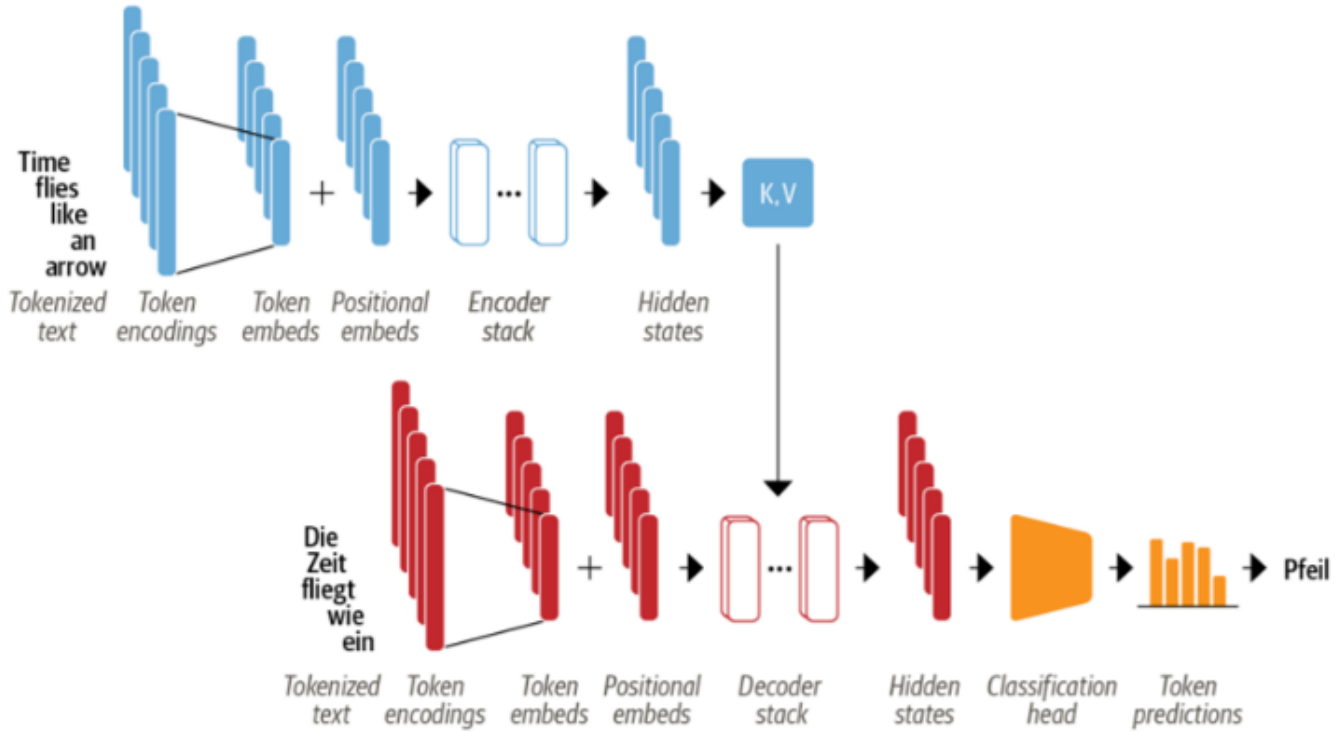
트랜스포머를 활용한 자연어 처리

3장 트랜스포머 파헤치기

Encoder - Decoder

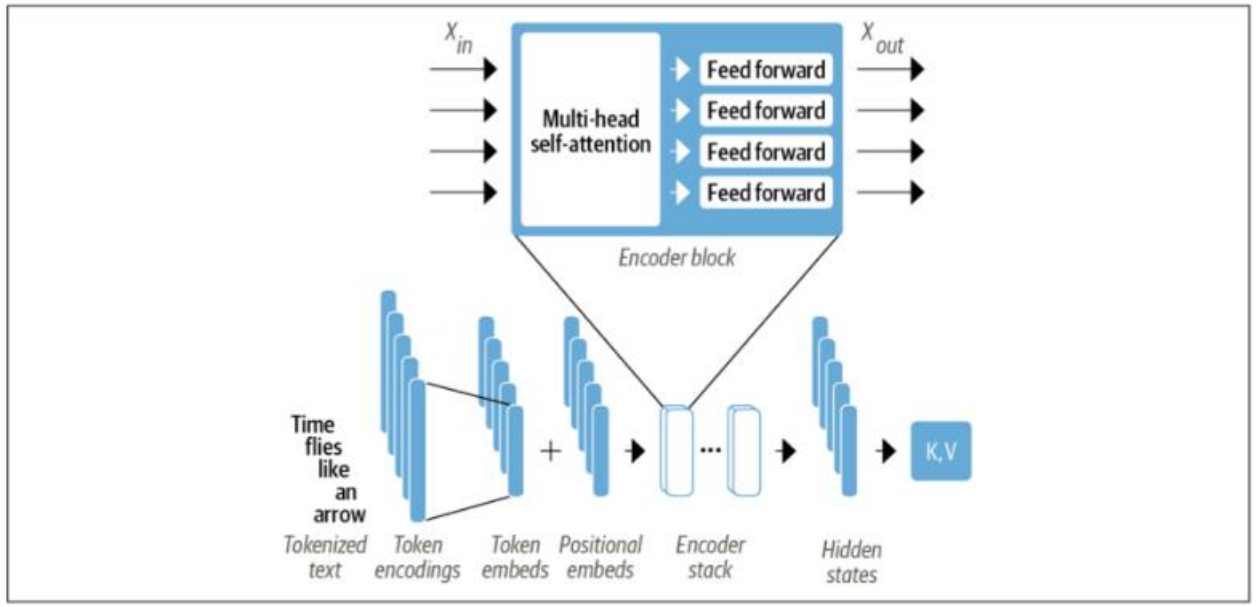


Encoder 유형

Decoder 유형

Encoder-Decoder 유형

Encoder

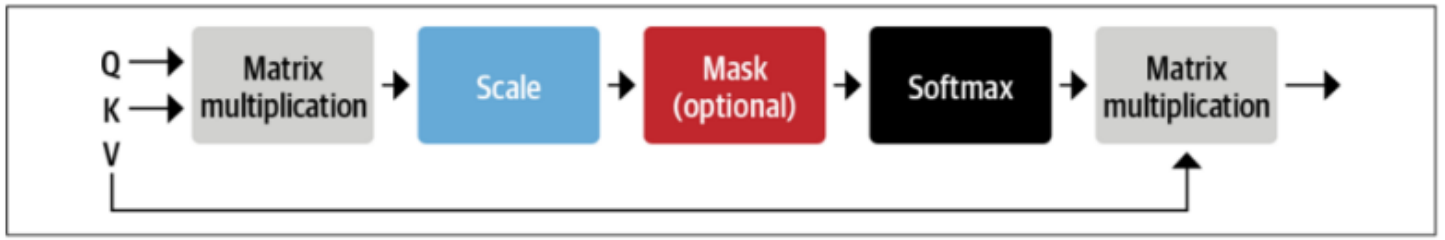


$$x'_i = \sum_{j=1}^{n} w_{ji} x_j$$

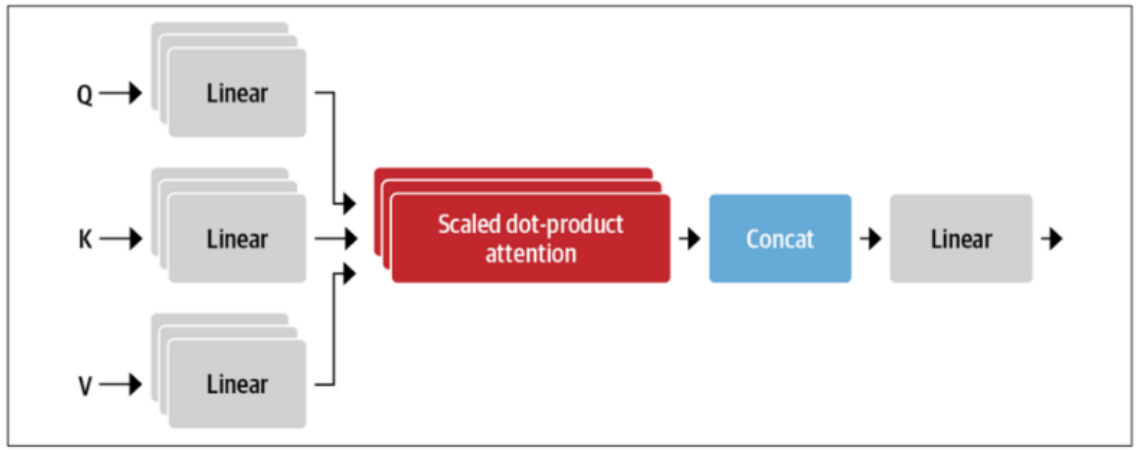
토큰 임베딩의 sequence $x_1, x_2, \ldots, x_n$ 이 주어지면 $x'_1, x'_2, \ldots, x'_n$ 를 생성하는 공식

Self Attention

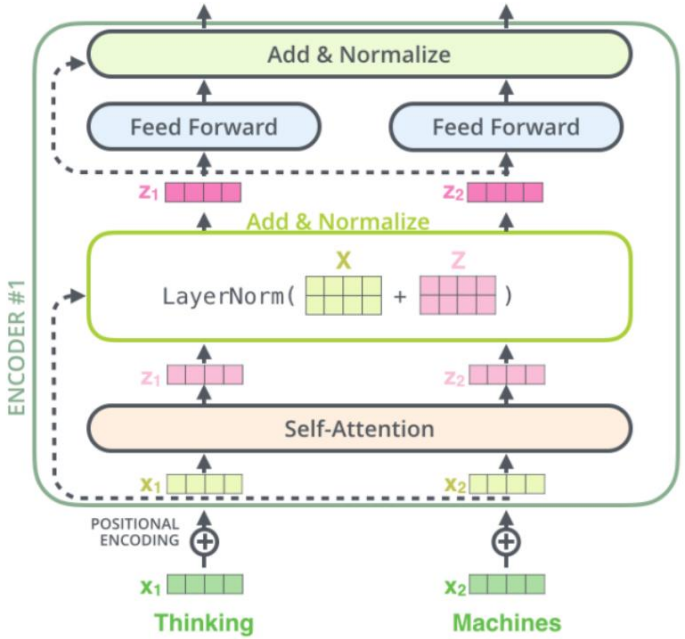
Scaled dot-product attention



Multi-head attention



Feed-Forward layer



Multi-head attention

```
class MultiHeadAttention(nn.Module):
    def __init__(self, config):
        super().__init__()
        embed_dim = config.hidden_size
        num_heads = config.num_attention_heads
        head_dim = embed_dim // num_heads
        self.heads = nn.ModuleList(
            [AttentionHead(embed_dim, head_dim) for _ in range(num_heads)]
        )
        self.output_linear = nn.Linear(embed_dim, embed_dim)

    def forward(self, hidden_state):
        x = torch.cat([h(hidden_state) for h in self.heads], dim=-1)
        x = self.output_linear(x)
        return x
```

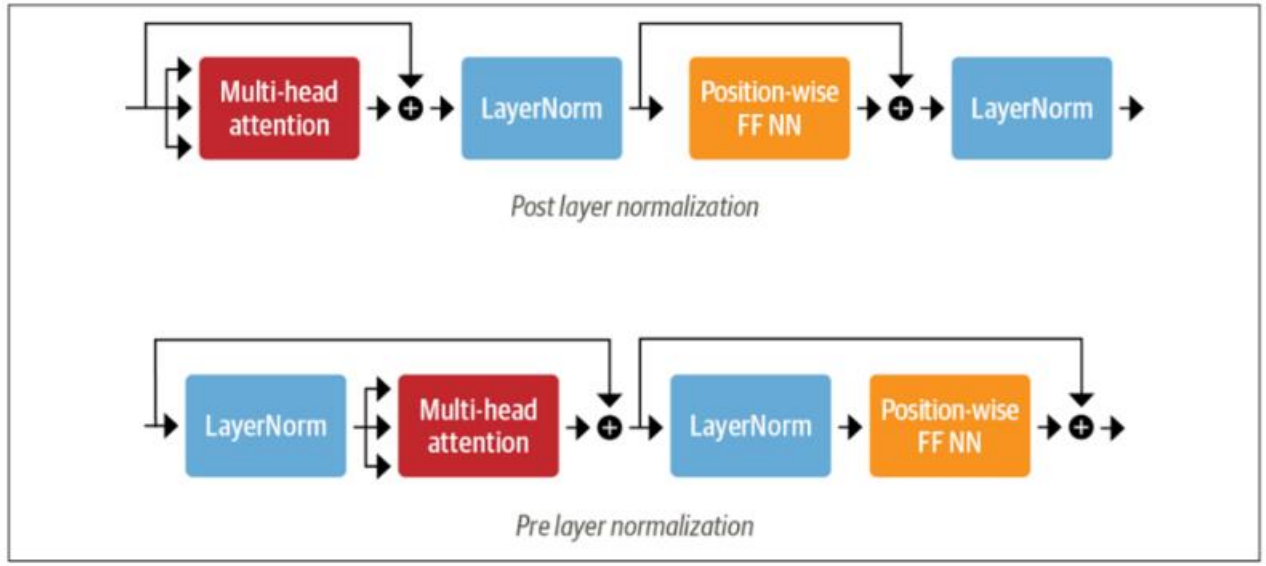
Feed-Forward layer

```
class FeedForward(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.linear_1 = nn.Linear(config.hidden_size, config.intermediate_size)
        self.linear_2 = nn.Linear(config.intermediate_size, config.hidden_size)
        self.gelu = nn.GELU()
        self.dropout = nn.Dropout(config.hidden_dropout_prob)

    def forward(self, x):
        x = self.linear_1(x)
        x = self.gelu(x)
        x = self.linear_2(x)
        x = self.dropout(x)
        return x
```

층 정규화, 스킵 연결

사후 층 정규화



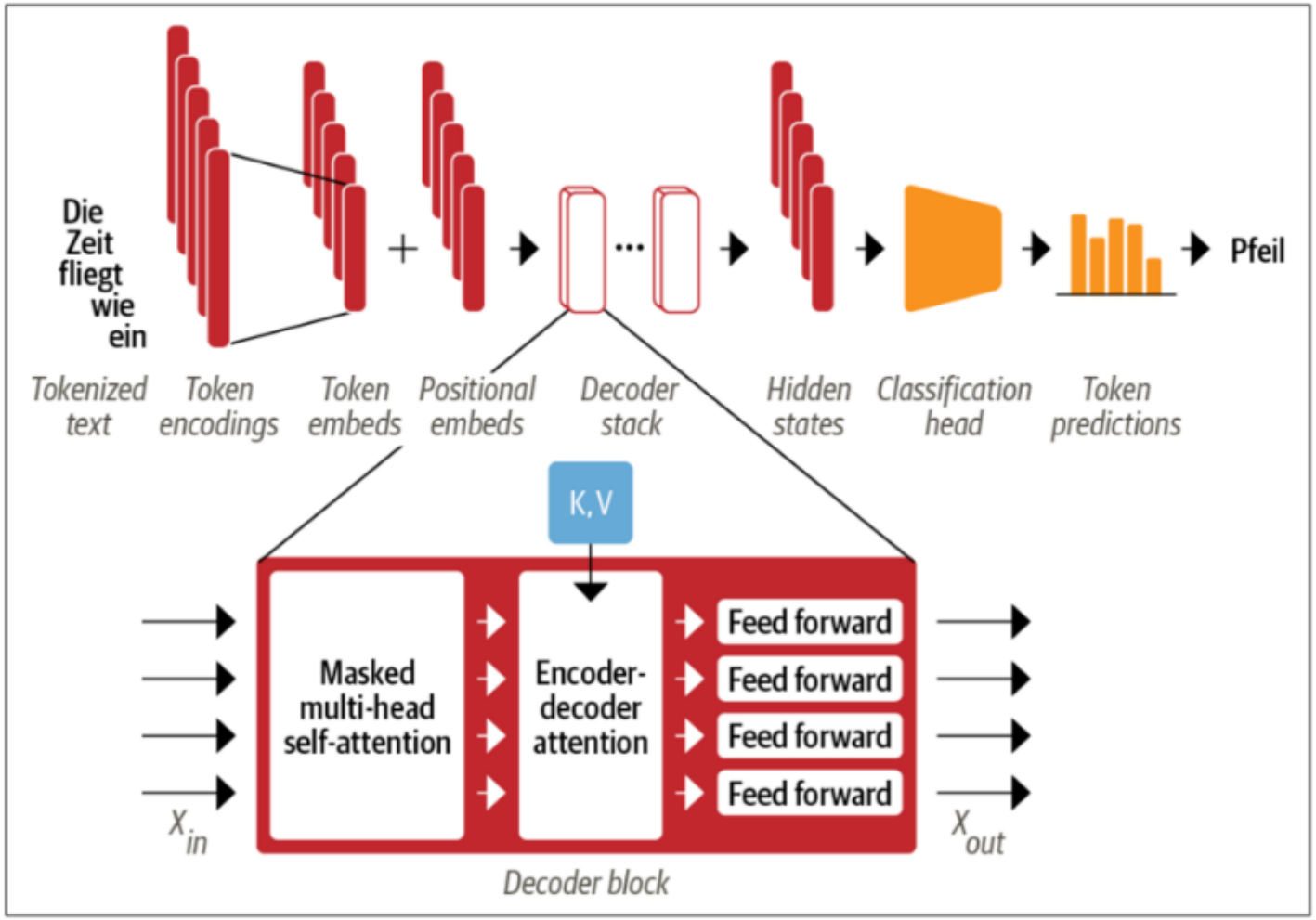
사전 층 정규화

Positional Embeding

```
class Embeddings(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.token_embeddings = nn.Embedding(config.vocab_size,
                                             config.hidden_size)
        self.position_embeddings = nn.Embedding(config.max_position_embeddings,
                                                config.hidden_size)
        self.layer_norm = nn.LayerNorm(config.hidden_size, eps=1e-12)
        self.dropout = nn.Dropout()

    def forward(self, input_ids):
        # 입력 시퀀스에 대해 위치 ID를 만듭니다.
        seq_length = input_ids.size(1)
        position_ids = torch.arange(seq_length, dtype=torch.long).unsqueeze(0)
        # 토큰 임베딩과 위치 임베딩을 만듭니다.
        token_embeddings = self.token_embeddings(input_ids)
        position_embeddings = self.position_embeddings(position_ids)
        # 토큰 임베딩과 위치 임베딩을 합칩니다.
        embeddings = token_embeddings + position_embeddings
        embeddings = self.layer_norm(embeddings)
        embeddings = self.dropout(embeddings)
        return embeddings
```

Decoder


Die
Zeit
fliegt
wie
ein
Pfeil
Tokenized text
Token encodings
Token embeds
Positional embeds
Decoder stack
Hidden states
Classification head
Token predictions
K, V
Masked multi-head self-attention
Encoder-decoder attention
Feed forward
Feed forward
Feed forward
Feed forward
X_{in}
X_{out}
Decoder block

Transformer architecture