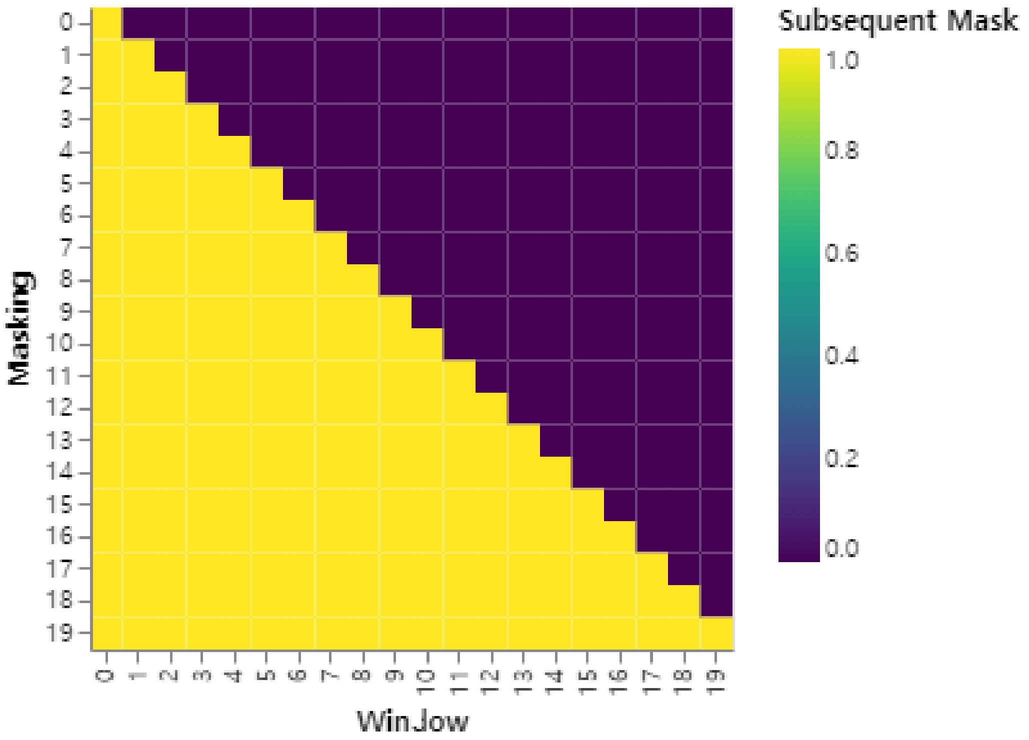
Mask example

```
def example_mask():
    LS_data = pd.concat(
        [
            pd.DataFrame(
                {
                    "Subsequent Mask": subsequent_mask(20)[0][x, y].flatten(),
                    "Window": y,
                    "Masking": x,
                }
            )
            for y in range(20)
            for x in range(20)
        ]
    )

    return (
        alt.Chart(LS_data)
        .mark_rect()
        .properties(height=250, width=250)
        .encode(
            alt.X("Window:O"),
            alt.Y("Masking:O"),
            alt.Color("Subsequent Mask:Q", scale=alt.Scale(scheme="viridis")),
        )
        .interactive()
    )


show_example(example_mask)
```

1. Basic Architecture

Mask example

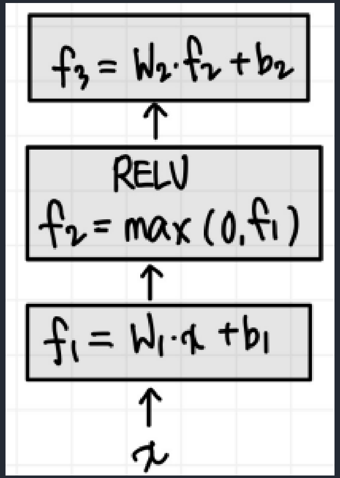
```
def example_mask():
    LS_data = pd.concat(
        [
            pd.DataFrame(
                {
                    "Subsequent Mask": subsequent_mask(20)[0][x, y].flatten(),
                    "Window": y,
                    "Masking": x,
                }
            )
            for y in range(20)
            for x in range(20)
        ]
    )

    return (
        alt.Chart(LS_data)
        .mark_rect()
        .properties(height=250, width=250)
        .encode(
            alt.X("Window:O"),
            alt.Y("Masking:O"),
            alt.Color("Subsequent Mask:Q", scale=alt.Scale(scheme="viridis")),
        )
        .interactive()
    )


show_example(example_mask)
```



1. Basic Architecture

Position wise FeedForward

Feed Forward Network



RELU 함수로 non-linearity(비선형성) 을 더해준다!

-> 선형 함수를 계속해서 쌓으면, 결국 한개의 선형 함수로 나타내는 것과 마찬가지이다. 그러면 layer를 쌓는 의미가 없다.

$$y_1 = ax_1$$
$$y_2 = a(y_1) = a^2x_1$$
$$y_3 = a(y_2) = a^3x_1$$
$$y_n = bx_1, b = a^n$$

```python
class PositionwiseFeedForward(nn.Module):
    "Implements FFN equation."

    def __init__(self, d_model, d_ff, dropout=0.1):
        super(PositionwiseFeedForward, self).__init__()
        self.w_1 = nn.Linear(d_model, d_ff)
        self.w_2 = nn.Linear(d_ff, d_model)
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        return self.w_2(self.dropout(self.w_1(x).relu()))
```

1. Basic Architecture

Embedding

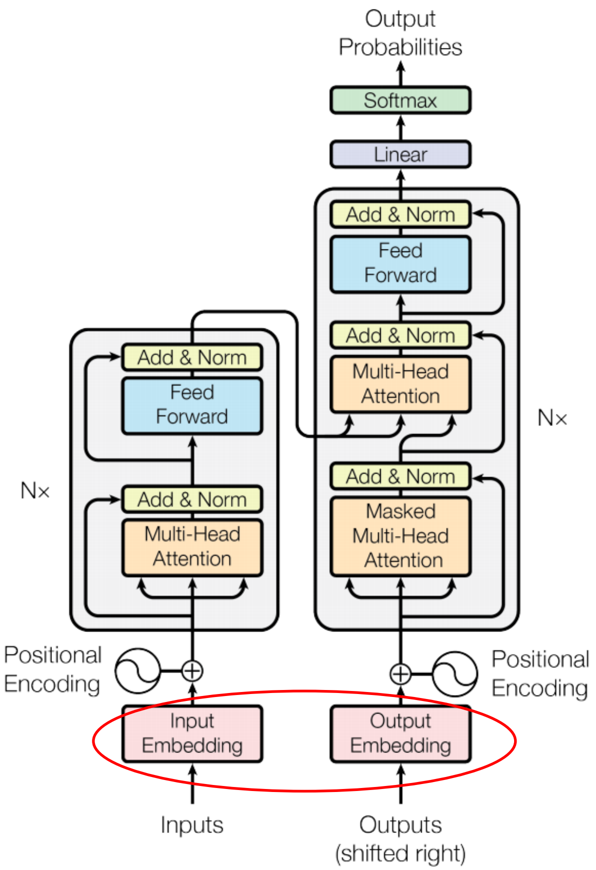


Figure 1: The Transformer - model architecture.

```
class Embeddings(nn.Module):
    def __init__(self, d_model, vocab):
        super(Embeddings, self).__init__()
        self.lut = nn.Embedding(vocab, d_model)
        self.d_model = d_model

    def forward(self, x):
        return self.lut(x) * math.sqrt(self.d_model)
```

1. Basic Architecture

Embedding Tools

In paper) 바이트 페어 인코딩(BPE)

빈번히 등장하는 쌍에 대한 unit 화
이후 하나의 유닛으로 만들어 줌.
음절과 어절 중간 정도의 단계로 최소
의미쌍을 형성하는 방법론.

Coffee 5회
Caffeine 6회
Heroine 1회

3개 단어의 빈도수,
음절쌍 출현율 고려해서 묶음

In sota models like BERT) Wordpiece

띄어쓰기를 표지로 하여 유닛을 만듦.
어절 단위로 이를 진행하기 때문에 한
국어에 맞지 않을 수 있음.

Coffee 5회
Caffeine 6회
Heroine 1회

1. Basic Architecture

Positional Encoding

1. 각 위치값은 시퀀스의 길이나 입력값에 관계없이 동일한 위치값을 가져야 한다.

2. 모든 위치값이 입력값에 비해 너무 크면 안된다.

3. Positional Encoding의 값의 증가가 너무 빠르면 안 된다.

4. 위치 차이에 의한 Positional Encoding값의 차이를 거리로 이용할 수 있어야 한다. 예를 들어 0번째, 1번째 Positional Encoding 값의 차이가 1번째, 2번째 Positional Encoding값의 차이와 유사해야 한다.

5. Positional Encoding 값은 위치에 따라 서로 다른 값을 가져야 한다. 위치 정보를 나타내는 만큼 서로 다른 값을 나타내어야 학습할 때 의미 있게 사용할 수 있다.

1. Basic Architecture

Positional Encoding

순차적이지 않은 투입 -> 상대 위치 정보 부재로 이어짐.
인코더와 디코더에 대한 입력 임베딩에 인코딩을 추가함으로 정보를 입력한다.

각 위치(pos)에 대한 사인 및 코사인 트랜스포메이션 사용함.

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

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

```python
class PositionalEncoding(nn.Module):
    "Implement the PE function."

    def __init__(self, d_model, dropout, max_len=5000):
        super(PositionalEncoding, self).__init__()
        self.dropout = nn.Dropout(p=dropout)

        # Compute the positional encodings once in log space.
        pe = torch.zeros(max_len, d_model)
        position = torch.arange(0, max_len).unsqueeze(1)
        div_term = torch.exp(
            torch.arange(0, d_model, 2) * -(math.log(10000.0) / d_model)
        )
        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)
        pe = pe.unsqueeze(0)
        self.register_buffer("pe", pe)

    def forward(self, x):
        x = x + self.pe[:, : x.size(1)].requires_grad_(False)
        return self.dropout(x)
```

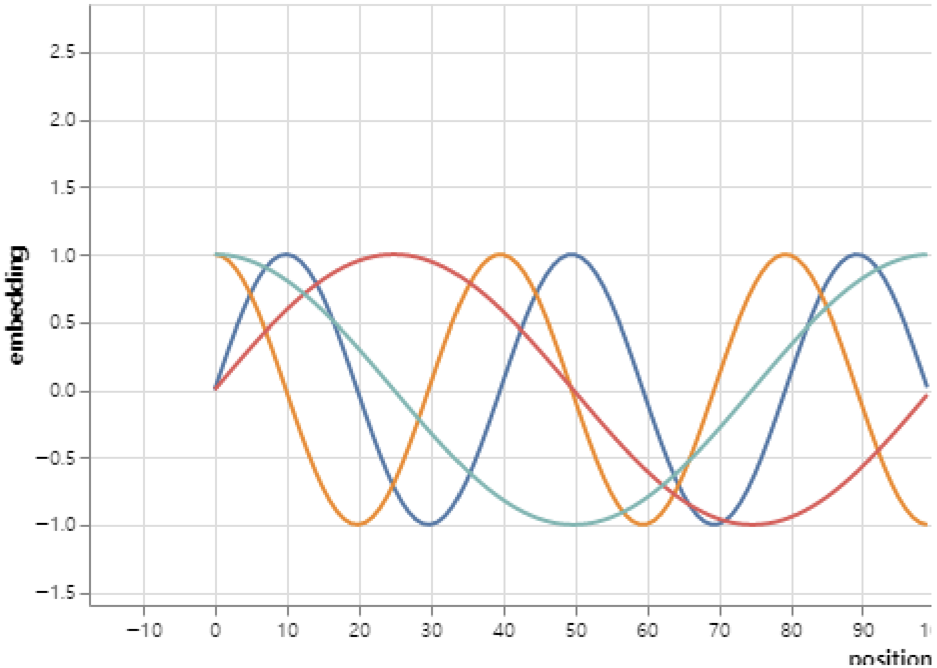
1. Basic Architecture

Positional Encoding

```python
def example_positional():
    pe = PositionalEncoding(20, 0)
    y = pe.forward(torch.zeros(1, 100, 20))

    data = pd.concat(
        [
            pd.DataFrame(
                {
                    "embedding": y[0, :, dim],
                    "dimension": dim,
                    "position": list(range(100)),
                }
            )
            for dim in [4, 5, 6, 7]
        ]
    )

    return (
        alt.Chart(data)
        .mark_line()
        .properties(width=800)
        .encode(x="position", y="embedding", color="dimension:N")
        .interactive()
    )


show_example(example_positional)
```

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

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



1. Basic Architecture

Full Model

```
def make_model(
    src_vocab, tgt_vocab, N=6, d_model=512, d_ff=2048, h=8, dropout=0.1
):
    "Helper: Construct a model from hyperparameters."
    c = copy.deepcopy
    attn = MultiHeadedAttention(h, d_model)
    ff = PositionwiseFeedForward(d_model, d_ff, dropout)
    position = PositionalEncoding(d_model, dropout)
    model = EncoderDecoder(
        Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),
        Decoder(DecoderLayer(d_model, c(attn), c(attn), c(ff), dropout), N),
        nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
        nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
        Generator(d_model, tgt_vocab),
    )

    # This was important from their code.
    # Initialize parameters with Glorot / fan_avg.
    for p in model.parameters():
        if p.dim() > 1:
            nn.init.xavier_uniform_(p)
    return model
```

1. Basic Architecture

Inference test

```
def inference_test():
    test_model = make_model(11, 11, 2)
    test_model.eval()
    src = torch.LongTensor([[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]])
    src_mask = torch.ones(1, 1, 10)

    memory = test_model.encode(src, src_mask)
    ys = torch.zeros(1, 1).type_as(src)

    for i in range(9):
        out = test_model.decode(
            memory, src_mask, ys, subsequent_mask(ys.size(1)).type_as(src.data)
        )
        prob = test_model.generator(out[:, -1])
        _, next_word = torch.max(prob, dim=1)
        next_word = next_word.data[0]
        ys = torch.cat(
            [ys, torch.empty(1, 1).type_as(src.data).fill_(next_word)], dim=1
        )

    print("Example Untrained Model Prediction:", ys)


def run_tests():
    for _ in range(10):
        inference_test()


show_example(run_tests)
```

객체 지정 잘 되었는지 테스트.

학습 없이 Src 맞추기.

가중치 랜덤이라 엉망인 성능

```
Example Untrained Model Prediction: tensor([[0, 5, 4, 5, 7, 7, 7, 7, 7, 7]])
Example Untrained Model Prediction: tensor([[0, 3, 2, 3, 2, 7, 2, 3, 2, 3]])
Example Untrained Model Prediction: tensor([[0, 4, 4, 4, 1, 8, 8, 8, 8, 1]])
Example Untrained Model Prediction: tensor([[0, 5, 5, 5, 5, 5, 5, 5, 5, 5]])
Example Untrained Model Prediction: tensor([[ 0, 10, 10, 10, 10, 10, 10, 10, 10, 10]])
Example Untrained Model Prediction: tensor([[0, 5, 9, 5, 5, 5, 4, 5, 5, 5]])
Example Untrained Model Prediction: tensor([[0, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
Example Untrained Model Prediction: tensor([[0, 3, 0, 3, 0, 0, 0, 0, 0, 0]])
Example Untrained Model Prediction: tensor([[0, 1, 1, 1, 0, 1, 1, 1, 1, 1]])
Example Untrained Model Prediction: tensor([[0, 9, 6, 3, 2, 2, 2, 0, 7, 0]])
```

2. Before Training

Before Training

```
class Batch:
    """Object for holding a batch of data with mask during training."""

    def __init__(self, src, tgt=None, pad=2):  # 2 = <blank>
        self.src = src
        self.src_mask = (src != pad).unsqueeze(-2)
        if tgt is not None:
            self.tgt = tgt[:, :-1]
            self.tgt_y = tgt[:, 1:]
            self.tgt_mask = self.make_std_mask(self.tgt, pad)
            self.ntokens = (self.tgt_y != pad).data.sum()

    @staticmethod
    def make_std_mask(tgt, pad):
        "Create a mask to hide padding and future words."
        tgt_mask = (tgt != pad).unsqueeze(-2)
        tgt_mask = tgt_mask & subsequent_mask(tgt.size(-1)).type_as(
            tgt_mask.data
        )
        return tgt_mask
```

아까 만든 sub_mask 이용해서 띄어쓰기 제외 한 마스크 만들어주는 task

2. Before Training

Before Training

```
class TrainState:
    """Track number of steps, examples, and tokens processed"""

    step: int = 0  # Steps in the current epoch
    accum_step: int = 0  # Number of gradient accumulation steps
    samples: int = 0  # total # of examples used
    tokens: int = 0  # total # of tokens processed
```

2. Before Training

Before Training

Epoch마다 train_state 업데이트

```python
def run_epoch(
    data_iter,
    model,
    loss_compute,
    optimizer,
    scheduler,
    mode="train",
    accum_iter=1,
    train_state=TrainState(),
):
    """Train a single epoch"""
    start = time.time()
    total_tokens = 0
    total_loss = 0
    tokens = 0
    n_accum = 0
    for i, batch in enumerate(data_iter):
        out = model.forward(
            batch.src, batch.tgt, batch.src_mask, batch.tgt_mask
        )
        loss, loss_node = loss_compute(out, batch.tgt_y, batch.ntokens)
        # loss_node = loss_node / accum_iter
        if mode == "train" or mode == "train+log":
            loss_node.backward()
            train_state.step += 1
            train_state.samples += batch.src.shape[0]
            train_state.tokens += batch.ntokens
            if i % accum_iter == 0:
                optimizer.step()
                optimizer.zero_grad(set_to_none=True)
                n_accum += 1
                train_state.accum_step += 1
            scheduler.step()
```

2. Before Training

Before Training

epoch마다 steps, accu_step 등 업데이트

```python
        total_loss += loss
        total_tokens += batch.ntokens
        tokens += batch.ntokens
        if i % 40 == 1 and (mode == "train" or mode == "train+log"):
            lr = optimizer.param_groups[0]["lr"]
            elapsed = time.time() - start
            print(
                (
                    "Epoch Step: %6d | Accumulation Step: %3d | Loss: %6.2f "
                    + "| Tokens / Sec: %7.1f | Learning Rate: %6.1e"
                )
                % (i, n_accum, loss / batch.ntokens, tokens / elapsed, lr)
            )
            start = time.time()
            tokens = 0
        del loss
        del loss_node
    return total_loss / total_tokens, train_state
```

Before Training

논문에서 adam 사용)

Learning late warmup 방식 이용 :

Refer 3

```python
def rate(step, model_size, factor, warmup):
    """
    we have to default the step to 1 for LambdaLR function
    to avoid zero raising to negative power.
    """
    if step == 0:
        step = 1
    return factor * (
        model_size ** (-0.5) * min(step ** (-0.5), step * warmup ** (-1.5))
    )
```

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

2. Before Training

Before Training

Example로 warmup learning rate 변화 추이 살펴보기

```
def example_learning_schedule():
    opts = [
        [512, 1, 4000],  # example 1
        [512, 1, 8000],  # example 2
        [256, 1, 4000],  # example 3
    ]

    dummy_model = torch.nn.Linear(1, 1)
    learning_rates = []

    # we have 3 examples in opts list.
    for idx, example in enumerate(opts):
        # run 20000 epoch for each example
        optimizer = torch.optim.Adam(
            dummy_model.parameters(), lr=1, betas=(0.9, 0.98), eps=1e-9
        )
        lr_scheduler = LambdaLR(
            optimizer=optimizer, lr_lambda=lambda step: rate(step, *example)
        )
        tmp = []
        # take 20K dummy training steps, save the learning rate at each step
        for step in range(20000):
            tmp.append(optimizer.param_groups[0]["lr"])
            optimizer.step()
            lr_scheduler.step()
        learning_rates.append(tmp)

    learning_rates = torch.tensor(learning_rates)
```

2. Before Training

Before Training

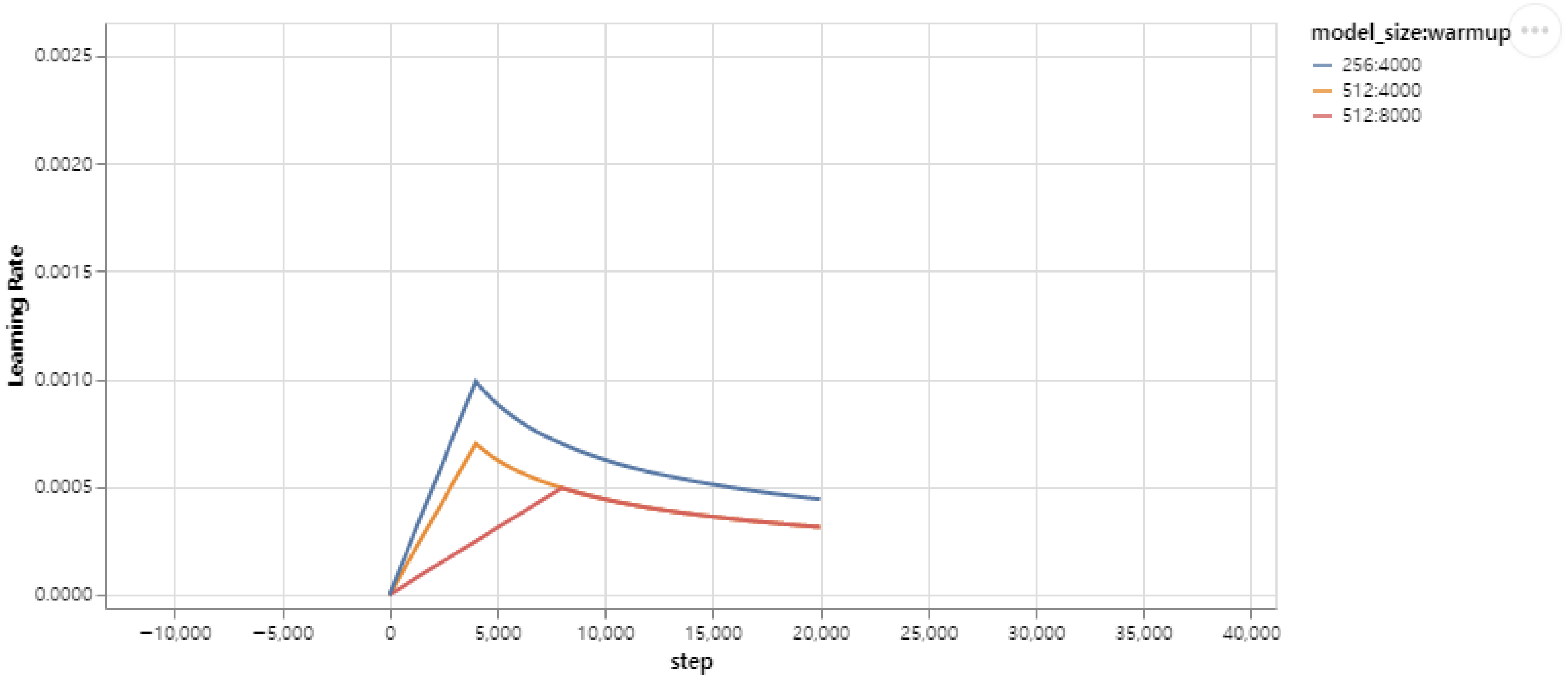
모델 사이즈와 warmup idx에 따른

Learning rate 변화 추이 그래프

```
    # Enable altair to handle more than 5000 rows
    alt.data_transformers.disable_max_rows()

    opts_data = pd.concat(
        [
            pd.DataFrame(
                {
                    "Learning Rate": learning_rates[warmup_idx, :],
                    "model_size:warmup": ["512:4000", "512:8000", "256:4000"][
                        warmup_idx
                    ],
                    "step": range(20000),
                }
            )
            for warmup_idx in [0, 1, 2]
        ]
    )

    return (
        alt.Chart(opts_data)
        .mark_line()
        .properties(width=600)
        .encode(x="step", y="Learning Rate", color="model_size:warmup:N")
        .interactive()
    )


example_learning_schedule()
```

2. Before Training

Before Training



2. Before Training

Before Training

$$q'(k|x) = (1-\epsilon)\delta_{k,y} + \epsilon u(k)$$

Train 과정에서 정답일 확률이 지나치게 높게 나오는 문제를 해결하기 위한 코드.

Refer 4.

```
class LabelSmoothing(nn.Module):
    "Implement label smoothing."

    def __init__(self, size, padding_idx, smoothing=0.0):
        super(LabelSmoothing, self).__init__()
        self.criterion = nn.KLDivLoss(reduction="sum")
        self.padding_idx = padding_idx
        self.confidence = 1.0 - smoothing
        self.smoothing = smoothing
        self.size = size
        self.true_dist = None

    def forward(self, x, target):
        assert x.size(1) == self.size
        true_dist = x.data.clone()
        true_dist.fill_(self.smoothing / (self.size - 2))
        true_dist.scatter_(1, target.data.unsqueeze(1), self.confidence)
        true_dist[:, self.padding_idx] = 0
        mask = torch.nonzero(target.data == self.padding_idx)
        if mask.dim() > 0:
            true_dist.index_fill_(0, mask.squeeze(), 0.0)
        self.true_dist = true_dist
        return self.criterion(x, true_dist.clone().detach())
```

2. Before Training

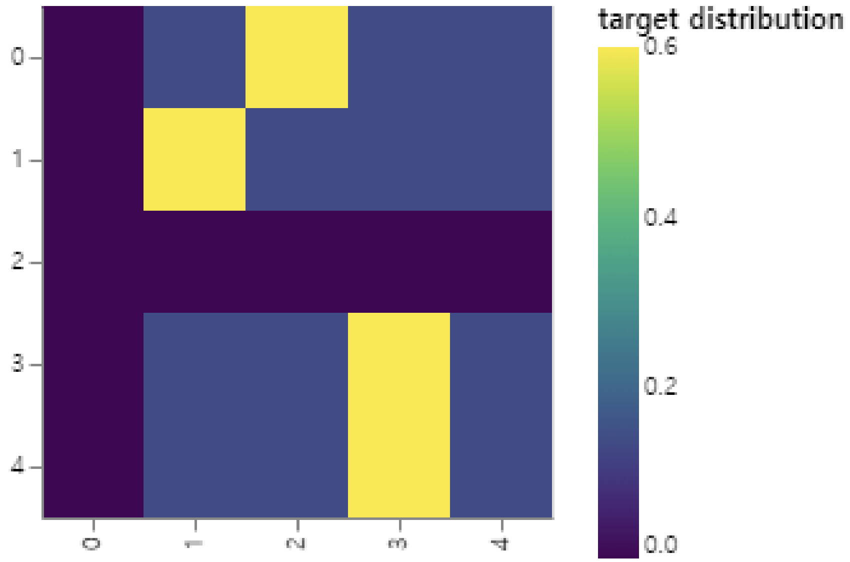
Before Training

```python
# Example of label smoothing.


def example_label_smoothing():
    crit = LabelSmoothing(5, 0, 0.4)
    predict = torch.FloatTensor(
        [
            [0, 0.2, 0.7, 0.1, 0],
            [0, 0.2, 0.7, 0.1, 0],
            [0, 0.2, 0.7, 0.1, 0],
            [0, 0.2, 0.7, 0.1, 0],
            [0, 0.2, 0.7, 0.1, 0],
        ]
    )
    crit(x=predict.log(), target=torch.LongTensor([2, 1, 0, 3, 3]))
    LS_data = pd.concat(
        [
            pd.DataFrame(
                {
                    "target distribution": crit.true_dist[x, y].flatten(),
                    "columns": y,
                    "rows": x,
                }
            )
            for y in range(5)
            for x in range(5)
        ]
    )
```

2. Before Training

Before Training

```
    return (
        alt.Chart(LS_data)
        .mark_rect(color="Blue", opacity=1)
        .properties(height=200, width=200)
        .encode(
            alt.X("columns:O", title=None),
            alt.Y("rows:O", title=None),
            alt.Color(
                "target distribution:Q", scale=alt.Scale(scheme="viridis")
            ),
        )
        .interactive()
    )


show_example(example_label_smoothing)
```



2. Before Training

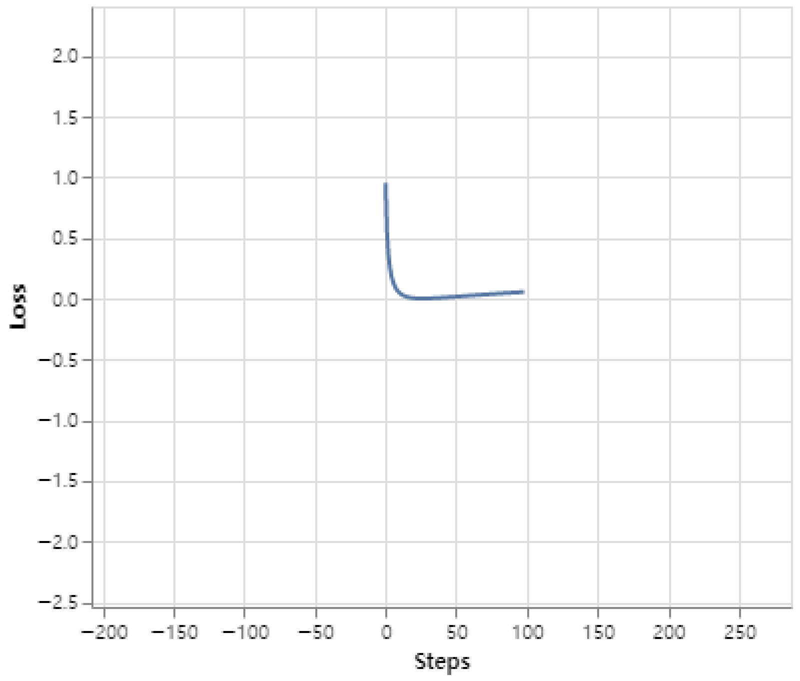
Before Training

```
def loss(x, crit):
    d = x + 3 * 1
    predict = torch.FloatTensor([[0, x / d, 1 / d, 1 / d, 1 / d]])
    return crit(predict.log(), torch.LongTensor([1])).data


def penalization_visualization():
    crit = LabelSmoothing(5, 0, 0.1)
    loss_data = pd.DataFrame(
        {
            "Loss": [loss(x, crit) for x in range(1, 100)],
            "Steps": list(range(99)),
        }
    ).astype("float")

    return (
        alt.Chart(loss_data)
        .mark_line()
        .properties(width=350)
        .encode(
            x="Steps",
            y="Loss",
        )
        .interactive()
    )


show_example(penalization_visualization)
```

2. Before Training

Before Training

```
def loss(x, crit):
    d = x + 3 * 1
    predict = torch.FloatTensor([[0, x / d, 1 / d, 1 / d, 1 / d]])
    return crit(predict.log(), torch.LongTensor([1])).data


def penalization_visualization():
    crit = LabelSmoothing(5, 0, 0.1)
    loss_data = pd.DataFrame(
        {
            "Loss": [loss(x, crit) for x in range(1, 100)],
            "Steps": list(range(99)),
        }
    ).astype("float")

    return (
        alt.Chart(loss_data)
        .mark_line()
        .properties(width=350)
        .encode(
            x="Steps",
            y="Loss",
        )
        .interactive()
    )


show_example(penalization_visualization)
```



3. Copy Model

Data generating

```
def data_gen(V, batch_size, nbatches):
    "Generate random data for a src-tgt copy task."
    for i in range(nbatches):
        data = torch.randint(1, V, size=(batch_size, 10))
        data[:, 0] = 1
        src = data.requires_grad_(False).clone().detach()
        tgt = data.requires_grad_(False).clone().detach()
        yield Batch(src, tgt, 0)
```

1부터 V-1 범위의 랜덤int 10개로 이뤄진 데이터, 배치 단위로 묶여 shape은 (b_size, 10)
첫 열, 즉 시작은 1로 고정.

3. Copy Model

Loss computation

```
class SimpleLossCompute:
    "A simple loss compute and train function."

    def __init__(self, generator, criterion):
        self.generator = generator
        self.criterion = criterion

    def __call__(self, x, y, norm):
        x = self.generator(x)
        sloss = (
            self.criterion(
                x.contiguous().view(-1, x.size(-1)), y.contiguous().view(-1)
            )
            / norm
        )
        return sloss.data * norm, sloss
```

3. Copy Model

Greedy Decode

```
def greedy_decode(model, src, src_mask, max_len, start_symbol):
    memory = model.encode(src, src_mask)
    ys = torch.zeros(1, 1).fill_(start_symbol).type_as(src.data)
    for i in range(max_len - 1):
        out = model.decode(
            memory, src_mask, ys, subsequent_mask(ys.size(1)).type_as(src.data)
        )
        prob = model.generator(out[:, -1])
        _, next_word = torch.max(prob, dim=1)
        next_word = next_word.data[0]
        ys = torch.cat(
            [ys, torch.zeros(1, 1).type_as(src.data).fill_(next_word)], dim=1
        )
    return ys
```

3. Copy Model

Greedy Decode

```
# Train the simple copy task.


def example_simple_model():
    V = 11
    criterion = LabelSmoothing(size=V, padding_idx=0, smoothing=0.0)
    model = make_model(V, V, N=2)

    optimizer = torch.optim.Adam(
        model.parameters(), lr=0.5, betas=(0.9, 0.98), eps=1e-9
    )
    lr_scheduler = LambdaLR(
        optimizer=optimizer,
        lr_lambda=lambda step: rate(
            step, model_size=model.src_embed[0].d_model, factor=1.0, warmup=400
        ),
    )

    batch_size = 80
```

3. Copy Model

Greedy Decode

```
    for epoch in range(20):
        model.train()
        run_epoch(
            data_gen(V, batch_size, 20),
            model,
            SimpleLossCompute(model.generator, criterion),
            optimizer,
            lr_scheduler,
            mode="train",
        )
        model.eval()
        run_epoch(
            data_gen(V, batch_size, 5),
            model,
            SimpleLossCompute(model.generator, criterion),
            DummyOptimizer(),
            DummyScheduler(),
            mode="eval",
        )[0]

    model.eval()
    src = torch.LongTensor([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
    max_len = src.shape[1]
    src_mask = torch.ones(1, 1, max_len)
    print(greedy_decode(model, src, src_mask, max_len=max_len, start_symbol=0))


# execute_example(example_simple_model)
```

3. Copy Model

Greedy Decode

```
for epoch in range(20):
    model.train()
    run_epoch(
        data_gen(V, batch_size, 20),
        model,
        SimpleLossCompute(model.generator, criterion),
        optimizer,
        lr_scheduler,
        mode="train",
    )
    model.eval()
    run_epoch(
        data_gen(V, batch_size, 5),
        model,
        SimpleLossCompute(model.generator, criterion),
        DummyOptimizer(),
        DummyScheduler(),
        mode="eval",
    )[0]

model.eval()
src = torch.LongTensor([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])
max_len = src.shape[1]
src_mask = torch.ones(1, 1, max_len)
print(greedy_decode(model, src, src_mask, max_len=max_len, start_symbol=0))


# execute_example(example_simple_model)
```

```
Epoch Step:      1 | Accumulation Step:   2 | Loss:   2.21 | Tokens / Sec:   321.4 | Learning Rate: 5.5e-06
Epoch Step:      1 | Accumulation Step:   2 | Loss:   1.58 | Tokens / Sec:   383.6 | Learning Rate: 6.1e-05
Epoch Step:      1 | Accumulation Step:   2 | Loss:   1.43 | Tokens / Sec:   401.1 | Learning Rate: 1.2e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   1.44 | Tokens / Sec:   395.4 | Learning Rate: 1.7e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   1.36 | Tokens / Sec:   320.1 | Learning Rate: 2.3e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   1.19 | Tokens / Sec:   397.9 | Learning Rate: 2.8e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   1.17 | Tokens / Sec:   402.4 | Learning Rate: 3.4e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   1.08 | Tokens / Sec:   395.1 | Learning Rate: 3.9e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.68 | Tokens / Sec:   395.7 | Learning Rate: 4.5e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.94 | Tokens / Sec:   397.2 | Learning Rate: 5.0e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.75 | Tokens / Sec:   398.9 | Learning Rate: 5.6e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.84 | Tokens / Sec:   272.4 | Learning Rate: 6.1e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.45 | Tokens / Sec:   392.2 | Learning Rate: 6.7e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.77 | Tokens / Sec:   392.1 | Learning Rate: 7.2e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.60 | Tokens / Sec:   392.7 | Learning Rate: 7.8e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.51 | Tokens / Sec:   399.4 | Learning Rate: 8.3e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.58 | Tokens / Sec:   308.5 | Learning Rate: 8.9e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.65 | Tokens / Sec:   399.6 | Learning Rate: 9.4e-04
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.69 | Tokens / Sec:   402.8 | Learning Rate: 1.0e-03
Epoch Step:      1 | Accumulation Step:   2 | Loss:   0.69 | Tokens / Sec:   275.3 | Learning Rate: 1.1e-03
tensor([[0, 4, 2, 3, 4, 5, 5, 4, 2, 1]])
```

4. Reference

1. http://nlp.seas.harvard.edu/annotated-transformer/#conclusion (harvard : annotated transformer)
2. https://arxiv.org/abs/1607.06450 (layer Normalization)
3. https://arxiv.org/abs/1812.01187 (learning rate warmup)
4. https://arxiv.org/pdf/1512.00567.pdf (label smoothing warmup)

DATA
SCIENCE LAB

발표자 조찬형 010-7721-4677
E-mail: chanspick28@yonsei.ac.kr