[1]:
```python
import numpy as np

# 임의의 Q, K, V 벡터 설정
Q = np.array([[1.0, 0.0]])
K = np.array([[1.0, 0.0],
              [0.0, 1.0]])
V = np.array([[10, 0],
              [0, 10]])

# Scaled dot product
scale = np.sqrt(Q.shape[-1])  # sqrt(d_k)
scores = (Q @ K.T) / scale

# softmax 적용
exp_scores = np.exp(scores)
weights = exp_scores / np.sum(exp_scores, axis=-1, keepdims=True)

# attention 결과 계산
attention = weights @ V

print("Attention weights:", weights)
print("Attention output:", attention)
```

```
Attention weights: [[0.66976155 0.33023845]]
Attention output: [[6.69761549 3.30238451]]
```

📌 **Attention** 계산 요약 테이블

구성 요소	설명	계산 결과/설명
Q · K^T	유사도 계산 (Query와 Key 간 내적)	`[[1.0, 0.0]] · [[1.0, 0.0], [0.0, 1.0]]^T = [[1.0, 0.0]]`
/ sqrt(d□)	분산 조절 (스케일링) → softmax의 안정성 확보	`[[1.0, 0.0]] / sqrt(2) ≈ [[0.7071, 0.0]]`
softmax	유사도를 확률 분포로 변환 → attention 가중치로 사용	`softmax([[0.7071, 0.0]]) ≈ [[0.6698, 0.3302]]`
@ V	Value 벡터들과의 가중합 (Context Vector)	`[[0.6698, 0.3302]] @ [[10, 0], [0, 10]] = [[6.698, 3.302]]`

정리

Softmax는 왜 저렇게 정의되는 걸까**?**

Attention에서 유사도 값을 softmax로 바꾸는 이유는 이해됩니다.
예를 들어, Query가 각 Key와 얼마나 관련이 있는지를 수치적으로 나타낸 후, 그것을 [0, 1] 사이의 확률 분포로 바꿔야 Value 벡터들의 가중합을 할 수 있기 때문입니다.

즉, softmax는 "입력 값이 크면 더 중요하게 취급하고, 전체의 상대적인 중요도로 변환하되, 결과는 확률처럼 해석 가능하게 만들자"라는 목적에 맞는 함수입니다.

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

위 식이 단순하면서도 각 값들이 0보다 큽니다. (당연히 각각은 1보다는 작습니다.) 이는 지수함수라서 그렇습니다. 언뜻 빈도 수가 없어도 exp(0)은 1이기 때문에 분자가 양수이므로, 결과값이 차원에 따라 0으로 수렴할 수는 있어도 0이 될 수는 없다는 것입니다. 반대로 원소가 1개이기 전까지는 softmax는 전체 합만 1이 되는 확률밀도함수 역할을 할 수 있습니다. (원소가 1개이면 1개가 전체 합이므로 그 값이 1이 되는건 자명합니다.)

즉, transformer와 attention layer가 거대 차원 벡터와 행렬이므로 softmax 각각의 값은 0과 1 사이입니다.

왜 **scaling(/√d_k)**이 필요한가**?**

🔍 문제**: dot product**의 값이 너무 커질 수 있다

Query Q 와 Key K 벡터는 일반적으로 차원이 클수록
그들의 내적(dot product) 값의 분산이 커지게 됩니다.

score=Q·K^T

예를 들어, 각 요소가 평균 0, 분산 1인 경우 Q·K^T의 분산은 **d_k**에 비례합니다.

→ 즉, 차원 **d_k**이 클수록 score 값이 과도하게 커질 수 있음.

🧪 **softmax**와의 상호작용 문제

- softmax는 입력 값이 클수록 더 극단적인 출력을 내놓습니다.
- 따라서 점수가 커지면 softmax는 **0** 또는 **1**에 가까운 값을 출력하게 되고,
  이 경우 **gradient**가 거의 **0**이 되어 학습이 매우 어려워집니다 (vanishing gradient problem). 특히 오차역전파 때 가중치 학습이 중단됨.

✅ 해결 방법**: sqrt{d_k}**

scaled score=Q⋅K^T/sqrt{**d_k**}

- 이렇게 나눠주면 분산이 일정 수준으로 조절되어 softmax가 안정적으로 작동합니다.
- 결국 이는 학습 안정성과 효율성을 위한 스케일 정규화 기법입니다.

가중합은 왜 필요한가**? (Value** 결합**)**

왜 **softmax**만으로는 충분하지 않을까**?**

- softmax는 **"**어느 단어에 주목할지**"**에 대한 확률적 가중치를 제공합니다.
- 하지만 그 자체로는 정보가 없습니다.
  → 무엇에 주목할지는 정했지만, 그걸 어떻게 활용할지는 정해지지 않았어요.

그래서 필요한 게 **"**가중합**(weighted sum)"**

- attention weight는 단지 "중요도"일 뿐이고,
- 그 중요도대로 **Value** 벡터를 가중합해야
  → 의미 있는 **context vector (**문맥 표현**)** 이 나옵니다.