Chapter 15: Memproses Urutan Menggunakan RNN dan CNN

Pendahuluan

Hingga saat ini, kita telah membahas arsitektur jaringan saraf yang hanya memproses data secara statis tanpa mempertimbangkan urutan waktu, seperti *Multi-Layer Perceptrons* (MLP) dan *Convolutional Neural Networks* (CNN). Namun, banyak jenis data di dunia nyata yang bersifat **sekuensial** atau berurutan, seperti data deret waktu (misalnya, harga saham), teks (urutan kata), audio (urutan sampel suara), dan video. Chapter ini memperkenalkan **Recurrent Neural Networks (RNNs)**, sebuah kelas jaringan yang dirancang khusus untuk menangani data sekuensial dengan memiliki semacam "memori" dari input sebelumnya. Selain itu, bab ini juga membahas bagaimana arsitektur berbasis CNN dapat diadaptasi untuk memproses urutan secara efisien.

1. Recurrent Neural Networks (RNNs)

Inovasi utama dari RNN adalah **neuron rekuren**. Tidak seperti neuron biasa yang hanya menerima input dari lapisan sebelumnya, neuron rekuren juga menerima **outputnya sendiri dari langkah waktu (time step) sebelumnya**. Adanya umpan balik atau *loop* ini memungkinkan jaringan untuk mempertahankan informasi dari masa lalu, yang berfungsi sebagai bentuk memori.

Mekanisme Kerja RNN

Sebuah lapisan RNN memproses urutan data satu per satu. Pada setiap langkah waktu *t*, lapisan tersebut menerima dua input:

1. Vektor input untuk langkah waktu saat ini, **x**(t).
2. Vektor output dari langkah waktu sebelumnya, **h**(t-1), yang juga dikenal sebagai **hidden state** (keadaan tersembunyi).

Kedua input ini kemudian digabungkan untuk menghasilkan output baru, **h**(t), yang kemudian diteruskan ke langkah waktu berikutnya dan juga dapat digunakan untuk menghasilkan prediksi. Proses ini dapat diringkas dalam rumus berikut:

- **Rumus Hidden State:** $\mathbf{h}(t) = \phi(\mathbf{W}\_xh \cdot \mathbf{x}(t) + \mathbf{W}\_hh \cdot \mathbf{h}(t-1) + \mathbf{b}\_h)$

Di mana:

- **h**(t) adalah *hidden state* pada langkah waktu *t*.
- **x**(t) adalah vektor input pada langkah waktu *t*.
- **h**(t-1) adalah *hidden state* dari langkah waktu sebelumnya.
- **W**_xh adalah matriks bobot untuk koneksi dari input.
- **W**_hh adalah matriks bobot untuk koneksi rekuren (dari *hidden state* sebelumnya).
- **b**_h adalah vektor bias.
- ϕ adalah fungsi aktivasi, biasanya tanh.

Penting untuk dicatat bahwa bobot **W**_xh dan **W**_hh digunakan bersama (*shared*) di semua langkah waktu.

Pelatihan RNN: Backpropagation Through Time (BPTT)

Untuk melatih RNN, jaringan "dibuka" atau "di-unroll" sepanjang urutan waktu, mengubahnya menjadi jaringan saraf *feedforward* yang sangat dalam, di mana setiap lapisan sesuai dengan satu langkah waktu. Setelah di-unroll, algoritma *backpropagation* standar dapat diterapkan. Proses ini disebut **Backpropagation Through Time (BPTT)**.

2. Tantangan Memori Jangka Panjang

RNN sederhana memiliki keterbatasan signifikan: mereka kesulitan menangani **dependensi jangka panjang** (*long-term dependencies*). Karena gradien harus disebarkan melalui banyak langkah waktu selama BPTT, mereka cenderung menjadi sangat kecil (*vanishing gradients*), mirip dengan masalah pada DNN yang sangat dalam. Akibatnya, RNN kesulitan untuk mengingat informasi dari langkah waktu yang sangat jauh di masa lalu.

3. Varian RNN Lanjutan

Untuk mengatasi masalah memori jangka pendek, beberapa arsitektur sel rekuren yang lebih canggih telah dikembangkan.

LSTM (Long Short-Term Memory)

Sel LSTM adalah varian RNN yang sangat sukses dan populer. Ia dirancang khusus untuk mengingat informasi dalam jangka waktu yang lama. Inovasi utamanya adalah penggunaan **mekanisme gerbang (gate mechanism)** yang mengatur aliran informasi. Sebuah sel LSTM memiliki dua vektor keadaan:

1. **h**(t): *Hidden state* jangka pendek.
2. **c**(t): *Cell state* atau memori jangka panjang.

Mekanisme gerbang mengontrol bagaimana informasi ditambahkan atau dihapus dari *cell state*:

- **Forget Gate (Gerbang Lupa)**: Menentukan informasi apa dari *cell state* sebelumnya (**c**(t-1)) yang harus dibuang atau dilupakan.
- **Input Gate (Gerbang Masukan)**: Menentukan informasi baru apa dari input saat ini yang relevan dan harus disimpan ke dalam *cell state*.
- **Output Gate (Gerbang Keluaran)**: Menentukan bagian mana dari *cell state* yang akan dikeluarkan sebagai *hidden state* baru (**h**(t)).

Setiap gerbang ini adalah lapisan *dense* kecil dengan fungsi aktivasi sigmoid, yang menghasilkan nilai antara 0 (tutup gerbang) dan 1 (buka gerbang), sehingga secara dinamis mengontrol aliran informasi.

GRU (Gated Recurrent Unit)

Sel GRU adalah versi yang lebih sederhana dari LSTM yang seringkali memberikan performa serupa. GRU menggabungkan *cell state* dan *hidden state* menjadi satu vektor tunggal. Ia juga hanya memiliki dua gerbang:

- **Update Gate**: Mengontrol seberapa banyak informasi dari *hidden state* sebelumnya yang harus dipertahankan.
- **Reset Gate**: Mengontrol seberapa banyak informasi dari *hidden state* sebelumnya yang harus dilupakan saat menghitung kandidat output baru.

Karena arsitekturnya yang lebih sederhana, GRU sedikit lebih cepat secara komputasi daripada LSTM.

4. Memproses Urutan dengan CNN

Meskipun RNN tampaknya pilihan alami untuk data sekuensial, arsitektur berbasis CNN juga bisa sangat efektif dan seringkali jauh lebih cepat.

1D Convolutional Layers

Alih-alih menggunakan filter 2D seperti pada gambar, kita dapat menggunakan **filter 1D** yang meluncur di atas urutan data. Filter ini dapat mengenali pola-pola lokal atau *n-gram* dalam urutan.

Menangkap Dependensi Jangka Panjang dengan CNN

Untuk membuat CNN dapat melihat pola jangka panjang, kita bisa menumpuk banyak lapisan konvolusional. Namun, cara yang lebih efisien adalah dengan menggunakan **dilated convolutions**. *Dilation* (pelebaran) memungkinkan filter untuk "melompati" beberapa input pada setiap langkahnya. Dengan meningkatkan tingkat *dilation* secara eksponensial di lapisan yang lebih dalam, bidang reseptif jaringan dapat tumbuh sangat cepat, memungkinkannya untuk menghubungkan input yang sangat jauh tanpa memerlukan banyak lapisan.

Arsitektur **WaveNet**, yang terdiri dari tumpukan lapisan 1D *dilated convolutional*, adalah contoh utama dari pendekatan ini. Keuntungan besar CNN dibandingkan RNN adalah bahwa perhitungannya dapat **diparalelkan sepenuhnya**, tidak seperti RNN yang harus memproses urutan langkah demi langkah. Ini membuat pelatihan CNN pada urutan data menjadi jauh lebih cepat.