Lampiran

Kode Program

index.html

```
<!DOCTYPE html>
<html >
<head>
  <meta charset="UTF-8">
  <title>Simple Neural Machine Translation</title>
  <link href='https://fonts.googleapis.com/css?family=Pacifico'
rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Arimo'
rel='stylesheet' type='text/css'>
<link href='https://fonts.googleapis.com/css?family=Hind:300'
rel='stylesheet' type='text/css'>
<link
href='https://fonts.googleapis.com/css?family=Open+Sans+Condensed:300'
rel='stylesheet' type='text/css'>
<link rel="stylesheet" href="{{ url_for('static',
filename='css/style.css') }}">

</head>

<body style="background: #311b92;">
<div class="header">
  <!--<div class="container-fluid">-->
  <div>
    <h2>Kawi Language to Indonesian Language Neural Machine Translation
Version 0.0 </h2>
  </div>
</div>

 <div class="login">
 <h1></h1>
       <!-- Main Input For Receiving Query to our ML -->
    <form action='/translate' method="POST">
      </br>
        </br>
          </br>
```

```
      <input type="text" name="src_language" placeholder="Input Kawi
Sentence Here" required="required" />

        <button type="submit" class="btn btn-primary btn-block btn-
large">Translate</button>
    </form>
   <br>
 <!-- {% if prediction_text%}-->
     <p> Hasil penerjemah adalah : " {{ prediction_text }}"</p>
 <!-- {% endif %} -->

 </div>


</body>
</html>
```

style.css

```
@import url(https://fonts.googleapis.com/css?family=Open+Sans);
.btn { display: inline-block; *display: inline; *zoom: 1; padding: 4px
10px 4px; margin-bottom: 0; font-size: 13px; line-height: 18px; color:
#333333; text-align: center;text-shadow: 0 1px 1px rgba(255, 255, 255,
0.75); vertical-align: middle; background-color: #f5f5f5; background-
image: -moz-linear-gradient(top, #ffffff, #e6e6e6); background-image: -
ms-linear-gradient(top, #ffffff, #e6e6e6); background-image: -webkit-
gradient(linear, 0 0, 0 100%, from(#ffffff), to(#e6e6e6)); background-
image: -webkit-linear-gradient(top, #ffffff, #e6e6e6); background-
image: -o-linear-gradient(top, #ffffff, #e6e6e6); background-image:
linear-gradient(top, #ffffff, #e6e6e6); background-repeat: repeat-x;
filter:
progid:dximagetransform.microsoft.gradient(startColorstr=#ffffff,
endColorstr=#e6e6e6, GradientType=0); border-color: #e6e6e6 #e6e6e6
#e6e6e6; border-color: rgba(0, 0, 0, 0.1) rgba(0, 0, 0, 0.1) rgba(0, 0,
0, 0.25); border: 1px solid #e6e6e6; -webkit-border-radius: 4px; -moz-
border-radius: 4px; border-radius: 4px; -webkit-box-shadow: inset 0 1px
0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.05); -moz-box-
shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0,
```

```
0.05); box-shadow: inset 0 1px 0 rgba(255, 255, 255, 0.2), 0 1px 2px
rgba(0, 0, 0, 0.05); cursor: pointer; *margin-left: .3em; }
.btn:hover, .btn:active, .btn.active, .btn.disabled, .btn[disabled] {
background-color: #e6e6e6; }
.btn-large { padding: 9px 14px; font-size: 15px; line-height: normal; -
webkit-border-radius: 5px; -moz-border-radius: 5px; border-radius: 5px;
}
.btn:hover { color: #333333; text-decoration: none; background-color:
#e6e6e6; background-position: 0 -15px; -webkit-transition: background-
position 0.1s linear; -moz-transition: background-position 0.1s linear;
-ms-transition: background-position 0.1s linear; -o-transition:
background-position 0.1s linear; transition: background-position 0.1s
linear; }
.btn-primary, .btn-primary:hover { text-shadow: 0 -1px 0 rgba(0, 0, 0,
0.25); color: #ffffff; }
.btn-primary.active { color: rgba(255, 255, 255, 0.75); }
.btn-primary { background-color: #4a77d4; background-image: -moz-
linear-gradient(top, #6eb6de, #4a77d4); background-image: -ms-linear-
gradient(top, #6eb6de, #4a77d4); background-image: -webkit-
gradient(linear, 0 0, 0 100%, from(#6eb6de), to(#4a77d4)); background-
image: -webkit-linear-gradient(top, #6eb6de, #4a77d4); background-
image: -o-linear-gradient(top, #6eb6de, #4a77d4); background-image:
linear-gradient(top, #6eb6de, #4a77d4); background-repeat: repeat-x;
filter:
progid:dximagetransform.microsoft.gradient(startColorstr=#6eb6de,
endColorstr=#4a77d4, GradientType=0);  border: 1px solid #3762bc; text-
shadow: 1px 1px 1px rgba(0,0,0,0.4); box-shadow: inset 0 1px 0
rgba(255, 255, 255, 0.2), 0 1px 2px rgba(0, 0, 0, 0.5); }
.btn-primary:hover, .btn-primary:active, .btn-primary.active, .btn-
primary.disabled, .btn-primary[disabled] { filter: none; background-
color: #4a77d4; }
.btn-block { width: 100%; display:block; }

* { -webkit-box-sizing:border-box; -moz-box-sizing:border-box; -ms-box-
sizing:border-box; -o-box-sizing:border-box; box-sizing:border-box; }

html { width: 100%; height:100%; overflow:hidden; }

body {
    width: 100%;
    height:100%;
```

```
    font-family: 'Helvetica';
    background: #311b92;
    color: #fff;
    font-size: 24px;
    text-align:center;
    letter-spacing:1.4px;

}
.login {
    position: absolute;
    top: 40%;
    left: 50%;
    margin: -150px 0 0 -150px;
    width:400px;
    height:400px;
    padding-top: 10px;
}

.header {
    position: fixed;
    top:10px;
    margin-right:auto;
}

.login h1 { color: #fff; text-shadow: 0 0 10px rgba(0,0,0,0.3); letter-
spacing:1px; text-align:center; }

.header h1, h2, h3, h4, h5, h6 {color: #fff; text-shadow: 0 0 10px
rgba(0,0,0,0.3); letter-spacing:1px; text-align:center;}
input {
    width: 100%;
    margin-bottom: 10px;
    background: rgba(0,0,0,0.3);
    border: none;
    outline: none;
    padding: 100px;
    font-size: 13px;
    color: #fff;
    text-shadow: 1px 1px 1px rgba(0,0,0,0.3);
    border: 10px solid rgba(0,0,0,0.3);
    border-radius: 200px;
```

```
    box-shadow: inset 0 -5px 45px rgba(100,100,100,0.2), 0 1px 1px
rgba(255,255,255,0.2);
    -webkit-transition: box-shadow .5s ease;
    -moz-transition: box-shadow .5s ease;
    -o-transition: box-shadow .5s ease;
    -ms-transition: box-shadow .5s ease;
    transition: box-shadow .5s ease;
}

.center {
    text-align: center;
    }


input:focus { box-shadow: inset 0 -5px 45px rgba(100,100,100,0.4), 0
1px 1px rgba(255,255,255,0.2); }
```

model mesin penerjemah.py

```
import warnings
warnings.filterwarnings('ignore')
import string
import re
from unicodedata import normalize
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical
from keras.model mesin penerjemahs import load_model mesin penerjemah
from keras.layers import
LSTM,Dense,Embedding,RepeatVector,TimeDistributed, Activation, Dropout,
Reshape, Flatten, Input
from keras.callbacks import EarlyStopping
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import pandas as pd
```

```
from string import punctuation
import matplotlib.pyplot as plt
from IPython.display import Markdown, display
from gensim.model mesin penerjemahs import Word2Vec

def print(string):
    # Print with Markdowns
    display(Markdown(string))

dataset=pd.read_excel("Dataset_NMT_BKI.xlsx")

corpus_text = '\n'.join(dataset[:1088]['Source Language'])
sentences = corpus_text.split('\n')
sentences = [line.lower().split(' ') for line in sentences]

corpus_text2 = '\n'.join(dataset[:1088]['Target Language'])
sentences2 = corpus_text2.split('\n')
sentences2 = [line.lower().split(' ') for line in sentences2]

def clean(s):
    return [w.strip(',."!?:;()\'') for w in s]
sentences = [clean(s) for s in sentences if len(s) > 0]
sentences2 = [clean(s) for s in sentences2 if len(s) > 0]

from gensim.model mesin penerjemahs import Word2Vec
# import torch

w2v_Kawi = Word2Vec(sentences, vector_size=256, window=2, min_count=1,
workers=4, epochs = 100)
w2v_Kawi.save("w2v_Kawi.model mesin penerjemah")

vectors_Kawi = w2v_Kawi.wv
del w2v_Kawi

total_sentences = 1088

test_proportion = 0

train_test_threshold = int( (1-test_proportion) * total_sentences)
```

```
# Clean the sentences

     #add padding on punctuation
dataset['Source Language']=dataset['Source Language'].apply(lambda x:
re.sub('([.,!?():""''-])', r' \1 ', x))
dataset['Target Language']=dataset['Target Language'].apply(lambda x:
re.sub('([.,!?():""''-])', r' \1 ', x))

dataset['Source Language']=dataset['Source Language'].apply(lambda x:
re.sub('\s{2,}', ' ', x))
dataset['Target Language']=dataset['Target Language'].apply(lambda x:
re.sub('\s{2,}', ' ', x))

      #Lowercase
dataset['Source Language']= dataset['Source Language'].apply(lambda x:
x.lower())
dataset['Target Language']= dataset['Target Language'].apply(lambda x:
x.lower())

dataset['Source Language']=dataset['Source Language'].apply(lambda x:
re.sub('([.,!?():""''-])', '', x))
dataset['Target Language']=dataset['Target Language'].apply(lambda x:
re.sub('([.,!?():""''-])', '', x))

      #Remove extra spaces
dataset['Source Language']=dataset['Source Language'].apply(lambda x:
x.strip())
dataset['Target Language']=dataset['Target Language'].apply(lambda x:
x.strip())

dataset['Source Language']=dataset['Source Language'].apply(lambda x:
re.sub(" +", " ", x))
dataset['Target Language']=dataset['Target Language'].apply(lambda x:
re.sub(" +", " ", x))


dataset.drop(labels = "Id", axis = 1,inplace = True)
dataset.drop(labels = "Corpus Title", axis = 1,inplace = True)
dataset.drop(labels = "Expert", axis = 1,inplace = True)

    # Select one part of the dataset
```

```
dataset = dataset.values
dataset = dataset[:total_sentences]


train, test = dataset[:train_test_threshold],
dataset[train_test_threshold:]


source_str = "KAWI"
target_str = "INDONESIAN"

idx_src = 1
idx_tar = 0

def create_tokenizer(lines):
    # fit a tokenizer
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer

def max_len(lines):
    # max sentence length
    return max(len(line.split()) for line in lines)

def encode_sequences(tokenizer, length, lines):
    # encode and pad sequences
    X = tokenizer.texts_to_sequences(lines) # integer encode sequences
    X = pad_sequences(X, maxlen=length, padding='post') # pad sequences
with 0 values
    return X

def encode_output(sequences, vocab_size):
    # one hot encode target sequence
    ylist = list()
    for sequence in sequences:
        encoded = to_categorical(sequence, num_classes=vocab_size)
        ylist.append(encoded)
    y = np.array(ylist)
    y = y.reshape(sequences.shape[0], sequences.shape[1], vocab_size)
    return y
```

```
# Prepare source tokenizer
src_tokenizer = create_tokenizer(dataset[:, idx_src])
src_vocab_size = len(src_tokenizer.word_index) + 1
src_length = max_len(dataset[:, idx_src])

# Prepare target tokenizer
tar_tokenizer = create_tokenizer(dataset[:, idx_tar])
tar_vocab_size = len(tar_tokenizer.word_index) + 1
tar_length = max_len(dataset[:, idx_tar])

# Prepare training data
trainX = encode_sequences(src_tokenizer, src_length, train[:, idx_src])
trainY = encode_sequences(tar_tokenizer, tar_length, train[:, idx_tar])
trainY = encode_output(trainY, tar_vocab_size)

# Prepare test data
testX = encode_sequences(src_tokenizer, src_length, test[:, idx_src])
testY = encode_sequences(tar_tokenizer, tar_length, test[:, idx_tar])
testY = encode_output(testY, tar_vocab_size)

layer_weight_shape = 256
#embedding_matrix Kawi
embedding_matrix_Kawi = np.zeros((src_vocab_size, layer_weight_shape))
for word, i in src_tokenizer.word_index.items():
    embedding_vector = vectors_Kawi.vector_size
    if embedding_vector is not None:
        embedding_matrix_Kawi[i] = embedding_vector


from tensorflow.keras import layers
from tensorflow.keras.layers import Layer

# class Attention(Layer):
class Attention(Layer):
    def __init__(self,**kwargs):
        super(Attention,self).__init__(**kwargs)

    def build(self,input_shape):
        self.W=self.add_weight(name="att_weight",shape=(input_shape[-
1],1),initializer="normal")
```

```
        self.b=self.add_weight(name="att_bias",shape=(input_shape[1],1)
,initializer="zeros")
        super(Attention, self).build(input_shape)

    def call(self,x):
        et=K.squeeze(K.tanh(K.dot(x,self.W)+self.b),axis=-1)
        at=K.softmax(et)
        at=K.expand_dims(at,axis=-1)
        output=x*at
        return K.sum(output,axis=1)

    def compute_output_shape(self,input_shape):
        return (input_shape[0],input_shape[-1])

    def get_config(self):
        return super(Attention,self).get_config()



import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.layers import concatenate
from keras import initializers,regularizers, constraints
from keras import backend as K
K.clear_session()
latent_dim = 256

#V1
encoder_inputs = keras.Input(shape=(src_length,))
embedding_Kawi = Embedding(src_vocab_size, layer_weight_shape,
weights=[embedding_matrix_Kawi], input_length=src_length,
trainable=True)
encoder1 = embedding_Kawi(encoder_inputs)
encoder2 = LSTM(256, return_sequences= True)(encoder1)
Attention_layer = Attention()(encoder2)
decoder_inputs = RepeatVector(tar_length)(Attention_layer)
decoder2 = LSTM(256,return_sequences=True)(decoder_inputs)
decoder_outputs = TimeDistributed(Dense(tar_vocab_size,
activation='softmax'))(decoder2)
```

```
model mesin penerjemah = keras.Model mesin
penerjemah(inputs=encoder_inputs, outputs=decoder_outputs)
#Retrieve the config
Attention_config = model mesin penerjemah.get_config()

custom_objects = {"Attention": Attention}
with keras.utils.custom_object_scope(custom_objects):
    model mesin penerjemah = keras.Model mesin
penerjemah.from_config(Attention_config)


from tensorflow.keras.optimizers import Adam
opt = Adam (learning_rate= 0.001)
model mesin penerjemah.compile(optimizer=opt,
loss='categorical_crossentropy')


history = model mesin penerjemah.fit(trainX,
          trainY,
          epochs=200    ,
          batch_size=128,
          validation_split=0.1,
          verbose=1,
          callbacks=[
                        EarlyStopping(
                        monitor='val_loss',
                        patience=1000,
                        restore_best_weights=True
                    )
              ])
# Save the model mesin penerjemah in h5 format
model mesin penerjemah.save("translator_model mesin penerjemah.h5")
```

app.py

```
from importlib import import_module
import warnings
warnings.filterwarnings('ignore')
import string
import re
```

```
from unicodedata import normalize
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical
from keras.model mesin penerjemahs import load_model mesin penerjemah
from keras.layers import
LSTM,Dense,Embedding,RepeatVector,TimeDistributed, Activation, Dropout,
Reshape, Flatten, Input
from keras.callbacks import EarlyStopping
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import pandas as pd
from string import punctuation
import matplotlib.pyplot as plt
from IPython.display import Markdown, display
from gensim.model mesin penerjemahs import Word2Vec

# Load libraries
from flask import Flask, render_template, request
import pandas as pd
import tensorflow as tf
import keras
from keras.model mesin penerjemahs import load_model mesin penerjemah
from keras import backend as K
import model mesin penerjemah
from model mesin penerjemah import encode_sequences
from model mesin penerjemah import encode_output
from model mesin penerjemah import src_tokenizer
from model mesin penerjemah import tar_tokenizer
from model mesin penerjemah import src_length
from model mesin penerjemah import tar_length
import requests

app = Flask(__name__)

from tensorflow.keras import layers
from tensorflow.keras.layers import Layer

# class Attention(Layer):
```

```
class Attention(Layer):
    def __init__(self,**kwargs):
        super(Attention,self).__init__(**kwargs)

    def build(self,input_shape):
        self.W=self.add_weight(name="att_weight",shape=(input_shape[-
1],1),initializer="normal")
        self.b=self.add_weight(name="att_bias",shape=(input_shape[1],1)
,initializer="zeros")
        super(Attention, self).build(input_shape)

    def call(self,x):
        et=K.squeeze(K.tanh(K.dot(x,self.W)+self.b),axis=-1)
        at=K.softmax(et)
        at=K.expand_dims(at,axis=-1)
        output=x*at
        return K.sum(output,axis=1)

    def compute_output_shape(self,input_shape):
        return (input_shape[0],input_shape[-1])

    def get_config(self):
        return super(Attention,self).get_config()


model mesin penerjemah = load_model mesin penerjemah ('translator_model
mesin penerjemah2.h5', custom_objects={'Attention': Attention})



def word_for_id(integer, tokenizer):
    # map an integer to a word
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None

def predict_seq(model mesin penerjemah, tokenizer, source):
    prediction = model mesin penerjemah.predict(source, verbose=0)[0]
    integers = [np.argmax(vector) for vector in prediction]
    target = list()
```

```
    for i in integers:
        word = word_for_id(i, tokenizer)
        if word is None:
            break
        target.append(word)
    return ' '.join(target)

def clean(s):
    return [w.strip(',."!?:;()\'') for w in s]



#route

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/translate', methods=['POST'])
def get_translation():
    if request.method == 'POST':
        result = request.form

        KawiSentence = str(result['src_language'])
        KawiSentence = KawiSentence.split('\n')
        KawiSentence = [line.lower().split(' ') for line in
KawiSentence]
        KawiSentence = [clean(s) for s in KawiSentence if len(s) > 0]

        Kawi_sequence = encode_sequences (src_tokenizer, src_length,
KawiSentence)

        indo_sentence = predict_seq(model mesin
penerjemah,tar_tokenizer,Kawi_sequence)

        return render_template('index.html',
prediction_text=indo_sentence)

if __name__ == '__main__':
    app.run(port=5000, debug=True)
```