BAB II

TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

2.1.1 DeepType: On-Device Deep Learning for Input Personalization Service with Minimal Privacy Concern[6]

Penelitian ini dilakukan oleh Xu, Mengwei; Qian, Feng; Mei, Qiaozhu; Huang Kang; dan Liu Xuanzhe masing-masing dari *Peking University*, *University of Minnesota*, *University of Michigan*, dan *Kika Tech* dengan tujuan untuk mengembangkan suatu sistem yang dapat meningkatkan privasi pada sistem *next-word prediction* dengan melatih model pada *on-device*. Model global akan dilatih dengan dataset yang didapatkan dari text corpora yang tersedia secara publik, seperti dari Wikipedia dan Twitter dengan menggunakan LSTM. Model kemudian diunduh pada perangkat dan digunakan untuk memberikan prediksi dan melakukan pelatihan model. Agar model dapat dilatih secara akurat dan efisien dengan dataset yang terbatas (input pengguna saja), teknik pelatihan inkremental dan vocabulary dinamis akan dilakukan. Pelatihan model akan dilakukan secara offline, dimana model akan dilatih dengan dataset yang terakumulasi pada perangkat, dan online, dimana model akan dilatih selagi pengguna sedang menggunakan sistem *next-word prediction*. Penelitian mendapatkan bahwa sistem Deeptype berhasil meningkatkan efisiensi penulisan huruf untuk pengguna tanpa menggunakan daya yang cukup besar untuk perangkat tersebut.

2.1.2 Predictive Text System for Bahasa with Frequency, n-gram, Probability Table, and Syntatic using Grammar[7]

Penelitian ini dilakukan oleh Suhartono, Derwin; Wong, Garry; Kusuma, Polim; dan Saputra, Silviana dari Universitas Bina Nusantara dengan tujuan untuk membuat sistem *next-word prediction* yang menggunakan gabungan metode frekuensi, n-gram, tabel probabilitas,

dan sintatik menggunakan grammar. Metode Frekuensi digunakan untuk mengurutkan kata-kata berdasarkan berapa banyak kata tersebut diketik oleh pengguna. N-gram digunakan untuk melatih model *next-word prediction*. N-gram yang digunakan merupakan bi-gram. Tabel probabilitas digunakan untuk menyimpan frase yang sudah disediakan dan data yang dilatih. Sintatik menggunakan grammar digunakan untuk memprediksi kata selanjutnya dari input pengguna berdasarkan hubungan sintatik antara kata sebelumnya dan kata selanjutnya. Hasil model kemudian dievaluasi dengan menggunakan formula KSPC yang digunakan untuk mengetahui seberapa banyak penulisan huruf yang dapat dikurangi. Didapatkan bahwa dengan menggunakan metode tersebut, pengguna dapat mengurangi penulisan huruf sebanyak 50% secara rata-rata.

2.1.3 A RNN based Approach for next word prediction in Assamese Phonetic Transcription[9]

Penelitian ini dilakukan oleh Barman, Partha Pratim; dan Boruah, Abhijit dari *Dibrugarh University* dengan tujuan untuk membuat sistem *next-word prediction* pada bahasa Assam dan pada bahasa Assam yang di transkripsikan secara fonetis dengan menggunakan LSTM, dan learning rate sebesar 0,001. Model akan dilatih sebanyak 100.000 epoch, masing masing menggunakan hyperparameter besar layer berbeda, dimana akurasi dan loss dicatat setiap 1.000 epoch. Ditemukan bahwa performa model meningkat sewaktu dimensi layer dinaikkan dari 128 ke 256, tetapi kemudian menurun sewaktu dinaikkan ke 512. Penelitian juga menemukan bahwa meningkatkan learning rate ke 0,003 menurunkan performa model.

2.1.4 Next Words Prediction Using Recurrent Neural Network[10]

Penelitian ini dilakukan oleh Ambulgekar, Sourabh; Malewadikar, Sanket; Garande, Raju; dan Joshi, Bharti dari *Institute of Technology Mumbai* dengan tujuan untuk menggunakan RNN pada *next-word prediction.* Peneliti berharap dapat membuat sistem yang dapat

memprediksi lebih dari 10 kata setelah pengguna telah mengetik 40 huruf dengan menggunakan LSTM. Peneliti memutuskan untuk menunggu sampai 40 huruf agar sistem dapat mendapatkan gambaran besar tentang apa yang ditulis oleh pengguna. Peneliti menemukan bahwa akurasi sistem sekitar 56% setelah program dilatih sepanjang 5 epoch dan dengan menggunakan hidden layer sebanyak 128.

2.1.5 A comparison of LSTM and GRU networks for learning symbolic sequences[2]

Penelitian ini dilakukan oleh Cahuantzi, Roberto; Chen, Xinye; dan Güttel, Stefan dari *University of Manchester* dengan tujuan untuk mencari hubungan di antara hyperparameter RNN dan kompleksitas panjang data yang dapat dihafal dengan menggunakan dua tipe RNN yaitu GRU dan LSTM. Kompleksitas diukur melalui panjang input dan jumlah vocabulary yang digunakan, dimana data dengan kompleksitas tinggi mempunyai panjang input sebesar 5.000 – 10.000 dan jumlah vocabulary sebanyak 10 – 52. Sebaliknya data dengan kompleksitas rendah mempunyai panjang input sebesar 2 – 12 dan jumlah vocabulary sebanyak 2 – 6. Penelitian tersebut mendapatkan bahwa dua hyperparameter terpenting berupa learning rate dan jumlah layer model. Didapatkan bahwa semakin besar dimensi layer, semakin akurat suatu model. Penelitian juga mendapatkan bahwa dalam pemilihan learning rate, learning rate yang digunakan tidak boleh terlalu besar ataupun terlalu kecil. Didapatkan bahwa waktu pelatihan yang diperlukan model untuk akurasi lebih atau sama dengan 0,99 meningkat jika learning rate di-set lebih dari atau kurang dari 0,01. Dalam performa pada GRU dan LSTM, penelitian menemukan bahwa GRU menggunakan lebih sedikit waktu daripada LSTM pada data dengan kompleksitas rendah. Sementara pada data dengan kompleksitas tinggi, LSTM menggunakan lebih sedikit waktu daripada GRU. Akan tetapi, pada sisi akurasi model, LSTM memiliki akurasi yang lebih besar di kedua jenis sistem daripada GRU.

2.1.6 Kesimpulan Penelitian

Dari penelitian-penelitian terdahulu terdapat beberapa hal yang didapatkan dan akan digunakan untuk implementasi oleh penulis.

1. LSTM kerap digunakan untuk pembuatan sistem *next-word prediction* seperti yang dapat dilihat pada pembahasan subbab 2.1.4 dan 2.1.5, dengan tingkat akurasi yang cukup memuaskan.
2. LSTM lebih cocok digunakan untuk sistem yang menggunakan data dengan kompleksitas tinggi daripada GRU.
3. LSTM mempunyai akurasi lebih besar daripada GRU.
4. Jumlah layer dan learning rate merupakan hyperparameter terpenting sewaktu melakukan pembuatan LSTM, seperti yang dapat dilihat pada pembahasan subbab 2.1.3 dan 2.1.5.
5. Beberapa aspek harus dilakukan dalam pembuatan model, seperti melakukan model kompresi, pengurangan besar vocabulary, dan teknik dalam mengatasi inferensi dan pelatihan ulang, untuk mencapai performa yang optimal, seperti yang dapat dilihat pada pembahasan subbab 2.1.1.

2.2 Tinjauan Teori

2.2.1 Next-Word Prediction

Next-word prediction adalah salah satu tugas dasar NLP yang digunakan untuk memprediksikan kara apa yang akan muncul selanjutnya dari input yang diberikan oleh pengguna. Teknologi tersebut dapat diimplementasikan pada penulisan teks atau email untuk membantu sang pengguna. Sebelum sebuah sistem dapat dilatih untuk melakukan tugas tersebut, data yang dipakai untuk melatih tersebut harus diproses terlebih dahulu. Hal ini dilakukan dengan melakukan *cleaning, tokenization,* dan *word embedding.*

Cleaning dilakukan untuk menormalisasi data sehingga kualitas model dapat setinggi mungkin. Hal ini dilakukan dengan menghapus tanda baca, mengubah semua kalimat menjadi huruf kecil, dan lainnya,

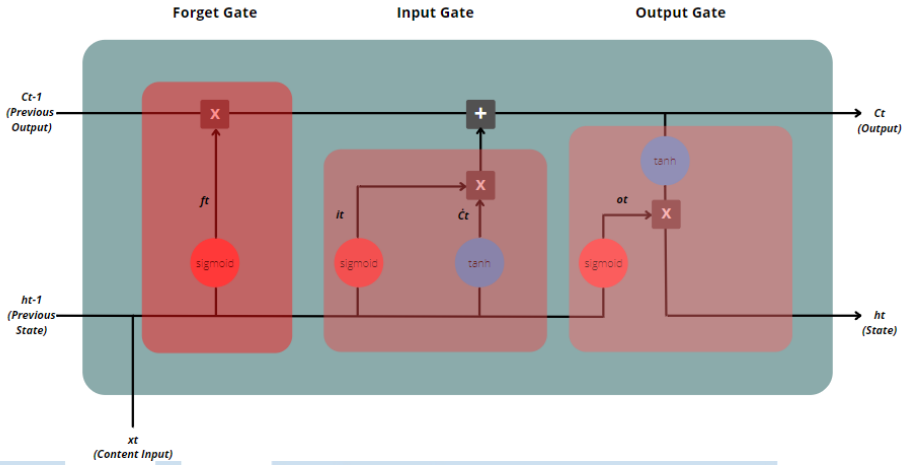
Tokenization adalah proses yang dilakukan untuk memecah kalimat menjadi kata tersendiri ataupun huruf tersendiri, atau juga dikenal sebagai *token*. Hal ini dilakukan sehingga sistem dapat mengerti urutan kata pada suatu kalimat. Setiap *token* kemudian akan diubah menjadi nilai numerik agar dapat dibaca oleh sistem.

Untuk menghemat waktu dan memori, data juga akan dilakukan *word embedding. Word embedding* adalah sebuah proses yang bertujuan untuk memetakan data yang disimpan pada vektor berdimensi tinggi ke dalam ruang berdimensi lebih rendah.

2.2.2 Long Short Term Memory

Long Short Term Memory (LSTM) merupakan salah satu implementasi RNN yang dibuat dengan tujuan untuk menangani masalah yang sering ditemukan pada RNN, yaitu ketergantungan output hanya pada output sebelumnya sebaliknya dari seluruh output sebelumnya yang menyebabkan pemrosesan data, seperti halnya pada NLP tidak efektif [9]. LSTM menerapkan *feedback connection* sehingga LSTM dapat memproses setiap data sebagai satu kesatuan dibandingkan dengan data independen yang hanya bergantung pada data sebelumnya.

Terdapat 3 hal utama yang dipakai dalam LSTM, yaitu *cell state, hidden state,* dan *input. Cell state* digunakan untuk menyimpan informasi dari interval sebelumnya, *Hidden state* digunakan untuk merepresentasikan output (dalam kasus ini prediksi) dari noda LSTM sebelumnya, dan *input* digunakan sebagai input data pada time step tersebut. Agar *cell state* dapat merepresentasikan informasi dari interval sebelumnya secara akurat, LSTM menggunakan konsep yang dikenal sebagai *gate*. Terdapat 3 *gate* yang mengontrol bagaimana suatu data diproses.



Gambar 2.2.1 Struktur LSTM

Yang pertama adalah *forget gate*. *Forget gate* digunakan oleh LSTM untuk mengetahui apakah input sebelumnya relevan atau tidak. (*Neural Network*) NN di dalam *Forget gate* akan memproses *input* (xt) dan *hidden state* sebelumnya (ht-1) dengan menggunakan sigmoid activation untuk mendapatkan bagian *input* mana saja yang relevan dengan *hidden state* sebelumnya. NN akan mengeluarkan nilai di antara 0 sampai 1, dimana 0 berarti input tersebut sama sekali tidak relevan, dan 1 yang berarti input tersebut sangat relevan. Hasil kemudian akan dikali dengan data *cell state* sebelumnya.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Yang kedua adalah *input gate*. *Input gate* terdiri dari NN yang memproses *input* (xt) dan *hidden state* sebelumnya (ht-1) dengan menggunakan sigmoid activation. Hasil dari sigmoid activation kemudian akan dikalikan dengan hasil dari tanh activation yang mengambil nilai *input* (xt) dan *hidden state* sebelumnya (ht-1) sebagai input, yang akan mengeluarkan nilai di antara -1 dan 1. Hal ini dilakukan sebagai *feature extraction* sehingga LSTM dapat memproses data tersebut. Data juga akan dikalikan dengan hasil dari sigmoid agar LSTM dapat mengetahui data mana saja yang relevan. Data kemudian akan

dikalikan dengan hasil perkalian antara *cell state* sebelumnya (Ct-1) dan hasil dari forget gate untuk mendapatkan nilai *cell state* (Ct).

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\dot{C}_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$
$$C_t = (C_{t-1} * f_t) + (i_t * \dot{C}_t)$$

Yang terakhir adalah *output gate. Output gate* menggunakan konsep *encoding* dan *scaling* untuk mendapatkan output. Data pada *cell state (Ct)* akan di *encode* dengan menggunakan tanh, kemudian data pada *cell state* akan dikalikan dengan hasil *scaling*, yang menggunakan Sigmoid Activation, pada gabungan dari *input* (xt) dan *hidden state* sebelumnya (ht-1).

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

2.2.3 On-Device Machine Learning

Salah satu cara inferensi ataupun pelatihan dapat dilakukan adalah secara langsung pada perangkat, daripada mengirimkan data ke server dan melakukan inferensi ataupun melakukan pelatihan di server. Hal ini juga dikenal sebagai *on-device.* Beberapa keuntungan yang didapatkan dengan melakukan inferensi ataupun pelatihan secara *on-device* berupa lebih tingginya privasi[10], lebih cepat penampilan hasil, dan tidak memerlukan koneksi internet. Akan tetapi, terdapat beberapa hal yang harus dipertimbangkan sewaktu menggunakan rancang sistem tersebut. Salah satunya berupa mempertimbangkan cara mengoptimalkan model untuk menghemat sumber daya seperti besar model, memori, pemakanan daya, dan data yang minimal untuk melakukan pelatihan model.