Chapter 16: Natural Language Processing dengan RNN dan Attention

Pendahuluan

Natural Language Processing (NLP) adalah cabang dari Kecerdasan Buatan yang berfokus pada interaksi antara komputer dan bahasa manusia.[1] Chapter ini mengaplikasikan model sekuensial yang telah kita pelajari (seperti RNN) untuk tugas-tugas NLP dan memperkenalkan dua konsep yang telah merevolusi bidang ini: mekanisme Attention dan arsitektur Transformer. Langkah pertama dalam setiap tugas NLP adalah mengubah teks menjadi angka yang dapat diproses oleh model. Proses ini biasanya melibatkan tokenisasi (memecah teks menjadi unit-unit seperti kata atau sub-kata) dan numerikalisasi (mengonversi setiap token menjadi representasi numerik).

1. Representasi Teks: Word Embeddings

Merepresentasikan kata sebagai angka unik atau vektor *one-hot* tidaklah efisien untuk kosakata yang besar dan tidak menangkap makna semantik. Solusi standar yang digunakan saat ini adalah Word Embeddings.

Konsep Inti:

Word embedding adalah representasi setiap kata dalam sebuah kosakata sebagai vektor numerik yang padat (dense vector) dan berdimensi relatif rendah (misalnya, 300 dimensi). Vektor-vektor ini dapat dilatih, dan selama pelatihan, model akan belajar menempatkan kata-kata yang memiliki makna serupa berdekatan satu sama lain di dalam ruang vektor.

Pre-trained Word Embeddings:

Alih-alih melatih embedding dari awal, seringkali lebih efektif untuk menggunakan pre-trained embeddings seperti GloVe atau Word2Vec. Embedding ini telah dilatih pada korpus teks yang sangat besar (seperti seluruh Wikipedia) dan telah menangkap hubungan semantik dan sintaksis yang kaya antar kata. Menggunakan embedding ini memberikan "pengetahuan awal" yang sangat berharga bagi model kita.

2. Arsitektur Sequence-to-Sequence (Seq2Seq)

Untuk tugas yang memetakan satu urutan ke urutan lain (misalnya, terjemahan mesin dari satu bahasa ke bahasa lain), arsitektur standar yang digunakan adalah model Encoder-Decoder.

Struktur Encoder-Decoder:

1. Encoder: Sebuah RNN (biasanya LSTM atau GRU) yang membaca kalimat input kata demi kata. Pada setiap langkah, ia memperbarui *hidden state*-nya. Setelah memproses seluruh kalimat input, *hidden state* terakhir dari Encoder dianggap sebagai representasi ringkas dari seluruh kalimat, yang disebut context vector (atau "pikiran", *thought vector*).
2. Decoder: Sebuah RNN lain yang diinisialisasi dengan *context vector* dari Encoder sebagai *hidden state* awalnya. Decoder kemudian menghasilkan kalimat output kata demi kata. Pada setiap langkah, output yang dihasilkannya diumpankan kembali

sebagai input untuk langkah berikutnya, hingga ia menghasilkan token khusus <end_of_sequence>.

Keterbatasan:

Arsitektur ini memiliki kelemahan signifikan: seluruh makna dari kalimat input (yang bisa sangat panjang) harus dipadatkan ke dalam satu context vector berukuran tetap. Ini menjadi bottleneck informasi dan membuat model kesulitan menangani kalimat yang panjang.

3. Mekanisme Attention

Mekanisme Attention diperkenalkan untuk mengatasi masalah *bottleneck* pada model Encoder-Decoder. Idenya adalah memberikan Decoder kemampuan untuk "melihat kembali" semua *hidden state* dari Encoder pada setiap langkah saat ia menghasilkan output.

Konsep Inti:

Attention memungkinkan Decoder untuk memfokuskan perhatiannya pada bagian-bagian yang paling relevan dari kalimat input saat menghasilkan setiap kata output. Misalnya, saat menerjemahkan kalimat, untuk menghasilkan kata output tertentu, model mungkin perlu memberikan perhatian lebih pada satu atau dua kata spesifik dari kalimat input.

Cara Kerja (Konseptual):

1. Pada setiap langkah decoding, *hidden state* Decoder saat ini dibandingkan dengan *semua hidden state* Encoder untuk menghitung attention scores. Skor ini mengukur seberapa cocok atau relevan setiap kata input dengan kata output yang akan dihasilkan.
2. Skor ini kemudian dilewatkan melalui fungsi softmax untuk menghasilkan attention weights, yaitu serangkaian bobot yang jumlahnya satu. Bobot yang tinggi menandakan relevansi yang tinggi.
3. Context vector yang dinamis dihitung sebagai jumlah terbobot (*weighted sum*) dari semua *hidden state* Encoder, menggunakan *attention weights* sebagai bobotnya.
4. *Context vector* dinamis ini kemudian digabungkan dengan output dari langkah decoding sebelumnya dan digunakan sebagai input untuk menghasilkan kata output berikutnya.

Dengan mekanisme ini, informasi tidak lagi dipaksa melalui satu *bottleneck*. Decoder dapat mengakses seluruh urutan input pada setiap langkah, yang secara drastis meningkatkan kemampuan model untuk menangani urutan yang panjang.

4. Arsitektur Transformer

Arsitektur Transformer, yang diperkenalkan pada tahun 2017, adalah sebuah terobosan besar. Model ini menunjukkan bahwa kita bisa mendapatkan kinerja canggih tanpa menggunakan RNN sama sekali, dan hanya mengandalkan mekanisme *attention*. Karena tidak bersifat rekuren, Transformer dapat diproses secara jauh lebih paralel, membuatnya sangat efisien untuk dilatih pada dataset besar.

Komponen Kunci Transformer:

- Positional Encodings: Karena tidak ada *loop* rekuren, model tidak memiliki informasi tentang urutan kata. Untuk mengatasinya, sebuah vektor positional encoding ditambahkan ke setiap *word embedding*. Vektor ini memberikan informasi unik tentang posisi absolut atau relatif setiap kata dalam urutan.
- Multi-Head Attention: Alih-alih hanya menghitung *attention* satu kali, Transformer melakukannya beberapa kali secara paralel dalam "kepala-kepala" (*heads*) yang berbeda. Setiap *head* dapat belajar untuk fokus pada aspek atau hubungan yang berbeda dalam kalimat. Hasil dari semua *head* kemudian digabungkan untuk menghasilkan representasi akhir.
- Self-Attention: Ini adalah mekanisme di mana setiap kata dalam sebuah urutan melihat semua kata lain dalam urutan yang sama untuk menghitung representasi dirinya sendiri. Ini memungkinkan model untuk memahami konteks internal kalimat (misalnya, mengerti bahwa kata "it" merujuk pada "the cat").
- Struktur Encoder-Decoder: Seperti model Seq2Seq, Transformer juga memiliki tumpukan *Encoder layer* dan *Decoder layer*.
    - Encoder Layer: Setiap *encoder layer* memiliki dua sub-lapisan utama: sebuah lapisan Multi-Head Self-Attention dan sebuah Position-wise Feed-Forward Network. Terdapat koneksi residual dan normalisasi lapisan di sekitar setiap sub-lapisan.
    - Decoder Layer: Setiap *decoder layer* mirip, tetapi memiliki tiga sub-lapisan: sebuah lapisan Masked Multi-Head Self-Attention (di-masking agar tidak bisa "mengintip" kata-kata di masa depan), sebuah lapisan Multi-Head Encoder-Decoder Attention (di mana ia memperhatikan output dari Encoder), dan sebuah Feed-Forward Network.

Arsitektur Transformer telah menjadi dasar bagi hampir semua model NLP canggih saat ini, termasuk model bahasa besar seperti BERT (yang menggunakan tumpukan Encoder) dan GPT (yang menggunakan tumpukan Decoder).