Task 1

Pos Tagging

Nixon Andhika / 13517059
Ferdy Santoso / 13517116
Jan Meyer Saragih / 13517131

Imports

In [1]:
```
import numpy as np
import pickle
import os
from nltk.corpus import wordnet, brown, treebank, conll2000
from keras.models import Sequential, Model, load_model
from keras.layers import (
    InputLayer,
    LSTM,
    Embedding,
    TimeDistributed,
    Dense,
    Bidirectional,
    Activation,
    Dropout
)
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.optimizers import Adam
from keras.utils.np_utils import to_categorical
from keras import backend

from sklearn.model_selection import train_test_split
```

Constants

In [2]:
```
TEST_SIZE = 0.1
VAL_SIZE = 0.15
EPOCH_COUNT = 3
BATCH_SIZE = 128
```

Dataset

Dataset yang digunakan adalah dataset dari nltk corpus library. Di dataset, setiap kata telah dilabeli dengan POS Tag.

In [3]:
```
treebank_corpus = treebank.tagged_sents(tagset='universal')
brown_corpus = brown.tagged_sents(tagset='universal')
conll_corpus = conll2000.tagged_sents(tagset='universal')
tagged_sentences = treebank_corpus + brown_corpus + conll_corpus
```

Preprocessing

Separate Word and Tag

Dataset yang diimpor memiliki data berupa tuple (word, tag) sehingga perlu dipisah terlebih dahulu. Setiap sentence words (list of word) dimasukkan ke variabel X sedangkan setiap sentence tags (list of tags) dimasukkan ke variabel Y. Selain itu, dibentuk list semua kata unik dari dataset yang disimpan dalam variabel words dan list semua tag unik yang disimpan dalam variabel tags.

In [4]:
```
X = []
Y = []

for sentence in tagged_sentences:
    words_temp = []
    tags_temp = []
    for pair in sentence:
        words_temp.append(pair[0])
        tags_temp.append(pair[1])
    X.append(words_temp)
    Y.append(tags_temp)

words = set([word.lower() for sentence in X for word in sentence])
tags = set([tag for sentence in Y for tag in sentence])
```

Tokenization

Dilakukan tokenisasi terhadap variabel X yang berisi sentence words dan variabel Y yang berisi sentence tags. Tokenisasi dilakukan dengan Tokenizer dari Keras. Dilakukan fit_on_texts untuk membentuk vocabulary index dari setiap kata.

In [5]:
```
# Tokenizing words
word_tokenizer = Tokenizer(lower=True, oov_token='<<OOV>>')
word_tokenizer.fit_on_texts(X)

# Tokenizing tags
tag_tokenizer = Tokenizer(lower=False)
tag_tokenizer.fit_on_texts(Y)
```

Text to Sequence

Hasil tokenisasi yang masih berupa kata kemudian diubah menjadi sekuens integer menggunakan texts_to_sequences. Hasil yang didapatkan adalah hasil perubahan setiap kata menjadi indeksnya

pada kamus dari Tokenizer. Untuk Tokenizer yang digunakan ke tag, ditambahkan satu entry '<>' = 0 karena akan dilakukan padding dengan nilai 0.

In [6]:
```
# Words sequencing
X_sequence = word_tokenizer.texts_to_sequences(X)

# Tags sequencing
Y_sequence = tag_tokenizer.texts_to_sequences(Y)

# Adding PAD tag to dictionary
tag_tokenizer.word_index['<<PAD>>'] = 0
```

Splitting Training Data and Test Data

Dilakukan split data menjadi training data dan testing data. Didefinisikan MAX_LENGTH untuk ukuran data yang akan dimasukkan ke network. Splitting dilakukan dengan train_test_split.

In [7]:
```
X_train_, X_test_, Y_train_, Y_test_ = train_test_split(X_sequence, Y_sequence, t

# Defining input layer size
MAX_LENGTH = len(max(X_train_, key=len))
```

Sequence Padding

Karena Keras membutuhkan ukuran yang didefinisikan lebih dulu, dilakukan padding hingga MAX_LENGTH untuk menyamakan ukuran setiap data.

In [8]:
```
X_train_ = pad_sequences(X_train_, maxlen=MAX_LENGTH, padding='pre')
X_test_ = pad_sequences(X_test_, maxlen=MAX_LENGTH, padding='pre')
Y_train_ = pad_sequences(Y_train_, maxlen=MAX_LENGTH, padding='pre')
Y_test_ = pad_sequences(Y_test_, maxlen=MAX_LENGTH, padding='pre')
```

One-Hot Encoding

One-Hot Encoding dilakukan untuk merepresentasikan index tag menjadi list of bit sehingga dapat lebih dimengerti oleh model machine learning. One-Hot Encoding dilakukan menggunakan to_categorical.

In [9]:
```
Y_train_ = to_categorical(Y_train_)
```

Defining Network Architecture

Arsitektur jaringan adalah sekuensial. Pada pembelajaran digunakan Bidirectional LSTM karena lebih baik untuk sequence classification problem. Ditambahkan pula layer Dropout untuk mengurangi overfitting. Fungsi aktivasi yang digunakan adalah softmax.

In [10]:
```
bi_lstm_model = Sequential()
bi_lstm_model.add(InputLayer(input_shape=(MAX_LENGTH,)))
bi_lstm_model.add(Embedding(len(word_tokenizer.word_index), 128))
bi_lstm_model.add(Bidirectional(LSTM(256, return_sequences=True)))
bi_lstm_model.add(Dropout(0.1))
bi_lstm_model.add(TimeDistributed(Dense(len(tag_tokenizer.word_index))))
bi_lstm_model.add(Activation('softmax'))
bi_lstm_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 271, 128)          7609472
_________________________________________________________________
bidirectional (Bidirectional (None, 271, 512)          788480
_________________________________________________________________
dropout (Dropout)            (None, 271, 512)          0
_________________________________________________________________
time_distributed (TimeDistri (None, 271, 13)           6669
_________________________________________________________________
activation (Activation)      (None, 271, 13)           0
=================================================================
Total params: 8,404,621
Trainable params: 8,404,621
Non-trainable params: 0
_________________________________________________________________
```

Compile and Training Network

Jaringan kemudian di-compile dan dilakukan pelatihan. Jumlah epoch yang digunakan adalah EPOCH_COUNT yaitu 3 karena pada eksperimen, 3 epoch telah menghasilkan akurasi yang cukup tinggi.

In [11]:
```
bi_lstm_model.compile(loss='categorical_crossentropy', optimizer=Adam(0.001), met
bi_lstm_model.fit(X_train_, Y_train_, batch_size=BATCH_SIZE, epochs=EPOCH_COUNT,
```

```
Epoch 1/3
432/432 [==============================] - 120s 277ms/step - loss: 0.1333 - acc
uracy: 0.9632 - val_loss: 0.0182 - val_accuracy: 0.9948
Epoch 2/3
432/432 [==============================] - 115s 267ms/step - loss: 0.0116 - acc
uracy: 0.9965 - val_loss: 0.0094 - val_accuracy: 0.9969
Epoch 3/3
432/432 [==============================] - 118s 273ms/step - loss: 0.0070 - acc
uracy: 0.9977 - val_loss: 0.0085 - val_accuracy: 0.9972
```

Out[11]:
```
<tensorflow.python.keras.callbacks.History at 0x23dc2d20b50>
```

Saving model and Tokenizer

Model yang dihasilkan kemudian disimpan ke sebuah file h5 dan Tokenizer serta MAX_LENGTH

disimpan ke file pickle.

In [12]:
```
bi_lstm_model.save("model/bi_lstm_model.h5")

pickle_files = [word_tokenizer, tag_tokenizer, MAX_LENGTH]

if not os.path.exists('PickledData/'):
    os.makedirs('PickledData/')

with open('PickledData/data.pkl', 'wb') as f:
    pickle.dump(pickle_files, f)
```

Doing Pos Tagging

Test Data

In [13]:
```
test_samples = [
    ['skyrim', 'nt', 'good', 'game', 'without', 'mods', 'fact', 'might', 'pay', '
    ['addictive', 'game', 'ever', 'made'],
    ['counter', 'strike', 'even', 'fight', 'highly', 'trained', 'american', 'anti
]
```

Load Model and Tokenizer

In [14]:
```
def load(path):
    with open(path, 'rb') as f:
        word2int, tag2int, MAX_LENGTH = pickle.load(f)
        return word2int, tag2int, MAX_LENGTH
```

Tagging Method

Untuk melakukan tagging, digunakan method pos_tag(). Algoritma diawali dengan me-load model dari file h5 dan me-load variabel Tokenizer serta MAX_LENGTH dari file pickle. Input yang berupa list of list of token kemudian diubah menjadi sekuens integer menggunakan Tokenizer yang di-load. Sekuens yang didapat di-padding hingga sebesar MAX_LENGTH. Model kemudian melakukan prediksi dengan menggunakan predict().

Kamus tag yang ada pada tag_tokenizer (Tokenizer POS TAG) di-reverse sehingga key menjadi index dan value menjadi kata POS TAG. Hasil tag didapatkan dengan memanggil sequences_to_tags() dengan parameter hasil prediksi dan kamus yang telah di-reverse. sequence_to_tags() akan mengembalikan value dari key dengan key berupa indeks prediksi dengan probabilitas terbesar.

Setelah hasil tag didapatkan, dilakukan pengecekan panjang sentence asli dengan panjang sentence tag. Jika sentence tag yang didapatkan kurang panjang, dilakukan penambahan 'NOUN' di depan karena berdasarkan percobaan terdapat beberapa kasus 'NOUN' di awal sentence

hilang. Setelah panjang keduanya sama, dibentuk tuple (word, tag) yang disimpan pada list result. Hasil list result kemudian dikembalikan sebagai hasil sentence yang telah di-tag.

In [21]:
```
def pos_tag(token_list):
    bi_lstm_model = load_model("model/bi_lstm_model.h5")
    word_tokenizer, tag_tokenizer, MAX_LENGTH = load('PickledData/data.pkl')

    input_sequences = word_tokenizer.texts_to_sequences(token_list)
    input_sequences = pad_sequences(input_sequences, maxlen=MAX_LENGTH, padding='
    predictions = bi_lstm_model.predict(input_sequences)

    reverse_tag_map = dict(map(reversed, tag_tokenizer.word_index.items()))
    tag_result = sequences_to_tags(predictions, reverse_tag_map)

    result = []
    for i in range(len(token_list)):
        if (len(token_list[i]) != len(tag_result[i])):
            diff = len(token_list[i]) - len(tag_result[i])
            if (diff > 0):
                for j in range(diff):
                    tag_result[i].insert(0, 'NOUN')
        result.append(list(zip(token_list[i], tag_result[i])))

    return result

def sequences_to_tags(predictions, tag_map):
    tag_result = []
    for prediction in predictions:
        not_padding = False
        tag_list = []
        for index in prediction:
            tag = tag_map[np.argmax(index)]
            if (tag != "<<PAD>>"):
                not_padding = True
            if (not_padding):
                tag_list.append(tag)

        tag_result.append(tag_list)

    return tag_result
```

Doing Pos Tag

In [22]:
```
result = pos_tag(test_samples)
for res in result:
    print(res)
```

```
[('skyrim', 'NOUN'), ('nt', 'VERB'), ('good', 'ADJ'), ('game', 'NOUN'), ('witho
ut', 'ADP'), ('mods', 'NOUN'), ('fact', 'NOUN'), ('might', 'VERB'), ('pay', 'VE
RB'), ('mods', 'NOUN'), ('make', 'VERB'), ('bugthesda', 'NOUN'), ('s', 'NOUN'),
('game', 'NOUN'), ('playable', 'NOUN'), ('rubbish', 'NOUN')]
[('addictive', 'ADJ'), ('game', 'NOUN'), ('ever', 'ADV'), ('made', 'VERB')]
[('counter', 'ADV'), ('strike', 'VERB'), ('even', 'ADV'), ('fight', 'VERB'),
('highly', 'ADV'), ('trained', 'VERB'), ('american', 'ADJ'), ('antiterrorist',
'NOUN'), ('team', 'NOUN'), ('using', 'VERB'), ('latest', 'ADJ'), ('military',
'NOUN'), ('technology', 'NOUN'), ('battle', 'NOUN'), ('group', 'NOUN'), ('reall
y', 'ADV'), ('madmen', 'ADJ'), ('possessing', 'NOUN'), ('crude', 'ADJ'), ('bom
b', 'NOUN'), ('surplus', 'NOUN'), ('ussr', 'NOUN'), ('s', 'NOUN'), ('army', 'NO
UN'), ('supplies', 'NOUN'), ('despite', 'ADP'), ('training', 'NOUN'), ('technol
ogy', 'NOUN'), ('terrorists', 'NOUN'), ('still', 'ADV'), ('good', 'ADJ'), ('cha
nce', 'NOUN'), ('blowing', 'NOUN'), ('market', 'NOUN'), ('therefore', 'ADV'),
('much', 'ADJ'), ('like', 'ADJ'), ('real', 'ADJ'), ('life', 'NOUN'), ('game',
'NOUN'), ('currently', 'ADV'), ('full', 'ADJ'), ('hackers', 'NOUN'), ('fly', 'N
OUN'), ('top', 'NOUN'), ('map', 'NOUN'), ('unless', 'ADP'), ('hack', 'NOUN'),
('like', 'ADJ'), ('getting', 'NOUN'), ('aerial', 'ADJ'), ('teabag', 'NOUN'),
('please', 'VERB'), ('play', 'NOUN'), ('better', 'ADJ'), ('counter', 'NOUN'),
('strike', 'NOUN'), ('sauce', 'NOUN'), ('counter', 'NOUN'), ('strike', 'NOUN'),
('go', 'VERB'), ('game', 'NOUN'), ('game', 'NOUN'), ('day', 'NOUN'), ('exists',
'VERB'), ('historical', 'ADJ'), ('purposes', 'NOUN'), ('remember', 'VERB'), ('t
imes', 'NOUN'), ('internet', 'NOUN'), ('cafe', 'NOUN'), ('mosque', 'NOUN'), ('f
ull', 'ADJ'), ('game', 'NOUN')]
```