Fidelya Fredelina
22/496507/TK/54405

Laporan Implementasi Arsitektur Transformer (Decoder-Only)

Transformer adalah model *neural network* yang berfungsi untuk mempelajari pola semantik menggunakan lapisan *attention* tanpa memperhatikan urutan (*reccuren*t). Transformer terdiri dari lapiran *encoder* dan *decoder*. Arsitektur Transformer yang diusulkan di laporan merupakan arstiektur decoder-only. Komponen-komponen utamanya terdiri dari:

Embedding

Embedding merupakan proses pertama untuk memetakan input menjadi data yang dapat dicerna oleh model. Pada tahap ini, terdapat *token embedding* dan *positional embedding.* Input yang diterima oleh model adalah barisan token dari lapisan *encoder*, yang dapat disederhanakan menjadi daftar angka.

Positional encoder dibutuhkan untuk menangkap makna yang berbasis posisi, seperti menentukan kata yang dimaksud oleh kata pengganti. Implementasi untuk laporan ini menggunakan *sinusoidal positional encoder.* Hal ini karena lapisan *attention* bersifat invarian terhadap permutasi, sehingga informasi posisi harus disisipkan. *Positional encoding* berbasis sinusoidal menyediakan cara kontinu dan deterministik untuk mewakili posisi secara general. Cara ini juga digunakan di paper *Attention Is All You Need* pada tahun 2017.

Attention

Komponen ini merupakan komponen inti dalam arsitektur Transformer. *Attention* adalah mekanisme yang merepresentasikan hubungan semantik antar kata. Input dari komponen ini merupakan 3 vektor yaitu Q (*query*), K (*key*), dan V (*value*). Q dan K merupakan representasi vektor dari input yang diberikan. Kedua vektor kemudian dikombinasikan menggunakan operasi perkalian matriks yang akan menghasilkan matriks nilai (*score*). Intuisinya, karena kedua vektor merepresentasikan token yang sama, maka ketika token dikalikan dengan dirinya sendiri atau dengan token yang berdekatan (hasil dari *token embedding* dan *positional encoder*) akan menghasilkan nilai yang tinggi. Matriks nilai akan dibagi dengan akar dari dimensi input untuk menjaga kestabilan matriks. Setelah itu, matriks nilai akan dipetakan dengan fungsi aktivasi softmax sehingga luaran berada di rentang 0 sampai 1. Matriks ini kemudian dikalikan dengan matriks V sehingga menghasilkan nilai *attention* yang tinggi jika kata-kata saling berhubungan dan rendah jika kata-kata tidak berhubungan.

Lapisan *attention* tidak hanya terdiri dari 1 lapisan, tetapi proses ini dilakukan terhadap sub-vektor Q,K, dan V secara berulang. Proses ini dinamakan sebagai *multi head attention*, setiap “head” merujuk pada jendela operasi mekanisme *attention*. Matriks awal akan dipecah menjadi sub-vektor dengan dimensi yang ditentukan oleh parameter head. Kemudian, ketika operasi *attention* sudah berhasil dilakukan untuk semua head, hasil akan dikombinasikan (*concatenation*) menjadi 1 matriks hasil. Proses ini untuk menjamin model dapat menangkap semua arti semantik dari token input.

Namun, untuk layer *decoder* terdapat 1 tahapan terakhir yang membedakannya dengan layer *encoder*, yaitu penggunaan *causal masking*. Tahap ini sederhananya merupakan perkalian

dengan matriks segitiga bawah yang memastikan bahwa nilai *attention* untuk *decoder* hanya berlaku untuk token-token sebelumnya, bukan untuk token setelahnya (token masa depan). Luaran dari tahap ini merupakan nilai *attention* yang dapat digunakan oleh layer *decoder* untuk melakukan prediksi token berikutnya.

Feed-Forward Network

Komponen ini merupakan komponen paling sederhana dalam arsitektur Transfomer karena bentuknya mirip dengan arsitektur *neural network* pada umumnya. Di dalam komponen ini terdapat 2 lapisan normalisasi + linear yang dipisahkan oleh lapisan aktivasi (menggunakan fungsi ReLU).

Decoder Block

Semua komponen di atas akan digabungkan di komponen terakhir ini. Pertama, lakukan operasi *embedding* menggunakan komponen *embedding*. Kemudian, luaran dari komponen *embedding* dijadikan input untuk komponen *attention*. Setiap proses tersebut akan dijalankan menggunakan *feed-forward neural network*. Pada akhirnya, output dari *decoder block* adalah logits (output sebelum normalisasi), distribusi probabilitas token berikutnya, *attention mask* (untuk membuktikan bahwa *causal masking* telah berhasil dilakukan).

Hasil

Logits	Prob. Distribution	Attn. Mask
[[-0.20382276 -1.07228022 0.45859058 ... -1.53566832 1.1386206 -0.43753761] [-0.1968479 -1.11111099 0.56103369 ... -1.52786765 1.13202422 -0.42666431] [-0.12637 -1.14452886 0.49920907 ... -1.5481254 1.07426507 -0.49111354] [-0.13759375 -1.07870462 0.53987516 ... -1.54506618 1.22821305 -0.56491684]]	[[0.00052059 0.00021844 0.00100966 ... 0.00013743 0.00199301 0.00041209] [0.00052479 0.00021034 0.00111977 ... 0.00013865 0.00198201 0.00041704] [0.00056299 0.00020338 0.0010524 ... 0.00013584 0.00187036 0.00039092] [0.00055703 0.00021735 0.00109673 ... 0.00013634 0.00218294 0.00036332]]	[1. 0. 0. ... 0. 0. 0.] [0.55963568 0.44036432 0. ... 0. 0. 0.] [0.40191512 0.32076231 0.27732256 ... 0. 0. 0.] ... [0.03330384 0.0308654 0.02735823 ... 0.0794265 0. 0.] [0.03088882 0.02913827 0.02650244 ... 0.07895001 0.0599843 0.] [0.02757499 0.02622227 0.02416445 ... 0.07926138 0.06188486 0.06590904]]