MODUL PRAKTIKUM DEEP LEARNING

Modul 8: Attention Mechanism and Transformer

Program Studi Sains Data
Fakultas Sains
Institut Teknologi Sumatera

Tahun 2025

Daftar Isi

1 Tujuan Pembelajaran

Setelah mengikuti praktikum ini, mahasiswa diharapkan mampu:

1. Memahami konsep dasar attention mechanism dalam deep learning
2. Mengimplementasikan queries, keys, dan values dalam attention
3. Memahami dan mengimplementasikan attention pooling
4. Mengimplementasikan berbagai attention scoring functions
5. Memahami mekanisme Bahdanau attention
6. Mengimplementasikan multi-head attention
7. Memahami self-attention dan positional encoding
8. Membangun arsitektur Transformer lengkap

2 Teori dan Konsep

2.1 Pengantar Attention Mechanism

Attention mechanism adalah inovasi penting dalam deep learning yang memungkinkan model untuk fokus secara dinamis pada bagian-bagian berbeda dari input saat menghasilkan output. Berbeda dengan arsitektur RNN tradisional yang memampatkan seluruh input menjadi vektor konteks tetap, attention memungkinkan model untuk "mengakses" seluruh input sequence pada setiap langkah decoding.

2.1.1 Motivasi

Pada model sequence-to-sequence tradisional, encoder mengompresi seluruh input menjadi satu vektor konteks dengan dimensi tetap. Hal ini menjadi bottleneck untuk sequence yang panjang karena:

- Informasi penting bisa hilang dalam kompresi
- Model kesulitan mengingat dependensi jangka panjang
- Tidak ada mekanisme untuk fokus pada bagian relevan dari input

2.2 Queries, Keys, dan Values

Attention mechanism dapat dipandang sebagai operasi database yang terdiri dari tiga komponen utama:

2.2.1 Definisi Formal

Misalkan kita memiliki database $\mathcal{D} = \{(k_1, v_1), \ldots, (k_n, v_n)\}$ yang terdiri dari n pasangan key-value, dan sebuah query q. Attention dapat didefinisikan sebagai:

$$\text{Attention}(q, \mathcal{D}) = \sum_{i=1}^{n} \alpha(q, k_i) v_i \tag{1}$$

dimana $\alpha(q, k_i)$ adalah attention weight yang menentukan seberapa besar kontribusi dari value v_i.

2.2.2 Komponen Attention

- **Query (q)**: Representasi dari elemen yang sedang diproses, yang akan "mencari" informasi relevan
- **Keys (k)**: Representasi dari semua elemen yang tersedia, digunakan untuk matching dengan query
- **Values (v)**: Informasi aktual yang akan diagregasi berdasarkan attention weights

2.2.3 Normalisasi Attention Weights

Untuk memastikan attention weights membentuk distribusi probabilitas, kita normalisasi menggunakan softmax:

$$\alpha(q, k_i) = \frac{\exp(s(q, k_i))}{\sum_{j=1}^{n} \exp(s(q, k_j))} \tag{2}$$

dimana $s(q, k_i)$ adalah scoring function yang mengukur kompatibilitas antara query dan key.

2.3 Attention Pooling by Similarity

Attention pooling mengagregasi values berdasarkan kesamaan antara queries dan keys.

2.3.1 Nadaraya-Watson Kernel Regression

Sebagai contoh klasik, Nadaraya-Watson estimator menggunakan kernel similarity:

$$f(q) = \sum_{i=1}^{n} \frac{K(q, k_i)}{\sum_{j=1}^{n} K(q, k_j)} v_i \tag{3}$$

Kernel Functions yang Umum:

$$K_{\text{Gaussian}}(q, k) = \exp\left(-\frac{1}{2}\|q - k\|^2\right) \tag{4}$$

$$K_{\text{Boxcar}}(q, k) = \nVdash(\|q - k\| \leq 1) \tag{5}$$

$$K_{\text{Epanechnikov}}(q, k) = \max(0, 1 - \|q - k\|) \tag{6}$$

2.4 Attention Scoring Functions

Scoring function menentukan seberapa baik query dan key "match" satu sama lain.

2.4.1 Scaled Dot-Product Attention

Ini adalah scoring function paling populer dalam Transformer:

$$s(q, k_i) = \frac{q^T k_i}{\sqrt{d}} \tag{7}$$

dimana d adalah dimensi dari query dan key. Pembagian dengan $\sqrt{d}$ mencegah nilai dot product menjadi terlalu besar.

Attention Lengkap:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{8}$$

dimana:

- $Q \in \mathbb{R}^{n\times d}$: matrix dari n queries
- $K \in \mathbb{R}^{m\times d}$: matrix dari m keys
- $V \in \mathbb{R}^{m\times d_v}$: matrix dari m values

2.4.2 Additive Attention (Bahdanau Attention)

Alternatif untuk dot-product, menggunakan feed-forward network:

$$s(q, k) = w_v^T \tanh(W_q q + W_k k) \tag{9}$$

dimana $W_q \in \mathbb{R}^{h\times d_q}$, $W_k \in \mathbb{R}^{h\times d_k}$, dan $w_v \in \mathbb{R}^h$ adalah parameter yang dapat dipelajari.

2.5 Bahdanau Attention Mechanism

Bahdanau attention adalah salah satu implementasi attention pertama untuk sequence-to-sequence learning.

2.5.1 Arsitektur

Dalam konteks machine translation:

1. Encoder menghasilkan hidden states $h_1, \ldots, h_T$ untuk input sequence
2. Pada setiap decoding step t', decoder menggunakan:
   - Previous decoder state $s_{t'-1}$ sebagai query
   - Semua encoder hidden states h_t sebagai keys dan values
3. Context vector dihitung:
$$c_{t'} = \sum_{t=1}^{T} \alpha_{t',t} h_t \tag{10}$$
4. Attention weights:
$$\alpha_{t',t} = \frac{\exp(e_{t',t})}{\sum_{j=1}^{T} \exp(e_{t',j})} \tag{11}$$
   dimana $e_{t',t} = s(s_{t'-1}, h_t)$ adalah alignment score

2.6 Multi-Head Attention

Multi-head attention memungkinkan model untuk fokus pada informasi dari berbagai representation subspaces secara paralel.

2.6.1 Formulasi

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{12}$$

dimana setiap head dihitung sebagai:

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{13}$$

Parameter Matrices:

- $W_i^Q \in \mathbb{R}^{d \times d_k}$: projection matrix untuk queries
- $W_i^K \in \mathbb{R}^{d \times d_k}$: projection matrix untuk keys
- $W_i^V \in \mathbb{R}^{d \times d_v}$: projection matrix untuk values
- $W^O \in \mathbb{R}^{hd_v \times d}$: output projection matrix

2.6.2 Keuntungan Multi-Head

1. Memungkinkan model attend ke berbagai posisi secara paralel
2. Setiap head dapat mempelajari aspek berbeda dari input
3. Meningkatkan kapasitas model tanpa menambah kompleksitas komputasi secara signifikan

2.7 Self-Attention dan Positional Encoding

2.7.1 Self-Attention

Dalam self-attention, queries, keys, dan values semuanya berasal dari sequence yang sama:

$$\text{SelfAttention}(X) = \text{Attention}(XW^Q, XW^K, XW^V) \tag{14}$$

dimana $X \in \mathbb{R}^{n \times d}$ adalah input sequence.

2.7.2 Positional Encoding

Karena self-attention tidak memiliki notion of order, kita perlu menambahkan informasi posisi:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \tag{15}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right) \tag{16}$$

dimana:

- *pos* adalah posisi token dalam sequence

- i adalah dimensi
- d adalah dimensi model

Keuntungan Positional Encoding:

- Dapat menangani sequence dengan panjang arbitrary
- Mengandung informasi posisi absolut dan relatif
- Dapat direpresentasikan sebagai linear projection

2.8 Transformer Architecture

2.8.1 Arsitektur Lengkap

Transformer terdiri dari encoder dan decoder stack, masing-masing dengan komponen:

Encoder Layer:

1. Multi-head self-attention
2. Add & Norm (residual connection + layer normalization)
3. Position-wise feed-forward network
4. Add & Norm

Decoder Layer:

1. Masked multi-head self-attention
2. Add & Norm
3. Multi-head cross-attention (attend to encoder output)
4. Add & Norm
5. Position-wise feed-forward network
6. Add & Norm

2.8.2 Position-wise Feed-Forward Network

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{17}$$

atau dengan aktivasi GELU:

$$\text{FFN}(x) = \text{GELU}(xW_1 + b_1)W_2 + b_2 \tag{18}$$

2.8.3 Layer Normalization

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{19}$$

dimana μ dan σ^2 adalah mean dan variance dihitung across feature dimension.

2.8.4 Masked Attention

Dalam decoder, kita perlu mencegah attending ke future tokens:

$$\text{mask}_{ij} = \begin{cases} 0 & \text{if } i \geq j \\ -\infty & \text{if } i < j \end{cases} \tag{20}$$

Attention dengan mask:

$$\text{Attention}_{\text{masked}}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}} + M\right)V \tag{21}$$

2.8.5 Kompleksitas Komputasi

Layer Type	Complexity	Sequential	Max Path
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$

Tabel 1: Perbandingan kompleksitas berbagai layer types

dimana n adalah sequence length, d adalah dimensi representasi, dan k adalah kernel size.

3 Percobaan

3.1 Percobaan 1: Implementasi Attention Scoring Functions

3.1.1 Tujuan

Mengimplementasikan dan membandingkan berbagai attention scoring functions.

3.1.2 Langkah Percobaan

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

def softmax(x, axis=-1):
    """Compute softmax values"""
    exp_x = np.exp(x - np.max(x, axis=axis, keepdims=True))
    return exp_x / np.sum(exp_x, axis=axis, keepdims=True)

def scaled_dot_product_attention(Q, K, V, mask=None):
    """
    Scaled Dot-Product Attention

    Args:
        Q: Queries (batch_size, n_queries, d_k)
        K: Keys (batch_size, n_keys, d_k)
        V: Values (batch_size, n_keys, d_v)
        mask: Optional mask (batch_size, n_queries, n_keys)

    Returns:
        output: Attention output
        attention_weights: Attention weights
    """
    d_k = Q.shape[-1]

    # Compute attention scores
    scores = np.matmul(Q, K.transpose(0, 2, 1)) / np.sqrt(d_k)

    # Apply mask if provided
    if mask is not None:
        scores = scores + (mask * -1e9)

    # Compute attention weights
    attention_weights = softmax(scores, axis=-1)

    # Compute output
    output = np.matmul(attention_weights, V)

    return output, attention_weights

def additive_attention(Q, K, V, W_q, W_k, w_v):
    """
    Additive Attention (Bahdanau)

    Args:
        Q: Queries (batch_size, n_queries, d_q)
        K: Keys (batch_size, n_keys, d_k)
        V: Values (batch_size, n_keys, d_v)
        W_q: Query projection (d_q, h)
        W_k: Key projection (d_k, h)
        w_v: Score projection (h,)

    Returns:
        output: Attention output
        attention_weights: Attention weights
    """
    batch_size, n_queries, _ = Q.shape
    n_keys = K.shape[1]

    # Project queries and keys
    Q_proj = np.matmul(Q, W_q)  # (batch, n_queries, h)
    K_proj = np.matmul(K, W_k)  # (batch, n_keys, h)

    # Expand dimensions for broadcasting
    Q_exp = Q_proj[:, :, np.newaxis, :]  # (batch, n_queries, 1, h)
    K_exp = K_proj[:, np.newaxis, :, :]  # (batch, 1, n_keys, h)

    # Compute scores
```

```python
    features = np.tanh(Q_exp + K_exp)  # (batch, n_queries, n_keys,
        h)
    scores = np.dot(features, w_v)     # (batch, n_queries, n_keys)

    # Compute attention weights
    attention_weights = softmax(scores, axis=-1)

    # Compute output
    output = np.matmul(attention_weights, V)

    return output, attention_weights

# Test implementations
np.random.seed(42)

# Generate sample data
batch_size, n_queries, n_keys = 2, 4, 6
d_k, d_v, h = 8, 10, 16

Q = np.random.randn(batch_size, n_queries, d_k)
K = np.random.randn(batch_size, n_keys, d_k)
V = np.random.randn(batch_size, n_keys, d_v)

# Test Scaled Dot-Product Attention
output_sdp, weights_sdp = scaled_dot_product_attention(Q, K, V)
print("Scaled␣Dot-Product␣Attention:")
print(f"Output␣shape:␣{output_sdp.shape}")
print(f"Attention␣weights␣shape:␣{weights_sdp.shape}")
print(f"Attention␣weights␣sum:␣{weights_sdp.sum(axis=-1)}")

# Test Additive Attention
W_q = np.random.randn(d_k, h) * 0.1
W_k = np.random.randn(d_k, h) * 0.1
w_v = np.random.randn(h) * 0.1

output_add, weights_add = additive_attention(Q, K, V, W_q, W_k, w_v
    )
print("\nAdditive␣Attention:")
print(f"Output␣shape:␣{output_add.shape}")
print(f"Attention␣weights␣shape:␣{weights_add.shape}")
print(f"Attention␣weights␣sum:␣{weights_add.sum(axis=-1)}")

# Visualize attention weights
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

im1 = axes[0].imshow(weights_sdp[0], cmap='Blues', aspect='auto')
axes[0].set_title('Scaled␣Dot-Product␣Attention␣Weights')
axes[0].set_xlabel('Keys')
axes[0].set_ylabel('Queries')
plt.colorbar(im1, ax=axes[0])

```

```
im2 = axes[1].imshow(weights_add[0], cmap='Blues', aspect='auto')
axes[1].set_title('Additive␣Attention␣Weights')
axes[1].set_xlabel('Keys')
axes[1].set_ylabel('Queries')
plt.colorbar(im2, ax=axes[1])

plt.tight_layout()
plt.savefig('attention_comparison.png', dpi=300, bbox_inches='tight
    ')
plt.show()
```

Listing 1: Implementasi Scoring Functions

3.1.3 Analisis

1. Bandingkan pattern attention weights dari kedua metode
2. Perhatikan bahwa sum dari attention weights untuk setiap query adalah 1
3. Scaled dot-product lebih efisien secara komputasi
4. Additive attention lebih fleksibel untuk dimensi yang berbeda

3.2 Percobaan 2: Multi-Head Attention

3.2.1 Tujuan

Mengimplementasikan multi-head attention mechanism.

3.2.2 Langkah Percobaan

```
class MultiHeadAttention:
    def __init__(self, d_model, num_heads):
        """
        Multi-Head Attention

        Args:
            d_model: Dimension of the model
            num_heads: Number of attention heads
        """
        assert d_model % num_heads == 0, "d_model␣must␣be␣divisible
           ␣by␣num_heads"

        self.d_model = d_model
        self.num_heads = num_heads
        self.d_k = d_model // num_heads

        # Initialize projection matrices
        self.W_q = np.random.randn(d_model, d_model) * 0.1
        self.W_k = np.random.randn(d_model, d_model) * 0.1
        self.W_v = np.random.randn(d_model, d_model) * 0.1
        self.W_o = np.random.randn(d_model, d_model) * 0.1
```

```python

    def split_heads(self, x):
        """
        Split the last dimension into (num_heads, d_k)

        Args:
            x: Input tensor (batch_size, seq_len, d_model)

        Returns:
            Reshaped tensor (batch_size, num_heads, seq_len, d_k)
        """
        batch_size, seq_len, _ = x.shape
        x = x.reshape(batch_size, seq_len, self.num_heads, self.d_k)
        return x.transpose(0, 2, 1, 3)

    def combine_heads(self, x):
        """
        Combine heads back to original shape

        Args:
            x: Input tensor (batch_size, num_heads, seq_len, d_k)

        Returns:
            Combined tensor (batch_size, seq_len, d_model)
        """
        batch_size, _, seq_len, _ = x.shape
        x = x.transpose(0, 2, 1, 3)
        return x.reshape(batch_size, seq_len, self.d_model)

    def forward(self, Q, K, V, mask=None):
        """
        Forward pass of multi-head attention

        Args:
            Q: Queries (batch_size, seq_len_q, d_model)
            K: Keys (batch_size, seq_len_k, d_model)
            V: Values (batch_size, seq_len_v, d_model)
            mask: Optional mask

        Returns:
            output: Attention output
            attention_weights: Attention weights for all heads
        """
        batch_size = Q.shape[0]

        # Linear projections
        Q = np.matmul(Q, self.W_q)
        K = np.matmul(K, self.W_k)
        V = np.matmul(V, self.W_v)

        # Split into multiple heads
        Q = self.split_heads(Q)  # (batch, num_heads, seq_len_q,
            d_k)
        K = self.split_heads(K)  # (batch, num_heads, seq_len_k,
            d_k)
        V = self.split_heads(V)  # (batch, num_heads, seq_len_v,
            d_k)

        # Scaled dot-product attention
        scores = np.matmul(Q, K.transpose(0, 1, 3, 2)) / np.sqrt(
            self.d_k)

        if mask is not None:
            scores = scores + (mask * -1e9)

        attention_weights = softmax(scores, axis=-1)
        attention_output = np.matmul(attention_weights, V)

        # Combine heads
        output = self.combine_heads(attention_output)

        # Final linear projection
        output = np.matmul(output, self.W_o)

        return output, attention_weights

# Test Multi-Head Attention
np.random.seed(42)

d_model, num_heads = 512, 8
batch_size, seq_len = 2, 10

mha = MultiHeadAttention(d_model, num_heads)

Q = np.random.randn(batch_size, seq_len, d_model)
K = np.random.randn(batch_size, seq_len, d_model)
V = np.random.randn(batch_size, seq_len, d_model)

output, attention_weights = mha.forward(Q, K, V)

print(f"Input␣shape:␣{Q.shape}")
print(f"Output␣shape:␣{output.shape}")
print(f"Attention␣weights␣shape:␣{attention_weights.shape}")
print(f"Number␣of␣heads:␣{num_heads}")
print(f"d_k␣per␣head:␣{mha.d_k}")

# Visualize attention patterns for different heads
fig, axes = plt.subplots(2, 4, figsize=(16, 8))
axes = axes.flatten()

for i in range(num_heads):
    im = axes[i].imshow(attention_weights[0, i], cmap='viridis',
       aspect='auto')
    axes[i].set_title(f'Head␣{i+1}')
    axes[i].set_xlabel('Keys')
    axes[i].set_ylabel('Queries')
    plt.colorbar(im, ax=axes[i])

plt.tight_layout()
plt.savefig('multihead_attention.png', dpi=300, bbox_inches='tight'
    )
plt.show()
```

Listing 2: Multi-Head Attention Implementation

3.2.3 Analisis

1. Perhatikan bagaimana setiap head memiliki pattern attention yang berbeda
2. Setiap head dapat fokus pada aspek berbeda dari input
3. Multi-head attention meningkatkan ekspresivitas model
4. Total parameter meningkat tetapi kompleksitas per head berkurang

3.3 Percobaan 3: Positional Encoding

3.3.1 Tujuan

Mengimplementasikan dan memvisualisasikan positional encoding.

3.3.2 Langkah Percobaan

```
def positional_encoding(max_len, d_model):
    """
    Generate positional encoding

    Args:
        max_len: Maximum sequence length
        d_model: Dimension of the model

    Returns:
        pos_encoding: Positional encoding matrix (max_len, d_model)
    """
    pos_encoding = np.zeros((max_len, d_model))
    position = np.arange(0, max_len)[:, np.newaxis]
    div_term = np.exp(np.arange(0, d_model, 2) *
                      -(np.log(10000.0) / d_model))

    # Apply sin to even indices
    pos_encoding[:, 0::2] = np.sin(position * div_term)

    # Apply cos to odd indices
    pos_encoding[:, 1::2] = np.cos(position * div_term)

    return pos_encoding

# Generate positional encoding
max_len, d_model = 100, 512
pos_enc = positional_encoding(max_len, d_model)

print(f"Positional␣encoding␣shape:␣{pos_enc.shape}")

# Visualize positional encoding
plt.figure(figsize=(15, 5))

# Plot full positional encoding
plt.subplot(1, 2, 1)
plt.imshow(pos_enc.T, cmap='RdBu', aspect='auto')
plt.colorbar()
plt.xlabel('Position')
plt.ylabel('Dimension')
plt.title('Positional␣Encoding␣Heatmap')

# Plot specific dimensions over positions
plt.subplot(1, 2, 2)
dims_to_plot = [0, 1, 64, 65, 128, 129]
for dim in dims_to_plot:
    plt.plot(pos_enc[:, dim], label=f'Dim␣{dim}')
plt.xlabel('Position')
plt.ylabel('Value')
plt.title('Positional␣Encoding␣Values␣for␣Different␣Dimensions')
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('positional_encoding.png', dpi=300, bbox_inches='tight'
    )
plt.show()

# Test relative position property
def demonstrate_relative_position(pos_enc, offset=5):
    """
    Demonstrate that relative positions can be represented
    as linear transformations
    """
    position_i = 10
    position_j = position_i + offset

    # Get encodings
    pe_i = pos_enc[position_i]
    pe_j = pos_enc[position_j]

    # Compute similarity
    similarity = np.dot(pe_i, pe_j) / (np.linalg.norm(pe_i) *
                                        np.linalg.norm(pe_j))

    print(f"\nRelative␣Position␣Analysis:")
    print(f"Position␣{position_i}␣to␣{position_j}␣(offset={offset})")
    print(f"Cosine␣similarity:␣{similarity:.4f}")

    return similarity

# Test different offsets
offsets = range(0, 20)
similarities = [demonstrate_relative_position(pos_enc, offset)
                for offset in offsets]

plt.figure(figsize=(10, 5))
plt.plot(offsets, similarities, 'o-')
plt.xlabel('Position␣Offset')
plt.ylabel('Cosine␣Similarity')
plt.title('Positional␣Encoding␣Similarity␣vs␣Position␣Offset')
plt.grid(True, alpha=0.3)
plt.savefig('pe_similarity.png', dpi=300, bbox_inches='tight')
plt.show()
```

Listing 3: Positional Encoding

3.3.3 Analisis

1. Positional encoding menambahkan informasi posisi ke embeddings
2. Dimensi dengan frekuensi rendah berubah lambat across positions
3. Dimensi dengan frekuensi tinggi berubah cepat
4. Model dapat mempelajari relative positions dari pattern ini

3.4 Percobaan 4: Transformer Encoder Layer

3.4.1 Tujuan

Mengimplementasikan complete Transformer encoder layer.

3.4.2 Langkah Percobaan

```
class LayerNormalization:
    def __init__(self, features, eps=1e-6):
        self.eps = eps
        self.gamma = np.ones(features)
        self.beta = np.zeros(features)

    def forward(self, x):
        mean = np.mean(x, axis=-1, keepdims=True)
        std = np.std(x, axis=-1, keepdims=True)
        return self.gamma * (x - mean) / (std + self.eps) + self.
           beta

class PositionwiseFeedForward:
    def __init__(self, d_model, d_ff):
        self.W1 = np.random.randn(d_model, d_ff) * 0.1
        self.b1 = np.zeros(d_ff)
        self.W2 = np.random.randn(d_ff, d_model) * 0.1
        self.b2 = np.zeros(d_model)

    def forward(self, x):
        # First layer with ReLU
        hidden = np.maximum(0, np.matmul(x, self.W1) + self.b1)
        # Second layer
        output = np.matmul(hidden, self.W2) + self.b2
        return output

class TransformerEncoderLayer:
    def __init__(self, d_model, num_heads, d_ff, dropout=0.1):
        self.mha = MultiHeadAttention(d_model, num_heads)
        self.ffn = PositionwiseFeedForward(d_model, d_ff)
        self.layernorm1 = LayerNormalization(d_model)
        self.layernorm2 = LayerNormalization(d_model)
        self.dropout = dropout

    def forward(self, x, mask=None):
        # Multi-head attention with residual connection and layer
           norm
        attn_output, attn_weights = self.mha.forward(x, x, x, mask)

        # Dropout (simulated by scaling)
        attn_output = attn_output * (1 - self.dropout)

        # Add & Norm
        x = self.layernorm1.forward(x + attn_output)

        # Feed-forward with residual connection and layer norm
        ffn_output = self.ffn.forward(x)

        # Dropout
        ffn_output = ffn_output * (1 - self.dropout)

        # Add & Norm
        output = self.layernorm2.forward(x + ffn_output)

        return output, attn_weights

# Test Transformer Encoder Layer
np.random.seed(42)

```

```python
d_model, num_heads, d_ff = 512, 8, 2048
batch_size, seq_len = 2, 10

encoder_layer = TransformerEncoderLayer(d_model, num_heads, d_ff)

# Generate input with positional encoding
x = np.random.randn(batch_size, seq_len, d_model)
pos_enc = positional_encoding(seq_len, d_model)
x = x + pos_enc

# Forward pass
output, attn_weights = encoder_layer.forward(x)

print(f"Input shape: {x.shape}")
print(f"Output shape: {output.shape}")
print(f"Attention weights shape: {attn_weights.shape}")

# Visualize attention for first sample
plt.figure(figsize=(10, 8))
avg_attn = np.mean(attn_weights[0], axis=0)
plt.imshow(avg_attn, cmap='viridis', aspect='auto')
plt.colorbar(label='Attention Weight')
plt.xlabel('Key Position')
plt.ylabel('Query Position')
plt.title('Average Attention Weights Across All Heads')
plt.savefig('encoder_attention.png', dpi=300, bbox_inches='tight')
plt.show()
```

Listing 4: Transformer Encoder Layer

3.4.3 Analisis

1. Encoder layer menggabungkan multi-head attention dan feed-forward network
2. Residual connections membantu gradient flow
3. Layer normalization menstabilkan training
4. Setiap token dapat attend ke semua token lainnya

3.5 Percobaan 5: Sequence-to-Sequence dengan Attention

3.5.1 Tujuan

Mengimplementasikan complete sequence-to-sequence model dengan attention untuk machine translation.

3.5.2 Dataset

Gunakan dataset English-French translation pairs dari Tatoeba Project atau Manythings.org.

3.5.3 Langkah Percobaan

```python
import pandas as pd
from collections import Counter
import re

class Vocabulary:
    def __init__(self, freq_threshold=2):
        self.itos = {0: "<PAD>", 1: "<SOS>", 2: "<EOS>", 3: "<UNK>"}
        self.stoi = {"<PAD>": 0, "<SOS>": 1, "<EOS>": 2, "<UNK>": 3}
        self.freq_threshold = freq_threshold

    def build_vocabulary(self, sentence_list):
        frequencies = Counter()
        idx = 4

        for sentence in sentence_list:
            for word in sentence.split():
                frequencies[word] += 1

                if frequencies[word] == self.freq_threshold:
                    self.stoi[word] = idx
                    self.itos[idx] = word
                    idx += 1

    def numericalize(self, text):
        tokenized = text.split()
        return [self.stoi.get(token, self.stoi["<UNK>"])
                for token in tokenized]

    def __len__(self):
        return len(self.itos)

def preprocess_text(text):
    """Simple text preprocessing"""
    text = text.lower()
    text = re.sub(r"([?.!,])", r" \1 ", text)
    text = re.sub(r'[" "]+', " ", text)
    text = text.strip()
    return text

# Load and prepare data
def load_translation_data(file_path, num_examples=10000):
    """
    Load translation data
    Expected format: English\tFrench
    """
    data = pd.read_csv(file_path, sep='\t', header=None,
                       names=['english', 'french'], nrows=
                          num_examples)

    data['english'] = data['english'].apply(preprocess_text)
    data['french'] = data['french'].apply(preprocess_text)

    return data

# Simple Seq2Seq with Attention (using simplified encoder-decoder)
class SimpleSeq2SeqAttention:
    def __init__(self, src_vocab_size, tgt_vocab_size,
                 embed_size, hidden_size, num_heads=4):
        self.embed_size = embed_size
        self.hidden_size = hidden_size

        # Embeddings
        self.src_embedding = np.random.randn(src_vocab_size,
                                             embed_size) * 0.1
        self.tgt_embedding = np.random.randn(tgt_vocab_size,
                                             embed_size) * 0.1

        # Encoder (simplified as linear transformation)
        self.encoder_transform = np.random.randn(embed_size,
                                                 hidden_size) * 0.1

        # Attention mechanism
        self.attention = MultiHeadAttention(hidden_size, num_heads)

        # Decoder
        self.decoder_transform = np.random.randn(hidden_size,
                                                 hidden_size) * 0.1
        self.output_projection = np.random.randn(hidden_size,
                                                 tgt_vocab_size) *
                                                    0.1

    def encode(self, src_seq):
        """
        Encode source sequence

        Args:
            src_seq: Source sequence indices (batch_size, src_len)

        Returns:
            Encoder outputs (batch_size, src_len, hidden_size)
        """
        # Get embeddings
        src_embed = self.src_embedding[src_seq]

        # Add positional encoding
        pos_enc = positional_encoding(src_seq.shape[1], self.
           embed_size)
```

```python
        src_embed = src_embed + pos_enc

        # Transform to hidden size
        encoder_output = np.matmul(src_embed, self.encoder_transform)

        return encoder_output

    def decode_step(self, tgt_input, encoder_output, mask=None):
        """
        Single decoding step

        Args:
            tgt_input: Target input (batch_size, tgt_len, hidden_size)
            encoder_output: Encoder outputs (batch_size, src_len, hidden_size)
            mask: Optional attention mask

        Returns:
            decoder_output: Predictions (batch_size, tgt_len, tgt_vocab_size)
            attention_weights: Attention weights
        """
        # Cross-attention
        attended, attention_weights = self.attention.forward(
            tgt_input, encoder_output, encoder_output, mask
        )

        # Decoder transformation
        decoder_hidden = np.matmul(attended, self.decoder_transform)
        decoder_hidden = np.maximum(0, decoder_hidden)  # ReLU

        # Output projection
        decoder_output = np.matmul(decoder_hidden, self.output_projection)

        return decoder_output, attention_weights

    def predict(self, src_seq, max_len=20):
        """
        Generate translation

        Args:
            src_seq: Source sequence (batch_size, src_len)
            max_len: Maximum target length

        Returns:
            predictions: Generated sequence
        """
        # Encode
        encoder_output = self.encode(src_seq)

        batch_size = src_seq.shape[0]
        predictions = np.ones((batch_size, 1), dtype=int)  # Start with <SOS>

        for _ in range(max_len):
            # Get target embeddings
            tgt_embed = self.tgt_embedding[predictions]
            pos_enc = positional_encoding(predictions.shape[1],
                                          self.embed_size)
            tgt_embed = tgt_embed + pos_enc[:predictions.shape[1]]

            # Transform to hidden size
            tgt_hidden = np.matmul(tgt_embed, self.encoder_transform)

            # Decode
            output, _ = self.decode_step(tgt_hidden, encoder_output)

            # Get next token
            next_token = np.argmax(output[:, -1, :], axis=-1, keepdims=True)
            predictions = np.concatenate([predictions, next_token], axis=1)

            # Stop if all sequences generated <EOS>
            if np.all(next_token == 2):  # 2 is <EOS>
                break

        return predictions

# Example usage (with dummy data for demonstration)
print("Creating vocabularies and model...")

# Create dummy data
src_vocab_size, tgt_vocab_size = 5000, 5000
embed_size, hidden_size = 256, 512

model = SimpleSeq2SeqAttention(src_vocab_size, tgt_vocab_size,
                                embed_size, hidden_size)

# Test with random input
batch_size, src_len = 2, 10
src_seq = np.random.randint(0, src_vocab_size, (batch_size, src_len))

print(f"Source sequence shape: {src_seq.shape}")

```

```python
# Encode
encoder_output = model.encode(src_seq)
print(f"Encoder output shape: {encoder_output.shape}")

# Generate translation
predictions = model.predict(src_seq, max_len=15)
print(f"Predictions shape: {predictions.shape}")
print(f"Sample prediction: {predictions[0]}")
```

Listing 5: Seq2Seq dengan Attention

3.5.4 Analisis

1. Model dapat menangani sequences dengan panjang berbeda
2. Attention memungkinkan decoder fokus pada bagian relevan dari input
3. Positional encoding penting untuk mempertahankan informasi urutan
4. Model dapat di-extend dengan multiple layers untuk performa lebih baik

4 Latihan

4.1 Latihan 1: Attention Visualization

Tugas:

1. Implementasikan fungsi untuk memvisualisasikan attention weights sebagai heatmap
2. Gunakan dataset teks sederhana (misalnya kalimat dalam bahasa Indonesia)
3. Visualisasikan bagaimana attention weights berubah untuk berbagai query positions
4. Analisis pattern yang muncul

Pertanyaan:

1. Apa perbedaan pattern attention antara short-range dan long-range dependencies?
2. Bagaimana attention weights berubah untuk kata-kata yang berbeda jenis (noun, verb, dll)?

4.2 Latihan 2: Comparing Attention Mechanisms

Tugas:

1. Implementasikan tiga attention mechanisms:
   - Scaled Dot-Product Attention
   - Additive Attention
   - Multiplicative Attention (variant dari dot-product)

2. Bandingkan kompleksitas komputasi dan memory usage
3. Test pada sequence dengan panjang berbeda (10, 50, 100, 500)
4. Plot hasil comparison

Deliverables:

- Code implementation
- Computational complexity analysis
- Performance comparison plots
- Written analysis (min. 500 words)

4.3 Latihan 3: Multi-Head Attention Analysis

Tugas:

1. Implementasikan multi-head attention dengan jumlah heads yang berbeda (1, 2, 4, 8, 16)
2. Analisis bagaimana setiap head mempelajari pattern yang berbeda
3. Visualisasikan attention patterns untuk setiap head
4. Compute attention entropy untuk mengukur "fokus" dari attention

Attention Entropy Formula:

$$H = -\sum_{i=1}^{n} \alpha_i \log(\alpha_i) \tag{22}$$

Pertanyaan Analisis:

1. Apakah semua heads mempelajari pattern yang unik?
2. Bagaimana jumlah heads mempengaruhi kapasitas model?
3. Trade-off apa yang ada antara jumlah heads dan performa?

4.4 Latihan 4: Positional Encoding Variants

Tugas:

1. Implementasikan tiga jenis positional encoding:
   - Sinusoidal (original Transformer)
   - Learned positional embeddings
   - Relative positional encoding
2. Bandingkan performa pada task sequence classification
3. Analisis generalisasi ke sequence lengths yang tidak terlihat saat training

Relative Positional Encoding:

$$\text{RelPE}(i, j) = \text{clip}(i - j, -k, k) \tag{23}$$

dimana k adalah maximum relative distance.

4.5 Latihan 5: Mini Transformer untuk Text Classification

Tugas: Implementasikan complete Transformer encoder untuk sentiment analysis.

Spesifikasi:

- Dataset: IMDb Movie Reviews atau AG News
- Architecture:
  - Embedding layer (d_model = 256)
  - Positional encoding
  - 2 Transformer encoder layers
  - 4 attention heads
  - Feed-forward dimension = 1024
  - Classification head
- Training: 10 epochs
- Evaluation metrics: Accuracy, F1-score

Deliverables:

1. Complete code implementation
2. Training curves (loss and accuracy)
3. Attention visualization for sample predictions
4. Error analysis
5. Comparison dengan baseline model (e.g., LSTM)

4.6 Latihan 6: Masked Self-Attention

Tugas:

1. Implementasikan masked self-attention untuk language modeling
2. Generate causal attention mask
3. Implementasikan autoregressive generation
4. Test pada dataset teks sederhana

Causal Mask:

$$M_{ij} = \begin{cases} 0 & \text{if } i \geq j \\ -\infty & \text{if } i < j \end{cases} \tag{24}$$

Tasks:

1. Implement causal mask generation
2. Verify that future tokens are not attended
3. Generate text sequences autoregressively
4. Measure generation quality (perplexity)

5 Dataset Rekomendasi

5.1 Text Classification

1. **IMDb Movie Reviews**
   - URL: `https://ai.stanford.edu/~amaas/data/sentiment/`
   - Size: 50,000 reviews
   - Task: Binary sentiment classification
   - Format: Text files
2. **AG News**
   - URL: `https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dat`
   - Size: 120,000 training, 7,600 test
   - Task: 4-class news categorization
   - Categories: World, Sports, Business, Sci/Tech
3. **20 Newsgroups**
   - URL: `http://qwone.com/~jason/20Newsgroups/`
   - Size: 20,000 documents
   - Task: Multi-class classification (20 classes)
   - Format: Text documents

5.2 Machine Translation

1. **Tatoeba Translation Pairs**
   - URL: `https://tatoeba.org/en/downloads`
   - Languages: 300+ languages
   - Size: Varies by language pair
   - Format: Tab-separated values
2. **WMT Translation Datasets**
   - URL: `https://www.statmt.org/wmt22/`
   - Task: Various language pairs
   - Size: Millions of sentence pairs
   - Quality: Professional translations
3. **Multi30k**
   - URL: `https://github.com/multi30k/dataset`
   - Languages: English, German, French, Czech
   - Size: 31,000 image descriptions
   - Task: Multilingual image captioning

5.3 Sequence Labeling

1. **CoNLL 2003 NER**
   - URL: `https://www.clips.uantwerpen.be/conll2003/ner/`
   - Task: Named Entity Recognition
   - Tags: PER, LOC, ORG, MISC
   - Format: CoNLL format
2. **Universal Dependencies**
   - URL: `https://universaldependencies.org/`
   - Languages: 100+ languages
   - Task: POS tagging, dependency parsing
   - Format: CoNLL-U

5.4 Question Answering

1. **SQuAD 2.0**
   - URL: `https://rajpurkar.github.io/SQuAD-explorer/`
   - Size: 100,000+ questions
   - Task: Extractive QA
   - Format: JSON
2. **Natural Questions**
   - URL: `https://ai.google.com/research/NaturalQuestions`
   - Size: 300,000+ questions
   - Source: Real Google searches
   - Format: JSON

5.5 Indonesian Language Datasets

1. **IndoNLU**
   - URL: `https://github.com/indobenchmark/indonlu`
   - Tasks: Sentiment, NER, QA, etc.
   - Language: Indonesian
   - Size: Varies by task
2. **Indonesian News Dataset**
   - URL: `https://www.kaggle.com/datasets/scolianni/indonesian-news-dataset`
   - Size: 100,000+ articles
   - Task: Classification, summarization
   - Categories: Multiple news categories

6 Tips Implementasi

6.1 Memory Optimization

1. **Gradient Checkpointing**: Trade compute for memory
2. **Mixed Precision Training**: Use float16 untuk forward pass
3. **Gradient Accumulation**: Simulate larger batch sizes
4. **Efficient Attention**: Gunakan sparse atau linear attention untuk long sequences

6.2 Training Best Practices

1. **Learning Rate Warmup**: Mulai dengan learning rate kecil
2. **Label Smoothing**: Regularization technique untuk classification
3. **Dropout**: Apply pada attention dan feed-forward layers
4. **Layer Normalization**: Stabilize training
5. **Weight Initialization**: Xavier/He initialization

6.3 Debugging Strategies

1. Check attention weight sums (should be 1)
2. Verify mask is applied correctly
3. Monitor gradient norms
4. Visualize attention patterns
5. Start with small model and simple data

7 Evaluasi dan Penilaian

7.1 Komponen Penilaian

Komponen	**Bobot**	**Keterangan**
Kehadiran	10%	Partisipasi aktif
Percobaan 1-4	30%	Implementasi dan analisis
Latihan 1-3	20%	Problem solving
Latihan 4-6	25%	Advanced implementation
Laporan	15%	Dokumentasi dan analisis

Tabel 2: Distribusi penilaian praktikum

7.2 Kriteria Penilaian Laporan

1. **Kelengkapan** (25%)
   - Semua percobaan dan latihan diselesaikan
   - Code lengkap dan dapat dijalankan
   - Visualisasi informatif
2. **Analisis** (35%)
   - Pemahaman konsep mendalam
   - Interpretasi hasil yang tepat
   - Perbandingan metode yang komprehensif
3. **Implementasi** (25%)
   - Code efficiency
   - Proper error handling
   - Code documentation
4. **Presentasi** (15%)
   - Struktur laporan jelas
   - Visualisasi berkualitas
   - Penulisan profesional

8 Referensi

1. Vaswani, A., et al. (2017). "Attention Is All You Need." NeurIPS.
2. Bahdanau, D., Cho, K., & Bengio, Y. (2014). "Neural Machine Translation by Jointly Learning to Align and Translate." ICLR.
3. Devlin, J., et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." NAACL.
4. Dosovitskiy, A., et al. (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." ICLR.
5. Brown, T., et al. (2020). "Language Models are Few-Shot Learners." NeurIPS.
6. Dive into Deep Learning. `https://d2l.ai`
7. The Illustrated Transformer. `http://jalammar.github.io/illustrated-transformer/`
8. Attention? Attention! `https://lilianweng.github.io/posts/2018-06-24-attention/`

9 Lampiran

9.1 Lampiran A: Cheat Sheet Formulas

9.1.1 Attention Mechanisms

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{25}$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(h_1, \ldots, h_n)W^O \tag{26}$$

$$h_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{27}$$

9.1.2 Positional Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d}}\right) \tag{28}$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d}}\right) \tag{29}$$

9.1.3 Layer Normalization

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta \tag{30}$$

9.2 Lampiran B: Common Hyperparameters

Parameter	**Base**	**Large**
d_{model}	512	1024
d_{ff}	2048	4096
h (num heads)	8	16
N (num layers)	6	12
Dropout	0.1	0.1

Tabel 3: Hyperparameter standar untuk Transformer

9.3 Lampiran C: Troubleshooting Guide

Problem	**Solution**
NaN loss	Check learning rate, gradient clipping, initialization
Poor convergence	Increase warmup steps, adjust LR schedule
Out of memory	Reduce batch size, use gradient accumulation
Attention weights not summing to 1	Check softmax axis, mask application
No learning	Check loss function, verify gradients flowing

Tabel 4: Common problems dan solutions