TensorFlow 2

Chapter ini mendalami core mechanics TensorFlow 2 yang merevolusi framework dengan eager execution default dan AutoGraph. Bab ini menjelaskan tiga building blocks fundamental (`tf.Variable`, `tf.Tensor`, `tf.Operation`) dan implementasi neural network operations dasar seperti matrix multiplication, convolution, dan pooling. Perbedaan mendasar dengan TF1 dijelaskan secara teknis. [1]

Eager Execution vs Graph Mode: Deep Dive

TensorFlow 1.x: Static computational graphs dengan placeholders dan sessions. Debugging sulit karena imperative code terpisah dari graph definition.

TensorFlow 2.x: Eager execution default (seperti NumPy). Operations dieksekusi immediately, debugging seperti Python biasa.

AutoGraph: `@tf.function` mengkonversi Python control flow (if, for, while) ke TensorFlow ops otomatis:

```
@tf.function
def quadratic_loss(y_true, y_pred):
    if tf.reduce_max(y_pred) > 1.0:
        return tf.reduce_mean(tf.square(y_pred - y_pred))
    return tf.reduce_mean(tf.square(y_pred - y_true))
```

Performance: `@tf.function` mengkompilasi Python ke optimized graph saat pertama dipanggil, kemudian reuse graph untuk input shapes serupa (graph tracing).

TensorFlow Building Blocks: Technical Details

1. `tf.Variable` Mutable Parameters)

Model parameters yang bisa di-update via gradient descent:

```
# Constant initialization
w = tf.Variable(tf.ones((784, 10)), trainable=True)

# Random initialization
w = tf.Variable(tf.random.normal((784, 10)), trainable=True)

# Update operations
w.assign(w * 0.9 + 0.1 * gradient)
w[0, :].assign(tf.zeros(10))  # Slicing updates
```

Attributes: `shape`, `dtype`, `trainable`, `name`. `.numpy()` untuk convert ke NumPy.

2. `tf.Tensor` Immutable Outputs)

Hasil operasi TensorFlow, immutable:

```
x = tf.constant([[1, 2], [3, 4]], dtype=tf.float32)
y = tf.matmul(x, x)  # tf.Tensor, shape=(2,2)
```

Tensor Types: DenseTensor, RaggedTensor (variable-length sequences), SparseTensor.

3. `tf.Operation` Computations)

Arithmetic (`tf.add`, `tf.matmul`), reductions (`tf.reduce_sum`, `tf.reduce_mean`), activations (`tf.nn.relu`). [1]

Neural Network Computations Implementation

Matrix Multiplication MLP forward pass):

```
X = tf.random.normal((32, 784))  # batch_size x features
W = tf.Variable(tf.random.normal((784, 10)))
b = tf.Variable(tf.zeros(10))
logits = tf.matmul(X, W) + b
```

Convolution CNN feature extraction):

```
inputs = tf.random.normal((32, 28, 28, 1))
filters = tf.Variable(tf.random.normal((3, 3, 1, 32)))
conv_out = tf.nn.conv2d(inputs, filters, strides=1, padding='SAME')
```

Pooling (spatial downsampling):

```
pool_out = tf.nn.max_pool2d(conv_out, ksize=2, strides=2, padding='SAME')
```

Complete MLP Example:

```
class SimpleMLP(tf.keras.Model):
    def __init__(self):
        super().__init__()
        self.W1 = tf.Variable(tf.random.normal((784, 128)))
        self.b1 = tf.Variable(tf.zeros(128))
        self.W2 = tf.Variable(tf.random.normal((128, 10)))
        self.b2 = tf.Variable(tf.zeros(10))

    @tf.function
    def call(self, X):
        h = tf.nn.relu(tf.matmul(X, self.W1) + self.b1)
        return tf.matmul(h, self.W2) + self.b2
```

Kesimpulan

Chapter 2 membangun pemahaman low-level TensorFlow yang esensial untuk semua model development. Master building blocks ini sebelum lanjut ke Keras high-level APIs. [1]