Modul Praktikum

Kecerdasan Buatan

8

Daftar Isi

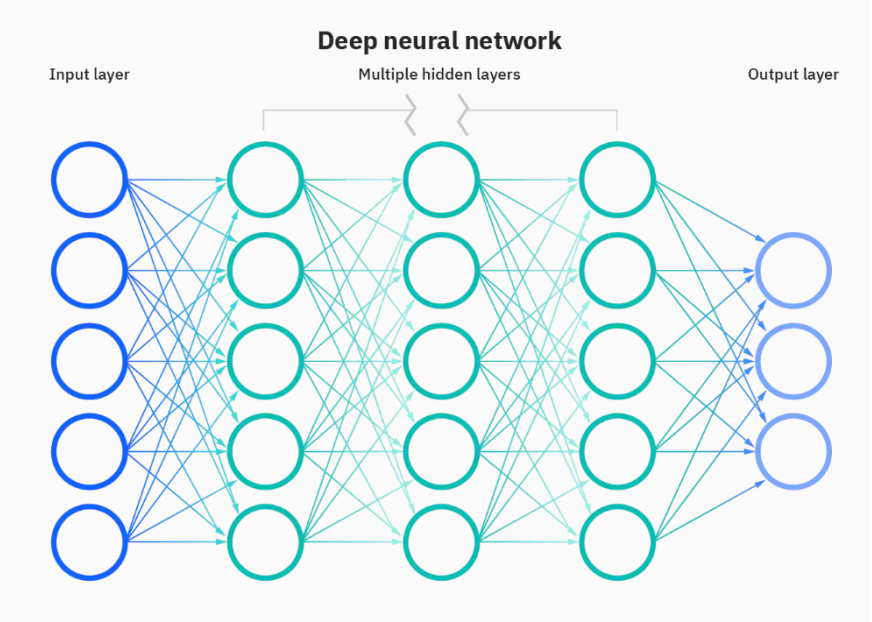
TensorFlow – Deep Learning

Deep learning merupakan subbidang machine learning yang algortimatnya terinspirasi dari struktur syaraf manusia. Struktur tersebut dinamakan Artificial Neural Network yang disingkat ANN. ANN memiliki kemampuan untuk belajar dan beradaptasi sesuai informasi yang diberikan. Deep learning menggunakan level hirarki jaringan saraf tiru yang dibuat seperti otak manusia dengan neuron nodes terhubung seperti jaring.

Neuron-neuron dalam neural network disusun dalam grup yang disebut layer. ANN terdiri dari 3 layer:

1. Input layer → berisi variabel data input
2. Hidden layer/processing layer → berisi Langkah pengenalan objek
3. Output layer → berisi hasil pengenalan suatu objek

Deep Neural Network (DNN) adalah salah satu algoritma dalam ANN. Algoritma ini memiliki pengertian sebagai jaringan saraf tiru yang memiliki lebih dari satu lapisan saraf tersebunyi. Pada modul ini kita akan membangun DNN menggunakan Keras.



Keras merupakah high API yang di-design untuk mempermudah dalam pembangunan jaringan saraf tiru. Library open-source neural network ini dirancang untuk menyediakan experiment dengan DNN. Untuk menggunakannya, kita dapat menggunakan library open source TensorFlow.

Sequential Model

Model sequential atau model yang berurutan adalah bentuk model yang paling umum digunakan dalam membuat DNN. Model ini bekerja dengan membuat tumpukan lapisan/layer dimana setiap layer memiliki input dan output. Model ini menampung semua layer neural network yang kita buat.

`tf.keras.Sequential()`

Contoh

```
import tensorflow as tf

model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(16, activation='relu'))
model.add(tf.keras.layers.Dense(32, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

Layer Type

Langkah pertama dari data modeling adalah membuat model. Ada banyak jenis layer yang dapat digunakan dalam ANN. untuk mendefinisikan layer dalam ANN kita dapat menggunakan `tf.keras.layers`. Berikut merupakan beberapa jenis layer yang dapat digunakan.

- Core
  - Input object
  - Dense
  - Embedding
- Convolution
  - Conv1D
  - Conv2D
- Pooling
  - MaxPooling2D
  - AveragePooling2D
- Recurrent
  - LSTM
  - Bidirectional
- Normalization
  - BatchNormalization
  - LayerNormalization
- Regularization
  - Dropout
  - GaussianDropout
- Reshaping
  - Flatten
  - Reshape

DNN

Membangung layer yang saling terhubung. Layer ini merupakan jenis layer yang paling sering digunakan.

`tf.keras.layers.Dense()`

- **units**: jumlah neurons
- **activation**: fungsi aktivasi yang digunakan
- **use_bias**: menentukan menggunakan bias atau tida. default:bias
- **kernel_initializer**: skema inisialisasi yang menciptakan weight layer (kernel)
- **bias_initializer**: skema inisialisasi yang menciptakan weight layer (bias)
- **kernel_regularizer**: skema regularisasi yang berlaku pada weight layer (kernel)
- **bias_regularizer**: skema regularisasi yang berlaku pada weight layer (bias)
- **activity_regularizer**: item biasa yang diterapkan pada output, objek regular
- **kernel_constraint**: constraint yang diterapkan pada weight
- **bias_constraint:** constraint yang diterapkan pada weight

Contoh 1

```
import tensorflow.keras.layers as layers

model = tf.keras.Sequential()
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(32, kernel_initializer=tf.keras.initializers.he_normal))
model.add(layers.Dense(32, kernel_regularizer=tf.keras.regularizers.l2(0.01))
model.add(layers.Dense(10, activation='softmax'))
```

Contoh 2

```
model = tf.keras.Sequential([
tf.keras.layers.Dense(32, activation='relu'),
tf.keras.layers.Dense(32, kernel_initializer=tf.keras.initializers.he_normal),
tf.keras.layers.Dense(32, kernel_regularizer=tf.keras.regularizers.l2(0.01)),
tf.keras.layers.Dense(10, activation='softmax')
])
```

Setelah layer disusun, kita dapat melihat detail parameter setiap layer yang telah disusun pada model yang dibuat.

`model.summary()`

Contoh

```
model = tf.keras.Sequential([
 tf.keras.layers.Dense(units=64, input_shape=[10,10,1]),
tf.keras.layers.Dense(32, activation='relu'),
tf.keras.layers.Dense(32, kernel_initializer=tf.keras.initializers.he_normal),
tf.keras.layers.Dense(32, kernel_regularizer=tf.keras.regularizers.l2(0.01)),
tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()
```

Output

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_13 (Dense)             (None, 10, 10, 64)        128

 dense_14 (Dense)             (None, 10, 10, 32)        2080

 dense_15 (Dense)             (None, 10, 10, 32)        1056

 dense_16 (Dense)             (None, 10, 10, 10)        330

=================================================================
Total params: 3,594
Trainable params: 3,594
Non-trainable params: 0
_________________________________________________________________
```

Training Model

Training model merupakan proses belajar mesin dari dataset yang kita berikan. Biasanya dataset yang digunakan dibagi menjadi train set dan val set. Beban komputasi pada proses ini sangat bergantung pada spesifikasi komputer, banyak dataset, dan tebal tipisnya layer deep learning yang dibuat. Tujuan utama dari training model adalah mendapatkan akurasi yang tinggi dengan beban komputasi sekecil mungkin.

Model Compile

Setelah model selesai dibangun, langkah selanjutnya adalah mengatur configurasi model.

```
model.compile()
```

- **optimizer**: nama optimizer(string) atau instance optimizer
- **loss**: nama fungsi loss(string) atau instance loss
- **metrics**: kriteria evaluasi model saat training dan testing
- **loss_weight**: digunakan jika memiliki beberapa output
- **weighted_metrics**: list matriks yang akan dievaluasi dan weigted oleh sample_weight atau class_weigth saat training dan testing.
- **run_eagerly**: mengatur eager execution secara manual
- ****kwargs**: argumen tambahan

Optimizer

Optimizer adalah class atau metode yang digunakan untuk mengubah atribut model seperti weight dan learning rate untuk mengurangi loss. Optimizer bekerja berdasarkan hasil yang diberikan fungsi loss. Berikut merupakan optimizer yang paling sering digunakan.

- adam
- RMSprop
- SGD

Fungsi Loss

Fungsi loss bertujuan menghitung kuantitas yang harus diminimalkan model dalam training. Berikut merupakan beberapa funsi loss.

- Probabilistic losses
  - Binary crossentropy
  - Categorical crossentropy

    - Sparse categorical Crossentropy
- Regression losses
    - MSE
    - MAE
    - Huber
- Hinge losses for 'maximum-margin' classification
    - Hinge
    - Squared hinge
    - Categorical hinge

Contoh

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

Model Fit

Setelah configurasi model telah dilakukan, langkah terakhir data data modeling adalah training data.

`mode.fit()`

- **x**: input trainset
- **y**: target trainset
- **batch_size**: jumlah sampel untuk setiap pembaruan gradien. nilai default (32)
- **epochs**: jumlah iterasi training model
- **verbose**: menentukan tampilan dari progress training
    - 0: no display
    - 1: menampilkan animasi progress bar
    - 2: hanya menampilkan angka
- **callbacks**: fungsi callback yang digunakan
- **validation_split**: jumlah valset dari trainset
- **validation_data**: validation set. overwrite validation_split
- **shuffle**: mengacak data sebelum setiap iterasi. invalid jika steps_per_epoch tidak berniali none
- **initial_epoch**: training akan dimulai pada epoch yang ditentukan
- **steps_per_epoch**: ukuran sampel per batch. (num_samples//batch_size)
- **validation_steps**: total steps untuk divalidasi sebelum berhenti. valid ketika steps_per_epoch ditentukan.
- **validation_batch_size**: jumlah sampel per batch validasi

- **validation_freq**: hanya relevan ketiak data validasi disediakan.(integer)
- **max_queue_size**: jumlah maksimal antrian generator. digunakan untuk generator atau input sequence.(integer)
- **workers**: jumlah processing yang digunakan ketika menggunakan process-based threading. digunakan untuk generator atau input sequence.(integer)
- **use_multiprocessing**: process-based threading. digunakan untuk generator atau input sequence.(boolean)

Contoh

```
model.fit(train_x, train_y, epochs=10, batch_size=100,
validation_data=(val_x, val_y))
```

Callback Functions

Fungsi Callback adalah fungsi tambahan yang dijalankan saat proses training. Biasanya fungsi callback akan dijalankan ketika proses training memenuhi kondisi tertentu. Sebagai contoh fungsi callback akan menghentikan proses training ketika tingkat akurasi terpenuhi sebelum model training menyelesaikan semua epoch nya.

Contoh:

Menggunakan Class

```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
       if(logs.get('accuracy') > 0.84 and logs.get('val_accuracy') > 0.84):
       self.model.stop_training = True
```

contoh fungsi callback diatas berguna untuk menghentikan proses training ketika tingkat akurasi mencapai lebih dari 84% dan akurasi validasi diatas 84%.

Meinstansi dan menggunakan Class MyCallback

```
callback_function = myCallback

model.fit(train_x, train_y, epochs=10, batch_size=100,
validation_data=(val_x, val_y), callbacks = callback_function)
```

Model Evaluation

Kita dapat menguji model dengan melakukan evaluasi pada model yang telah ditraining untuk melihat akurasi pada data testing. Evaluasi dilakukan menggunakan input testing set(x) dan target testing set(y)

`model.evaluate()`

Contoh

```
model.evaluate( x_test, y_test, batch_size=32)
```

Output

```
1000/1000 [==============================] - 0s 45us/sample - loss: 12.2881 -
categorical_accuracy: 0.0770
[12.288104843139648, 0.077]
```

Model Prediction

Cara lain menguji model yang dapat dilakukan adalah prediksi. Prediksi dilakukan hanya menggunakan input testing set.

`model.predict()`

Contoh

```
result = model.predict(test_x)
print(result)
```

Output

```
[[0.04431767 0.24562006 0.05260926 ... 0.1016549  0.13826898 0.15511878]
 [0.06296062 0.12550288 0.07593573 ... 0.06219672 0.21190381 0.12361749]
 [0.07203944 0.19570401 0.11178136 ... 0.05625525 0.20609994 0.13041474]
 ...
 [0.09224506 0.09908539 0.13944311 ... 0.08630784 0.15009451 0.17172746]
 [0.08499582 0.17338121 0.0804626  ... 0.04409525 0.27150458 0.07133815]
 [0.05191234 0.11740112 0.08346355 ... 0.0842929  0.20141983 0.19982798]]
```

Model Saving

Proses training model terkadang tidak berlangsung sebentar. Oleh karena itu sangat penting untuk menyimpan model yang telah dibuat. Model yang telah mencapai standar akurasi dari developer dapat disimpan ke dalam file model. Model dapat disimpan dalam bentuk JSON, YAML, atau HDF5.

HDF5

`model.save()` → menyimpan weight, arsitektur, cofigurasi, dan state optimizer ke satu file

```
model.save(‘./model/model_finished’)
model.save(‘./model/model_finished.h5’)
```

`model.save_weights()` → hanya menyimpan weight

```
model.save_weights('./model/model_weights')
model.save_weights('./model/model_weights.h5')
```

YAML

`model.to_yaml()`

```
model_yaml = model.to_yaml()
with open("model.yaml", "w") as yaml_file:
    yaml_file.write(model_yaml)
```

JSON

`model.to_json()`

```
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
```

Model Loading

File model yang ada dapat digunakan dengan menggunakan model loading.

HDF5

`tf.keras.models.load_model()`

```
model.load_model('./model/model_finished')
model.load_model('./model/model_finished.h5')
```

`tf.keras.models.load_weights()`

```
model.load_weights('./model/model_weights')
model.load_weights('./model/model_weights.h5')
```

YAML

`tf.keras.models.model_from_yaml()`

```
yaml_file = open('model.yaml', 'r')
loaded_model_yaml = yaml_file.read()
yaml_file.close()
loaded_model = model_from_yaml(loaded_model_yaml)
```

JSON

`tf.keras.models.model_from_json()`

```
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
```

Contoh Penerapan Deep Neural Network

Regresi Sederhana

```
import tensorflow as tf
import numpy as np

#membuat variable x sebagai atribute dan y sebagai label
x = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], dtype=float)
y = np.array([2.0, 4.0, 6.0, 8.0, 10.0, 12.0], dtype=float)

#membuat sequential dimana hanya 1 dense layer yang dibutuhkan
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, input_shape=[1])
])

#model di kompilasi dengan optimizer sgd dan loss MSE
model.compile(optimizer='sgd', loss='mean_squared_error')

#model di training dengan epoch yang dapat kita atur
model.fit(x, y, epochs=100)

#mencoba prediksi y dengan model yang telah dilatih
y_pred = model.predict([6.0, 7.0, 8.0])
print(y_pred)
```

Output:

Epoch 97/100

1/1 [==============================] - 0s 5ms/step - loss: 0.0595

Epoch 98/100

1/1 [==============================] - 0s 9ms/step - loss: 0.0591

Epoch 99/100

1/1 [==============================] - 0s 0s/step - loss: 0.0587

Epoch 100/100

1/1 [==============================] - 0s 8ms/step - loss: 0.0583

<keras.callbacks.History at 0x2118de03310>

1/1 [==============================] - 0s 97ms/step

Prediksi y diatas adalah [[11.779914]
[13.651868]
[15.523822]]

Pada contoh diatas, model regression diukur berdasarkan nilai **loss** nya. Semakin rendah nilai loss nya, maka semakin bagus hasil trainingnya. Maka kita bisa mencoba menambah jumlah epoch untuk mencapai tingkat loss terendah sekitar **< 0.002**. Biasa nya kita lakukan hingga ribuan epoch, dan kita juga bisa menambahkan fungsi callback jika nilai loss sudah terpenuhi.

Klasifikasi dengan dataset MNIST

Pada contoh ini, kita akan coba membuat klasifikasi digital handwritting dari dataset mnist menggunakan DNN.

1. Import library yang dibutuhkan

```
import os
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, optimizers, datasets
from matplotlib import pyplot as plt
import numpy as np
```

2. Load dataset mnist dari tensorflow dan split menjadi data training testing

```
(x_train_raw, y_train_raw), (x_test_raw, y_test_raw) = datasets.mnist.load_data()
```

3. Encoding label dengan one-hot encoding menggunakan **keras.utils**

```
num_classes = 10
y_train = keras.utils.to_categorical(y_train_raw, num_classes)
y_test = keras.utils.to_categorical(y_test_raw, num_classes)
print(y_train[0])
```

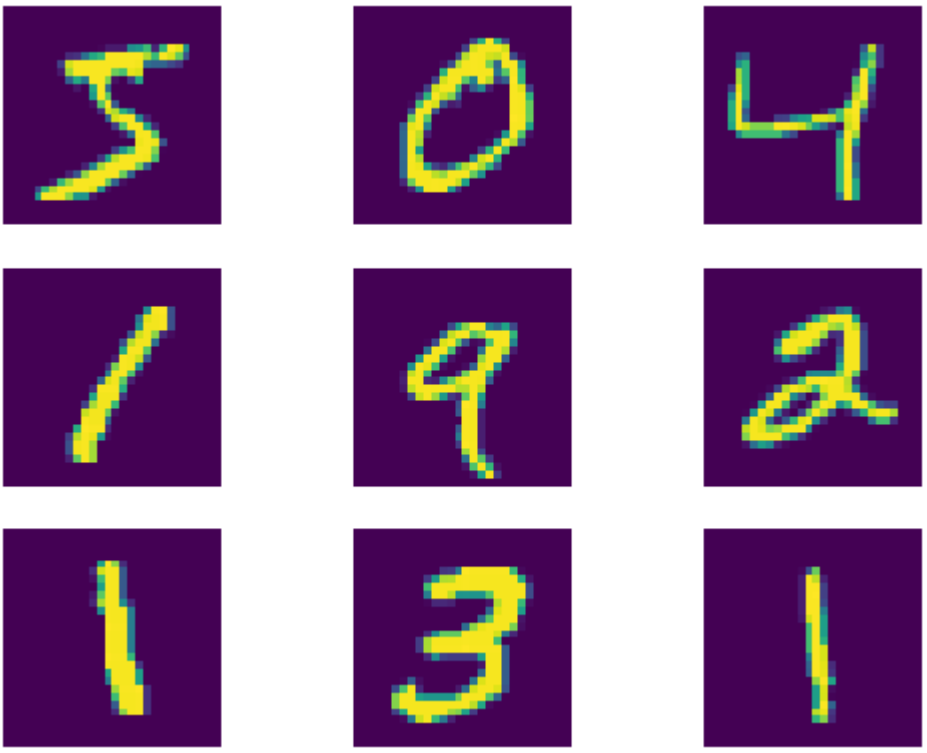
Output:

[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]

4. Menampilkan sampel digital handwriting mnist dataset

```
plt.figure()
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(x_train_raw[i])
    #plt.ylabel(y[i].numpy())
    plt.axis('off')
plt.show()
```

Output:



5. Konversikan gambar 28x28 menjadi 784x1 vector / lakukan reshaping dan normalisasi kan nilai pixel gamnbarnya

```
#Convert a 28 x 28 image into a 784 x 1 vector.
x_train = x_train_raw.reshape(60000, 784)
x_test = x_test_raw.reshape(10000, 784)

#Normalize image pixel values.
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
```

6. Buat sequential model menggunakan 3 dense layer dengan fungsi activation ReLu dan softmax activation

```
#Create a deep neural network (DNN) model that consists of three fully connected
layers and two RELU activation functions.

model = keras.Sequential([
    layers.Dense(512, activation='relu', input_dim = 784),
    layers.Dense(256, activation='relu'),
    layers.Dense(124, activation='relu'),
layers.Dense(num_classes, activation='softmax')])

model.summary()
```

Output:

Model: "sequential_1"

_____

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 512)	401920
dense_2 (Dense)	(None, 256)	131328
dense_3 (Dense)	(None, 124)	31868
dense_4 (Dense)	(None, 10)	1250

Total params: 566,366

Trainable params: 566,366

Non-trainable params: 0

_____

7. Compile model menggunakan optimizer adam, categorical_crossentropy loss function, dan metrics accuracy

```
Optimizer = optimizers.Adam(0.001)
model.compile(loss=keras.losses.categorical_crossentropy, optimizer=Optimizer,
metrics=['accuracy'])
```

8. Training model dengan batch_size 128, dan 10 epoch

```
#Fit the training data to the model by using the fit method.
model.fit(x_train, y_train,
          batch_size=128,
          epochs=10,
          verbose=1)
```

Output:

Epoch 1/10

469/469 [==============================] - 7s 12ms/step - loss: 0.2295 - accuracy: 0.9311

Epoch 2/10

469/469 [==============================] - 6s 12ms/step - loss: 0.0863 - accuracy: 0.9734

Epoch 3/10

469/469 [==============================] - 6s 12ms/step - loss: 0.0559 - accuracy: 0.9827

Epoch 4/10

469/469 [==============================] - 6s 12ms/step - loss: 0.0406 - accuracy: 0.9868

Epoch 5/10

469/469 [==============================] - 6s 12ms/step - loss: 0.0315 - accuracy: 0.9891

Epoch 6/10

469/469 [==============================] - 6s 12ms/step - loss: 0.0254 - accuracy: 0.9915

Epoch 7/10

469/469 [==============================] - 6s 12ms/step - loss: 0.0215 - accuracy: 0.9929

Epoch 8/10

469/469 [==============================] - 6s 12ms/step - loss: 0.0205 - accuracy: 0.9933

Epoch 9/10

469/469 [==============================] - 6s 12ms/step - loss: 0.0158 - accuracy: 0.9947

Epoch 10/10

469/469 [==============================] - 6s 12ms/step - loss: 0.0158 - accuracy: 0.9944

<keras.callbacks.History at 0x2118e688280>

9. Evaluate model menggunakan data testing

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Output:

Test loss: 0.08676622062921524

Test accuracy: 0.9801999926567078

Source Online

1. Layers
2. Optimizer
3. Optimizer
4. Optimizer
5. Loss
6. Loss
7. verbose
8. steps_per_epoch
9. model_saving
10. save()