Chapter 10: Pengenalan Jaringan Saraf Tiruan dengan Keras

Pendahuluan

Chapter 10 menandai transisi dari Machine Learning klasik ke dunia **Deep Learning**. Bab ini memperkenalkan fondasi dari sebagian besar model Deep Learning: **Artificial Neural Networks (ANN)** atau Jaringan Saraf Tiruan (JST). Konsep ini terinspirasi oleh jaringan neuron biologis di otak manusia, yang bertujuan untuk meniru cara otak memproses informasi. Setelah melalui periode "musim dingin AI" (*AI winter*), ANN kembali populer berkat tiga faktor utama: ketersediaan dataset yang sangat besar, peningkatan besar dalam daya komputasi (terutama melalui GPU), dan perbaikan algoritma pelatihan.

1. Dari Neuron Biologis ke Neuron Artifisial

Perceptron

Bentuk JST yang paling awal dan paling sederhana adalah **Perceptron**, yang ditemukan pada tahun 1957. Perceptron terdiri dari satu unit komputasi tunggal yang disebut *neuron* atau *node*. Neuron ini menerima sejumlah input, masing-masing memiliki bobot (*weight*) terkait, dan menghasilkan satu output biner.

Struktur dan Prediksi:

Output dari Perceptron dihitung dengan terlebih dahulu menghitung jumlah terbobot dari input, ditambah sebuah bias term. Hasilnya kemudian dilewatkan melalui sebuah fungsi aktivasi langkah (step function).

- **Rumus Perhitungan:** $z = (\mathbf{w} \cdot \mathbf{x}) + b = (w_1x_1 + w_2x_2 + \dots + w_nx_n) + b$
- **Rumus Prediksi:** $\hat{y} = \text{step}(z)$

di mana **w** adalah vektor bobot, **x** adalah vektor fitur input, *b* adalah bias, dan $\hat{y}$ adalah prediksi output.

Pelatihan Perceptron:

Perceptron dilatih dengan aturan sederhana: ia memproses instance satu per satu, dan untuk setiap instance, ia membuat prediksi. Jika prediksi salah, ia akan memperbarui bobot koneksi untuk mengurangi kesalahan tersebut.

- **Aturan Pembaruan Bobot:** w_i (langkah selanjutnya) = w_i (saat ini) + $\eta * (y - \hat{y}) * x_i$

di mana w_i adalah bobot ke-i, η adalah *learning rate*, y adalah label target yang sebenarnya, $\hat{y}$ adalah prediksi, dan x_i adalah nilai input ke-i.

Keterbatasan Perceptron:

Pada tahun 1969, Marvin Minsky dan Seymour Papert menunjukkan dalam buku mereka, Perceptrons, bahwa model ini memiliki keterbatasan yang serius. Perceptron terbukti tidak mampu menyelesaikan beberapa masalah yang sangat sederhana, seperti masalah XOR. Mereka hanya dapat menyelesaikan masalah yang dapat dipisahkan secara linear (linearly separable). Keterbatasan ini menyebabkan kekecewaan besar dan menjadi salah satu pemicu "musim dingin AI".

2. Multi-Layer Perceptron dan Backpropagation

Keterbatasan Perceptron dapat diatasi dengan menumpuk beberapa lapis neuron. Arsitektur ini disebut **Multi-Layer Perceptron (MLP)**.

Arsitektur MLP

Sebuah MLP terdiri dari:

1. Satu **Input Layer** (Lapisan Input).
2. Satu atau lebih **Hidden Layers** (Lapisan Tersembunyi).
3. Satu **Output Layer** (Lapisan Output).

Setiap lapisan, kecuali lapisan output, memiliki sebuah *bias neuron* dan terhubung sepenuhnya ke lapisan berikutnya. Ketika sebuah ANN memiliki dua atau lebih lapisan tersembunyi, ia disebut sebagai **Deep Neural Network (DNN)**.

Fungsi Aktivasi

Untuk melatih MLP, fungsi aktivasi langkah tidak dapat digunakan karena gradiennya nol di mana-mana (kecuali di titik nol), sehingga algoritma Gradient Descent tidak dapat bekerja. Oleh karena itu, fungsi aktivasi non-linear yang dapat diturunkan (*differentiable*) diperlukan.

- **Fungsi Logistik (Sigmoid):** $\sigma(z) = 1 / (1 + \exp(-z))$. Menghasilkan output antara 0 dan 1.
- **Fungsi Hyperbolic Tangent (tanh):** $\tanh(z) = (\exp(z) – \exp(-z)) / (\exp(z) + \exp(-z))$. Menghasilkan output antara -1 dan 1.
- **Fungsi Rectified Linear Unit (ReLU):** $\text{ReLU}(z) = \max(0, z)$. Fungsi ini sangat populer karena lebih cepat secara komputasi dan tidak mengalami masalah gradien jenuh (*saturating gradients*) untuk nilai positif.

Algoritma Backpropagation

Backpropagation pada dasarnya adalah algoritma **Gradient Descent** yang dioptimalkan untuk ANN. Algoritma ini melatih model dalam dua langkah berulang:

1. **Forward Pass (Langkah Maju):** Data input dimasukkan ke lapisan input dan mengalir maju melalui lapisan-lapisan tersembunyi hingga ke lapisan output. Di setiap lapisan, perhitungan jumlah terbobot dan fungsi aktivasi dilakukan. Hasil akhir dari langkah ini adalah prediksi dari jaringan.
2. **Backward Pass (Langkah Mundur):**
    - Pertama, algoritma mengukur kesalahan model menggunakan sebuah *loss function* (misalnya, Cross-Entropy).
    - Kemudian, ia menghitung seberapa besar kontribusi setiap koneksi di lapisan terakhir terhadap kesalahan tersebut (gradiennya).
    - Dengan menggunakan **aturan rantai (chain rule)** dari kalkulus, ia menyebarkan gradien kesalahan ini secara mundur melalui jaringan, dari lapisan output ke lapisan input.

    - Terakhir, algoritma melakukan langkah Gradient Descent untuk memperbarui semua bobot koneksi di jaringan menggunakan gradien yang telah dihitung.

Proses ini diulang berkali-kali (disebut *epochs*) hingga model konvergen.

3. Implementasi MLP untuk Tugas Spesifik

Arsitektur MLP, terutama lapisan output dan *loss function*-nya, perlu disesuaikan tergantung pada jenis tugas yang dihadapi.

MLP untuk Regresi

- **Lapisan Output**: Untuk memprediksi satu nilai (regresi univariat), lapisan output hanya memiliki **satu neuron**. Untuk memprediksi beberapa nilai (regresi multivariat), diperlukan satu neuron output per dimensi output.
- **Fungsi Aktivasi Output**: Umumnya, **tidak ada fungsi aktivasi** yang digunakan di lapisan output, sehingga output dapat memiliki rentang nilai apa pun.
- **Loss Function**: *Loss function* yang umum digunakan adalah **Mean Squared Error (MSE)**.

MLP untuk Klasifikasi

- **Klasifikasi Biner**:
    - **Lapisan Output**: Hanya membutuhkan **satu neuron output**.
    - **Fungsi Aktivasi Output**: Menggunakan fungsi **logistik (sigmoid)** untuk menghasilkan output antara 0 dan 1, yang dapat diinterpretasikan sebagai probabilitas kelas positif.
    - **Loss Function**: Menggunakan **Binary Cross-Entropy**.
- **Klasifikasi Multikelas**:
    - **Lapisan Output**: Membutuhkan **satu neuron output per kelas**.
    - **Fungsi Aktivasi Output**: Menggunakan fungsi **softmax**, yang memastikan bahwa semua probabilitas output dari setiap neuron berjumlah 1.
    - **Loss Function**: Menggunakan **Categorical Cross-Entropy**.

4. Abstraksi API Keras

Keras adalah API Deep Learning tingkat tinggi yang menyediakan antarmuka yang sederhana dan intuitif untuk membangun dan melatih model jaringan saraf. Keras menawarkan beberapa paradigma untuk membangun model:

- **Sequential API**: Cara paling sederhana untuk membangun model. Model didefinisikan sebagai tumpukan lapisan linear, satu per satu. Ini sangat cocok untuk sebagian besar kasus umum, tetapi tidak fleksibel untuk arsitektur yang kompleks.

- **Functional API**: Pendekatan yang lebih fleksibel yang memungkinkan pembangunan arsitektur non-linear, model dengan beberapa input atau output, atau model dengan lapisan yang digunakan bersama (*shared layers*).
- **Subclassing API**: Pendekatan paling fleksibel di mana pengguna membuat *subclass* dari kelas Model Keras. Ini memberikan kontrol penuh atas setiap aspek model dan sangat berguna untuk penelitian atau saat membangun arsitektur yang sangat dinamis dan kompleks.