BAB II
TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

Dalam pengenalan aktivitas manusia, beberapa penelitian telah melakukan eksperimen dengan menggunakan *Machine Learning*, baik menggunakan teknik klasifikasi maupun teknik regresi. Penelitian Kwon dan Choi (2018) menerapkan model *Multilayer Perceptron Neural Network* (MLP) untuk mengklasifikasikan aktivitas seseorang berdasarkan lokasi tertentu. Dataset yang digunakan berdasarkan dari sensor *accelerometer smartwatch* yang dikenakan di pergelangan tangan yang dapat memberikan informasi gerakan tubuh. Proses yang dilakukan adalah data sensor dari *smartwatch* dibagi menjadi segmen waktu dan diekstraksi dengan menghitung rata-rata standar deviasi dari setiap data sensor sebagai fitur. Fitur vektor waktu digunakan sebagai *input* untuk diklasifikasikan dengan model MLP. Hasil klasifikasi menunjukkan akurasi yang signifikan yaitu sebesar 95,2%.

Penelitian lain yang meneliti pengenalan aktivitas manusia menggunakan model *Deep Learning*, salah satunya adalah penelitian Almeida dan Azkune (2018). Penelitian tersebut mengusulkan model prediksi menggunakan model *Long Short Term Memory* (LSTM) dan algoritma *Word2Vec* sebagai representasi suatu tindakan ke sebuah vektor. Penelitian tersebut menggunakan dataset berbasis sensor pada kasus pengenalan aktivitas seseorang di sebuah apartemen selama 28 hari dan terdapat tujuh label aktivitas. Proses yang dilakukan adalah modul *input* menerima data sensor dan memetakan suatu indakan ke dalam lapisan *embedding*. Bobot lapisan diinisialisasi dengan menggunakan nilai yang diperoleh dengan menggunakan algoritma *Word2Vec*, kemudian diproses oleh modul pemodelan berurut dengan model LSTM. Hasil eksperimen dalam penelitian tersebut menghasilkan akurasi tertinggi sebesar 85,8%.

Penerapan model *Recurrent Neural Network* juga telah dilakukan untuk pengenalan aktivitas dan deteksi anomali perilaku (Arifoglu dan Bouchachia, 2018). Dalam penelitiannya melakukan evaluasi untuk memprediksi aktivitas dan mendeteksi

aktivitas yang tidak normal dengan menggunakan beberapa varian arsitektur RNN, seperti LSTM, *Vanilla* RNN (VRNN), dan GRU. Penelitian tersebut melatih ketiga arsitektur RNN tersebut dengan menggunakan dataset yang berbeda. Berdasarkan hasil evaluasinya, arsitektur LSTM menghasilkan kinerja yang baik dalam pengenalan aktivitas dengan akurasi tertinggi sebesar 96,7%, sedangkan akurasi arsitektur GRU sebesar 96,1% dan Vanilla sebesar 95,5%.

Keunggulan kinerja model RNN dengan arsitektur LSTM dalam pengenalan aktivitas juga telah dibuktikan dalam penelitian Singh dkk (2017). Dalam penelitian tersebut mengusulkan model *Convolutional Neural Network* (CNN) dan model RNN untuk memodelkan pengenalan aktivitas. Model CNN digunakan karena mampu menangkap (*capture*) beberapa pergerakan aktivitas melalui ekstraksi fitur. Berdasarkan hasil eksperimen, kedua model *Deep Learning* CNN dan RNN menghasilkan kinerja yang baik. Secara keseluruhan eksperimen yang dilakukan, arsitektur LSTM menunjukkan kinerja yang lebih baik dibandingkan CNN, dengan perbandingan akurasi 89,8% (LSTM) dan 88,2% (CNN).

Penelitian mengenai optimasi model *Neural Network* telah dilakukan oleh Wibowo dan Nuqoyati (2019). Dalam penelitian tersebut menerapkan algoritma optimasi dan pemilihan fungsi aktivasi terbaik dalam klasifikasi penyakit kanker menggunakan *microRNA* sebagai fitur. Berdasarkan hasil eksperimennya, penerapan algoritma optimasi dan pemilihan fungsi aktivasi sangat mempengaruhi kinerja model *Neural Network*. Hasil eksperimen menunjukkan bahwa algoritma optimasi Adam dan RMSProp menghasilkan akurasi tertinggi dengan pemilihan fungsi aktivasi ReLU, yaitu sebesar 98,53% untuk optimasi Adam dan sebesar 98,54% untuk optimasi RMSProp.

Optimasi Adam menjadi metode optimasi terbaik saat ini karena kemampuannya yang sangat cepat mencapai konvergensi selama pelatihan (Kingma dan Ba, 2015). Pada kasus tertentu optimasi Adam tidak menghasilkan generalisasi yang optimal dibandingkan optimasi SGD (Wilson dkk., 2017). Penelitian (Keskar dan Socher, 2017) mengusulkan penggabungan kedua metode optimasi tersebut dengan cara melakukan transisi optimasi Adam ke optimasi SGD selama pelatihan. Dalam

penelitiannya tersebut menunjukkan bahwa proses transisi optimasi yang dilakukan terlalu cepat akan menghasilkan generalisasi yang lebih baik dan mendekati hasil optimasi SGD, sedangkan proses transisi yang dilakukan terlalu lambat menghasilkan generalisasi yang sama dengan optimasi Adam.

2.2 Dasar Teori

2.2.1 *Human Activity Recognition*

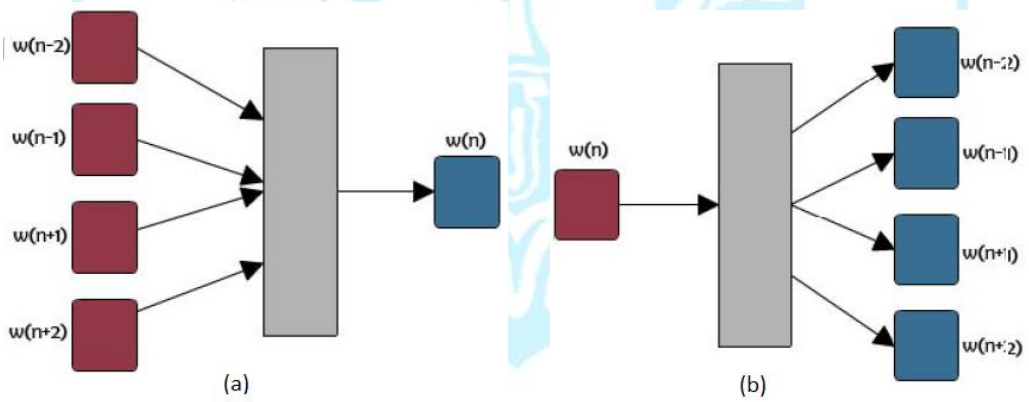
Human Activity Recognition (HAR) atau pengenalan aktivitas manusia menjadi salah satu topik penelitian paling tren pada saat ini. Penelitian di bidang ini bertujuan untuk memantau perilaku seseorang dan menemukan pola aktivitas seseorang yang dilakukan dalam kehidupan sehari-harinya, serta mendeteksi ketidaknormalan perilaku manusia (Arifoglu dan Bouchia, 2018). Ketidaknormalan perilaku manusia dapat diidentifikasi dengan pemodelan aktivitas yang dapat dilakukan dalam pengembangan lingkungan cerdas sebagai berikut (Dhiman dan Vishwakarma, 2019):

1. *Wearable Device*, penggunaan alat yang dikenakan pada tubuh untuk mendeteksi setiap pergerakan tubuh, seperti penggunaan sensor *accelerometer*. *Accelerometer* merupakan sensor yang umum digunakan untuk mengukur perilaku manusia melalui fisik. Penelitian Kwon dan Choi (2018) menggunakan sensor *accelerometer* untuk mengumpulkan data pemantauan aktivitas melalui gerakan tubuh dan mengklasifikasikan aktivitas berdasarkan lokasi kejadian.
2. *Camera Based Monitoring*, merupakan teknik yang memanfaatkan kamera untuk memonitoring aktivitas manusia. Pada umumnya teknik ini digunakan untuk menganalisis perubahan bentuk tubuh dan mendeteksi sesuatu yang menyentuh lantai yang direpresentasikan sebagai gambar 3D RGB *velocity*. Pengenalan aktivitas penangkapan gambar 3D sangat membantu merepresentasikan kecepatan gerakan tubuh, namun kelemahan teknik ini adalah dataset gambar 3D RGB tidak memberikan urutan setiap gerakan dan tidak melakukan pencatatan tindakan.
3. *Binary Sensor Smarthome*, merupakan teknik dengan memanfaatkan sensor yang diletakkan dalam alat di lingkungan rumah dan bertujuan untuk memantau setiap

aktivitas secara berurutan dari waktu ke waktu. Teknik ini umum digunakan untuk pengenalan aktivitas manusia dengan memprediksi perilaku berdasarkan data historis.

2.2.2 Representasi *Word2Vec*

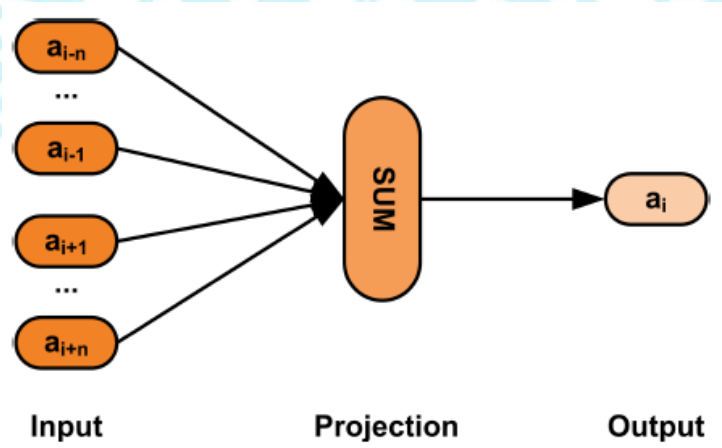
Algoritma *word2vec* diusulkan oleh Mikolov dkk (2013) dengan keunggulan mampu menangkap sintaks dan makna semantik dalam bahasa alami. *Word2Vec* bekerja dengan cara yang mirip dengan *autoencoder*, artinya dapat dilatih menggunakan sejumlah teks yang tidak berlabel (Enriquez dkk., 2016). Namun, *word2vec* tidak dilatih untuk menghasilkan *ouput* yang sama dengan *input*. Terdapat dua jenis model arsitektur *word2vec*, yaitu model *skip-gram* dan *continous bag of words* (Jatnika dkk., 2019). Gambar 2.1 (a) dan (3) menunjukkan perbedaan antara kedua model.



Gambar 2.1 (a) Arsitektur Word2Vec CBOW dan (b) skip-gram
(EnriqueZ dkk., 2016)

Model arsitektur CBOW memprediksi sebuah kata berdasarkan pada konteks kata sekitarnya, sedangkan *skip-gram* menggunakan kata untuk memprediksi kata-kata sekitarnya. Dalam arsitektur CBOW, kata digunakan sebagai *output* dan konteksnya sebagai *input*, sedangkan dalam arsitektur *skip-gram* dilakukan sebaliknya. Kedua model mempelajari hubungan mendasar antara kata-kata dan konteksnya. *Skip-gram* berfungsi lebih baik untuk jumlah teks pelatihan yang lebih sedikit, sementara keunggulan CBOW adalah peningkatan kecepatan pelatihan dan kualitas representasi yang lebih tinggi.

Dalam pemodelan aktivitas manusia, umumnya *input* telah direpresentasikan sebagai ID, *string*, atau vektor *one-hot*. Kekurangan dari representasi tersebut adalah tidak mengandung informasi apapun tentang makna tindakan (Almeida dan Azkune, 2018). Menggunakan vektor *one-hot* saja tidak memungkinkan untuk menghitung seberapa mirip dua tindakan, karena kelemahannya adalah tidak dapat memecahkan kesamaan kata pada *input*, dan informasi ini tidak tersedia untuk model yang akan menggunakan representasi tindakan. Solusi permasalahan tersebut adalah dengan menggunakan *word2vec embedding* untuk merepresentasikan kata-kata. Gambar 2.2 menunjukkan contoh representasi aktivitas menggunakan *embedding*.



Gambar 2.2 Representasi aktivitas sebagai *embedding*
(Almeida dan Azkune, 2018)

Gambar 2.2 merupakan arsitektur CBOW yang digunakan sebagai representasi aktivitas dalam penelitian (Almeida dan Azkune, 2018). Diberikan urutan aktivitas $S_{act} = [a_1, a_2, \dots, a_{la}]$ dimana la adalah panjangnya urutan dan $a_i \in R^{da}$ menunjukkan vektor aktivitas dari i aktivitas dalam urutan, Dengan persamaan $Context(a_i) = [a_{i-n}, \dots, a_{i-1}, a_{1+1}, \dots, a_{i+n}]$ merupakan konteks dari a_i, dimana 2_n adalah panjangnya konteks. Perumusan $p(a_i|Context(a_i))$ menjadi probabilitas ke-i dalam urutan aktivitas a_i.

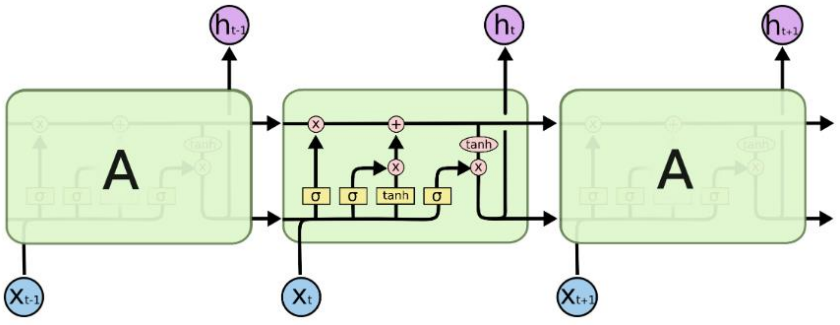
2.2.3 *Long Short Term Memory* (LSTM)

Model *Long Short Term Memory* (LSTM) adalah arsitektur RNN yang dikembangkan untuk menghindari ketergantungan jangka panjang pada RNN (Tax,

2018). Model LSTM memiliki keunggulan menyelesaikan masalah data yang bersifat berurutan (Singh dkk., 2017). Komponen utama lapisan LSTM disebut sel memori. Sel memori terdiri dari empat elemen utama, yaitu *input gate*, *neuron* dengan koneksi berulang, *forget gate*, dan *output gate*. Masukkan yang disediakan untuk LSTM mengontrol operasi yang akan dilakukan oleh *gate* di sel memori (Song dkk., 2019), seperti *write* (*input gate*), *read* (*output gate*), *reset* (*forget gate*).

Arsitektur LSTM memungkinkan untuk menambah atau menghapus informasi dari *cell state*. Hal ini disebut sebagai *gates* yang merupakan pengatur apakah informasi akan diteruskan atau diberhentikan. *Gates* terdiri dari lapisan *sigmoid* dan *pointwise multiplication operation*, yang mana *output* dari lapisan *sigmoid* adalah angka 1 atau 0 yang menunjukkan apakah informasi tersebut akan diteruskan atau diberhentikan. Lebih lanjut bahwa jika bernilai 0 menunjukkan tidak ada informasi yang akan diteruskan, sebaliknya jika bernilai 1 maka informasi akan diteruskan. Dalam arsitektur LSTM terdapat tiga *gate* utama yang mengatur alur informasi, yaitu sebagai berikut (Arifoglu dan Bouchachia, 2018):

a. *Forget gate*, merupakan *gate* yang memutuskan informasi mana yang akan dihapus dari *cell*.
b. *Input gate*, merupakan *gate* yang memutuskan nilai dari *input* untuk di perbarui pada *state* memori.
c. *Output gate*, merupakan *gate* yang memutuskan apa yang akan dihasilkan sesuai dengan *input* dan memori pada *cell*.



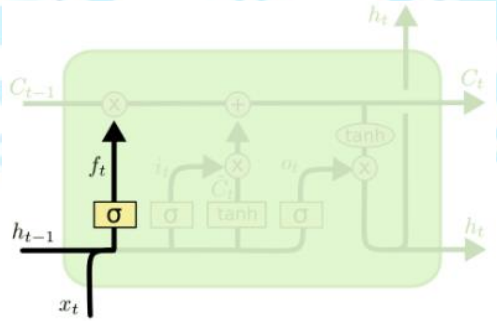
Gambar 2.3 Perulangan empat lapisan pada aristektur LSTM

(Olah, 2015)

Pada umumnya dalam *Neural Network* terdapat bobot yang digunakan sebagai parameter dalam pemrosesannya, begitu juga jenis arsitektur *Recurrent Neural Network*. Namun dalam arsitektur *Recurrent Neural Network* memiliki bobot berulang atau *recurrent weight* (*U*) di setiap jaringannya, hal tersebut dikarenakan dalam pemrosesannya dilakukan secara berulang. Berikut ini merupakan langkah-langkah perulangan pada arsitektur LSTM (Olah, 2015):

1. Memutuskan informasi yang akan dibuang

Langkah pertama adalah memutuskan informasi apa yang akan dibuang dari *state cell*. Keputusan ini dibuat oleh lapisan *sigmoid*, yaitu *forget gate layer*. Pada lapisan ini (*forget get*), akan memproses h_{t-1} dan x_t sebagai *input*, dan menghasilkan *output* berupa angka 0 dan 1 pada *cell state* C_{t-1}. Angka mendekati 1 menunjukkan informasi akan disimpan, sedangkan angka 0 menunjukkan informasi akan dibuang.



Gambar 2.4 Langkah pada *forget gate layer* (Olah, 2015)

Persamaan *forget gate* dinotasikan pada persamaan 2.1, sebagai berikut:

$$f_t = \sigma(W_f . x_t + U_f . h_{t-1} + b_f) \quad \text{.....(2.1)}$$

Keterangan:

f_t = *forget gate*

σ = fungsi *sigmoid*

W_f = nilai bobot untuk *forget gate*

U_f = nilai bobot berulang untuk *forget gate*

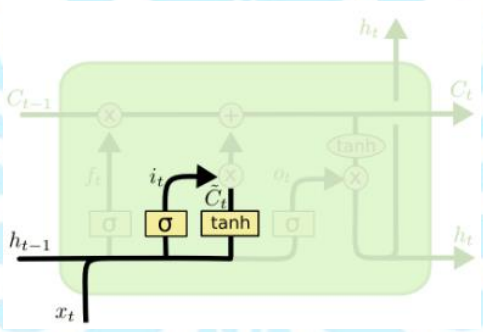
h_{t-1} = nilai *output* sebelum orde ke t

x_t = nilai *input* pada orde ke t

b_f = nilai bias pada *forget gate*

2. Simpan informasi ke dalam *cell state*

Langkah kedua adalah memutuskan informasi apa yang harus disimpan di *cell state* C_t. Terdapat dua bagian dalam tahapan ini, bagian pertama lapisan *sigmoid*, yaitu *input gate layer* memutuskan nilai mana yang akan diperbarui, bagian kedua yaitu *tanh layer* membuat kandidat dengan nilai baru, $\tilde{C}_t$, yang dapat ditambahkan ke *cell state*. Selanjutnya menggabungkan *ouput* dari *input gate layer* dan *tanh layer* untuk memperbarui *cell state*.



Gambar 2.5 Langkah pada *input gate layer* dan *tanh layer* (Olah, 2015)

Persamaan *forget gate* dinotasikan pada persamaan 2.2, sebagai berikut:

$$i_t = \sigma(W_i . x_t + U_i . h_{t-1} + b_i) \quad \text{.....(2.2)}$$

Keterangan:

i_t = *input gate*

σ = fungsi *sigmoid*

W_i = nilai bobot untuk *input gate*

U_i = nilai bobot berulang untuk *input gate*

h_{t-1} = nilai *output* sebelum orde ke t

x_t = nilai *input* pada order ke t

b_i = nilai bias pada *input gate*

Persamaan kandidat baru dinotasikan pada persamaan 2.3 sebagai berikut:

$$\tilde{C}_t = tanh(W_c . x_t + U_c . h_{t-1} + b_c) \quad \text{.....(2.3)}$$

Keterangan:

$\tilde{C}_t$ = nilai baru yang dapat ditambahkan ke *cell state*

$tanh$ = fungsi *tanh*

W_c = nilai bobot untuk *cell state*

U_c = nilai bobot berulang untuk *cell state*

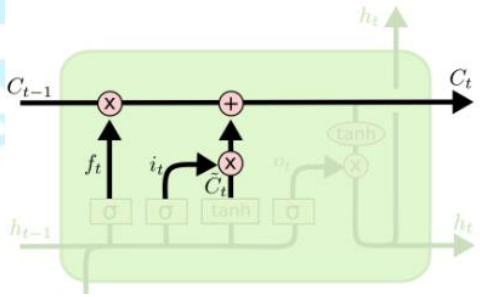
h_{t-1} = nilai *output* sebelum order ke t

x_t = nilai *input* pada orde ke t

b_c = nilai bias untuk *cell state*

3. Memperbarui *cell state*

Langkah ketiga adalah memperbarui *cell state* yang lama, C_{t-1}, menjadi *cell state* baru, C_t, dengan mengkalikan *state* lama dengan f_t. Kemudian ditambahkan dengan nilai kandidat baru, $i_t * \tilde{C}_t$, yang digunakan untuk memperbarui *state*.



Gambar 2.6 Langkah memperbarui *cell state* (Olah, 2015)

Persamaan *cell state* dinotasikan pada persamaan 2.4, sebagai berikut:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad \text{.....(2.4)}$$

Keterangan:

C_t = *cell state*

f_t = *forget state*

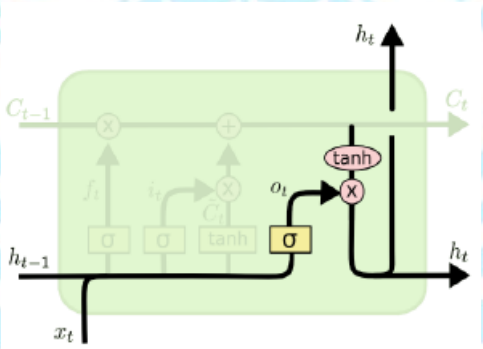
C_{t-1} = *cell state* sebelum orde ke t

i_t = *input gate*

$\tilde{C}_t$ = nilai baru yang dapat ditambahkan ke *cell state*

4. Konfirmasi hasil *output* h_t

Langkah terakhir adalah mengkonfirmasi hasil *output* pada *output gate*. Hasil *output* harus sesuai dengan *cell state* yang telah diproses. Pertama lapisan *sigmoid* menentukan bagian dari *cell state* yang menjadi hasil *output*. Kemudian, *output* dari *cell state* dimasukkan ke dalam lapisan *tanh* (untuk mengganti nilai menjadi antara -1 dan 1) dan dikalikan dengan *sigmoid gate*, agar *output* yang dihasilkan sesuai dengan apa yang ditentukan sebelumnya.



Gambar 2.7 Langkah menentukan *output* (Olah, 2015)

Persamaan *output gate* dinotasikan pada persamaan 2.5, sebagai berikut:

$$o_t = \sigma(W_o.x_t + U_o.h_{t-1} + b_o) \quad \text{.....(2.5)}$$

Keterangan:

o_t = *output gate*

σ = fungsi *sigmoid*

W_o = nilai bobot untuk *output gate*

U_o = nilai bobot berulang untuk *output gate*

h_{t-1} = nilai *output* sebelum orde ke t

x_t = nilai *input* pada orde ke t

b_o = nilai bias pada *output gate*

Sedangkan persamaan nilai *output* orde t diuraikan pada persamaan 2.6, sebagai berikut:

$$h_t = o_t * \tanh(c_t) \quad \text{.....(2.6)}$$

Keterangan:

h_t = nilai *output* orde t

o_t = *output gate*

$tanh$ = fungsi *tanh*

C_t = *cell state*

2.2.4 Pelatihan LSTM

Dalam pemrosesannya model LSTM memiliki proses yang diadaptasi dari *Recurrent Neural Network*. Sama seperti model ANN atau RNN, model LSTM memiliki tiga tahapan dalam pemrosesannya, yaitu *feedforward propagation*, *backpropagation*, dan memperbarui nilai parameter. Dalam model LSTM, proses *backpropagation* dilakukan dengan proses *backpropagation through time* (BPTT), hal tersebut dikarenakan model LSTM melakukan proses perulangan dalam node tersembunyi atau *hidden node*. Berikut Langkah pelatihan menggunakan model LSTM (Song dkk., 2019) :

1. Inisialisasi parameter yang akan digunakan, yaitu itu inisialisasi bobot (W) dan bobot berulang (U) dengan bilangan acak, serta inisialisasi bias b.

a. Fase I : *Feedforward Propagation*

2. Menghitung nilai *forget gate* dengan persamaan 2.1 dan menghitung nilai fungsi aktivasi sigmoid dengan persamaan (2.20).
3. Menghitung nilai *input gate* dengan persamaan 2.2 dan menghitung nilai fungsi aktivasi sigmoid dengan persamaan (2.20).
4. Menghitung nilai kandidat *cell state* dengan persamaan 2.3 dan menghitung nilai fungsi aktivasi tanh dengan persamaan (2.21).
5. Menghitung nilai *cell state* dengan persamaan 2.4.
6. Menghitung nilai *output gate* dengan persamaan 2.5 dan menghitung nilai fungsi aktivasi sigmoid dengan persamaan (2.20).
7. Menghitung nilai keluaran akhir dengan persamaan 2.6.
8. Menghitung nilai *Loss* atau nilai *error* dengan persamaan 2.7.

$$L(y,\hat{y}) = \frac{1}{2}(y,\hat{y})^2 \text{...(2.7)}$$

b. Fase II : *Backpropagation*

9. Menghitung nilai *derivative* dari *error* dengan persamaan 2.8.

$$\frac{\partial L}{\partial \hat{y}} = -(y-\hat{y}) \text{...(2.8)}$$

10. Menghitung nilai *derivative* dari keluaran akhir (*output*) dengan persamaan 2.9.

$$\delta h_t = \Delta_t + \Delta h_t \text{...(2.9)}$$

11. Menghitung nilai *derivative* dari *output gate* dengan persamaan 2.10.

$$\delta o_t = \delta h_t . tanh(C_t). o_t(1-o_t) \text{..(2.10)}$$

12. Menghitung nilai *derivative* dari *cell state* dengan persamaan 2.11.

$$\delta C_t = \delta h_t . o_t . [1 - tanh^2(C_t)] + \delta C_{t+1} . f_{t+1} \text{.......................(2.11)}$$

13. Menghitung nilai *derivative* dari kandidat *cell state* dengan persamaan 2.12.

$$\delta \tilde{C}_t = \delta C_t . i_t . \left[1 - tanh^2(\tilde{C}_t)\right] \text{..(2.12)}$$

14. Menghitung nilai *derivative* dari *input gate* dengan persamaan 2.13.

$$\delta i_t = \delta C_t . \delta \tilde{C}_t . i_t . [1 - i_t] \text{..(2.13)}$$

15. Menghitung nilai *derivative* dari *forget gate* dengan persamaan 2.14.

$$\delta f_t = \delta C_t . \delta C_{t-1} . f_t . [1 - f_t] \text{..(2.14)}$$

16. Menghitung nilai δx_t dengan persamaan 2.15.

$$\delta x_t = W_{\tilde{C}}^T . \delta \bar{C}_t + W_i^T . \delta \bar{\iota}_t + W_f^T . \delta \bar{f}_t + W_o^T . \delta \bar{o}_t$$

$$\delta x_t = W^T . \delta gates_t \text{...(2.15)}$$

17. Menghitung nilai *gradient* bobot (W) dari setiap *gates* atau gerbang LSTM dengan persamaan 2.16.

$$\delta W_{gates} = \sum_{t=0}^{T} (\delta gates_t \, . \, x_t) \quad \text{(2.16)}$$

18. Menghitung nilai *gradient* bobot berulang (U) dari setiap *gates* atau gerbang LSTM dengan persamaan 2.17.

$$\delta U_{gates} = \sum_{t=0}^{T-1} (\delta gates_{t+1} \, . \, h_t) \quad \text{(2.17)}$$

19. Menghitung nilai *gradient* bias (b) dari setiap *gates* atau gerbang LSTM dengan persamaan 2.18.

$$\delta U_{gates} = \sum_{t=0}^{T} (\delta gates_{t+1} \, . \, h_t) \quad \text{(2.18)}$$

c. Fase III : Pembaruan Bobot

20. Menghitung nilai perubahan bobot dengan persamaan 2.19.

$$\delta W_{t+1} = W_t - \alpha \frac{\partial L}{\partial W_t} \quad \text{(2.19)}$$

2.2.5 Fungsi Aktivasi

Fungsi aktivasi adalah fungsi yang digunakan dalam jaringan saraf untuk menghitung jumlah *input* dan *bias* yang terbobot (Nwankpa dkk., 2018). Hal ini memanipulasi data yang disajikan melalui beberapa pemrosesan gradien, biasanya *Gradient Descent* dan kemudian menghasilkan *output* untuk jaringan saraf, yang berisi parameter dalam data. Fungsi aktivasi dapat berupa linier atau non-linier tergantung pada fungsi yang merepresentasikannya, dan digunakan untuk mengontrol *output* dari jaringan saraf luar.

2.2.5.1 *Sigmoid Function*

Sigmoid merupakan fungsi aktivasi non linier yang digunakan dalam jaringan saraf. Fungsi *Sigmoid* mengubah *range* nilai *input x* menjadi nilai antara 0 dan 1 (Balaji dan Baskaran, 2013). Fungsi *Sigmoid* ditunjukkan oleh persamaan 2.20. $S(x)$ akan menghasilkan sebuah kurva dalam rentang 0-1 pada sumbu *y*. Jika *x* adalah bilangan

positif sangat besar, maka nilai e^{-x} memiliki nilai mendekati 0 yang menghasilkan *output* mendekati 1. Sedangkan jika *x* adalah bilangan negatif sangat besar, maka nilai e^{-x} memiliki nilai yang besar dan menghasilkan *output* mendekati 0.

$$f(x) = \frac{1}{(1+e^{-x})} \quad \text{.....(2.20)}$$

Keterangan :

$f_{sigmoid}(x)$ = fungsi *sigmoid* dari x

e = *epsilon*

2.2.5.2 *Hyperbolic Tangent Function* (Tanh)

Fungsi aktivasi *tanh* merupakan salah satu fungsi aktivasi yang digunakan untuk kasus multi klasifikasi. Kelemahan fungsi aktivasi *tanh* adalah tidak menyelesaikan *vanishing gradient* yang umum terjadi pada *sigmoid* (Nwankpa dkk., 2018). Fungsi aktivasi *tanh* ditunjukkan oleh persamaan 2.21. Jika x merupakan bilangan negatif sangat besar, maka nilai *tanh* akan mendekati -1, ketika nilai x merupakan bilangan positif sangat besar, maka nilai *tanh* mendekati 1.

$$f(x) = \left(\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}\right) \quad \text{.....(2.21)}$$

Keterangan:

$f_{tanh}(x)$ = fungsi *tanh* dari x

e = *epsilon*

2.2.5.3 *Rectified Linear Unit* (ReLU)

Fungsi ReLU merupakan salah fungsi aktivasi yang telah terbukti menjadi fungsi yang sering digunakan dalam *Deep Learning*. Hal ini dikarenakan fungsi ReLU menawarkan kinerja dan generalisasi yang lebih baik dalam *Deep Learning* dibandingkan dengan fungsi aktivasi *sigmoid* dan *tanh* (Nwankpa dkk., 2018). Aktivasi ReLU akan memberikan keluaran 0 ketika $x < 0$ dan merepresentasikan fungsi linier ketika $x \geq 0$. Fungsi aktivasi ReLU melakukan operasi ambang batas untuk setiap

elemen *input* di mana nilai kurang dari nol diatur ke nol sehingga ReLU dinotasikan sebagai persamaan berikut:

$$f(x) = \max(0, x) = \begin{cases} x_i, if\ x_i \geq 0 \\ 0, if\ x_i < 0 \end{cases} \text{.....(2.22)}$$

Range dari fungsi aktivasi ReLU adalah [0, ∞], semua *input* bernilai negatif pada fungsi aktivasi ReLU akan menjadi nilai 0. Fungsi ini memperbaiki nilai *input* yang kurang dari nol sehingga diubah menjadi nol dan menghilangkan masalah gradien (*vanishing gradient*) yang diamati pada jenis aktivasi sebelumnya.

2.2.5.4 *Softmax Function*

Fungi aktivasi *Softmax* adalah jenis dari fungsi aktivasi yang digunakan dalam komputasi saraf. Ini digunakan untuk menghitung distribusi probabilitas dan vektor bilangan *real*. Fungsi *Softmax* biasanya digunakan sebagai *output* dari model klasifikasi untuk merepresentasikan distribusi kemungkinan terhadap sejumlah kelas (Nwankpa dkk., 2018). Fungsi *Softmax* menghasilkan *output* yang kisaran nilai antara 0 dan 1, dengan jumlah probabilitas sama dengan 1. Fungsi *softmax* dihitung menggunakan persamaan berikut:

$$f(x_i) = \frac{exp\ (x_i)}{\sum_j exp\ (x_j)} \text{.....(2.23)}$$

Keterangan :

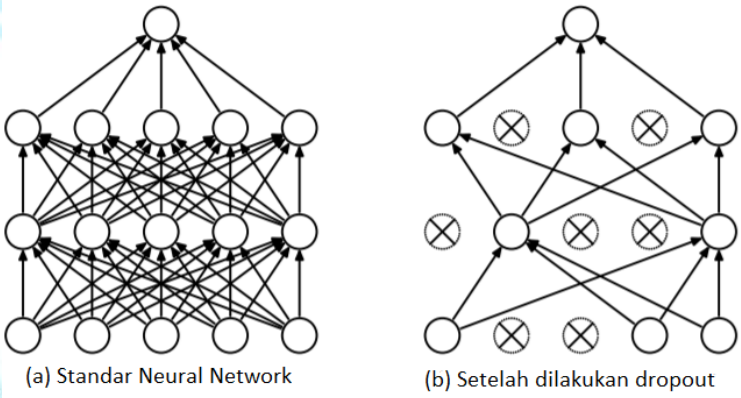
x = nilai *input* lapisan sebelumnya

i, j = indeks unit dan lapisan

Fungsi *softmax* digunakan dalam model multi kelas di mana fungsi ini mengembalikan probabilitas setiap kelas. Fungsi *softmax* sebagian besar muncul di hampir semua lapisan *output* dari arsitektur *Deep Learning*. Perbedaan utama antara fungsi aktivasi *Sigmoid* dan *Softmax* adalah *Sigmoid* digunakan dalam klasifikasi biner sedangkan *Softmax* digunakan untuk tugas klasifikasi multivarian (Nwankpa dkk., 2018).

2.2.6 *Dropout*

Dropout merupakan teknik regularisasi model *Neural Network* untuk mengurangi *overfitting* pada dataset. *Dropout* adalah teknik di mana *neuron* yang dipilih secara acak diabaikan selama pelatihan (*dropout* secara acak). Ini berarti bahwa kontribusinya terhadap aktivasi *neuron* untuk sementara dihapus pada *forward pass* dan setiap pembaruan bobot tidak diterapkan pada *neuron* pada *backward pass*. Istilah *dropout* merujuk pada menggugurkan unit (yang tersembunyi ataupun yang terlihat) dalam sebuah jaringan syaraf tiruan. Menggugurkan unit merupakan upaya menghilangkan sementara unit tersebut dari jaringan syaraf tiruan dengan semua koneksi yang masuk dan keluar menuju unit tersebut (Srivastava dkk., 2014). Seperti yang ditunjukkan pada gambar 2.8.



Gambar 2.8 (a) Standar Neural Network dengan 2 hidden layer, (b) contoh setelah dilakukan dropout pada network (Srivastava dkk., 2014)

Dalam kasus yang sederhana, setiap unit dipertahankan dengan probabilitas tetap p, tidak tergantung pada unit lain, di mana p dapat dipilih menggunakan set validasi atau hanya dapat ditetapkan pada 0,5, yang mendekati optimal untuk berbagai jaringan. Namun, untuk unit *input*, probabilitas retensi optimal biasanya lebih dekat ke 1 dari pada ke 0,5 (Srivastava dkk., 2014).

2.2.7 Optimasi Adam

Optimasi Adam merupakan metode optimisasi berbasis stokastik orde pertama dari fungsi objektif stokastik untuk memperbarui bobot jaringan yang berulang berdasarkan data pelatihan (Chang dkk., 2019). Optimasi Adam bertujuan untuk

menghitung *adaptive learning rate* untuk setiap parameter. *Learning rate* menentukan seberapa banyak perubahan bobot pada jaringan. Berikut ini merupakan persamaan untuk optimasi Adam (Wibowo dkk., 2018):

1. Menghitung gradien g_t pada waktu *t*

$$g_t = \nabla_\theta f_t(\theta_{t-1}) \quad \text{(2.24)}$$

2. Memperbarui bias momen vektor pertama

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \quad \text{(2.25)}$$

3. Memperbarui bias momen vektor kedua

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \quad \text{(2.26)}$$

4. Menghitung bias momen vektor pertama

$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t} \quad \text{(2.27)}$$

5. Menghitung momen vektor kedua

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \quad \text{(2.28)}$$

6. Memperbarui parameter dengan menggunakan *learning rate*

$$\theta_{t+1} = \theta_t - \frac{n}{\sqrt{\hat{v}_t + \varepsilon}} \widehat{m}_t \quad \text{(2.29)}$$

2.2.8 Optimasi SGD (*Stochastic Gradient Descent*)

Metode *Stochastic Gradient Descent* (SGD) bertujuan untuk meminimalkan risiko empiris pada suatu model dengan cara menghitung berulang-ulang fungsi turunan pada contoh pelatihan tunggal dan memperbarui parameter model yang sesuai (Hardt dkk., 2015). Fungsi SGD ditunjukkan oleh persamaan 2.30.

$$x' = x - \alpha(f(x)) \quad \text{(2.30)}$$

2.2.9 *Cross Entropy Loss Function*

Loss function merupakan metode untuk mengevaluasi seberapa baik algoritma memodelkan suatu data (Li dkk., 2019). *Loss function* memiliki kurva yang bertujuan untuk memberi tahu cara mengubah parameter untuk membuat model lebih akurat.

Cara kerja *loss function* adalah membandingkan hasil prediksi dari *output layer* dengan target. *Loss function* yang populer saat ini adalah *cross entropy.* Metode *cross entropy* mengukur kinerja model klasifikasi yang *output* nya merupakan nilai probabilitas antara 0 dan 1. *Cross entropy* dinotasikan dengan persamaan 2.31.

$$H(p,q) = -\sum_x p(x) \log\,(q(x)) \quad \text{.....(2.31)}$$

2.2.10 Akurasi *Top-N*

Pada akurasi *Top-N*, pengukuran kinerja model diukur berdasarkan dari probabilitas tertinggi *N* yang dihasilkan dari lapisan Softmax. Akurasi *Top-N* dapat digunakan dalam permasalahan klasifikasi yang memiliki banyak kelas (*multi class*). Persamaan akurasi *Top-N* didefinisikan pada persamaan 2.32.

$$Top\_N = \frac{1}{n}\sum_{i=1}^{n} 1[y_i \in C_i^N] \quad \text{.....(2.32)}$$

dimana y_i adalah label aktual dan C_i^k adalah hasil keluaran dari lapisan Softmax dengan probabilitas tertinggi *N*, *n* merepresentasikan jumlah data, $1[.] \rightarrow \{0,1\}$ merepresentasikan penilaian, jika hasil keluaran probabilitas tertinggi dari lapisan Softmax sesuai dengan label aktual, maka diberi nilai 1, dan jika tidak sesuai maka diberi nilai 0.