Richard Rivaldo
13519185 – K04

Poem Generator

Bagaimana approach kalian dalam membuat puisi?

Proses pembuatan *Poem Creator* bertumpu pada model *Deep Learning* yang dibuat dengan menggunakan bantuan *Tensorflow Keras* sebagai pustaka utamanya. Selain itu, juga digunakan beberapa pustaka lain seperti *Regex built-in* Python dan *Numpy* dalam pemrosesan matriks. Pada projek ini, juga dicoba untuk menggunakan dan melatih model *words embedding* sendiri dengan menggunakan *dataset* yang dipakai dan *Global Vector Embeddings* (GloVe).

Pertama, akan dibuka dan dibaca terlebih dahulu data .txt yang digunakan di dalam program dan disimpan ke dalam sebuah *string* besar. *String* ini kemudian akan dipisahkan berdasarkan barisnya sehingga akan dihasilkan kumpulan larik dalam puisi dari data yang dibaca. Setiap larik ini kemudian akan dibersihkan dari semua tanda baca yang ada, terkecuali tanda `'` dan `-` yang bisa memberikan suatu makna lebih untuk suatu kata tertentu.

Larik yang telah dibersihkan dan hanya mengandung kata-kata teratur kemudian akan ditokenisasi secara manual. Hal ini dilakukan dengan menggunakan bantuan *Regex* dan pemisahan dilakukan dengan separator berupa spasi atau `--` karena pada puisi yang dibaca terdapat beberapa pasang kata yang dipisahkan dengan separator tersebut. Selain itu, kata-kata yang bersifat *uppercase* juga akan diubah menjadi *sentence case* dengan kapitalisasi biasa. Proses ini akan menghasilkan kumpulan token kata dari puisi tersebut.

Selanjutnya, akan dibangun korpus kata yang dibuat dengan menggunakan bantuan pustaka GloVe untuk Python dari token tersebut. Korpus yang dibangun akan mengkalkulasi matriks *co-occurrence* dengan *default window* yang bernilai 20. Nilai *window* merupakan seberapa lebar perhitungan akan dilakukan untuk suatu kata pada indeks tertentu. Nilai *default* di atas dipilih karena secara observasi dapat dikatakan bahwa setiap larik memiliki jumlah kata yang rata-rata kurang dari 15. Dengan bantuan API *Tokenizer* pada pustaka Tensorflow, juga akan dibuat objek *tokenizer* dari token tersebut.

Pembuatan objek *tokenizer* ditujukan untuk memudahkan konversi dari kata menuju sekuens *integer* ataupun sebaliknya. Token tidak akan dijadikan *lowercase* dan pada kamus dari *tokenizer* juga ada token khusus untuk *out-of-vocabulary* token. Dengan demikian, nantinya *vocabulary* yang ada pada *tokenizer* berisikan semua token kata yang ada, ditambah dengan token OOV pada indeks pertama.

Matriks yang dihasilkan pada korpus digunakan dalam proses *fitting* untuk model GloVe yang dibuat dengan beberapa parameter tertentu, misalnya *learning rate* dan jumlah komponen. Jumlah komponen menyatakan ukuran vektor penyusun *embeddings matrix* yang nantinya dihasilkan untuk semua token yang ada sebelumnya. Untuk menghasilkan *embeddings matrix*, hanya perlu dicari indeks setiap token pada kamus model *embedding*. Setiap vektor untuk kata tertentu bisa diakses pada kumpulan vektor yang ada untuk membangun matriks tersebut. *Embeddings matrix* yang dihasilkan akan memiliki dimensi berupa jumlah token kata dikali dengan jumlah komponen yang ditentukan.

Dari kumpulan token awal yang dikelompokkan berdasarkan lariknya, maka token-token ini disatukan kembali menjadi sebuah larik utuh. Dari larik ini kemudian akan dihasilkan sekuens yang telah disebutkan di atas dengan menggunakan *tokenizer* yang ada. Dari sekuens yang dihasilkan itu pula, juga akan diiterasi sehingga akan menghasilkan beberapa *subset* sekuens yang lebih kecil. *Subset* ini digunakan untuk mensimulasikan adanya prediksi kata yang akan muncul jika diberikan kata-kata sebelumnya. Ukuran minimum dari *subset* ini adalah 2 buah token yang telah dikonversi menjadi sekuens.

Kumpulan sekuens yang berukuran besar ini akan digunakan untuk memproduksi *training data*. Fitur pada data dibangun dari semua elemen sekuens terkecuali elemen terakhir sekuens yang nantinya akan digunakan untuk pembangunan label. Karena ukuran dari *features* bersifat *varying*, maka bisa digunakan *padding sequence* untuk input tersebut. Namun, pada projek ini akan digunakan *ragged tensor* yang konsepnya mirip, namun lebih baik dibandingkan *padding.* Adapun labelnya sendiri akan dijadikan ke dalam bentuk kategorikal dengan menggunakan *one-hot encoding* dengan banyak token sebagai jumlah kelasnya.

Model utama akan dibangun secara sekeunsial. Pada *input layer* model akan dideklarasikan terlebih dahulu bahwa input yang diterima bersifat *ragged*. *Input layer* kemudian akan ditransformasi menjadi *densely-connected layer*. Model dilanjutkan dengan *Embedding layer* dengan dimensi input berupa banyak kata, dan dimensi keluaran adalah dimensi vektor pada *embedding*. Pada *layer* ini juga akan diinisialisasi bobot *embedding* dengan menggunakan matriks yang telah dihasilkan.

Selanjutnya, akan digunakan *Bidirectional Long Short-Term Memory* (LSTM) yang menerima banyak unit tertentu dan mengembalikan sekuens pada saat sudah melewati lapisan ini. Lapisan berikutnya diteruskan oleh LSTM biasa dengan jumlah unit yang sama. Lapisan LSTM ini kemudian dilanjutkan dengan dua buah lapisan *Dense* yang masing-masing memiliki fungsi aktivasi berupa *Leaky ReLU* dan *Softmax* untuk klasifikasi multikelas. Model juga akan menggunakan *Dropout Regularization* yang akan mematikan beberapa persen neuron secara bergantian untuk menghindari *overfitting.*

Model dikompilasi dengan menggunakan Adam sebagai *optimizer* nya dan *loss function* berupa *categorical cross-entropy* untuk permasalahan klasfikasi tersebut. Metriks yang digunakan merupakan metriks sederhana, yaitu akurasi, meskipun lebih cocok digunakan metriks F1 atau *Precision-Recall*. Model kemudian dilatih dengan *default* jumlah *epochs* adalah 100.

Pipeline untuk menghasilkan puisi adalah sebagai berikut. Puisi menerima parameter jumlah larik dan jumlah maksimum kata yang ada pada suatu larik. Dari jumlah maksimum tersebut akan dihitung jumlah minimum kata pada suatu larik yang merupakan pengurangan jumlah maksimum dengan 5. Oleh karena itu, nilai minimum untuk jumlah maksimum tersebut adalah 8 sehingga paling tidak dihasilkan tiga buah kata pada suatu larik.

Akan terdapat sebuah kalimat yang digunakan sebagai *seed* dan kalimat ini akan dibersihkan serta ditokenisasi dengan proses yang sama sebelumnya. Untuk setiap lariknya, akan diacak jumlah kata yang akan dihasilkan pada larik tersebut dengan jumlah minimum dan maksimum yang dihitung sebelumnya. Dengan menggunakan *tokenizer* sebelumnya, token tersebut kemudian akan dikonversi menjadi sekuens, dilanjutkan dengan konversi menjadi *ragged tensor*. Model kemudian akan melakukan prediksi dengan masukan *tensor* tersebut dan indeks yang didapat bisa digunakan untuk mencari kata terkait pada *tokenizer*.

Seed text kemudian akan ditambahkan dengan kata yang telah diprediksi tadi secara terus-menerus sampai proses generasi selesai. Semua kata yang dihasilkan pada suatu iterasi akan disimpan secara teratur pada *list* biasa untuk nantinya ditampilkan secara terurut dengan *formatting* biasa.