Modul Praktikum

Kecerdasan Buatan

12

Daftar Isi

TensorFlow – Natural Language Processing

Natural Language Processing

Natural Language Processing (NLP) adalah cabang ilmu komputer dalam AI yang memberikan komputer kemampuan untuk memahami teks atau ucapan seperti manusia. NLP menggabungkan komputasi model berbasis linguistik dari bahasa manusia dengan statistik, machine learning, dan model deep learning.

Mengembangkan model untuk masalah computer vision relatif lebih mudah karena nilai-nilai piksel pada gambar sudah dalam format bilangan numerik yang dapat diolah model. Namun bagaimana dengan teks dan kata-kata?

Salah satu cara yang dapat dilakukan adalah mengonversi setiap huruf pada sebuah kata ke dalam bilangan tertentu misalnya ke dalam format ASCII seperti di bawah.

Namun teknik ini memiliki masalah tersendiri karena, 2 buah kata berbeda dapat terdiri dari huruf-huruf yang sama seperti “Palu” dan “Lupa”. Walaupun 2 kata tersebut memiliki huruf-huruf yang sama, namun artinya sangat berbeda. Hal yang lebih baik dilakukan dibanding mengkonversi setiap huruf dalam kata, adalah dengan mengonversi setiap kata dalam sebuah kalimat ke bilangan tertentu. Dengan begini Setiap kata yang memiliki arti berbeda, akan diperlakukan berbeda oleh model. Seperti kata lupa dan palu.

Encoding Words

Untuk memudahkan komputer dalam mempelajari data berupa teks bahasa alami manusia. Perlu adanya proses encoding kata, proses ini berfungsi memberi token atau index untuk setiap kata yang terdapat pada teks. Sebelum memberi token atau index, kita harus memisahkan setiap kata dari kalimatnya. Pada Tensorflow, kita dapat memisahkan kata menggunakan Tokenizer.

Contoh:

Kita mempunyai 2 kalimat.

```
from tensorflow.keras.preprocessing.text import Tokenizer
kalimat = [
    'saya suka kucing',
    'saya suka tikus']
```

Gunakan tokenizer

```
tokenizer = Tokenizer(num_words=100)
tokenizer.fit_on_texts(kalimat)
```

Gunakan word_index untuk memberi index setiap kata, print hasil word index agar dapat melihat hasil proses tokenizing

```
word_index = tokenizer.word_index
print(word_index)
```

Output

```
{'saya': 1, 'suka': 2, 'kucing': 3, 'tikus': 4}
```

Sequence

Fungsi sequences sendiri berfungsi layaknya word_index, namun jika word_index mengembalikan pecahan kata dengan index berupa dictionary. Sequence mengembalikan array yang merupakan index dari penyusun kalimat tersebut. Kita dapat menggunakan fungsi **texts_to_sequences()**

```
sequences = tokenizer.texts_to_sequences(kalimat)
print("Word Index : ", word_index)
print("Sequences : ", sequences)
```

Output

```
Word Index :  {'saya': 1, 'suka': 2, 'kucing': 3, 'tikus': 4}

Sequences :  [[1, 2, 3], [1, 2, 4]]
```

Padding

Fungsi padding berguna untuk menyamakan panjang kalimat pada sequences. Sama seperti proses resize pada gambar, setiap kalimat pada sequence di ubah panjang nya sama rata berupa array 2 dimensi. Kita dapat menggunakan **pad_sequences()** untuk melakukan padding.

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
padded = pad_sequences(sequences, maxlen=5, padding='post')
print(padded)
```

Output

```
[[1 2 3 0 0]

 [1 2 4 0 0]]
```

Modeling

Proses modeling pada Natural Language Processing berbeda dibanding dengan Image Processing, dimana penggunaan layer bergantung dengan dataset.

Embedding

Embedding layer memungkinkan kita untuk mengubah setiap kata menjadi vektor dengan panjang tetap dengan ukuran yang ditentukan. Vektor yang dihasilkan adalah vektor padat dengan nilai riil, bukan hanya 0 dan 1. Panjang vektor kata yang tetap membantu kita untuk merepresentasikan kata-kata dengan cara yang lebih baik bersama dengan pengurangan dimensi.

Dengan cara ini embedding layer berfungsi seperti tabel pencarian. Kata-kata adalah kunci dalam tabel ini, sedangkan vektor kata padat adalah nilainya. Untuk lebih memahaminya, mari kita lihat implementasi layer Keras Embedding.

Embedding layer biasa digunakan pada model sebagai input layer.

```
tf.keras.layers.Embedding(
    input_dim,
    output_dim,
    embeddings_initializer='uniform',
    embeddings_regularizer=None,
    activity_regularizer=None,
    embeddings_constraint=None,
    mask_zero=False,
    input_length=None,
    **kwargs
)
```

Keterangan:

- input_dim
  - dimensi input, berupa integer
- output_dim
  - output dimensi, berupa integer, dimensi dense embedding
- embeddings_initializer
  - inisialisasi embedding matrix
- embeddings_regularizer
  - Fungsi regularizer untuk embedding matrix
- embeddings_constraint
  - Fungsi constraint untuk embedding matrix
- mask_zero
  - Boolean, berguna saat menggunakan lapisan berulang yang mungkin memerlukan input panjang variabel.
- input_length
  - Panjang input sequences jika bersifat constant, sangat dibutuhkan jika akan dikoneksikan ke Flatten kemudian Dense layer.

Conv1D Layer

Berbeda dengan Conv2D, lapisan ini membuat kernel konvolusi yang digabungkan dengan input lapisan melalui satu dimensi spasial (atau temporal) untuk menghasilkan tensor keluaran. Jika use_bias True, vektor bias dibuat dan ditambahkan ke output. Terakhir, jika aktivasi bukan Tidak ada, aktivasi juga diterapkan ke output.

```
tf.keras.layers.Conv1D(
    filters,
    kernel_size,
    strides=1,
    padding='valid',
    data_format='channels_last',
    dilation_rate=1,
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

Keterangan filter

- Bilangan bulat, dimensi ruang keluaran (yaitu jumlah filter keluaran dalam konvolusi).

- kernel_size
  - Bilangan bulat atau tupel/daftar 2 bilangan bulat, menentukan tinggi dan lebar jendela konvolusi 1D. Dapat berupa bilangan bulat tunggal untuk menentukan nilai yang sama untuk semua dimensi spasial.
- strides
  - Sebuah bilangan bulat atau tupel/daftar dari 2 bilangan bulat, menentukan langkah konvolusi sepanjang tinggi dan lebar. Dapat berupa bilangan bulat tunggal untuk menentukan nilai yang sama untuk semua dimensi spasial. Menentukan nilai langkah apa pun != 1

tidak kompatibel dengan menentukan nilai kecepatan_dilatasi apa pun != 1.

- padding
  - salah satu dari "valid" atau "sama" (peka huruf besar-kecil). "valid" berarti tidak ada padding. "sama" menghasilkan padding dengan nol secara merata ke kiri/kanan atau atas/bawah input. Saat padding="same" dan strides=1, output memiliki ukuran yang sama dengan input.
- activation
  - Fungsi aktivasi untuk digunakan. Jika Anda tidak menentukan apa pun, tidak ada aktivasi yang diterapkan (lihat keras.activations).

Bidirectional

Didefinisikan sebagai menghubungkan dua hidden layer dari arah yang berlawanan ke output yang sama. Karena pembelajaran mendalam yang generatif ini, lapisan keluaran mendapatkan informasi dari masa lalu atau backward dan masa depan atau forward secara bersamaan. Biasanya Bidirectional layer berisi layer RNN seperti LSTM (Long-Short Term Memory) atau GRU (Gated Recurrent Unit).

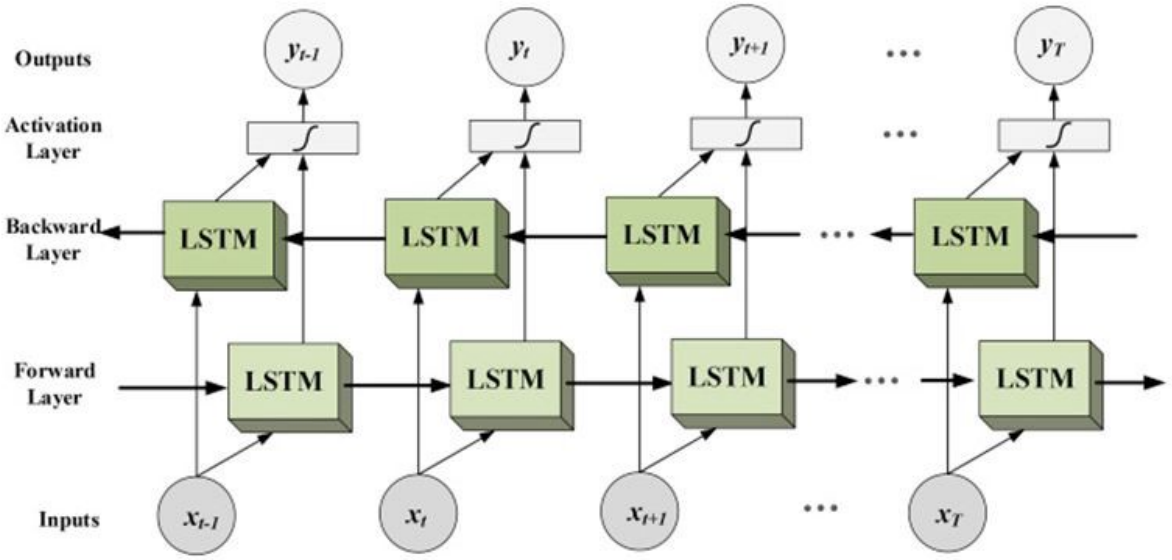
```
tf.keras.layers.Bidirectional(
    layer,
    merge_mode='concat',
    weights=None,
    backward_layer=None,
    **kwargs
)
```

Keterangan :

- layer
  - Implementasi layer yang digunakan
- merge_mode
  - Mode di mana keluaran RNN forward dan backward akan digabungkan. Salah satu dari {'jumlah', 'mul', 'concat', 'ave', Tidak ada}. Jika Tidak Ada, keluaran tidak akan digabungkan, keluaran akan dikembalikan sebagai daftar. Nilai default adalah 'concat'.

- backward_layer
  - Instance keras.layers.RNN, atau keras.layers.Layer opsional untuk digunakan untuk menangani pemrosesan input mundur.

LSTM Layer



Long Short-Term Memory (LSTM) merupakan model varian dari Recurrent Neural Network (RNN). LSTM muncul karena dapat mengingat informasi jangka panjang (long term dependency) LSTM menggantikan simpul hidden layer di RNN dengan sel LSTM yang yang berfungsi untuk menyimpan informasi sebelumnya. Dalam LSTM terdapat tiga gerbang yang mengendalikan penggunaan dan memperbarui informasi terks terdahulu yaitu input gate, forget gate, dan ouput gate. Sel memori dan tiga gerbang dirancang untuk dapat membaca, menyimpan, dan memperbarui informasi terdahulu.

```
tf.keras.layers.LSTM(
    units,
    activation='tanh',
    recurrent_activation='sigmoid',
    use_bias=True,
    kernel_initializer='glorot_uniform',
    recurrent_initializer='orthogonal',
    bias_initializer='zeros',
    unit_forget_bias=True,
    kernel_regularizer=None,
    recurrent_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
```

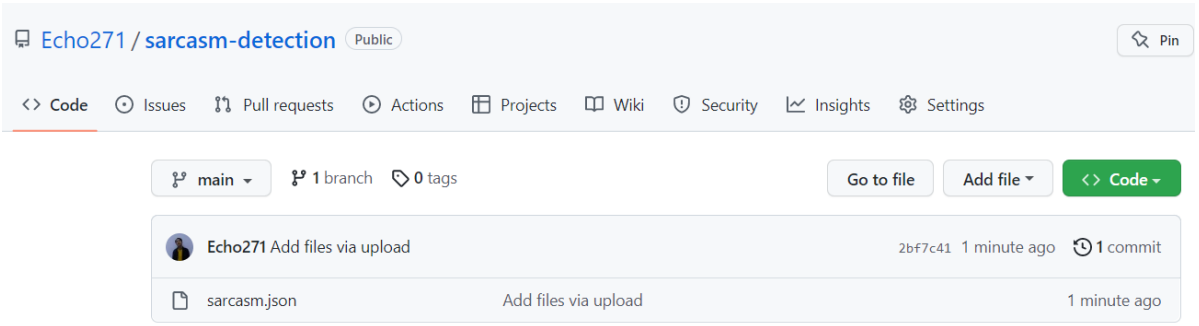
```
    kernel_constraint=None,
    recurrent_constraint=None,
    bias_constraint=None,
    dropout=0.0,
    recurrent_dropout=0.0,
    return_sequences=False,
    return_state=False,
    go_backwards=False,
    stateful=False,
    time_major=False,
    unroll=False,
    **kwargs
)
```

Adapun parameter yang biasa digunakan pada LSTM layer sebagai berikut.

- units
  - jumlah unit neuron
- activation
  - Fungsi aktivasi untuk digunakan. Jika Anda tidak menentukan apa pun, tidak ada aktivasi yang diterapkan (lihat keras.activations).
- return_sequences
  - Booleans. Mengembalikan output terakhir kedalam output order, atau full sequence. Default: False.

CONTOH PENERAPAN NLP

1. Siapkan dataset yang akan digunakan, kali ini kita akan menggunakan dataset sarcasm dengan format json. Untuk dataset bisa di download pada link berikut.



2. Import library yang dibutuhkan.

```
import tensorflow as tf
import json
import numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

3. Ekstrak dataset kedalam program, kemudian pisahkan headline dan label ke dalam list.

```
# Load file JSON ke program
with open("./sarcasm.json", 'r') as f:
    datastore = json.load(f)

# Buat list untuk menyimpan
sentences = []
labels = []

# Ambil setiap kalimat dan label ke dalam list
for item in datastore:
    sentences.append(item['headline'])
    labels.append(item['is_sarcastic'])
```

4. Lalu Split data menjadi data training dan data testing.

```
training_size = 20000

# Split data kalimat
training_sentences = sentences[0:training_size]
testing_sentences = sentences[training_size:]

# Split data label
training_labels = labels[0:training_size]
testing_labels = labels[training_size:]
```

5. Deklarasikan variabel yang akan digunakan saat pre processing.

```
vocab_size = 10000
max_length = 120
trunc_type='post'
padding_type='post'
oov_tok = ""
```

6. Inisialisasi tokenizer dengan parameter vocab size dan oov token

```
# Inisialisasi tokenizer
tokenizer = Tokenizer(num_words=vocab_size, oov_token=oov_tok)
```

7. Generate setiap kata ke dictionary menggunakan fit_on_text dari tokenizer

```
# Generate kalimat ke dalam dictionary
tokenizer.fit_on_texts(training_sentences)
word_index = tokenizer.word_index
```

8. Generate sequences dan pad masing-masing data yang telah di split.

```
# Generate sequences dan pad pada training data
training_sequences = tokenizer.texts_to_sequences(training_sentences)
training_padded   =   pad_sequences(training_sequences,   maxlen=max_length,
padding=padding_type, truncating=trunc_type)

# Generate sequences dan pad pada testing data
```

```
testing_sequences = tokenizer.texts_to_sequences(testing_sentences)
testing_padded    =    pad_sequences(testing_sequences,    maxlen=max_length,
padding=padding_type, truncating=trunc_type)
```

9. Konversi list label menjadi numpy array.

```
# Konversi list label menjadi numpy array
training_labels = np.array(training_labels)
testing_labels = np.array(testing_labels)
```

10. Buat model menggunakan LSTM.

```
# Parameter
embedding_dim = 16
lstm_dim = 32
dense_dim = 24

# Model dengan LSTM
model_lstm = tf.keras.Sequential([
                    tf.keras.layers.Embedding(vocab_size,     embedding_dim,
input_length=max_length),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(lstm_dim)),
    tf.keras.layers.Dense(dense_dim, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

11. Compile Model menggunakan binary crossentropy sebagai loss dan adam sebagai optimizer.

```
model_lstm.compile(loss='binary_crossentropy',optimizer='adam',metrics=['a
ccuracy'])
```

12. Latih model dengan menginputkan sequences yang telah di padding dengan epoch 10 kali.

```
# Training model
```

```
history_lstm = model_lstm.fit(training_padded, training_labels, epochs=10, validation_data=(testing_padded, testing_labels))
```

```
Epoch 1/10
625/625 [==============================] - 16s 23ms/step - loss: 0.4180 - accuracy: 0.7922 - val_loss: 0.3292 - val_accuracy: 0.8557
Epoch 2/10
625/625 [==============================] - 17s 27ms/step - loss: 0.2176 - accuracy: 0.9125 - val_loss: 0.3455 - val_accuracy: 0.8545
Epoch 3/10
625/625 [==============================] - 19s 30ms/step - loss: 0.1347 - accuracy: 0.9502 - val_loss: 0.3788 - val_accuracy: 0.8548
Epoch 4/10
625/625 [==============================] - 17s 27ms/step - loss: 0.0809 - accuracy: 0.9725 - val_loss: 0.4695 - val_accuracy: 0.8493
Epoch 5/10
625/625 [==============================] - 16s 26ms/step - loss: 0.0511 - accuracy: 0.9832 - val_loss: 0.5789 - val_accuracy: 0.8410
Epoch 6/10
625/625 [==============================] - 15s 25ms/step - loss: 0.0300 - accuracy: 0.9913 - val_loss: 0.7343 - val_accuracy: 0.8380
Epoch 7/10
625/625 [==============================] - 14s 23ms/step - loss: 0.0203 - accuracy: 0.9941 - val_loss: 0.7892 - val_accuracy: 0.8384
Epoch 8/10
625/625 [==============================] - 15s 24ms/step - loss: 0.0167 - accuracy: 0.9947 - val_loss: 0.8767 - val_accuracy: 0.8326
Epoch 9/10
625/625 [==============================] - 15s 25ms/step - loss: 0.0143 - accuracy: 0.9953 - val_loss: 0.9348 - val_accuracy: 0.8359
Epoch 10/10
625/625 [==============================] - 16s 25ms/step - loss: 0.0108 - accuracy: 0.9969 - val_loss: 1.0568 - val_accuracy: 0.8278
```

Source Online

1. https://medium.com/sysinfo/natural-language-processing-nlp-b54d6506efe2
2. https://rifqifai.com/cara-kerja-long-short-term-memory-lstm/