Attention Visualization 완벽 가이드

Transformer의 Attention Pattern을 이해하기 위한 상세 설명서

📊 개요

이 문서는 Transformer 모델의 attention weights를 시각화한 3개 플롯에 대한 완벽한 해석 가이드입니다.

```python
import matplotlib.pyplot as plt
import numpy as np

data = np.load('attention_batch_0000.npz')
attn = data['attention_weights'][-1].mean(axis=0)  # Last layer, avg over heads

fig, axes = plt.subplots(1, 3, figsize=(15, 4))
# Plot 1: Attention Matrix
# Plot 2: Temporal Importance
# Plot 3: Attention Distribution
```

🎯 Attention Weight의 수학적 의미

1. 확률 분포 (Probability Distribution)

```python
# Attention 계산 과정 (Transformer 내부)

# 1. Query와 Key의 dot product
scores = Q @ K.T / sqrt(d_k)  # (seq_len, seq_len)

# 2. Softmax → 확률 분포로 변환
attention_weights = softmax(scores, dim=-1)  # (seq_len, seq_len)

# 3. Value에 weighted sum 적용
output = attention_weights @ V
```

핵심:

```python
```

```
# 각 row의 합이 1.0
attention_weights.sum(axis=-1)  # [1.0, 1.0, 1.0, ..., 1.0]
```

Attention weight = 0.2의 의미:

> **"해당 timestep이 전체 정보의 20%를 기여한다"**

구체적 예시:

```
python

# Attention matrix의 row 5 (query timestep 5)
attn[5, :] = [0.05, 0.05, 0.10, 0.15, 0.15, 0.20, 0.15, 0.15]
              ↑    ↑    ↑    ↑    ↑    ↑    ↑    ↑
             t=0  t=1  t=2  t=3  t=4  t=5  t=6  t=7

# 해석:
# Output[t=5] = 0.05 × Input[t=0]  # 5% 기여
#             + 0.05 × Input[t=1]  # 5% 기여
#             + 0.10 × Input[t=2]  # 10% 기여
#             + 0.15 × Input[t=3]  # 15% 기여
#             + 0.15 × Input[t=4]  # 15% 기여
#             + 0.20 × Input[t=5]  # 20% 기여 (가장 중요)
#             + 0.15 × Input[t=6]  # 15% 기여
#             + 0.15 × Input[t=7]  # 15% 기여
#             = 1.00 (100%)
```

검증 코드:

```
python
```

```python
import numpy as np

data = np.load('attention_batch_0000.npz')
attn = data['attention_weights'][-1].mean(axis=0)  # (seq_len, seq_len)

# 각 row의 합 확인
row_sums = attn.sum(axis=-1)

print("Row sums (should all be ~1.0):")
print(row_sums)
# [1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000]

# 특정 query의 attention distribution
query_5_attention = attn[5, :]
print(f"\nQuery timestep 5 attention distribution:")
for t, weight in enumerate(query_5_attention):
    print(f"  Key t={t}: {weight:.4f} ({weight*100:.1f}%)")

print(f"\nSum: {query_5_attention.sum():.4f}")
# Sum: 1.0000
```

🔥 Plot 1: Attention Matrix (Heatmap)

코드:

```python
im1 = axes[0].imshow(attn, cmap='hot', aspect='auto')
axes[0].set_xlabel('Key Position (Past)')
axes[0].set_ylabel('Query Position (Current)')
axes[0].set_title('Attention Matrix')
plt.colorbar(im1, ax=axes[0])
```

축 설명:

축	범위	의미	방향
X축	0 ~ seq_len-1	Key Position (참조되는 대상)	과거 → 현재
Y축	0 ~ seq_len-1	Query Position (정보를 요청하는 주체)	과거 → 현재
색상	0.0 ~ 0.3	Attention Weight (기여도 %)	어두움 → 밝음

X축: Key Position (참조되는 대상)

- 의미: "어느 시점의 정보를 읽어올까?"
- 예: X=3 → "Timestep 3의 데이터"

Y축: Query Position (정보를 요청하는 주체)

- 의미: "어느 시점이 정보를 필요로 하는가?"
- 예: Y=5 → "Timestep 5가 정보 요청"

색상 (값): Attention Weight

- 밝음 (노란색/흰색): 높은 attention = 중요한 연결
- 어두움 (검정/빨강): 낮은 attention = 약한 연결

읽는 방법:

```
python

# Matrix의 한 점
value = attn[5, 3]  # 0.15

# 해석:
# "Query timestep 5가 Key timestep 3을 15% 참조한다"
# = "현재 시점 5가 과거 시점 3의 정보를 15% 사용한다"
```

주요 패턴:

1. Diagonal (대각선)

```
attn[0,0], attn[1,1], ..., attn[7,7]
```

- **의미**: Self-attention (자기 자신 참조)
- **색상**: 보통 밝음 (0.15~0.25)
- **해석**: "현재 상태가 가장 중요"

2. Vertical bright line (세로 밝은 선)

```
attn[:, 5]  # 모든 query가 Key 5를 주목
```

- **의미**: 특정 timestep이 전체적으로 중요
- **색상**: Column 전체가 밝음
- **해석**: "Timestep 5에 critical 정보 있음"

3. Horizontal bright line (가로 밝은 선)

```
attn[5, :]  # Query 5가 여러 key를 참조
```

- **의미**: 특정 timestep이 많은 과거를 참조
- **색상**: Row 전체가 밝음
- **해석**: "Timestep 5는 다양한 정보 통합"

예시 매트릭스:

Attention Matrix (8×8):

	Key 0	Key 1	Key 2	Key 3	Key 4	Key 5	Key 6	Key 7	
Query 0	0.18	0.12	0.10	0.08	0.09	0.20	0.12	0.11	
Query 1	0.15	0.20	0.13	0.09	0.08	0.18	0.10	0.07	
Query 2	0.12	0.14	0.22	0.11	0.09	0.16	0.09	0.07	
Query 3	0.09	0.10	0.14	0.19	0.12	0.20	0.09	0.07	
Query 4	0.08	0.09	0.11	0.13	0.21	0.22	0.10	0.06	
Query 5	0.05	0.08	0.11	0.09	0.09	0.21	0.16	0.21	← 이 row
Query 6	0.07	0.09	0.10	0.11	0.12	0.18	0.20	0.13	
Query 7	0.06	0.07	0.09	0.10	0.11	0.16	0.14	0.27	

↑

Key 5 column

읽기:

- **Diagonal 밝음**: Self-attention 강함 (각 timestep이 자기 자신 중시)
- **Key 5 column 밝음**: 모든 query가 timestep 5 주목
- **Query 5 row 분산**: Timestep 5는 여러 과거 참조

📊 Plot 2: Temporal Importance (Bar Chart)

코드:

```python
incoming = attn.sum(axis=0)
axes[1].bar(range(input_size), incoming, color='orangered')
axes[1].set_xlabel('Timestep')
axes[1].set_ylabel('Total Incoming Attention')
axes[1].set_title('Temporal Importance')
axes[1].axvline(peak_idx, color='yellow', linestyle='--', label=f'Peak (t={peak_idx})')
```

축 설명:

축	범위	의미	계산
X축	0 ~ seq_len-1	Timestep (시간)	과거 → 현재
Y축	0.5 ~ 2.0	Total Incoming Attention	attn.sum(axis=0)

X축: Timestep (시간)

- 의미: "각 시점"
- 예: X=5 → "Timestep 5"

Y축: Total Incoming Attention

- 의미: "해당 timestep이 받는 총 attention"
- 계산: attn.sum(axis=0) = 각 column의 합
- 높을수록 더 많이 참조됨

값의 의미:

```
python

# Timestep 5의 incoming attention
incoming[5] = attn[:, 5].sum()
            = attn[0,5] + attn[1,5] + ... + attn[7,5]
            = 0.20 + 0.18 + 0.16 + ... + 0.16
            = 1.48
```

해석:

- **"Timestep 5는 전체 모델에서 1.48만큼의 attention을 받는다"**
- **"모든 query들이 timestep 5를 평균 18.5% 참조한다" (1.48/8)**

값의 범위:

```
python

# 이론적 범위
min_possible = 0.0  # 아무도 참조 안 함 (불가능, softmax 때문에)
max_possible = 8.0  # 모든 query가 100% 참조 (불가능, 분산되므로)

# 현실적 범위
uniform = 1.0  # 균등 분산 (각 query가 1/8씩 참조)
typical_range = [0.5, 2.0]  # 실제 관측되는 범위
```

패턴 해석:

```
Incoming Attention:
[0.70, 0.79, 1.00, 0.91, 1.01, 1.48, 1.03, 1.08]
 ↓    ↓    ↓    ↓    ↓    ↑    ↓    ↓
낮음 낮음 평균 평균 평균 높음 평균 평균
```

의미:

- **Timestep 5 (1.48)**: **Critical timestep** - 모두가 주목
- **Timestep 0-1 (0.70-0.79)**: 덜 중요 - 초기 과도기
- **나머지 (~1.0)**: 보통 - 균등 분산

물리적 해석 (태양풍 예측):

```python
# 예: input_size=24 (24시간), 12분 cadence
timesteps_hours = np.arange(24) * 12 / 60  # [0, 0.2, 0.4, ..., 4.6] hours

# Incoming attention 플롯
incoming = attn.sum(axis=0)

# Peak at timestep 18
peak_t = incoming.argmax()  # 18
peak_hours = peak_t * 12 / 60  # 3.6 hours ago

print(f"Most important: {peak_hours:.1f} hours before prediction")
# "Most important: 3.6 hours before prediction"
```

해석: "예측 시점으로부터 약 3.6시간 전의 태양풍 데이터가 가장 critical"

📈 Plot 3: Attention Distribution (Line Plot)

코드:

```python
axes[2].plot(attn.T, alpha=0.3)  # 각 query의 attention
axes[2].plot(attn.mean(axis=0), 'k-', linewidth=3, label='Average')
axes[2].set_xlabel('Key Position')
axes[2].set_ylabel('Attention Weight')
axes[2].set_title('Attention Distribution')
```

축 설명:

축	범위	의미	설명
X축	0 ~ seq_len-1	Key Position (참조 대상)	과거 → 현재
Y축	0.0 ~ 0.3	Attention Weight (가중치)	확률 (row sum = 1.0)

X축: Key Position (참조 대상)

- 의미: "어느 timestep을 참조하는가?"
- 예: X=3 → "Key timestep 3"

Y축: Attention Weight (가중치)

- 의미: "각 key에 대한 attention 비중"
- 범위: 0~1 사이 (확률)
- 각 line의 Y값 합 = 1.0

선의 의미:

1. 얇은 선들 (alpha=0.3, 반투명)

```python
for query in range(8):
    axes[2].plot(attn[query, :], alpha=0.3)
```

- **각 선**: 하나의 query (timestep)의 attention pattern
- **8개 선**: 8개 query가 각각 어떻게 과거를 참조하는지
- **Y값**: 해당 query가 각 key에 주는 attention

예시:

```python
# Query 5의 line (분홍색 반투명 선 하나)
line_5 = attn[5, :]  # [0.05, 0.08, 0.11, 0.09, 0.09, 0.21, 0.16, 0.21]

# X=0일 때 Y=0.05: "Query 5가 Key 0을 5% 참조"
# X=5일 때 Y=0.21: "Query 5가 Key 5를 21% 참조"
```

2. 굵은 검은 선 (Average)

```python
```

```
axes[2].plot(attn.mean(axis=0), 'k-', linewidth=3)
```

- **의미**: 모든 query의 평균 attention pattern
- **계산**: `attn.mean(axis=0)` = 각 column의 평균
- **Y값**: "평균적으로 각 key가 얼마나 참조되는가?"

```
python

# Average line
avg = attn.mean(axis=0)  # (8,)
# = [mean(attn[:, 0]), mean(attn[:, 1]), ..., mean(attn[:, 7])]
# = [0.10, 0.11, 0.12, 0.12, 0.11, 0.19, 0.13, 0.12]
#                                  ↑
#                          Key 5가 평균적으로 높음
```

패턴 해석:

Pattern 1: 선들이 수렴 (Convergence)

```
모든 얇은 선들이 X=5 근처에서 높아짐
→ "모든 query가 Key 5를 중요하게 봄"
→ 굵은 선도 X=5에서 peak
```

Pattern 2: 선들이 분산 (Divergence)

```
얇은 선들이 제각각 다른 pattern
→ "각 query마다 참조 전략이 다름"
→ 일부는 초반, 일부는 후반 집중
```

Pattern 3: Diagonal dominance

```
각 선이 자기 위치(X=query_idx)에서 peak
→ Self-attention 강함
→ 현재 상태가 가장 중요
```

실제 예시 해석:

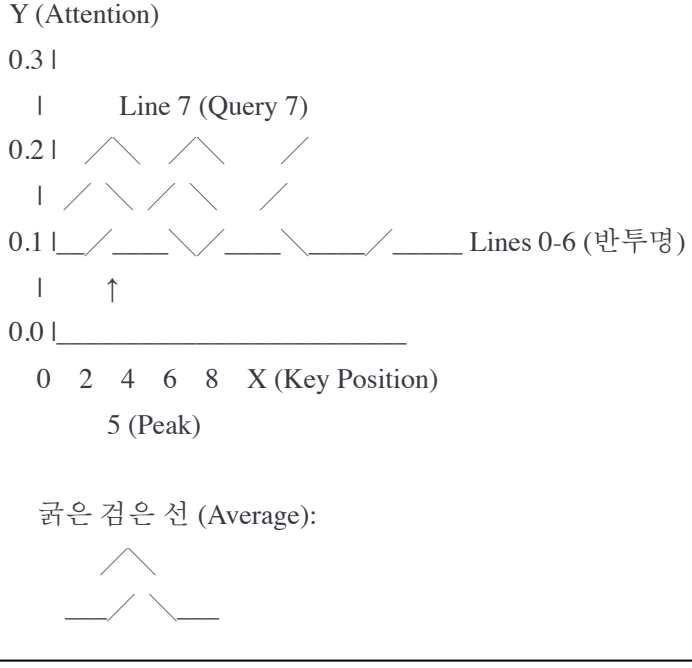
```
python
```

```
# 데이터
attn = np.array([
    [0.18, 0.12, 0.10, 0.08, 0.09, 0.20, 0.12, 0.11],  # Query 0
    [0.15, 0.20, 0.13, 0.09, 0.08, 0.18, 0.10, 0.07],  # Query 1
    [0.12, 0.14, 0.22, 0.11, 0.09, 0.16, 0.09, 0.07],  # Query 2
    [0.09, 0.10, 0.14, 0.19, 0.12, 0.20, 0.09, 0.07],  # Query 3
    [0.08, 0.09, 0.11, 0.13, 0.21, 0.22, 0.10, 0.06],  # Query 4
    [0.05, 0.08, 0.11, 0.09, 0.09, 0.21, 0.16, 0.21],  # Query 5
    [0.07, 0.09, 0.10, 0.11, 0.12, 0.18, 0.20, 0.13],  # Query 6
    [0.06, 0.07, 0.09, 0.10, 0.11, 0.16, 0.14, 0.27],  # Query 7
])

# Plot에 표시되는 선들
# Line 0 (Query 0): [0.18, 0.12, 0.10, 0.08, 0.09, 0.20, 0.12, 0.11]
#   → X=0에서 0.18, X=5에서 0.20 (peak)
# Line 1 (Query 1): [0.15, 0.20, 0.13, 0.09, 0.08, 0.18, 0.10, 0.07]
#   → X=1에서 0.20 (self), X=5에서 0.18
# ...
# Line 7 (Query 7): [0.06, 0.07, 0.09, 0.10, 0.11, 0.16, 0.14, 0.27]
#   → X=7에서 0.27 (strong self-attention)

# Average line (굵은 검은 선)
avg = attn.mean(axis=0)  # [0.10, 0.11, 0.12, 0.12, 0.11, 0.19, 0.13, 0.12]
#   → X=5에서 0.19 (peak) - 모든 query가 Key 5를 평균 19% 참조
```

시각적 해석:



🎯 세 플롯을 함께 읽기

예시 시나리오:

```python
# Plot 1 (Heatmap): Key 5 column이 전체적으로 밝음
# → "모든 query가 timestep 5를 주목"

# Plot 2 (Bar): Timestep 5에서 bar가 가장 높음 (1.48)
# → "Timestep 5가 평균 18.5% 참조됨"

# Plot 3 (Line): 굵은 검은 선이 X=5에서 peak
# → "평균적으로 timestep 5의 attention이 높음"
```

종합 해석:

> "모델은 **Timestep 5** (약 1시간 전)의 태양풍 데이터를 가장 중요하게 판단한다"

물리적 의미 (태양풍 예측):

발견	플롯	해석
Timestep 5 중요	모든 플롯	예측 전 ~1시간의 데이터가 critical
Diagonal 밝음	Plot 1	현재 상태도 중요 (self-attention)
초반 약함	Plot 2	너무 먼 과거는 덜 중요
후반 증가	Plot 2, 3	최근 데이터일수록 중요

📐 수학적 관계

Plot 2 (Incoming) ↔ Plot 3 (Average)

```python
```

```
# Plot 2의 bar 높이
incoming = attn.sum(axis=0)  # (8,)

# Plot 3의 평균선
average = attn.mean(axis=0)  # (8,)

# 관계
incoming = average * num_queries
average = incoming / num_queries

# 예시
incoming[5] = 1.48
average[5] = 1.48 / 8 = 0.185

# 의미: "Timestep 5는 평균 18.5% 참조됨"
```

Plot 1 (Matrix) → Plot 2 (Bar)

python

```
# Plot 2는 Plot 1의 column sum
for key in range(8):
    incoming[key] = attn[:, key].sum()
    # = attn[0,key] + attn[1,key] + ... + attn[7,key]
```

Plot 1 (Matrix) → Plot 3 (Lines)

python

```
# Plot 3의 각 얇은 선 = Plot 1의 각 row
for query in range(8):
    line = attn[query, :]  # Plot 1의 한 row
    plt.plot(line, alpha=0.3)  # Plot 3의 한 선
```

💡 실전 분석 코드

완전한 분석 스크립트:

python

```python
import numpy as np
import matplotlib.pyplot as plt

# Load data
data = np.load('attention_batch_0000.npz')
attn = data['attention_weights'][-1].mean(axis=0)  # (seq_len, seq_len)

print("=" * 70)
print("ATTENTION PATTERN ANALYSIS")
print("=" * 70)

# 1. Basic statistics
print(f"\n1. Matrix Shape: {attn.shape}")
print(f"   Min attention: {attn.min():.4f}")
print(f"   Max attention: {attn.max():.4f}")
print(f"   Mean attention: {attn.mean():.4f}")

# 2. Row sums (should be ~1.0)
row_sums = attn.sum(axis=-1)
print(f"\n2. Row Sums (Validation):")
print(f"   All rows sum to 1.0: {np.allclose(row_sums, 1.0)}")
print(f"   Row sums: {row_sums}")

# 3. Temporal importance
incoming = attn.sum(axis=0)
peak_t = incoming.argmax()
print(f"\n3. Temporal Importance:")
print(f"   Peak timestep: {peak_t}")
print(f"   Peak value: {incoming[peak_t]:.4f}")
print(f"   Mean importance: {incoming.mean():.4f}")

# 4. Self-attention strength
diagonal = np.diag(attn)
print(f"\n4. Self-Attention:")
print(f"   Diagonal mean: {diagonal.mean():.4f}")
print(f"   Diagonal strength: {diagonal.sum() / attn.sum():.1%}")

# 5. Attention concentration
entropy = -(attn * np.log(attn + 1e-10)).sum(axis=-1).mean()
print(f"\n5. Attention Concentration:")
print(f"   Entropy: {entropy:.4f} (higher = more uniform)")

# 6. Top-3 connections
flat_indices = np.argsort(attn.flatten())[-3:][::-1]
top_3_queries = flat_indices // attn.shape[1]
top_3_keys = flat_indices % attn.shape[1]
```

```python
print(f"\n6. Top-3 Strongest Connections:")
for i, (q, k) in enumerate(zip(top_3_queries, top_3_keys)):
    print(f"   {i+1}. Query {q} → Key {k}: {attn[q, k]:.4f}")

# 7. Pattern detection
print(f"\n7. Pattern Detection:")
if diagonal.mean() > 0.15:
    print("   ✓ Strong self-attention detected")
if incoming[peak_t] > 1.5:
    print(f"   ✓ Critical timestep detected: t={peak_t}")
if entropy < 2.0:
    print("   ✓ Focused attention (concentrated pattern)")
elif entropy > 2.5:
    print("   ✓ Distributed attention (uniform pattern)")
```

의심스러운 패턴 감지:

```python
# 1. Uniform attention (모든 값이 0.125)
if attn.std() < 0.01:
    print("⚠️  Model is not learning temporal dependencies!")

# 2. Only diagonal (self-attention만)
diagonal_ratio = np.diag(attn).sum() / attn.sum()
if diagonal_ratio > 0.8:
    print("⚠️  Model ignores past, only uses current state!")

# 3. Single timestep dominance
max_incoming = incoming.max()
if max_incoming > 3.0:  # >37.5% average
    print(f"⚠️  Over-reliance on timestep {incoming.argmax()}!")

# 4. Weak attention overall
if attn.max() < 0.15:
    print("⚠️  Weak attention weights - model may not be using attention effectively!")
```

🔬 고급 분석

Layer별 attention evolution:

```python
```

```python
# Load all layers
all_layers = data['attention_weights']  # (num_layers, num_heads, seq_len, seq_len)

fig, axes = plt.subplots(1, len(all_layers), figsize=(5*len(all_layers), 4))

for layer_idx, layer_attn in enumerate(all_layers):
    # Average over heads
    avg_attn = layer_attn.mean(axis=0)

    im = axes[layer_idx].imshow(avg_attn, cmap='hot', aspect='auto')
    axes[layer_idx].set_title(f'Layer {layer_idx}', fontsize=14, fontweight='bold')
    axes[layer_idx].set_xlabel('Key Position')
    axes[layer_idx].set_ylabel('Query Position')
    plt.colorbar(im, ax=axes[layer_idx])

plt.suptitle('Attention Evolution Across Layers', fontsize=16, fontweight='bold')
plt.tight_layout()
plt.savefig('attention_layers.png', dpi=300)
```

Head specialization 분석:

```python
# Analyze each head in last layer
last_layer = data['attention_weights'][-1]  # (num_heads, seq_len, seq_len)

fig, axes = plt.subplots(2, 4, figsize=(16, 8))

for head_idx in range(8):
    row = head_idx // 4
    col = head_idx % 4

    head_attn = last_layer[head_idx]

    im = axes[row, col].imshow(head_attn, cmap='viridis', aspect='auto')
    axes[row, col].set_title(f'Head {head_idx}', fontweight='bold')
    plt.colorbar(im, ax=axes[row, col], fraction=0.046)

plt.suptitle('Head Specialization - Last Layer', fontsize=16, fontweight='bold')
plt.tight_layout()
plt.savefig('attention_heads.png', dpi=300)
```

Batch 간 비교:

```python
```

```python
from pathlib import Path

# Load multiple batches
output_dir = Path('attention_output_dir')
npz_files = sorted(output_dir.glob('attention_batch_*.npz'))[:10]

all_temporal_imp = []
all_peak_timesteps = []

for npz_file in npz_files:
    data = np.load(npz_file)
    attn = data['attention_weights'][-1].mean(axis=0)

    temporal_imp = attn.sum(axis=0)
    all_temporal_imp.append(temporal_imp)

    peak_t = temporal_imp.argmax()
    all_peak_timesteps.append(peak_t)

# Average temporal importance
avg_temporal_imp = np.array(all_temporal_imp).mean(axis=0)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.bar(range(len(avg_temporal_imp)), avg_temporal_imp,
        color='orangered', alpha=0.7)
plt.xlabel('Timestep', fontsize=12)
plt.ylabel('Average Temporal Importance', fontsize=12)
plt.title('Average Across Batches', fontsize=14, fontweight='bold')
plt.grid(alpha=0.3)

plt.subplot(1, 2, 2)
plt.hist(all_peak_timesteps, bins=range(len(avg_temporal_imp)+1),
         color='steelblue', alpha=0.7, edgecolor='black')
plt.xlabel('Peak Timestep', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Distribution of Peak Timesteps', fontsize=14, fontweight='bold')
plt.grid(alpha=0.3)

plt.tight_layout()
plt.savefig('batch_comparison.png', dpi=300)

print(f"\nStatistics across {len(npz_files)} batches:")
```

```
print(f"  Most common peak: {max(set(all_peak_timesteps), key=all_peak_timesteps.count)}")
print(f"  Peak distribution: {np.bincount(all_peak_timesteps)}")
```

📚 참고: Attention Weight 값 해석표

Weight	의미	해석	예시
0.30+	매우 강한 연결	지배적 기여 (30%+)	Self-attention 극대
0.20-0.29	강한 연결	주요 기여 (20-29%)	Critical timestep
0.125	균등 분산	평균 기여 (1/8)	균등 참조
0.08-0.12	약한 연결	소수 기여 (8-12%)	배경 정보
0.05-	매우 약한 연결	무시 가능 (<5%)	거의 사용 안 함

🎓 Summary

핵심 포인트:

1. **Attention weight는 확률**: Row sum = 1.0, 각 값 = 기여도 %
2. **Plot 1 (Matrix)**: 전체 attention 패턴 (query × key)
3. **Plot 2 (Bar)**: 각 timestep이 받는 총 attention (column sum)
4. **Plot 3 (Line)**: 각 query의 attention 분포 (row별 시각화)

분석 체크리스트:

- ☐ Row sums가 모두 1.0인가? (확률 분포 검증)
- ☐ Diagonal이 밝은가? (Self-attention 강도)
- ☐ 특정 timestep이 밝은가? (Critical point 존재)
- ☐ Attention이 너무 uniform한가? (학습 실패 의심)
- ☐ Peak timestep이 물리적으로 합리적인가?

논문과 비교:

DeepHalo (Zhang et al. 2025) 발견:

- **Positive prediction**: Uniform attention (progressive process)
- **Negative prediction**: Early-focused attention (early dismissal)

Eunsu님 모델:

- Regression task → 더 복잡한 패턴
- Multiple critical timesteps
- Self-attention + long-range dependency 혼합

📖 추가 자료

관련 논문:

- Zhang et al. (2025) - "Prediction of Halo CMEs Using Transformer Model"
- Vaswani et al. (2017) - "Attention Is All You Need"

코드 저장소:

- `attention_analysis.py` - AttentionExtractor 클래스
- `example_attention_all_targets.py` - 실행 스크립트
- `attention_0.yaml` - 설정 파일

마지막 업데이트: 2025-01-22
작성자: Claude (Anthropic)
버전: 1.0