Bab 12: Custom Models and Training with TensorFlow

Tujuan Bab

Menjelaskan bagaimana membangun model kustom dan melakukan proses pelatihan manual menggunakan low-level API TensorFlow 2. Ini berguna saat model atau logika pelatihan tidak bisa diakomodasi oleh API fit() milik Keras.

Konsep Utama

1. TensorFlow sebagai Low-Level API

Keras menyediakan antarmuka yang mudah digunakan untuk pelatihan standar. Namun, saat memerlukan kontrol penuh—misalnya:

- Membuat *loss function* khusus
- Mengatur sendiri proses forward/backward
- Menggunakan metrik yang tidak tersedia di Keras

Maka, diperlukan pendekatan *low-level* menggunakan TensorFlow dan tf.GradientTape.

2. Custom Loss Functions and Metrics

Mendefinisikan fungsi loss sendiri:

```
def huber_loss(y_true, y_pred):
    error = y_true - y_pred
    is_small_error = tf.abs(error) < 1
    squared_loss = tf.square(error) / 2
    linear_loss  = tf.abs(error) - 0.5
    return tf.where(is_small_error, squared_loss, linear_loss)
```

3. Training Step Manual

Alih-alih menggunakan model.fit(), proses pelatihan ditulis manual:

```
with tf.GradientTape() as tape:
    y_pred = model(X_batch, training=True)
    loss = loss_fn(y_batch, y_pred)
grads = tape.gradient(loss, model.trainable_variables)
optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

4. Custom Training Loop

Loop pelatihan ditulis penuh untuk mengatur logika pelatihan dan validasi.

```
for epoch in range(n_epochs):
    for X_batch, y_batch in train_dataset:
        # step pelatihan seperti di atas
    # evaluasi validasi, logging, dll
```

5. Custom Training Step di Subclassing Model

Jika membuat model menggunakan subclass:

```
class MyModel(keras.Model):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)
        self.hidden = keras.layers.Dense(30, activation="relu")
        self.out = keras.layers.Dense(1)

    def call(self, inputs):
        return self.out(self.hidden(inputs))
```

Dapat juga menulis fungsi train_step() sendiri untuk menggantikan perilaku .fit():

```
def train_step(self, data):
  X, y = data
  with tf.GradientTape() as tape:
    y_pred = self(X, training=True)
    loss = self.compiled_loss(y, y_pred)
  grads = tape.gradient(loss, self.trainable_variables)
  self.optimizer.apply_gradients(zip(grads, self.trainable_variables))
  self.compiled_metrics.update_state(y, y_pred)
  return {m.name: m.result() for m in self.metrics}
```

6. Saving and Restoring Checkpoints (tf.train.Checkpoint)

Berfungsi untuk menyimpan bobot model dan optimizer secara manual:

```
checkpoint = tf.train.Checkpoint(model=model, optimizer=optimizer)
checkpoint.save("/tmp/my_model")
```

7. TF Functions (tf.function)

Digunakan untuk mempercepat eksekusi kode dengan mengompilasi Python menjadi grafik TensorFlow.

```
@tf.function
def train_step(...):
  ...
```

Proyek / Notebook Praktik

Isi:

- Membuat model dengan subclassing keras.Model
- Menulis fungsi loss dan metrik khusus
- Mengatur proses pelatihan manual dengan tf.GradientTape
- Menyimpan dan meload model dengan Checkpoint
- Mempercepat pelatihan dengan @tf.function

Inti Pelajaran

Konsep	Penjelasan
Subclassing Model	Membuat arsitektur model dengan fleksibilitas penuh
GradientTape	Alat untuk merekam operasi agar bisa menghitung gradien
Custom Training Loop	Diperlukan saat kontrol penuh terhadap pelatihan dibutuhkan
tf.function	Meningkatkan performa dengan mengompilasi fungsi
Checkpoint	Menyimpan status model dan optimizer secara manual

Kesimpulan

Bab ini membuka kemungkinan eksplorasi penuh terhadap TensorFlow dengan menjauh dari pendekatan abstrak ala Keras. Ini sangat penting untuk riset, debugging, atau membangun model-model unik seperti GAN, meta-learning, atau reinforcement learning, yang tidak cocok ditangani dengan fit() saja.