# Transformer_Notes

在 Transformer 架構中，鍵（Key）和值（Value）是自注意力機制中的核心概念，這些概念在信息檢索和上下文建模中具有重要意義。以下是對它們的詳細解釋：

## 1. 鍵（Key）

- **定義**：鍵是用來表示輸入數據中每個元素的重要特徵。每個輸入元素（如字詞）都會生成一個鍵向量。
- **功能**：在計算注意力時，查詢（Query）向量會與所有鍵向量進行匹配，以確定哪些元素對當前查詢最相關。這樣的匹配通常是通過計算相似度（如點積）來實現的。

## 2. 值（Value）

- **定義**：值是與鍵相關聯的實際信息內容。每個鍵都有一個對應的值向量，這些值向量包含了需要傳遞的具體信息。
- **功能**：在計算出注意力權重後，最終的輸出是根據這些權重對所有值進行加權求和。這意味著模型會根據查詢的相關性，選擇性地聚焦於某些值向量。

## 3. 自注意力機制中的作用

- **匹配和選擇**：當處理一個序列（如句子）時，自注意力機制會計算每個查詢與所有鍵的相似度，從而決定哪些值在生成當前輸出時應該被考慮。
- **上下文建模**：通過這種方式，Transformer 能夠捕捉到序列中各個元素之間的關係，無論它們在序列中的距離有多遠。這使得 Transformer 在處理長文本或複雜依賴關係時特別有效。

## 4. 總結

- 鍵和值在 Transformer 中的作用是通過查詢來檢索相關信息，並根據這些信息生成上下文表示。這種設計使得模型能夠靈活地聚焦於不同的輸入元素，從而提高了語言理解和生成的能力。

---

## Codes with 多頭注意力

好的！以下是基於之前的自注意力機制，加入多頭注意力的 NumPy 實現。這個版本將展示如何在

自注意力機制的基礎上擴展到多頭注意力。

## 多頭注意力的 NumPy 實現

```
import numpy as np

def softmax(x):
    exp_x = np.exp(x - np.max(x))  # 避免溢出
    return exp_x / exp_x.sum(axis=-1, keepdims=True)

class MultiHeadAttention:
    def __init__(self, embed_size, heads):
        self.embed_size = embed_size
        self.heads = heads
        self.head_dim = embed_size // heads

        # 隨機初始化查詢、鍵和值的權重
        self.W_q = np.random.rand(embed_size, embed_size)
        self.W_k = np.random.rand(embed_size, embed_size)
        self.W_v = np.random.rand(embed_size, embed_size)
        self.W_o = np.random.rand(embed_size, embed_size)

    def split_heads(self, x):
        # 將嵌入分成多個頭
        N, seq_length, embed_size = x.shape
        x = x.reshape(N, seq_length, self.heads, self.head_dim)
        return x.transpose(0, 2, 1, 3)  # (N, heads, seq_length, head_dim)

    def forward(self, inputs):
        N, seq_length, embed_size = inputs.shape

        # 計算查詢、鍵和值
        Q = inputs @ self.W_q  # (N, seq_length, embed_size)
        K = inputs @ self.W_k  # (N, seq_length, embed_size)
        V = inputs @ self.W_v  # (N, seq_length, embed_size)

        # 分頭
        Q = self.split_heads(Q)  # (N, heads, seq_length, head_dim)
        K = self.split_heads(K)  # (N, heads, seq_length, head_dim)
        V = self.split_heads(V)  # (N, heads, seq_length, head_dim)
```

```
        # 計算注意力得分
        scores = Q @ K.transpose(0, 1, 3, 2) / np.sqrt(self.head_dim)  # (N, heads, seq_length, seq_length)

        # 計算注意力權重
        attention_weights = softmax(scores)  # (N, heads, seq_length, seq_length)

        # 計算加權值
        out = attention_weights @ V  # (N, heads, seq_length, head_dim)

        # 合併頭部
            out = out.transpose(0, 2, 1, 3).reshape(N, seq_length, self.embed_size)  # (N, seq_length,
embed_size)

        # 最終輸出
        output = out @ self.W_o  # (N, seq_length, embed_size)
        return output, attention_weights

# 測試多頭注意力
if __name__ == "__main__":
    # 假設有一個簡單的句子，嵌入維度為6，使用2個頭
    sentence = np.array([
        [1, 0, 0, 0, 0, 0],  # 我
        [0, 1, 0, 0, 0, 0],  # 喜歡
        [0, 0, 1, 0, 0, 0],  # 吃
        [0, 0, 0, 1, 0, 0],  # 蘋果
    ]).reshape(1, 4, 6)  # (N, seq_length, embed_size)

    multi_head_attention = MultiHeadAttention(embed_size=6, heads=2)
    output, attention_weights = multi_head_attention.forward(sentence)

    print("輸出：")
    print(output)
    print("\n注意力權重：")
    print(attention_weights)
```

## 說明

1. **MultiHeadAttention 類**：
    - 初始化時設置嵌入維度和頭數，並隨機初始化查詢、鍵和值的權重矩陣。

2. `split_heads` **方法**：
    - 將嵌入分成多個頭，便於後續計算。
3. `forward` **方法**：
    - 計算查詢、鍵和值。
    - 將查詢、鍵和值分成多個頭。
    - 計算注意力得分和權重。
    - 計算加權值，然後合併頭部。
    - 最後使用輸出權重矩陣生成最終輸出。
4. **測試**：
    - 用一個簡單的句子來測試多頭注意力機制。

### 執行程式碼

將上述程式碼複製到 Python 環境中運行，即可看到多頭注意力的輸出和注意力權重。這個實現展示了多頭注意力的基本概念，實際的 Transformer 還包含其他組件，如位置編碼和前饋網絡等。

---

## Codes with 位置編碼

好的！以下是將「位置編碼」加入到多頭注意力的 NumPy 實現中。位置編碼的目的是為了讓模型能夠考慮到序列中單詞的位置信息，這在 Transformer 中是非常重要的。

### 加入位置編碼的多頭注意力 NumPy 實現

```
import numpy as np

def softmax(x):
    exp_x = np.exp(x - np.max(x))  # 避免溢出
    return exp_x / exp_x.sum(axis=-1, keepdims=True)

def positional_encoding(seq_length, embed_size):
    pos = np.arange(seq_length)[:, np.newaxis]  # (seq_length, 1)
```

```
    i = np.arange(embed_size)[np.newaxis, :]  # (1, embed_size)
    angle_rates = 1 / np.power(10000, (2 * (i // 2)) / embed_size)  # (1, embed_size)
    angle = pos * angle_rates  # (seq_length, embed_size)

    # 偶數維度使用 sin，奇數維度使用 cos
    angle[:, 0::2] = np.sin(angle[:, 0::2])  # 偶數維度
    angle[:, 1::2] = np.cos(angle[:, 1::2])  # 奇數維度
    return angle

class MultiHeadAttention:
    def __init__(self, embed_size, heads):
        self.embed_size = embed_size
        self.heads = heads
        self.head_dim = embed_size // heads

        # 隨機初始化查詢、鍵和值的權重
        self.W_q = np.random.rand(embed_size, embed_size)
        self.W_k = np.random.rand(embed_size, embed_size)
        self.W_v = np.random.rand(embed_size, embed_size)
        self.W_o = np.random.rand(embed_size, embed_size)

    def split_heads(self, x):
        # 將嵌入分成多個頭
        N, seq_length, embed_size = x.shape
        x = x.reshape(N, seq_length, self.heads, self.head_dim)
        return x.transpose(0, 2, 1, 3)  # (N, heads, seq_length, head_dim)

    def forward(self, inputs):
        N, seq_length, embed_size = inputs.shape

        # 計算查詢、鍵和值
        Q = inputs @ self.W_q  # (N, seq_length, embed_size)
        K = inputs @ self.W_k  # (N, seq_length, embed_size)
        V = inputs @ self.W_v  # (N, seq_length, embed_size)

        # 分頭
        Q = self.split_heads(Q)  # (N, heads, seq_length, head_dim)
        K = self.split_heads(K)  # (N, heads, seq_length, head_dim)
        V = self.split_heads(V)  # (N, heads, seq_length, head_dim)

        # 計算注意力得分
        scores = Q @ K.transpose(0, 1, 3, 2) / np.sqrt(self.head_dim)  # (N, heads, seq_length, seq_length)
```

```
        # 計算注意力權重
        attention_weights = softmax(scores)  # (N, heads, seq_length, seq_length)

        # 計算加權值
        out = attention_weights @ V  # (N, heads, seq_length, head_dim)

        # 合併頭部
        out = out.transpose(0, 2, 1, 3).reshape(N, seq_length, self.embed_size)  # (N, seq_length, embed_size)

        # 最終輸出
        output = out @ self.W_o  # (N, seq_length, embed_size)
        return output, attention_weights

# 測試多頭注意力與位置編碼
if __name__ == "__main__":
    # 假設有一個簡單的句子，嵌入維度為6，使用2個頭
    sentence = np.array([
        [1, 0, 0, 0, 0, 0],  # 我
        [0, 1, 0, 0, 0, 0],  # 喜歡
        [0, 0, 1, 0, 0, 0],  # 吃
        [0, 0, 0, 1, 0, 0],  # 蘋果
    ]).reshape(1, 4, 6)  # (N, seq_length, embed_size)

    # 添加位置編碼
    position_enc = positional_encoding(seq_length=4, embed_size=6)
    sentence += position_enc  # 將位置編碼加到輸入上

    multi_head_attention = MultiHeadAttention(embed_size=6, heads=2)
    output, attention_weights = multi_head_attention.forward(sentence)

    print("輸出：")
    print(output)
    print("\n注意力權重：")
    print(attention_weights)
```

## 說明

1. **位置編碼函數** `positional_encoding` ：
    - 計算位置編碼，使用正弦和餘弦函數來生成每個位置的編碼。

- 偶數維度使用 `sin`，奇數維度使用 `cos`。

2. **在測試部分**：
   - 將位置編碼加到輸入句子上，以便在計算多頭注意力時考慮到位置信息。

## 執行程式碼

將上述程式碼複製到 Python 環境中運行，即可看到多頭注意力的輸出和注意力權重，並已經考慮了位置編碼。這個實現展示了如何在多頭注意力中融入位置信息，使得模型能夠更好地捕捉序列中的結構。

---

好的！以下是將「前饋網絡」加入到多頭注意力的 NumPy 實現中。前饋網絡通常是在多頭注意力層之後的組件，負責進一步處理注意力層的輸出。

## 加入前饋網絡的多頭注意力 NumPy 實現

```
import numpy as np

def softmax(x):
    exp_x = np.exp(x - np.max(x))  # 避免溢出
    return exp_x / exp_x.sum(axis=-1, keepdims=True)

def positional_encoding(seq_length, embed_size):
    pos = np.arange(seq_length)[:, np.newaxis]  # (seq_length, 1)
    i = np.arange(embed_size)[np.newaxis, :]  # (1, embed_size)
    angle_rates = 1 / np.power(10000, (2 * (i // 2)) / embed_size)  # (1, embed_size)
    angle = pos * angle_rates  # (seq_length, embed_size)

    # 偶數維度使用 sin，奇數維度使用 cos
    angle[:, 0::2] = np.sin(angle[:, 0::2])  # 偶數維度
    angle[:, 1::2] = np.cos(angle[:, 1::2])  # 奇數維度
    return angle

class FeedForwardNetwork:
    def __init__(self, embed_size, ffn_hidden_size):
        self.W1 = np.random.rand(embed_size, ffn_hidden_size)  # 第一層權重
        self.b1 = np.random.rand(ffn_hidden_size)  # 第一層偏置
        self.W2 = np.random.rand(ffn_hidden_size, embed_size)  # 第二層權重
```

```
        self.b2 = np.random.rand(embed_size)  # 第二層偏置

    def forward(self, x):
        # 前饋網絡
        x = np.maximum(0, x @ self.W1 + self.b1)  # ReLU 激活
        x = x @ self.W2 + self.b2  # 線性變換
        return x

class MultiHeadAttention:
    def __init__(self, embed_size, heads):
        self.embed_size = embed_size
        self.heads = heads
        self.head_dim = embed_size // heads

        # 隨機初始化查詢、鍵和值的權重
        self.W_q = np.random.rand(embed_size, embed_size)
        self.W_k = np.random.rand(embed_size, embed_size)
        self.W_v = np.random.rand(embed_size, embed_size)
        self.W_o = np.random.rand(embed_size, embed_size)

    def split_heads(self, x):
        # 將嵌入分成多個頭
        N, seq_length, embed_size = x.shape
        x = x.reshape(N, seq_length, self.heads, self.head_dim)
        return x.transpose(0, 2, 1, 3)  # (N, heads, seq_length, head_dim)

    def forward(self, inputs):
        N, seq_length, embed_size = inputs.shape

        # 計算查詢、鍵和值
        Q = inputs @ self.W_q  # (N, seq_length, embed_size)
        K = inputs @ self.W_k  # (N, seq_length, embed_size)
        V = inputs @ self.W_v  # (N, seq_length, embed_size)

        # 分頭
        Q = self.split_heads(Q)  # (N, heads, seq_length, head_dim)
        K = self.split_heads(K)  # (N, heads, seq_length, head_dim)
        V = self.split_heads(V)  # (N, heads, seq_length, head_dim)

        # 計算注意力得分
        scores = Q @ K.transpose(0, 1, 3, 2) / np.sqrt(self.head_dim)  # (N, heads, seq_length, seq_length)
```

```
        # 計算注意力權重
        attention_weights = softmax(scores)  # (N, heads, seq_length, seq_length)

        # 計算加權值
        out = attention_weights @ V  # (N, heads, seq_length, head_dim)

        # 合併頭部
        out = out.transpose(0, 2, 1, 3).reshape(N, seq_length, self.embed_size)  # (N, seq_length, embed_size)

        # 最終輸出
        output = out @ self.W_o  # (N, seq_length, embed_size)
        return output, attention_weights

# 測試多頭注意力與位置編碼和前饋網絡
if __name__ == "__main__":
    # 假設有一個簡單的句子，嵌入維度為6，使用2個頭
    sentence = np.array([
        [1, 0, 0, 0, 0, 0],  # 我
        [0, 1, 0, 0, 0, 0],  # 喜歡
        [0, 0, 1, 0, 0, 0],  # 吃
        [0, 0, 0, 1, 0, 0],  # 蘋果
    ]).reshape(1, 4, 6)  # (N, seq_length, embed_size)

    # 添加位置編碼
    position_enc = positional_encoding(seq_length=4, embed_size=6)
    sentence += position_enc  # 將位置編碼加到輸入上

    # 初始化多頭注意力和前饋網絡
    multi_head_attention = MultiHeadAttention(embed_size=6, heads=2)
    feed_forward_network = FeedForwardNetwork(embed_size=6, ffn_hidden_size=12)

    # 計算多頭注意力的輸出
    attention_output, attention_weights = multi_head_attention.forward(sentence)

    # 計算前饋網絡的輸出
    final_output = feed_forward_network.forward(attention_output)

    print("最終輸出：")
    print(final_output)
    print("\n注意力權重：")
    print(attention_weights)
```

## 說明

1. **前饋網絡類** `FeedForwardNetwork` ：
    - 包含兩層全連接層，第一層使用 ReLU 激活函數，第二層進行線性變換。
    - 初始化時隨機生成權重和偏置。
2. **在測試部分**：
    - 將位置編碼加到輸入句子上。
    - 計算多頭注意力的輸出，然後將其傳遞到前饋網絡進行處理。

## 執行程式碼

將上述程式碼複製到 Python 環境中運行，即可看到最終輸出和注意力權重。這個實現展示了如何在多頭注意力中融入位置信息，並在其後添加前饋網絡，以進一步處理注意力層的輸出。這樣的結構是 Transformer 的基本組成部分。