PETIR: Jurnal Pengkajian dan Penerapan Teknik Informatika
Vol. 14, No. 2, September 2021, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v14i2.1164

Perbandingan *Pre-trained Word Embedding* dan *Embedding Layer* untuk *Named-Entity Recognition* Bahasa Indonesia

Meredita Susanty[1]; Sahrul[2]

[1, 2] Program Studi Ilmu Komputer, Fakultas Sains dan Komputer, Universitas Pertamina
[1] meredita.susanty@universitaspertamina.ac.id

ABSTRACT

Named-Entity Recognition (NER) is used to extract information from text by identifying entities such as the name of the person, organization, location, time, and other entities. Recently, machine learning approaches, particularly deep-learning, are widely used to recognize patterns of entities in sentences. Embedding, a process to convert text data into a number or vector of numbers, translates high dimensional vectors into relatively low-dimensional space. Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words. The embedding process can be performed using the supervised learning method, which requires a large number of labeled data sets or an unsupervised learning approach. This study compares the two embedding methods; trainable embedding layer (supervised learning) and pre-trained word embedding (unsupervised learning). The trainable embedding layer uses the embedding layer provided by the Keras library while pre-trained word embedding uses word2vec, GloVe, and fastText to build NER using the BiLSTM architecture. The results show that GloVe had better performance than other embedding techniques with a micro average f1 score of 76.48.

Keywords: *Natural Language Processing, Named-Entity Recognition, Word Embedding*

ABSTRAK

Named-Entity Recognition (NER) digunakan untuk mengekstrak informasi dari teks dengan cara mengidentifikasi entitas seperti nama orang, organisasi, lokasi, waktu, dan entitas lainnya. Belakangan ini pendekatan machine learning khususnya deep learning banyak digunakan untuk mengenali pola entitas dalam kalimat. Sebelum diolah dengan model machine learning, data teks harus terlebih dahulu diubah ke dalam bentuk angka atau vector angka yang disebut dengan embedding. Hasil embedding tersebut digunakan dalam pemodelan machine learning dan berfungsi sebagai dasar untuk komputasi natural language processing. Proses embedding ini bisa dilakukan dengan menggunakan metode supervised learning yang membutuhkan data set yang sudah berlabel dalam jumlah besar atau pendekatan unsupervised learning. Penelitian ini membandingkan kedua metode embedding tersebut; trainable embedding layer yang merupakan supervised learning dengan menggunakan embedding layer yang disediakan oleh pustaka Keras dan pre-trained word embedding yang merupakan pendekatan unsupervised learning menggunakan word2vec, GloVe dan fastText untuk membangun NER menggunakan arsitektur BiLSTM. Hasil penelitian menunjukkan GloVe memiliki performa yang lebih baik dibanding teknik embedding lainnya dengan nilai f1 score rataan mikro 76,48.

Kata kunci: *Natural Language Processing, Named-Entity Recognition, Word Embedding*

Vol. 14, No. 2, September 2021, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v14i2.1164

1. PENDAHULUAN

Named-Entity Recognition (NER) merupakan sub-bagian dari riset *Natural Language Processing* (NLP) yang termasuk dalam bidang *Artificial Intelligence* (AI). NER digunakan untuk mengekstrak informasi dari teks dengan cara mengidentifikasi entitas seperti nama orang, organisasi, lokasi, waktu, dan entitas lainnya [1]. Pendekatan *Machine Learning* (ML) dan *Deep Learning* (DL) yang saat ini banyak digunakan belakangan ini untuk mengenali pola entitas dalam kalimat [2], [3] mengatasi keterbatasan pada metode *rule-based* yang menggunakaan aturan yang telah didefinisikan berdasarkan pengetahuan linguistik dengan analisis yang dilakukan pada tingkatan sintaksis dan semantik [4].

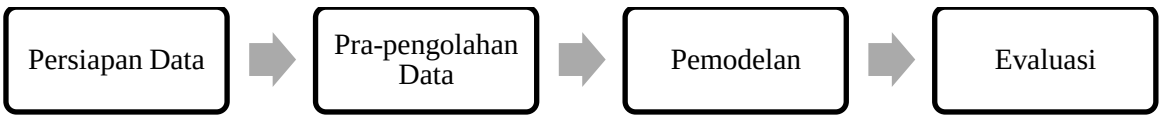
NER telah banyak digunakan pada data teks dengan menggunakan metode *supervised learning* untuk mengklasifikasikan NEs sesuai domain penelitian yang dilakukan [5], [6]. Pelatihan model *supervised learning* memerlukan data set yang dilengkapi dengan label. Terdapat beberapa pendekatan yang dapat digunakan untuk melabeli data NER, dua diantaranya adalah skema pendekatan yang digunakan pada MUC-6 dan CoNLL 2009 [7]. Secara umum, teknik pelabelan pada NER disebut dengan *sequence labeling* dengan memberi label setiap NEs dalam kalimat atau frasa tertentu [8].

Sebelum diolah dengan model ML, data teks harus terlebih dahulu diubah ke dalam bentuk angka. Proses pengubahan data teks ke dalam bentuk vektor angka disebut dengan *embedding*. Hasil *embedding* tersebut digunakan dalam pemodelan ML dan berfungsi sebagai dasar untuk komputasi NLP. Proses *embedding* ini bisa dilakukan dengan menggunakan metode *supervised learning* yang membutuhkan dataset yang sudah berlabel dalam jumlah besar atau pendekatan *unsupervised learning* [9]. Penelitian ini bertujuan membandingkan kedua metode *embedding* tersebut. Ada empat metode *embedding* yang diuji pada penelitian ini, yaitu *trainable embedding layer* yang disediakan oleh pustaka Keras[1] dan *pre-trained word embedding* menggunakan word2vec, GloVe dan fastText. *Trainable embedding layer* merupakan *supervised learning* yang menggunakan nilai bobot yang diinisiasi secara acak dan kemudian diperbaharui selama proses pelatihan model dengan menggunakan algoritma *back-propagation*, sedangkan *word embedding* merupakan jenis *unsupervised learning* yang didasarkan pada intuisi bahwa kata-kata yang sering muncul bersamaan memiliki konteks serupa sehingga diletakkan berdekatan satu sama lain dalam *abstract space* yang terdiri dari N dimensi [10], [11].

Dalam penelitian ini NER dibangun dengan pendekatan DL menggunakan arsitektur model BiLSTM, dengan lapisan *embedding* sebagai lapisan pertama pada model ini, Keempat metode *embedding* dibandingkan untuk mendapatkan rekomendasi metode yang paling baik digunakan untuk membangun NER dalam Bahasa Indonesia.

2. METODE/PERANCANGAN PENELITIAN

Penelitian ini mengikuti tahapan yang dijelaskan pada gambar 1 yang terdiri dari pengumpulan dan persiapan data yang digunakan, pra-pengolahan data, pemodelan yang dibangun dengan menggunakan teknik *deep learning*, serta validasi dan pengujian terhadap model yang dibuat.



Gambar 1. Metodologi Penelitian

[1] https://keras.io/

Vol. 14, No. 2, September 2021, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v14i2.1164

Data set yang digunakan pada penelitian ini adalah data berupa kumpulan kalimat yang telah diberi label dan diperoleh dari sumber *data set* publik dari dua repositori di github, yakni repositori nlpexperiments[2] dan indonesian-ner[3]. *Data set* tersebut menggunakan format pelabelan MUC-6 [12] dan tag XML. Format pelabelan data set tersebut diubah dari format MUC-6 [12] dan tag XML menjadi format BILOU. BILOU merupakan singkatan dari ***B****eginning*, ***I****nside*, dan ***L****ast* token untuk NEs *multi-token*, serta ***O****utside* dan ***U****nit* untuk NEs *single-token*. Format pelabelan BILOU secara signifikan mengungguli BIO yang telah digunakan secara luas [7].

Data dalam bentuk kumpulan tag NEs yang diperoleh kemudian diolah agar data tersebut dapat diproses dengan model yang menggunakan arsitektur LSTM. Tujuan utama pada tahap ini adalah mengubah data ke dalam bentuk pelabelan sekuensial. Langkah-langkah yang dilakukan adalah sebagai berikut.

1. Seluruh token dan label diberi indeks.
2. Menggabungkan setiap token beserta labelnya ke dalam sebuah *sequence* sama yang akan digunakan sebagai observasi data. Misal kalimat "Jokowi adalah presiden Indonesia." Diubah menjadi [[("Jokowi", "U-PERSON"), ("adalah", "O"), ("presiden", "O"), ("Indonesia", "U-LOCATION"), (".", "O"), (...)], ...]
3. Token dan label tersebut kemudian dipisah untuk membentuk data masukan X dan luaran y.
4. Setiap observasi data masukan X harus memiliki panjang *sequence* yang sama sehingga ditambahkan *padding* dengan kata "PADDING" dan diberi indeks terakhir. Label *padding* yang ditambahkan adalah O.
5. Luaran y kemudian diubah ke dalam bentuk *one-hot-encoding*.
6. Dimensi X dan y masing-masing adalah (jumlah seluruh observasi data, panjang *sequence*) dan (jumlah observasi data, panjang *sequence*, jumlah kelas).
7. Langkah terakhir adalah membagi data ke dalam data latih, data validasi, dan data uji.

Model NER dibangun menggunakan arsitektur BiLSTM. BiLSTM dipilih karena telah digunakan di beberapa penelitian terdahulu untuk pelabelan *sequence* dan memiliki performa yang baik [2], [3], [13]. Tidak seperti penelitian terdahulu [2], [9], penelitian ini hanya menggunakan satu lapisan konvolusi dan satu jenis ukuran kernel dan tidak menggunakan *character-level embedding* untuk menjaga agar model tetap sederhana dengan waktu pemrosesan yang rendah.

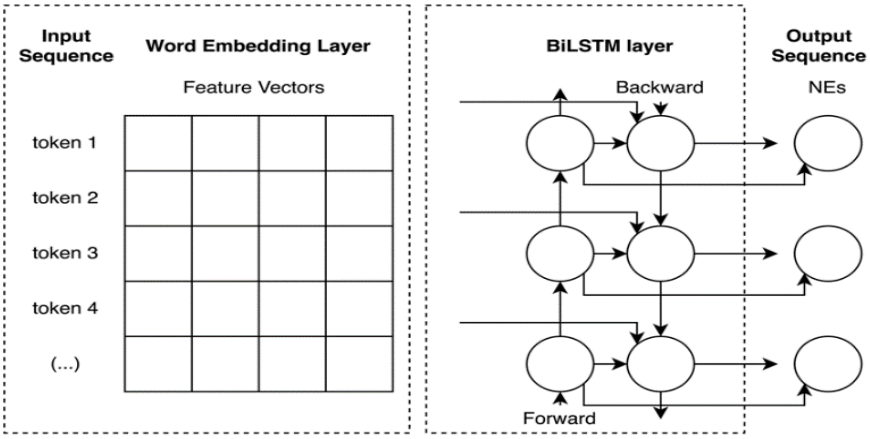
Performa model diukur dengan menggunakan metrik akurasi *f1 score* selama proses pelatihan dan pengujian untuk setiap label. Metrik *f1 score* adalah metrik *fscore* dengan nilai $\beta=1$. Performa setiap model secara keseluruhan diukur menggunakan rataan mikro. Pemilihan rataan mikro pada penelitian ini didasarkan pada kemungkinan distribusi label pada data yang digunakan tidak seimbang, sehingga metrik *f1 score* yang diperoleh tetap mempertimbangkan distribusi setiap kelas. Semakin tinggi nilai metrik evaluasi baik untuk setiap label maupun nilai rataan secara keseluruhan berarti semakin tinggi performa model tersebut. Label O (termasuk *padding*) dihilangkan pada saat evaluasi performa model.

[2] https://github.com/yohanesgultom/nlp-experiments
[3] https://github.com/yusufsyaifudin/indonesia-ner

PETIR: Jurnal Pengkajian dan Penerapan Teknik Informatika
Vol. 14, No. 2, September 2021, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v14i2.1164

2.1 Arsitektur

Model yang dibangun setidaknya terdiri dari 2 bagian utama seperti yang ditunjukkan pada Gambar 2. Masukan pada model adalah *sequence* X dengan dimensi sesuai dengan ukuran *mini batch* setiap iterasi dan panjang *sequence*. *Padding* pada sequence adalah *padding* yang ditambahkan saat pra-pemrosesan data agar seluruh observasi data memiliki panjang *sequence* yang sama.



Gambar 2. Arsitektur model BiLSTM yang digunakan

Lapisan pertama pada model tersebut adalah lapisan *embedding* yang digunakan untuk mengubah masukan *sequence* ke dalam bentuk vektor berdimensi-n. Ada dua jenis metode *embedding* yang diuji, yaitu *trainable embedding layer* dan *pre-trained word embedding*. Kedua metode tersebut memiliki fungsi yang sama, mengubah data teks (kumpulan kata atau token) ke dalam representasi angka yang menyimpan hubungan semantik antar kata. Setiap baris pada vektor tersebut merupakan representasi dari setiap token dengan panjang n. Luaran dari lapisan *word-level embbeding* adalah vektor dengan dimensi yang sama dengan panjang *sequence* x n yang merupakan representasi baru dari masukan *sequence*.

Luaran dari lapisan *embedding* selanjutnya digunakan sebagai masukan pada lapisan BiLSTM dengan *forward* dan *backward* LSTM. Lapisan ini merupakan lapisan inti dari model yang dibangun untuk menentukan luaran NEs dari data. *Output sequence* yang merupakan label setiap token hasil prediksi model diperoleh dari penambahan luaran *forward* dan *backward* LSTM.

Selama proses pelatihan digunakan *softmax loss* atau *categorical cross entropy* sebagai *loss function* pada model. Pelatihan dilakukan dengan maksimal jumlah *epoch* tertentu, tetapi dihentikan apabila selama beberapa *epoch* nilai *loss function* pada model tidak mengalami penurunan. Hal tersebut dilakukan untuk menghindari *overfit* pada model terhadap data latih.

Seluruh model yang dibangun pada penelitian ini menggunakan bahasa pemrograman Python dan pustaka Tensorflow[4] dengan antarmuka Keras. Proses optimasi akan dilakukan untuk setiap model dengan melakukan hiper-parameter *tuning* dengan mencari kombinasi nilai hiper-parameter yang menghasilkan performa paling tinggi. Proses hiper-parameter *tuning* menggunakan bantuan pustaka talos[5].

[4] https://www.tensorflow.org/
[5] https://github.com/autonomio/talos

Vol. 14, No. 2, September 2021, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v14i2.1164

2.2 Trainable Embedding Layer

Trainable embedding layer yang disediakan oleh pustaka Keras merupakan *supervised learning* dengan nilai bobot yang diinisiasi secara acak dan kemudian diperbaharui selama proses pelatihan model dengan menggunakan algoritma *back-propagation*.

2.3 Pre-trained Word Embedding

Pre-trained word embedding merupakan jenis *unsupervised learning* yang berupaya mencari kata yang sering muncul dalam konteks serupa dan diletakkan berdekatan satu sama lain dalam *embedding space* [10]. Saat *word embedding layer* menggunaakan *pre-trained embedding*, parameter pada *embedding layer* dari pustaka Keras diubah menjadi *pre-trained embeddings*. Hal ini dilakukan agar representasi angka setiap kata yang digunakan sebagai bobot pada lapisan *embedding* tidak diperbaharui selama proses pelatihan. Pada penelitian ini ada 3 *pre-trained word-level embedding* yang dibandingkan word2vec [10], [14], GloVe [11], dan fastText [15]. Word2vec dan fastText dilatih dengan menggunakan bantuan pustaka gensim[6], GloVe dilatih dengan menggunakan sumber kode terbuka dari The Standford Natural Language Processing Group[7], dan dengan *data set* yang sama dari wikipedia bahasa Indonesia[8].

3. HASIL DAN PEMBAHASAN

Dari gabungan dua sumber *data set* yang digunakan diperoleh total 4,892 kalimat, 98,122 kata 13,031 kata unik, dan 5 jenis NEs, yaitu nama orang, organisasi, lokasi, kuantitas, dan waktu. Untuk mengubah kedua *data set* tersebut ke dalam format BILOU dilakukan proses *parsing* yang berbeda. *Data set nlp-experiments* yang menggunakan format pelabelan MUC-6 terlebih dahulu dipisah per kalimat (akhir kalimat ditandai dengan tanda baca titik). Kemudian data yang diapit oleh tag <ENAMEX type=”<NE TYPE>”></ENAME> diekstrak dan diberi label sesuai nilai *type* di *tag* tersebut. Token yang muncul dua kali secara berurutan dengan tipe *tag* yang sama diberi label B-<NE TYPE> untuk token pertama dan L-<NE TYPE> untuk token kedua. Apabila token tersebut muncul lebih dari dua kali secara berurutan dengan tipe *tag* yang sama, maka perlakuannya sama dengan dua token, tetapi token ke-2 sampai token sebelum token terakhir diberi label I-<NE TYPE>. Untuk token yang hanya muncul satu kali dan tidak diapit oleh *tag* diberi label masing-masing U-<NE TYPE> dan O. Proses parsing untuk *data set* indonesian-ner kurang lebih sama, yang membedakan adalah *tag* pada *data set* tersebut menggunakan tipe NEs, sehingga proses *parsing* menggunakan nama *tag* langsung untuk menentukan label setiap token.

Dari proses parsing tersebut diperoleh 21 jenis label dalam format BILOU, dimana yang terbanyak adalah label O. Sedangkan label yang termasuk dalam NEs paling banyak adalah U-PERSON dengan jumlah 1,861 dan yang paling sedikit adalah label U-QUANTITY yang hanya berjumlah 6 token. Jumlah kemunculan untuk setiap label dapat dilihat pada Tabel 1. *Data set* yang digunakan sangat tidak seimbang jumlah antar labelnya dimana yang paling banyak adalah token label O dengan 83,35% dari *data set*.

Pada tahapan pra-pemrosesan pemilihan jumlah token yang digunakan sebagai panjang *sequence* pada data harus dilakukan dengan baik untuk menghindari penambahan *noise* terlalu banyak dan/atau kehilangan informasi akibat dari pemilihan panjang *sequence* yang kurang tepat.

[6] https://radimrehurek.com/gensim/index.html
[7] https://nlp.stanford.edu/projects/glove/
[8] https://dumps.wikimedia.org/idwiki/latest/

Vol. 14, No. 2, September 2021, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v14i2.1164

Pada penelitian ini, panjang *sequence* pada data set ditentukan berdasarkan pertimbangan jumlah token pada setiap observasi data. Jumlah token pada data set paling banyak berada pada rentang 14 (*quartile* 1) sampai dengan 25 (*quartile* 3) token, serta yang paling panjang adalah 109 token. Berdasarkan distribusi jumlah token tersebut, jumlah token yang dipilih untuk diuji pada tahap pemodelan adalah 20, 40, 60, dan 80 token. Jumlah data latih, data validasi, dan data uji masing-masing adalah 70%, 15%, dan 15% dari data set.

Gambar 3 menunjukkan arsitektur model yang digunakan. Pelatihan model menggunakan empat jenis panjang *sequence*, yaitu 20, 40, 60, dan 80. Model tersebut dilatih dengan menggunakan nilai *hiper-parameter* awal sebagai berikut; *max epoch* 100, *loss function softwamx loss with* f1 *score micro average*, *early stopping monitor* = val_f1_score_micro *and patience* = 10, *batch size* 128, *learning algorithm* Adm *with learning rate* 0,001, *dropout* 0,1 and LSTM unit 100 . Gambar 4 dan Gambar 5 menjelaskan *learning curve* proses pelatihan model yang memiliki panjang *sequence* paling optimal sesuai dengan perolehan nilai *f1 score* pada Tabel 2.

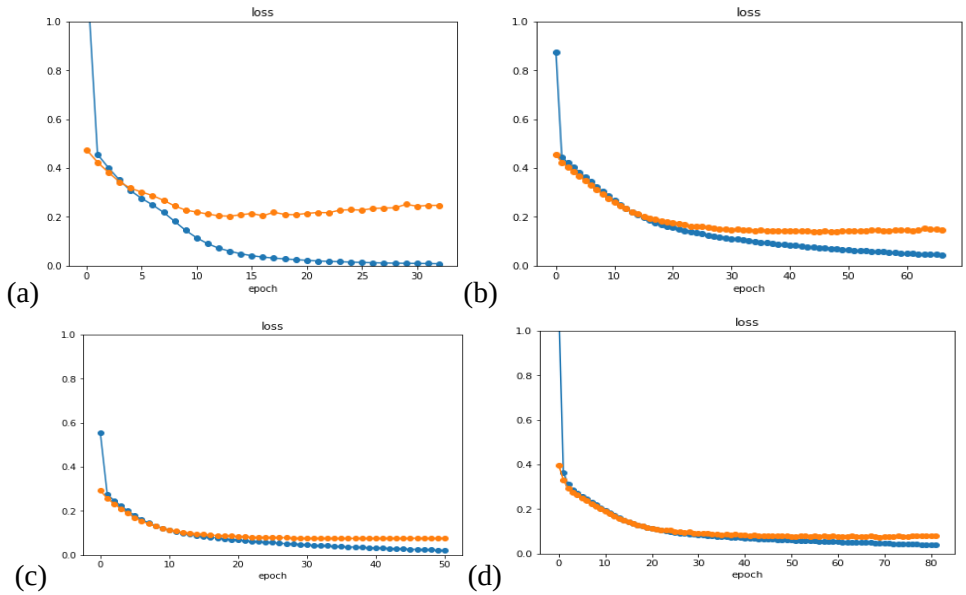
Tabel 1. Jumlah Kemunculan Label

Label	Jumlah	(%)
O	81787	83,35%
U-PERSON	1861	1,89%
U-ORGANIZATION	1722	1,75%
U-LOCATION	1502	1,53%
B-PERSON	1467	1,49%
L-PERSON	1464	1,49%
L-ORGANIZATION	1036	1,05%
B-ORGANIZATION	1035	1,05%
I-ORGANIZATION	941	0,95%
I-TIME	865	0,88%
L-LOCATION	728	0,74%
B-LOCATION	726	0,73%
I-LOCATION	488	0,49%
L-QUANTITY	474	0,48%
B-QUANTITY	474	0,48%
I-PERSON	470	0,47%
I-QUANTITY	363	0,36%
B-TIME	282	0,28%
L-TIME	282	0,28%
U-TIME	149	0,15%
U-QUANTITY	6	0,006%

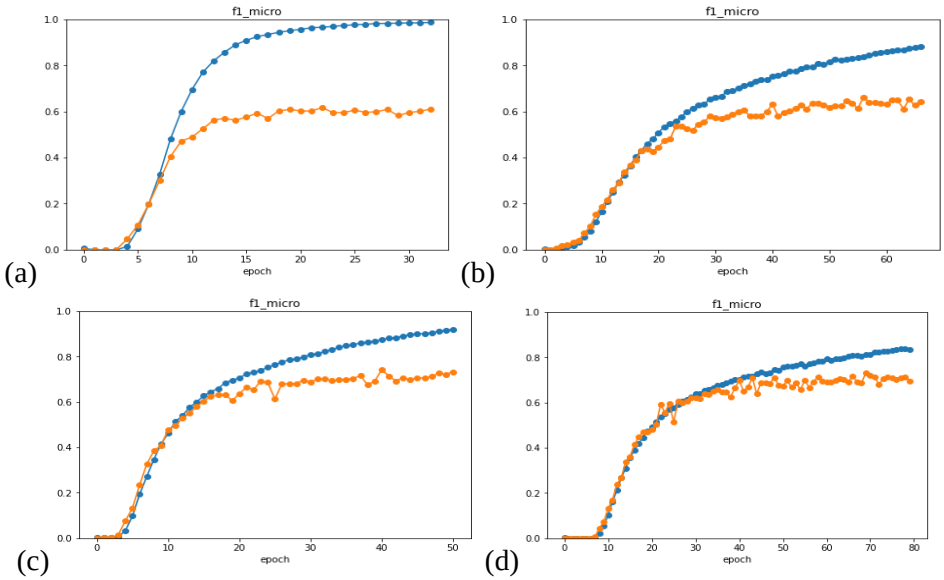
Gambar 3. Arsitektur model BiLSTM

Vol. 14, No. 2, September 2021, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v14i2.1164

Model yang menggunakan GloVe dan fastText cenderung lebih tahan terhadap *overfit* dibandingkan dengan *trainable* dan word2vec *embedding*. Hal tersebut terlihat dari perbedaan nilai *loss* dan f1 *score* pelatihan dan validasi selama proses pelatihan model yang ditunjukkan pada Gambar 4 dan Gambar 5. *Trainable embedding* mengalami *overfit* terhadap data latih besar kemungkinan disebabkan karena jumlah *data set* tidak dapat mengakomodasi jumlah *learning parameters* pada model dengan jumlah 4,234,321 parameter. Dengan jumlah *learning parameters* yang jauh lebih rendah, *pre-trained embeddings* lebih tahan terhadap *overfit*. Tabel 2 menunjukkan hasil evaluasi model menggunakan nilai *hiper-parameter* awal yang sama untuk setiap panjang *sequence* yang berbeda.



Gambar 4. Perubahan nilai *loss*: (a) *trainable embedding layer* (b) word2vec (c) GloVe (d) fastText



Gambar 5. Perubahan nilai *f1 score*: (a) *trainable embedding layer* (b) word2vec (c) GloVe (d) fastText

Model yang menggunakan *pre-trained embedding* memiliki nilai f1 *score* lebih tinggi daripada model yang menggunakan *trainable embedding* untuk semua skema pemilihan panjang *sequence* dan

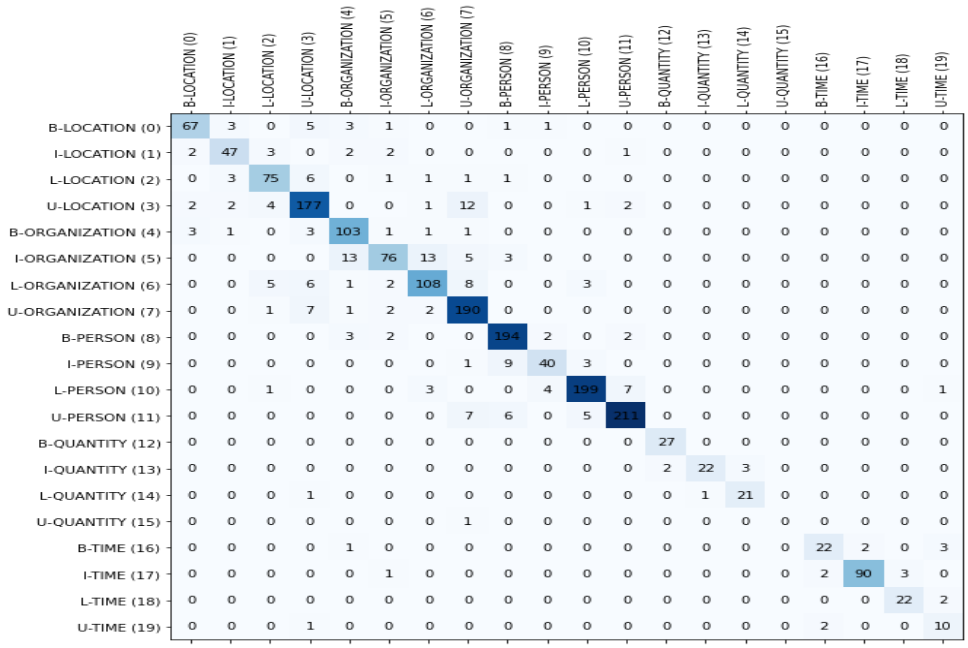
Vol. 14, No. 2, September 2021, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v14i2.1164

dengan *training time* yang lebih rendah. *Trainable* dan word2vec *embedding* memiliki nilai f1 *score* tertinggi pada panjang *sequence* 40 token, sedangkan GloVE dan fastText pada 60 token. GloVe unggul dibandingkan dengan teknik *embedding* yang lain dengan nilai f1 *score* 70,43.

Gambar 6 adalah *confusion matrix* hasil evaluasi menggunakan data uji pada model dengan perolehan nilai f1 *score* tertinggi. Beberapa kesalahan prediksi terjadi pada entitas dan label yang sama, seperti I-ORGANIZATION diprediksi 13 kali sebagai B-ORGANIZATION, I-ORGANIZATION diprediksi 13 kali sebagai L-ORGANIZATION, dan U-LOCATION diprediksi 12 kali sebagai U-ORGANIZATION.

Tabel 2. Nilai *f1 score* setiap pemilihan panjang *sequence*

Panjang *Sequence*	*Trainable embedding layer*		word2vec		GloVe		fastText	
	# Training Time / Epoch	F1 *Score*	*# Training Time / Epoch*	F1 *Score*	*# Training Time / Epoch*	F1 *Score*	*# Training Time / Epoch*	F1 *Score*
20	76,70 / 36	58,60	42,36 / 67	63,75	25,06 / 39	65,47	32,89 / 51	65,52
40	75,65 / 33	**60,85**	54,89 / 67	**64,24**	29,79 / 40	68,54	43,57 / 58	65,39
60	93,72 / 37	59,79	58,59 / 64	61,73	46,55 / 51	**70,43**	74,71 / 82	**67,90**
80	116,39 / 43	59,47	81,39 / 77	63,56	61,40 / 58	67,69	74,00 / 70	63,37



Gambar 6. *Confusion matrix* pada data uji pada model BiLSTM

Optimasi *hiper-parameter* dilakukan dengan mencari nilai kombinasi *hiper-parameter* baru dari nilai awal. Pemilihan nilai uji merupakan bagian terpenting pada proses optimasi *hiper-parameter* agar parameter terbaik termasuk dalam daftar nilai uji yang didefinisikan. Terdapat lima

Vol. 14, No. 2, September 2021, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v14i2.1164

parameter yang akan dioptimasi, yaitu *learning algorithm* (*optimizer*) & *learning rate*, jumlah LSTM *units*, dan nilai *dropout rate*. Pemilihan nilai uji untuk *learning algorithm* didasarkan pada [16] yang menunjukkan bahwa *learning algorithm* Nadam, Adam, dan RMSprop memperoleh nilai *f1 score* tertinggi dibanding *optimizer lainnya*. Nilai uji *hiper-parameter* yang lain dipilih dengan mengambil beberapa nilai di bawah dan di atas nilai awal. Misalnya, nilai awal parameter LSTM *units* adalah 100, maka nilai uji yang dipilih adalah [50; 100; 200]. Pengecualian untuk nilai *dropout rate* yang hanya dipilih nilai yang lebih besar dari nilai awal karena model cenderung mengalami *overfit* berdasarkan *learning curve* yang ditunjukkan pada Gambar 4 dan Gambar 5. Hasil optimasi *hiper-parameter* untuk kedua model tersebut ditunjukkan pada Tabel 3. Pemilihan nilai terbaik tersebut didasarkan pada perolehan nilai f1 *score* validasi tertinggi berdasarkan nilai parameter uji. Setelah proses optimasi nilai *hiper-parameter*, model kembali dievaluasi dengan menggunakan data uji. Hasil evaluasi kedua model tersebut ditunjukkan pada Tabel 4 dan Tabel 5.

Tabel 3. Optimasi nilai hiper-parameter

Round	***Hiper-parameter***	**Nilai Uji**	**Nilai Terbaik / F1 *Score***			
			Trainable embedding layer	**word2vec**	**GloVe**	**fastText**
1	*Learning algorithm*	[Nadam; Adam; RMSprop]	RMSprop / 67,07	RMSprop / 71,99	RMSprop / 76,12	RMSprop / 78,50
	Learning rate	[0,01; 0,001; 0,0001]	0,01 / 67,07	0,01 / 71,99	0,001 / 76,12	0,01 / 78,50
2	LSTM *units*	[50; 100; 200]	100 / 69,27	50 / 73.40	100 / 77,28	50 / 79,40
3	*Dropout*	[0,1; 0,3; 0,5; 0,7]	0,7 / 74,48	0,5 / 78,31	0,7 / 76,72	0,5 / 82,59

Tabel 4. Perbandingan performa model pada data uji dengan nilai hiper-parameter terbaik

Model	***Trainable Parameters***	***Training Time / Epoch***	**Initial F1 *Score***	***Optimized* F1 *Score***
trainable embedding layer	4,234,321	168 / 25	60,85	72,44
word2vec	142,521	62 / 23	64,24	73,24
GloVe	142,521	532 / 58	70,43	**76,48**
fastText	142,521	267 / 37	67,90	73,83

Tabel 5. Perbandingan performa *word embedding* pada rata-rata nilai *f1 score* setiap NEs dan posisi token

		% Kemunculan	***trainable embedding layer***	**word2vec**	**GloVe**	**fastText**
NEs	PERSON	5,36%	75,58	81,09	82,73	**83,19**
	ORGANIZATION	4,82%	67,12	**69,53**	69,34	67,33
	LOCATION	3,50%	66,61	69,94	72,74	**72,76**

Vol. 14, No. 2, September 2021, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v14i2.1164

	TIME	1,60%	69,47	65,71	**69,82**	68,08
	QUANTITY	1,34%	**57,63**	38,18	43,21	49,63
Posisi Token	B	4,06%	64,97	67,28	**68,32**	74,05
	I	3,18%	63,50	60,80	**67,53**	64,75
	L	4,06%	65,46	69,63	72,20	**73,03**
	O	83,35%	98,41	98,67	**99,15**	99,13
	U	5,34%	**75,20**	61,84	61,73	60,97

Optimasi nilai *hiper-parameter* dapat meningkatkan performa model dengan signifikan dari nilai parameter awal. *Trainable embedding layer* mengalami peningkatan nilai f1 *score* dari 60,85 menjadi 72,44, word2vec dari 64,24 menjadi 73,24, GloVe dari 70,43 menjadi 76,48, dan fastText dari 67,90 menjadi 73,83. *Trainable embedding layer* mengalami peningkatan performa yang paling signifikan setelah proses optimasi nilai *hiper-parameter* dengan nilai *dropout* paling optimal adalah 0,7. Hal tersebut memperkuat dugaan di awal bahwa model tersebut mengalami *overfit* terhadap data latih. Dari keempat teknik *embedding* yang diuji, GloVe memperoleh nilai initial dan *optimized* f1 *score* tertinggi.

Tabel 5 menunjukkan perolehan rata-rata nilai f1 *score* untuk setiap NEs dan posisi token. Jumlah % kemunculan NEs dan posisi token cenderung berbanding lurus dengan rata-rata nilai f1 *score* yang diperoleh. *Pre-trained embeddings* unggul dibandingkan dengan *trainable embedding layer* dimana fastText adalah yang terbaik untuk NEs dan GloVe untuk posisi token.

4. KESIMPULAN DAN SARAN

Pada penelitian ini, telah dilakukan perbandingan berbagai teknik *word embedding* pada permasalahan NER untuk Bahasa Indonesia dengan menggunakan arsitektur model BiLSTM. Pendekatan *unsupervised* menggunakan *pre-trained embedding* memberikan performa yang lebih baik dibandingkan pendekatan *supervised* menggunakan *trainable embedding layer* karena pendekatan ini lebih tahan terhadap *overfit*. Optimasi *hiper-parameter* meningkatkan performa *trainable embedding layer*, namun performa *pre-trained* embedding tetap lebih unggul. Diantara beberapa *pre-trained embedding* yang dibandingkan dalam penelitian ini, teknik *embedding* dengan perolehan nilai f1 *score* rataan mikro terbaik adalah GloVe dengan nilai 76,48.

Untuk penelitian selanjutnya sebaiknya menggunakan jumlah *data set* yang lebih banyak untuk mengurangi *overfit* model terhadap data latih.

DAFTAR PUSTAKA

[1] R. Alfred, L. C. Leong, C. K. On, and P. Anthony, "Malay named entity recognition based on rule-based approach," *Int. J. Mach. Learn. Comput.*, vol. 4, no. 3, pp. 300–306, Jun. 2014.
[2] W. Gunawan, D. Suhartono, F. Purnomo, and A. Ongko, "Named-entity recognition for indonesian language using bidirectional LSTM-CNNs," in *Procedia Computer Science*, 2018, vol. 135, pp. 425–432.
[3] J. P. C. Chiu and E. Nichols, "Named entity recognition with bidirectional LSTM-CNNs," *Trans. Assoc. Comput. Linguist.*, vol. 4, pp. 357–370, Nov. 2015.
[4] E. Utami and S. Hartati, "Pendekatan metode rule based dalam mengalihbahasakan teks bahasa inggris ke teks bahasa indonesia," *J. Inform.*, vol. 8, no. 1, pp. 42–53, Jul. 2007.
[5] A. Willyawan, "Named entity recognition (NER) bahasa indonesia menggunakan conditional random field dan pos-tagging," Universitas Sumatera Utara, 2018.
[6] Dayinta Warih Wulandari, Putra Pandu Adikara, and Sigit Adinugroho, "Named entity

Vol. 14, No. 2, September 2021, P-ISSN 1978-9262, E-ISSN 2655-5018
DOI: https://doi.org/10.33322/petir.v14i2.1164

recognition (NER) pada dokumen biologi menggunakan rule based dan naïve bayes classifier," Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 2, no. 11, pp. 4555–4563, Apr. 2018.

[7] L. Ratinov and D. Roth, "Design challenges and misconceptions in named entity recognition," in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*, 2009, pp. 147–155.

[8] A. Akhundov, D. Trautmann, and G. Groh, "Sequence labeling: a practical approach," Aug. 2018.

[9] Y. Luo, H. Zhao, and J. Zhan, "Named Entity Recognition Only from Word Embeddings," 2019.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, 2013.

[11] J. Pennington, R. Socher, and C. D. Manning, "GloVe: global vectors for word representation."

[12] R. Grishman and B. Sundheim, "Message understanding conference-6: a brief history," in *The 16th International Conference on Computational Linguistics*, 1996, vol. 1, pp. 466–471.

[13] X. Ma and E. Hovy, "End-to-end sequence labeling via bi-directional LSTM-CNNs-CRF," *54th Annu. Meet. Assoc. Comput. Linguist. ACL 2016 - Long Pap.*, vol. 2, pp. 1064–1074, Mar. 2016.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," 2013.

[15] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," 2016.

[16] N. Reimers and I. Gurevych, "Optimal hyperparameters for deep LSTM-networks for sequence labeling tasks."