

BAB VI

KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan hasil penelitian Pembangunan Model Pembelajaran Mesin untuk Identifikasi Struktur Kalimat Bahasa Indonesia, dapat disimpulkan:

1. Model LSTM dan BiLSTM yang dibuat dengan *API Keras* dapat digunakan untuk mengidentifikasi struktur kalimat Bahasa Indonesia.
2. Kemampuan model untuk mengidentifikasi struktur kalimat Bahasa Indonesia sudah cukup baik. Seluruh model meraih skor F1 di atas 74% pada tiap *epoch* yang diamati dan data uji.
3. Model BiLSTM dengan 2 *layer* dan 50 unit *cell* yang dilatih hingga *epoch* ke 50 menghasilkan skor F1 yang lebih baik dibandingkan dengan model lain ketika dihadapkan dengan data *test*.

6.2 Saran

Kemampuan model untuk mengidentifikasi struktur kalimat Bahasa Indonesia sudah cukup baik. Peningkatan kemampuan model untuk mengidentifikasi masih dapat ditingkatkan. Beberapa saran untuk penelitian Pembangunan Model Pembelajaran Mesin untuk Identifikasi Struktur Kalimat Bahasa Indonesia adalah sebagai berikut:

1. Saran peneliti untuk meningkatkan kemampuan model yaitu dengan melakukan *feature engineering* pada dataset kalimat Bahasa Indonesia.
2. Selain *feature engineering*, jumlah sampel yang digunakan dalam proses *training* harus ditingkatkan.
3. Penelitian menggunakan model yang lain. Model *attention-based* dapat digunakan karena model *attention* menunjukkan performa yang baik dalam berbagai tugas *Natural Language Processing*.

DAFTAR PUSTAKA

- Alwi, H., Lapoliwa, H., & Darmowidjojo, S. (2007). *Tata Bahasa Baku Bahasa Indonesia* (3rd Edition ed.). Jakarta: Balai Pustaka.
- Brownlee, J. (2017, July 3). *Machine Learning Mastery*. Dipetik July 7, 2018, dari Machine Learning Mastery: <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>
- Brunner, C. (2018, January 12). *NBHerard*. Dipetik January 15, 2018, dari <http://nbherard.com>
- Chen, D., & Manning, C. (2014). A Fast and Accurate Dependency Parser using Neural Networks. (hal. 740-750). California: Conference on Empirical Methods in Natural Language Processing (EMNLP).
- Deng, L., & Yu, D. (2014). *Deep Learning: Methods and Application* (1st Edition ed.). Redmond: now.
- Goldberg, Y. (2016, November). A Primer on Neural Network Models for Natural Language Processing. *Journal of Artificial Intelligence Research*, 57, 345-420.
- Grave, E., Bojanowski, P., Joulin, A., & Mikolov, T. (2016). Enriching Word Vectors with Subword Information. *arXiv, preprint arXiv:1607.04606*.
- He, L., Lee, K., Lewis, M., & Zettlemoyer, L. (2017). Deep Semantic Role Labeling: What works and What's next. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* (hal. 473-483). Vancouver: Association for Computational Linguistics.
- Hirschberg, J., & Manning, C. D. (2015, July). Advances in natural language processing. *Science*, 349(6245), 261-266.
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(*), 1735-1780.
- Indonesia, B. P. (2015). *Seri Penyuluhan Bahasa Indonesia: Kalimat* (1st Edition ed.). Jakarta: Pusat Pembinaan dan Pemasyarakatan Kementerian Pendidikan dan Kebudayaan Indonesia.
- Iswandi, I., Suwardi, I., & Maulidevi, N. (2013, October). Otomasi Interpretasi Data Akuntansi Berbasis Natural Language Processing. *Jurnal Sistem Informasi*, 5(2), 622-628.
- Kingma, D. P., & Ba, J. L. (2015). ADAM: A Method For Stochastic Optimization. *International Conference On Learning Representations*. San Diego.

- Kiperwasser, E., & Goldberg, Y. (2016). Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations. *TACL*, 4, 313-327.
- Lutz, M. (2013). *Learning Python* (5th Edition ed.). Sebastopol: O'Reilly.
- Mitchell, T. M. (1997). *Machine Learning* (1st Edition ed.). Mcgraww-Hill.
- Nurzahputra, A., & Muslim, M. (2016). Analisis Sentimen pada Opini Mahasiswa Menggunakan Natural Language Processing. Semarang: Seminar Nasional Ilmu Komputer.
- Olah, C. (2015, August 27). *colah's Blog*. Dipetik July 7, 2018, dari <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- Pustejovsky, J., & Stubbs, A. (2012). *Natural Language Annotation for Machine Learning* (1st Edition ed.). Sebastopol: O'Reilly.
- Ramachandran, P., Zoph, B., & Le, V. Q. (2017). *Searching for Activation Function*. Cornell University, Cornell University Library. Ithaca: www.arXiv.org.
- Sanger, J., & Feldman, R. (2007). *The Text Mining Handbook : Advanced Approaches in Analyzing Unstructured Data* (1st Edition ed.). New York: Cambridge University Press.
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. (Y. Bengio, Penyunt.) *Journal of Machine Learning Research*, 15, 1929-1958.
- Tan, Z., Wang, M., Xie, J., Chen, Y., & Shi, X. (2017). *Deep Semantic Role Labeling with Self-Attention*. Cornell University, Cornell University Library. New York: arXiv.org.

LAMPIRAN

Berikut ini adalah lampiran kode untuk membangun model identifikasi struktur kalimat bahasa Indonesia:

```
import os
import sys
import random
import numpy as np
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers import Dense, Input,
Embedding, Bidirectional, LSTM, TimeDistributed
from keras import metrics, optimizers
from keras.models import Model
from keras import backend as K
from keras.callbacks import Callback, ModelCheckpoint,
CSVLogger

from configuration import *
from tokenize_with_nltk import *
from dictionary import Dictionary
from measurement import ZeroPaddedF1Score

import tensorflow as tf
from keras_contrib.layers import CRF
from matplotlib import pyplot
from pandas import DataFrame
import time
from keras.utils import plot_model

Checkpoint_PATH = '/home/wiseguy909/Desktop/final/saved/model.hdf5'
BASE_DIR = '/home/wiseguy909/Desktop/final'
GLOVE_DIR = os.path.join(BASE_DIR, '')
MAX_SEQUENCE_LENGTH = 100
MAX_NUM_WORDS = 20000
EMBEDDING_DIM = 300
config = get_config('/home/wiseguy909/Desktop/final/config.json')
train_path = '/home/wiseguy909/Desktop/final/3840kalimat_SPOK_random_
split_train_70.txt'
dev_path = '/home/wiseguy909/Desktop/final/3840kalimat_SPOK_random_sp
lit_eval_20.txt'
WORD_DICT_PATH = "/home/wiseguy909/Desktop/final/word_dict.txt"
```

```

LABELDICT_PATH = "/home/wiseguy909/Desktop/final/label_dict.txt"
CSVLoggerPATH =
"/home/wiseguy909/Desktop/final/saved/Model_3840/Model.csv"

csv_logger = CSVLogger(CSVLoggerPATH, append=True, separator=';')

word_dict = Dictionary(unknown_token=UNKNOWN_TOKEN)
label_dict = Dictionary()

word_dict.load(WORDDICT_PATH)
label_dict.load(LABELDICT_PATH)

# first, build index mapping words in the embeddings set
# to their embedding vector

print('Indexing word vectors.')
embeddings_index = {}
with open(os.path.join(GLOVE_DIR, 'wiki.id.vec')) as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
    if not UNKNOWN_TOKEN in embeddings_index:
        embeddings_index[UNKNOWN_TOKEN] =
np.asarray([random.gauss(0, 0.01) for _ in range(EMBEDDING_DIM)])
print('Found %s word vectors.' % len(embeddings_index))

# tokenisasi & ubah ke tensor
raw_train_sents = get_sentences_with_nltk(train_path,
config.use_se_marker)
raw_dev_sents = get_sentences_with_nltk(dev_path,
config.use_se_marker)

train_token=[string_sequence_to_ids(sent[0], word_dict, True,
embeddings_index) for sent in raw_train_sents]
train_label=[string_sequence_to_ids(sent[2], label_dict) for sent
in raw_train_sents]

LABELDICT_PATH = "/home/wiseguy909/Desktop/final/label_dict.txt"
CSVLoggerPATH =

```

```

dev_token=[string_sequence_to_ids(sent[0], word_dict, True,
embeddings_index) for sent in raw_dev_sents]
dev_label=[string_sequence_to_ids(sent[2], label_dict) for
sent in raw_dev_sents]

print('Found %s word dict after converting sequence to id.'
% word_dict.size())

#convert to categorical label & pad the sequence

x_train = pad_sequences(train_token,
maxlen=MAX_SEQUENCE_LENGTH,padding='post')
x_val=
pad_sequences(dev_token,maxlen=MAX_SEQUENCE_LENGTH,padding='
post')

y_train= []
y_val = []
for s in train_label:
    y_train.append(pad_sequences(to_categorical(np.asarray
ay(s)),maxlen=12,padding='post'))
for s in dev_label:
    y_val.append(pad_sequences(to_categorical(np.asarray
(s)),maxlen=12,padding='post'))
y_train =
pad_sequences(y_train,maxlen=MAX_SEQUENCE_LENGTH,padding='po
st')
y_val
= pad_sequences(y_val,maxlen=MAX_SEQUENCE_LENGTH,padding='p
ost')
print('X train Shape {}'.format(x_train.shape))
print("X val shape {}".format(x_val.shape))
print("Y train shape {}".format(y_train.shape))
print("Y val shape {}".format (y_val.shape))

dev_token=[string_sequence_to_ids(sent[0], word_dict, True,
embeddings_index) for sent in raw_dev_sents]
dev_label=[string_sequence_to_ids(sent[2], label_dict) for
sent in raw_dev_sents]

print('Found %s word dict after converting sequence to id.'
% word_dict.size())

```

```

prepare embedding matrix
print('Preparing embedding matrix.')
word_index= word_dict.idx2str
num_words = min(MAX_NUM_WORDS, len(word_index) + 1)
embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))
for i,word in enumerate(word_dict.idx2str):
    if i >= num_words:
        continue
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

# load pre-trained word embeddings into an Embedding layer
# note that we set trainable = False so as to keep the embeddings
fixed
embedding_layer = Embedding(num_words,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
                            input_length=MAX_SEQUENCE_LENGTH,
                            trainable=False,mask_zero=True)

# init Keras Callback
ZeroPaddedF1Score = ZeroPaddedF1Score()
ModelCheck =
ModelCheckpoint(Checkpoint_PATH,monitor='val_viterbi_acc',
                verbose=0,save_best_only=False)
sequence_input = Input(shape=(MAX_SEQUENCE_LENGTH,),
dtype='int32',name='sequence_input')
embedded_sequences = embedding_layer(sequence_input)
x = Bidirectional(LSTM(75,return_sequences=True,
unit_forget_bias=True,recurrent_dropout=config.recurrent_dropo
ut_prob,dropout=config.layer_dropout))(embedded_sequences)
x = Bidirectional(LSTM(75,return_sequences=True,
unit_forget_bias=True,recurrent_dropout=config.recurrent_dropo
ut_prob,dropout=config.layer_dropout))(x)
td = TimeDistributed(Dense(50,activation='relu'))(x)
crf = CRF(12)
preds = crf(td)
model = Model(inputs=[sequence_input], outputs=[preds])

prepare embedding matrix
print('Preparing embedding matrix.')
word_index= word_dict.idx2str
num_words = min(MAX_NUM_WORDS, len(word_index) + 1)
embedding_matrix = np.zeros((num_words, EMBEDDING_DIM))

```

```

model.compile(loss=crf.loss_function,
               optimizer=adam,metrics=[crf.accuracy])

print('Training model..')
start = time.time()
history = model.fit(x_train,y_train,batch_size=32,

                    epochs=config.max_epochs,validation_data=(x_val,y_val),

                    callbacks=[ZeroPaddedF1Score,ModelCheck,csv_logger]
                    )
end = time.time()
print('Model memerlukan %0.2F untuk training'.format(end-
start))

train[str(i)] = history.history['loss']
val[str(i)] = history.history['val_loss']
acc[str(i)] = history.history['viterbi_acc']
val_acc[str(i)] = history.history['val_viterbi_acc']
val_f1[str(i)] = ZeroPaddedF1Score.val_f1s

# plot loss & akurasi
pyplot.plot(train,label='train')
pyplot.plot(val,label='validation')
pyplot.title('model train vs valid')
pyplot.ylabel('loss')
pyplot.xlabel('epoch')
pyplot.legend(['train','validation'],loc='upper right')
pyplot.show()

pyplot.plot(acc,label='train acc')
pyplot.plot(val_acc,label='validation acc')
pyplot.plot(val_f1,color='r',label='F1-Score')
pyplot.yticks(np.arange(0,1.1,0.1))
pyplot.title('model acc')
pyplot.ylabel('acc')
pyplot.xlabel('epoch')
pyplot.legend(['train','validation','F1-Score'],loc='lower
right')
pyplot.axhline(y=0.7, color='r', linestyle='-'
',alpha=0.4,linewidth=0.2)
pyplot.axhline(y=0.8, color='r', linestyle='-'
',alpha=0.4,linewidth=0.2)
pyplot.show()

model.compile(loss=crf.loss_function

```

tokenize_with_nltk.py

```
from nltk.tokenize import word_tokenize
import nltk

UNKNOWN_TOKEN = '*UNKNOWN*'

def get_sentences_with_nltk(filepath, use_se_marker=False):
    """ Read tokenized SRL sentences from file.
        File format: {predicate_id} [word0, word1 ...] ||| [label0,
        label1 ...]
        Return:
            A list of sentences, with structure: [[words], predicate,
            [labels]]
    """
    sentences = []
    with open(filepath) as f:
        for line in f.readlines():
            inputs = line.strip().split('|||')
            lefthand_input = word_tokenize(inputs[0].decode("utf-8"))
            # If gold tags are not provided, create a sequence of
            dummy tags.
            righthand_input = inputs[1].strip().split() if len(inputs)
            > 1 \
                                else ['O' for _ in lefthand_input[1:]]

            predicate = int(lefthand_input[0])
            words = lefthand_input[1:]
            labels = righthand_input
            sentences.append((words, predicate, labels))
    return sentences

def tokenize(sequence):
    """ Read tokenized SRL sentences from file.
        File format: {predicate_id} [word0, word1 ...] ||| [label0,
        label1 ...]
        Return:
            A list of sentences, with structure: [[words], predicate,
            [labels]]
    """
    sentences = []
    words = word_tokenize(sequence)
    sentences.append((words))
    return sentences
```

```

def string_sequence_to_ids(str_seq, dictionary, lowercase=False,
pretrained_embeddings=None):
    """token/kata yang tidak terdapat dalam wordvec akan dianggap
    sebagai UNKNOWN_TOKEN
    """
    ids = []
    for s in str_seq:
        if s is None:
            ids.append(-1)
            continue
        if lowercase:
            s = s.lower()
        if (pretrained_embeddings != None) and not (s in
pretrained_embeddings) :
            s = UNKNOWN_TOKEN
        ids.append(dictionary.add(s))
    return ids

def label_to_ids(str_seq, dictionary):
    """jika menggunakan fungsi ini pastikan ketika menampilkan
    prediksi index label juga ditambah 1
    """
    ids = []
    for s in str_seq:
        if s is None:
            ids.append(-1)
            continue
        if s not in dictionary :
            dictionary.add(s)
        ids.append(dictionary.get_index(s) + 1)
    return ids

def sequence_to_ids_predict(str_seq, dictionary,
lowercase=False):
    """token/kata yang tidak terdapat dalam wordvec akan dianggap
    sebagai UNKNOWN_TOKEN
    """
    ids = []
    for s in str_seq:
        if s is None:
            ids.append(-1)
            continue
        if lowercase:
            s = s.lower()
        if s not in dictionary.str2idx :
            s = UNKNOWN_TOKEN
        ids.append(dictionary.get_index(s))
    return ids

```

Dictionary.py

```
class Dictionary(object):
    def __init__(self, unknown_token=None):
        self.str2idx = {}
        self.idx2str = []
        self.accept_new = True
        self.unknown_token = None
        self.unknown_id = None
        if unknown_token != None:
            self.set_unknown_token(unknown_token)

    def set_unknown_token(self, unknown_token):
        self.unknown_token = unknown_token
        mask = "*Masking*" #set masking token at index zero
        self.add(mask)
        self.unknown_id = self.add(unknown_token)

    def add(self, new_str):
        if not new_str in self.str2idx:
            if self.accept_new:
                self.str2idx[new_str] = len(self.idx2str)
                self.idx2str.append(new_str)
            else:
                if self.unknown_id is None:
                    raise LookupError('Trying to add new token to a
freezed dictionary with no pre-defined unknown token: ' +
new_str)
                return self.unknown_id

        return self.str2idx[new_str]
    def add_all(self, str_list):
        return [self.add(s) for s in str_list]

    def get_index(self, input_str):
        if input_str in self.str2idx:
            return self.str2idx[input_str]
        return None

    def size(self):
        return len(self.idx2str)

class Dictionary(object):
    def __init__(self, unknown_token=None):
        self.str2idx = {}
```

```

def save(self, filename):
    with open(filename, 'w') as f:
        for s in self.idx2str:
            f.write(s + '\n')
        f.close()

def load(self, filename):
    with open(filename, 'r') as f:
        for line in f:
            line = line.strip()
            if line != '':
                self.add(line)
        f.close()

```

Configuration.py

```

import json
from argparse import Namespace

def get_config(config_filepath):
    with open(config_filepath, 'r') as config_file:
        conf = json.load(config_file, object_hook=lambda d:
Namespace(**d))
    return conf

import json
from argparse import Namespace

```

Measurement.py

```

from keras.callbacks import Callback
import numpy as np
from sklearn.metrics import f1_score, recall_score, precision_score
#measure F1-Score using callback

from keras.callbacks import Callback
import numpy as np
from sklearn.metrics import f1_score, recall_score, precision_score
#measure F1-Score using callback

```

```

#measure F1-Score using callback
def zero_padded_f1(y_true, y_pred):
    y_pred_flat, y_true_flat = [], []
    for y_pred_i, y_true_i in zip(y_pred.flatten(),
y_true.flatten()):
        if y_true_i != 0:
            y_pred_flat.append(y_pred_i)
            y_true_flat.append(y_true_i)
    result = f1_score(y_true_flat, y_pred_flat, average='micro')
    recall =
recall_score(y_true_flat, y_pred_flat, average='micro')
    precision =
precision_score(y_true_flat, y_pred_flat, average='micro')
    return result, recall, precision
class ZeroPaddedF1Score(Callback):
    def on_train_begin(self, logs={}):
        self.val_f1s = []
        self.rec = []
        self.prec = []

    def on_epoch_end(self, epoch, logs={}):
        y_true = np.argmax(self.validation_data[1], axis=-1)
        y_pred =
np.argmax(self.model.predict([self.validation_data[0]]), axis=-
1)

        val_f1, rec, prec = zero_padded_f1(y_true, y_pred)
        logs['val_f1s'] = val_f1
        logs['val_recall'] = rec
        logs['val_precision'] = prec
        self.val_f1s.append(val_f1)
        self.rec.append(rec)
        self.prec.append(prec)
        print(' - Val F1: %f - Val Recall: %f - Val Precision:
%f' % (val_f1, rec, prec))

#measure F1-Score using callback

```

Kode prediksi adalah sebagai berikut:

```
import numpy as np
import random
from keras.models import load_model
from keras_contrib.layers import CRF
from tokenize_with_nltk import *
from dictionary import Dictionary
from keras.preprocessing.sequence import pad_sequences

word_dict = Dictionary(unknown_token=UNKNOWN_TOKEN)
EMBEDDING_DIM = 300
MAX_SEQUENCE_LENGTH = 100
label_dict = Dictionary()
def create_custom_objects():
    instanceHolder = {"instance": None}
    class ClassWrapper(CRF):
        def __init__(self, *args, **kwargs):
            instanceHolder["instance"] = self
            super(ClassWrapper, self).__init__(*args,
**kwargs)
        def loss(*args):
            method = getattr(instanceHolder["instance"],
"loss_function")
            return method(*args)
        def accuracy(*args):
            method = getattr(instanceHolder["instance"],
"accuracy")
            return method(*args)
        def viterbi_acc(*args):
            method = getattr(instanceHolder["instance"],
"viterbi_acc")
            return method(*args)
    return {"ClassWrapper": ClassWrapper, "CRF": ClassWrapper,
"loss": loss, "accuracy": accuracy, "viterbi_acc": viterbi_acc}

word_dict.load('/home/wiseguy909/Desktop/final/word_dict_with_
embedding_word.txt')
label_dict.load('/home/wiseguy909/Desktop/final/label_dict.txt
')
print("word dict size: {}".format(len(word_dict.str2idx)))
model =
load_model("/home/wiseguy909/Desktop/final/saved/model.hdf5",
custom_objects=create_custom_objects())
print('Model loaded.')
```

```
import numpy as np
```

```

text = "Kepakaran Teguh diakui banyak orang."

token = tokenize(text)

train_token = [sequence_to_ids_predict(w,word_dict,True) for w
in token]

x_train = pad_sequences(train_token,
maxlen=MAX_SEQUENCE_LENGTH,padding='post')

print('x_train shape: {}'.format(x_train.shape))
print(x_train)
pred = model.predict(x_train)
pred = np.argmax(pred, axis=-1)
print("{:15}||{}".format("Word", "Pred"))
print(30 * "=")
for w, p in zip(token[0], pred[0]):
    #if w != 0:
    print("{:15}: {}".format(w, label_dict.idx2str[p]))

text = "Kepakaran Teguh diakui banyak orang."

token = tokenize(text)

train_token = [sequence_to_ids_predict(w,word_dict,True) for w
in token]

```

Contoh hasil prediksi adalah sebagai berikut:

No	Token	Gold	Prediksi
1	Gadis	B-S	B-S
	itu	I-S	I-S
	cantik	B-P	B-P
	.	O	O
2	Pak	B-S	B-S
	Jokowi	I-S	I-S
	meresmikan	B-P	B-P
	jalan	B-O	B-O
	Tol	I-O	I-O
	.	O	O
3	Pak	B-S	B-S
	Jokowi	I-S	I-S
	meresmikan	B-P	B-P

	jalan	B-O	B-O
	Tol	I-O	I-O
	di	B-K	I-O
	Sragen	I-K	I-O
	.	O	O
4	Perilaku	B-S	B-S
	Jegar	I-S	I-S
	dibenci	B-P	B-P
	semua	B-Pel	B-O
	orang	I-Pel	I-O
	.	O	O
5	Mario	B-S	B-S
	berada	B-P	B-P
	di	B-K	B-K
	Jakarta	I-K	I-K
	.	O	O
6	Jegar	B-S	B-S
	dibenci	B-P	B-P
	semua	B-Pel	B-O
	orang	I-Pel	I-O
	karena	B-K	B-K
	dia	I-K	I-K
	pelit	I-K	I-K
	.	O	O

Cell tabel yang berwarna oranye adalah hasil prediksi label yang salah.