BAB 2
TINJAUAN PUSTAKA

2.1 Kata dan Token

Menurut KBBI V Daring, kata adalah unsur bahasa yang diucapkan atau dituliskan yang merupakan perwujudan kesatuan perasaan dan pikiran yang dapat digunakan dalam berbahasa. Dari pengertian tersebut ada beberapa hal yang tidak bisa dianggap kata seperti tanda baca (contoh: koma). Istilah token merujuk pada seluruh kata, angka atau huruf yang (umumnya) dipisahkan oleh spasi pada sebuah teks. Oleh karena itu cakupan token lebih luas dibanding kata.

2.2 Kelas Kata

Menurut KBBI V Daring, kelas kata adalah golongan atau kategori dari sebuah kata yang didasarkan pada bentuk, fungsi atau maknanya. Artinya kelas kata mengandung informasi dari sebuah kata dan hal tersebut bisa dimanfaatkan untuk keperluan apapun termasuk oleh bidang *Natural Language Processing*.

2.3 POS Tagging

Kelas kata atau *part-of-speech* (POS) adalah sebuah kelas yang diberikan kepada sebuah kata. Pemberian part-of-speech didasarkan pada fungsi kata tersebut pada kalimat yang memuatnya. Oleh karena itu sebuah kata bisa memiliki kelas kata yang berbeda pada satu kalimat dengan kalimat lain. Contoh jika melihat KBBI V Daring maka bisa didapatkan beragam ambiguitas yang ada pada sebuah kata. "Abu-abu", misalnya, pada kalimat "*Abu-abu merupakan salah satu warna kegemaran masyarakat ketika membeli jaket sweater"* memiliki kelas kata *noun*. Sedangkan pada kalimat "*Sanski bagi penyedia TKI Bodong Masih Abu-abu"* kelas kata yang tepat adalah *adjective*. KBBI memberikan contoh lain, kata "Apa" pada kalimat "*ular apa ini?"* memiliki kelas kata *pronoun* dan pada kalimat "*apa besok ada ulangan*?" memiliki kelas kata *particle*. Kasus *POS Tagging* menjadi perhatian karena menjadi komponen yang membantu sistem lain seperti pada *information retrieval, information extraction* dan *speech processing* [15].

2.3.1 Penelitian Sebelumnya

Adapun Tabel 2.1 memperlihatkan ringkasan dari hasil studi literatur terhadap penelitian-penelitian yang sudah dilakukan pada kasus *POS Tagging* bahasa Indonesia.

Tabel 2.1 Hasil Penelitian Sebelumnya

Penelitian	Ukuran *Corpus*	Metode	Performa		Keterangan
			Metode	Hasil	
Wicaksono dan Purwarianti	15.000 token	HMM	Akurasi	91.30% (OOV 30%)	
				94.46% (OOV 21%)	
				96.50% (OOV 15%)	
Yuwana et. al	15.000 token	Max. Entropy	Akurasi (rata-rata)	**88.43%**	10-fold validation
		Unigram		86.20%	
		HMM		80.05%	
Amrullah et. al	1 juta token	Unigram	Akurasi	**88.37%**	
		HMM		62.69%	
Pisceldo et. al	14.000 token	Max. Entropy	Akurasi (rata-rata)	**85.02%**	10-fold validation
		CRF		82.89%	
	25.000 token	Max. Entropy		**97.57%**	
		CRF		91.15%	
Yuwana et. al	15.000 token	Deep Learning	Akurasi	**94.50%**	
		Max. Entropy		88.43%	
		HMM		80.05%	
Kurniawan dan Aji	250.000 token	bi-LSTM+CRF	F1 score	**97.47**	5-fold validation
		CRF		96.22	
Fu et. al	355.000 token	bi-LSTM+CRF	Akurasi (rata-rata)	**95.68%**	10-fold validation
		CRF		95.12%	

Penelitian terhadap *POS Tagging* bahasa Indonesia sudah dilakukan oleh beberapa penelitian. Penelitian yang penting untuk disoroti adalah penelitian Wicaksono dan Purwarianti [2] yang menggunakan metode HMM pada *corpus* berukuran 15.000 token dengan hasil akurasi sebesar 96.50%, 94.46% dan 91.30% dengan masing-masing terhadap data *testing* yang memiliki kadar OOV (*Out-of-Vocabulary*) sebesar 15%, 21% dan 30%. Hal ini menunjukkan performa metode yang diusulkan sensitif terhadap kadar *unseen word*. Fitur atau parameter yang digunakan oleh penelitian mereka hingga mendapatkan akurasi sebesar yang telah disebutkan adalah *Succedding POS Tag*, KBBI Katelog, Prefiks+Sufiks dan *Trigram*. Kemudian Yuwana et. al [16] mencoba membandingkan HMM dengan 5

metode lainnya pada corpus berukuran 15.000 token dengan validasi menggunakan *10-cross validation* menghasilkan kesimpulan rata-rata akurasi *Maximum Entropy* (88.43%) unggul dibanding *Unigram* (86.20%) yang unggul dibanding HMM (80.05%). Hasil ini kontras dengan penelitian Wicaksono dan Purwarianti. Tetapi perbedaan tersebut bisa disebabkan oleh perbedaan fitur yang digunakan oleh kedua penelitian dan perbedaan pengukuran performa. Namun kesimpulan kasar yang bisa diambil adalah metode *Max. Entropy* lebih unggul dibanding metode lainnya ketika diuji pada fitur dan *corpus* yang sama dengan metode lain. Penelitian Amrullah et. al [17] menggunakan 3 metode berbeda terhadap *corpus* berukuran 1 juta token dimana akurasi *Unigram* (88.37%) unggul dibanding HMM (62.69%). Hal ini sejalan dengan hasil penelitian Yuwana et. al yang mendapati *Unigram* unggul dibanding HMM. Sehingga bisa diambil kesimpulan dari penelitian-penelitian tersebut bahwa *Max. Entropy* lebih unggul dibanding metode lainnya dan *Unigram* lebih unggul dibanding HMM.

Penelitian Pisceldo et. al [18] menggunakan metode CRF dan *Max. Entropy* pada 2 corpus berbeda masing-masing berukuran ~14.000 token dan ~25.000 token menggunakan *10-fold validation* dimana akurasi *Max. Entropy* (85.02%) unggul dibanding CRF (82.89%) pada *corpus* I dan akurasi *Max. Entropy* (97.57%) sekali lagi unggul dibanding CRF (91.15%) pada *corpus* II. Hal ini sejalan dengan kesimpulan paragraf sebelumnya dimana *Max. Entropy* lebih unggul dibanding metode lainnya.

Penelitian Yuwana et. al [19] membandingkan 3 metode berbeda pada *corpus* berukuran 15.000 token dimana akurasi metode *Deep Learning* dengan Tensorflow dan Keras (94.50%) unggul dibanding *Max. Entropy* (88.43%) yang unggul dibanding HMM (80.05%). Hal ini menunjukkan performa algoritma *Deep Learning* bisa mengungguli metode-metode unggulan sebelumnya.

Penelitian Kurniawan dan Aji [5] menggunakan metode CRF dan bi-LSTM+CRF terhadap *corpus* berukuran 250.000 token dengan *5-fold validation* dimana F_1 score bi-LSTM+CRF (97.47) unggul dibanding CRF (96.22). Penelitian Fu et. al [4] menggunakan 3 metode berbeda pada *corpus* berukuran 355.000 token dengan *10-fold validation* dimaan rata-rata akurasi bi-LSTM+CRF (95.68%)

unggul dibanding CRF (95.12%) yang unggul dibanding *Seq2Seq* (94.14%). Hal tersebut sejalan dengan penelitian Kurniawan dan Aji yang mendapati bi-LSTM sebagai metode terunggul.

2.4 Tagset

Daftar *part-of-speech* atau kelas kata yang dikenalkan oleh Dionysius Thrax of Alexandria (dan mungkin dengan pihak lain yang membantu) memperkenalkan delapan *part-of-speech*: *noun, verb, pronoun, preposition, participle, conjunction, adverb, article*. Kelas kata ini kemudian menjadi dasar bagi *part-of-speech* Yunani, Latin dan mayoritas bahasa Eropa lainnya untuk 2000 tahun berikutnya. Daftar *part-of-speech* inilah yang disebut ***tagset*** pada bidang *Natural Language Processing*. Pada bidang NLP, satu penelitian dengan penelitian lain tak jarang menggunakan *tagset* berbeda. Pada *corpus Penn Treebank* misalnya menggunakan kelas kata dengan jumlah sebanyak 45 dibandingkan 8 kelas kata yang diperkenalkan Thrax. Di antara mereka ada yang menciptakan *tagset* baru dan ada juga yang menggunakan *tagset* dari penelitian sebelumnya.

2.5 Preprocessing

Praproses atau *preprocessing* adalah proses awal yang menjamin data masukan sesuai dengan standar sebelum dikonsumsi oleh algoritma utama. Adapun berikut penjelasan sebagian metode *preprocessing* penelitian ini yang perlu ditegaskan.

2.5.1 Case Folding

Adalah proses yang mengubah seluruh karakter menjadi huruf kecil atau besar. Hal ini bermanfaat bagi penerapan suatu algoritma agar tidak membedakan dua potongan teks yang sama hanya karena perbedaan huruf besar dan kecil.

2.5.2 Tokenisasi Kalimat

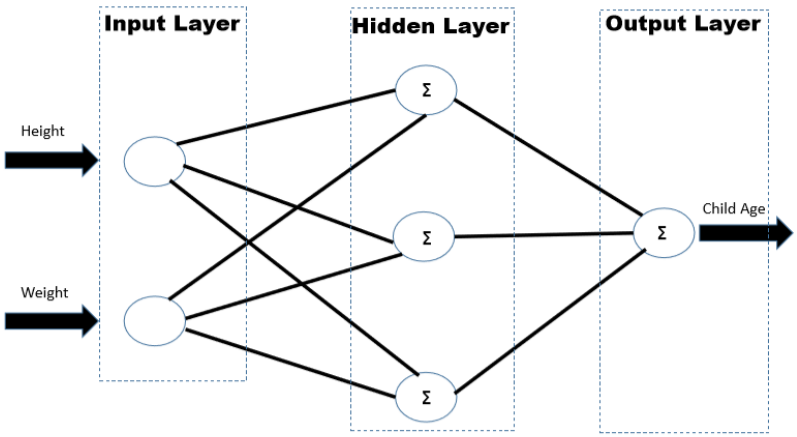
Adalah proses memecah sebuah dokumen atau rangkaian token menjadi kumpulan kalimat. Proses ini biasanya bermanfaat untuk menerapkan algoritma kecerdasan buatan terhadap kasus *sentence-level*.

2.5.3 One Hot Encoding

Adalah metode konversi dari teks menjadi angka. Biasanya angka ini adalah vektor, sedangkan teks yang dimaksud merujuk pada token. Konversi dilakukan mengingat algoritma *deep learning* tidak bisa memproses teks secara langsung. Ide metode ini adalah memberikan token sebuah ID kemudian setiap id diubah menjadi vektor. Sebagai contoh "abu-abu" memiliki id=1 dan "ada" memiliki id=2. Lalu masing-masing akan diubah menjadi vektor $\begin{bmatrix}1\\0\end{bmatrix}$ dan $\begin{bmatrix}0\\1\end{bmatrix}$ dimana elemen yang berangka 1 adalah elemen yang posisinya sama dengan ID sang token.

2.6 Elman Recurrent Neural Network

Artificial Neural Network (ANN) adalah algoritma yang mampu melakukan ekstraksi pola data meskipun pola tersebut bersifat kompleks. Algoritma ini biasa digunakan pada kasus klasifikasi dan prediksi. Contohnya diberikan data anak (tinggi dan berat) maka algoritma harus menjawab berapa umur mereka seperti yang ditunjukkan Gambar 2.3.

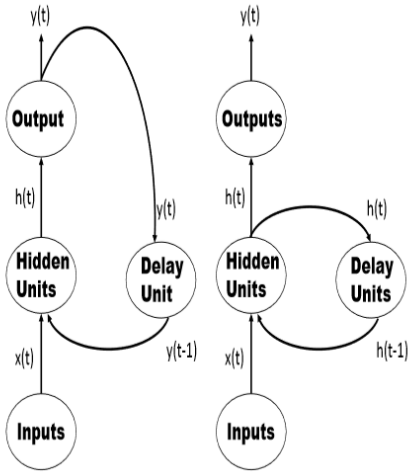


Gambar 2.3 ***Neural Network*** [20]

ANN memiliki banyak varian. *Feedforward Neural Network* adalah salah satunya. *Feedforward* umumnya terdiri dari layer input, layer hidden dan layer output, seperti yang ditunjukkan Gambar 2.3. layer input adalah tempat untuk memasukkan informasi yang akan diolah neural network. Informasi ini biasa disebut **fitur** oleh banyak penelitian. Fitur inilah yang menjadi patokan ketika melakukan prediksi. Jika melihat Gambar 2.3 maka tampak fitur yang digunakan ialah tinggi dan berat badan anak. Maka fitur dialirkan kepada layer hidden hingga akhirnya menuju layer output. Hasil dari layer output adalah sebuah prediksi. Pada Gambar 2.3 misalnya, layer output akan menghasilkan prediksi umur anak. Prediksi ini berbentuk probabilitas seperti yang terdapat pada pernyataan "80 persen yakin anak ini berumur 15 tahun dan 20 persen yakin anak ini bukan 15 tahun"—maka prediksi ANN adalah anak tersebut berumur 15 tahun karena ANN lebih yakin 15 ketimbang bukan 15.

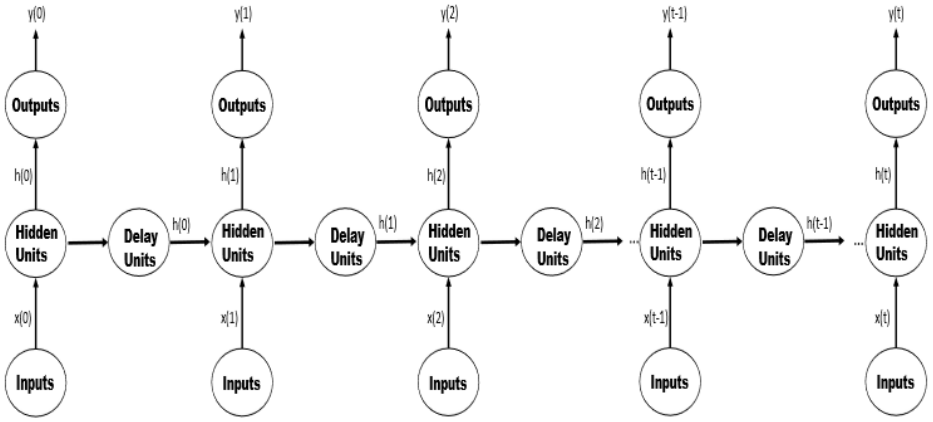
Kemampuan *feedforward neural network* sudah diuji oleh banyak penelitian dan performanya bisa dikatakan baik dan tak jarang melebihi performa algoritma lain. Sayangnya *feedforward neural network* berasumsi bahwa data bersifat *independent*. Asumsi ini membuat *feedforward* tidak tepat untuk digunakan pada kasus *time series prediction* seperti *sequence labelling*, termasuk *POS Tagging*. Ini karena **fitur konteks** sangat berperan besar bagi *sequence labelling*. Fitur konteks yang sering dipakai pada kasus *sequence labelling* mengisyaratkan bahwa data mereka tidaklah *independent*. Dengan kata lain, data bergantung oleh data lain (*dependent*). Contohnya pada kasus *POS Tagging*, untuk bisa mengetahui kelas kata sebenarnya (label) dari sebuah kata bisa didapatkan dengan memeriksa kata sebelumnya dan kata sesudahnya (fitur konteks). Meski begitu, *feedforward* tetap bisa dipakai untuk kasus *sequence labeling* namun dengan batasan. Ini ditunjukkan oleh penelitian sebelumnya yang menggabungkan fitur konteks (3 kata sebelumnya dan 2 kata sesudahnya) dengan fitur lain (kata saat ini). Fitur konteks oleh cara ini biasa disebut *windows context*. Kekurangan yang dimiliki oleh cara ini adalah fitur konteks dibatasi dan muncul parameter baru yang harus dipikirkan dan dicari, yaitu jumlah kata sebelum dan sudah. Oleh karena itu diperlukan algoritma yang bisa mengakomodasi kebutuhan yang dimiliki *time series prediction* seperti *sequence*

labeling. *Recurrent Neural Network* (RNN) adalah salah satu algoritma *neural network* yang didesain khusus untuk memenuhi kebutuhan tersebut.



Gambar 2.4 (Kiri) Jordan RNN dan (Kanan) Elman RNN [20]

Bentuk RNN mirip dengan bentuk yang dimiliki *feedforward*. Namun tidak seperti *feedforward* dimana aliran bersifat satu arah, aliran yang ada di RNN bisa berputar atau kembali ke layer sebelumnya, seperti yang ditunjukkan Gambar 2.4. Bentuk yang ditampilkan oleh Gambar 2.4 adalah penyederhanaan dari bentuk RNN sebenarnya. Bila dijabarkan maka bentuk utuhnya akan seperti yang ditampilkan oleh Gambar 2.5. Bentuk tersebut terdiri dari banyak input, hidden state dan output. Satu *time step* mengandung satu input, satu hidden state dan satu output. Jumlah *time step* tidak dibatasi dan membuat RNN bisa mengakomodasi kebutuhan *sequence labeling* dimana sebuah data sekuensial bisa dimasukkan seluruhnya secara langsung ke dalam RNN. Setiap hidden state terhubung dengan hidden state milik *time step* sebelumnya dan inilah yang menjadi sumber dari kelebihan yang dimiliki oleh RNN. Sambungan hidden state ini bisa mengalirkan informasi dari data sebelum-sebelumnya ke data selanjutnya sehingga pada setiap proses prediksi yang dilakukan, informasi mengenai masa lalu selalu dipertimbangkan, dan inilah informasi konteks yang dibutuhkan oleh *sequence labeling*.



Gambar 2.5 *Unfolding* Elman RNN [20]

Bobot yang dimiliki Elman RNN hanya ada 3: W_{ih}, W_{hh} dan W_{ho}. Untuk notasi penamaan bobot sengaja seperti itu untuk memudahkan dalam mengetahui letak dimana bobot berada. Contohnya W_{ih} yang mengandung index i dan h yang berarti bobot ini berada di antara input dan hidden state. Hal yang perlu dicatat adalah ketiga bobot ini digunakan bersama-sama oleh setiap *time step*.

Untuk proses perhitungan Elman dimulai dari proses perhitungan hidden state hingga output. Adapun rumus untuk menghitung hidden state dapat dilihat pada Persamaan (2.1).

$$s_t = f(W_{hh} \cdot s_{t-1} + W_{ih} \cdot x_t) \quad \textbf{(2.1)}$$

Dimana:

s_t : hidden state pada *time step* t.

f : fungsi aktifasi.

s_{t-1} : satu hidden state sebelumnya dari *time step* t.

x_t : vektor input.

W_{hh} : bobot yang berada di antara setiap hidden state.

W_{ih} : bobot yang berada di antara input dan hidden state.

Penelitian ini menggunakan tanh sebagai fungsi aktifasi f. Kemudian untuk perhitungan output dilakukan menggunakan Persamaan (2.2).

$$o_t = g(W_{ho} \cdot s_t) \quad \textbf{(2.2)}$$

Dimana:

o_t : output pada *time step* t.

g : fungsi aktifasi.

s_t : hidden state *time step* t.

W_{ho} : bobot yang berada di antara hidden state dan output.

Adapun penelitian ini akan menggunakan *softmax* sebagai fungsi aktifasi g pada layer output.

2.6.1 Training

Proses *training* adalah proses wajib yang ada pada banyak algoritma *machine learning*. Adapun alur proses *training* algoritma Elman adalah sebagai berikut.

1. Untuk satu *epoch* lakukan proses-proses berikut ini.
    a. Ambil satu data *training*.
    b. Lakukan proses *forward propagation* dengan menggunakan Persamaan (2.1) dan (2.2).
    c. Lakukan proses *backward propagation* dengan menggunakan Persamaan (2.4), (2.7), (2.8), (2.11) dan (2.12).
    d. Hitung bobot baru menggunakan nilai turunan hasil dari proses *backward*.
    e. Kembali ke langkah (1.a) hingga data *training* habis.
2. Lakukan langkah (1) kembali untuk memproses ke *epoch* berikutnya hingga jumlah *epoch* yang telah ditentukan.

Proses *training* dilakukan terhadap satu data sekuensial. Dengan terlebih dahulu melakukan proses *Forward Propagation* baru kemudian dilanjutkan ke proses *Backward Propagation (Backpropagation)* dan hasilnya akan digunakan oleh algoritma optimasi seperti *Gradient Descent* untuk mendapatkan bobot baru. Untuk

Elman digunakan *Backpropagation Through Time* (BPTT) yang sebenarnya mirip dengan algoritma *Backpropagation* biasa kecuali dilakukan terhadap *time step*.

Adapun *forward propagation* adalah proses perhitungan yang bertujuan untuk mendapatkan nilai s_t dan o_t menggunakan Persamaan (2.1) dan Persamaan (2.2). Hal ini dilakukan karena BPTT memerlukan nilai tersebut untuk bisa melakukan pekerjaannya.

Adapun *Backpropagation* atau BPTT adalah sebuah algoritma yang digunakan untuk menghitung nilai turunan terhadap *error* dengan menggunakan *chain rule*. Nilai turunan adalah komponen terpenting karena akan digunakan oleh algoritma optimasi seperti *Gradient Descent* sehingga dihasilkan bobot baru yang lebih optimal. Untuk menghitung BPTT maka perlu ditentukan terlebih dahulu fungsi *loss* atau *error* yang akan digunakan karena berbeda fungsi *loss* akan menghasilkan rumus BPTT yang berbeda pula. Untuk penelitian ini, fungsi *loss* yang digunakan adalah *cross-entropy*. Adapun rumusnya bisa dilihat pada Persamaan (2.3).

$$error = \sum_t E_t$$

Dengan: **(2.3)**

$$E_t = -\sum_i y_{ti} \cdot \log o_{ti}$$

Dimana:

y_t : label sebenarnya pada *time step* t (vektor).

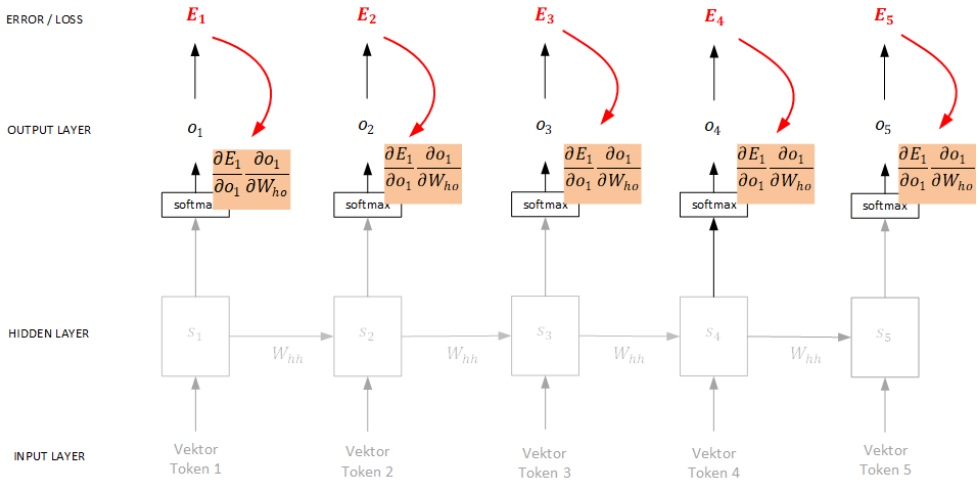
o_t : label prediksi pada *time step* t (vektor).

y_{ti} : elemen ke-i dari vektor y_t

o_{ti} : elemen ke-i dari vektor o_t

E_t : *error* dari output o_t

Untuk menurunkan bobot W_{ho} maka harus mempertimbangkan bahwa bobot ini dipakai oleh semua *time step* pada bagian outputnya seperti yang ditunjukkan Gambar 2.6.

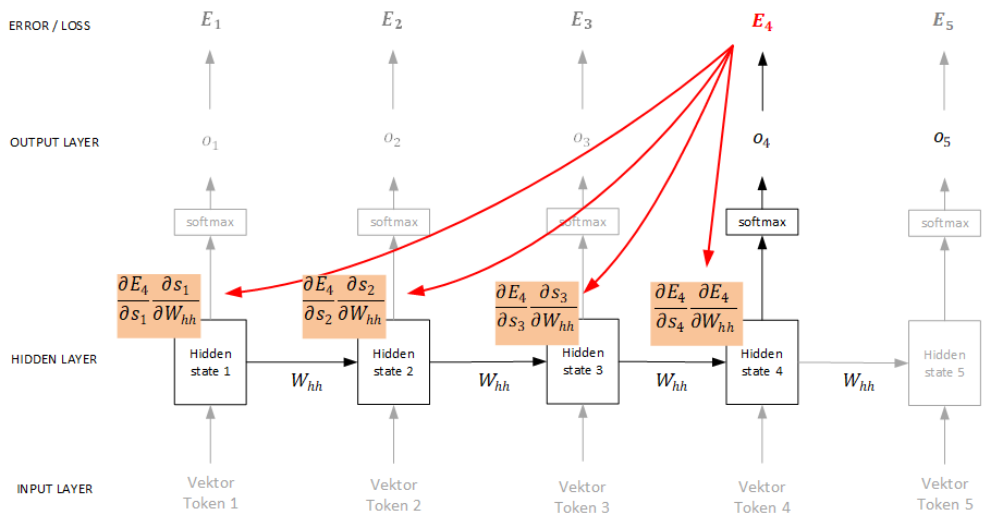


Gambar 2.6 Ilustrasi Penurunan Bobot W_{ho}

Adapun rumus BPTT untuk menghitung turunan W_{ho} terhadap *error* ditunjukkan oleh Persamaan (2.4).

$$\frac{\partial E}{\partial W_{ho}} = \sum_t \frac{\partial E_t}{\partial W_{ho}} = \sum_t (o_t - y_t) \otimes s_t \quad \textbf{(2.4)}$$

Sedangkan rumus BPTT untuk menghitung turunan W_{hh} terhadap *error* jauh lebih berbeda dengan W_{ho} mengingat bobot ini dipakai di semua hidden state maka perhitungan *chain rule* dimulai dari *error time step* t hingga hidden state *time step* 1 seperti yang ditunjukkan Gambar 2.7.



Gambar 2.7 Ilustrasi Penurunan Bobot W_{hh} terhadap E_4

Adapun rumus BPTT untuk menghitung turunan W_{hh} terhadap *error* ditunjukkan oleh Persamaan (2.5)

$$\frac{\partial E}{\partial W_{hh}} = \sum_t \frac{\partial E_t}{\partial W_{hh}} \tag{2.5}$$

Seperti yang ditunjukkan Gambar 2.7, maka untuk menghitung turunan *error* pada *time step* t terhadap bobot digunakan Persamaan (2.6).

$$\frac{\partial E_t}{\partial W_{hh}} = \left(\frac{\partial E_t}{\partial s_t}\frac{\partial s_t}{\partial W_{hh}}\right)\left(\frac{\partial E_t}{\partial s_{t-1}}\frac{\partial s_{t-1}}{\partial W_{hh}}\right)\left(\frac{\partial E_t}{\partial s_{t-2}}\frac{\partial s_{t-2}}{\partial W_{hh}}\right)\ldots\left(\frac{\partial E_t}{\partial s_1}\frac{\partial s_1}{\partial W_{hh}}\right) \tag{2.6}$$

Dimana rumus dari suku pertama persamaan di atas ditunjukkan oleh Persamaan (2.7).

$$\frac{\partial E_t}{\partial s_t}\frac{\partial s_t}{\partial W_{hh}} = \delta_T \otimes s_{t-1} \tag{2.7}$$

Dengan:

$$\delta_T = \left[W_{ho}^T.(o_T - y_T)\right] \odot (1 - {s_T}^2)$$

Sedangkan untuk suku kedua hingga terakhir, yaitu $\left(\frac{\partial E_t}{\partial s_{t-1}}\frac{\partial s_{t-1}}{\partial W_{hh}}\right)$ hingga $\left(\frac{\partial E_t}{\partial s_1}\frac{\partial s_1}{\partial W_{hh}}\right)$ rumusnya agak berbeda seperti yang ditunjukkan Persamaan (2.8).

$$\frac{\partial E_t}{\partial s_{t-b}}\frac{\partial s_{t-b}}{\partial W_{hh}} = \delta_t \otimes s_{t-1} \quad \textbf{(2.8)}$$

Dengan:

$$\delta_t = \left(W_{hh}^T . \delta_{t+1}\right) \odot (1 - {s_t}^2)$$

Untuk menghitung turunan bobot W_{ih} prosesnya persis sama dengan cara menurunkan bobot W_{hh} karena mereka sama-sama digunakan di setiap *time step*. Adapun rumus BPTT untuk menghitung turunan W_{ih} terhadap *error* ditunjukkan Persamaan (2.9).

$$\frac{\partial E}{\partial W_{ih}} = \sum_t \frac{\partial E_t}{\partial W_{ih}} \quad \textbf{(2.9)}$$

Dengan rumus turunan *error* pada *time step* t terhadap bobot ditunjukkan Persamaan (2.10).

$$\frac{\partial E_t}{\partial W_{ih}} = \left(\frac{\partial E_t}{\partial s_t}\frac{\partial s_t}{\partial W_{ih}}\right)\left(\frac{\partial E_t}{\partial s_{t-1}}\frac{\partial s_{t-1}}{\partial W_{ih}}\right)\left(\frac{\partial E_t}{\partial s_{t-2}}\frac{\partial s_{t-2}}{\partial W_{ih}}\right)\ldots\left(\frac{\partial E_t}{\partial s_1}\frac{\partial s_1}{\partial W_{ih}}\right) \quad \textbf{(2.10)}$$

Dimana rumus dari suku pertama persamaan di atas ditunjukkan oleh Persamaan (2.11).

$$\frac{\partial E_t}{\partial s_t}\frac{\partial s_t}{\partial W_{ih}} = \delta_T \otimes x_t \quad \textbf{(2.11)}$$

Dengan δ_T adalah δ_T yang sama yang digunakan ketika menurunkan bobot W_{hh}. Sehingga cukup menghitung ini sekali maka bisa digunakan baik oleh W_{hh} maupun W_{ih}.

Sedangkan untuk suku kedua hingga terakhir, yaitu $\left(\frac{\partial E_t}{\partial s_{t-1}}\frac{\partial s_{t-1}}{\partial W_{ih}}\right)$ hingga $\left(\frac{\partial E_t}{\partial s_1}\frac{\partial s_1}{\partial W_{ih}}\right)$ rumusnya ditunjukkan oleh Persamaan (2.12).

$$\frac{\partial E_t}{\partial s_{t-b}}\frac{\partial s_{t-b}}{\partial W_{ih}} = \delta_t \otimes x_t \tag{2.12}$$

Dengan δ_t adalah δ_t yang sama yang digunakan ketika menurunkan bobot W_{hh}. Sehingga cukup menghitung ini sekali maka bisa digunakan baik oleh W_{hh} maupun W_{ih}.

2.6.2 Testing

Pada tahap ini, bobot optimal sudah dimiliki oleh Elman RNN sehingga proses BPTT tidak diperlukan lagi. Oleh karena itu pada tahap *testing*, hal yang dilakukan hanya proses perhitungan *forward propagation*. Alur proses *testing* dari algoritma Elman adalah sebagai berikut.

1. Ambil satu data *testing*.
2. Lakukan proses *forward propagation* dengan menggunakan Persamaan (2.1) dan (2.2).
3. Kembali ke langkah (1.a) hingga data *testing* habis.

2.7 Minibatch Stochastic Gradient Descent

Algoritma *machine learning* dikatakan algoritma yang bagus apabila memberikan sedikit kesalahan (*error*). Untuk mengukur kesalahan ini, digunakan yang namanya *loss function*, seperti *cross-entropy*. Biasanya, apabila angka *loss function* ini kecil atau mendekati nol maka performa dikatakan bagus. Besar kecilnya nilai *loss* bergantung pada bobot—tentunya algoritma itu sendiri juga berperan. Maka pada setiap algoritma *machine learning* terdapat yang namanya fase pelatihan (*training*) yang bertujuan mencari bobot optimal, yaitu bobot yang memberikan *error* minimal. Terdapat beberapa algoritma pelatihan algoritma. Salah satunya adalah *Gradient Descent* (GD).

Gradient Descent—dikenal juga sebagai *Batch Gradient Descent*—memiliki varian yang dinamakan *Stochastic Gradient Descent*. Algoritma keluarga GD adalah pemeran utama dalam mencari bobot optimal. Cara kerja mereka adalah dengan melakukan banyak perulangan (*epoch*) dimana pada tiap-tiap *epoch* bobot diubah. Setelah diubah, bobot diharapkan memberikan *error* lebih kecil dibandingkan bobot sebelumnya. Adapun rumus perubahan bobot tersebut ditampilkan oleh Persamaan (2.13).

$$bobot_{baru} = bobot_{lama} - \left(learningrate\ \frac{\partial(error)}{\partial(bobot_{lama})}\right) \quad \textbf{(2.13)}$$

Dimana:

Learning rate	:	Nilai yang ditentukan manual, menginsyaratkan jumlah langkah perubahan. Perlu kehati-hatian untuk menentukan nilai ini agar tidak gagal konvergen.
$\frac{\partial(error)}{\partial(bobot)}$	:	Turunan *error* terhadap bobot, memberikan arah perubahan apakah bobot harus dikurangi atau ditambah. Turunan ini biasanya dicari oleh algoritma *Backpropagation.*

Setelah banyak *epoch* dilakukan, biasanya bobot optimal—atau setidaknya yang mendekati—sudah bisa didapatkan. Perbedaan *Batch* GD dan SGD hanya sedikit. Perbedaan itu ditunjukkan oleh nama depan mereka. *Batch* GD menggunakan seluruh data untuk mendapatkan bobot baru—menggunakan persamaan (2.13) di atas. Sedangkan *Stochastic* GD menggunakan satu buah data untuk mendapatkan bobot baru. Jika data yang digunakan adalah beberapa atau sebagian maka didapatkan varian GD baru yang dikenal sebagai *Minibatch Stochastic Gradient Descent*.

2.8 Pengujian Performa

Pada bidang NLP, terdapat beberapa metode yang digunakan untuk mengukur performa sebuah algoritma. Setiap metode pengukur performa dipilih berdasarkan karakteristiknya terhadap kasus yang sedang ditangani. Berikut adalah penjelasan dari metode pengukur performa yang digunakan penelitian ini.

2.8.1 Akurasi

Adalah salah satu metode pengukuran performa algoritma untuk kasus klasifikasi. Metode ini memberikan informasi mengenai jumlah persentase keberhasilan sebuah algoritma melakukan prediksi secara benar. Adapun persamaan dari akurasi bisa dilihat pada Persamaan (2.14) [21].

$$Akurasi = \frac{Jumlah\ Label\ Benar}{Total\ Label} \tag{2.14}$$

2.8.2 F1 Score

Akurasi adalah salah satu metode pengukuran performa. Kelemahan akurasi adalah tidak cocok untuk dataset yang tidak seimbang, seperti *dataset* Dinakaramani yang mengandung banyak *Noun*. Di sisi lain, F_1 *score* sering kali digunakan ketika *dataset* tidak seimbang. Perhitungan F_1 *score* dilakukan per kelas kemudian ambil nilai keseluruhan dengan cara di ratakan (*averaged*). F_1 *score* memiliki beberapa versi, yaitu *micro*, *macro* dan *weighted macro* F_1 *score*. Pada penelitian ini versi yang digunakan adalah *weighted macro* F_1 *score* yang persamaannya bisa dilihat pada Persamaan (2.15) ini [22] [23].

$$weighted\ F_1\ score = \frac{1}{\sum_i |X_i|} \times \sum_i |X_i|\ F_1\ score(X_i) \tag{2.15}$$

Dimana:

$|X_i|$: jumlah anggota kelas X_i

Dengan:

$$recall = \frac{Jumlah\ Prediksi\ Benar\ Kelas\ X}{Total\ Anggota\ Kelas\ X} \quad \textbf{(2.16)}$$

$$precision = \frac{Jumlah\ Prediksi\ Benar\ Kelas\ X}{Total\ Prediksi\ terhadap\ Kelas\ X} \quad \textbf{(2.17)}$$

$$F_1\ score(X) = \left(2 \times \frac{(precision\ \times recall)}{(precision + recall)}\right) \quad \textbf{(2.18)}$$

2.9 Operasi Matriks

2.9.1 Outer Product

Adalah perkalian dua vektor berukuran n dan m yang menghasilkan matriks $n\ x\ m$. Adapun *outer product* didefinisikan oleh Persamaan (2.19).

$$a \otimes b = ab^T = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} [b_1 \quad b_2 \quad b_3] = \begin{bmatrix} a_1b_1 & a_1b_2 & a_1b_3 \\ a_2b_1 & a_2b_2 & a_2b_3 \\ a_3b_1 & a_3b_2 & a_3b_3 \\ a_4b_1 & a_4b_2 & a_4b_3 \end{bmatrix} \quad \textbf{(2.19)}$$

2.9.2 Hadamard Product

Adalah perkalian dua matriks berdimensi sama yang menghasilkan matriks dengan dimensi yang sama pula. Perkalian matriks A dan matriks B dengan *hadamard product* menghasilkan matriks baru $A \odot B$ dimana elemen-elemennya didefinisikan oleh Persamaan (2.20).

$$(A \odot B)_{ij} = (A)_{ij}(B)_{ij} \quad \textbf{(2.20)}$$

2.10 Pemodelan Sistem

Pemodelan sistem merupakan proses yang mengembangkan gambaran abstrak mengenai sistem yang akan dibangun. Adapun berikut penjelasan pemodalan sistem yang digunakan oleh penelitian ini:

2.10.1 Flowchart

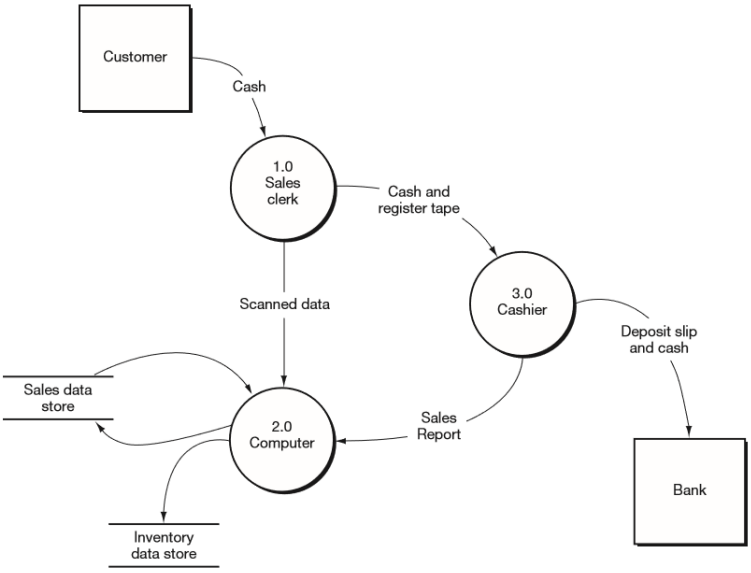
Flowchart merupakan gambaran mengenai proses atau algoritma yang ada di dalam sistem. *Flowchart* menggambarkan solusi dari sebuah masalah. Aliran *flowchart* bersifat atas ke bawah dan kiri ke kanan.

2.10.2 Diagram Konteks

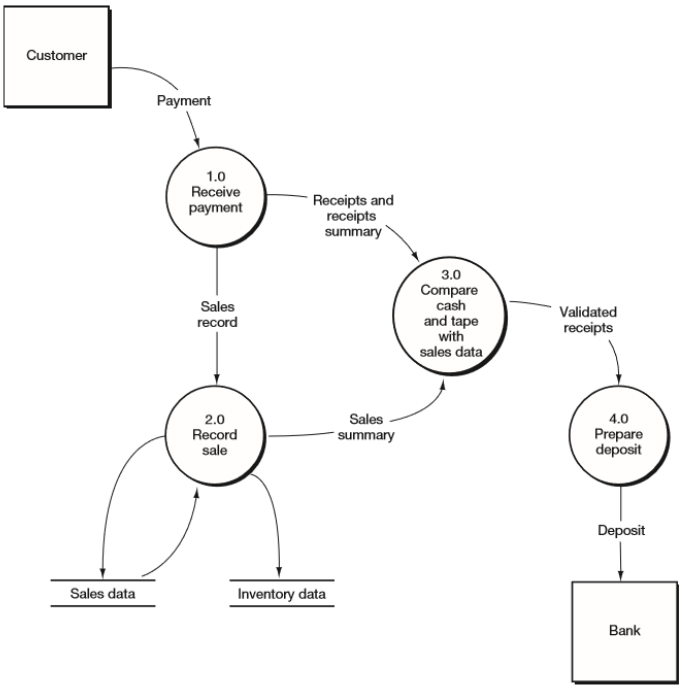
Diagram konteks adalah level pertama dari DFD yang menampilkan gambaran tentang sebuah sistem informasi. Diagram konteks hanya menampilkan aliran data dari entitas luar ke sistem atau dari sistem ke entitas luar. Entitas di luar disini bisa orang, tempat, benda yang mengirim data atau menerima data dari sistem [24]. Simbol yang digunakan sama dengan yang ada di DFD kecuali tanpa simbol *data store*.

2.10.3 Data Flow Diagram

Data Flow Diagram (DFD) menampilkan gambaran mengenai sistem. DFD memperlihatkan komponen-komponen sistem. Juga menampilkan aliran data antara komponen dengan *sources*, *destinations* dan *data storage* [24]. Ada dua jenis DFD, yaitu *Pyshical Data Flow Diagram* dan *Logical Data Flow Diagram*. Dimana *Pyshical* menampilkan entitas internal sistem dan entitas luar serta aliran data di antara mereka. Pada *Pyshical*, entitas internal didefinisikan sebagai entitas baik orang, tempat atau benda di dalam sistem yang melakukan operasi terhadap data. Sehingga *Pyshical* memperlihatkan bagaimana atau dimana atau oleh siapa sebuah proses di dalam sistem dikerjakan. Contoh DFD jenis ini bisa dilihat pada Gambar 2.8. Kemudian jenis lain DFD yaitu *Logical* menampilkan proses yang ada di sistem dan aliran data antara proses. *Logical* menekankan pada kebutuhan fungsional yang akan dilakukan oleh sistem. Contoh DFD jenis ini bisa dilihat pada Gambar 2.9.



Gambar 2.8 Contoh *Physical* DFD [24]



Gambar 2.9 Contoh *Logical* DFD [24]

2.11 Bahasa Pemrograman

Python adalah bahasa pemrograman yang telah menjadi *lingua franca* dalam bidang *data science*. Hal tersebut salah satunya difaktori oleh banyaknya pustaka (*library*) yang tersedia bagi keperluan *data science* seperti *data loading*, *visualization*, *statistics*, *natural language preprocessing*, *image processing*, *machine learning algorithms* dan masih banyak lagi. Selain itu didukung oleh banyak perkakas seperti Jupyter Notebook yang memudahkan *prototyping* ketika membangun algoritma *machine learning* secara interaktif [25].