BAB II
LANDASAN TEORI

Bab ini menjelaskan analisis berbagai teori dan hasil penelitian yang relevan dengan masalah yang akan diteliti. Dalam bagian ini melakukan sintesis terhadap teori yang relevan agar diperoleh legitimasi konseptual terhadap variabel yang akan diteliti.

2.1 *Hoax*

Hoax mengacu literatur jurnalistik berdasarkan hukum, terdapat istilah berita *hoax* yang disebut dengan libel yaitu berita bohong yang berisikan tentang penghinaan, penistaan, pencemaran nama baik, hasutan, dan lain sebagainya yang merugikan orang lain yang dituangkan dalam tulisan dan slander yaitu secara lisan (Silitonga, 2019). Tujuan *hoax* adalah upaya untuk menipu pembaca untuk mempercayai sesuatu, padahal pembuat berita palsu itu sendiri tahu bahwa berita tersebut adalah palsu (Rahadi, 2017). Menurut Alessandro Bondielli istilah *hoax* biasanya disebut sebagai "virus pikiran", hal ini dikarenakan kemampuannya untuk mereplikasi diri, mengadaptasi, memutasi dan bertahan di dalam pikiran manusia (Alessandro Bondielli a, 2019).

Penyebaran informasi *hoax* bertujuan sebagai bahan lelucon, iseng, dan biasanya untuk menjatuhkan pesaing (*black campaign*). Dampak yang dihasilkan oleh *hoax* merupakan dampak yang tidak bisa disadari secara langsung, karena akan menyerang pemikiran pembacanya dan jika tidak berhati-hati akan mempengaruhi cara berpikir pembacanya (Rahadi, 2017). Berita *hoax* adalah sebuah publikasi yang terlihat seperti berita faktual, namun ternyata berisi kebohongan dan fitnah. Biasanya berita *Hoax* sengaja dibuat untuk menyebarkan propaganda atau pesan kebencian atas seseorang atau instansi tertentu (Aditiawarman, 2019).

Contoh pemberitaan palsu yang paling umum adalah mengklaim sesuatu barang atau kejadian dengan suatu sebutan yang berbeda dengan barang/kejadian sejatinya. Menurut Dedi Rianto *hoax* sendiri memiliki jenis tersendiri seperti ini (Rahadi, 2017) :

1. *Fake News*

Fake News atau berita bohong adalah salah satu jenis *hoax*. Berita bohong ini bertujuan untuk memalsukan kebenaran dalam suatu berita. Penulis berita biasanya berita bohong biasanya menambahkan hal-hal yang tidak benar.

2. *Clickbait*

Clickbait adalah jenis *hoax* yang merupakan suatu tautan berupa jebakan. Biasanya tautan tersebut diletakan secara strategis dalam suatu situs agar menarik orang untuk masuk kedalam situs tautan tersebut. Tautan berupa fakta namun untuk judul biasanya dilebih-lebihkan.

3. *Satire*

Satire adalah sebuah tulisan atau berita yang menggunakan humor, ironi, dan hal tersebut dibesar-besarkan guna untuk mengomentari suatu kejadian yang sedang hangat dibicarakan.

2.2 *Natual Language Processing*

Bahasa dibedakan menjadi dua, yaitu bahasa alami dan bahasa buatan. Bahasa alami adalah bentuk representasi dari suatu pesan berupa ucapan *(spoken language).* Bahasa juga dapat dinyatakan dalam bentuk tulisan yang biasa digunakan untuk berkomunikasi antar manusia, seperti Bahasa Inggris, Bahasa Indonesia, Bahasa Arab, dan lain-lain. Sedangkan Bahasa buatan adalah bahasa yang dibuat khusus untuk tujuan tertentu, seperti bahasa pemodelan dan bahasa pemograman komputer (Arman, 2004).

Natural Language Processing atau biasa disingkat NLP merupakan cabang dari ilmu komputer yang berkembang dari studi Bahasa dan komputasi linguistik dalam cabang kecerdasan buatan (James Pustejovsky, 2012). Dalam proses *natural language processing* terdapat beberapa kesulitan diantaranya sering terjadi ambiguitas atau makna ganda dan jumlah kosa kata dalam bahasa alami yang semakin besar dan berkembang dari waktu ke waktu (Priansa, 2017).

Tujuan dari NLP adalah untuk mengembangkan dan merancang aplikasi yang mampu menjadi penengah antar interaksi manusia dan mesin dengan perangkat lain melalui penggunaan Bahasa alami. NLP memodelkan pengetahuan terhadap Bahasa, dari segi kata, dan bagaimana kata-kata menjadi suatu kalimat dan konteks

kata dalam kalimat. Beberapa bidang pengetahuan pada *Natural Language Processing* (Victor Amrizal, 2013) :

1. *Fonetik dan fonologi :* berhubungan dengan suara yang menghasilkan kata yang dapat dikenali. Bidang ini menjadi penting dalam proses aplikasi yang memakai metode *speech based system.*

2. *Morfologi :* yaitu pengetahuan tentang kata dan bentuknya dimanfaatkan untuk membedakan satu kata dengan lainnya. Pada tingkat ini juga dapat dipisahkan antara kata dan elemen lain seperti tanda baca.

3. *Sintaksis :* yaitu pemahaman tentang urutan kata dalam pembentukan kalimat dan hubungan antar kata tersebut dalam proses perubahan bentuk dari kalimat menjadi bentuk yang sistematis. Meliputi proses pengaturan tata letak suatu kata dalam kalimat akan membentuk kalimat yang dapat dikenali. Selain itu dapat pula dikenali bagian -bagian kalimat dalam suatu kalimat yang lebih besar.

4. *Semantik :* yaitu pemetaan bentuk struktur sintaksis dengan memanfaatkan tiap kata ke dalam bentuk yang lebih mendasar dan tidak tergantung struktur kalimat . Semantik mempelajari arti suatu kata dan bagaimana dari arti kata-arti kata tersebut membentuk suatu arti dari kalimat yang utuh. Dalam tingkatan ini belum tercakup konteks dari kalimat tersebut.

5. *Pragmatik :* pengetahuan pada tingkatan ini berkaitan dengan masing-masing konteks yang berbeda tergantung pada situasi dan tujuan pembuatan sistem.

6. *Discourse Knowledge* : melakukan pengenalan apakah suatu kalimat yang sudah dibaca dan dikenali sebelumnya akan mempengaruhi arti dari kalimat selanjutnya. Informasi ini penting diketahui untuk melakukan pengolahan arti terhadap kata ganti orang dan untuk mengartikan aspek sementara dari informasi.

7. *World Knowledge* : mencakup arti sebuah kata secara umum dan apakah ada arti khusus bagi suatu kata dalam suatu percakapan dengan konteks tertentu.

2.3 *Deep Learning*

Deep Learning merupakan suatu bidang yang ada pada *Machine Learning* yang diperkenalkan oleh Dechter pada tahun 1986 (Geoffrey E. Hinton, 2006),. Deep

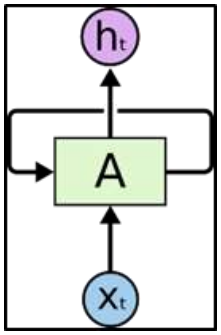
learning terbagi menjadi tiga kategori pendekatan yaitu *supervised learning*, *unsupervised learning*, dan *reinforcement learning*. Salah satunya potensi dari deep learning adalah mengganti fitur buatan tangan dengan algoritma yang efisien untuk pembelajaran *hierarkis unsupervised* atau *semi-supervised feature learning* dan *hierarchical feature extraction* (Schmidhuber, 2015).

Deep Learning juga telah menunjukan kontribusi luas dalam berbagai bidang penelitian dengan membantu kelemahan metode sebelumnya yang membuat sistem kurang kompleks. Deep Learning juga telah digunakan pada bidang Natural Language Processing dengan hasil yang memuaskan (Nene, 2017).

Deep learning (Geoffrey E. Hinton, 2006) adalah varian dari jaringan syaraf tiruan yang disebut *deep belief nets.* Ide jaringan saraf tiruan ini adalah dengan men-train dua layer kemudian menambahkan satu layer diatasnya, kemudian train hanya teratas dan begitu seterusnya. Dengan ini dapat men-train model jaringan saraf tiruan dengan layer lebih banyak dari model-model sebelumnya. Beberapa arsitektur *deep learning* antara lain *Deep Feedforward Network, Recurrent Neural Network, Convolutional Neural Network* yang merupakan pengembangan dari jaringan syaraf tiruan untuk memberikan ketepatan tugas seperti deteksi objek, pengenalan suara, terjemahan bahasa dan lain – lain. Penggunaan metode jaringan syaraf tiruan biasa digunakan dalam bidang deep learning (MathWorks, 2018). Hal yang membedakan antara machine learning dan deep learning adalah penggunaan layer yang jumlahnya dapat mencapai ratusan sehingga dapat disebut *deep*.

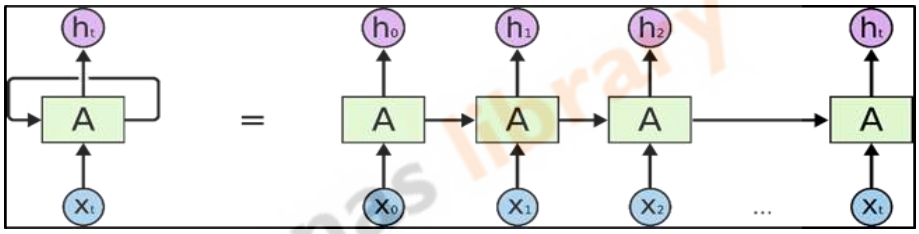
2.4 *Recurrent Neural Netowrk*

Recurrent Neural Network atau biasa disingkat *RNN* adalah jenis jaringan syaraf tiruan untuk memproses data sekeunsial seperti pengenalan ucapan, pemodelan Bahasa dan lain-lain (Maaz Amajd, 2017). *RNN* bekerja dimana pemrosesannya dilakukan secara berulang seperti pada gambar 2.1.



Gambar 2.1 Proses Perulangan *RNN* (Olah, Understanding LSTM Networks, 2015)

Pada gambar 2.1 (x_t) adalah sebagai input, (h_t) sebagai output dan terdapat alur perulangan dimana memungkinkan informasi dilewatkan dari satu langkah jaringan ke langkah berikutnya (Britz, 2015). *RNN* juga dianggap sebagai banyak salinan yang sama, masing-masing menyampaikan pesan kepada penerus seperti pada gambar 2.2.

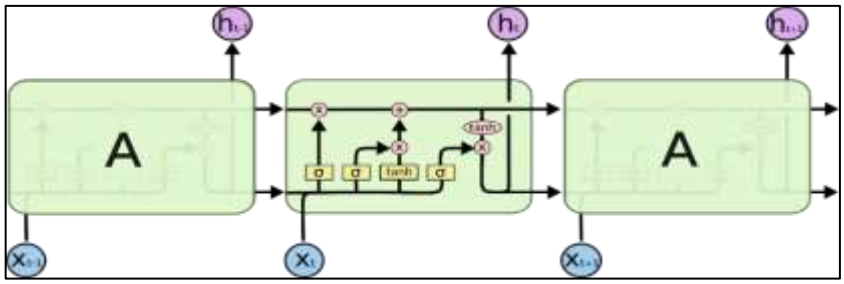


Gambar 2.2 Salinan Jaringan Pada *RNN* (Olah, Understanding LSTM Networks, 2015)

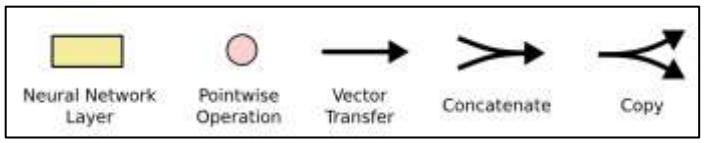
Pada gambar 2.2 dalam satu *RNN* terdapat banya salinan yang sama. (x_t) sebagai input, (h_t) sebagai output dan terdapat alur perulangan yang memungkinkan informasi dilewatkan dari satu langkah jaringan ke langkah berikutnya.

2.5 *Long Short-Term Memory*

Long Short Term Memory (*LSTM*) (Sepp Hochreiter, 1997) adalah salah satu variasi dari Recurrent Neural Network yang dibuat untuk menghindari masalah ketergantungan jangka Panjang pada *Recurrent Nueral Network* (*RNN*). *LSTM* dapat mengingat informasi jangka panjang. Sama seperti *Recurrent Neural Netowork, LSTM* juga terdiri dari modul pemrosesan berulang seperti pada gambar 2.3.

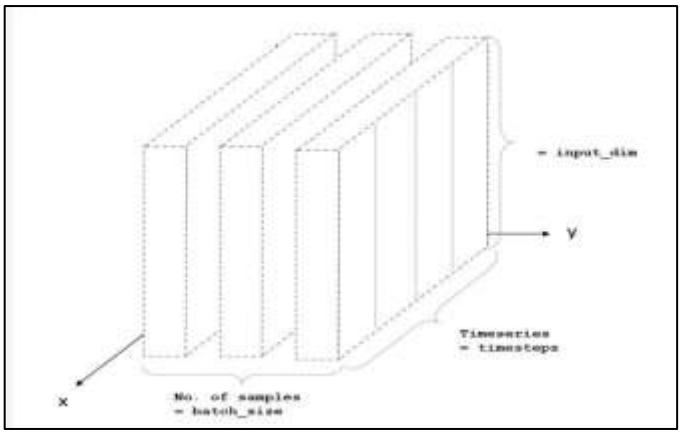


Gambar 2,3 Perulangan Layer Pada *LSTM* (Olah, Understanding LSTM Networks, 2015)



Gambar 2.4 Notasi Pada *LSTM* (Olah, Understanding LSTM Networks, 2015)

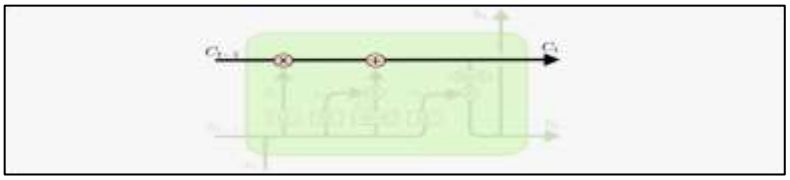
Pada gambar 2.4 setiap baris membawa seluruh vektor, dari output satu node ke input lainnya. Output satu node yang dimaksud adalah output dari jaringan lstm sebelumnya yang dijadikan input dalam kondisi lstm saat ini maka output tersebut menjadi (h_{t-1}). Input dan output lstm adalah vektor. Contoh vector sederhana [0,4,6] pada vector ini memiliki 3 fitur input yaitu 0, 4, dan 6 yang dimana angka tersebut mewakilkan informasi dari data yang akan diproses. Pada fitur input biasanya disebut 3D tensor with shape yaitu: (*batch_size*, *timesteps*, *feature*). *Batch_size* adalah ukuran urutan dari data yang akan diproses, *timesteps* adalah urutan waktu dari data, kemudian *feature* atau *input_dim* adalah ukuran dari dimensi data pada dasarnya menyesuaikan dengan *input_length* pada layer *embedding*. Contoh bentuk fitur input *shape* dari *LSTM* sendiri seperti pada gambar 2.5.



Gambar 2.5 Shape *LSTM (Keras)* (Verma, 2014)

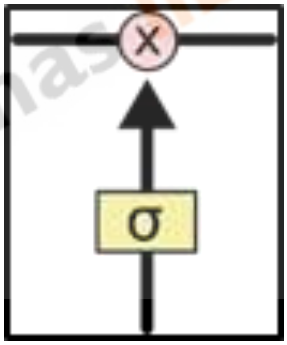
2.5.1 Ide Inti LSTM

Ide dari *LSTM* adalah jalur yang menghubungkan konteks lama (C_{t-1}) ke konteks baru (C_t) di bagian atas modul *LSTM* seperti pada gambar 2.7.



Gambar 2.6 Cell State (Olah, Understanding LSTM Networks, 2015)

Pada gambar 2.6 disebut juga cell state, memory cell atau jalur memori. Dengan adanya jalur pada gambar 2.6, suatu nilai pada konteks yang lama akan dengan mudah dihubungkan ke konteks yang baru dengan sedikit sekali modifikasi, kalau diperlukan. *LSTM* memang memiliki kemampuan untuk menghapus atau menambahkan informasi ke keadaan sel, diatur dengan cermat oleh struktur yang disebut gerbang sigmoid. Gerbang sigmoid (sigmoid gate) yang mengatur berapa banyak informasi bisa yang lewat. Bentuk gerbang sigmoid seperti pada gambar 2.7.



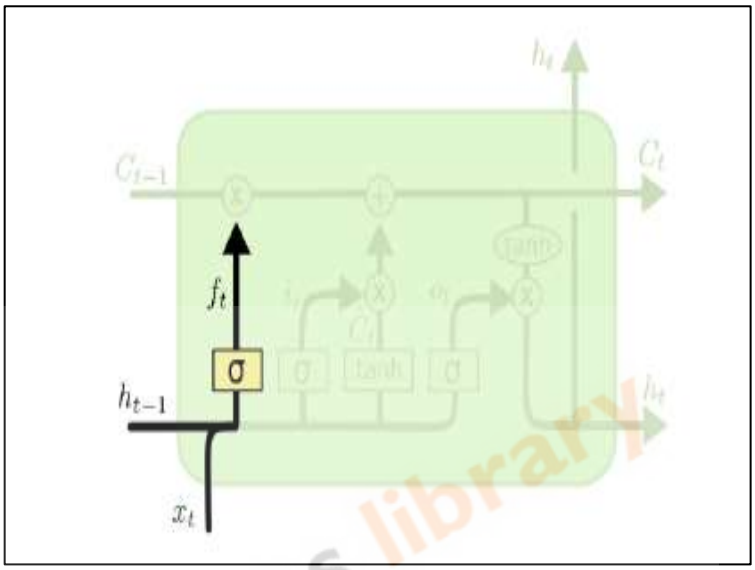
Gambar 2.7 *Sigmoid Gate* (Olah, Understanding LSTM Networks, 2015)

Output dari sigmoid layer adalah angka 1 atau 0 yang menunjukkan apakah informasi tersebut akan disimpan atau dilupakan. Angka 0 menunjukkan bahwa informasi tersebut dihapus, sedangkan angka 1 menunjukkan bahwa informasi disimpan.

2.5.2 Langkah-langkah LSTM

- Pada sub bab ini menjelaskan proses atau langkah-langkah yang dilakukan LSTM. Langkah pertama dalam *LSTM* adalah *forget gate* memutuskan informasi apa yang akan dibuang dari kondisi sel. Keputusan ini dibuat oleh sigmoid gate layer yang disebut *forget gate*

layer. Pada (h_{t-1}) dan (x_t) yang digabungkan *(concatenate)* akan melalui aktivasi sigmoid dimana informasi saat ini pada (h_{t-1}) dan (x_t) yang digabungkan akan disimpan atau ditolak (dihapus). Bila informasi tersebut difungsikan 1 menunjukkan informasi tersebut disimpan sementara angka 0 menunjukkan informasi tersebut dihapus (Olah, Understanding LSTM Networks, 2015).



Gambar 2.8 *Forget gate* (Olah, Understanding LSTM Networks, 2015)

Pada gambar 2.8 gerbang *forget* (Sepp Hochreiter, 1997) memiliki persamaan pada persamaan 2.1.

$$f_t = \sigma(w_f . [h_{t-1}, x_t] + b_f) \quad \text{.....(2.1)}$$

Dimana:

f_t = forget gate

σ = fungsi sigmoid

W_f = weight untuk forget gate

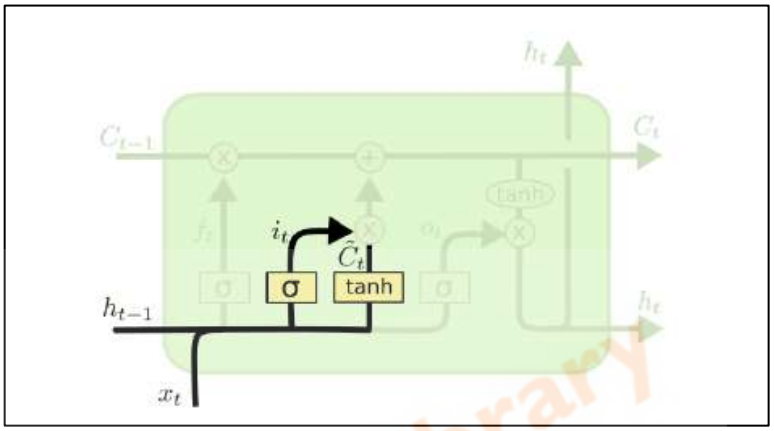
h_{t-1} = output sebelum orde ke t

x_t = input pada orde ke t

b_f = bias pada forget gate

notasi $[h_{t-1}, x_t]$ merupakan operasi *(concatenate)* dari (h_{t-1}) dan (x_t) yang akan masuk ke *forget gate* (f_t) terlepas informasi disimpan atau dihapus tergantung pada kondisi saat itu setelah melalui aktivasi sigmoid seperti pada gambar 2.9

- Langkah kedua adalah *input gate* memutuskan informasi baru apa yang akan digunakan pada keadaan sel (C_t). Proses ini memiliki dua bagian. Pertama, lapisan sigmoid yang disebut *input gate* (i_t) yang akan memutuskan informasi yang akan diperbarui. Informasi yang digunakan adalah informasi saat ini yaitu . (h_{t-1}) dan (x_t). Selanjutnya, lapisan tanh berisi informasi yang akan dijadikan kandidat konteks baru yaitu $\tilde{C}_t$,. Kedua bagian itu akan digabungkan untuk pembaruan konteks baru.



Gambar 2.9 Input Layer dan Tanh Layer (Olah, Understanding LSTM Networks, 2015)

Pada gambar 2.9 input gate memiliki persamaan (Sepp Hochreiter, 1997) seperti pada persamaan 2.2.

$$i_t = \sigma(Wi.\,[h_{t-1}, x_t] + b_i) \quad \text{.....(2.2)}$$

Dimana:

i_t = *input gate*

σ = fungsi sigmoid

Wi = *weight* untuk *input gate*

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

x_t = input pada orde ke t

b_i = bias pada *input gate*

Berdasarkan persamaan 2.2 akan diputuskan informasi yang akan diperbarui menggunakan fungsi aktivasi sigmoid. Informasi (notasi $[h_{t-1}, x_t]$) tersebut bukan hasil dari *forget gate* tetapi informasi saat ini. Pada gambar 2.10 kandidat baru ($\tilde{C}_t$) memiliki persamaan (Sepp Hochreiter, 1997) seperti pada persamaan 2.3

$$\tilde{C}_t = tanh(WC.\,[h_{t-1}, x_t] + b_C) \quad \text{.....(2.3)}$$

Dimana:

$\tilde{C}_t$= informasi kandidat konteks baru yang akan ditambahkan ke cell state

$tanh$ = fungsi tanh

WC = weight untuk cell state

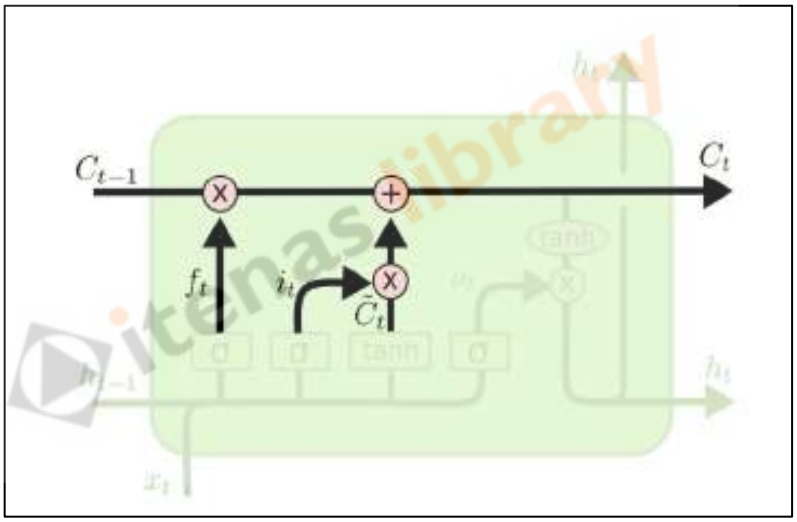
h_{t-1} = output sebelum orde ke t

x_t = input pada orde ke t

b_C = bias untuk cell state

Berdasarkan persamaan 2.3 pada fungsi tanh yaitu akan membuat informasi tersebut menjadi informasi kandidat ($\tilde{C}_t$) disimpan pada *memory cell*.

- Kemudian langkah ketiga memperbaharui *cell state* atau konteks, yaitu konteks lama (C_{t-1}) akan diperbaharui menjadi konteks baru (C_t) yang didapatkan dari menggabungkan hasil yang terdapat pada *forget gate* dan *input gate* seperti pada gambar 2.11.



Gambar 2.10 Mengupdate Cell State Lama ke Cell State Baru (Olah, Understanding LSTM Networks, 2015)

Pada gambar 2.10 gerbang *cell gates* memiliki persamaan (Sepp Hochreiter, 1997) seperti pada persamaan 2.4.

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

Dimana :

C_t = Cell state

f_t = forget gate

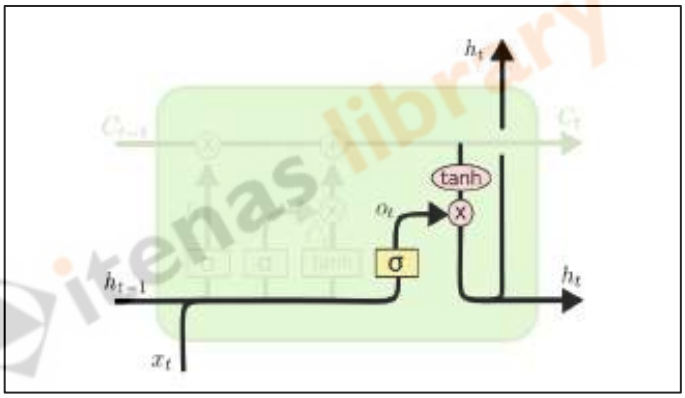
C_{t-1}= Cell state sebelum orde ke t

i_t = input gate

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

Berdasarkan persamaan 2.4 yaitu mengalikan hasil dari *forget gate* (ft) dengan *cell state* sebelumnya (C_{t-1}). Kemudian mengalikan *input gate* (i_t) dengan nilai kandidat konteks baru ($\tilde{C}_t$). Lalu hasil perkalian *forget gate* (ft) dengan *cell state* sebelumnya (C_{t-1}) ditambahkan dengan hasil perkalian *input gate* (i_t) dengan kandidat konteks baru ($\tilde{C}_t$) utnuk menghasilkan *cell state* (C_t).

- Kemudian proses *output gate* untuk menentukan hasil output. Pertama melakukan fungsi sigmoid dimana (h_{t-1}) dan (x_t) digabungkan *(concatenate)* menjadi output. Pada (h_{t-1}) dan (x_t) di *output gate* adalah ini informasi saat ini. Lalu konteks atau *cell state* (C_t) akan melalui tanh layer. Lalu hasil yang telah diproses pada fungsi tanh akan dikalikan dengan hasil dari fungsi sigmoid tadi sehingga menghasilkan output (h_t) seperti pada gambar 2.11.



Gambar 2.11 *Output Gates* (Olah, Understanding LSTM Networks, 2015)

Pada gambar 2.11 *sigmoid* pada *output gates* memiliki persamaan (Sepp Hochreiter, 1997) seperti pada persamaan 2.5.

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

Dimana :

o_t = *output gate*

σ = fungsi sigmoid

Wo = nilai *weight* 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*

Berdasarkan persamaan pada persamaan 2.5 adalah fungsi sigmoid dimana notasi (h_{t-1}) dan (x_t) digabungkan *(concatenate)* akan digunakan menjadi output gate (o_t).

Pada gambar 2.11 hasil *output gate* memiliki persamaan (Sepp Hochreiter, 1997) seperti pada persamaan 2.6.

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

Dimana :

h_t = output orde t

ot = *output gate*

$tanh$ = fungsi tanh

C_t = *Cell state*

Berdasarkan persamaan pada tabel 2.6 adalah hasil yang telah diproses pada fungsi tanh yaitu tanh (C_t) akan dikalikan dengan hasil dari fungsi sigmoid tadi yaitu *output gate (*o_t) untuk menghasilkan informasi orde ke t (h_t) yaitu hasil dari proses panjang lstm. Pada (h_t) akan menjadi (h_{t-1}) untuk proses lstm selanjutnya.

2.6 *Word Embedding*

Word Embedding adalah istilah yang digunakan dalam merepresentasikan sebuah kata ke *vector* kata yang dipopulerkan oleh Mikolov dimana menghasilkan fitur yang padat pada dimensi *vector* yang rendah (Tomas Mikolov, 2013). Eksperimen yang banyak dilakukan penelitian lain menunjukkan bahwa word embedding mampu menghasilkan fitur vector yang lebih baik.

Sebelumnya *word embedding* disebut konsep *pre-trained model* yang diperkenalkan oleh Ronan dan Jason yang menunjukkan bahwa konsep ini merupakan pendekatan yang luar biasa untuk masalah NLP (Ronan Collobert, 2008). Konsep *pre-trainned model* inilah yang nantinya digunakan Mikolov ketika menghasilkan sebuah pre-trained model yang diberi nama Word2Vec.

Ketika proses training dilakukan pada *word embedding*, maka hasil yang didapatkan salah satunya berupa kemiripan atau kedekatan antara kata dan juga relasi yang lainnya (PRIANSYA, 2017).

2.6.1 *Word2Vec*

Word2vec bertujuan untuk merepresentasikan kata-kata kedalam vector yang rapat dan memliki dimensi yang rendah yang dimana disebut mengimplementasikan metode *word embedding* (Tomas Mikolov, 2013). Dan menurut kadam word2vec merupakan sekelompok model untuk menghasilkan *word embedding* dimana model tersebut mengkonstruksi *linguistic* dari kata-kata tersebut (Kadam, 2017).

Dalam pembentukan metode *Word2Vec* ada dua pendekatan yang digunakan yaitu *Continuous Bag-of-Words model (CBOW)* dan *Skip-gram* model (Tomas Mikolov, 2013). *word2vec* memiliki banyak keunggulan dibandingkan dengan metode word representation lainnya, seperti lebih cepat dalam proses training-nya, lebih efisien, dapat menangani dataset dengan skala yang besar (PRIANSYA, 2017).

Metode word to vec ini terdiri dari dua buah algoritma utama word embedding didalamnya, yaitu: *continuous bag of word (CBOW*) dan *skip gram*. Algoritma CBOW memprediksi sebuah kata berdasarkan konteks kata-kata yang ada di sektiarnya yang digunakan untuk melihat panjang tertentu dari sebuah kata pada dokumen masukan. Sedangkan algoritma *skip gram* digunakan untuk memprediski konteks kata yang berdekatan dengan kata lainnya baik itu kata (Tomas Mikolov, 2013).

2.6.2. *Glove*

Glove atau *Global Vectors for Word Representation* adalah bertujuan untuk merepresentasikan vector kata sama seperti *Word2vec*. Pada dasarnya *Glove* merupakan bentuk umum dari algoritma nya *Skip-gram* (Jeffrey Pennington, 2014). Inti dari model Skip-gram yaitu memaksimalkan peluang logaritmik dari vektor kata dengan konteksnya.

Oleh sebab itu proses pelatihan perlu dilakukan dengan data yang sangat besar untuk mencapai titik optimumnya. Sehingga dapat diperoleh vektor representasi semua kata dalam kosa kata yang ada dan dapat digunakan dalam proses NLP selanjutnya. Berbeda dengan word2vec yang dibuat oleh (Tomas Mikolov, 2013) dari google sedangkan glove dibuat oleh (Jeffrey Pennington, 2014) dari Stanford.

Representasi vector yang dimiliki Glove memiliki beberapa dimensi antara lain 50, 100, 200, dan 300. Agar mempermudah dalam waktu komputasi (DERWIN SUHARTONO, 2016).

2.7 Pengujian Kinerja Sistem

Pada penelitian ini dilakukan pengujian kinerja sistem dari prediksi yang diberikan classifier menggunakan *Confusion.Matrix.* Adapun pengukuran menggunakan parameter istilah yaitu *acuracy*, *precision*, *recall*, *f-measure* (Dea Herwinda Kalokasari, 2017). Kemudian pada *Confusion.Matrix* terdapat 4 (empat) istilah sebagai representasi hasil proses klasifikasi diantaranya *True Positive* (TP), *True Negative* (TN), *False Positive* (FP) dan *False Negative* (FN) (Aini Suri Talita1, 2019).

Table 2.1 Confusion Matrix

		Positive	*Negative*
Prediksi	*Positive*	*True Positive* (TP)	*False Negative* (FN)
	Negative	*False Positive* (FP)	*True Negative* (TN)

$$precission = \left(\frac{TP}{TP + FP}\right) \times 100\% \quad \text{.....(2.7)}$$

$$recall = \left(\frac{TP}{TP + FN}\right) \times 100\% \quad \text{.....(2.8)}$$

$$accuracy = \left(\frac{TP + TN}{TP + FP + FN + TN}\right) \times 100\% \quad \text{.....(2.9)}$$

$$F\ Measure = 2\ x\ \frac{(Precision\ x\ Recall)}{(Precision + Recall)} \quad \text{.....(2.10)}$$

Dimana :

TP = Dikategorikan *hoax* sebenarnya memang *hoax.*

TN = Dikategorikan *real* atau *non-hoax* sebenarnya memang *real* atau *non-hoax.*

FP = Dikategorikan *hoax* namun sebenarnya *real* atau *non-hoax.*

FN = Dikategorikan *real* atau *non-hoax* namun sebenarnya *hoax.*

2.8 Studi Kasus

Studi kasus digunakan untuk memberi contoh perubahan yang terjadi pada teks pada setiap proses pengolahan dengan menggunakan metode-metode yang digunakan pada penelitian ini.

2.8.1 Studi Kasus *Case Folding*

Penerapan case folding bertujuan untuk menyamakan seluruh jenis karakter yang terdapat dalam teks berita sehingga memudahkan dalam proses penghapusan karakter atau kata-kata tertentu yang tidak diinginkan dalam penelitian ini. Contoh seperti pada tabel 2.2.

Table 2.2 *Case Folding*

Sebelum Case Folding	Setelah Case Folding
"The Taliban was kill American People"!!!	["the taliban was kill american people"!!!]

2.8.2 Studi Kasus *Punctuation Removal*

Pada tahapan ini tanda baca (seperti ? ! , / = + - \ > < ; “ ( ) {} [ ] . : | dan lainnya) akan diganti dengan spasi. Penghapusan ini dilakukan karena tanda baca tidak dihiraukan selama proses training sehingga dengan penghapusan tanda baca proses training akan menjadi lebih sederhana. Contoh seperti pada tabel 2.3.

Table 2.3 *Punctuation Removal*

Sebelum Punctuation Removal	Setelah Punctuation Removal
"the taliban was kill american people"!!!	[the taliban was kill american people]

2.8.3 Studi Kasus *Stopword Removal*

Stopword merupakan kata-kata yang sering muncul dan biasanya diabaikan dalam pemrosesan. Penghapusan stopword bertujuan untuk mengurangi jumlah kata dalam sebuah dokumen yang nantinya akan berpengaruh dalam kecepatan dan performa dalam kegiatan NLP. Contoh seperti pada tabel 2.4. dan daftar kata yang di hapus pada tabel 2.5.

Table 2.4 *Stopword Removal*

Sebelum Stopword Removal	Setelah Stopword Removal
the taliban was kill american people	[taliban kill american people]

Table 2.5 Daftar Kata Yang Di Hapus

Daftar Stopword yang dihapus
'the', 'was'

2.8.4 Studi Kasus *Tokenize*

Tokenisasi berfungsi melakukan pemisahan untuk setiap kata, rangkaian angka, dan rangkaian angka dengan huruf yang memiliki makna tetentu. Contoh seperti pada tabel 2.6.

Table 2.6 *Tokenize*

Sebelum Tokenize	Setelah Tokenize
taliban kill american people	['taliban', 'kill', 'american', 'people']

2.8.5 Studi Kasus *Word Embedding*

Embedding layer merupakan layer yang menerima input berupa 2D tensor integer menggunakan fungi yang disediakan oleh *keras Timeseries data preprocessing* yaitu pad_sequence. Sebelum diubah menjadi tensor harus menentukan jumlah kata maksimal yang paling sering muncul didalam dataset yang akan dijadikan kamus untuk proses training model. Pada penelitian ini menggunakan 100 kata (*Num Words*) yang frekuensinya paling banyak di dataset. Kemudian ditentukan maksimal jumlah kata (*maxlen*) dari masing-masing teks sebanyak 10 kata (berdasarkan 100 kamus yang ditentukan diawal). Kemudian 1000 kata tersebut dikonversi menjadi sequence of integer (urutan kata yang paling sering muncul dalam bentuk integer). Setelah format menjadi sequence of integer kemudian diubah menjadi tensor 2-Dimensi dengan ketentuan (100, 10) artinya 100 sequence dengan masing-masing panjang 10.

Semua sequences yang akan menjadi input di embedding layer harus mempunyai panjang yang sama. Untuk itu dilakukan pemampatan sequences

apabila kurang dari 10 maka sequence akan dihapuskan, dan apabila sequences lebih dari 10 maka akan dipotong dengan batas panjang maksimal 10 atau yang biasa disebut *maxlen*.

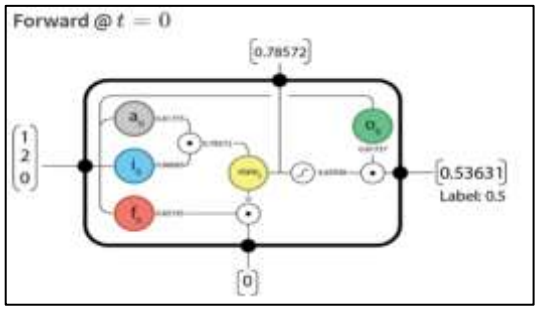
Table 2.7 Bentuk 2d Tensor shape

2D tensor integer
[[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.] [2. 0. 0. 0. 0. 0. 0. 0. 0. 0.] [3. 0. 0. 0. 0. 0. 0. 0. 0. 0.] [4. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

Kemudian masuk ke proses embedding layer dengan input 2d tensor integer yang dimana embedding layer membutuhkan input shape yaitu (batch_size, input_length) yang dimana sudah ada pada 2d tensor integer jika dilihat ke bentuk shape. Pada 2d tensor integer batch_size dan input_length nya adalah (4, 10). Lalu pada layer ini akan menghasilkan 3D tensor with shape yaitu: (batch_size, input_length, output_dim).

2.8.6 Studi Kasus *Long Short-Term Memory*

Pada kasus *LSTM* disini yaitu dimana ide dari *LSTM* sendiri adalah jalur yang menghubungkan konteks lama (C_{t-1}) ke konteks baru (C_t) seperti yang dijelaskan pada sub-bab 2.6.1. Konteks (C_t) disebut juga cell state, memory cell atau jalur memori. Dengan adanya jalur di atas, suatu nilai pada konteks yang lama akan dengan mudah dihubungkan ke konteks yang baru dengan sedikit sekali modifikasi, kalau diperlukan. Pada kasus ini membutuhkan fitur input berupa 3D tensor with shape yaitu: (*batch_size*, *timesteps*, *feature*). *Batch_size* adalah ukuran urutan dari data yang akan diproses, *timesteps* adalah urutan waktu dari data, kemudian *feature* atau *input_dim* adalah ukuran dari dimensi data pada dasarnya menyesuaikan dengan *input_length* pada layer *embedding*. Contoh hasil pada 3D tensor yang dihasilkan adalah (*batch_size*=14, *timesteps* =2, *feature*=10). Bentuk *shape* dari *LSTM* sendiri seperti pada gambar 2.5.



Gambar 2.12 Arsitektur Dalam 1 Unit *LSTM* (Gomez, 2016)

Pada gambar 2.12 merupakan bentuk dari arsitektur setiap 1 *lstm*. Pada arsitektur *LSTM* sendiri setiap proses ada 4 cara kerja yang dinamakan *gates* yaitu *forget gates* untuk rumus terdapat pada persamaan 2.1, *input gates* untuk rumus terdapat pada persamaan 2.2, *cell gates* untuk rumus terdapat pada persamaan 2.3, dan *output gates* untuk rumus terdapat pada persamaan 2.4. Kemudian $(x_t) = [\frac{1}{2}]$ adalah vektor sebagai input atau informasi yang akan diproses dan diperoleh dari word embedding yang dimana merepresentasikan sebuah kata dan (C_{t-1}) = 0 adalah sebuah informasi pada memori sebelumnya untuk LSTM t_0. Kemudian setiap gerbang atau *gate* mempunyai bobot dan bias masing-masing untuk setiap (x_t) dan (C_{t-1}). Bobot awal dari gerbang tersebut diperoleh secara random dengan menginisialisasi nilai yang relative kecil (Mitchell, 1997). Untuk tahapan nya sebagai berikut:

1. Pada *forget gates* (f_t) atau (f_o) memutuskan informasi apa yang akan dibuang dari kondisi sel. Contoh informasi yang dipakai saat ini pada tabel 2.8. Keputusan ini dibuat oleh aktivasi sigmoid gate yang disebut *forget gate*. Pada (h_{t-1}) dan (x_t)akan digabungkan *(concatenate)* yaitu mengkalikan bobot pada (x_t) yaitu [0.7 0.45] dengan nilai (x_t) yaitu $[\frac{1}{2}]$ kemudian untuk (h_{t-1}) juga sama mengkalikan bobot pada (h_{t-1}) yaitu [0.1] dengan nilai (h_{t-1}) yaitu 0. Kemudian dari kedua hasil tersebut ditambakan dengan bias gerbang forget yatu $[0.15]$. seperti persamaan 2.1.

Table 2.8 Forget Gate

Forget Gate (f_t)

$$f_t = \sigma(w_f.[ht_{-1}, x_t] + b_f)$$

$$f_t = \sigma([0.7\ 0.45]\begin{bmatrix}1\\2\end{bmatrix}+[0.1][0]+[0.15])$$

$$f_t=\sigma[(1.6]+[0]+[0.15])$$

$$f_t=\sigma(1.75)= 0.85195$$

2. Pada *input gates* memutuskan informasi baru apa yang akan digunakan pada keadaan sel (C_t). Proses ini memiliki dua bagian. Pertama, lapisan sigmoid yang disebut input gate (i_t) atau (i_o) yang akan memutuskan informasi yang akan diperbarui untuk tahapan proses operasi mnya sama seperti *forget gate*. Informasi yang digunakan adalah informasi saat ini yaitu . (h_{t-1}) dan (x_t) menggunakan persamaan 2.2. Selanjutnya, lapisan tanh berisi informasi yang akan dijadikan kandidat konteks baru yaitu ($\tilde{C}_t$) atau (a_o). Untuk operasi nya sama seperti gerbang *forget* bedanya adalah pada ($\tilde{C}_t$) atau (a_o) tidak memakai fungsi sigmoid tetapi memakai fungsi tanh seperti pada persamaan 2.3,. Kedua bagian itu akan dipakai untuk pembaruan konteks baru atau *cell state* (C_t).

Table 2.9 input Gate

Input Gate (i_t)
$i_t = \sigma(w_i.[ht_{-1}, x_t] + b_i)$ $i_t = \sigma([0.95\ 0.8]\ [\begin{matrix}1\\2\end{matrix}] +[0.8][0]+[0.65])$ $i_t=\sigma[(2.55]+[0]+[0.65])$ $i_t=\sigma(3.2)= 0.96083$ $\tilde{c}_t = \tanh(w_c.[ht_{-1},xt]+b_c)$ $\tilde{c}_t = \tanh([0.45\ 0.25]\ [\begin{matrix}1\\2\end{matrix}] +[0.15][0]+[0.2])$ $\tilde{c}_t=\tanh[(0.95]+[0]+[0.2])$ $\tilde{c}_t=\tanh(1.15)=0.81775$

3. Pada *cell gates* yaitu konteks lama (C_{t-1}) akan diperbaharui menjadi konteks baru (C_t) yang didapatkan dari hasil *forget gate* atau (f_t) atau (f_o) dan hasil *input gate*s (i_t) atau (i_o) menggunakan persamaan pada table 2.4 yaitu mengalikan hasil dari *forget gate* (ft) = 0.85195 dengan *cell state* sebelumnya (C_{t-1}) = 0. Kemudian mengalikan *input gate* (i_t) = 0.81775 dengan informasi kandidat konteks baru ($\tilde{C}_t$) atau. (a_o) = 0.96083 . Lalu hasil perkalian *forget gate* (ft) dengan *cell state* sebelumnya (C_{t-1}) ditambahkan dengan hasil perkalian *input gate* (i_t) dengan kandidat konteks baru ($\tilde{C}_t$) utnuk menghasilkan *cell state* (C_t).

   Berikut adalah contoh hasil proses pada *cell gates* konteks (C_t).

   Table 2.10 Konteks (C_t) atau *cell gates*

Konteks (C_t)
$c_t = f_t * c_{t-1} + i_t . \tilde{c}_t$
$c_t = 0.85195 \times 0 + 0.81775 \times 0.96083 = 0.78572$

   Pada konteks (C_t) atau *cell gates* yang telah dihubungkan oleh konteks lama (C_{t-1}) setelah melalui proses panjang yaitu dari forget gates, lalu *input gates* – *cell gates* tidak banyak mengalami perubahan. Pada vector(matrix) tabel 2.10 adalah informasi baru yang dihasilkan oleh *cell gates* yang dimana informasi ini akan dijadikan konteks lama (C_{t-1}) pada LSTM selanjutnya. Jika kondisinya LSTM saat ini maka (C_t).

4. Kemudian *output gates* menentukan hasil output. Pertama melakukan fungsi sigmoid dimana (h_{t-1}) dan (x_t) digabungkan *(concatenate)* menjadi output *(o_t)*. Pada (h_{t-1}) dan (x_t) di *output gate* adalah ini informasi saat ini. Pada proses output *(o_t)*. menggunakan persamaan 2.5. Lalu konteks atau cell state (C_t) yang telah diproses pada *cell gates* akan melalui tanh layer. Lalu hasil yang telah diproses pada fungsi tanh akan dikalikan dengan hasil dari (o_t) tadi sehingga menghasilkan output orde

ke t (h_t) menggunakan persamaan 2.6. Berikut adalah output orde ke t (h_t). Berikut adalah hasil dari *output gates* dengan menggunakan *keras.*

Table 2.11 Output orde t (h_t)

Output orde t (h_t).
$o_t = \sigma(w_o.[ht_{-1}, x_t]+b_o)$ $o_t = \sigma([0.6\ 0.4]\ [\frac{1}{2}] +[0.25][0]+[0.1])$ $o_t=\sigma[(1.4]+[0]+[0.1])$ $o_t=\sigma(1.5)=0.81757$ $h_t = o_t*\tanh(c_t)$ $h_t = \tanh(0.78572)\times 0.81757$ $h_t=0.65597\times 0.81757=0.53631$

2.8.7 Studi Kasus *Softmax*

Fungsi softmax diterapkan pada vektor 1D pada hasil (h_t). Perhitungan hasil aktivasi softmax dijelaskan sebagai berikut:

Misalkan diambil 2 *feature vector* dari hasil matriks (h_t) [0.53631,0.31812]

1. Jumlahkan hasil eksponen pada setiap elemen di lapisan (h_t)

$$exp^{(x1)} = e^{0.53631} = 1.70969$$
$$exp^{(x2)} = e^{0.31812} = 1.37454$$

$\sum_j exp^{(xi)} = 1.70969 + 1.37454 = 3.08423$

2. Lakukan operasi pembagian antara *ekseponen* elemen (h_t) dengan hasil penjumlahan eksponennya untuk mendapatkan nilai probabilitas dari *softmax*

$\hat{y} = \frac{0.53631}{3.08423}, \frac{0.31812}{3.08423},$

$\hat{y} = [0.165340, 0.10314]$