

# **EKSPLORASI DAN ANALISIS PERBANDINGAN METODE N-GRAM DAN NEURALL BASED LANGUAGE MODEL**

**Laporan Tugas Eksperimen I**

**Kelas MK Pemrosesan Bahasa Alami (CII4G3)**



**Disusun Oleh :**

**Muhammad Raihan Muhith (1301184245)**

**Program Studi Sarjana Informatika**

**Fakultas Informatika**

**Universitas Telkom**

**Bandung**

**2022**

## DAFTAR ISI

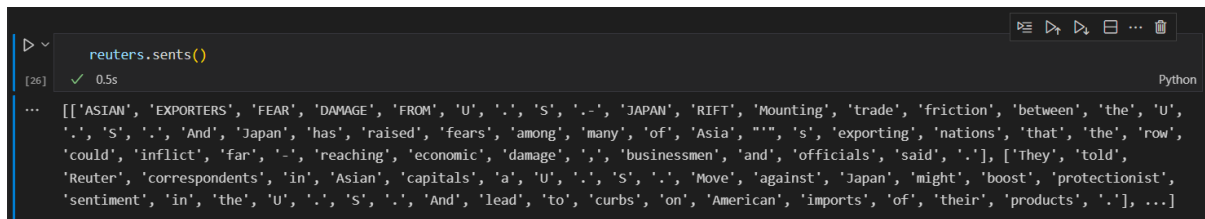
DAFTAR ISI.....	2
1. Pendahuluan.....	3
2. Persiapan Data .....	3
3. Membangun Model Bahasa N-Gram.....	3
4. Membangun Model Bahasa Neural Based (LSTM) .....	5
4.1. Tokenisasi.....	5
4.2. Generasi Predictor dan Label .....	6
4.3. Identifikasi Network Layer dan Training Model .....	6
4.4. Penggenerasian Teks Otomatis .....	6
5. Kesimpulan.....	7
DAFTAR PUSTAKA .....	7
LAMPIRAN.....	8

## 1. Pendahuluan

Pendekatan-pendekatan untuk membangun model bahasa sangatlah banyak, pendekatan yang dilakukan pada tugas eksperimen ini adalah dengan menggunakan pendekatan N-Gram dan *Neural Based*, sedangkan *task* yang akan dikerjakan adalah Automatic Text Generation bahasa inggris. Korpus yang digunakan adalah *reuters*, *reuters* merupakan korpus yang telah disediakan oleh library NLTK yang mengandung sebanyak 10,788 artikel berita dengan total 1.3 juta kata.

## 2. Persiapan Data

Langkah pertama pada proses pengerjaan tugas ini tentu saja dengan mempersiapkan data terlebih dahulu yang akan digunakan sebagai korpus pelatihan model, telah dijelaskan sebelumnya bahwa korpus yang digunakan merupakan korpus yang telah disediakan oleh library NLTK, sehingga untuk mengaksesnya hanya perlu mengunduh library NLTK dan setelah itu import dengan cara `from nltk.corpus import reuters`. Berikut ditampilkan korpus yang akan digunakan untuk pelatihan model pada Gambar 1.



```
reuters.sents()
[26] ✓ 0.5s
Python
... [['ASIAN', 'EXPORTERS', 'FEAR', 'DAMAGE', 'FROM', 'U', '.', 'S', '...', 'JAPAN', 'RIFT', 'Mounting', 'trade', 'friction', 'between', 'the', 'U',
      '.', 'S', '...', 'And', 'Japan', 'has', 'raised', 'fears', 'among', 'many', 'of', 'Asia', '...', 's', 'exporting', 'nations', 'that', 'the', 'row',
      'could', 'inflict', 'far', '-', 'reaching', 'economic', 'damage', '...', 'businessmen', 'and', 'officials', 'said', '.'], ['They', 'told',
      'Reuter', 'correspondents', 'in', 'Asian', 'capitals', 'a', 'U', '.', 'S', '...', 'Move', 'against', 'Japan', 'might', 'boost', 'protectionist',
      'sentiment', 'in', 'the', 'U', '.', 'S', '...', 'And', 'lead', 'to', 'curbs', 'on', 'American', 'imports', 'of', 'their', 'products', '.'], ...]
```

Gambar 1. Korpus

Setelah korpus terdefiniskan, langkah selanjutnya adalah proses membersihkan data. proses ini bertujuan agar data yang dilatih menjadi berkualitas, hal ini dilakukan agar mendapatkan hasil akurasi yang memuaskan. Proses cleaning data yang dilakukan sebatas pembersihan dari karakter-karakter yang tidak dibutuhkan dan penormalisasian teks menjadi huruf kecil.

## 3. Membangun Model Bahasa N-Gram

Pada dasarnya, kunci dari pendekatan N-Gram dalam melakukan Automatic Text Generation adalah dengan menghitung probabilitas himpunan kata yang muncul setelah given N yang diberikan dan dilakukan sampai sudah tidak ada lagi kata yang bisa dilanjutkan. Model N-Gram yang dibangun merupakan model bahasa trigrams dari korpus reuters. Dalam menunjang proses pembuatan model, penulis menggunakan bantuan library NLTK, karena library tersebut sudah menyediakan fungsi bigrams dan trigrams. Pada Gambar 2. telah ditampilkan code yang dibangun untuk menghasilkan model bahasa trigrams.

Kode yang dibangun merupakan kode yang sederhana, pertama korpus di split menjadi trigram menggunakan fungsi `trigrams()` dan kemudian dihitung jumlah kemunculan trigram pada korpus. Hasil hitung tersebut, kemudian digunakan untuk menghitung probabilitas dengan *given* 2 kata sebelumnya [1]. Berikut akan ditampilkan tabel perhitungan probabilitas kemunculan kata apabila kita menggunakan given “The President” untuk memprediksi kata selanjutnya.

**Tabel 1.** Probabilitas kata dengan given "The President"

Words	P("word"   "The President")
"of"	0.25
"And"	0.14
"is"	0.05
"had"	0.03
"to"	0.11
"added"	0.02
"remains"	0.02
...	...

Dari Tabel 1. apabila dengan asumsi probabilitas perhitungan sesuai, maka kata yang paling banyak digunakan setelah kata "the president" pada korpus reuters ialah "of". Berhubungan dengan *task* Automatic Text Generation, dalam proses melengkapi kalimat, sistem bisa mengambil kata dengan probabilitas tertinggi atau secara random, tergantung dengan keperluannya. Dalam pengerjaan tugas ini, penulis membangun penggenerasian teks dengan cara mengambil kata selanjutnya secara random sampai tidak ada lagi probabilitas yang perlu dihitung, atau dengan kata lain sudah tidak ada kata yang dapat melengkapi kalimat tersebut.

Setelah probabilitas setiap kemunculan kata trigrams pada korpus dihitung, maka langkah selanjutnya adalah membangun algoritma Automatic Text Generation-nya, proses ini memiliki input sequence of text sepanjang 2 kata (karena trigram), dan output merupakan kalimat yang telah tergenerasi secara otomatis dari given input. Eksperimen dilakukan dengan mencoba menggenerasi teks dengan *given* input "the president", "the government", dan kata yang memiliki kemungkinan tidak ada dalam korpus (diluar vocab). Pada Gambar 2. ditampilkan pengujian yang telah dilakukan terhadap ketiga kasus tersebut.

```
Input : 'The President'
Output : the president and general manager for economic growth after 1990 .

Input : 'The Government'
Output : the government ' s meeting was delayed up to 150 yen .

Input : 'i like'
Output : {}
```

**Gambar 2.** Text Generated dengan Trigram LM

Dapat dilihat bahwa pada Gambar 3. ketika sistem mendapatkan input "I like", hasil yang dikeluarkan tidak ada, hal ini terjadi karena potongan kata "I like" tidak ada dalam korpus. Hal ini merupakan salah satu kekurangan dari model N-Gram, yaitu kondisi dimana potongan sebanyak N-1 kata tidak ada dalam korpus sehingga tidak memiliki probabilitas. Hal ini juga yang mengakibatkan proses penggenerasian teks terhenti. Kekurangan lainnya adalah panjang input yang dibutuhkan harus menyesuaikan berapa N yang telah kita definisikan dalam pembangunan model, sehingga dapat dikatakan bahwa model ini tidak fleksibel.

## 4. Membangun Model Bahasa Neural Based (LSTM)

Pendekatan Neural Based dalam membangun model bahasa merupakan suatu hal yang populer, karena dengan pendekatan ini model yang dihasilkan dapat mengatasi masalah-masalah yang timbul dari membangun model bahasa menggunakan pendekatan N-Gram, salah satu contohnya adalah masalah *sparsity*, yaitu kondisi dimana potongan kata sepanjang N tidak muncul dalam korpus sehingga menimbulkan probabilitas menjadi nol. Penulis menggunakan algoritma Long Short Term Memory dalam melakukan pelatihan model, karena metode ini sangat baik dalam melakukan training untuk jenis inputan yang sangat besar, terlebih algoritma ini terbukti lebih baik dari algoritma RNN [2].

Berhubung algoritma ini merupakan salah satu algoritma deep learning dan penulis menggunakan bantuan library Keras untuk membantu pembangunan model, maka dalam proses membangunnya ada beberapa langkah yang harus dilakukan sebelum pembuatan model agar data inputan compatible dengan yang diminta oleh LSTM nantinya, berikut akan dijelaskan secara singkat kelanjutannya.

### 4.1. Tokenisasi

Pemodelan bahasa membutuhkan data inputan sequence of text, karena hal ini berhubungan dengan tujuannya yaitu memprediksi kata/token berikutnya. Langkah Tokenisasi ini adalah proses ekstraksi token (istilah/kata) dari sebuah korpus. Library Keras memiliki model bawaan untuk tokenisasi yang dapat digunakan untuk mendapatkan token dan indeksinya di korpus. Setelah langkah ini dijalankan, setiap dokumen akan berupa urutan token (*sequence of Tokens*). Sequence of Tokens merupakan representasi angka dari sequence of text berdasarkan index kata pada korpus. Berikut akan ditampilkan contoh data setelah dijalankan langkah ini pada Tabel 2.

**Tabel 2.** Sequence of Tokens

Sequence of Text	Sequence of Tokens
"The"	[24]
"The kid"	[24, 16]
"The kid is"	[24, 16, 47]
"The kid is in"	[24, 16, 47, 432]
"The kid is in the"	[24, 16, 47, 432, 24]
"The kid is in the backyard"	[24, 16, 47, 432, 24, 312]
"The kid is in the backyard playing"	[24, 16, 47, 432, 24, 312, 131]
...	...

Pada proses persiapan data (*cleaning*), penulis mencoba untuk TIDAK menggunakan korpus yang berisi kumpulan dari *sequence of text* melainkan menggabungkan kumpulan *sequence of text* tersebut menjadi satu kesatuan yang utuh. Sehingga pada tabel 2. Jika diteruskan, yang dihasilkan oleh baris terakhir dari tabel tersebut akan sepanjang *length of corpus*.

## 4.2. Generasi Predictor dan Label

Setelah *Sequence of Text* diubah menjadi *Sequence of Tokens*, sebelum mulai melatih model, kita perlu pad urutan dan membuat panjang mereka sama, lalu untuk memasukkan data ini ke dalam model training, kita perlu membuat predictor dan label. Penulis membuat urutan N-gram sebagai prediktor dan kata berikutnya dari N-gram sebagai label. Berikut ilustrasinya berdasarkan tabel 2. apabila masih dalam berbentuk *sequence of text*.

Predictor	Label
The	Kid
The kid	Is
The kid is	In
The kid is in	The
The kid is in the	Backyard
The kid is in the backyard	Playing
...	...

## 4.3. Identifikasi Network Layer dan Training Model

Setelah memperoleh vektor input X dan vektor label Y yang dapat digunakan untuk tujuan pelatihan model, langkah selanjutnya adalah mendefinisikan parameter-parameter apa serta kombinasi layer apa yang akan digunakan untuk membangun Neural Network. Penulis menggunakan tiga jenis layer, yaitu input layer, hidden layer, dan output layer. Input layer berupa Embedding, hidden layer berupa satu LSTM dan dropout layer, dan output layer berupa dense layer. Berikut ditampilkan ringkasan model yang telah dibangun dengan total tiga jenis layer pada Gambar 3.

```
Model: "sequential"
Layer (type)                Output Shape              Param #
=====
embedding (Embedding)       (None, 136, 10)          309520
lstm (LSTM)                  (None, 100)               44400
dropout (Dropout)           (None, 100)               0
dense (Dense)                (None, 30952)             3126152
=====
Total params: 3,480,072
Trainable params: 3,480,072
Non-trainable params: 0
```

**Gambar 3.** Ringkasan Model

Setelah semua aspek telah diinisialisasi, langkah selanjutnya adalah melatih model tersebut. Penulis melakukan pelatihan model dengan 100 epoch. Hasil pelatihan model menunjukkan bahwa model berhasil mengklasifikasikan predictor dan label sebesar 72.31%.

## 4.4. Penggenerasian Teks Otomatis

Sama seperti pada pendekatan N-Gram, setelah model sudah selesai dibangun dan dilatih, maka langkah terakhir yaitu membangun algoritma Automatic Text Generation-nya. Algoritma

yang dibangun membutuhkan inputan berupa kata/kalimat inisial (bebas panjangnya) dan jumlah kata yang ingin digenerasi. Output dari algoritma ini tentunya adalah kalimat yang telah tergenerasi secara otomatis berdasarkan pemahaman model dari hasil pelatihan sebelumnya. Berikut akan ditampilkan hasil dari membangun Neural Based Language Model pada Gambar 4.

```
Inisial Kata : the
jumlah kata yang ingin digenerasi : 10
Hasil Generasi : The Company Said It Is Also Ready To Expand Effect Here

Inisial Kata : the government
jumlah kata yang ingin digenerasi : 20
Hasil Generasi : The Government Agency Said That Harcourt Mining On International Debt With Trade U S And Lead To Curbs To Block Imports Of

Inisial Kata : i like
jumlah kata yang ingin digenerasi : 5
Hasil Generasi : I Like Details And Sugar Reserves In
```

**Gambar 4.** Text Generated dengan Neural Based LM

## 5. Kesimpulan

Neural Based Language Model merupakan salah satu alternatif lain yang dapat memperbaiki masalah-masalah yang ditimbulkan oleh N-Gram Language model, dapat dilihat pada Gambar 5, tidak seperti hasil pada trigram, Neural Based LM dapat memberikan hasil generasi dari berbagai kombinasi inputan, seperti panjang sequence of text dan panjang kalimat yang ingin di generasi. Hal ini dikarenakan dengan menggunakan pendekatan Neural Based, proses pelatihan model dengan menggunakan Neural Based menggunakan looping (rekursif) dalam arsitektur jaringan saraf yang bertindak sebagai 'keadaan memori' neuron. Keadaan ini memungkinkan neuron yang memiliki kemampuan untuk mengingat apa yang telah dipelajari sejauh ini. Berbeda dengan N-Gram, kesuksesan model dengan pendekatan tersebut bergantung dari seberapa banyak N yang didefinisikan, namun apabila N terlalu sedikit, maka hasil generasi tidak bagus, dan apabila N terlalu besar, masalah *sparsity* akan semakin memburuk. Sehingga dari hasil Tugas Eksperimen 1 ini, penulis dapat menyimpulkan bahwa Neural Based LM dapat dikatakan lebih baik dari N-Gram LM, karena lebih fleksibel dan tidak ada masalah *sparsity*.

## DAFTAR PUSTAKA

- [1] M. Sanad, Z. Rizvi, L. To, and B. This, "A Comprehensive Guide to Build your own Language Model in Python !," 8 August, 2019.  
<https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-language-model-nlp-python-code/> (accessed May 26, 2022).
- [2] S. Bansal, "Beginners Guide to Text Generation using LSTMs," 2019.  
<https://www.kaggle.com/code/shivamb/beginners-guide-to-text-generation-using-lstms/notebook>.

## LAMPIRAN

### a. Kode N-Gram Language Modeling

```
from nltk import trigrams, bigrams
from collections import defaultdict
import os
import pandas as pd
from nltk.corpus import reuters
import random

# Create a placeholder for model
model = defaultdict(lambda: defaultdict(lambda: 0))

# Count frequency of co-occurrence
for sentence in reuters.sents():
    for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
        model[(w1, w2)][w3] += 1

# Let's transform the counts to probabilities
for w1_w2 in model:
    total_count = float(sum(model[w1_w2].values()))
    for w3 in model[w1_w2]:
        model[w1_w2][w3] /= total_count

# starting words
text = ["the", "president"]
sentence_finished = False

# while not sentence_finished:
while not sentence_finished:
    # select a random probability threshold
    r = random.random()
    accumulator = .0

    for word in model[tuple(text[-2:]).keys():
        accumulator += model[tuple(text[-2:])[word]
        # select words that are above the probability threshold
        if accumulator >= r:
            text.append(word)
            break

    if text[-2:] == [None, None]:
        sentence_finished = True
```



## b. Kode Neural Based Language Modeling

```
import tensorflow as tf

from keras.preprocessing.sequence import pad_sequences
from keras.layers import Embedding, LSTM, Dense, Dropout
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential

from tensorflow.keras.utils import to_categorical

from nltk.corpus import reuters

import os, string
import pandas as pd

import numpy as np

list_headlines = []

for word in reuters.sents():
    temp = ''
    temp = ' '.join([t for t in word if t])
    list_headlines.append(temp)

def clean_text(txt):
    txt = "".join(v for v in txt if v not in string.punctuation).lower()
    txt = txt.encode("utf8").decode("ascii", 'ignore')
    return txt

corpus = [clean_text(x) for x in list_headlines]

tokenizer = Tokenizer()
```

```
def get_sequence_of_tokens(corpus):
    ## tokenization
    tokenizer.fit_on_texts(corpus)
    total_words = len(tokenizer.word_index) + 1

    ## convert data to sequence of tokens
    input_sequences = []
    for line in corpus:
        token_list = tokenizer.texts_to_sequences([line])[0]
        for i in range(1, len(token_list)):
            n_gram_sequence = token_list[:i+1]
            input_sequences.append(n_gram_sequence)
    return input_sequences, total_words

inp_sequences, total_words = get_sequence_of_tokens(corpus)

def generate_padded_sequences(input_sequences):
    max_sequence_len = max([len(x) for x in input_sequences])
    input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

    predictors, label = input_sequences[:, :-1], input_sequences[:, -1]
    label = to_categorical(label, num_classes=total_words)
    return predictors, label, max_sequence_len

predictors, label, max_sequence_len = generate_padded_sequences(inp_sequences)

def create_model(max_sequence_len, total_words):
    input_len = max_sequence_len - 1
    model = Sequential()

    # Add Input Embedding Layer
    model.add(Embedding(total_words, 10, input_length=input_len))
```

```

# Add Hidden Layer 1 - LSTM Layer
model.add(LSTM(100))
model.add(Dropout(0.1))

# Add Output Layer
model.add(Dense(total_words, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

return model

model = create_model(max_sequence_len, total_words)
model.fit(predictors, label, epochs=100, verbose=1)

def generate_text(seed_text, next_words, model, max_sequence_len):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
        predicted = model.predict(token_list, verbose=0)
        predicted = np.argmax(predicted,axis=1)

        output_word = ""
        for word,index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " "+output_word
    return seed_text.title()

```

**LINK VIDEO :**

[https://youtu.be/lKN\\_b8D6bKQ](https://youtu.be/lKN_b8D6bKQ)