BAB II

TINJAUAN PUSTAKA

**2.1 Emosi**

Emosi adalah kunci perasaan dan pikiran orang (Jabreel & Moreno, 2019). Perasaan intens yang diarahkan pada sesuatu atau seseorang sebagai respon terhadap peristiwa internal maupun eksternal bagi individu digambarkan sebagai emosi (Gaind et al., 2019). Emosi memperkenalkan cara mengekspresikan perasaan dalam bentuk komunikasi, sehingga menambah cita rasa hidup (Rohman et al., 2019). Emosi bersifat umum dan penting dalam semua aspek kehidupan manusia (Fanesya et al., 2019).

Variasi jenis emosi dapat didasarkan kepribadian, jenis kelamin, lokasi, etnis, budaya, situasi, di samping banyak parameter psikologis, sosial dan individu lainnya (Rohman et al., 2020). Model emosi yang sangat populer adalah model emosi Ekman. Model Ekman membagi emosi menjadi 6 label emosi yaitu bahagia, marah, takut, jijik, sedih, dan terkejut (Kowalska & Wróbel, 2017), sebenarnya banyak model emosi yang lain namun pada model ekman label-label emosi bersifat universal pada budaya yang berbeda.

**2.2 Deteksi Emosi**

Deteksi Emosi merupakan bagian dari area yang lebih luas dari komputasi afektif bertujuan untuk memungkinkan komputer mengenali dan mengekspresikan emosi (Picard, 1997). Pendekatan klasifikasi teks digunakan untuk melakukan deteksi emosi (Fanesya et al., 2019). Manfaat deteksi emosi dapat digunakan untuk mengambil keputusan seperti di bidang pendidikan, bisnis, politik, psikologi, sehingga menunjukkan pentingnya deteksi emosi (Fanesya et al., 2019), selain itu deteksi emosi juga bermanfaat untuk rekrutmen karyawan, koseling , dan profil psikologis karena merupakan bagian dari prediksi kepribadian (Dandannavar et al., 2018) . Emosi dapat dideteksi melalui suara, ekspresi wajah, gerakan tangan, gerakan tubuh, detak jantung, tekanan darah, teks (Consoli, 2009).

**2.3 Media Sosial**

Media sosial merupakan platform komunikasi, interaksi, dan berbagi informasi yang populer melalui internet (Wilson et al., 2012). Media sosial dapat digunakan oleh penggunanya untuk berekspreksi (Sari & Haranto, 2019). Pada media sosial, setiap acara, berita atau aktivitas di seluruh dunia dapat dibagikan, didiskusikan, diposting dan dikomentari oleh jutaan orang (Gaind et al., 2019). Teks, gambar, suara , dan video dapat dibagikan di media sosial (Kotler et al., 2012). Berdasarkan penjelasan tersebut dapat disimpulkan bahwa media sosial merupakan platform berbagi informasi, acara, aktifitas berupa teks, gambar, suara, video melalui internet untuk mengekspresikan penggunanya, dimana informasi, acara, aktifitas tersebut dapat didiskusikan, dan dikomentari oleh jutaan orang. Contoh media sosial seperti *facebook*, *twitter*, *instagram*, *line*.

**2.4 Twitter**

Twitter adalah salah satu media sosial dimana pengguna dapat membaca dan berbagi pesan singkat dengan jumlah maksimal 280 karakter, *twitter* termasuk media sosial dengan kategori *microblogging* (Setiawan & Setyohadi, 2017), hal ini yang membuat *twitter* berbeda dengan media sosial yang lain. *Twitter* didirikan oleh Jack Dorsey sejak 2006, *twitter* kini telah berkembang menjadi media sosial populer (Juwiantho et al., 2020). *Twitter* memiliki laju pertumbuhan pengguna tertinggi di indonesia, pengguna aktif *twitter* di Indonesia menempati posisi ketiga di Asia Pasifik dari tahun 2012 sampai tahun 2018 (Saputri et al., 2019). Oleh karena itu *twitter* dipilih menjadi objek pada penelitian ini.

**2.5 Web Scraping**

*Web scraping* atau *screen scraping* merupakan proses pengambilan dan analisis sebuah dokumen semi-terstruktur tertentu yang diambil dari internet, biasanya berupa halaman-halaman web dalam bahasa *markup* seperti HTML, XHTML untuk digunakan bagi kepentingan lain (Turland, 2010).

*Web scaping* pertama kali dilakukan dengan cara manual, yaitu menyalin data dari website ke komputer kita atau penyimpanan local. Namun, cara ini kurang efektif jika digunakan untuk mengambil data dalam jumlah banyak, karena membutuhkan tenaga dan waktu yang banyak. Cara lain *web scraping* dapat dilakukan dengan *coding*, aplikasi ataupun *extension browser*. Pada penelitian ini menggunakan *coding* untuk *web scraping* dengan memanfaatkan *Application Programming Interface* (API) *key* *twitter* dan *library* dari bahasa pemprograman *python*.

Langkah-langkah dalam melakukan *web scraping* ada beberapa tahapan, sebagai berikut (Josi et al., 2014) :

1. *Create scraping template* : mempelajari dokumen HTML dari website yang akan diambil informasinya untuk tag HTML yang mengapit informasi yang akan diambil.
2. *Explore site navigation* : mempelajari teknik navigasi pada website yang akan diambil informasinya untuk ditirukan pada aplikasi *web scraper* yang akan dibuat.
3. *Automate navigation and extraction* : berdasarkan informasi yang didapat pada tahap dan tahap 2, aplikasi *web scaper* dibuat untuk mengotomatisasi pengambilan informasi dari website yang ditentukan.
4. *Extracted data and package* : informasi yang didapat dari tahap 3 disimpan dalam format data tertentu, seperti *database*, csv.

**2.6 Preprocessing**

*Preprocessing* data seringkali mempengaruhi kinerja dari *machine learning*, sehingga lebih efektif (S. B. Kotsiantis, 2006). *Preprocessing* berguna untuk membersihkan data dari derau dan bagian tidak informatif yang tidak dibutuhkan, sehingga teks siap untuk diklasifikasi (Haddi et al., 2013). *Preprocessing* akan menjadikan data bersih sehingga proses pembuatan vektor kata dan klasifikasi lebih akurat (Nurrohmat & SN, 2019). Tahapan *preprocessing* yang akan digunakan pada penelitian ini adalah *case folding*, *remove punction*, *remove number*, *tokenizing*, *stop removal*, *stemming*.

**2.6.1 Case Folding**

*Case Folding* adalah mengubah seluruh huruf dalam teks tersebut menjadi *lower case* atau huruf kecil semua (Salam et al., 2018), dengan huruf yang diterima dari “a” sampai “z”. Tujuannya agar terdapat standarisasi dalam penulisan. Misalnya terdapat data “Emosi Saya” setelah proses *case folding* menjadi “emosi saya”. Contoh lain dapat dilihat pada **tabel 2.1**.

Tabel **2.1** Case Folding

|  |  |
| --- | --- |
| Data sebelum *case folding* | “Suaranya bagus, apalagi kalau dIiringi gitaR” |
| Data setelah *Case Folding* | “suaranya bagus, apalagi kalau diiringi gitar” |

**2.6.2 Remove Punctuation**

*Remove punctuation* adalah menghapus tanda baca pada teks dengan tujuan mengurangi beban pemprosesan klasifikasi karena dianggap tidak penting dan termasuk *dilimiter*. Beberapa contoh tanda baca yang yang dihapus titik (.), koma(,), tanda tanya (?), *slash* (/), *hastag* (#), tanda seru (!) dan lain-lain. Misal terdapat data “sabar pak?” setelah melalui *remove punctuation* menjadi “sabar pak” Contoh lain dapat dilihat pada **tabel 2.2**.

Tabel **2.2**

|  |  |
| --- | --- |
| Data sebelum *remove punctuation* | “suaranya bagus !!!, apalagi kalau diiringi gitar” |
| Data setelah *remove punctuation* | “suaranya bagus apalagi kalau diiringi gitar” |

**2.6.3 Remove Number**

*Remove number* adalah menghapus angka pada suatu teks, penghapusan ini sebab angka dianggap tidak memiliki arti dan termasuk *delimiter*, mirip seperti *remove punctuation* hanya berbeda pada objek yang dihapus. Misalnya terdapat data “dokumen2 yang dikumpulkan” setelah melalui proses *remove number* menjadi “dokumen yang dikumpulkan”.Contoh lain dapat dilihat pada **tabel 2.3**.

Tabel **2.3**

|  |  |
| --- | --- |
| Data sebelum *remove number* | “barang yang dijual bagus2” |
| Data setelah *remove number* | “barang yang dijual bagus” |

**2.6.4 Tokenizing**

*Tokenizing* atau tokenisasiadalah pemotongan kalimat berdasarkan tiap-tiap kata penyusunnya, biasanya pemotongan berdasarkan *whitespace* seperti spasi, tab, dan enter.Tiap-tiap kata hasil *tokenizing* disebut token (Juwiantho et al., 2020). Misalnya terdapat data “ada saya semua bahagia” setelah melalui proses *tokenizing* akan menghasilkan 4 token, yaitu : “ada”, “saya”, “semua”, “bahagia”. Contoh lain dapat dilihat pada **tabel 2.4**

Tabel **2.4** Tokenizing

|  |  |
| --- | --- |
| Sebelum proses *Tokenizing* | konten yang bermutu namun sayang suaranya kurang jelas |
| Setelah proses *Tokenizing* | [‘konten’,’yang’,’bermutu’,’namun’,’sayang’,’suaranya’,  ’kurang’,’jelas’] |

**2.6.5 Stopword Removal**

*Stopword removal* atau *filtering* merupakan proses untuk menghilangkan kata-kata yang tidak memiliki arti atau makna, namun tidak akan mengubah makna dari komentar tersebut (Juwiantho et al., 2020). Tahap *stopword removal* akan mengurangi ukuran indeks, waktu pemprosesan klasifikasi, dan *noise* dari suatu data. Biasanya *stopword* berupa kata ganti orang dan kata hubung, seperti “aku”, “kamu”, “kita”, “dan”, “atau”. Contoh proses *stopword removal* yang datanya berasal dari proses *tokenizing* sebelumnya dapat dilihat pada **tabel 2.5**

Tabel **2.5** Stopword Removal

|  |  |
| --- | --- |
| Sebelum Stop Removal | [‘konten’,’yang’,’bermutu’,’namun’,’sayang’,’suaranya’,  ’kurang’,’jelas’] |
| Setelah Stop Removal | [‘konten’,’bermutu’,’sayang’,’suaranya’,’kurang’,’jelas’] |

**2.6.6 Stemming**

*Stemming* atau *lemmatization* merupakan proses untuk mentransformasi kata kerja yang berimbuhan pada suatu dokumen menjadi kata dasar (*root word*), dengan menghapus awalan, akhiran maupun sisipan . *Stemming* bertujuan untuk mengurangi varian kata dengan makna yang hampir sama pada suatu dokumen, serta meningkatkan performa pada tahap *information retrieval*. Pada penelitian ini akan menggunakan *library* sastrawi, karena data pada penelitian ini menggunakan bahasa Indonesia. *Libarary* sastrawi menerapkan algoritma Nazief dan Adriani. Berikut langkah-langkah algoritma Nazief & Adriani dalam *stemming* sebagai berikut (Nazief dan Adriani, 2005) :

1. Pertama – tama, periksa kata dengan kamus kata, jika pemeriksaan ada kata tersebut akan dianggap kata dasar dan proses berhenti, jika tidak maka akan lanjut proses ke-2
2. Pada tahap ke-2, hapus *inflectional suffixes* yaitu (“-kah”, “-lah”, “-tah”, “-pun”), kemudian hapus *inflectional possessive pronoun suffixes* seperti (“-ku”, “-mu”, “-nya”), kemudian data hasil *infectional suffixes* dan *inflectional possessive pronoun suffixes* dicek pada kamus kata, jika hasilnya ada maka proses akan berhenti, jika tidak maka lanjut pada proses ke-3. Contoh ada data “hasilnyapun” setelah dihapus *inflectional suffixes* menjadi “hasilnya”, kemudian data dihapus *inflectional possessive pronoun suffixes* menjadi “hasil”.
3. Tahap ke-3, hapus *derivational suffixes* (imbuhan turunan) yaitu (“-i”, “-kan”, “-an”), setelah proses penghapusan *derivational suffixes* (“-i”, “-kan”, “-an”)kata akan dicari pada kamus kata, jika ada maka proses berhenti, namun jika tidak maka huruf terakhir sebelum kata *derivational suffixes* (“-i”, “-kan”, “-an”) dihapus, setelah itu akan kata akan dicari pada kamus, jika ada maka proses berhenti, namun jika tidak kembalikan huruf yang dihapus sebelum *derivational suffixes* (“-i”, “-kan”, “-an”), kemudian lanjut pada proses ke-4. Contoh data “menggabungkan” setelah penghapusan *derivational suffixes* (“-i”, “-kan”, “-an”)menjadi “menggabung”, karena tidak ada dalam kamus, maka hapus huruf sebelum *derivational suffixes* (“-i”, “-kan”, “-an”)menjadi “menggabun”, setelah dicek tidak ada dalam kamus, maka kembalikan kata yang dihapus tadi menjadi “menggabung” dan lanjut pada proses ke-4.
4. Tahap ke-4, hapus *derivational prefixes* yaitu (“be-“, “di-“, “ke-“, “me-“, “pe-“, “se-“, “te-“). Jika pada tahap ke-3 ada *derivational suffixes* (“-i”, “-kan”, “-an”)yang dihapus maka ke langkah 4a, jika tidak ada yang dihapus ke langkah 4b.

a. kata dicek pada tabel **2.6**, jika ada maka proses berhenti, jika tidak maka lanjut ke tahap 4b.

**Tabel 2.6** Kata awalan akhiran dilarang (Nazief dan Adriani, 2005)

|  |  |
| --- | --- |
| **Awalan** | **Akhiran** |
| be- | -i |
| di- | -an |
| ke | -i, -kan |
| me- | -an |
| se- | -i, -kan |
| te- | -an |

b. Untuk i sama dengan 1 sampai 3, tentukan tipe awalan kemudian hapus awalan. Jika kata dasar belum ditemukan juga lakukan tahap ke-5, jika sudah maka proses berhenti. Aturan peluruhan kata dapat dilihat pada Tabel **2.7** dan tabel **2.8**.

**Tabel 2.7** Aturan peluruhan kata dasar

|  |  |  |
| --- | --- | --- |
| Aturan | Awalan | Akhiran |
| 1 | berV . . . | ber-V . . . | be-rV |
| 2 | belajar | bel-ajar |
| 3 | berClerC2 | be-ClerC2 . . dimana C1!= {‘r’|’l’} |
| 4 | terV. . . | ter-V . . | te-rV . . |
| 5 | terCer. . . | terCer. . . dimana C!=’r’ |
| 6 | teClerC2 | te-CleC2 . . . dimana C1!=’r’ |
| 7 | me{I|r|w|y}V. . . | me-{I|r|w|y}V. . . |
| 8 | mem{b|fv}. . . | mem-{b|f|v}. . . |
| 9 | mempe. . . | m-pe. . |
| 10 | mem{r|V|V}. . . | me-m{r|V|V}. . . | me-p{r|V|V}. . . |
| 11 | men{c|d|j|z}. . . | Men-{c|d|j|z}. . . |
| 12 | menV. . . | Me-nV. . . | me-tV. . . |
| 13 | meng{g|h|q|k}. . . | Meng-{g|h|q|k}. . . |
| 14 | mengV. . . | meng-V. . . | meng-kV. . . |
| 15 | mengeC. . . | meng-C. . . |
| 16 | menyV. . . | me-ny. . . | men-sV. . . |
| 17 | memV. . . | mem-pV. . . |
| 18 | pe{w|y}V. . . | pe-{w|y}V. . . |
| 19 | perV. . . | per-V. . . | pe-rV. . . |
| 20 | pem{b|f|v}. . . | pem-{b|f|v}. . . |
| 21 | pem{rV|V}. . . | Pe-m{rV|V}. . . | pe-p{rV|V}. . . |
| 22 | pen{c|d|j|z}. . . | Pen-{c|d|j|z}. . . |
| 23 | penV. . . | Pe-nV. . . | pe-tV. . . |
| 24 | peng{g|h|q}. . . | Peng-{g|h|q} |
| 25 | pengV. . . | Peng-V | peng-kV |
| 26 | penyV. . . | Pe-nya | peny-sV |
| 27 | pelV. . . | Pe-IV. . .; kecuali untuk kata “pelajar” |
| 28 | peCP. . . | Pe-CP. . . dimana C!={r|w|y|I|m|n} dan P!=’er’ |

Tabel **2.8** Lanjutan aturan peluruhan kata dasar

|  |  |  |
| --- | --- | --- |
| Aturan | Awalan | Akhiran |
| 29 | perCerV. . . | Per-CerV. . . dimana C!={r|w|y|I|m|n} |

Langkah-langkah penentuan tipe awalan sebagai berikut :

1. Jika awalannya adalah: “di-”, “ke-“ atau “se-“ maka tipe awalannya secara berturut-turut adalah “di-“, “ke-“ atau “se-”.
2. Jika awalannya adalah: “te-“, “me-“, “be-“ atau “pe-“ maka dibutuhkan sebuah proses tambahan untuk menentukan tipe awalannya.
3. Jika dua karakter pertama bukan “di-“, “ke-“, “se-“, “te-“, “be-“, “me-“ atau “pe-“ maka berhenti.
4. Jika tipe awalan adalah “none” maka berhenti. Hapus awalan jika ditemukan.

5. Tahap ke-5 melakukan *recording*, pada awal kata yang dipenggal, diberi tambahan karakter *recording* **Tabel 2.7** dan **Tabel 2.8**. Jika terdapat kombinasi awalan yang dihilangkan sebelumnya atau tiga awalan yang telah dihilangkan maka proses berhenti

6. Tahap ke-6, jika semua Langkah sudah selesai, namun belum menemukan kata dasar, maka akan dikembalikan ke kata awal dan dianggap kata awal merupakan kata dasar, proses selesai.

Contoh proses *stemming* dapat dilihat pada **tabel 2.9**

Tabel **2.6** Stemming

|  |  |
| --- | --- |
| Sebelum Stemming | [‘konten’,’bermutu’,’sayang’,’suaranya’,’kurang’,’jelas’] |
| Setelah Stemming | [‘konten’,’mutu’,’sayang’,’suara’,’kurang’,’jelas’] |

**2.6 Klasifikasi**

Klasifikasi merupakan proses mendapatkan model atau fungsi dari analisis atau mempelajari himpunan dokumen teks yang belum diketahui kelasnya, model atau fungsi dapat digunakan untuk mengkategorikan dokumen teks lain yang belum diketahui kelasnya kedalam satu atau lebih kelas yang telah ditentukan berdasarkan isinya (Sebastiani, 2002). Klasifikasi bertujuan untuk mengkategorikan kelas dari suatu dokumen teks yang belum diketahui kelasnya (Gaikwad et al., 2014). Berdasarkan jumlah label klasifikasi dapat dibagi menjadi 2 kategori, yaitu :

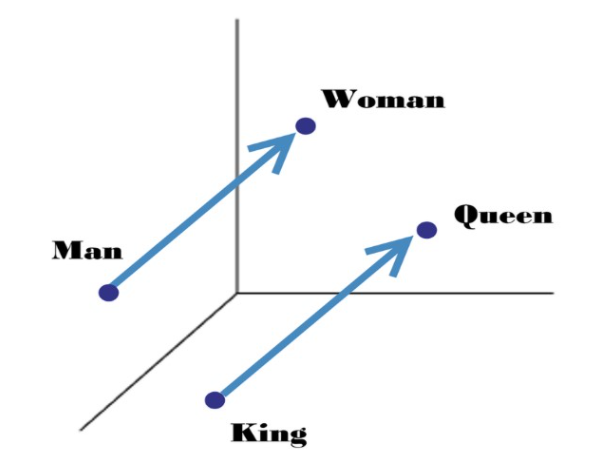
1. *Binaryclass*, mengklasifikasikan menjadi 2 kelas saja misalnya positif dan negatif, hoax dan fakta.
2. *Multiclass*, mengklasifikasikan menjadi lebih dari 3 kelas misalnya bahagia, sedih, marah.

**2.7 Word Embedding**

*Word embedding* merupakan suatu teknik memetakan kata-kata berdasarkan suatu kamus yang sudah ada sehingga menghasilkan vektor-vektor angka yang berisi angka rill (Halim et al., 2020), hal ini dilakukan karena *deep learning* tidak dapat memproses data *string* atau teks. Oleh karena itu data harus diubah dari data *string* atau teks menjadi vektor angka menggunakan *word embedding*. *Word embedding* juga dapat menangkap makna semantik dan sintaktik kata (Nurdin et al., 2020). Hasil dari *word embedding* dapat digunakan ke dalam ruang geografis yang dinamakan ruang *embedding* (Faadilah, 2020). Misalnya terdapat data “indonesia tanah air ku” akan diubah menjadi vektor menghasilkan kamus kata [“indonesia”, ”tanah”, “air”, “ku”]. Metode *one-hot encoding* akan merepresentasikan kata dengan angka 1, dan angka 0 untuk kata lainnya dalam bentuk vektor. Sehingga kata “indonesia” akan direpresentasikan dalam vektor menjadi [ 1, 0, 0, 0], kata “tanah” menjadi [ 0, 1, 0, 0], kata “air” menjadi [ 0, 0, 1, 0], kata “ku” menjadi [ 0, 0, 0, 1]. Selain metode *one-hot encoding* terdapat metode lain seperti *word2vec*, GloVe, *fasttext*.

**2.8 Word2vec**

*Word2vec* merupakan salah satu metode *word embedding* yang diperkenalkan oleh Mikolov dkk pada tahun 2013. *Word2vec* memetakan setiap kata ke dalam vektor yang dapat membawa makna semantik atau sintatikal dari kata tersebut (Nurdin et al., 2020). Proses memetakan kata pada *word embedding* menggunakan *unsupervised neural network* yang terdiri dari sebuah *hidden layer* dan *fully connected layer* (Nurdin et al., 2020). *Word2vec* memiliki kelebihan dalam similaritas makna kata yang didapatkan dari memperhatikan kesamaan kata-kata disekitar kata target (Lim et al., 2020), karena kelebihan itu metode *word2vec* sangat populer. Misalnya ada kata “king” cenderung dimiliki kata “man” maka kata “woman” akan sama artinya dengan kata “queen”, hubungan kata ini dapat dilihat pada gambar **2.1**. *Word2vec* memiliki 2 teknik yaitu *continous bag of words* dan *skip gram model*.

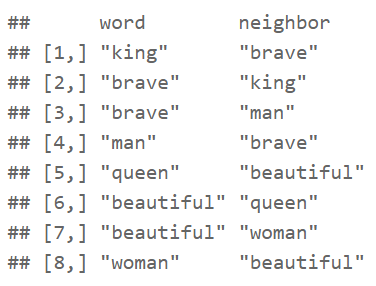


**Gambar 2.1** Hubungan antar kata *word2vec* (Abdullah, 2018)

**2.8.1 Continous bag of words**

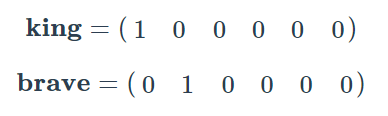
*Continous bag of words* (CBOW)merupakan salah satu arsitektur *word2vec* yang digunakan untuk memprediksi kata yang berhubungan dengan beberapa input konteks kata, kelebihan dari arsitektur CBOW adalah memiliki waktu *training* yang lebih cepat dan tingkat akurasi yang lebih baik untuk kata yang sering muncul. Misalnya terdapat kalimat “king brave man” dan “queen beautiful woman”, dengan *windows size* 1 maka akan menghasilkan **Gambar 2.2**. *windows size* merupakan jumlah tetangga terdekat yang digunakan untuk *training* model.

**Gambar 2.2** hasil *windows size* 1



Model yang ingin dibuat dari data training dengan input “king” dan “brave” hasilnya “man”. Alur kerja *continuous bag of words* (CBOW)untuk membuat model sebagai berikut (Abdullah, 2018):

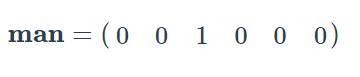
1. Konversikan *input layer* dan *output layer* (target) menjadi *one-hot vector* dengan ukuran [1 x *V*], dimana *V* merupakan jumlah kata hasil tokenisasi atau jumlah *vocabulary*. Pada contoh ini kata “king”, “brave” dan “man” dikonversikan *one-hot vector* menghasilkan *input layer* 2[1 x *V*] dengan *V* = 6 dan 2 adalah kata “king” dan “brave”, bentuk matriks *input layer* dapat dilihat pada **Gambar 2.3**.



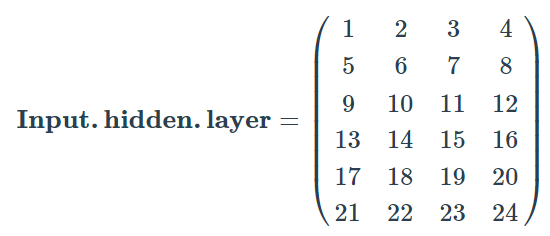
**Gambar 2.3** Matriks *input layer*

*Output layer* 1[1 x *V*] dimana 1 adalah kata “man”, bentuk matriks *output layer* dapat dilihat pada **Gambar 2.4**.

**Gambar 2.4** Matriks*ouput layer*

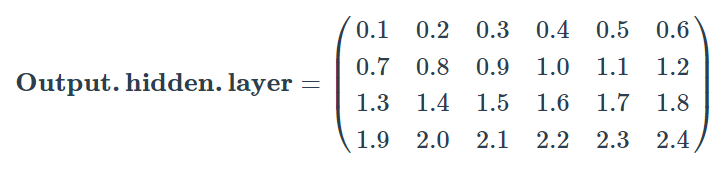


1. Langkah selanjutnya adalah pembobotan. Terdapat dua pembobotan yaitu antara *input layer* dengan *hidden layer* dan antara *hidden layer* dengan *output layer*. Matriks *input hidden layer* berukuran [*V* x *N*] sedangkan matriks *output hidden layer* berukuran [*N* x *V*], dimana nilai N merupakan jumlah *neuron*. Misalnya jumlah *neuron* 4 maka matriks *input hidden layer* yang terbentuk adalah [10 x 4] seperti **Gambar 2.5**.



**Gambar 2.5** *Input hidden layer*

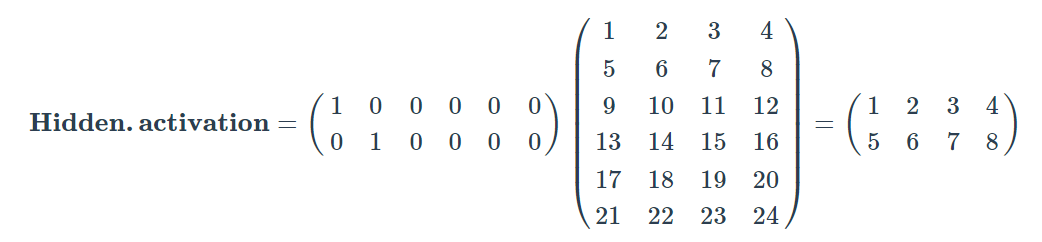
Sedangkan matriks *output hidden layer* yang terbentuk adalah [4 x 6], seperti **Gambar 2.6**.



**Gambar 2.6** *Output hidden layer*

1. Selanjutnya mengalikan *input layer* dengan *weight* dari *input hidden layer*. Hasil perkalian antara *input layer* dengan *weight* dinamakan *hidden activation*. Pada iterasi pertama, bobot yang diberikan adalah bobot acak atau random, dapat dilihat pada **Gambar 2.7**.

**Gambar 2.7** *Hidden activation*

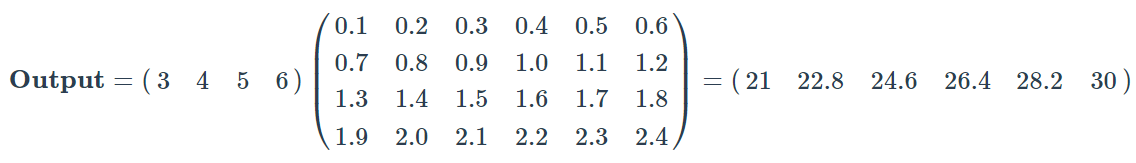


Kemudian dicari nilai rata-rata *hidden activation* maka menghasilkan matriks (3 4 5 6), dapat dilihat pada **Gambar 2.8**



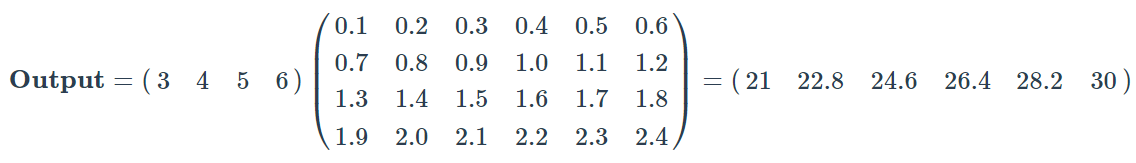
**Gambar 2.8** Rata rata *hidden activation*

1. Langkah selanjutnya, melakukan perkalian antara *hidden activation* dengan *weight* dari *hidden output*. Hasil perkalian ini menghasilkan *output*. Proses perkalian *hidden activation* dengan *weight* dapat dilihat pada **Gambar 2.9**.



**Gambar 2.9** *Output*

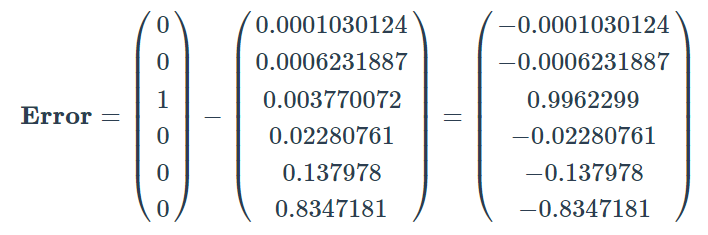
1. Setelah mendapatkan *output*, selanjutnya *output* tersebut akan ditransformasikan dengan nilai *softmax probabilities*. Hasil transformasi dapat dilihat pada **Gambar 2.10**.



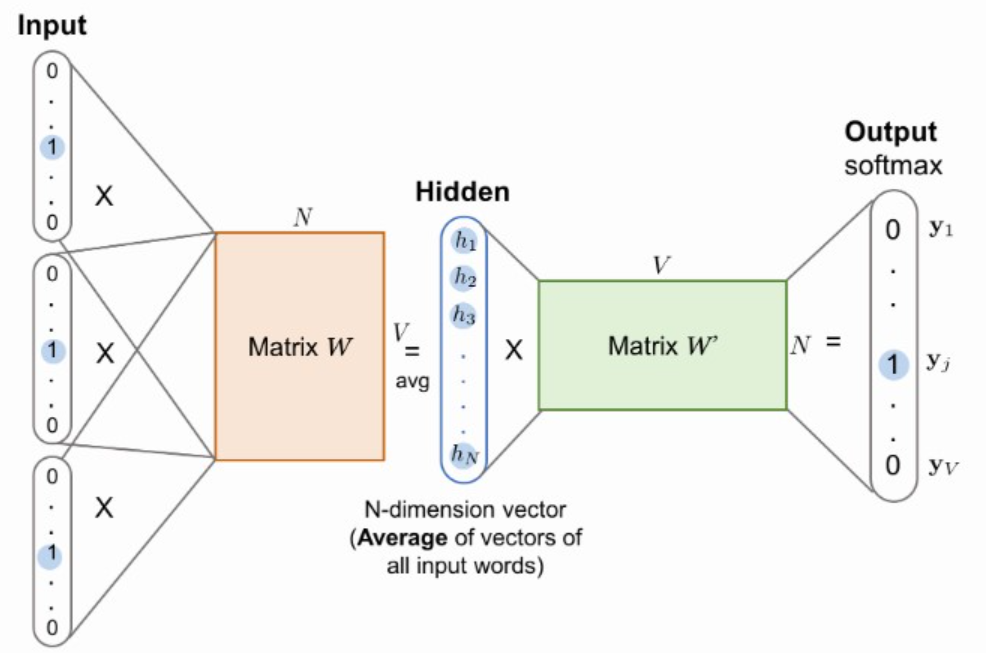
**Gambar 2.10** Hasil transformasi

1. Langkah selanjutnya, menghitung nilai *error* antara output dengan target kata, kemudian memperbaiki nilai dengan melakukan *backpropagation* untuk *re-adjust weight*nya. Cara menghitung nilai *error* dapat dilihat pada **Gambar 2.11**

**Gambar 2.11** Menghitung nilai *error*



Berdasarkan alur kerja *continuous bag of words* (CBOW) yang telah dijabarkan diatas maka alur kerja CBOW dapat dibuatkan ilustrasi pada **Gambar 2.12**



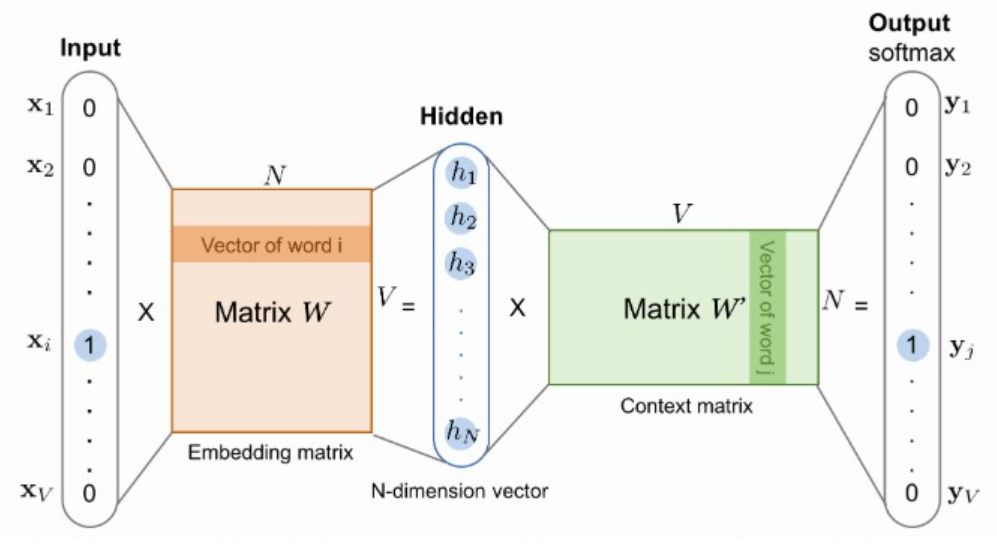
**Gambar 2.12** Ilustrasi alur kerja CBOW

**2.8.2 Skip gram model**

*Skip gram model* merupakan salah satu arsitektur *word2vec*, arsitektur ini merupakan kebalikan dari arsitektur CBOW, dimana arsitektur ini berguna untuk memprediksi konteks (output) yang berhubungan disekitar *current word* kata input, kelebihan arsitektur ini adalah dapat bekerja dengan baik meski data *training* hanya sedikit dan arsitektur ini mampu merepresentasikan kata yang jarang muncul atau langka. Dengan contoh yang sama seperti CBOW, yaitu “king” dan “brave”. Alur kerja *skip gram model* dapat dilihat sebagai berikut (Abdullah, 2018) :

1. Konversikan nilai *input* menjadi matriks berukuran [1 x *V*], matriks *input hidden weight* berukuran [*V* x *N*] dimana *N* merupakan jumlah *neuron* dalam *hidden layer* dan ukuran matriks *hidden output* C[1 x *V*].
2. Jumlah konteks kata dinamakan C
3. Kemudian menghitung *output* prediksi dengan cara melakukan perkalian antara matriks *hidden activation* dengan *weight* antara *hidden layer* dengan *output layer*.
4. Setelah mendapatkan nilai *ouput* dari proses sebelumnya, *ouput* tersebut ditransformasikan menggunakan fungsi *softmax* untuk mendapatkan nilai probabilitasnya.
5. Langkah selanjutnya, yaitu menghitung *error* dengan mencari selisih antara nilai *output* dengan data target. Kemudian dilakukan proses *backpropagation* untuk *re-adjust weight*nya.

Berdasarkan penjabaran alur kerja *skip gram model*, maka proses alur kerja *skip gram model* dapat diilustrasikan pada **Gambar 2.13**



**Gambar 2.13** Ilustrasi alur kerja *skip gram model*

**2.9 Glo-Ve**

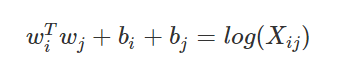
*Global vectors for word representation* (Glo-Ve)merupakan salah satu metode *word embedding* yang mengandalkan *co-occurrence* kata atau statistik kemunculan kata dalam kumpulan kata atau korpus yang ditangkap langsung oleh model untuk memperoleh hubungan semantik antar kata dalam korpus. Glo-Ve menggunakan metode *global matrix factorization* yang mewakili jumlah kemunculan atau frekuensi dalam suatu korpus (Pennington et al., 2014). Alur kerja Glo-ve terdiri dari langkah-langkah sebagai berikut (Selivanov, 2020) :

1. Bentuk matriks *word co-occurence* X dimana didalam matriks tersebut berisi kumpulan statistik *word co-occurence*. Matriks tersebut terdapat elemen Xij yang merepresentasikan jumlah kata *i* muncul dalam konteks kata *j*. setiap kata dicari kontaks yang berhubungan diantara sebelum *window size* dan diantara setelah *window size*. Penjabaran langkah ini dapat dilihat pada **Gambar 2.14**.



**Gambar 2.14** Penjabaran langkah pertama

1. Langkah selanjutnya menentukan *soft constraints* untuk setiap pasangan kata, rumus dapat dilihat pada **Gambar 2.15**.

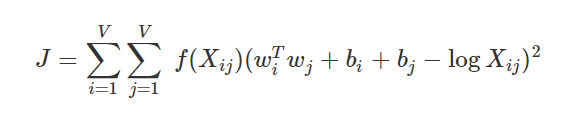


**Gambar 2.15** rumus *soft constraints*

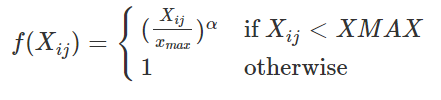
Dimana *wi* – vektor merupakan kata utama, *wj* – vektor merupakan kata konteks, *bi* merupakan skalar bias untuk kata-kata utama, *bj* merupakan skalar bias kata-kata konteks.

1. Langkah selanjutnya menentukan *cost function*, rumus *cost function* dapat dilihat pada **Gambar 2.16**.

**Gambar 2.16** rumus *cost function*



Dimana *f* adalah fungsi pembobotan yang berguna untuk mencegah pembelajaran hanya dari pasangan kata yang sangat umum. Rumus *f* dapat dilihat pada **Gambar 2.17**.



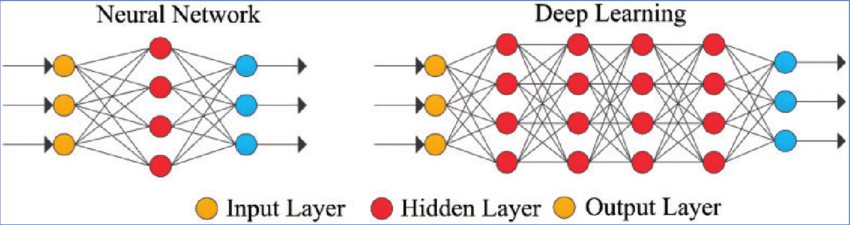
**Gambar 2.17** rumus *f*

**2.10 Fasttext**

*Fasttext* merupakan metode *word embedding* pengembangan dari *word2vec*, dimana metode ini mempelajari representasi kata dengan memperhatikan informasi *subword* dengan menggunakan *n-gram* ke dalam model skipgram. Hal inilah yang membuat *fasttext* dapat menangkap kata-kata yang lebih pendek dan memahami sufiks dan prefix dari kata (Bojanowski et al., 2017). Misalnya terdapat kata “cinta” dengan *n-gram*nya3 maka menjadi < ci, cin, int, nta, ta >. *Fasttext* memliki kelebihan yaitu dapat menangani masalah *out of vocabulary* dimana masalah ini tidak dapat diselesaikan oleh *word embedding word2vec* dan *Glo-Ve*, *fasttext* dapat menangani masalah tersebut karena ketika ada kata yang tidak muncul selama *training* model sehingga tidak ditemukan *embedding* vektornya, maka kata tersebut akan dipecah menggunakan *n-gram* untuk mendapatkan *embedding* vektornya. Sedangkan pada *word embedding* lain jika terdapat kata yang tidak muncul selama *training* model, maka akan menghasilkan *error*.

**2.11 Deep Learning**

*Deep learning* merupakan salah satu bidang *machine learning* berbasis jaringan syaraf tiruan atau *neural network*, *deep learning* terdiri dari beberapa *hidden layer* yang membentuk tumpukan (Nurfita dan Ariyanto, 2018). *Deep learning* membutuhkan waktu yang sedikit untuk melakukan *training* karena *deep learning* mampu menangani masalah gradien semakin rendah pada propagasi balik (Ahmad, 2017). Arsitektur *deep learning* dapat dilihat pada **Gambar 2.18**.



**Gambar 2.18** Arsitektur *deep learning* (Xing and Du, 2019)

*Deep learning* efektif dalam menangani berbagai permasalahan dengan data skala besar seperti *speech recognition*, *image recognition*, *natural language comprehension*, *playing games*, *biomedical applications* (Temizel et al., 2020). Hasil *training* pada *deep learning* berupa model yang digunakan untuk prediksi. *Deep learning* memiliki kelebihan yaitu, model yang dihasilkan dapat membedakan suatu kelas dengan mudah, hal ini karena *deep learning*  memiliki *feature engineering* yang berfungsi untuk mengekstrak pola yang penting dari suatu data. Contoh algoritma yang menerapkan *deep learning* seperti *Convolutional Neural Network* (CNN), *Recurrent Neural Network* (RNN), *Long Short Term Memory* (LSTM).

**2.12 Recurrent Neural Network**

*Recurrent Neural Network* (RNN) merupakan salah satu algoritma yang menerapkan *deep learning*, RNN memproses data sekuensial dengan mengatur proses *input* dan *output* sehingga menghasilkan *output* berdasarkan perhitungan sebelumnya (Patel & Tiwari, 2019). Pengaturan *input* dan *output* pada RNN berupa *output* pada *network* sebelumnyadigunakan kembali sebagai *input network* (Radjabaycolle et al., 2016), dengan kata lain informasi sebelumnya tetap dipertahankan untuk mendapatkan pola dengan jaringan *loop* (Rumelhart et al., 1986). RNN efisien untuk menangani masalah *Natural Language Processing* (NLP) menggunakan sel memori untuk menangkap informasi tentang urutan panjang (Patel & Tiwari, 2019). Arsitektur RNN dapat dilihat pada **Gambar 2.19**.

Data sekuensial memiliki karakteristik dimana pemprosesan sample dilakukan sesuai urutan tertentu seperti waktu, suatu sample memiliki hubungan erat satu sama lain dalam urutan, beberapa contoh kasus data sekensial sebagai berikut (Abdullah, 2019) :

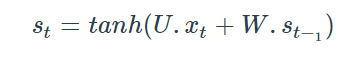
1. Rangkaian kata-kata pada klasifikasi sentimen, *input* berupa teks dan menghasilkan *output* kelas sentimen.
2. Rangkaian kata penerjemah bahasa, *input* berupa teks dan menghasilkan *output* berupa terjemahan bahasa.
3. Data runtun waktu (*time series*), *input* berupa data numerik dan menghasilkan *output* berupa *forecasting result*.
4. *Speech recognition*, input berupa audio dan menghasilkan *output* berupa teks.

**Gambar 2.19** Arsitektur RNN (Abdullah, 2019)

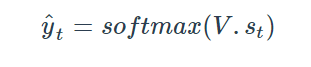


Pada **Gambar 2.19** *input layer* diwakilkan dengan simbol *x*, *hidden layer* diwakilkan dengan simbol *s*, dan *output layer* diwakilkan dengan simbol *y*. *Input* ke jaringan waktu *t* adalah *xt*, *output* dilambangkan sebagai *yt*, dan *st* adalah keadaan *hidden layer*. Pada saat *t* (berada ditengah), RNN menghitung *st* terlebih dahulu karena RNN memperhatikan nilai *output* pada proses sebelumnya (*t* – 1) (Abdullah, 2019). Cara menghitung *st* dengan cara mengalikan matriks *input xt* dengan parameter *U*, kemudian dijumlahkan dengan hasil perkalian *st-1* dengan parameter *W*. Hasil perhitungan tersebut kemudian diproses dengan fungsi aktivasi *tanh*, detail perhitungan *s­t*dapat dilihat pada **Gambar 2.20**.

**Gambar 2.20** Rumus *St*

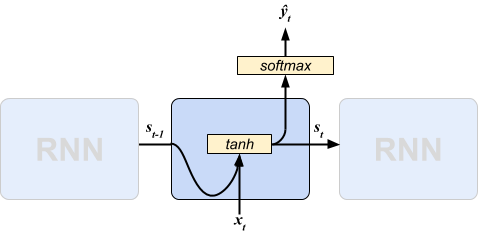


Setelah mendapatkan nilai *st*, kemudian menghitung *output yt* dengan mengalikan hasil *st* dengan parameter *V*. Hasil tersebut kemudian diproses menggunakan fungsi aktivasi *softmax*, detail perhitungan *yt* dapat dilihat pada **Gambar 2.21**.



**Gambar 2.21** Rumus *yt*

Proses menghitung *st* dan *yt* diilustrasikan pada **Gambar 2.22**



**Gambar 2.22** Ilustrasi proses *st* dan *yt* (Abdullah, 2019)

Untuk menemukan parameter *U*, *V*, *W* yang menghasilkan *error* minimum RNN menggunakan algoritma *Backpropagation Through Time* (BPTT), dimana menemukan parameter *U*, *V*, *W* yang menghasilkan *error* minimum merupakan tujuan dari *training* model RNN. BPTT memperhatikan deret waktu sebelumnya, sehingga untuk mendapatkan untuk menghitung gradien pada waktu *t*, maka turunan pada langkah *t* – 1, *t ­*– 2, *t* – 3, hingga pada saat *t* = 1 harus dihitung (Abdullah, 2019). Dengan demikian RNN mengingat informasi untuk beberapa langkah waktu dalam *hidden layer* oleh BTPP.

RNN memiliki kekurangan yang dinamakan *vanishing gradient*. Pada RNN, *vanishing gradient* terjadi karena nilai *gradient* bernilai 0 ataumendekati 0 seiring bertambahnya data yang dilatih RNN, karena nilai *gradient* digunakan untuk memperbaharui bobot sehingga tidak dapat memberi banyak kontribusi dan berhenti melakukan proses *learning* (Bengio et al., 1994). Hal ini disebabkan oleh sifat perkalian antar bilangan pecahan pada proses *training*. Misalnya bilangan pecahan dikalikan dengan bilangan pecahan maka hasilnya , hasil tadi dikalikan lagi dengan bilangan pecahan maka hasilnya menjadi , perkalian bilangan dilakukan sebanyak jumlah data sekuensial, nilai *gradient* akan mengecil secara ekponensial mendekati 0. Untuk menangani masalah ini muncul arsitektur pengembangan dari RNN yang dinamakan *Long Short Term Memory* (LSTM).

**2.13 Long Short Term Memory**

*Long Short Term Memory* (LSTM) merupakan algoritma *deep learning* pengembangan dari arsitektur RNN. RNN memiliki masalah *vanishing gradient*, LSTM mampu mengatasi masalah *vanishing gradient* dengan *memory cell* dan *gate units* (*input gate, forget gate, output gate*) sehingga LSTM dapat membaca, menyimpan, dan memperbaharui informasi (Rao & Spasojevic, 2016). Pada RNN hanya menggunakan satu *layer neuron* dengan fungsi aktivasi *tanh*, namun pada LSTM lebih kompleks dengan memiliki empat *layer*, hal inilah yang membuat LSTM dapat mengatasi masalah *vanishing gradient* (Olah, 2015). Arsitektur LSTM dapat dilihat pada **Gambar 2.23**.



**Gambar 2.23** Arsitektur LSTM (Olah, 2015)

Pada gambar 2.3 terdapat beberapa simbol yang mewakilkan proses LSTM, seperti persegi panjang berwarna kuning melambangkan *layer neuron*, lingkaran berwarna merah muda melambangkan operasielemen, panah berkelok berwarna hitam melambangkan aliran informasi. Untuk penjelasan lebih lanjut dapat dilihat pada **Gambar 2.24** dan **Gambar 2.25**.

**Gambar 2.24** Penjelasan simbol (Olah, 2015)





**Gambar 2.25** Penjelasan simbol lanjut

Pada LSTM blok memori dinamakan *cell*, *cell* inilah yang membangun struktur LSTM. LSTM memiliki dua jalur yang berfungsi untuk menyampaikan informasi ke *cell* selanjutnya, yaitu *cell state* dan *hidden state*. *Cell state* merupakan kunci utama LSTM, garis horizontal yang melewati bagian atas. Pada *cell state*,LSTM dapat menghapus atau menambah informasi dengan *gate*. *Gate* merupakan jalan agar informasi masuk, *gate* terdiri dari *layer sigmoid* dan operasi perkalian *pointwise* (Olah, 2015). Struktur *cell state* dapat dilihat pada **Gambar 2.26**.



**Gambar 2.26** Struktur *cell state* (Olah, 2015)

Seperti yang dijelaskan sebelumnya *gate* terdiri dari *layer sigmoid* dan operasi perkalian *pointwise*, *layer sigmoid* dapat dilihat pada **Gambar 2.27**.

**Gambar 2.27** *Layer sigmoid* pada LSTM(Olah, 2015)



*Layer sigmoid* menerapkan fungsi *sigmoid* yang menghasilkan *output* berupa antara angka 0 dan 1, dimana angka tersebut merepresentasikan jumlah dari setiap komponen yang harus dilewati. Nilai 0 berarti informasi tidak diteruskan, sedangkan nilai 1 berarti informasi diteruskan. LSTM memiliki tiga *gate* untuk melindungi dan mengontrol *cell*. Tiga *gate* tersebut yaitu *input gate*, *forget gate*, *output gate*. *Input gate* berfungsi untuk menentukan suatu *input* ditambahkan atau tidak kedalam *memory cell state*. *Forget gate* berfungsi untuk menentukan *memory cell* sebelumnya harus dilupakan atau tidak. *Output gate* berfungsi menentukan besarnya pengaruh *memory cell state* terhadap hasil prediksi yang akan dihasilkan.

Langkah pertama dalam LSTM adalah menentukan informasi yang akan dihapus oleh *cell state*. Langkah ini diatur oleh *layer sigmoid* yang dinamakan *forget gate*. Pada *forget gate* output dari langkah sebelumnya yaitu *ht-1* dan *xt* akan diproses menggunakan *fungsi sigmoid*, pada proses ini akan menghasilkan *output* berupa nilai antara angka 0 sampai 1 pada *cell state Ct-1*, angka 0 berarti informasi akan disingkirkan, sedangkan angka 1 berarti akan dipertahankan.Persamaan *forget gate* diuraikan pada persamaan 2.1, sedangkan alur informasi pada *forget gate* dapat dilihat pada **Gambar 2.28**

𝑓𝑡= 𝜎 (𝑥𝑡.𝑊𝑓+ℎ𝑡−1.𝑈𝑓+𝑏𝑓) . . .(2.1)

Keterangan :

*ft*  : *forget gate*

σ : fungsi *sigmoid*

*Wf* : *weight* pada *forget gate*

*Uf* : *recrrent weight* pada *forget gate*

*ht-1*: *output* pada langkah sebelum orde t

*xt* : nilai input pada orde t

*bf* : nilai bias pada *forget gate*



**Gambar 2.28** Alur informasi pada *forget gate* (Olah, 2015)

Langkah kedua adalah menentukan informasi yang akan disimpan pada *cell state*. Pada langkah ini memiliki memiliki dua bagian, yang pertama *layer sigmoid* dinamakan *input gate layer* berfungsi untuk menentukan nilai mana yang akan diperbaharui. Selanjutnya, *layer tanh* membuat vector nilai kandidat konteks, 𝐶̌𝑡untuk dimasukkan ke *cell state*. *Output* dari *input gate layer* dan *tanh layer* akan digabungkan untuk memperbaharui *cell state*. Proses pada langkah ini digambarkan pada **Gambar 2.28**. Sedangkan, untuk persamaan *input gate layer* dan persamaan nilai kandidat konteks dapat dilihat pada persamaan 2.2 dan persamaan 2.3



**Gambar 2.28** Proses *input gate layer* (Olah, 2015)

𝑖𝑡=𝜎 (𝑥𝑡.𝑊𝑖+ℎ𝑡−1𝑈𝑖+𝑏𝑖) . . . (2.2)

𝐶̌𝑡=𝑡𝑎𝑛ℎ (𝑥𝑡.𝑊𝑐+ℎ𝑡−1𝑈𝑐+𝑏𝑐) . . . (2.3)

Keterangan :

*it* : *input gate*

𝐶̌𝑡 : kandidat konteks

𝜎 : fungsi *sigmoid*

*Wi* : *weight* pada *input gate*

*Ui* : *recurrent weight* pada *input gate*

*bi* : nilai bias pada *input gate*

*Wc* : *weight* pada kandidat konteks

*Uc* : *recurrent weight* pada kandidat baru

*bc* : nilai bias pada kandidat konteks

*ht-1* : nilai *output* sebelum orde t

*xt* : nilai *input* pada orde t

Langkah ketiga adalah memperbaharui *cell state* lama *Ct-1* menjadi *cell state* baru *Ct*. Pada langkah sebelumnya, telah didapatkan nilai *input gate* dan kandidat konteks. Langkah ketiga akan terjadi proses perkalian *cell state* lama *Ct-1* dengan nilai *forget gate ft*, kemudian hasil perkalian *cell state* lama Ct-1 dengan nilai *forget gate* ditambah dengan hasil perkalian nilai *input gate* dengan nilai kandidat konteks 𝐶̌𝑡, akan menghasilkan *cell state* baru *Ct*. Proses ini diuraikan pada Gambar 2.29. Persamaan pada langkah mendapatkan *cell state* baru *Ct* dapat dilihat pada persamaan 2.4.



**Gambar 2.29** Langkah mendapatkan nilai *cell state* baru

𝐶𝑡= 𝑓𝑡 ∗ 𝐶𝑡−1+𝑖𝑡 ∗ 𝐶̌𝑡 . . . (2.4)

Keterangan :

*Ct* : *cell state* orde t

*ft* : *forget gate*

*Ct-1* : *cell state* sebelum orde t

*it* : nilai *input* pada orde t

𝐶̌𝑡 : kadidat konteks

Langkah terakhir menentukan *output* hasil keseluruhan proses. Pertama, *layer sigmoid* menentukan bagian dari *cell state* mana yang akan menjadi *output*. Selanjutnya nilai *cell state* *Ct*  diproses pada *layer tanh* yang bernilai antara 0 sampai 1, kemudian hasil nilai *Ct tanh* dikalikan dengan hasil nilai *Ct* sigmoid. Proses ini dijelaskan pada gambar 2.30. Persamaan proses ini dijabarkan pada persamaan 2.5 dan persamaan 2.6.



**Gambar 2.30** *Output gate*

𝑂𝑡= 𝜎 (𝑥𝑡 \* 𝑊𝑜+ℎ𝑡−1 \* 𝑈𝑜+𝑏𝑜) . . . (2.5)

ℎ𝑡= 𝑂𝑡 \* 𝑡𝑎𝑛ℎ(𝐶𝑡) . . . (2.6)

Keterangan :

*Ot* : *output gate*

𝜎 : fungsi *sigmoid*

*Wo* : nilai *weight* untuk *output gate*

*Uo*: *recurrent weight* pada *output gate*

*ht-1* : nilai *output* sebelum orde t

*xt* : nilai *input* orde t

*bo*: nilai bias pada *output gate*

*Ct* : *cell state*

Dari penjelasan proses LSTM, terdapat dua fungsi aktivasi yaitu, *sigmoid* dan *tanh*. Fungsi aktivasi *sigmoid* dijabarkan pada persamaan 2.7, sedangkan fungsi aktivasi *tanh* dijabarkan pada persamaan 2.8.

. . . (2.6)

. . . (2.7)

Keterangan :

𝜎 : fungsi *sigmoid*

x : data input

e : fungsi eksponensial

**2.14 Validasi dan Pengujian**

**2.15 Studi Pustaka**