Chapter 12: Model Kustom dan Pelatihan dengan TensorFlow

Pendahuluan

Meskipun Keras API menyediakan antarmuka tingkat tinggi yang sangat nyaman untuk membangun dan melatih model Jaringan Saraf Tiruan (JST), terkadang kita memerlukan fleksibilitas yang lebih besar. Untuk keperluan penelitian, implementasi arsitektur yang tidak konvensional, atau untuk mendapatkan kontrol penuh atas proses pelatihan, kita perlu turun ke level yang lebih rendah. Chapter ini membahas bagaimana menggunakan fungsionalitas inti dari TensorFlow untuk membuat komponen model yang sepenuhnya kustom—seperti *layer*, *loss function*, dan model itu sendiri—serta cara membangun *loop* pelatihan dari awal.

1. Dasar-Dasar TensorFlow

Sebelum masuk ke kustomisasi, penting untuk memahami komponen dasar TensorFlow:

- **Tensor**: Struktur data fundamental di TensorFlow. Tensor adalah array multi-dimensi, sangat mirip dengan ndarray dari NumPy. Kelebihannya adalah Tensor dapat dipercepat menggunakan GPU/TPU dan mendukung diferensiasi otomatis (*automatic differentiation*).
- **Operasi**: TensorFlow menyediakan pustaka operasi matematika yang kaya (misalnya tf.add, tf.matmul, tf.reduce_mean) yang dapat dijalankan pada Tensor. Operasi-operasi ini sangat dioptimalkan untuk performa.
- **Variabel**: Saat kita membutuhkan Tensor yang nilainya dapat dimodifikasi (seperti bobot dan bias model), kita menggunakan tf.Variable.

2. Kustomisasi Komponen Model Keras

Keras API dirancang untuk bisa diperluas. Hampir setiap bagian dari model standar dapat dikustomisasi.

Fungsi Loss Kustom

Kita dapat membuat fungsi *loss* sendiri dengan mendefinisikan sebuah fungsi Python sederhana yang menerima dua argumen: label sebenarnya (y_true) dan prediksi model (y_pred). Fungsi ini harus mengembalikan sebuah Tensor yang berisi nilai *loss* untuk setiap instance.

Aktivasi, Inisialisasi, Regularisasi, dan Constraint Kustom

Sama seperti fungsi *loss*, komponen-komponen ini juga dapat dibuat sebagai fungsi Python sederhana dan kemudian dilewatkan sebagai argumen saat membuat sebuah *layer*. Misalnya, kita bisa membuat fungsi aktivasi sendiri dan menggunakannya dengan argumen activation=nama_fungsi_kustom.

Layer Kustom

Untuk arsitektur yang benar-benar baru, kita mungkin perlu membuat *layer* (lapisan) sendiri.

- **Layer Tanpa Bobot**: Jika *layer* hanya melakukan transformasi sederhana tanpa parameter yang perlu dilatih (misalnya, exp(x)), kita cukup membungkusnya dalam sebuah keras.layers.Lambda.
- **Layer Dengan Bobot**: Untuk *layer* yang memiliki bobot yang dapat dilatih, kita harus membuat sebuah kelas yang mewarisi dari keras.layers.Layer. Kelas ini memiliki tiga metode utama:
    1. **init()**: Konstruktor, tempat kita mendefinisikan *hyperparameter* atau sub-lapisan.
    2. **build(input_shape)**: Metode ini dipanggil saat *layer* pertama kali digunakan. Di sinilah bobot-bobot *layer* dibuat (menggunakan metode self.add_weight()). Keuntungannya adalah ukuran bobot dapat ditentukan secara dinamis berdasarkan bentuk input.
    3. **call(inputs)**: Metode ini berisi logika utama dari *layer*, yaitu operasi yang dilakukan pada input untuk menghasilkan output (langkah *forward pass*).

Model Kustom

Mirip dengan membuat *layer* kustom, kita juga dapat membuat model kustom dengan membuat kelas yang mewarisi dari keras.Model. Ini memberikan fleksibilitas penuh untuk mendefinisikan bagaimana *layer-layer* berinteraksi, sangat berguna untuk arsitektur yang kompleks dengan beberapa input, output, atau alur data yang rumit.

3. Gradien dan Autodiff

Inti dari pelatihan JST adalah Gradient Descent, yang membutuhkan perhitungan gradien dari *loss function* terhadap semua parameter model. TensorFlow menyederhanakan proses ini melalui **Automatic Differentiation** atau **Autodiff**.

tf.GradientTape

Alat utama untuk Autodiff di TensorFlow adalah tf.GradientTape. Cara kerjanya adalah sebagai berikut:

1. Buat sebuah blok konteks with tf.GradientTape() as tape:.
2. Semua operasi yang melibatkan tf.Variable (seperti bobot model) di dalam blok ini akan "direkam" oleh tape.
3. Setelah blok selesai, kita bisa memanggil metode tape.gradient(target, sources) untuk menghitung gradien dari target (misalnya, *loss*) terhadap sources (misalnya, daftar parameter model yang dapat dilatih).

TensorFlow akan secara otomatis menghitung gradien ini dengan efisien menggunakan *reverse-mode autodiff*, yang merupakan mekanisme di balik *backpropagation*.

4. Loop Pelatihan Kustom

Meskipun metode model.fit() sangat nyaman, terkadang kita membutuhkan kontrol lebih. Dengan GradientTape, kita dapat menulis *loop* pelatihan kita sendiri dari awal.

Langkah-langkah dalam Loop Pelatihan Kustom:

1. Buat *loop* luar untuk setiap **epoch**.
2. Buat *loop* dalam untuk setiap **batch** data pelatihan.
3. Di dalam *loop* batch: a. Ambil satu batch data (**X**, **y**). b. Buka blok tf.GradientTape. c. Lakukan *forward pass*: dapatkan prediksi model (**y_pred** = model(**X**)). d. Hitung *loss* utama: loss = loss_function(**y**, **y_pred**). e. (Opsional) Tambahkan *loss* regularisasi ke *loss* utama. f. Setelah blok *tape*, hitung gradien: gradients = tape.gradient(loss, model.trainable_variables). g. Terapkan gradien ini ke bobot model menggunakan sebuah *optimizer*: optimizer.apply_gradients(zip(gradients, model.trainable_variables)).
4. Di akhir setiap epoch, hitung metrik pada set validasi dan tampilkan hasilnya.

5. Fungsi dan Grafik TensorFlow (TF Functions and Graphs)

Kode Python bersifat fleksibel tetapi relatif lambat. Untuk mendapatkan performa maksimal, TensorFlow dapat mengubah fungsi Python menjadi **grafik komputasi** (*computation graph*) yang sangat dioptimalkan.

tf.function

Ini adalah sebuah *decorator* (@tf.function) yang dapat kita tambahkan pada fungsi Python. Saat fungsi ini pertama kali dipanggil, TensorFlow akan:

1. **Menelusuri (Trace)** fungsi tersebut untuk mengidentifikasi semua operasi TensorFlow di dalamnya.
2. Membangun sebuah grafik komputasi statis dari operasi-operasi tersebut.
3. Mengoptimalkan grafik ini.

Pada pemanggilan berikutnya, TensorFlow akan langsung menjalankan grafik yang sudah dioptimalkan ini, melewati interpreter Python dan menghasilkan peningkatan kecepatan yang sangat signifikan. *Loop* pelatihan kustom yang dibungkus dalam @tf.function adalah cara standar untuk mendapatkan performa terbaik.

Kesimpulan

Chapter 12 membuka "kotak hitam" dari Keras dan TensorFlow, memberikan kita alat untuk bergerak melampaui model-model standar. Dengan memahami cara membuat *layer*, *loss*, dan model kustom, serta mengontrol proses pelatihan secara manual menggunakan GradientTape dan mengoptimalkannya dengan tf.function, kita mendapatkan kekuatan dan fleksibilitas untuk merancang, mengimplementasikan, dan bereksperimen dengan hampir semua arsitektur Jaringan Saraf Tiruan yang bisa dibayangkan.