State-of-the-Art in Deep Learning: Transformers

Chapter ini memberikan dekonstruksi matematis lengkap Transformer architecture yang merevolusi NLP dan computer vision. Transformer menggantikan RNN/LSTM dengan self- attention mechanism yang parallelizable dan context-aware. Bab ini menjelaskan setiap komponen dari scaled dot-product attention hingga multi-head attention dengan contoh implementasi. [1]

Text Representation dan Tokenization

Preprocessing pipeline:

```
# Tokenization + Vocabulary
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=10000)
tokenizer.fit_on_texts(corpus)
sequences = tokenizer.texts_to_sequences(corpus)

# Padding untuk fixed-length sequences
padded_sequences = tf.keras.preprocessing.sequence.pad_sequences(sequences, maxlen=100)
```

Embeddings + Positional Encoding:

```
class PositionalEncoding(tf.keras.layers.Layer):
    def __init__(self, max_len=5000, d_model=512):
        super().__init__()
        pe = 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(1e4) / d_model))
        pe[:, 0::2] = np.sin(position * div_term)
        pe[:, 1::2] = np.cos(position * div_term)
        self.pe = tf.constant(pe, dtype=tf.float32)

    def call(self, x):
        return x + self.pe[:tf.shape(x)[^1]]
```

Transformer Architecture: Mathematical Deep Dive

Self-Attention Mechanism Scaled Dot-Product):

```
Attention(Q, K, V) = softmax(QK^T / √d_k)V
```

Dimana Q Query, K Key, V Value matrices di-generate dari input embeddings.

Multi-Head Attention:

```
class MultiHeadAttention(tf.keras.layers.Layer): def __init__(self,
    d_model, num_heads):
        super().___init___() self.num_heads =
        num_heads self.d_model = d_model self.depth
        =d_model//num_heads

        self.wq = tf.keras.layers.Dense(d_model) self.wk =
        tf.keras.layers.Dense(d_model) self.wv =
        tf.keras.layers.Dense(d_model) self.dense=
        tf.keras.layers.Dense(d_model)

    def split_heads(self, x, batch_size):
        x=tf.reshape(x,(batch_size,-1,self.num_heads,self.depth)) return tf.transpose(x, perm=[0, 2,
        1, 3])

    defcall(self,q,k,v,mask=None): batch_size = tf.shape(q)[^0]

        Q=self.split_heads(self.wq(q),batch_size) K=
        self.split_heads(self.wk(k), batch_size)
        V=self.split_heads(self.wv(v), batch_size)

        # Scaled Dot-Product Attention
        matmul_qk=tf.matmul(Q,K,transpose_b=True) dk =
        tf.cast(tf.shape(K)[-1], tf.float32)
        scaled_attention = tf.nn.softmax(matmul_qk / tf.math.sqrt(dk), axis=-1)

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

        output=tf.matmul(scaled_attention,V)
        output=tf.transpose(output,perm=[0,2,1,3])
        return self.dense(tf.reshape(output, (batch_size, -1, self.d_model)))
```

Encoder-Decoder Stack: 6 encoder layers + 6 decoder layers dengan residual connections dan layer normalization.

Masked Self-Attention dan Training Considerations

Masked Attention Decoder): Mencegah peeking future tokens selama training:

```
def create_look_ahead_mask(size):
    mask = 1 - tf.linalg.band_part(tf.ones((size, size)), -1, 0)
    return mask
```

Loss Function: Label smoothing + cross-entropy untuk stabilisasi training.

Kesimpulan

Transformer architecture foundation untuk BERT, GPT, T5, dan Vision Transformers. Self- attention parallelizable 100x lebih cepat dari RNN untuk sequence length panjang.[1]

```
# Legacy ImageDataGenerator
train_datagen = ImageDataGenerator(
    rescale=1./255,
```