YAYASAN PRIMA AGUS TEKNIK

Deep Learning dengan Python

Dr. Budi Raharjo, S.Kom., M.Kom., MM.

Deep Learning dengan Python

Dr. Budi Raharjo, S.Kom., M.Kom., MM.

BIODATA PENULIS

Dr. Budi Raharjo, S.Kom, M.Kom, MM lahir di Semarang, tanggal 22 Februari 1985. Beliau adalah Alumni dari Universitas Bina Nusantara (BINUS University) Jakarta dan juga alumni Universitas Kristen Satya wacana (UKSW) Salatiga. Dr. Budi Raharjo telah menjadi Dosen pada Universitas STEKOM pada mata kuliah Kepemimpinan (Leadership), mata kuliah Pengantar Akuntansi, Manajemen Proses, Manajemen Akuntansi dan Manajemen Resiko Bisnis. Selain sebagai dosen Universitas STEKOM, Dr. Budi Raharjo, M.Kom, MM juga mempunyai bisnis sendiri dalam bidang perhotelan dan juga sebagai wirausaha dalam bidang pemasok unggas (ayam) beku, ke berbagai kota besar, khususnya Jakarta dan sekitarnya.

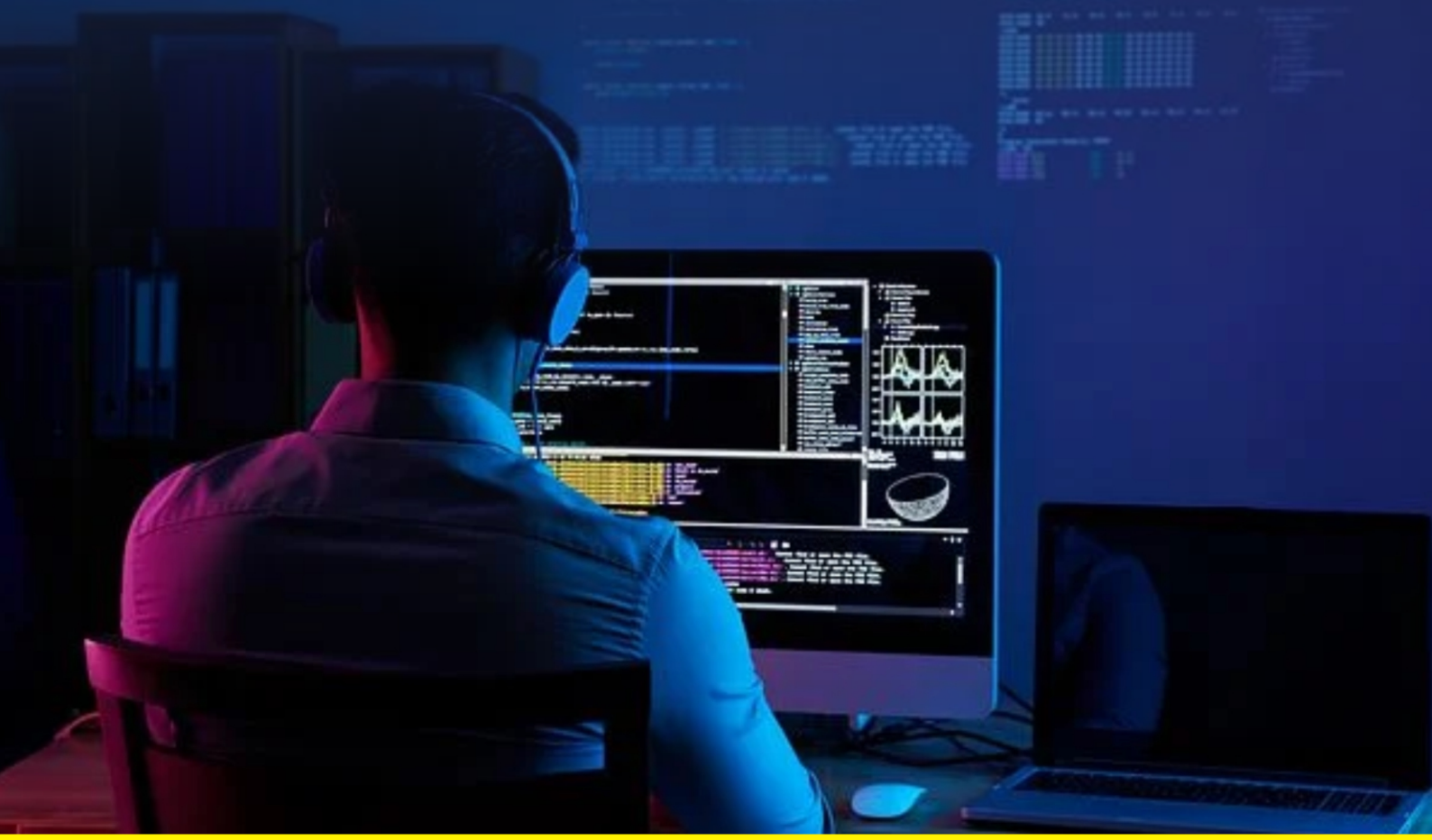
Pengalaman beliau berwirausaha menjadi bekal utama dalam penulisan buku ajar yang diterbitkan oleh Yayasan Prima Agus Teknik (YPAT) Semarang. Oleh sebab itu bukunya berisi langkah langkah praktis yang mudah diikuti oleh para mahasiswa, saat mahasiswa mengikuti proses perkuliahan pada Universitas Sains dan Teknologi Komputer (Universitas STEKOM). Jabatan struktural yang di embannya saat ini adalah Wakil Rektor 1 (Akademik) Universitas STEKOM Semarang.

YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-5734-33-0

Deep Learning dengan Python

Dr. Budi Raharjo, S.Kom., M.Kom., MM.



YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

Deep Learning dengan Python

Penulis :
Dr. Budi Raharjo, S.Kom., M.Kom., MM.

ISBN : 9 786235 734330

Editor :
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Penyunting :
Dr. Joseph Teguh Santoso, M.Kom.

Desain Sampul dan Tata Letak :
Irdha Yunianto, S.Ds., M.Kom

Penebit :
Yayasan Prima Agus Teknik

Redaksi :
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :
Universitas STEKOM
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : info@stekom.ac.id

KATA PENGANTAR

Puji syukur kami panjatkan atas selesainya buku yang berjudul "*Deep Learning dengan Python"* dengan baik. *Deep learning* atau disebut dengan Pembelajaran mendalam telah berjalan sangat jauh. Lahirnya ide untuk memahami pikiran manusia dan konsep asosiasionisme bagaimana kita memandang sesuatu dan bagaimana hubungan objek dan pandangan memengaruhi pemikiran dan tindakan kita. Hingga pemodelan asosiasionisme yang dimulai pada tahun 1870-an ketika Alexander Bain memperkenalkan konser pertama Jaringan Syaraf Tiruan dengan mengelompokkan neuron. Kami melihat bagaimana Pembelajaran mendalam telah meningkat secara dramatis dan dalam semua bentuk kehidupan mulai dari deteksi objek, pengenalan suara, terjemahan mesin, kendaraan otonom, deteksi wajah, dan penggunaan deteksi wajah dari tugas-tugas biasa seperti membuka kunci iPhoneX Anda untuk melakukan tugas yang lebih mendalam seperti deteksi dan pencegahan kejahatan.

Convolutional Neural Networks dan *Recurrent Neural Networks* bersinar terang karena mereka terus membantu memecahkan masalah dunia di semua bidang industri seperti Otomotif & Transportasi, Kesehatan & Kedokteran, Ritel untuk beberapa nama. Kemajuan besar sedang dibuat di bidang-bidang ini dan metrik seperti ini cukup menjelaskan tentang industri pembelajaran mendalam yang gamblang: Seperti jumlah Jurnal akademik Ilmu Komputer telah melonjak hampir 10x sejak tahun 1996, lebih banyak startup AI aktif sejak tahun 2000 dan salah satunya adalah tingkat kesalahan klasifikasi gambar telah turun dari 28% pada tahun 2012 menjadi 2,5% pada tahun 2017 dan terus menurun.

Penulis mendorong batas dan bergerak maju dengan Tim penulis untuk mengembangkan Permodelan Jaringan Syaraf tiruan yang memberikan pembelajaran mendalam dan keunggulan besar. Buku ini datang pada saat yang tepat. Industri serta mahasiswa membutuhkan sarana praktis untuk memperkuat pengetahuan mereka dalam Pembelajaran Mendalam dan menerapkan dalam pekerjaan mereka. Saya yakin bahwa buku ini akan memberi para pembaca apa yang mereka butuhkan.

TensorFlow semakin menjadi pemimpin pasar dan Keras juga diadopsi oleh para profesional untuk memecahkan masalah sulit dalam visi komputer dan *NLP (Natural Language Processing).* Tidak ada satu perusahaan pun di planet ini yang tidak berinvestasi di dua area aplikasi ini. Akhir Kata semoga Buku ini berguna bagi para pembaca.

Semarang, Januari 2022

Penulis

Dr. Budi Raharjo, S.Kom M.Kom, MM.

DAFTAR ISI

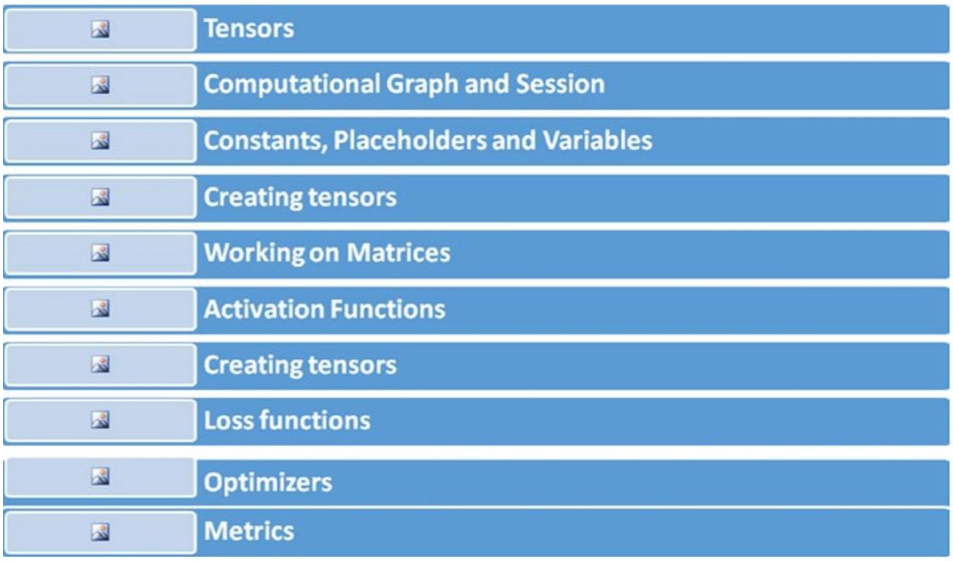
BAB 1
DASAR - DASAR TENSORFLOW

Bab ini akan membahas dasar-dasar TensorFlow, kerangka kerja *deep learning*, dan juga menjelaskan komponen-komponen dasar TensorFlow. Dalam pengenalan *deep learning* melakukan pekerjaan yang luar biasa dalam pengenalan pola, terutama dalam konteks gambar, suara, ucapan, bahasa, dan data deret waktu. Dengan bantuan *deep learning*, Anda dapat mengklasifikasikan, memprediksi, mengelompokkan, dan mengekstrak fitur. Pada November 2015, Google merilis TensorFlow, yang digunakan pada sebagian besar produk Google, seperti Google Search, deteksi spam, Speech Recognition/pengenalan ucapan, Google Assistant, Google Now, dan Google Photo.

TensorFlow memiliki kemampuan unik untuk melakukan komputasi subgraf parsial sehingga memungkinkan pelatihan terdistribusi dengan bantuan partisi *neural networks*. Dengan kata lain, TensorFlow memungkinkan paralelisme model dan paralelisme data. TensorFlow menyediakan beberapa API. TensorFlow Core (API level terendah) memberi Anda kontrol pemrograman lengkap. Perhatikan poin penting berikut terkait TensorFlow:

- Grafiknya adalah deskripsi perhitungan.
- Grafiknya memiliki simpul yang merupakan operasi.
- Ini mengeksekusi perhitungan dalam konteks sesi tertentu.
- Grafik harus diluncurkan dalam sesi untuk perhitungan apa pun.
- Sesi menempatkan operasi grafik ke perangkat seperti CPU dan GPU.
- Sesi menyediakan metode untuk mengeksekusi operasi grafik.

Untuk instalasi, Anda dapat mengunjungi https://www.tensorflow.org/install/. Selanjutnya Saya akan membahas topik-topik berikut:



Gambar 1.1 Topik yang dibahas dalam Buku ini

1.1 TENSOR

Sebelum Anda masuk ke *library* TensorFlow, mari kita pelajari unit dasar data di TensorFlow. Tensor adalah objek matematika dan generalisasi dari skalar, vektor, dan matriks. Tensor dapat direpresentasikan sebagai array multidimensi. Tensor peringkat nol (urutan) adalah skalar. Vektor/array adalah tensor peringkat 1, sedangkan matriks adalah

tensor peringkat 2. Singkatnya, tensor dapat dianggap sebagai array *n*-dimensi. Berikut adalah beberapa contoh tensor:

- 5: Ini adalah tensor peringkat 0; ini adalah skalar dengan bentuk [ ].
- [2.,5., 3.]: Ini adalah tensor peringkat 1; ini adalah vektor dengan bentuk [3].
- [[1., 2., 7.], [3., 5., 4.]]: Ini adalah tensor peringkat 2; itu adalah matriks dengan bentuk [2, 3].
- [[[1., 2., 3.]], [[7., 8., 9.]]]: Ini adalah tensor peringkat 3 dengan bentuk [2, 1, 3].

1.2 GRAFIK DAN SESI KOMPUTASI

TensorFlow populer dengan program TensorFlow Core-nya, ini memiliki dua tindakan utama.

- Membangun grafik komputasi dalam tahap konstruksi
- Menjalankan grafik komputasi dalam fase eksekusi

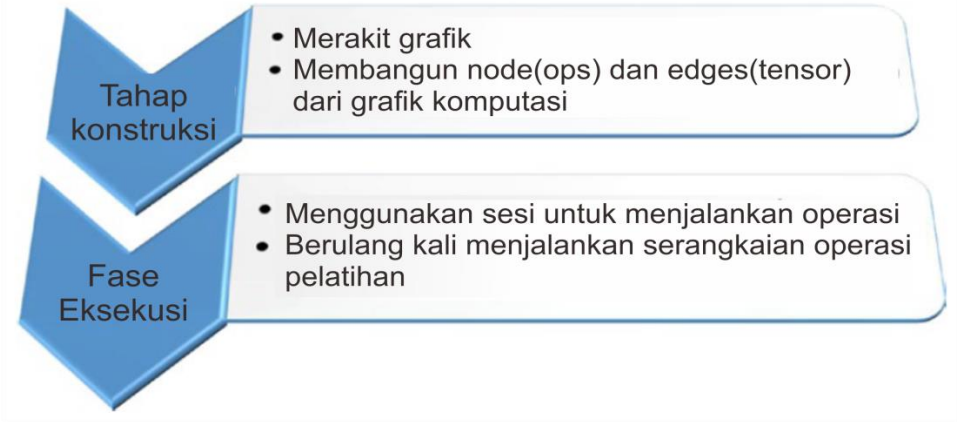
Mari kita pahami cara kerja TensorFlow.

- Program-programnya biasanya disusun menjadi fase konstruksi dan fase eksekusi.
- Tahap konstruksi merakit sebuah graf yang memiliki simpul (ops/operasi) dan sisi (tensor).
- Fase eksekusi menggunakan sesi untuk mengeksekusi ops (operasi) dalam grafik.

Operasi paling sederhana adalah konstanta yang meneruskan output ke operasi lain yang sedang melakukan komputasi.

- Contoh operasi: perkalian (atau penjumlahan atau pengurangan yang menggunakan dua matriks sebagai input dan melewatkan sebuah matriks sebagai output).
- Library TensorFlow memiliki grafik default tempat konstruktor ops menambahkan node.

Jadi, struktur program TensorFlow memiliki dua fase, yang ditunjukkan di sini:



Gambar 1.2 Fase dalam TensorFlow

Grafik komputasi adalah serangkaian operasi TensorFlow yang disusun menjadi grafik node. Mari kita lihat TensorFlow versus Numpy. Di Numpy, jika Anda berencana untuk mengalikan dua matriks, Anda harus membuat matriks dan mengalikannya. Namun di TensorFlow, Anda menyiapkan grafik (grafik default kecuali Anda membuat grafik lain). Selanjutnya, Anda perlu membuat variabel, *placeholder,* dan nilai konstan, lalu membuat sesi dan menginisialisasi variabel. Terakhir, Anda memasukkan data tersebut ke *placeholder* untuk menjalankan tindakan apa pun.

Untuk benar-benar mengevaluasi node, Anda harus menjalankan grafik komputasi dalam satu sesi. Sesi merangkum kontrol dan status *runtime* TensorFlow. Kode berikut membuat objek Sesi:

sess = tf.Session()

Kemudian memanggil metode run untuk menjalankan grafik komputasi yang cukup untuk mengevaluasi node1 dan node2. Grafik komputasi mendefinisikan komputasi. Itu tidak menghitung apa pun atau memegang nilai apa pun. Ini dimaksudkan untuk mendefinisikan operasi yang disebutkan dalam kode. Grafik default dibuat. Jadi, Anda tidak perlu membuatnya kecuali Anda ingin membuat grafik untuk tujuan lain. Sesi memungkinkan Anda untuk mengeksekusi grafik atau bagian grafik. Ini mengalokasikan sumber daya (pada satu atau lebih CPU atau GPU) untuk eksekusi. Ini memegang nilai aktual dari hasil dan variabel antara. Nilai variabel, yang dibuat di TensorFlow, hanya valid dalam satu sesi. Jika Anda mencoba mengkueri nilai setelahnya di sesi kedua, TensorFlow akan memunculkan kesalahan karena variabel tidak diinisialisasi di sana. Untuk menjalankan operasi apa pun, Anda perlu membuat sesi untuk grafik itu. Sesi juga akan mengalokasikan memori untuk menyimpan nilai variabel saat ini.
Berikut adalah kode untuk menunjukkan:

```
import tensorflow as tf
sess = tf.Session()
```

```
# Creating a new graph(not default)
myGraph = tf.Graph()
with myGraph.as_default():
    variable = tf.Variable(30, name="navin")
    initialize = tf.global_variables_initializer()
```

```
with tf.Session(graph=myGraph) as sess:
    sess.run(initialize)
    print(sess.run(variable))
```

```
30
```

```
# Tensorboard can be used. It is optionalmy_
# Output graph can be seen on tensorboard
import os
merged = tf.summary.merge_all(key='summaries')
if not os.path.exists('tenosrboard_logs/'):
    os.makedirs('tenosrboard_logs/')

my_writer = tf.summary.FileWriter('/home/manaswi/tenosrboard_logs/', sess.graph)

def TB(cleanup=False):
    import webbrowser
    webbrowser.open('http://127.0.1.1:6006')
    !tensorboard --logdir='/home/manaswi/tenosrboard_logs'

    if cleanup:
        !rm -R tensorboard_logs/

TB(1)        # Launch graph on tensorborad on your browser
```

1.3 KONSTANTA, PLACEHOLDER, DAN VARIABEL

Program TensorFlow menggunakan struktur data tensor untuk mewakili semua data. Anda dapat menganggap tensor dari TensorFlow sebagai himpunan *n*-dimensi atau daftar *n*-

dimensi. Tensor memiliki tipe statis, peringkat, dan bentuk. Di sini grafik menghasilkan hasil yang konstan. Variabel mempertahankan status di seluruh eksekusi grafik.

Secara umum, Anda akan terus berurusan dengan banyak gambar dalam *Deep learning,* jadi Anda harus menempatkan nilai piksel dengan tepat untuk setiap gambar dan terus mengulanginya. Untuk melatih model, Anda harus mampu memodifikasi grafik untuk menyetel beberapa objek seperti bobot dan bias. Singkatnya, variabel memungkinkan Anda menambahkan parameter yang dapat dilatih ke grafik. Mereka dibangun dengan tipe dan nilai awal. Mari membuat konstanta di TensorFlow dan mencetaknya.

```
import tensorflow as tf
x = tf.constant(12, dtype='float32')
sess = tf.Session()
print(sess.run(x))
```

Berikut adalah penjelasan dari kode sebelumnya secara sederhana:

1. Impor modul tensorflow dan beri nama tf.
2. Buat nilai konstanta (x) dan berikan nilai numerik 12.
3. Buat sesi untuk menghitung nilai.
4. Jalankan variabel x saja dan cetak nilainya saat ini.

Dua langkah pertama termasuk dalam fase konstruksi, dan dua langkah terakhir termasuk dalam fase eksekusi. Saya akan membahas fase konstruksi dan eksekusi TensorFlow sekarang. Anda dapat menulis ulang kode sebelumnya dengan cara lain, seperti yang ditunjukkan di sini:

```
import tensorflow as tf
x = tf.constant(12, dtype='float32')
with tf.Session() as sess:
    print(sess.run(x))
```

Sekarang Anda akan menjelajahi bagaimana Anda membuat variabel dan menginisialisasinya. Berikut adalah kode yang melakukannya:

```
import tensorflow as tf
x = tf.constant(12, dtype='float32')
y = tf.Variable(x+11)
model = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(model)
    print(sess.run(y))
```

```
23.0
```

Berikut penjelasan dari kode sebelumnya:

1. Impor modul tensorflow dan beri nama tf.
2. Buat nilai konstanta yang disebut x dan berikan nilai numeriknya 12.
3. Buat variabel bernama y dan definisikan sebagai persamaan 12+11.
4. Inisialisasi variabel dengan tf.global_variables_ initializer().
5. Buat sesi untuk menghitung nilai.
6. Jalankan model yang dibuat pada langkah 4.
7. Jalankan variabel y saja dan cetak nilainya saat ini.

Pada kode berikut ini coba Anda teliti lagi:

```
import tensorflow as tf
x = tf.constant([14, 23, 40, 30])
y = tf.Variable(x*2 + 100)
model = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(model)
    print(sess.run(y))
```

```
[128 146 180 160]
```

1.4 *PLACEHOLDER*

Placeholder adalah variabel yang dapat Anda beri makan sesuatu di lain waktu. Ini dimaksudkan untuk menerima input eksternal. *Placeholder* dapat memiliki satu atau beberapa dimensi, ini dimaksudkan untuk menyimpan himpunan *n*-dimensi.

```
import tensorflow as tf
x = tf.placeholder("float", None)
y = x*10 + 500
with tf.Session() as sess:
    placeX = sess.run(y, feed_dict={x: [0, 5, 15, 25]})
    print(placeX)
```

```
[500. 550. 650. 750.]
```

Berikut penjelasan dari kode sebelumnya:

1. Impor modul tensorflow dan beri nama tf.
2. Buat *placeholder* bernama x, dengan menyebutkan float type.
3. Buat tensor bernama y yang merupakan operasi perkalian x dengan 10 dan dijumlahkan 500. Perhatikan bahwa setiap nilai awal untuk x tidak ditentukan.
4. Buat sesi untuk menghitung nilai.
5. Tentukan nilai x di feed_dict untuk menjalankan y.
6. Cetak nilainya.

Dalam contoh berikut, Anda membuat matriks 2x4 (array 2D) untuk menyimpan beberapa angka di dalamnya. Anda kemudian menggunakan operasi yang sama seperti sebelumnya untuk mengalikan elemen dengan 10 dan menambahkan 1 ke dalamnya. Dimensi pertama dari *placeholder* adalah None, yang berarti sejumlah baris diperbolehkan. Anda juga dapat mempertimbangkan larik 2D sebagai pengganti larik 1D. Berikut kodenya:

```
import tensorflow as tf
x = tf.placeholder("float", [None, 4])
y = x*10 + 1
with tf.Session() as sess:
    dataX = [[12, 2, 0, -2],
             [14, 4, 1, 0]]
    placeX = sess.run(y, feed_dict={x: dataX})
    print(placeX)
```

```
[[121.  21.   1. -19.]
 [141.  41.  11.   1.]]
```

Ini adalah matriks 2x4. Jadi, jika Anda mengganti None dengan 2, Anda dapat melihat output yang sama.

```
import tensorflow as tf
x = tf.placeholder("float", [2, 4])
y = x*10 + 1
with tf.Session() as sess:
    dataX = [[12, 2, 0, -2],
             [14, 4, 1, 0]]
    placeX = sess.run(y, feed_dict={x: dataX})
    print(placeX)
```

```
[[121.  21.   1. -19.]
 [141.  41.  11.   1.]]
```

Tetapi jika Anda membuat *placeholder* dengan bentuk [3, 4] (perhatikan bahwa Anda akan memasukkan matriks 2x4 di lain waktu), ada kesalahan, seperti yang ditunjukkan di sini:

```
import tensorflow as tf
x = tf.placeholder("float", [3, 4])
y = x*10 + 1
with tf.Session() as sess:
    dataX = [[12, 2, 0, -2],
             [14, 4, 1, 0]]
    placeX = sess.run(y, feed_dict={x: dataX})
    print(placeX)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-c70a14b67e27> in <module>()
      5     dataX = [[12, 2, 0, -2],
      6              [14, 4, 1, 0]]
----> 7     placeX = sess.run(y, feed_dict={x: dataX})
      8     print(placeX)

~\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\client\session.py in run(self, fetches, feed_dict, options, run
_metadata)
    887     try:
    888       result = self._run(None, fetches, feed_dict, options_ptr,
--> 889                          run_metadata_ptr)
    890       if run_metadata:
    891         proto_data = tf_session.TF_GetBuffer(run_metadata_ptr)

~\Anaconda3\envs\tensorflow\lib\site-packages\tensorflow\python\client\session.py in _run(self, handle, fetches, feed_dict, opt
ions, run_metadata)
   1094                 'Cannot feed value of shape %r for Tensor %r, '
   1095                 'which has shape %r'
-> 1096                 % (np_val.shape, subfeed_t.name, str(subfeed_t.get_shape())))
   1097           if not self.graph.is_feedable(subfeed_t):
   1098             raise ValueError('Tensor %s may not be fed.' % subfeed_t)

ValueError: Cannot feed value of shape (2, 4) for Tensor 'Placeholder_5:0', which has shape '(3, 4)'
```

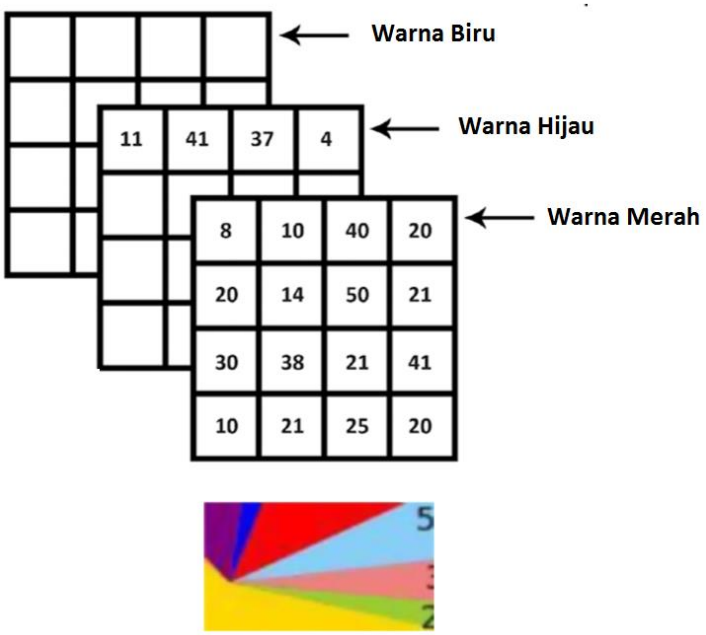
Bobot dan Bias sebagai Variabel yang harus disetel
W = tf.Variable([2], dtype=tf.float32)
b = tf.Variable([3], dtype=tf.float32)
Training dataset yang akan diberi makan saat pelatihan sebagai Placeholder
x = tf.placeholder(tf.float32)
Linear Model y = W * x + b
Konstanta diinisialisasi saat Anda memanggil tf.constant, dan nilainya tidak akan pernah bisa berubah. Sebaliknya, variabel tidak diinisialisasi saat Anda memanggil tf.Variable. Untuk menginisialisasi semua variabel dalam program TensorFlow, Anda harus secara eksplisit memanggil operasi khusus sebagai berikut.

```
sess.run(tf.global_variables_initializer())
```

Penting untuk disadari bahwa init adalah pegangan untuk subgraf TensorFlow yang menginisialisasi semua variabel global. Sampai Anda memanggil sess.run, variabel tidak diinisialisasi.

1.5 MEMBUAT TENSOR

Sebuah gambar adalah tensor dari urutan ketiga di mana dimensi milik tinggi, lebar, dan jumlah saluran (Merah, Biru, dan Hijau). Di sini Anda dapat melihat bagaimana gambar diubah menjadi tensor:



Gambar 1.3 Gambar yang diubah menjadi Tensor

```
image = tf.image.decode_jpeg(tf.read_file("./Desktop/image.jpg"), channels=3)
sess = tf.InteractiveSession()
print(sess.run(tf.shape(image)))
```

```
[218 178   3]
```

```
print(sess.run(image[10:15,0:4,1]))
```

```
[[47 48 48 47]
 [45 45 45 44]
 [43 43 43 42]
 [41 42 42 41]
 [41 41 41 40]]
```

Anda dapat menghasilkan berbagai jenis tensor seperti tensor tetap, tensor acak, dan tensor sekuensial.

Tensor Tetap

Berikut adalah tensor tetap:

```
import tensorflow as tf
sess = tf.Session()
A = tf.zeros([2,3])
print(sess.run(A))
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

```
B = tf.ones([4,3])
print(sess.run(B))
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

```
import tensorflow as tf
sess = tf.Session()
A = tf.zeros([2,3])
print(sess.run(A))
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

2 Rows
3 Columns

```
B = tf.ones([4,3])
print(sess.run(B))
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

4 Rows
3 Columns

tf:.fill membuat tensor bentuk (2×3) yang memiliki nomor unik.

```
C = tf.fill([2,3], 13)
print(sess.run(C))
```

```
[[13 13 13]
 [13 13 13]]
```

tf.diag membuat matriks diagonal yang memiliki elemen diagonal tertentu.

```
D = tf.diag([4,-3,2])
print(sess.run(D))
```

```
[[ 4  0  0]
 [ 0 -3  0]
 [ 0  0  2]]
```

tf.constant membuat tensor konstan.

```
E = tf.constant([5,2,4,2])
print(sess.run(E))
```

```
[5 2 4 2]
```

Urutan tensor

tf.range membuat urutan angka mulai dari nilai yang ditentukan dan memiliki kenaikan yang ditentukan.

```
G = tf.range(start=6, limit=45, delta=3)
print(sess.run(G))
```

```
[ 6  9 12 15 18 21 24 27 30 33 36 39 42]
```

tf.linspace membuat urutan nilai dengan jarak yang sama.

```
H = tf.linspace(10.0, 92.0, 5)
print(sess.run(H))
```

```
[10.  30.5 51.  71.5 92. ]
```

Tensor Acak

tf.random_uniform menghasilkan nilai acak dari distribusi yang seragam dalam suatu rentang.

```
R1 = tf.random_uniform([2,3], minval=0, maxval=4)
print(sess.run(R1))
```

```
[[0.74450636 1.9570832  3.1126966 ]
 [2.359518   2.101438   2.65689   ]]
```

tf.random_normal menghasilkan nilai acak dari distribusi normal yang memiliki mean dan standar deviasi yang ditentukan.

```
R2 = tf.random_normal([2,3], mean=5, stddev=4)
print(sess.run(R2))
```

```
[[-1.8996243  3.2514744  5.9602127]
 [ 8.307009   4.84437    6.8460846]]
```

```
print(sess.run(tf.diag([3,-2,4])))
```

```
[[ 3  0  0]
 [ 0 -2  0]
 [ 0  0  4]]
```

```
R3 = tf.random_shuffle(tf.diag([3,-2,4]))
print(sess.run(R3))
```

```
[[ 3  0  0]
 [ 0  0  4]
 [ 0 -2  0]]
```

```
R4 = tf.random_crop(tf.diag([3,-2,4]), [3,2])
print(sess.run(R4))
```

```
[[ 3  0]
 [ 0 -2]
 [ 0  0]]
```

Bisakah Anda menebak hasilnya?

```
print(sess.run(tf.zeros([2,4])))
```

```
print(sess.run(tf.diag([3,1,5,-2])))
```

```
print(sess.run(tf.range(start=4, limit=16, delta=2)))
```

Jika Anda tidak dapat menemukan hasilnya, silakan merevisi bagian sebelumnya di mana saya membahas pembuatan tensor. Di sini Anda dapat melihat hasilnya:

```
print(sess.run(tf.zeros([2,4])))
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

```
print(sess.run(tf.diag([3,1,5,-2])))
```

```
[[ 3  0  0  0]
 [ 0  1  0  0]
 [ 0  0  5  0]
 [ 0  0  0 -2]]
```

```
print(sess.run(tf.range(start=4, limit=16, delta=2)))
```

```
[ 4  6  8 10 12 14]
```

1.6 BEKERJA PADA MATRIKS

Setelah Anda merasa nyaman membuat tensor, Anda dapat menikmati mengerjakan matriks (tensor 2D).

```
import tensorflow as tf
import numpy as np
sess = tf.Session()
A = tf.random_uniform([3,2])
B = tf.fill([2,4], 3.5)
C = tf.random_normal([3,4])
```

```
print(sess.run(A))
```

```
[[0.31633115 0.71407604]
 [0.18088198 0.36230946]
 [0.34481096 0.6156665 ]]
```

```
print(sess.run(B))
```

```
[[3.5 3.5 3.5 3.5]
 [3.5 3.5 3.5 3.5]]
```

```
print(sess.run(tf.matmul(A,B)))# Multiplication of Matrices
```

```
[[0.9453191 0.9453191 0.9453191 0.9453191]
 [4.4488316 4.4488316 4.4488316 4.4488316]
 [3.308284  3.308284  3.308284  3.308284 ]]
```

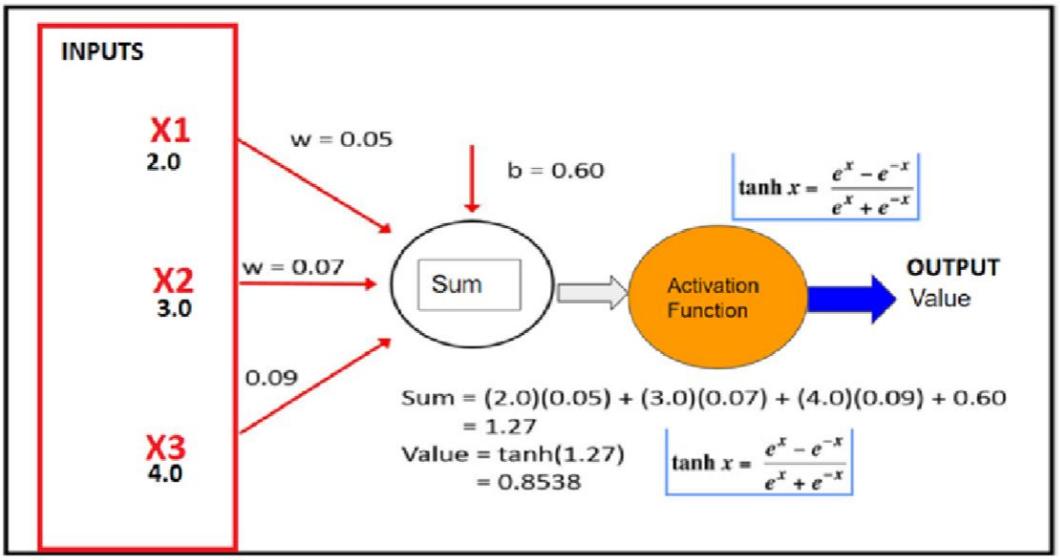
```
print(sess.run(tf.matmul(A,B) + C))# Multiplication & addition
```

```
[[4.6027136 4.5958595 6.9527874 4.413632 ]
 [3.3000264 4.4702578 5.0858393 5.168917 ]
 [3.1176403 4.626109  4.1446424 3.7285264]]
```

1.7 FUNGSI AKTIVASI

Ide fungsi aktivasi berasal dari analisis cara kerja neuron di otak manusia (lihat Gambar 1.4). Neuron menjadi aktif melampaui ambang batas tertentu, lebih dikenal sebagai

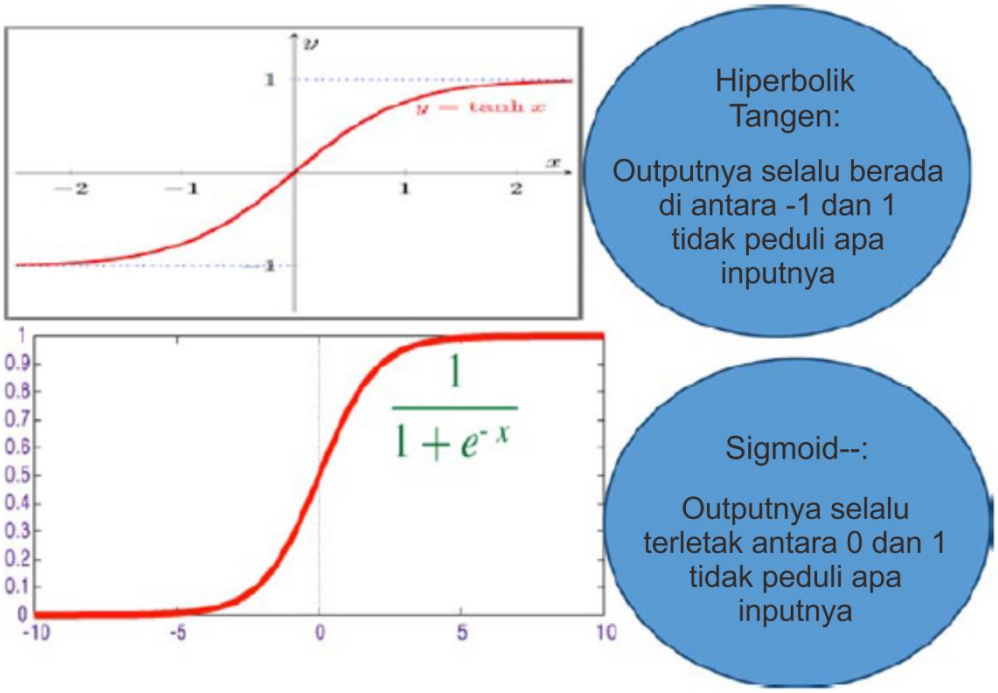
potensial aktivasi. Itu juga mencoba untuk menempatkan output ke dalam kisaran kecil dalam banyak kasus. Sigmoid, tangen hiperbolik (tan), ReLU, dan ELU adalah fungsi aktivasi yang paling populer. Mari kita lihat fungsi aktivasi yang populer.



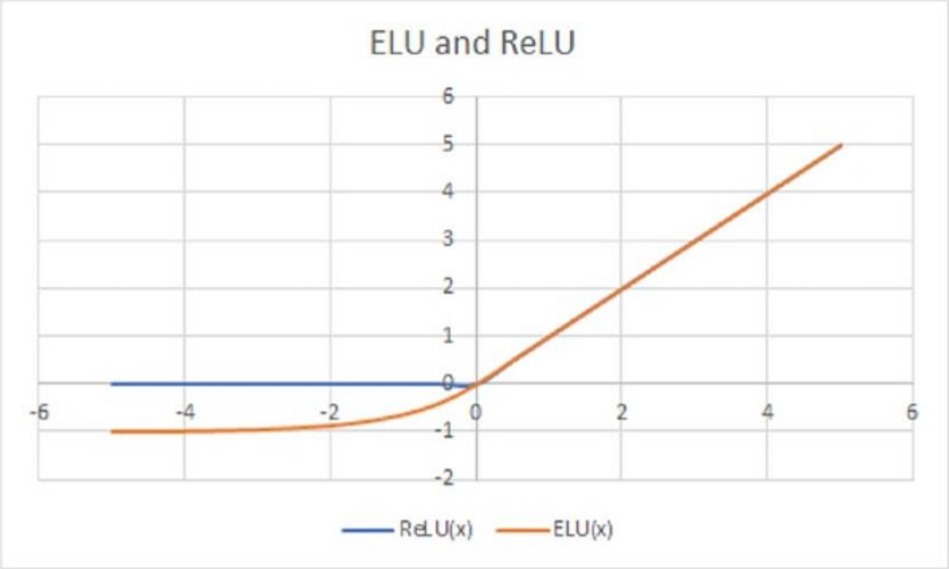
Gambar 1.4 Fungsi aktivasi

Tangen Hiperbolik dan Sigmoid

Gambar 1.5 menunjukkan fungsi aktivasi tangen hiperbolik dan sigmoid.



Gambar 1.5 Dua fungsi aktivasi populer



Gambar 1.6. Fungsi ReLU dan ELU

Kode berikut ini adalah kode demo :

```
E = tf.nn.tanh([10,2,1,0.5,0,-0.5,-1.,-2.,-10.])
print(sess.run(E))
```

```
[ 1.          0.9640276   0.7615942   0.46211717  0.         -0.46211717
 -0.7615942  -0.9640276  -1.        ]
```

```
J = tf.nn.sigmoid([10,2,1,0.5,0,-0.5,-1.,-2.,-10.])
print(sess.run(J))
```

```
[9.9995458e-01 8.8079703e-01 7.3105860e-01 6.2245935e-01 5.0000000e-01
 3.7754068e-01 2.6894143e-01 1.1920292e-01 4.5397872e-05]
```

ReLU dan ELU

Gambar 1.6 menunjukkan fungsi ReLU dan ELU.
Berikut adalah kode untuk menghasilkan fungsi-fungsi ini:

```
A = tf.nn.relu([-2,1,-3,13])
print(sess.run(A))
```

```
[ 0  1  0 13]
```

ReLU6

ReLU6 mirip dengan ReLU kecuali outputnya tidak boleh lebih dari enam.

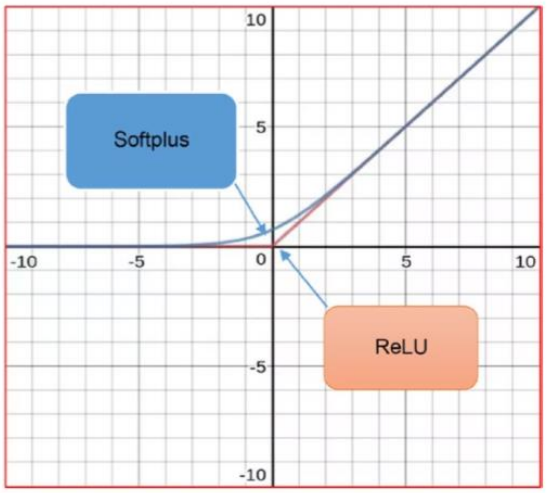
```
B = tf.nn.relu6([-2,1,-3,13])
print(sess.run(B))
```

```
[0 1 0 6]
```

```
C = tf.nn.relu([[-2,1,-3],[10,-16,-5]])
print(sess.run(C))
```

```
[[ 0  1  0]
 [10  0  0]]
```

Perhatikan bahwa tanh adalah fungsi sigmoid logistik yang diskalakan.

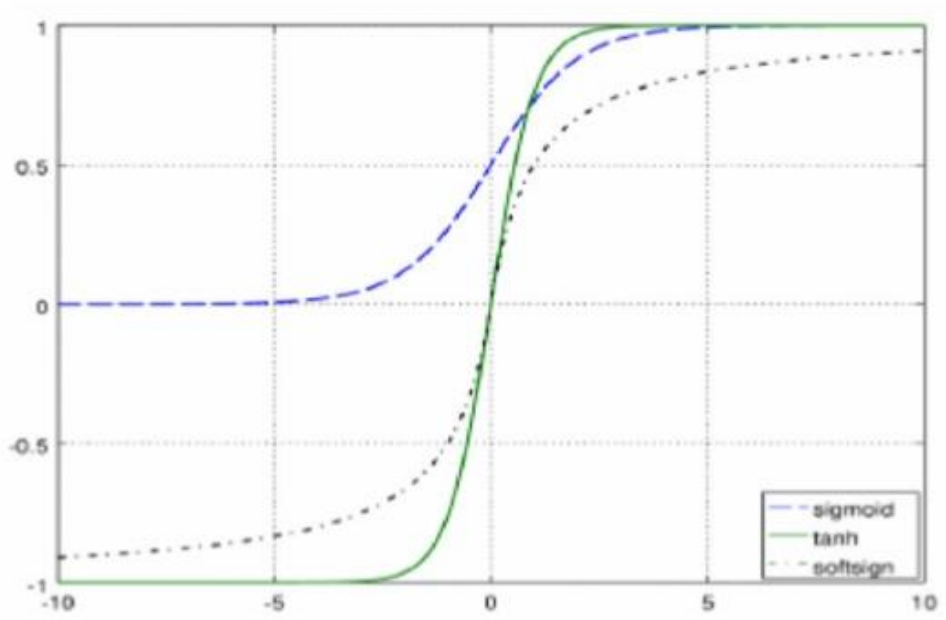



```
K = tf.nn.relu([10,2,1,0.5,0,-0.5,-1.,-2.,-10.])
print(sess.run(K))
```

```
[10.   2.   1.   0.5  0.   0.   0.   0.   0. ]
```

```
M = tf.nn.softplus([10,2,1,0.5,0,-0.5,-1.,-2.,-10.])
print(sess.run(M))
```

```
[1.0000046e+01 2.1269281e+00 1.3132616e+00 9.7407699e-01 6.9314718e-01
 4.7407699e-01 3.1326163e-01 1.2692805e-01 4.5417706e-05]
```




```
H = tf.nn.elu([10,2,1,0.5,0,-0.5,-1.,-2.,-10.])
print(sess.run(H))
```

```
[10.         2.         1.         0.5        0.        -0.39346933
 -0.63212055 -0.86466473 -0.9999546 ]
```

```
I = tf.nn.relu6([10,2,1,0.5,0,-0.5,-1.,-2.,-10.])
print(sess.run(I))
```

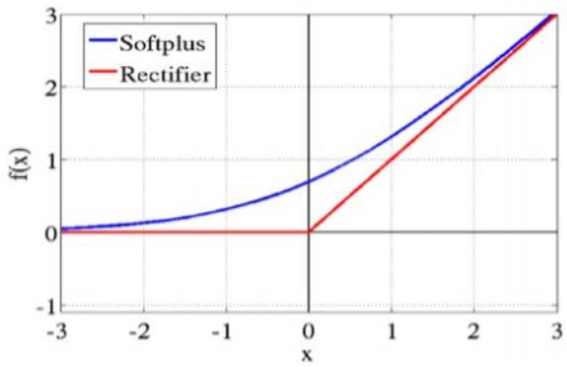
```
[6.  2.  1.  0.5 0.  0.  0.  0.  0. ]
```

```
G = tf.nn.softsign([10,2,1,0.5,0,-0.5,-1.,-2.,-10.])
print(sess.run(G))
```

```
[ 0.90909094  0.6666667   0.5         0.33333334  0.         -0.33333334
 -0.5        -0.6666667  -0.90909094]
```

```
F = tf.nn.softplus([10,2,1,0.5,0,-0.5,-1.,-2.,-10.])
print(sess.run(F))
```

```
[1.0000046e+01 2.1269281e+00 1.3132616e+00 9.7407699e-01 6.9314718e-01
 4.7407699e-01 3.1326163e-01 1.2692805e-01 4.5417706e-05]
```



1.7 FUNGSI LOSS

Fungsi loss (fungsi biaya) harus diminimalkan sehingga mendapatkan nilai terbaik untuk setiap parameter model. Misalnya, Anda perlu mendapatkan nilai terbaik dari bobot (slope) dan bias (y-intercept) untuk menjelaskan target (y) dalam bentuk prediktor (X). Metodenya adalah untuk mencapai nilai kemiringan yang terbaik, dan perpotongan y adalah untuk meminimalkan fungsi biaya/fungsi loss/jumlah kuadrat. Untuk model apapun, ada banyak parameter, dan struktur model dalam prediksi atau klasifikasi dinyatakan dalam nilai parameter.

Anda perlu mengevaluasi model Anda, dan untuk itu Anda perlu mendefinisikan fungsi biaya (fungsi loss). Minimisasi fungsi loss dapat menjadi kekuatan pendorong untuk menemukan nilai optimal dari setiap parameter. Untuk prediksi egress/numerik, L1 atau L2 dapat menjadi fungsi loss yang berguna. Untuk klasifikasi, entropi silang dapat menjadi fungsi loss yang berguna. Softmax atau sigmoid cross entropy bisa menjadi fungsi loss yang cukup populer.

Contoh Fungsi Loss

Berikut merupakan kode untuk menunjukkan fungsi Loss:

```
import tensorflow as tf
import numpy as np
sess = tf.Session()
```

```
#Assuming prediction model
pred=np.asarray([0.2,0.3,0.5,10.0,12.0,13.0,3.5,7.4,3.9,2.3])
#convert ndarray into tensor
x_val=tf.convert_to_tensor(pred)
#Assuming actual values
actual=np.asarray([0.1,0.4,0.6,9.0,11.0,12.0,3.4,7.1,3.8,2.0])
```

```
#L2 loss:L1=(pred-actual)^2
l2=tf.square(pred-actual)
l2_out=sess.run(tf.round(l2))
print(l2_out)
```

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

```
#L2 loss:L1=abs(pred-actual)
l1=tf.abs(pred-actual)
l1_out=sess.run(l1)
print(l1_out)
```

```
[ 0.1  0.1  0.1  1.   1.   1.   0.1  0.3  0.1  0.3]
```

```
#cross entropy loss
softmax_xentropy_variable=tf.nn.sigmoid_cross_entropy_with_logits(logits=l1_out,labels=l2_out)
print(sess.run(softmax_xentropy_variable))
```

```
[ 0.74439666  0.74439666  0.74439666  0.31326169  0.31326169  0.31326169
  0.74439666  0.85435524  0.74439666  0.85435524]
```

Berikut ini adalah daftar fungsi loss yang paling umum:

tf.contrib.losses.absolute_difference
tf.contrib.losses.add_loss
tf.contrib.losses.hinge_loss
tf.contrib.losses.compute_weighted_loss
tf.contrib.losses.cosine_distance
tf.contrib.losses.get_losses
tf.contrib.losses.get_regularization_losses
tf.contrib.losses.get_total_loss
tf.contrib.losses.log_loss
tf.contrib.losses.mean_pairwise_squared_error
tf.contrib.losses.mean_squared_error
tf.contrib.losses.sigmoid_cross_entropy
tf.contrib.losses.softmax_cross_entropy
tf.contrib.losses.sparse_softmax_cross_entropy
tf.contrib.losses.log(predictions,labels,weight=2.0)

1.9 PENGOPTIMAL

Sekarang Anda harus yakin bahwa Anda perlu menggunakan fungsi loss untuk mendapatkan nilai terbaik dari setiap parameter model. Bagaimana Anda bisa mendapatkan nilai terbaik? Pada bagian awal Anda harus mengasumsikan nilai awal bobot dan bias untuk model (regresi linier, dll.), lalu sekarang Anda perlu menemukan cara untuk mencapai nilai parameter terbaik. *Optimizer* adalah cara untuk mencapai nilai parameter terbaik. Dalam setiap iterasi, nilai berubah ke arah yang disarankan oleh Optimizer.

Misalkan Anda memiliki 16 nilai bobot (w1, w2, w3, …, w16) dan 4 bias (b1, b2, b3, b4). Dibagian awal Anda dapat mengasumsikan setiap bobot dan bias menjadi nol (atau satu atau angka apa pun), dan selanjutnya *Optimizer* akan memberikan saran pada w1 (dan parameter lainnya), apakah w1 harus meningkat atau menurun pada iterasi berikutnya

sambil tetap mengingat tujuan minimalisasi. Setelah banyak iterasi, w1 (dan parameter lainnya) akan stabil ke nilai parameter terbaik.

Dengan kata lain, TensorFlow, dan setiap *framework Deep learning* lainnya menyediakan *Optimizer* yang secara perlahan mengubah setiap parameter untuk meminimalkan fungsi loss. Tujuan dari Optimizeran adalah untuk memberikan arah pada bobot dan bias untuk perubahan pada iterasi berikutnya. Asumsikan bahwa Anda memiliki 64 bobot dan 16 bias; Anda mencoba mengubah nilai bobot dan bias di setiap iterasi (selama backpropagation) sehingga Anda mendapatkan nilai bobot dan bias yang benar setelah banyak iterasi sambil mencoba meminimalkan fungsi loss.

Memilih *Optimizer* terbaik agar model dapat menyatu dengan cepat dan mempelajari bobot dan bias dengan benar merupakan tugas yang sangat rumit. Teknik adaptif (adadelta, adagrad, dll.) adalah *Optimizer* yang baik untuk konvergen lebih cepat untuk jaringan saraf kompleks. Adam seharusnya adalah *Optimizer* terbaik untuk sebagian besar kasus. Ini juga mengungguli teknik adaptif lainnya (adadelta, adagrad, dll.), tetapi secara komputasi mahal. Untuk kumpulan data yang jarang, metode seperti SGD, NAG, dan momentum bukanlah pilihan terbaik; metode tingkat pembelajaran adaptif adalah. Manfaat tambahannya adalah Anda tidak perlu menyesuaikan kecepatan pembelajaran tetapi kemungkinan besar dapat mencapai hasil terbaik dengan nilai default.

Contoh Fungsi Loss

Berikut adalah contoh kode untuk menunjukkan fungsi loss:

```
# Importing libraries
import tensorflow as tf
```

```
# Assign the value into variable
x = tf.Variable(3, name='x', dtype=tf.float32)
log_x = tf.log(x)
log_x_squared = tf.square(log_x)
```

```
# Apply GradientDescentOptimizer
optimizer = tf.train.GradientDescentOptimizer(0.7)
train = optimizer.minimize(log_x_squared)
```

```
# Initialize Variables
init = tf.global_variables_initializer()
```

```
# Finally running computation
with tf.Session() as session:
    session.run(init)
    print("starting at", "x:", session.run(x), "log(x)^2:", session.run(log_x_squared))
    for step in range(10):
        session.run(train)
        print("step", step, "x:", session.run(x), "log(x)^2:", session.run(log_x_squared))
```

```
starting at x: 3.0 log(x)^2: 1.20695
step 0 x: 2.48731 log(x)^2: 0.830292
step 1 x: 1.97444 log(x)^2: 0.462786
step 2 x: 1.49207 log(x)^2: 0.160134
step 3 x: 1.1166 log(x)^2: 0.0121637
step 4 x: 0.97832 log(x)^2: 0.00048043
step 5 x: 1.00969 log(x)^2: 9.29177e-05
step 6 x: 0.99632 log(x)^2: 1.35901e-05
step 7 x: 1.0015 log(x)^2: 2.24809e-06
step 8 x: 0.999405 log(x)^2: 3.54772e-07
step 9 x: 1.00024 log(x)^2: 5.70574e-08
```

Optimizer Umum

Berikut ini adalah daftar optimizer umum:

tf.train.Optimizer
tf.train.GradientDescentOptimizer
tf.train.AdadeltaOptimizer
tf.train.AdagradOptimizer
tf.train.AdagradDAOptimizer
tf.train.MomentumOptimizer
tf.train.AdamOptimizer
tf.train.FtrlOptimizer
tf.train.ProximalGradientDescentOptimizer
tf.train.ProximalAdagradOptimizer
tf.train.RMSPropOptimizer

1.10 METRIK

Setelah Anda mempelajari beberapa cara untuk membangun model, sekarang saatnya untuk mengevaluasi model. Jadi, Anda perlu mengevaluasi regressor atau classifier. Ada banyak metrik evaluasi, di antaranya (yang pling populer) adalah akurasi klasifikasi, kerugian logaritmik, dan area di bawah kurva ROC.

Akurasi klasifikasi adalah rasio jumlah prediksi yang benar dengan jumlah semua prediksi. Ketika pengamatan untuk setiap kelas tidak terlalu miring, akurasi dapat dianggap sebagai metrik yang baik.

tf.contrib.metrics.accuracy(label_aktual, prediksi)

Selain diatas, masih ada metrik evaluasi lainnya.

Contoh Metrik

Bagian ini menunjukkan kode untuk didemonstrasikan. Di sini Anda membuat nilai aktual (sebut saja x) dan nilai prediksi (sebut saja y). Kemudian Anda memeriksa keakuratannya. Akurasi mewakili rasio berapa kali aktual sama dengan nilai yang diprediksi dan jumlah total instance.

Metrik Umum

Berikut ini adalah daftar metrik umum:

tf.contrib.metrics.streaming_root_mean_squared_error
tf.contrib.metrics.streaming_covariance
tf.contrib.metrics.streaming_pearson_correlation
tf.contrib.metrics.streaming_mean_cosine_distance
tf.contrib.metrics.streaming_percentage_less
tf.contrib.metrics.streaming_sensitivity_at_specificity
tf.contrib.metrics.streaming_sparse_average_precision_at_k
tf.contrib.metrics.streaming_sparse_precision_at_k tf.contrib.metrics.streaming_sparse_precision_at_top_k
tf.contrib.metrics.streaming_specificity_at_sensitivity
tf.contrib.metrics.streaming_concat
tf.contrib.metrics.streaming_false_negatives
tf.contrib.metrics.streaming_false_negatives_at_thresholds

```
# Importing libraries
import numpy as np
import tensorflow as tf
```

```
# Placeholders declaration
x=tf. placeholder(tf.int32, [5])
y=tf. placeholder(tf.int32, [5])
```

```
# Metrices declaration
acc, acc_op=tf.metrics.accuracy(labels=x, predictions=y)
```

```
# Session initialization
sess=tf.InteractiveSession()
sess.run(tf.global_variables_initializer())
sess.run(tf.local_variables_initializer())
```

```
# Print Accuracy
val_acc=sess.run(acc)
print(val_acc)
# You can see only 2nd and 3rd positions value are same
```

```
0.4
```

BAB 2
MEMAHAMI DAN BEKERJA DENGAN KERAS

Keras library Python tingkat tinggi yang ringkas, mudah dipelajari untuk *Deep learning* dan dapat berjalan di atas TensorFlow (atau Theano atau CNTK). *Keras* memungkinkan pengembang untuk fokus pada konsep utama *Deep learning*, seperti membuat layer untuk jaringan saraf sambil menjaga detail seluk beluk tensor, bentuknya, dan detail matematisnya. TensorFlow (atau Theano atau CNTK) harus menjadi backend untuk Keras.

Anda dapat menggunakan *Keras* untuk aplikasi *Deep learning* tanpa berinteraksi dengan TensorFlow (atau Theano atau CNTK) yang relatif kompleks. Ada dua jenis kerangka kerja utama: API sekuensial dan API fungsional. API sekuensial didasarkan pada gagasan tentang urutan layer; yang merupakan penggunaan Keras pada bagian yang paling umum dan mudah. Model sekuensial dapat dianggap sebagai tumpukan layer linier.

Singkatnya, Anda membuat model sekuensial di mana Anda dapat dengan mudah menambahkan layer, dan setiap layer dapat memiliki konvolusi, pengumpulan maksimum, aktivasi, *drop-out,* dan normalisasi batch. Mari melalui langkah-langkah utama untuk mengembangkan model *Deep learning* di Keras.

2.1 LANGKAH-LANGKAH UTAMA UNTUK MODEL *DEEP LEARNING*

Empat bagian utama dari model *deep learning* di Keras adalah sebagai berikut:

1. ***Tentukan modelnya***. Di sini Anda membuat model sekuensial dan menambahkan layer. Setiap layer dapat berisi satu atau lebih konvolusi, penyatuan, normalisasi batch, dan fungsi aktivasi.
2. ***Kompilasi modelnya.*** Di sini Anda menerapkan fungsi *loss* dan *Optimizer* sebelum memanggil fungsi compile() pada model.
3. ***Sesuaikan model dengan data pelatihan***. Di sini Anda melatih model pada data pengujian dengan calling the fit() pada model.
4. ***Buat prediksi***. Di sini Anda menggunakan model untuk menghasilkan prediksi pada data baru dengan memanggil fungsi seperti evaluate() dan predict().

Ada delapan langkah proses deep learning di Keras:

1. Muat data.
2. Praproses data.
3. Tentukan model.
4. Kompilasi model.
5. Sesuaikan model.
6. Evaluasi model.
7. Buat prediksi.
8. Simpan model.

Muat Data

Berikut adalah cara memuat data:

```
# Importing modules
import numpy as np
import os
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import adam
from keras.utils import np_utils
```

```
#Load Data
np.random.seed(100) # for reproducibility
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

#cifar-10 has images of airplane, automobile, bird, cat,
# deer, dog, frog, horse, ship and truck ( 10 unique labels)
# For each image. width = 32, height =32, Number of channels(RGB) = 3
```

Praproses Data

Berikut adalah cara Anda memproses data sebelumnya:

```
#Preprocess the data
#Flatten the data, MLP doesn't use the 2D structure of the data. 3072 = 3*32*32
X_train = X_train.reshape(50000, 3072) # 50,000 images for training
X_test = X_test.reshape(10000, 3072) # 10,000 images for test

# Gaussian Normalization( Z- score)
X_train = (X_train- np.mean(X_train))/np.std(X_train)
X_test = (X_test- np.mean(X_test))/np.std(X_test)
```

```
# Convert class vectors to binary class matrices (ie one-hot vectors)
labels = 10 #10 unique labels(0-9)
Y_train = np_utils.to_categorical(y_train, labels)
Y_test = np_utils.to_categorical(y_test, labels)
```

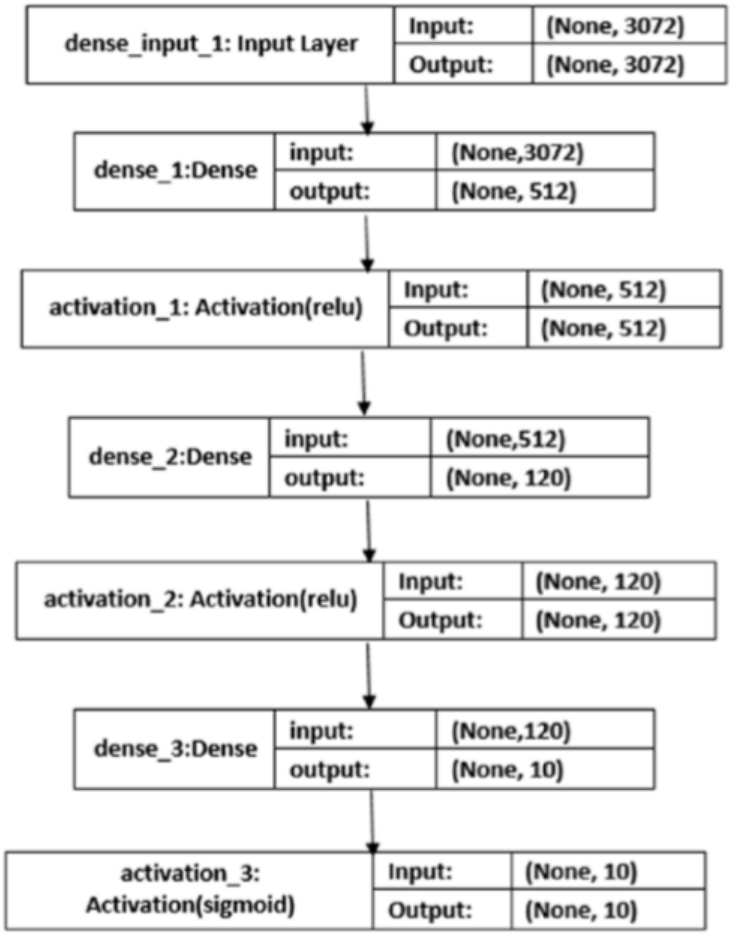
Tentukan Model

Model sekuensial di Keras didefinisikan sebagai *layer sequence*/urutan layer. Anda membuat model sekuensial dan kemudian menambahkan layer. Anda perlu memastikan layer input memiliki jumlah input yang tepat. Asumsikan Anda memiliki 3.072 variabel input; sehingga Anda perlu membuat layer tersembunyi pertama dengan 512 node/neuron. Di layer tersembunyi kedua, Anda memiliki 120 node/neuron. Akhirnya, Anda memiliki sepuluh node di layer output. Misalnya, sebuah gambar dipetakan ke sepuluh node yang menunjukkan kemungkinan menjadi label1 (pesawat terbang), label2 (mobil), label3 (kucing), …, label10 (truk). Node dengan probabilitas tertinggi adalah kelas/label yang diprediksi.

```
#Define the model achitecture
model = Sequential()
model.add(Dense(512, input_shape=(3072,))) # 3*32*32 = 3072
model.add(Activation('relu'))
model.add(Dropout(0.4)) # Regularization
model.add(Dense(120))
model.add(Activation('relu'))
model.add(Dropout(0.2))# Regularization
model.add(Dense(labels)) #Last layer with 10 outputs, each output per class
model.add(Activation('sigmoid'))
```

Satu gambar memiliki tiga saluran (RGB), dan di setiap saluran, gambar memiliki 32x32 = 1024 piksel. Jadi, setiap gambar memiliki 3×1024 = 3072 piksel (fitur/X/input). Dengan bantuan 3.072 fitur, Anda perlu memprediksi probabilitas label1 (Digit 0), label2 (Digit 1),

dan seterusnya. Ini berarti model memprediksi sepuluh output (Digit 0-9) di mana setiap output mewakili probabilitas label yang sesuai. Fungsi aktivasi terakhir (sigmoid, seperti yang ditunjukkan sebelumnya) memberikan 0 untuk sembilan output dan 1 hanya untuk satu output. Label tersebut adalah kelas yang diprediksi untuk gambar (Gambar 2-1). Misalnya, 3.072 fitur →512 node →120 node →10 node.



Gambar 2-1. Mendefinisikan model

Lantas, bagaimana cara mengetahui jumlah layer yang akan digunakan dan jenisnya? Tidak ada yang memiliki jawaban yang tepat. Anda harus memutuskan jumlah layer yang optimal dan parameter serta langkah di setiap layer sebagai metrik evaluasi. Pendekatan heuristik juga harus digunakan. Struktur jaringan terbaik ditemukan melalui proses eksperimen. Umumnya, Anda memerlukan jaringan yang cukup besar untuk menangkap struktur masalah.

Dalam contoh ini, Anda akan menggunakan struktur jaringan yang sepenuhnya terhubung dengan tiga layer. Kelas padat mendefinisikan layer yang terhubung penuh. Dalam hal ini, Anda menginisialisasi bobot jaringan ke angka acak kecil yang dihasilkan dari distribusi seragam (seragam) dalam hal ini antara 0 dan 0,05 karena itu adalah inisialisasi bobot seragam default di Keras. Alternatif tradisional lainnya akan normal untuk bilangan acak kecil yang dihasilkan dari distribusi Gaussian. Anda menggunakan atau snap ke klasifikasi keras dari kedua kelas dengan ambang default 0,5. Anda dapat menggabungkan semuanya dengan menambahkan setiap layer.

Kompilasi Model

Setelah mendefinisikan model dalam layer, Anda perlu mendeklarasikan fungsi loss, *Optimizer,* dan metrik evaluasi. Ketika model diusulkan, bobot awal dan nilai bias diasumsikan 0 atau 1, angka terdistribusi normal acak, atau angka nyaman lainnya. Tetapi nilai awal bukanlah nilai terbaik untuk model. Artinya nilai awal bobot dan bias tidak mampu menjelaskan target/label dalam bentuk prediktor (Xs). Jadi, Anda ingin mendapatkan nilai optimal untuk model tersebut. Perjalanan dari nilai awal ke nilai optimal membutuhkan motivasi, yang akan meminimalkan fungsi biaya/fungsi *loss*. Perjalanan membutuhkan jalur (perubahan di setiap iterasi), yang disarankan oleh *Optimizer.* Perjalanan juga membutuhkan pengukuran evaluasi, atau metrik evaluasi.

```
# Compile the model
# Use adam as an optimizer
adam = adam(0.01)
# the cross entropy between the true label and the output(softmax) of the model
model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=["accuracy"])
```

Fungsi loss yang populer adalah *binary cross entropy, categorical cross entropy, mean_squared_logarithmic_error, dan hinge loss*. Optimizer populer adalah *stochastic gradient descent /*penurunan gradien stokastik (SGD), RMSProp, adam, adagrad, dan adadelta. Metrik evaluasi yang populer adalah akurasi, ingatan, dan skor F1.

Singkatnya, langkah ini ditujukan untuk menyetel bobot dan bias berdasarkan fungsi loss melalui iterasi berdasarkan *Optimizer* yang dievaluasi oleh metrik seperti akurasi.

Sesuaikan dengan Model

Setelah mendefinisikan dan mengkompilasi model, Anda perlu membuat prediksi dengan mengeksekusi model pada beberapa data. Di sini Anda perlu menentukan zaman; ini adalah jumlah iterasi untuk proses pelatihan yang harus dijalankan melalui kumpulan data dan ukuran batch, yang merupakan jumlah instance yang dievaluasi sebelum pembaruan bobot. Untuk masalah ini, program akan berjalan untuk sejumlah kecil epoch (10), dan di setiap epoch, program akan menyelesaikan 50(=50.000/1.000) iterasi di mana ukuran batch adalah 1.000 dan training data set memiliki 50.000 instance/gambar . Sekali lagi, tidak ada aturan yang sulit untuk memilih ukuran batch. Tapi itu tidak boleh terlalu kecil, dan itu harus jauh lebih kecil dari ukuran kumpulan data pelatihan untuk mengkonsumsi lebih sedikit memori.

```
#Make the model learn ( Fit the model)
model.fit(X_train, Y_train,batch_size=1000, nb_epoch=10,validation_data=(X_test, Y_test))
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/10
 1000/50000 [..............................] - ETA: 6s - loss: 2.3028 - acc: 0.1060
C:\ProgramData\Anaconda3\lib\site-packages\keras\models.py:848: UserWarning: The `nb_epoch` argument in `fit` has been renamed
`epochs`.
  warnings.warn('The `nb_epoch` argument in `fit` '
50000/50000 [==============================] - 6s - loss: 2.3030 - acc: 0.0974 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 2/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.1012 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 3/10
50000/50000 [==============================] - 7s - loss: 2.3028 - acc: 0.0972 - val_loss: 2.3026 - val_acc: 0.1000
Epoch 4/10
50000/50000 [==============================] - 7s - loss: 2.3028 - acc: 0.0997 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 5/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.0975 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 6/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.0986 - val_loss: 2.3028 - val_acc: 0.1000
Epoch 7/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.0995 - val_loss: 2.3028 - val_acc: 0.1000
Epoch 8/10
50000/50000 [==============================] - 7s - loss: 2.3028 - acc: 0.0983 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 9/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.0998 - val_loss: 2.3027 - val_acc: 0.1000
Epoch 10/10
50000/50000 [==============================] - 7s - loss: 2.3029 - acc: 0.0972 - val_loss: 2.3027 - val_acc: 0.1000
<keras.callbacks.History at 0x2870136eef0>
```

Evaluasi Model

Setelah melatih jaringan saraf pada *training data set*, Anda perlu mengevaluasi kinerja jaringan. Perhatikan, cara ini hanya akan memberi Anda gambaran tentang seberapa baik Anda telah memodelkan kumpulan data (misalnya, akurasi kereta), Anda tidak akan tahu seberapa baik kinerja algoritma pada data baru. Ini untuk penyederhanaan, tetapi idealnya, Anda dapat memisahkan data ke dalam set data latih dan uji untuk pelatihan dan evaluasi model Anda. Anda dapat mengevaluasi model Anda pada kumpulan data pelatihan menggunakan fungsi evaluation() pada model Anda dan meneruskannya ke input dan output yang sama yang digunakan untuk melatih model. Ini akan menghasilkan prediksi untuk setiap pasangan input dan output dan mengumpulkan skor, termasuk kerugian rata-rata dan metrik apa pun yang telah Anda konfigurasikan, seperti akurasi.

```
#Evaluate how the model does on the test set
score = model.evaluate(X_test, Y_test, verbose=0)
#Accuracy Score
print('Test accuracy:', score[1])
```

Prediksi

Setelah Anda membangun dan mengevaluasi model, Anda perlu memprediksi data yang tidak diketahui.

```
#Predict digit(0-9) for test Data
model.predict_classes(X_test)
```

```
 9888/10000 [============================>.] - ETA: 0s
```

```
array([3, 8, 8, ..., 3, 4, 7], dtype=int64)
```

Save dan Reload Model

Berikut adalah langkah terakhir:

```
#Saving the model
model.save('model.h5')
jsonModel = model.to_json()
model.save_weights('modelWeight.h5')
```

```
#Load weight of the saved model
modelWt = model.load_weights('modelWeight.h5')
```

Opsional: Ringkas Model

Sekarang mari kita lihat bagaimana cara meringkas model.

```
#Summary of the model
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 512)               1573376
_________________________________________________________________
activation_7 (Activation)    (None, 512)               0
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_8 (Dense)              (None, 120)               61560
_________________________________________________________________
activation_8 (Activation)    (None, 120)               0
_________________________________________________________________
dropout_6 (Dropout)          (None, 120)               0
_________________________________________________________________
dense_9 (Dense)              (None, 10)                1210
_________________________________________________________________
activation_9 (Activation)    (None, 10)                0
=================================================================
Total params: 1,636,146
Trainable params: 1,636,146
Non-trainable params: 0
_________________________________________________________________
```

2.2 LANGKAH TAMBAHAN UNTUK MENINGKATKAN MODEL KERAS

Berikut adalah beberapa langkah lagi untuk meningkatkan model Anda:

1. Terkadang, proses pembuatan model tidak selesai karena gradien yang menghilang atau meledak. Jika ini masalahnya, Anda harus melakukan hal berikut:
   ```
   from keras.callbacks import EarlyStopping
   early_stopping_ monitor = EarlyStopping(patience=2)
   model.fit(x_train, y_train, batch_size=1000, epochs=10,
   validation_data=(x_test, y_test),
   callbacks=[early_stopping_monitor])
   ```
2. Modelkan bentuk output.
   #Bentuk array n-dim (output model pada posisi saat ini)
   model.output_shape
3. Model representasi ringkasan.
   model.summary()
4. Modelkan konfigurasi.
   model.get_config()
5. Buat daftar semua tensor bobot dalam model.
   model.get_weights()

Berikut saya bagikan kode lengkap untuk model Keras. Bisakah Anda mencoba menjelaskannya?

```
# A TYPING DEEP LEARNING MODEL WITH KERAS
    import numpy as np
    from keras.models import Sequential
    from keras.layers import Dense

# Loading Data

    data = np.random.random((500,100))
    labels = np.random.randint(2,size=(500,1))


# Create model
        model = Sequential()
        model.add(Dense(12, input_dim=8, activation='relu'))
        model.add(Dense(8, activation='relu'))
        model.add(Dense(1, activation='sigmoid'))

# Compile model

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

# Fit the model

    model.fit(X[train], Y[train], epochs=150, batch_size=10, verbose=0)
# Evaluate the model

    scores = model.evaluate(X[test], Y[test], verbose=0)
   print("%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
  cvscores.append(scores[1] * 100) print("%.2f%% (+/- %.2f%%)" %
(numpy.mean(cvscores), numpy.std(cvscores)))

# Predict
    predictions = model.predict(data)
```

2.3 KERAS DENGAN TENSORFLOW

Keras menyediakan jaringan saraf tingkat tinggi dengan memanfaatkan *Library Deep learning* yang kuat dan jelas di atas TensorFlow/Theano. Keras adalah tambahan yang bagus untuk TensorFlow karena layer dan modelnya kompatibel dengan tensor TensorFlow murni. Selain itu, ini dapat digunakan bersama library TensorFlow lainnya. Berikut adalah langkah-langkah yang terlibat dalam menggunakan Keras untuk TensorFlow:

1. Mulailah dengan membuat sesi TensorFlow dan mendaftarkannya ke Keras. Ini berarti Keras akan menggunakan sesi yang Anda daftarkan untuk menginisialisasi semua variabel yang dibuatnya secara internal.

```
============================================================
import TensorFlow as tf
sess = tf.Session()
from keras import backend as K
K.set_session(sess)
============================================================
```

2. Modul keras seperti model, layer, dan aktivasi digunakan untuk membangun model. Mesin Keras secara otomatis mengonversi modul ini menjadi skrip yang setara dengan TensorFlow.
3. Selain TensorFlow, Theano dan CNTK dapat digunakan sebagai backend Keras.
4. Bagian belakang TensorFlow memiliki konvensi untuk membuat bentuk input (ke layer pertama jaringan Anda) dalam urutan kedalaman, tinggi, lebar, di mana kedalaman dapat berarti jumlah saluran.
5. Anda perlu mengkonfigurasi file keras.json dengan benar agar menggunakan back end TensorFlow. Seharusnya terlihat seperti ini:

```
{
    "backend": "theano",
    "epsilon": 1e-07,
    "image_data_format": "channels_first",
    "floatx": "float32"
}
```

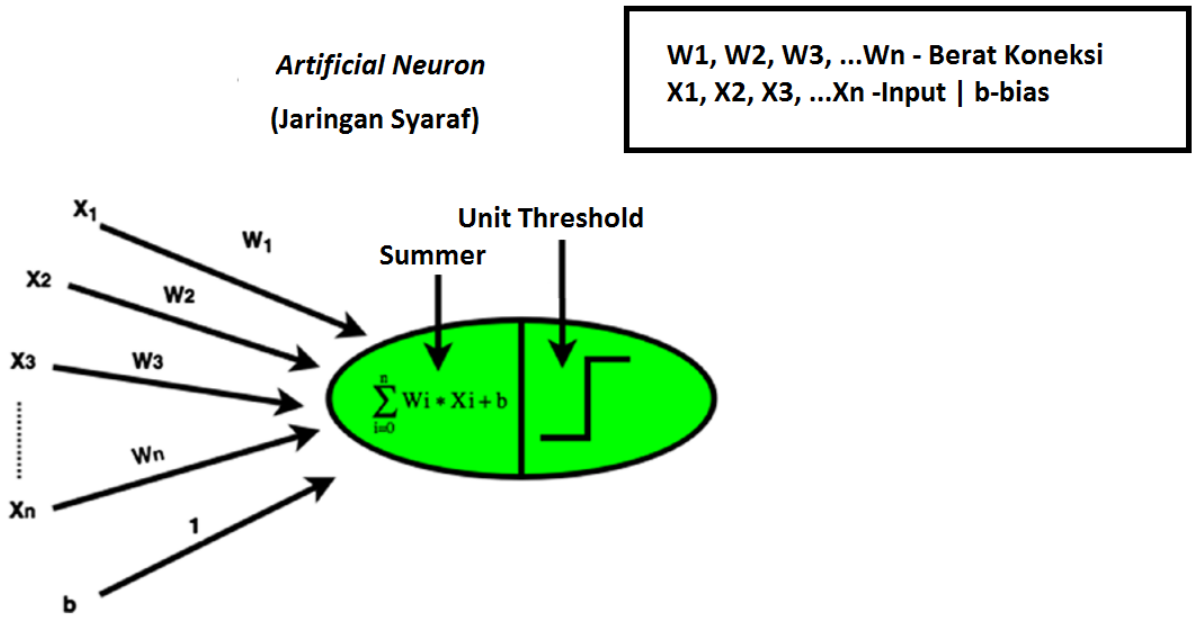
Di bab berikutnya, Anda akan mempelajari cara memanfaatkan Keras untuk bekerja di CNN, RNN, LSTM, dan aktivitas *Deep learning* lainnya.

BAB 3
MULTILAYER PERCEPTRON

Sebelum Anda mulai belajar tentang multilayer perceptron, Anda perlu mendapatkan gambaran besar tentang *artificial neural networks* (ANN)/jaringan saraf tiruan terlebih dulu. Mari kita mulai.

3.1 JARINGAN SYARAF BUATAN / ARTIFICIAL NEURAL NETWORK (ANN)

Artificial neural network (ANN) yang dalam bahasa Indonesia dikenal dengan Jaringan syaraf tiruan merupakan jaringan komputasi (sistem node dan interkoneksi antar node) yang terinspirasi oleh jaringan saraf biologis, yang merupakan jaringan kompleks neuron di otak manusia (lihat Gambar 3.1). Node yang dibuat di ANN seharusnya diprogram untuk berperilaku seperti neuron yang sebenarnya, dan karenanya mereka adalah neuron buatan. Gambar 3.1 menunjukkan jaringan node (neuron buatan) yang membentuk jaringan saraf tiruan.

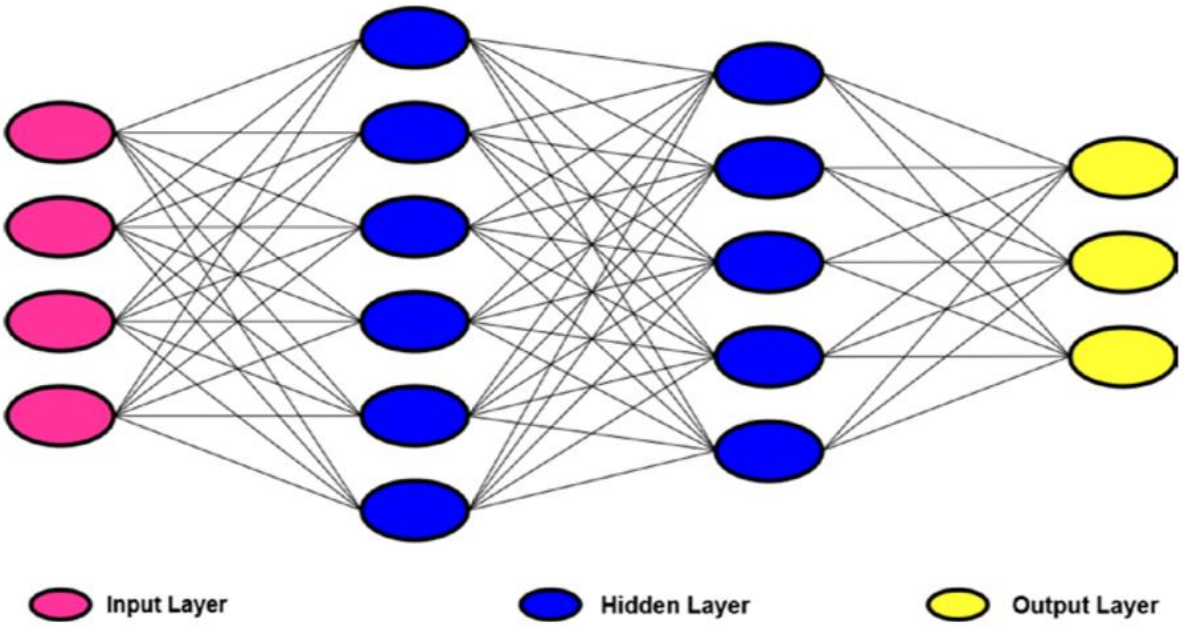


Gambar 3-1 Jaringan Saraf Tiruan

Jumlah layer dan jumlah *neuron/node* per layer dapat menjadi komponen struktural utama dari jaringan syaraf tiruan. Awalnya, bobot (mewakili interkoneksi) dan bias tidak cukup baik untuk membuat keputusan (klasifikasi, dll). Ini seperti otak bayi yang tidak memiliki pengalaman sebelumnya. Seorang bayi belajar dari pengalaman sehingga menjadi pembuat keputusan (classifier) yang baik. Pengalaman/data (berlabel) membantu jaringan saraf otak menyesuaikan bobot dan bias (saraf). Jaringan saraf tiruan melewati proses yang sama. Bobot disetel per iterasi untuk membuat pengklasifikasi yang baik. Karena menyetel dan dengan demikian mendapatkan bobot yang benar dengan tangan untuk ribuan neuron sangat memakan waktu, Anda menggunakan algoritme untuk melakukan tugas ini.
Proses penyetelan bobot itu disebut pembelajaran atau pelatihan. Hal ini sama dengan apa yang dilakukan manusia sehari-hari. Kami mencoba mengaktifkan komputer untuk bekerja seperti manusia.

Mari kita mulai menjelajahi model ANN yang paling sederhana. Jaringan saraf tipikal berisi sejumlah besar neuron buatan yang disebut unit yang disusun dalam serangkaian layer yang berbeda: layer input, hidden layer/layer tersembunyi, dan layer output (Gambar 3-2).

Jaringan saraf terhubung, yang berarti setiap neuron di layer tersembunyi terhubung sepenuhnya ke setiap neuron di layer input sebelumnya dan ke layer output berikutnya. Jaringan saraf belajar dengan menyesuaikan bobot dan bias di setiap layer secara iteratif untuk mendapatkan hasil yang optimal.



Gambar 3-2 Jaringan syaraf

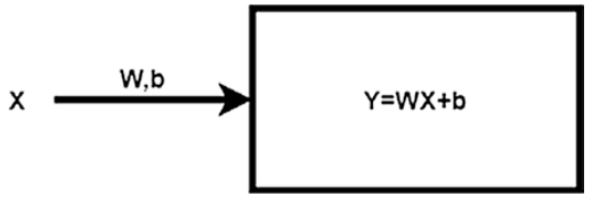
Perceptron Single-layer

Perceptron single-layer adalah *classifier biner linier* sederhana. Membutuhkan input dan bobot terkait dan menggabungkannya untuk menghasilkan output yang digunakan untuk klasifikasi. Tidak memiliki layer tersembunyi. Regresi logistik adalah perceptron single-layer.

3.2 *Multilayer Perceptron*

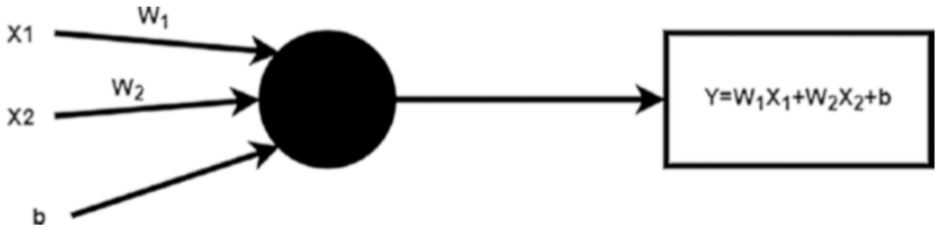
Multilayer perceptron (MLP) adalah contoh sederhana dari jaringan saraf tiruan umpan balik. Sebuah MLP terdiri dari setidaknya satu layer tersembunyi dari node selain layer input dan layer output. Setiap node dari layer selain layer input disebut neuron yang menggunakan fungsi aktivasi nonlinier seperti sigmoid atau ReLU. MLP menggunakan teknik pembelajaran terawasi yang disebut *backpropagation* untuk pelatihan, sambil meminimalkan fungsi loss seperti entropi silang. Ini menggunakan *Optimizer* untuk parameter penyetelan (berat dan bias). Beberapa layer dan aktivasi nonlinier membedakan MLP dari perceptron linier.

Multilayer Perceptron adalah bentuk dasar dari jaringan saraf yang dalam. Sebelum Anda mempelajari MLP, mari kita lihat model linier dan model logistik. Anda dapat menghargai perbedaan halus antara model linier, logistik, dan MLP dalam hal kompleksitas. Gambar 3-3 menunjukkan model linier dengan satu input (X) dan satu output (Y).



Gambar 3-3 Vektor input tunggal

Model input tunggal memiliki vektor X dengan bobot W dan bias b. Outputnya, Y, adalah WX + b, yang merupakan model linier. Gambar 3-4 menunjukkan beberapa input (X1 dan X2) dan satu output (Y).

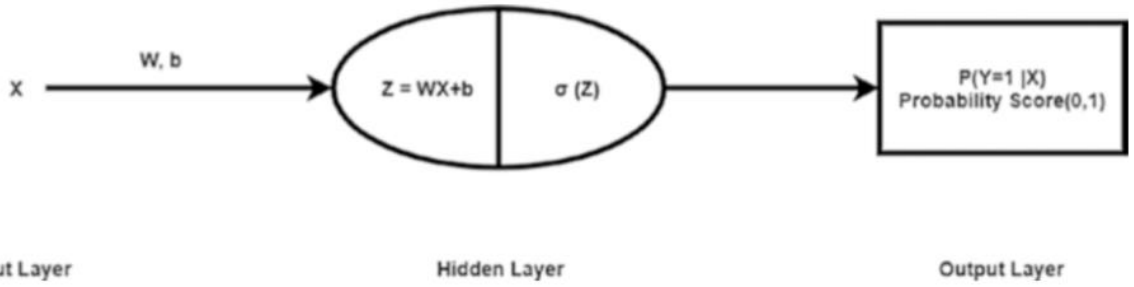


Gambar 3-4 model linier

Model linier ini memiliki dua fitur input: X1 dan X2 dengan bobot yang sesuai untuk setiap fitur input adalah W1, W2, dan bias b. Outputnya, Y, adalah W1X1 + W2X2 + b.

3.3 MODEL REGRESI LOGISTIK

Gambar 3-5 menunjukkan algoritma pembelajaran yang Anda gunakan saat label output Y adalah 0 atau 1 untuk masalah klasifikasi biner. Diberikan vektor fitur input X, Anda menginginkan probabilitas bahwa Y = 1 diberikan fitur input X. Ini juga disebut sebagai jaringan saraf dangkal atau jaringan saraf single-layer (tanpa layer tersembunyi; hanya dan layer output). Layer output, Y, adalah (Z), di mana Z adalah WX + b dan adalah fungsi sigmoid.



Gambar 3-5 Satu input (X) dan satu output (Y)

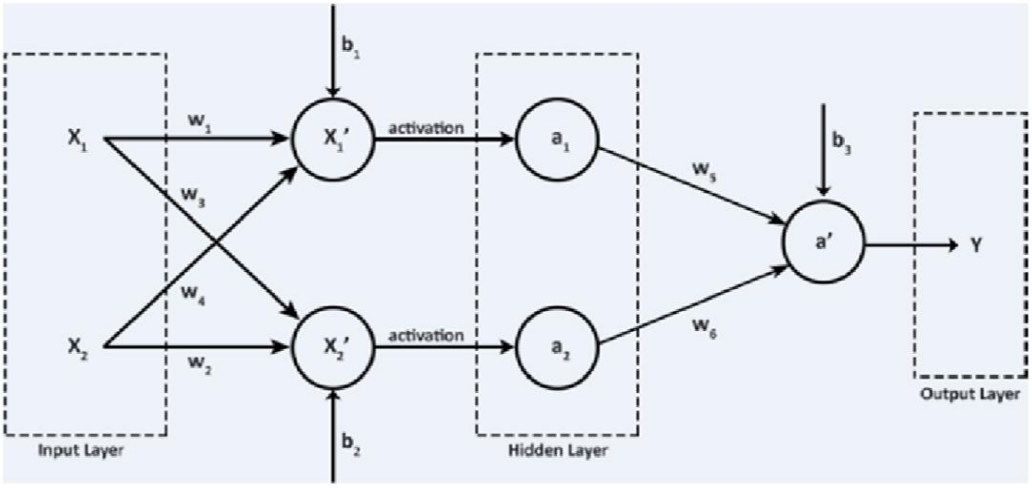
Gambar 3-6 menunjukkan algoritma pembelajaran yang Anda gunakan saat label output Y adalah 0 atau 1 untuk masalah klasifikasi biner.

Gambar 3-6 Beberapa input (X1 dan X1) dan satu output (Y)

Diberikan vektor fitur input X1 dan X2, Anda menginginkan probabilitas bahwa Y = 1 diberikan fitur input. Ini juga disebut perceptron. Layer output, Y, adalah σ(Z), di mana Z adalah WX + b.

$$\begin{bmatrix} X1 \\ X2 \end{bmatrix} \rightarrow \begin{bmatrix} W1 & W2 \\ W3 & W4 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \end{bmatrix} \rightarrow \sigma\left(\begin{bmatrix} W1*X1+W2*X2+b1 \\ W3*X1+W4*X2+b2 \end{bmatrix} \right)$$

Gambar 3-7 menunjukkan jaringan saraf dua layer, dengan layer tersembunyi dan layer output. Pertimbangkan bahwa Anda memiliki dua vektor fitur input X1 dan X2 yang terhubung ke dua neuron, X1' dan X2.' Parameter (bobot) yang terkait dari layer input ke layer tersembunyi adalah w1, w2, w3, w4, b1, b2.

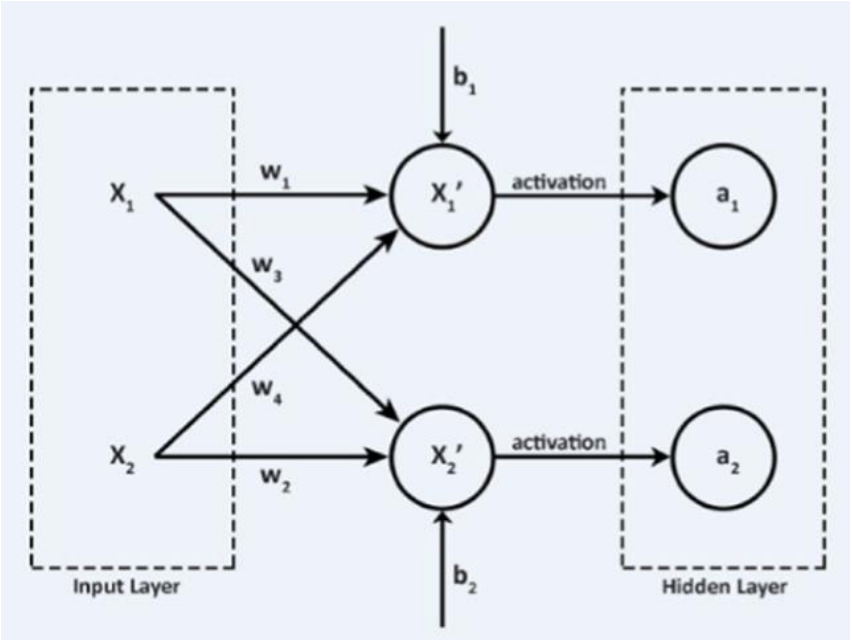


Gambar 3-7 Jaringan saraf dua layer

X1' dan X2' menghitung kombinasi linier (Gambar 3-8).

$$\begin{bmatrix} X1' \\ X2' \end{bmatrix} = \begin{bmatrix} w1 & w2 \\ w3 & w4 \end{bmatrix} \begin{bmatrix} X1 \\ X2 \end{bmatrix} + \begin{bmatrix} b1 \\ b2 \end{bmatrix}$$

(2×1)(2×2)(2×1)(2×1) adalah dimensi dari layer input dan layer tersembunyi.

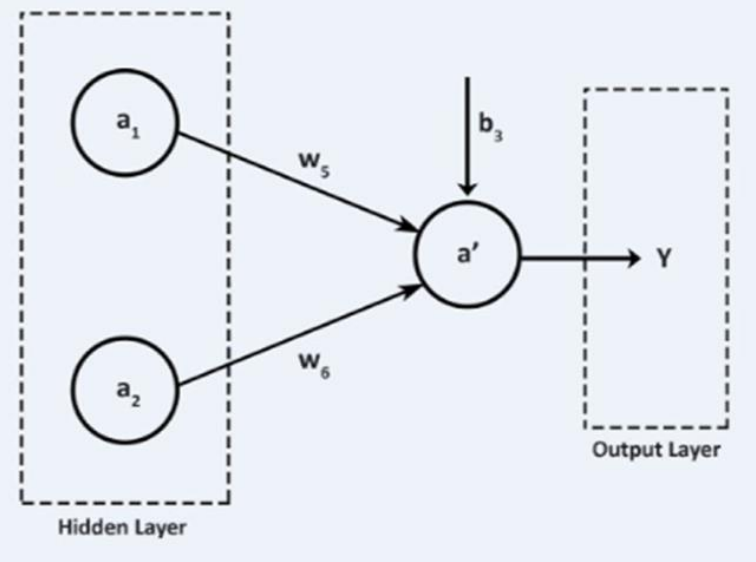


Gambar 3-8 Komputasi dalam jaringan saraf

Input linier X1’ dan X2’ melewati unit aktivasi a1 dan a2 di layer tersembunyi. a1 adalah σ(X1’) dan a2 adalah σ(X2’), sehingga Anda juga dapat menulis persamaan sebagai berikut:

$$\begin{bmatrix} a1 \\ a2 \end{bmatrix} = \sigma \begin{bmatrix} X1' \\ X2' \end{bmatrix}$$

Nilai maju menyebar dari layer tersembunyi ke layer output. Input a1 dan a2 serta parameter w5, w6, dan b3 melewati layer output a’ (Gambar 3-9).



Gambar 3-9 Propagasi maju

$a' = [w5 \;\; w6] \begin{bmatrix} a1 \\ a2 \end{bmatrix} + [b3]$menciptakan kombinasi linier (w5*a1 + w6*a2) + b3, yang akan melewati fungsi sigmoid nonlinier ke layer output akhir , Y

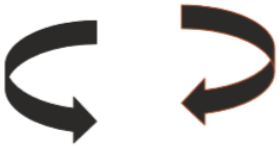
$$y = \sigma(a')$$

Misalkan struktur model awal dalam satu dimensi adalah Y = w*X + b, dengan parameter w dan b adalah bobot dan bias.

Pertimbangkan fungsi loss L(w, b) = 0,9 untuk nilai awal parameter w = 1 dan b = 1. Anda mendapatkan output ini: y = 1*X+1 & L(w ,b) = 0,9. Tujuannya adalah untuk meminimalkan kerugian dengan menyesuaikan parameter w dan b. Error tersebut akan dipropagasi kembali dari output layer ke *hidden* layer ke input layer untuk menyesuaikan parameter melalui *learning rate* dan *optimizer*. Terakhir, kami ingin membangun model (*regressor*) yang dapat menjelaskan Y dalam bentuk X. Untuk memulai proses membangun model, kami menginisialisasi bobot dan bias. Untuk kenyamanan, w = 1, b = 1 (Nilai awal), (*Optimizer*) penurunan gradien stokastik dengan kecepatan pembelajaran (α = 0,01). Berikut adalah langkah 1: Y = 1 * X + 1.

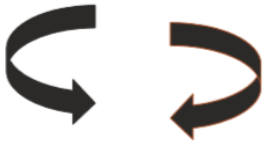
1.20 0.35

Parameter disesuaikan dengan w = 1,20 dan b = 0,35. Berikut langkah 2: Y1 = 1,20*X + 0,35.



1.24 0.31

Parameter disesuaikan dengan w = 1,24 dan b = 0,31. Berikut langkah 3: Y1 = 1,24*X + 0,31.



1.25 0.30

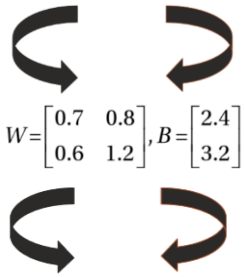
Setelah beberapa iterasi, bobot dan bias menjadi stabil. Seperti yang Anda lihat, perubahan awal tinggi saat menyetel. Setelah beberapa kali iterasi, perubahannya tidak signifikan. L(w, b) diminimalkan untuk w = 1,26 dan b = 0,29; maka, model akhir menjadi sebagai berikut:

$$Y = 1{,}26 * X + 0{,}29$$

Demikian pula, dalam dua dimensi, Anda dapat mempertimbangkan parameter, matriks bobot dan vektor bias. Mari kita asumsikan bahwa matriks bobot awal dan vektor bias sebagai $W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} dan\ B = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Anda mengulangi dan menyebarkan kembali kesalahan untuk menyesuaikan w dan b.

$Y = W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} * [X] + \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ adalah model awal. Matriks bobot (2x2) dan matriks bias (2x1) disetel pada setiap iterasi. Jadi, kita bisa melihat perubahan matriks bobot dan bias. Inilah langkah 1:



$$W = \begin{bmatrix} 0.7 & 0.8 \\ 0.6 & 1.2 \end{bmatrix}, B = \begin{bmatrix} 2.4 \\ 3.2 \end{bmatrix}$$

Inilah langkah 2:

$$\begin{bmatrix} 0.7 & 0.8 \\ 0.6 & 1.2 \end{bmatrix} \begin{bmatrix} 2.4 \\ 3.2 \end{bmatrix}$$

$$W = \begin{bmatrix} 0.6 & 0.7 \\ 0.4 & 1.3 \end{bmatrix}, B = \begin{bmatrix} 2.8 \\ 3.8 \end{bmatrix}$$

Inilah langkah 3:

$$\begin{bmatrix} 0.6 & 0.7 \\ 0.4 & 1.3 \end{bmatrix} \begin{bmatrix} 2.8 \\ 3.8 \end{bmatrix}$$

Anda dapat melihat perubahan dalam matriks bobot (2x2) dan matriks bias (2x1) dalam iterasi.

$$W = \begin{bmatrix} 0.5 & 0.6 \\ 0.3 & 1.3 \end{bmatrix}, B = \begin{bmatrix} 2.9 \\ 4.0 \end{bmatrix}$$

Model akhir setelah w dan b disesuaikan adalah sebagai berikut:

$$\mathrm{Y} = \begin{bmatrix} 0.4 & 0.5 \\ 0.2 & 1.3 \end{bmatrix} * [X] + \begin{bmatrix} 3.0 \\ 4.0 \end{bmatrix}$$

Dalam bab ini, Anda mempelajari bagaimana bobot dan bias disetel di setiap iterasi sambil menjaga tujuan meminimalkan fungsi loss. Itu dilakukan dengan bantuan Optimizer seperti penurunan gradien stokastik. Dalam bab ini, kita telah memahami ANN dan MLP sebagai *model deep learning* dasar. Di sini, kita dapat melihat MLP sebagai perkembangan alami dari regresi linier dan logistik. Kita telah melihat bagaimana bobot dan bias disetel pada setiap iterasi yang terjadi pada *backpropagation.* Tanpa masuk ke detail *backpropagation,* kita telah melihat aksi/hasil *backpropagation*. Dalam dua bab berikutnya, kita dapat mempelajari cara membangun model MLP di TensorFlow dan dengan keras.

BAB 4
REGRESI KE MLP DI TENSORFLOW

Orang-orang telah menggunakan regresi dan pengklasifikasi untuk waktu yang lama. Sekarang saatnya untuk beralih ke topik jaringan saraf. *Multilayered perceptron* (MLP) adalah model jaringan saraf sederhana di mana Anda dapat menambahkan satu atau lebih layer tersembunyi antara layer input dan output. Dalam bab ini, Anda akan melihat bagaimana TensorFlow dapat membantu Anda membangun model. Anda akan mulai dengan model paling dasar, yaitu model linier. Model logistik dan MLP juga dibahas dalam bab ini.

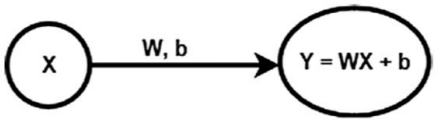
4.1 LANGKAH-LANGKAH TENSORFLOW UNTUK MEMBUAT MODEL

Di bagian ini, saya akan membahas langkah-langkah untuk membangun model di TensorFlow. Saya akan memandu Anda melalui langkah-langkah mudah dengan contoh kode di seluruh bab ini:

1. Muat data.
2. Pisahkan data ke dalam kereta dan uji.
3. Normalisasi jika diperlukan.
4. Inisialisasi placeholder yang akan berisi prediktor dan target.
5. Buat variabel (bobot dan bias) yang akan disetel.
6. Deklarasikan operasi model.
7. Deklarasikan fungsi loss dan optimizer.
8. Inisialisasi variabel dan sesi.
9. Sesuaikan model dengan menggunakan loop pelatihan.
10. Periksa dan tampilkan hasilnya dengan data uji.

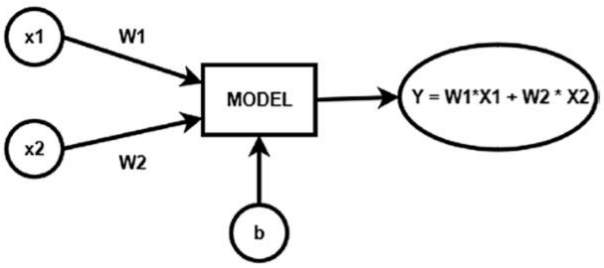
4.2 REGRESI LINIER DI TENSORFLOW

Pertama, Anda perlu memahami kode untuk regresi linier di TensorFlow. Gambar 4-1 menunjukkan model linier dasar. Seperti yang ditunjukkan pada Gambar 4-1, bobot (W) dan bias (b) harus disetel untuk mendapatkan nilai bobot dan bias yang tepat. Jadi, bobot (W) dan bias (b) adalah variabel dalam kode TensorFlow; Anda akan menyetel/memodifikasinya di setiap iterasi hingga Anda mendapatkan nilai yang stabil (benar).



Gambar 4-1 Model linier dasar

Anda perlu membuat *placeholder* untuk X. *Placeholder* memiliki bentuk tertentu dan berisi jenis tertentu. Jika Anda memiliki lebih dari satu fitur, Anda akan memiliki model kerja yang mirip dengan Gambar 4-2.



Gambar 4-2 Model linier dengan banyak input

Dalam kode berikut, Anda akan menggunakan kumpulan data Iris dari Seaborn, yang memiliki lima atribut. Anda akan mempertimbangkan panjang sepal sebagai input dan panjang petal sebagai nilai output. Tujuan utama dari model regresi ini adalah untuk memprediksi panjang kelopak ketika Anda diberi nilai panjang sepal. X adalah panjang sepal, dan Y adalah panjang petal. Regresi linier menggunakan TensorFlow pada data Iris

```
############ Linear Regression: TensorFlow Way #################
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn import datasets
import numpy as np
from sklearn.cross_validation import train_test_split
from matplotlib import pyplot
```

```
# 1. Load the data
# iris.data = [(Sepal Length, Sepal Width, Petal Length, Petal Width)]
iris = datasets.load_iris()
# X is Sepal.Length and Y is Petal Length
predictors_vals = np.array([predictors[0] for predictors in iris.data])
target_vals = np.array([predictors[2] for predictors in iris.data])
```

```
# 2.Split Data into train and test 80%-20%
x_trn, x_tst, y_trn, y_tst = train_test_split(predictors_vals, target_vals, test_size=0.2, random_state=12)
#training_idx = np.random.randint(x_vals.shape[0], size=80)
#training, test = x_vals[training_idx,:], x_vals[-training_idx,:]
```

```
# 3. Normalize if needed
# 4. Initialize placeholders that will contain predictors and target
predictor = tf.placeholder(shape=[None, 1], dtype=tf.float32)
target = tf.placeholder(shape=[None, 1], dtype=tf.float32)
```

```
#5. Create variables (Weight and Bias) that will be tuned up
A = tf.Variable(tf.zeros(shape=[1,1]))
b = tf.Variable(tf.ones(shape=[1,1]))
```

```
# 6. Declare model operations: Ax+b
model_output = tf.add(tf.matmul(predictor, A), b)
```

```
#7. Declare loss function and optimizer
#Declare loss function (L1 loss)
loss = tf.reduce_mean(tf.abs(target - model_output))
# Declare optimizer
my_opt = tf.train.GradientDescentOptimizer(0.01)
#my_opt = tftrain.AdamOptimizer(0.01)
train_step = my_opt.minimize(loss)
```

```
#8. Initialize variables and session
sess = tf.Session()
init = tf.global_variables_initializer()
sess.run(init)
```

```
#9. Fit Model by using Training Loops
# Training Loop
lossArray = []
batch_size = 40
for i in range(200):
    rand_rows = np.random.randint(0, len(x_trn)-1, size=batch_size)
    batchX = np.transpose([x_trn[rand_rows]])
    batchY = np.transpose([y_trn[rand_rows]])
    sess.run(train_step, feed_dict={predictor: batchX, target: batchY})
    batchLoss = sess.run(loss, feed_dict={predictor: batchX, target: batchY})
    lossArray.append(batchLoss)
    if (i+1)%50==0:
        print('Step Number' + str(i+1) + ' A = ' + str(sess.run(A)) + ' b = ' + str(sess.run(b)))
        print('L1 Loss = ' + str(batchLoss))

[slope] = sess.run(A)
[y_intercept] = sess.run(b)
```

```
# 10. Check  and Display the result on test data
lossArray = []
batch_size = 30
for i in range(100):
    rand_rows = np.random.randint(0, len(x_tst)-1, size=batch_size)
    batchX = np.transpose([x_tst[rand_rows]])
    batchY = np.transpose([y_tst[rand_rows]])
    sess.run(train_step, feed_dict={predictor: batchX, target: batchY})
    batchLoss = sess.run(loss, feed_dict={predictor: batchX, target: batchY})
    lossArray.append(batchLoss)
    if (i+1)%20==0:
        print('Step Number: ' + str(i+1) + ' A = ' + str(sess.run(A)) + ' b = ' + str(sess.run(b)))
        print('L1 Loss = ' + str(batchLoss))
# Get the optimal coefficients
[slope] = sess.run(A)
[y_intercept] = sess.run(b)
```

```
# Original Data and Plot
plt.plot(x_tst, y_tst, 'o', label='Actual Data')
test_fit = []
for i in x_tst:
  test_fit.append(slope*i+y_intercept)
# predicted values and Plot
plt.plot(x_tst, test_fit, 'r-', label='Predicted line', linewidth=3)
plt.legend(loc='lower right')
plt.title('Petal Length vs Sepal Length')
plt.ylabel('Petal Length')
plt.xlabel('Sepal Length')
plt.show()
```

```
# Plot loss over time
plt.plot(lossArray, 'r-')
plt.title('L1 Loss per loop')
plt.xlabel('Loop')
plt.ylabel('L1 Loss')
plt.show()
```

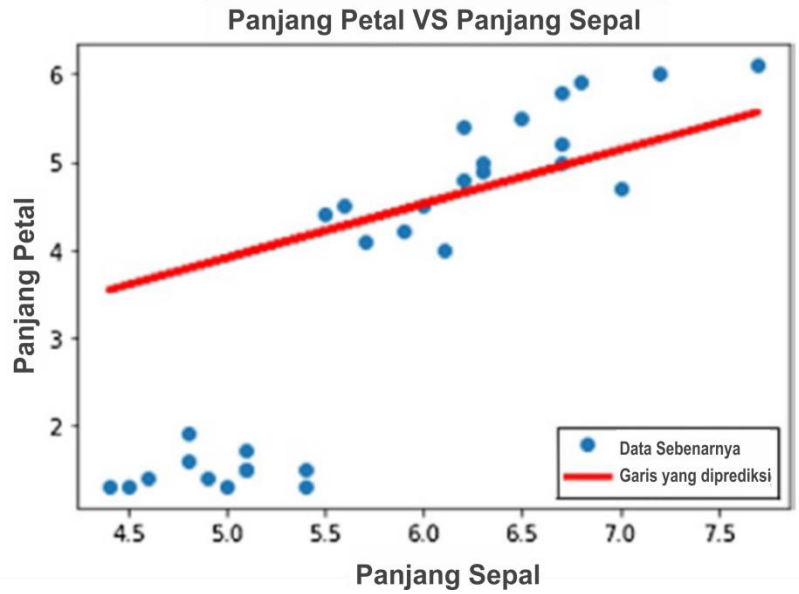
Jika Anda menjalankan kode, Anda akan melihat output yang ditunjukkan pada Gambar 4-3.

```
Step Number50 A = [[ 0.57060003]] b = [[ 1.05275011]]
L1 Loss = 0.965698
Step Number100 A = [[ 0.56647497]] b = [[ 1.00924981]]
L1 Loss = 1.124
Step Number150 A = [[ 0.56645012]] b = [[ 0.96424991]]
L1 Loss = 1.18043
Step Number200 A = [[ 0.58122498]] b = [[ 0.92174983]]
L1 Loss = 1.20376
Step Number: 20 A = [[ 0.58945829]] b = [[ 0.90308326]]
L1 Loss = 1.18207
Step Number: 40 A = [[ 0.62599164]] b = [[ 0.88975]]
L1 Loss = 0.826957
Step Number: 60 A = [[ 0.63695836]] b = [[ 0.87108338]]
L1 Loss = 0.838114
Step Number: 80 A = [[ 0.60072505]] b = [[ 0.8450833]]
L1 Loss = 1.52654
Step Number: 100 A = [[ 0.6150251]] b = [[ 0.8290832]]
L1 Loss = 1.25477
```

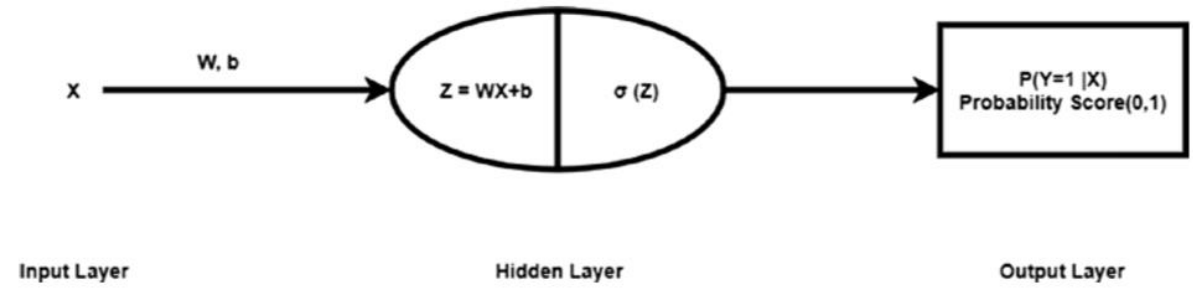
Gambar 4-3 Bobot, bias, dan kerugian di setiap langkah

4.3 MODEL REGRESI LOGISTIK

Untuk klasifikasi, pendekatan yang paling sederhana adalah regresi logistik. Di bagian ini, Anda akan mempelajari cara melakukan regresi logistik di TensorFlow. Di sini Anda membuat bobot dan bias sebagai variabel sehingga ada ruang lingkup untuk menyetel/mengubahnya per iterasi. *Placeholder* dibuat untuk memuat X. Anda perlu membuat *placeholder* untuk X. *Placeholder* memiliki bentuk tertentu dan berisi jenis tertentu, seperti yang ditunjukkan pada Gambar 4-5.



Gambar 4-4. Panjang kelopak versus panjang sepal



Gambar 4-5 Bagan model regresi logistik

Dalam kode berikut, Anda akan menggunakan kumpulan data Iris, yang memiliki lima atribut. Yang kelima adalah kelas sasaran. Anda akan mempertimbangkan panjang sepal dan lebar sepal sebagai atribut prediktor dan spesies bunga sebagai nilai target. Tujuan utama dari model regresi logistik ini adalah untuk memprediksi jenis spesies ketika Anda diberi nilai panjang sepal dan lebar sepal. Buat file Python dan impor semua Library yang diperlukan.

```
######Logistic Regression in Tensorflow##########################
import numpy as np
import tensorflow as tf
from sklearn import datasets
import pandas as pd
from sklearn.cross_validation import train_test_split
from matplotlib import pyplot
# 1. Loading Data
iris = datasets.load_iris()
# Predictors Two columns : Sepal Length and Sepal Width
predictors_vals = np.array([predictor[0:2] for predictor in iris.data])
# For setosa Species, target is 0.
target_vals = np.array([1. if predictor==0 else 0. for predictor in iris.target])
```

```
# 2. Split data into train/test = 75%/25%
predictors_vals_train, predictors_vals_test, target_vals_train, target_vals_test = train_test_split(predictors_vals, target_val
s,
                                                    train_size=0.75,
                                                    random_state=0)
```

```
# 3. Normalize if needed
#4.Initialize placeholders that will contain predictors and target
x_data = tf.placeholder(shape=[None, 2], dtype=tf.float32)
y_target = tf.placeholder(shape=[None, 1], dtype=tf.float32)
```

```
#5. Create variables (Weight and Bias) that will be tuned up
W = tf.Variable(tf.ones(shape=[2,1]))
b = tf.Variable(tf.ones(shape=[1,1]))
```

```
# 6. Declare model operations : y = xW +b
model = tf.add(tf.matmul(x_data, W), b)
```

```
#7. Declare loss function and Optimizer
loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits(logits=model, labels=y_target))
my_opt = tf.train.AdamOptimizer(0.02) #learning rate =0.02
train_step = my_opt.minimize(loss)
```

```
#8. Initialize variables and session
init = tf.global_variables_initializer()
sess=tf.Session()
sess.run(init)
```

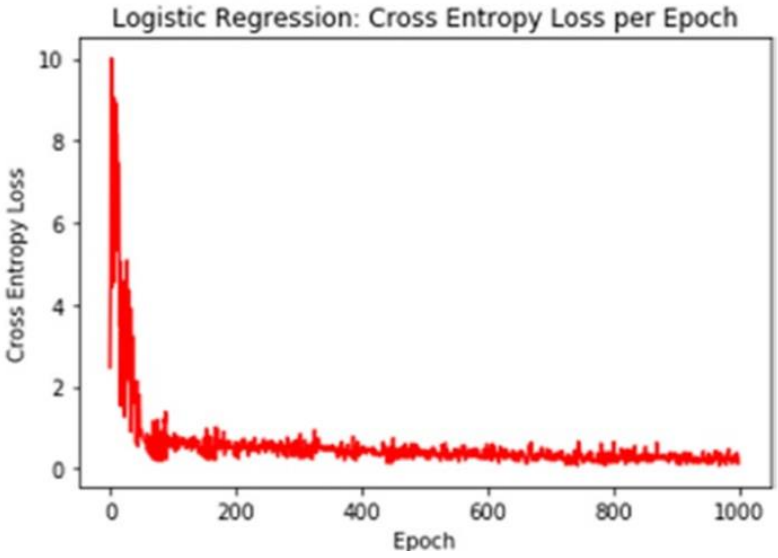
```
#9. Actual Prediction:
prediction = tf.round(tf.sigmoid(model))
predictions_correct = tf.cast(tf.equal(prediction, y_target), tf.float32)
accuracy = tf.reduce_mean(predictions_correct)
```

```
#10. Training loop
lossArray = []
trainAccuracy = []
testAccuracy = []
for i in range(1000):
    #Random instances for Batch size
    batch_size = 4 #Declare batch size
    batchIndex = np.random.choice(len(predictors_vals_train), size=batch_size)
    batchX = predictors_vals_train[batchIndex]
    batchY = np.transpose([target_vals_train[batchIndex]])
    # Tuning weight and bias while minimizing loss function through optimizer
    sess.run(train_step, feed_dict={x_data: batchX, y_target: batchY})
    #loss function per epoch/generation
    batchLoss = sess.run(loss, feed_dict={x_data: batchX, y_target: batchY})
    lossArray.append(batchLoss) # adding it to loss_vec
    # accuracy for each epoch for train
    batchAccuracyTrain = sess.run(accuracy, feed_dict={x_data: predictors_vals_train, y_target: np.transpose([target_vals_trai
n])})
    trainAccuracy.append(batchAccuracyTrain) # adding it to train_acc
    # accuracy for each epoch for test
    batchAccuracyTest = sess.run(accuracy, feed_dict={x_data: predictors_vals_test, y_target: np.transpose([target_vals_test])})
    testAccuracy.append(batchAccuracyTest)
    # printing loss after 10 epochs/generations to avoid verbosity
    if (i+1)%50==0:
        print('Loss = ' + str(batchLoss)+ ' and Accuracy = '   + str(batchAccuracyTrain))
```

```
# 11. Check model performance
pyplot.plot(lossArray, 'r-' )
pyplot.title('Logistic Regression: Cross Entropy Loss per Epoch')
pyplot.xlabel('Epoch')
pyplot.ylabel('Cross Entropy Loss')
pyplot.show()
```

,

Jika Anda menjalankan kode sebelumnya, plot cross entropy loss pada setiap epoch akan terlihat seperti Gambar 4-6.



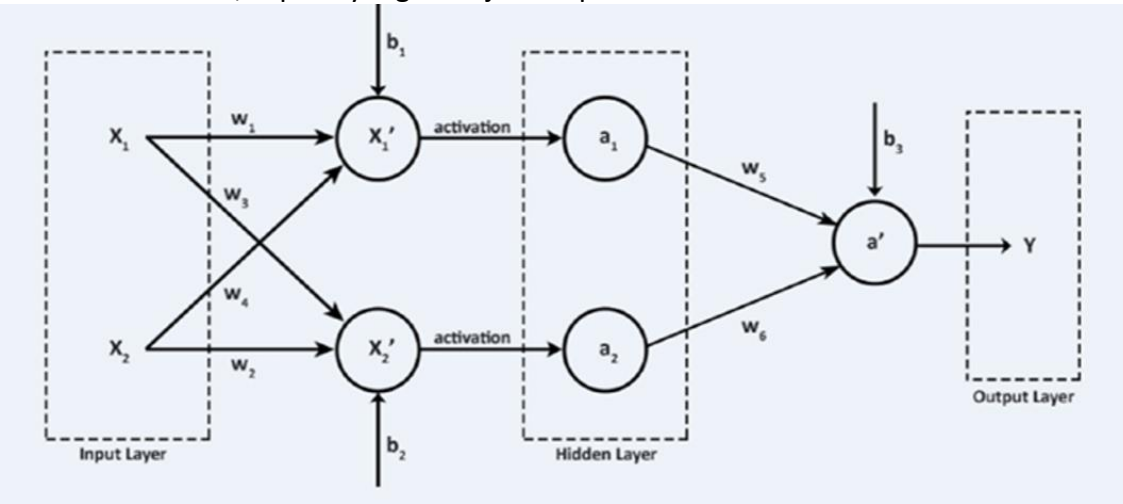
Gambar 4-6 plot cross entropy loss

Multilayer Perceptron di TensorFlow

Multilayer perceptron (MLP) adalah contoh sederhana dari jaringan saraf tiruan umpan balik. Sebuah MLP terdiri dari setidaknya satu layer tersembunyi dari node selain layer input dan layer output. Setiap node dari layer selain layer input disebut neuron yang menggunakan fungsi aktivasi nonlinier seperti sigmoid atau ReLU. MLP menggunakan teknik pembelajaran terawasi yang disebut backpropagation untuk pelatihan sambil meminimalkan fungsi loss seperti entropi silang dan menggunakan Optimizer untuk parameter penyetelan (bobot dan bias). Beberapa layer dan aktivasi non-linearnya membedakan MLP dari perceptron linier.

TensorFlow sangat cocok untuk membuat model MLP. Dalam MLP, Anda perlu menyetel bobot dan bias per iterasi. Ini berarti bobot dan bias terus berubah hingga menjadi stabil sambil meminimalkan fungsi loss. Jadi, Anda dapat membuat bobot dan bias sebagai

variabel di TensorFlow. Saya cenderung memberi mereka nilai awal (semua 0 atau semua 1 atau beberapa nilai terdistribusi normal secara acak). Placeholder harus memiliki jenis nilai dan bentuk tertentu, seperti yang ditunjukkan pada Gambar 4-7.



Gambar 4-7 Diagram alir untuk MLP

Impor semua Library yang diperlukan. Menerapkan MLP di TensorFlow.

```
# Implementing a one-hidden Layer Neural Network (MLP)

import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from sklearn import datasets
from sklearn.cross_validation import train_test_split
from matplotlib import pyplot
```

```
#2. Split data into train/test = 80%/20%

predictors_vals_train, predictors_vals_test, target_vals_train, target_vals_test= train_test_split(predictors_vals, target_val
s, test_size=0.2, random_state=12)
# 3. Normalize if needed
```

```
# 4.Initialize placeholders that will contain predictors and target
x_data = tf.placeholder(shape=[None, 3], dtype=tf.float32)
y_target = tf.placeholder(shape=[None, 1], dtype=tf.float32)
```

```
#5. Create variables (Weight and Bias) that will be tuned up
hidden_layer_nodes = 10
#For first layer
A1 = tf.Variable(tf.ones(shape=[3,hidden_layer_nodes])) # inputs -> hidden nodes
b1 = tf.Variable(tf.ones(shape=[hidden_layer_nodes]))   # one biases for each hidden node
#For second layer
A2 = tf.Variable(tf.ones(shape=[hidden_layer_nodes,1])) # hidden inputs -> 1 output
b2 = tf.Variable(tf.ones(shape=[1]))   # 1 bias for the output
```

```
# 6. Define Model Structure
hidden_output = tf.nn.relu(tf.add(tf.matmul(x_data, A1), b1))
final_output = tf.nn.relu(tf.add(tf.matmul(hidden_output, A2), b2))
```

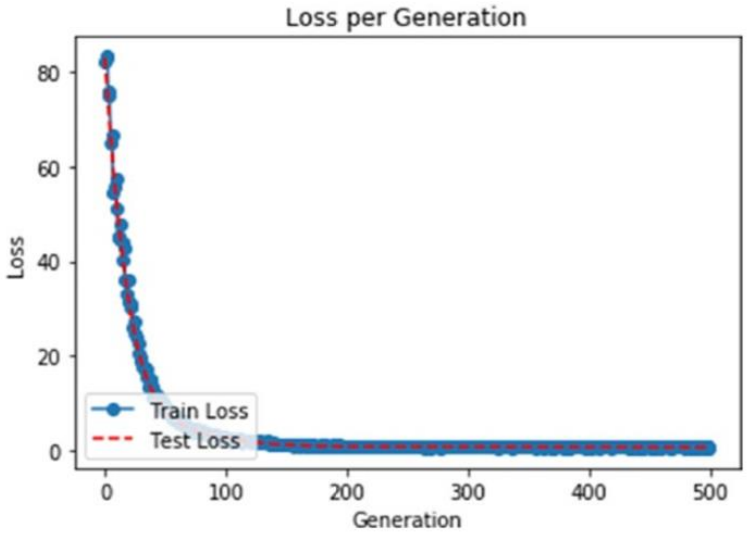
```
# 7. Declare loss function (MSE) and optimizer
loss = tf.reduce_mean(tf.square(y_target - final_output))
my_opt = tf.train.AdamOptimizer(0.02) # learning rate = 0.02
train_step = my_opt.minimize(loss)
```

```
# 8.Initialize variables and session
init = tf.global_variables_initializer()
# Create graph session
sess = tf.Session()
sess.run(init)
```

```
# 9. Training loop
lossArray = []
test_loss = []
batch_size =20
for i in range(500):
    batchIndex = np.random.choice(len(predictors_vals_train), size=batch_size)
    batchX = predictors_vals_train[batchIndex]
    batchY = np.transpose([target_vals_train[batchIndex]])
    sess.run(train_step, feed_dict={x_data: batchX, y_target: batchY})
    #
    batchLoss = sess.run(loss, feed_dict={x_data: batchX, y_target: batchY})
    lossArray.append(np.sqrt(batchLoss))

    test_temp_loss = sess.run(loss, feed_dict={x_data: predictors_vals_test, y_target: np.transpose([target_vals_test])})
    test_loss.append(np.sqrt(test_temp_loss))
    if (i+1)%50==0:
        print('Loss = ' + str(batchLoss))
```

```
# 10. Check model performance
#Plot loss(mean squared error) over time
pyplot.plot(lossArray, 'o-', label='Train Loss')
pyplot.plot(test_loss, 'r--', label='Test Loss')
pyplot.title('Loss per Generation')
pyplot.legend(loc='lower left')
pyplot.xlabel('Generation')
pyplot.ylabel('Loss')
pyplot.show()
```

Jika Anda akan menjalankan kode ini, Anda akan mendapatkan plot yang ditunjukkan pada Gambar 4-8.



Gambar 4-8 Hasil dari list program diatas

Dalam bab ini, saya membahas bagaimana Anda dapat membangun model linier, logistik, dan MLP di TensorFlow secara sistemik.

BAB 5
REGRESI KE MLP DI KERAS

Anda telah mengerjakan regresi sambil memecahkan aplikasi pembelajaran mesin. Regresi linier dan regresi nonlinier digunakan untuk memprediksi target numerik, sedangkan regresi logistik dan pengklasifikasi lainnya digunakan untuk memprediksi variabel target non-numerik. Dalam bab ini, saya akan membahas evolusi *Multilayer Perceptron*. Secara khusus, Anda akan membandingkan akurasi yang dihasilkan oleh model yang berbeda dengan dan tanpa menggunakan Keras.

5.1 MODEL LOG-LINEAR

Buat file Python baru dan impor paket berikut. Pastikan Anda telah menginstal Keras di sistem Anda.

```
####### Log-Linear Model ########################################
from sklearn.datasets import load_iris
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LogisticRegressionCV
from sklearn.linear_model import LinearRegression
import numpy as np
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense, Activation
```

Anda akan menggunakan kumpulan data Iris sebagai sumber data. Jadi, muat kumpulan data dari Seaborn.

```
# Load the iris dataset from seaborn.
iris = load_iris()
```

Kumpulan data Iris memiliki lima atribut. Anda akan menggunakan empat atribut pertama untuk memprediksi spesies, yang kelasnya ditentukan dalam atribut kelima dari kumpulan data.

```
# Use the first 4 variables to predict the species.
X, y = iris.data[:, :4], iris.target
```

Menggunakan fungsi scikit-learn, pisahkan kumpulan data pengujian dan pelatihan.

```
# Split both independent and dependent variables in half
# for cross-validation
train_X, test_X, train_y, test_y = train_test_split(X, y, train_size=0.5, random_state=0)
#print(type(train_X),len(train_y),len(test_X),len(test_y))
```

```
###################################
# scikit Learn for (Log) Linear Regression #
###################################
```

Gunakan fungsi model.fit untuk melatih model dengan kumpulan data pelatihan.

```
# Train a scikit-learn log-regression model
# lr =LogisticRegressionCV
# Train a scikit-learn linear-regression model
lr = LinearRegression()
lr.fit(train_X, train_y)
```

Saat model dilatih, Anda dapat memprediksi output dari set pengujian.

```
# Test the model. Print the accuracy on the test data
pred_y = lr.predict(test_X)
#print("Accuracy is {:.2f}".format(lr.score(test_X, test_y)))
```

5.2 JARINGAN SARAF KERAS UNTUK REGRESI LINIER

Sekarang, mari kita buat model jaringan saraf Keras untuk regresi linier.

```
# Build the keras model
model = Sequential()
# 4 features in the input layer (the four flower measurements)
# 16 hidden units
model.add(Dense(16, input_shape=(4,)))
model.add(Activation('sigmoid'))
# 3 classes in the ouput layer (corresponding to the 3 species)
model.add(Dense(3))
model.add(Activation('softmax'))
```

```
# Compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Gunakan fungsi model.fit untuk melatih model dengan kumpulan data pelatihan.

```
# Fit/Train the keras model
model.fit(train_X, train_y, verbose=1, batch_size=1, nb_epoch=100)
```

Saat model dilatih, Anda dapat memprediksi output dari set pengujian.

```
# Test the model. Print the accuracy on the test data
loss, accuracy = model.evaluate(test_X, test_y, verbose=0)
print("\nAccuracy is using keras prediction  {:.2f}".format(accuracy))
```

Cetak akurasi yang diperoleh kedua model.

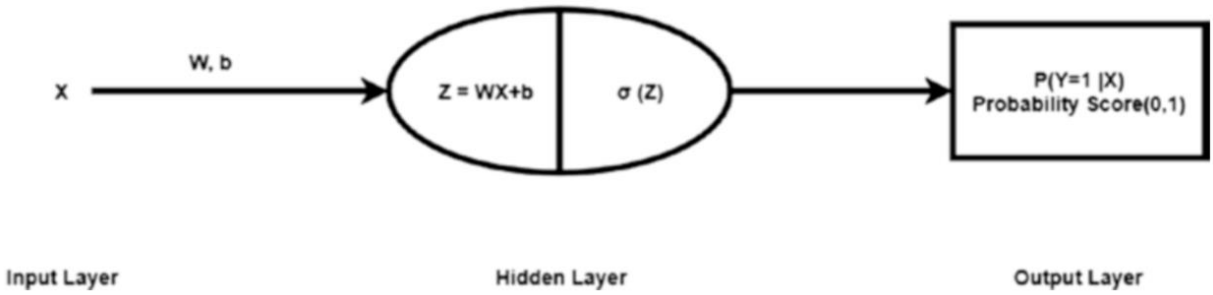
```
print("\nAccuracy is using keras prediction  {:.2f}".format(accuracy))

print("Accuracy is using regression  {:.2f}".format(lr.score(test_X, test_y)))
```

Jika Anda menjalankan kode, Anda akan melihat output berikut:

```
Using TensorFlow backend.

Epoch 1/100

75/75 [==============================] - 0s - loss: 1.2947 - acc: 0.4533

Epoch 2/100

75/75 [==============================] - 0s - loss: 1.0353 - acc: 0.6400

Epoch 3/100

75/75 [==============================] - 0s - loss: 0.8930 - acc: 0.6533

...                                 ...                                 ...

...                                 ...                                 ...

...                                 ...                                 ...

...                                 ...                                 ...

Epoch 99/100

75/75 [==============================] - 0s - loss: 0.1186 - acc: 0.9733

Epoch 100/100

75/75 [==============================] - 0s - loss: 0.1167 - acc: 0.9867


Accuracy is using keras prediction  0.99

Accuracy is using regression  0.89
```

5.3 REGRESI LOGISTIK

Di bagian ini, saya akan membagikan contoh regresi logistik sehingga Anda dapat membandingkan kode di scikit-learn dengan kode di Keras (lihat Gambar 5-1).



Gambar 5-1. Regresi logistik yang digunakan untuk klasifikasi

Buat file Python baru dan impor paket berikut. Pastikan Anda telah menginstal Keras di sistem Anda.

```
from sklearn.datasets import load_iris
import numpy as np
from sklearn.cross_validation import train_test_split
from sklearn.linear_model import LogisticRegressionCV
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.utils import np_utils
```

Anda akan menggunakan kumpulan data Iris sebagai sumber data. Jadi, muat kumpulan data dari scikit-learn.

```
# Load Data and Prepare data
iris = load_iris()
X, y = iris.data[:, :4], iris.target
```

Menggunakan fungsi scikit-learn, pisahkan kumpulan data pengujian dan pelatihan.

```
# Load Data and Prepare data
iris = load_iris()
X, y = iris.data[:, :4], iris.target
```

Scikit-Learn Untuk Regresi Logistik

Gunakan fungsi model.fit untuk melatih model dengan kumpulan data pelatihan. Setelah model dilatih, Anda dapat memprediksi output dari set tes.

```
################################
# scikit Learn for  Logistic Regression
################################
lr = LogisticRegressionCV()
lr.fit(train_X, train_y)
pred_y = lr.predict(test_X)
print("Test fraction correct (LR-Accuracy) = {:.2f}".format(lr.score(test_X, test_y)))
```

Jaringan Saraf Keras untuk Regresi Logistik

Encoding one-hot mengubah fitur ke format yang bekerja lebih baik dengan algoritma klasifikasi dan regresi.

```
#################################
# Keras Neural Network for Logistic Regression
#################################

# Make ONE-HOT enconding for converting into categorical variable
def one_hot_encode_object_array(arr):
    uniques, ids = np.unique(arr, return_inverse=True)
    return np_utils.to_categorical(ids, len(uniques))
```

```
# Dividing data into train and test data
train_y_ohe = one_hot_encode_object_array(train_y)
test_y_ohe = one_hot_encode_object_array(test_y)
#Creating a model
model = Sequential()
model.add(Dense(16, input_shape=(4,)))
model.add(Activation('sigmoid'))
model.add(Dense(3))
model.add(Activation('softmax'))
```

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

Gunakan fungsi model.fit untuk melatih model dengan kumpulan data pelatihan.

```
# Actual modelling
model.fit(train_X, train_y_ohe, verbose=0, batch_size=1, nb_epoch=100)
```

Gunakan fungsi model.evaluate untuk mengevaluasi kinerja model.

```
score, accuracy = model.evaluate(test_X, test_y_ohe, batch_size=16, verbose=0)
```

Cetak akurasi yang diperoleh kedua model. Akurasi untuk model berbasis scikit-learn

```
print("\n Test fraction correct (LR-Accuracy) logistic regression = {:.2f}".format(lr.score(test_X, test_y)))
```

Akurasinya adalah 0,83. Akurasi untuk model keras

```
print("Test fraction correct (NN-Accuracy) keras  = {:.2f}".format(accuracy))
```

Akurasinya adalah 0,99.
Jika Anda menjalankan kode, Anda akan melihat output berikut:

```
Using TensorFlow backend.
Test fraction correct (LR-Accuracy) logistic regression =
0.83
Test fraction correct (NN-Accuracy) keras  = 0.99
Epoch 1/100
75/75 [==============================] - 0s - loss: 1.2947 -
acc: 0.4533
Epoch 2/100
75/75 [==============================] - 0s - loss: 1.0353 -
acc: 0.6400
```

```
Epoch 3/100
75/75 [==============================] - 0s - loss: 0.8930 -
acc: 0.6533
...                                 ...                 ...
...                                 ...                 ...
...                                 ...                 ...
...                                 ...                 ...
Epoch 99/100
75/75 [==============================] - 0s - loss: 0.1186 -
acc: 0.9733
Epoch 100/100
75/75 [==============================] - 0s - loss: 0.1167 -
acc: 0.9867

Accuracy is using keras prediction  0.99
Accuracy is using regression  0.89
```

Untuk memberikan contoh kehidupan nyata, saya akan membahas beberapa kode yang menggunakan kumpulan data Fashion MNIST, yang merupakan kumpulan data gambar Zalando.com yang terdiri dari kumpulan pelatihan 60.000 contoh dan kumpulan uji 10.000 contoh. Setiap contoh adalah gambar skala abu-abu 28x28 yang terkait dengan label dari sepuluh kelas.

Data Fashion MNIST: Regresi Logistik di Keras

Buat file Python baru dan impor paket berikut. Pastikan Anda telah menginstal Keras di sistem Anda.

```
from __future__ import print_function
from keras.models import load_model
import keras
import fashion_mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
import numpy as np

batch_size = 128
num_classes = 10
epochs = 2
```

Seperti yang disebutkan, Anda akan menggunakan kumpulan data Fashion MNIST. Simpan data dan label dalam dua variabel berbeda.

```
# the data, shuffled and split between train and test sets
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

Normalisasikan kumpulan data, seperti yang ditunjukkan di sini:

```
#Gaussian Normalization of the dataset
x_train = (x_train-np.mean(x_train))/np.std(x_train)
x_test = (x_test-np.mean(x_test))/np.std(x_test)

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Tentukan modelnya, seperti yang ditunjukkan di sini:

```
#Building a model architecture
model = Sequential()
model.add(Dense(256, activation='elu', input_shape=(784,)))
model.add(Dropout(0.4))
model.add(Dense(512, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

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

model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_data=(x_test, y_test))
```

Simpan model dalam file .h5 (sehingga Anda dapat menggunakannya nanti secara langsung dengan fungsi load_model() dari keras.models) dan cetak akurasi model dalam set pengujian, seperti yang ditunjukkan di sini:

```
#saving the model using the 'model.save' function
model.save('my_model.h5')
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Jika Anda menjalankan kode sebelumnya, Anda akan melihat output berikut:

```
('train-images-idx3-ubyte.gz', <http.client.HTTPMessage object
at 0x00000171338E2B38>)

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_59 (Dense)             (None, 256)               200960
_________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_60 (Dense)             (None, 512)               131584
_________________________________________________________________
dense_61 (Dense)             (None, 10)                5130
=================================================================
Total params: 337,674
Trainable params: 337,674
Non-trainable params: 0
_________________________________________________________________
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
60000/60000 [==============================] - loss: 0.5188 -
acc: 0.8127 - val_loss: 0.4133 - val_acc: 0.8454
Epoch 2/2
60000/60000 [==============================] - loss: 0.3976 -
acc: 0.8545 - val_loss: 0.4010 - val_acc: 0.8513
Test loss: 0.400989927697
Test accuracy: 0.8513
```

5.4 MLP DI DATA IRIS

Multilayer Perceptron adalah model jaringan saraf minimal. Di bagian ini, saya akan menunjukkan kodenya.

Tulis Kode

Buat file Python baru dan impor paket berikut. Pastikan Anda telah menginstal Keras di sistem Anda.

```
##########MLP on iris data ####################

import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.utils import np_utils
```

Muat kumpulan data dengan membaca file CSV menggunakan Pandas.

```
#load and Prepare Data
datatrain = pd.read_csv('./Datasets/iris/iris_train.csv')
```

Tetapkan nilai numerik ke kelas kumpulan data.

```
#change string value to numeric
datatrain.set_value(datatrain['species']=='Iris-setosa',['species'],0)
datatrain.set_value(datatrain['species']=='Iris-versicolor',['species'],1)
datatrain.set_value(datatrain['species']=='Iris-virginica',['species'],2)
datatrain = datatrain.apply(pd.to_numeric)
```

Mengkonversi bingkai data ke array.

```
#change dataframe to array
datatrain_array = datatrain.as_matrix()
```

Pisahkan data dan target dan simpan dalam dua variabel berbeda.

```
# split x and y (feature and target)
xtrain = datatrain_array[:,:4]
ytrain = datatrain_array[:,4]
```

Ubah format target menggunakan Numpy.

```
#change target format
ytrain = np_utils.to_categorical(ytrain)
```

Buat Model Keras Berurutan

Di sini Anda akan membangun model Multilayer Perceptron dengan satu layer tersembunyi.

- Layer input: Layer input berisi empat neuron, mewakili fitur iris (panjang sepal, dll.).
- Layer tersembunyi: Layer tersembunyi berisi sepuluh neuron, dan aktivasinya menggunakan ReLU.
- Layer output: Layer output berisi tiga neuron, mewakili kelas-kelas layer softmax Iris.

```
#Build Keras model
#Multilayer perceptron model, with one hidden layer.
#Input layer : 4 neuron, represents the feature of Iris(Sepal Length etc)
#Hidden layer : 10 neuron, activation using ReLU
#Output layer : 3 neuron, represents the class of Iris, Softmax Layer
model = Sequential()
model.add(Dense(output_dim=10, input_dim=4))
model.add(Activation("relu"))
model.add(Dense(output_dim=3))
model.add(Activation("softmax"))
```

Kompilasi model dan pilih fungsi Optimizer dan kehilangan untuk melatih dan mengoptimalkan data Anda, seperti yang ditunjukkan di sini:

```
#Compile model :choose optimizer and loss function
#optimizer = stochastic gradient descent with no batch-size
#loss function = categorical cross entropy
#learning rate = default from keras.optimizer.SGD, 0.01
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
```

Latih model menggunakan fungsi model.fit, seperti yang ditunjukkan di sini:

```
#train
model.fit(xtrain, ytrain, nb_epoch=100, batch_size=120)
```

Muat dan siapkan data uji, seperti yang ditunjukkan di sini:

```
## Evaluate on test data
#load and Prepare Data
datatest = pd.read_csv('./Datasets/iris/iris_test.csv')
```

Ubah nilai string menjadi nilai numerik, seperti yang ditunjukkan di sini:

```
#change string value to numeric
datatest.set_value(datatest['species']=='Iris-setosa',['species'],0)
datatest.set_value(datatest['species']=='Iris-versicolor',['species'],1)
datatest.set_value(datatest['species']=='Iris-virginica',['species'],2)
datatest = datatest.apply(pd.to_numeric)
```

Konversikan bingkai data ke array, seperti yang ditunjukkan di sini:

```
#change dataframe to array
datatest_array = datatest.as_matrix()
```

Pisahkan x dan y, dengan kata lain, set fitur dan set target, seperti yang ditunjukkan di sini:

```
#split x and y (feature and target)
xtest= datatest_array[:,:4]
ytest = datatest_array[:,4]
```

Buat prediksi pada model yang dilatih, seperti yang ditunjukkan di sini:

```
#get prediction
classes = model.predict_classes(xtest, batch_size=120)
```

Hitung akurasi, seperti yang ditunjukkan di sini:

```
#get accuration
accuration = np.sum(classes == ytest)/30.0 * 100
```

Cetak akurasi yang dihasilkan oleh model, seperti yang ditunjukkan di sini:

```
print("Test Accuration : " + str(accuration) + '%')
print("Prediction :")
print(classes)
print("Target :")
print(np.asarray(ytest,dtype="int32"))
```

Jika Anda menjalankan kode, Anda akan melihat output berikut:

```
Epoch 1/100
120/120 [==============================] - 0s - loss: 2.7240 - acc: 0.3667
Epoch 2/100
120/120 [==============================] - 0s - loss: 2.4166 - acc: 0.3667
Epoch 3/100
120/120 [==============================] - 0s - loss: 2.1622 - acc: 0.4083
Epoch 4/100
120/120 [==============================] - 0s - loss: 1.9456 - acc: 0.6583
Epoch 98/100
120/120 [==============================] - 0s - loss: 0.5571 - acc: 0.9250
```

Epoch 99/100
120/120 [==============================] - 0s - loss: 0.5554 - acc: 0.9250
Epoch 100/100
120/120 [==============================] - 0s - loss: 0.5537 - acc: 0.9250

5.5 MLP PADA DATA MNIST (KLASIFIKASI DIGIT)

MNIST adalah kumpulan data standar untuk memprediksi angka tulisan tangan. Di bagian ini, Anda akan melihat bagaimana Anda dapat menerapkan konsep multilayer perceptrons dan membuat sistem pengenalan digit tulisan tangan. Buat file Python baru dan impor paket berikut. Pastikan Anda telah menginstal Keras di sistem Anda.

```
##########MLP : MNIST Data (Digit Classification) ########################

import numpy as np
import os
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.optimizers import RMSprop
from keras.utils import np_utils
```

Satu variabel penting didefinisikan.

```
np.random.seed(100) # for reproducibility
batch_size = 128 #Number of images used in each optimization step
nb_classes = 10 #One class per digit
nb_epoch = 20 #Number of times the whole data is used to learn
```

Muat kumpulan data menggunakan fungsi mnist.load_data().

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()

#Flatten the data, MLP doesn't use the 2D structure of the data. 784 = 28*28
X_train = X_train.reshape(60000, 784) # 60,000 digit images
X_test = X_test.reshape(10000, 784)
```

Jenis set pelatihan dan set tes dikonversi ke float32.

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

Kumpulan data dinormalisasi; dengan kata lain, mereka diatur ke Z-score.

```
# Gaussian Normalization( Z- score)
X_train = (X_train- np.mean(X_train))/np.std(X_train)
X_test = (X_test- np.mean(X_test))/np.std(X_test)
```

Menampilkan jumlah sampel pelatihan yang ada dalam kumpulan data dan juga jumlah kumpulan uji yang tersedia.

```
#Display number of training and test instances
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

Mengonversi vektor kelas menjadi matriks kelas biner.

```
# convert class vectors to binary class matrices (ie one-hot vectors)
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)
```

Tentukan model sekuensial dari Multilayer Perceptron.

```
#Define the model achitecture
model = Sequential()
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.2)) # Regularization
model.add(Dense(120))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Dense(10)) #Last layer with one output per class
model.add(Activation('softmax')) #We want a score simlar to a probability for each class
```

Gunakan Optimizer.

```
#Use rmsprop as an optimizer
rms = RMSprop()
```

Fungsi untuk mengoptimalkan adalah entropi silang antara label sebenarnya dan output (softmax) dari model.

```
#The function to optimize is the cross entropy between the true label and the output (softmax) of the model
model.compile(loss='categorical_crossentropy', optimizer=rms, metrics=["accuracy"])
```

Gunakan fungsi model.fit untuk melatih model.

```
#Make the model learn
model.fit(X_train, Y_train,
batch_size=batch_size, nb_epoch=nb_epoch,
verbose=2,
validation_data=(X_test, Y_test))
```

Menggunakan model, mengevaluasi fungsi untuk mengevaluasi kinerja model.

```
#Evaluate how the model does on the test set
score = model.evaluate(X_test, Y_test, verbose=0)
```

Cetak akurasi yang dihasilkan dalam model.

```
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

Jika Anda menjalankan kode, Anda akan mendapatkan output berikut: 60000 train sample 10000 test samples

Train on 60000 samples, validate on 10000 samples
Epoch 1/20
13s - loss: 0.2849 - acc: 0.9132 - val_loss: 0.1149 - val_acc: 0.9652
Epoch 2/20
11s - loss: 0.1299 - acc: 0.9611 - val_loss: 0.0880 - val_acc: 0.9741
Epoch 3/20
11s - loss: 0.0998 - acc: 0.9712 - val_loss: 0.1121 - val_acc: 0.9671
Epoch 4/20
Epoch 18/20
14s - loss: 0.0538 - acc: 0.9886 - val_loss: 0.1241 - val_acc: 0.9814
Epoch 19/20
12s - loss: 0.0522 - acc: 0.9888 - val_loss: 0.1154 - val_acc: 0.9829
Epoch 20/20
13s - loss: 0.0521 - acc: 0.9891 - val_loss: 0.1183 - val_acc: 0.9824
Test score: 0.118255248802
Test accuracy: 0.9824

Sekarang, saatnya membuat kumpulan data dan menggunakan Multilayer Perceptron. Di sini Anda akan membuat kumpulan data Anda sendiri menggunakan fungsi acak dan menjalankan model Multilayer Perceptron pada data yang dihasilkan.

5.6 MLP PADA DATA YANG DIHASILKAN SECARA ACAK

Buat file Python baru dan impor paket berikut. Pastikan Anda telah menginstal Keras di sistem Anda.

```
#####MLP on randomly generated Data ##############
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD
import numpy as np
```

Buat data menggunakan fungsi acak.

```
# Generate dummy data
x_train = np.random.random((1000, 20))
# Y having 10 possible categories
y_train = keras.utils.to_categorical(np.random.randint(10, size=(1000, 1)), num_classes=10)
x_test = np.random.random((100, 20))
y_test = keras.utils.to_categorical(np.random.randint(10, size=(100, 1)), num_classes=10)
```

Membuat model sekuensial.

```
#Create a model
model = Sequential()
# Dense(64) is a fully-connected layer with 64 hidden units.
# In the first layer, you must specify the expected input data shape:
# here, 20-dimensional vectors.
model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

Kompilasi modelnya.

```
#Compile the model
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',optimizer=sgd,metrics=['accuracy'])
```

Gunakan fungsi model.fit untuk melatih model

```
# Fit the model
model.fit(x_train, y_train,epochs=20,batch_size=128)
```

Evaluasi kinerja model menggunakan fungsi model.evaluate.

```
# Evaluate the model
score = model.evaluate(x_test, y_test, batch_size=128)
```

Jika Anda menjalankan kode, Anda akan mendapatkan output berikut:

Epoch 1/20
1000/1000 [==============================] - 0s - loss: 2.4432 - acc: 0.0970
Epoch 2/20
1000/1000 [==============================] - 0s - loss: 2.3927 - acc: 0.0850
Epoch 3/20
1000/1000 [==============================] - 0s - loss: 2.3361 - acc: 0.1190
Epoch 4/20
1000/1000 [==============================] - 0s - loss: 2.3354 - acc: 0.1000
Epoch 19/20
1000/1000 [==============================] - 0s - loss: 2.3034 - acc: 0.1160
Epoch 20/20
1000/1000 [==============================] - 0s - loss: 2.3055 - acc: 0.0980
100/100 [==============================] - 0s

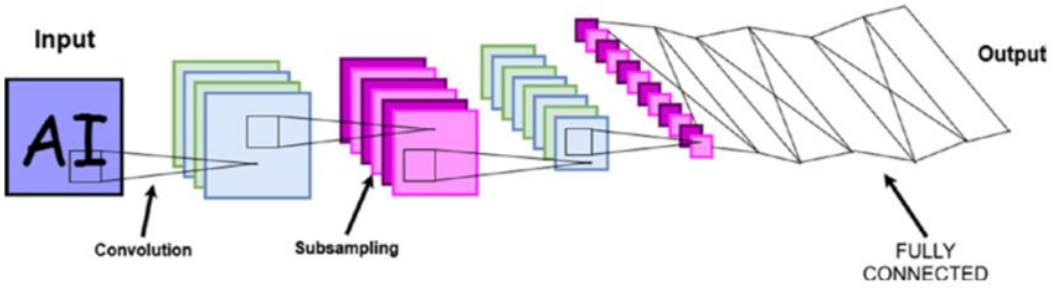
Dalam bab ini, saya membahas bagaimana membangun model linier, logistik, dan MLP di Keras secara sistemik.

BAB 6
JARINGAN SARAF KONVOLUSIONAL
CONVOLUTIONAL NEURAL NETWORKS (CNN)

Convolutional Neural Networks/Jaringan saraf CONVOLUTIONAL (CNN) adalah jaringan saraf tiruan feed-forward yang dalam di mana jaringan saraf mempertahankan struktur hierarkis dengan mempelajari representasi fitur internal dan menggeneralisasi fitur dalam masalah gambar umum seperti pengenalan objek dan masalah penglihatan komputer lainnya. Itu tidak terbatas pada gambar; itu juga mencapai hasil mutakhir dalam masalah pemrosesan bahasa alami dan pengenalan suara.

6.1 LAYER BERBEDA DI CNN

CNN terdiri dari beberapa layer, seperti yang ditunjukkan pada Gambar 6-1.

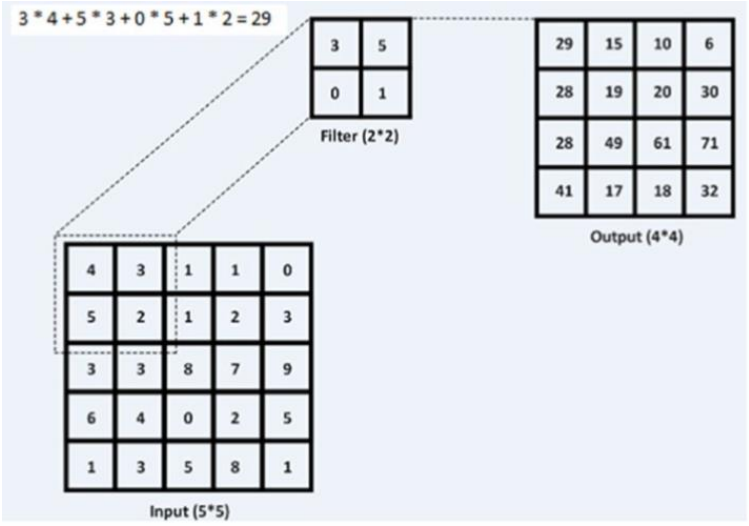


Gambar 6-1 Layer dalam jaringan saraf konvolusi

Layer konvolusi terdiri dari filter dan peta gambar. Pertimbangkan gambar input grayscale memiliki ukuran 5 × 5, yang merupakan matriks nilai 25 piksel. Data gambar dinyatakan sebagai matriks tiga dimensi lebar × tinggi × saluran.

Catatan Peta gambar adalah daftar koordinat yang berkaitan dengan gambar tertentu.

Konvolusi bertujuan untuk mengekstrak fitur dari gambar input, dan karenanya mempertahankan hubungan spasial antara piksel dengan mempelajari fitur gambar menggunakan kotak kecil data input. Invarian rotasi, invarian translasi, dan invarian skala dapat diharapkan. Misalnya, gambar kucing yang diputar atau gambar kucing yang diskalakan ulang dapat dengan mudah diidentifikasi oleh CNN karena langkah konvolusi.

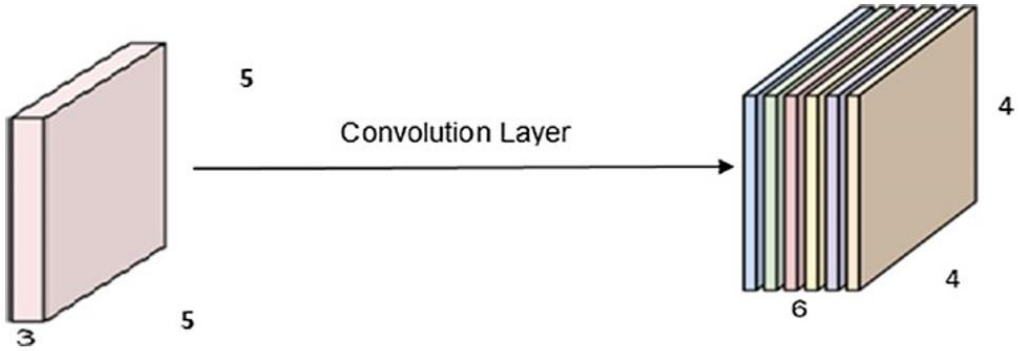


Gambar 6-2. Subsampling

Anda menggeser filter (matriks persegi) di atas gambar asli Anda (di sini, 1 piksel), dan pada setiap posisi tertentu, Anda menghitung perkalian elemen-bijaksana (antara matriks filter dan gambar asli) dan menambahkan output perkalian untuk mendapatkan bilangan bulat terakhir yang membentuk elemen matriks output. Subsampling hanyalah penggabungan rata-rata dengan bobot yang dapat dipelajari per peta fitur, seperti yang ditunjukkan pada Gambar 6-2.

Seperti yang ditunjukkan pada Gambar 6-2, filter memiliki bobot input dan menghasilkan neuron output. Katakanlah Anda mendefinisikan layer konvolusi dengan enam filter dan bidang reseptif dengan lebar 2 piksel dan tinggi 2 piksel dan menggunakan lebar langkah default 1, dan padding default diatur ke 0. Setiap filter menerima input dari 2x2 piksel, bagian gambar. Dengan kata lain, itu adalah 4 piksel sekaligus. Oleh karena itu, Anda dapat mengatakan itu akan membutuhkan bobot input 4 + 1 (bias).

Volume input adalah 5×5×3 (lebar × tinggi × jumlah saluran), ada enam filter ukuran 2×2 dengan langkah 1 dan pad 0. Oleh karena itu, jumlah parameter di layer ini untuk setiap filter memiliki 2* 2*3 + 1 = 13 parameter (ditambahkan +1 untuk bias). Karena ada enam filter, Anda mendapatkan 13*6 = 78 parameter.

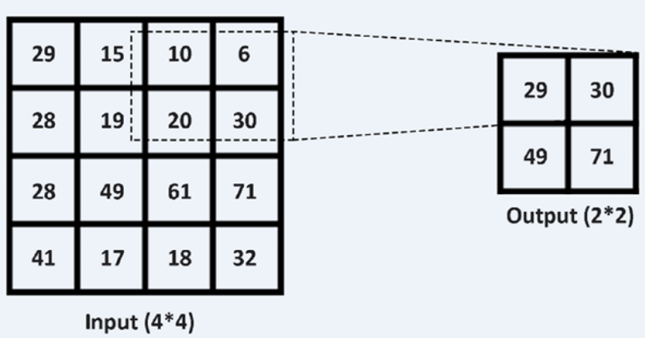


Gambar 6-3. volume Input

Berikut ringkasannya:

- Volume input berukuran W1 × H1 × D1.
- Model membutuhkan hyperparameter: jumlah filter (f), stride (S), jumlah padding nol (P).
- Ini menghasilkan volume ukuran W2 × H2 × D2.
- W2 = (W1-f+ 2P) /S + 1 = 4.
- H2 = (H1-f+2P)/S +1 = 4.
- D2 = Jumlah filter = f = 6.

Layer penyatuan mengurangi peta aktivasi layer sebelumnya. Ini diikuti oleh satu atau lebih layer konvolusi dan menggabungkan semua fitur yang dipelajari di peta aktivasi layer sebelumnya. Ini mengurangi *overfitting* data pelatihan dan menggeneralisasi fitur yang diwakili oleh jaringan. Ukuran bidang reseptif hampir selalu diatur ke 2x2 dan menggunakan langkah 1 atau 2 (atau lebih tinggi) untuk memastikan tidak ada tumpang tindih. Anda akan menggunakan operasi maks untuk setiap bidang reseptif sehingga aktivasi adalah nilai input maksimum. Di sini, setiap empat angka dipetakan menjadi hanya satu angka. Jadi, jumlah piksel turun menjadi seperempat dari aslinya pada langkah ini (Gambar 6-4).

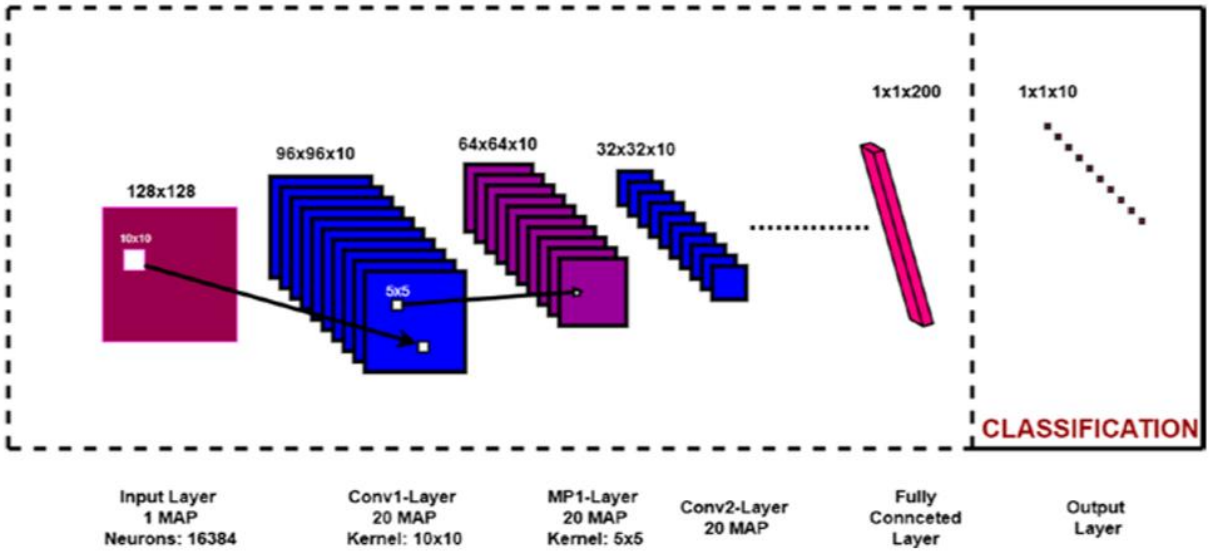


Gambar 6-4. *Maxpooling*-mengurangi jumlah piksel

Layer yang sepenuhnya terhubung adalah layer jaringan saraf tiruan feed-forward. Layer ini memiliki fungsi aktivasi nonlinier untuk menghasilkan probabilitas prediksi kelas. Mereka digunakan menjelang akhir setelah semua fitur diidentifikasi dan diekstraksi oleh layer konvolusi dan telah dikonsolidasikan oleh layer penyatuan dalam jaringan. Di sini, layer tersembunyi dan layer output adalah layer yang sepenuhnya terhubung.

6.2 ARSITEKTUR CNN

CNN adalah arsitektur jaringan saraf dalam *feed-forward* yang terdiri dari beberapa layer konvolusi, masing-masing diikuti oleh layer penyatuan, fungsi aktivasi, dan normalisasi batch opsional. Ini juga terdiri dari layer yang terhubung penuh. Saat gambar bergerak melalui jaringan, itu menjadi lebih kecil, sebagian besar karena pengumpulan maksimal. Layer terakhir menampilkan prediksi probabilitas kelas.



Gambar 6-5. Arsitektur CNN untuk Klasifikasi

Beberapa tahun terakhir telah melihat banyak arsitektur yang dikembangkan yang telah membuat kemajuan luar biasa di bidang klasifikasi gambar. Jaringan pra-pelatihan pemenang penghargaan (VGG16, VGG19, ResNet50, Inception V3, dan Xception) telah digunakan untuk berbagai tantangan klasifikasi gambar termasuk pencitraan medis. Pembelajaran transfer adalah jenis praktik di mana Anda menggunakan model yang telah dilatih sebelumnya selain beberapa layer. Ini dapat digunakan untuk menyelesaikan tantangan klasifikasi gambar di setiap bidang.

BAB 7
CNN DI TENSORFLOW

Bab ini akan menunjukkan cara menggunakan TensorFlow untuk membangun model CNN. Model CNN dapat membantu Anda membangun pengklasifikasi gambar yang dapat memprediksi/mengklasifikasikan gambar. Secara umum, Anda akan membuat beberapa layer dalam arsitektur model dengan nilai awal bobot dan bias. Kemudian Anda menyetel bobot dan bias dengan bantuan kumpulan data pelatihan. Ada pendekatan lain yang melibatkan penggunaan model terlatih seperti InceptionV3 untuk mengklasifikasikan gambar. Anda dapat menggunakan pendekatan pembelajaran transfer ini di mana Anda menambahkan beberapa layer (yang parameternya dilatih) di atas layer model yang telah dilatih sebelumnya (dengan nilai parameter utuh) untuk membuat pengklasifikasi yang sangat kuat. Dalam bab ini, saya akan menggunakan TensorFlow untuk menunjukkan cara mengembangkan jaringan konvolusi untuk berbagai aplikasi visi komputer. Lebih mudah untuk mengekspresikan arsitektur CNN sebagai grafik aliran data.

7.1 MENGAPA TENSORFLOW UNTUK MODEL CNN?

Di TensorFlow, gambar dapat direpresentasikan sebagai array tiga dimensi atau tensor bentuk (tinggi, lebar, dan saluran). TensorFlow memberikan fleksibilitas untuk melakukan iterasi dengan cepat, memungkinkan Anda melatih model lebih cepat, dan memungkinkan Anda menjalankan lebih banyak eksperimen. Saat membawa model TensorFlow ke produksi, Anda dapat menjalankannya di GPU dan TPU skala besar.

7.2 KODE TENSORFLOW UNTUK PENGKLASIFIKASI GAMBAR UNTUK DATA MNIST

Di bagian ini, saya akan memberi Anda contoh untuk memahami cara menerapkan CNN di TensorFlow. Kode berikut mengimpor kumpulan data MNIST dengan gambar angka abu-abu 28x28 dari paket kontribusi TensorFlow dan memuat semua Library yang diperlukan. Di sini, tujuannya adalah untuk membangun classifier untuk memprediksi digit yang diberikan pada gambar.

```
from tensorflow.contrib.learn.python.learn.datasets.mnist
import read_data_sets
from tensorflow.python.framework
import ops import tensorflow as tf
import numpy as np
Anda kemudian memulai sesi grafik.
# Start a graph session
sess = tf.Session()
```

Anda memuat data MNIST dan membuat rangkaian kereta dan pengujian.

```
# Load data from keras.datasets
import mnist (X_train, y_train),
(X_test, y_test) = mnist.load_data()
```

Anda kemudian menormalkan fitur rangkaian kereta dan pengujian.

```
# Z- score  or Gaussian Normalization
X_train = X_train - np.mean(X_train) / X_train.std()
X_test = X_test - np.mean(X_test) / X_test.std()
```

Karena ini adalah masalah klasifikasi multikelas, selalu lebih baik menggunakan pengkodean satu-panas dari nilai-nilai kelas output.

Convert labels into one-hot encoded vectors

```
num_class = 10
train_labels = tf.one_hot(y_train, num_class)
test_labels = tf.one_hot(y_test, num_class)
```

Mari kita atur parameter model sekarang karena gambar ini berwarna abu-abu. Oleh karena itu, kedalaman gambar (saluran) adalah 1.

```
# Set model parameters
batch_size = 784
samples =500
learning_rate = 0.03
img_width = X_train[0].shape[0]
img_height = X_train[0].shape[1]
target_size = max(train_labels) + 1
num_channels = 1 # greyscale = 1
channel
epoch = 200
 no_ channels = 1
conv1_features = 30
filt1_features = 5
conv2_features = 15
filt2_features = 3
max_pool_size1 = 2 # NxN window for 1st max pool layer
max_pool_size2 = 2 # NxN window for 2nd max pool layer
fully_connected_size1 = 150
```

Mari kita nyatakan placeholder untuk model. Fitur data input, variabel target, dan ukuran batch dapat diubah untuk set pelatihan dan evaluasi.

```
# Declare model placeholders
x_input_shape = (batch_size, img_width, img_height, num_channels)
x_input = tf.placeholder(tf.float32, shape=x_input_shape)
y_target = tf.placeholder(tf.int32, shape=(batch_size))
eval_input_shape = (samples, img_width, img_height, num_channels)
eval_input = tf.placeholder(tf.float32, shape=eval_input_shape)
eval_target = tf.placeholder(tf.int32, shape=(samples))
```

Mari kita nyatakan bobot variabel model dan nilai bias untuk input dan neuron layer tersembunyi.

```
# Declare model variables
W1 = tf.Variable(tf.random_normal([filt1_features,
filt1_features, no_channels, conv1_features]))
b1 = tf.Variable(tf.ones([conv1_features]))
W2 = tf.Variable(tf.random_normal([filt2_features,
filt2_features, conv1_features, conv2_features]))
b2 = tf.Variable(tf.ones([conv2_features]))
```

Mari kita deklarasikan variabel model untuk layer yang terhubung penuh dan tentukan bobot dan bias untuk 2 layer terakhir ini.

```
# Declare model variables for fully connected layers
resulting_width = img_width // (max_pool_size1 * max_pool_size2)
resulting_height = img_height // (max_pool_size1 * max_pool_size2)
full1_input_size = resulting_width * resulting_height * conv2_ features
```

```
W3 = tf.Variable(tf.truncated_normal([full1_input_size,  fully_connected_size1],
stddev=0.1, dtype=tf.float32))
b3 = tf.Variable(tf.truncated_normal([fully_connected_size1], stddev=0.1,
dtype=tf.float32))
W_out = tf.Variable(tf.truncated_normal([fully_connected_size1, target_size],
stddev=0.1, dtype=tf.float32))
b_out = tf.Variable(tf.truncated_normal([target_size], stddev=0.1, dtype=tf.float32))
```

Mari kita buat fungsi pembantu untuk menentukan layer convolutional dan pooling maks.

```
# Define helper functions for the convolution and maxpool layers:
def     conv_layer(x, W, b):
        conv = tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
        conv_with_b = tf.nn.bias_add(conv, b)
        conv_out = tf.nn.relu(conv_with_b)
        return conv_out
def     maxpool_layer(conv, k=2):
        return tf.nn.max_pool(conv, ksize=[1, k, k, 1],
        strides=[1, k, k, 1], padding='SAME')
```

Sebuah model jaringan saraf didefinisikan dengan dua layer tersembunyi dan dua layer yang terhubung penuh. Unit linier yang diperbaiki digunakan sebagai fungsi aktivasi untuk layer tersembunyi dan layer output akhir.

```
# Initialize Model Operations
    def my_conv_net(input_data):
# First Conv-ReLU-MaxPool Layer
    conv_out1 = conv_layer(input_data, W1, b1)
    maxpool_out1 = maxpool_layer(conv_out1)
# Second Conv-ReLU-MaxPool Layer
    conv_out2 = conv_layer(maxpool_out1, W2, b2)
    maxpool_out2 = maxpool_layer(conv_out2)
# Transform Output into a 1xN layer for next fully connected layer
    final_conv_shape = maxpool_out2.get_shape().as_list()
    final_shape = final_conv_shape[1] * final_conv_shape[2] *
    final_conv_shape[3]flat_output = tf.reshape(maxpool_out2, [final_conv_shape[0],
    final_shape])
# First Fully Connected Layer
    fully_connected1 = tf.nn.relu(tf.add(tf.matmul(flat_output, W3), b3))
# Second Fully Connected Layer final_model_output =
    tf.add(tf.matmul(fully_connected1, W_out), b_out)
return(final_model_output)
model_output = my_conv_net(x_input)
test_model_output = my_conv_net(eval_input)
```

Anda akan menggunakan fungsi entropi lintas softmax (cenderung bekerja lebih baik untuk klasifikasi multikelas) untuk menentukan kerugian yang beroperasi pada logit.

```
# Declare Loss Function (softmax cross entropy)
        loss = tf.reduce_mean(tf.nn.sparse_softmax_cross_entropy_with_
      logits(logits=model_output, labels=y_target))
```

Mari kita definisikan fungsi prediksi rangkaian kereta dan tes.

```
# Create a prediction function
prediction = tf.nn.softmax(model_output)
```

```
test_prediction = tf.nn.softmax(test_model_output)
```

Untuk menentukan akurasi model pada setiap batch, mari kita definisikan fungsi akurasi.

```
# Create accuracy function
    def get_accuracy(logits, targets):
    batch_predictions = np.argmax(logits, axis=1)
    num_correct = np.sum(np.equal(batch_predictions, targets))
    return(100. * num_correct/batch_predictions.shape[0])
```

Mari deklarasikan langkah pelatihan dan definisikan fungsi Optimizer.

```
# Create an optimizer
      my_optimizer = tf.train.AdamOptimizer(learning_rate, 0.9)
      train_step = my_optimizer.minimize(loss)
```

Mari kita inisialisasi semua variabel model yang dideklarasikan sebelumnya.

```
# Initialize Variables
    varInit = tf.global_variables_initializer()
    sess.run(varInit)
```

Mari kita mulai melatih model dan mengulang secara acak melalui kumpulan data. Anda ingin mengevaluasi model di kereta dan menguji kumpulan kumpulan dan mencatat kehilangan dan akurasi.

```
# Start training loop
train_loss = []
train_acc = []
test_acc = []
for i in range(epoch):
    random_index = np.random.choice(len(X_train), size=batch_size)
    random_x = X_train[random_index]
    random_x = np.expand_dims(random_x, 3)
    random_y = train_labels[random_index]
    train_dict = {x_input: random_x, y_target: random_y}
    sess.run(train_step, feed_dict=train_dict)
    temp_train_loss, temp_train_preds = sess.run([loss, prediction],
    feed_dict=train_dict)
    temp_train_acc = get_accuracy(temp_train_preds, random_y)
    eval_index = np.random.choice(len(X_test),  size=evaluation_size)
    eval_x = X_test[eval_index]    eval_x = np.expand_dims(eval_x, 3)
    eval_y = test_labels[eval_index]
    test_dict = {eval_input: eval_x, eval_target: eval_y}
    test_preds = sess.run(test_prediction, feed_dict=test_dict)
    temp_test_acc = get_accuracy(test_preds, eval_y)
```

Hasil model dicatat dalam format berikut dan dicetak dalam output:

```
# Record and print results
train_loss.append(temp_train_loss)
train_acc.append(temp_train_acc)
test_acc.append(temp_test_acc)
print('Epoch # {}. Train Loss: {:.2f}. Train Acc : {:.2f} .
temp_test_acc :  {:.2f}'.format(i+1,temp_train_loss,
temp_train_acc,temp_test_acc))
```

7.3 MENGGUNAKAN API TINGKAT TINGGI UNTUK MEMBUAT MODEL CNN

TFLearn, TensorLayer, tflayers, TF-Slim, tf.contrib.learn, Pretty Tensor, keras, dan Sonnet adalah API TensorFlow tingkat tinggi. Jika Anda menggunakan salah satu API tingkat tinggi ini, Anda dapat membuat model CNN dalam beberapa baris kode. Jadi, Anda dapat menjelajahi salah satu API ini untuk bekerja dengan cerdas.

BAB 8
CNN DI KERAS

Bab ini akan menunjukkan bagaimana cara menggunakan Keras untuk membangun model CNN. Model CNN dapat membantu Anda membangun pengklasifikasi gambar yang dapat memprediksi dan mengklasifikasikan gambar. Secara umum, Anda membuat beberapa layer dalam arsitektur model dengan nilai awal bobot dan bias. Kemudian Anda menyetel variabel bobot dan bias dengan bantuan kumpulan data pelatihan. Anda akan belajar cara membuat kode di Keras dalam konteks ini. Ada pendekatan lain yang melibatkan penggunaan model pra-pelatihan seperti InceptionV3 dan ResNet50 yang dapat mengklasifikasikan gambar.

Mari kita tentukan model CNN dan evaluasi seberapa baik kinerjanya. Anda akan menggunakan struktur dengan layer konvolusi; maka Anda akan menggunakan max pooling dan meratakan jaringan untuk sepenuhnya menghubungkan layer dan membuat prediksi.

8.1 MEMBUAT PENGKLASIFIKASI GAMBAR UNTUK DATA MNIST DI KERAS

Di sini saya akan mendemonstrasikan proses membangun pengklasifikasi untuk digit tulisan tangan menggunakan kumpulan data MNIST yang populer. Tugas ini merupakan tantangan besar untuk bermain dengan jaringan saraf, tetapi dapat dikelola di satu komputer. Database MNIST berisi 60.000 gambar pelatihan dan 10.000 gambar pengujian. Mulailah dengan mengimpor Numpy dan menyetel benih untuk generator nomor pseudorandom komputer. Ini memungkinkan Anda untuk mereproduksi hasil dari skrip Anda.
import numpy as np

```
# random seed for reproducibility
np.random.seed(123)
```

Selanjutnya, Anda mengimpor tipe model sekuensial dari Keras. Ini hanyalah tumpukan linier layer jaringan saraf.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2d
#Now we will import some utilities
from keras.utils import np_utils
#Fixed dimension ordering issue
from keras import backend as K
K.set_image_dim_ordering('th')
```

#Load image data from MNIST

```
#Load pre-shuffled MNIST data into train and test sets (X_train,y_train),(X_test,
y_test)=mnist.load_data()
```

#Preprocess imput data for Keras

```
# Reshape input data.
# reshape to be [samples][channels][width][height]
X_train=X_train.reshape(X_train.shape[0],1,28,28)
X_test=X_test.reshape(X_test.shape[0],1,28,28)
# to convert our data type to float32 and normalize our database
```

X_train=X_train.astype('float32')
X_test=X_test.astype('float32') print(X_train.shape)
 # Z-scoring or Gaussian Normalization
X_train=X_train - np.mean(X_train) / X_train.std()
X_test=X_test – np.mean(X_test) / X_test.std()
#(60000, 1, 28, 28)
 # convert 1-dim class arrays to 10 dim class metrices
#one hot encoding outputs y_train=np_utils.to_categorical(y_train)
y_test-np_utils.to_categorical(y_test) num_classes=y_test.shape[1] print(num_classes)
#10
#Define a simple CNN model
print(X_train.shape)
#(60000,1,28,28)

Perbaiki Struktur Jaringan

Struktur jaringannya adalah sebagai berikut:

- Jaringan memiliki layer input convolutional, dengan 32 peta fitur dengan ukuran 5×5. Fungsi aktivasi adalah unit linier yang diperbaiki.
- Layer kolam maksimal memiliki ukuran 2×2.
- Putus sekolah diatur menjadi 30 persen.
- Anda bisa meratakan layer.
- Jaringan memiliki layer yang terhubung penuh dengan 240 unit, dan fungsi aktivasi adalah unit linier eksponensial.
- Layer terakhir dari jaringan adalah layer output yang terhubung penuh dengan sepuluh unit, dan fungsi aktivasinya adalah softmax.

Kemudian Anda mengkompilasi model dengan menggunakan entropi silang biner sebagai fungsi loss dan adagrad sebagai Optimizer.

Tentukan Arsitektur Model

Arsitekturnya terdiri dari kombinasi layer convolutional dan layer max pooling dan layer padat di ujungnya.

 # create a model

model=Sequential()
model.add(Conv2D(32, (5,5), input_shape=(1,28,28), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.3))# Dropout, one form of regularization
model.add(Flatten())
model.add(Dense(240,activation='elu'))
model.add(Dense(num_classes, activation='softmax')) print(model.output_shape)
(None, 10)

#compile the model

model.compile(loss='binary_crossentropy', optimizer='adagrad',
matrices=['accuracy'])

Kemudian Anda menyesuaikan model dengan menggunakan training data set dengan mengambil ukuran batch 200. Model mengambil 200 instans/baris pertama (dari tanggal 1 hingga ke-200) dari training data set dan melatih jaringan. Kemudian model mengambil 200 instans kedua (dari tanggal 201 hingga ke-400) untuk jaringan pelatihan lagi. Dengan cara ini, Anda menyebarkan semua instance melalui jaringan. Model ini membutuhkan lebih sedikit memori saat Anda melatih jaringan dengan lebih sedikit instance setiap kali. Tetapi ukuran

batch yang kecil tidak menawarkan perkiraan gradien yang baik, dan karenanya menyetel bobot dan bias dapat menjadi tantangan.

Satu epoch berarti satu umpan maju dan satu umpan mundur dari semua contoh latihan. Dibutuhkan beberapa iterasi untuk menyelesaikan satu epoch. Di sini, Anda memiliki 60.000 contoh pelatihan, dan ukuran batch Anda adalah 200, jadi dibutuhkan 300 iterasi untuk menyelesaikan 1 epoch.

```
# Fit the model
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=1, batch_size=200)
```

 # Evaluate model on test data

```
# Final evaluation of the model
scores =model.evaluate(X_test, y_test, verbose=0)
print("CNN error: % .2f%%" % (100-scores[1]*100))
# CNN Error: 17.98%
# Save the model
# save model
model_json= model.to_join()
with open("model_json", "w") as
json_file:    json_file.write(model_json)
# serialize weights to HDFS
model.save_weights("model.h5")
```

8.2 MEMBUAT PENGKLASIFIKASI GAMBAR DENGAN DATA CIFAR-10

Bagian ini menjelaskan bagaimana Anda dapat membuat pengklasifikasi yang dapat mengklasifikasikan sepuluh label dari kumpulan data CIFAR-10 menggunakan model Keras CNN.

Perhatikan bahwa kumpulan data CiFar-10 terdiri dari 60.000 gambar berwarna 32×32 dalam 10 kelas, dengan 6.000 gambar per kelas. ada 50.000 gambar pelatihan dan 10.000 gambar uji.

```
############Building CNN Model with CIFAR10 data####################
# plot cifar10 instances
from keras.datasets import cifar10
from matplotlib import pyplot
from scipy.misc import toimage
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2d
#Now we will import some utilities
from keras.utils import np_utils
from keras.layers.normalization import BatchNormalization
#Fixed dimension ordering issue
from keras import backend as K
K.set_image_dim_ordering('th')
# fix random seed for reproducibility
seed=12
```

```
numpy.random.seed(seed)
#Preprocess imput data for Keras
# Reshape input data.
# reshape to be [samples][channels][width][height]
X_train=X_train.reshape(X_train.shape[0],3,32,32).
astype('float32')
X_test=X_test.reshape(X_test.shape[0],3,32,32). astype('float32')
# Z-scoring or Gaussian Normalization
X_train=X_train - np.mean(X_train) / X_train.std()
X_test=X_test – np.mean(X_test) / X_test.std()
# convert 1-dim class arrays to 10 dim class metrices
#one hot encoding outputs
y_train=np_utils.to_categorical(y_train)
y_test-np_utils.to_categorical(y_test)
num_classes=y_test.shape[1]
print(num_classes)    #10
#Define a simple CNN model
print(X_train.shape)
#(50000,3,32,32)
```

Tentukan Struktur Jaringan

Struktur jaringannya adalah sebagai berikut:

- Layer input convolutional memiliki 32 peta fitur dengan ukuran 5×5, dan fungsi aktivasinya adalah unit linier yang diperbaiki.
- Layer kolam maksimal memiliki ukuran 2×2.
- Layer convolutional memiliki 32 peta fitur dengan ukuran 5x5, dan fungsi aktivasi adalah unit linier yang diperbaiki.
- Jaringan memiliki normalisasi batch.
- Layer kolam maksimal memiliki ukuran 2×2.
- Putus sekolah diatur menjadi 30 persen.
- Anda bisa meratakan layer.
- Layer yang terhubung penuh memiliki 240 unit, dan fungsi aktivasi adalah unit linier eksponensial.
- Layer output yang terhubung penuh memiliki sepuluh unit, dan fungsi aktivasinya adalah softmax.

Kemudian Anda menyesuaikan model dengan menggunakan training data set dengan mengambil ukuran batch 200. Anda mengambil 200 instans/baris pertama (dari tanggal 1 hingga ke-200) dari training data set dan melatih jaringan. Kemudian Anda mengambil 200 instans kedua (dari tanggal 201 hingga ke-400) untuk melatih jaringan lagi. Dengan cara ini, Anda menyebarkan semua instance melalui jaringan. Satu epoch berarti satu umpan maju dan satu umpan mundur dari semua contoh latihan. Dibutuhkan beberapa iterasi untuk menyelesaikan satu epoch. Di sini, Anda memiliki 50.000 contoh pelatihan, dan ukuran batch Anda adalah 200, sehingga dibutuhkan 250 iterasi untuk menyelesaikan 1 epoch.

8.3 TENTUKAN ARSITEKTUR MODEL

Model sekuensial dibuat dengan kombinasi layer convolutional dan max pooling. Kemudian layer padat yang terhubung penuh terpasang.

 # create a model

```
model=Sequential()
```

```
model.add(Conv2D(32, (5,5), input_shape=(3,32,32),
activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Conv2D(32, (5,5), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.3))
# Dropout, one form of regularization
model.add(Flatten())
model.add(Dense(240,activation='elu'))
model.add(Dense(num_classes, activation='softmax'))
print(model.output_shape)
model.compile(loss='binary_crossentropy', optimizer='adagrad')
# fit model
model.fit(X_train, y_train, validation_data=(X_test,
y_test), epochs=1, batch_size=200)
# Final evaluation of the model
scores =model.evaluate(X_test, y_test, verbose=0)
print("CNN error: % .2f%%" % (100-scores[1]*100
```

8.4 MODEL TERLATIH

Di bagian ini, saya akan menunjukkan bagaimana Anda dapat menggunakan model pra-latihan seperti VGG dan inception untuk membangun classifier.

```
from keras import applications
from keras,models import Sequential, Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input,
decode_predictions
from keras.models import Model
model = VGG16(weights='imagenet', include_top=True)
model.summary()

#predicting for any new image based on the pre-trained model
# Loading Image
img = image.load_img('('./Data/horse.jpg', target_size=(224, 224))
img = image.img_to_array(img)
img = np.expand_dims(img, axis=0)
img=preprocess_input(img)

# Predict the output
preds = model.predict(img)

# decode the predictions
pred_class = decode_predictions(preds, top=3)[0][0]
print("Predicted Class: %s" %pred_class[1])
print("Confidence("Confidance: %s"% pred_class[2])
#Predicted Class: hartebeest
#Confidence: 0.964784
ResNet50 and InceptionV3 models can be easily utilized for
prediction/classification of new images.
from keras.applications import ResNet50
model = ResNet50(weights='imagenet' , include_top=True)
model.summary()

# create the base pre-trained model
from keras.applications import InceptionV3
model = InceptionV3(weights='imagenet')
model.summary
```

Model pra-pelatihan Inception-V3 dapat mendeteksi/mengklasifikasikan objek dari 22.000 kategori. Dapat mendeteksi/mengklasifikasikan nampan, obor, payung dan lain-lain. Dalam banyak skenario, kita perlu membangun pengklasifikasi sesuai kebutuhan kita. Untuk itu digunakan transfer learning dimana kita menggunakan model *pre-trained* (digunakan untuk ekstraksi fitur) dan *multiple neural.*

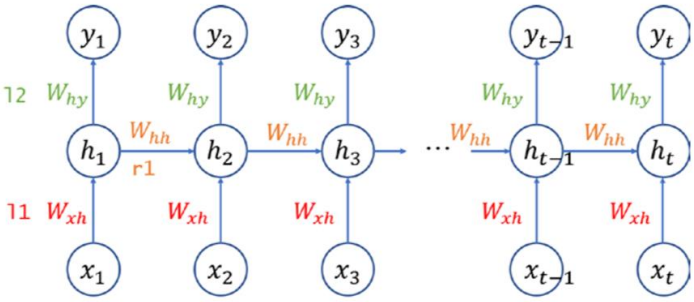
BAB 9
RNN DAN LSTM

Bab ini akan membahas konsep jaringan saraf berulang/*recurrent neural networks* (RNN) dan versi modifikasinya, memori jangka pendek panjang/ *long short-term memory* (LSTM). LSTM terutama digunakan untuk prediksi urutan. Anda akan belajar tentang variasi prediksi urutan dan kemudian belajar bagaimana melakukan prediksi deret waktu dengan bantuan model LSTM.

9.1 KONSEP RNN

Jaringan saraf berulang adalah jenis jaringan saraf tiruan yang paling cocok untuk mengenali pola dalam urutan data, seperti teks, video, ucapan, bahasa, genom, dan data deret waktu. RNN adalah algoritma yang sangat kuat yang dapat mengklasifikasikan, mengelompokkan, dan membuat prediksi tentang data, terutama deret waktu dan teks.

RNN dapat dilihat sebagai jaringan MLP dengan penambahan loop pada arsitekturnya. Pada Gambar 9-1, Anda dapat melihat bahwa ada layer input (dengan simpul seperti x1, x2, dan seterusnya), layer tersembunyi (dengan simpul seperti h1, h2, dan seterusnya), dan layer output (dengan node seperti y1, y2, dan seterusnya). Ini mirip dengan arsitektur MLP. Perbedaannya adalah bahwa node dari layer tersembunyi saling berhubungan. Dalam RNN/LSTM vanilla (dasar), node terhubung dalam satu arah. Ini berarti bahwa h2 bergantung pada h1 (dan x2), dan h3 bergantung pada h2 (dan x3). Node di layer tersembunyi ditentukan oleh node sebelumnya di layer tersembunyi.



$$h_t = \tanh(\, l1(x_t) + r1(h_{t-1})\,)$$
$$y_t = l2(h_t)$$

Gambar 9-1. Sebuah RNN

Jenis arsitektur ini memastikan bahwa output pada t=n bergantung pada input pada t=n, t=n-1, …, dan t=1. Dengan kata lain, output bergantung pada urutan data, bukan pada satu bagian data (Gambar 9-2).

(Input1) → Output1

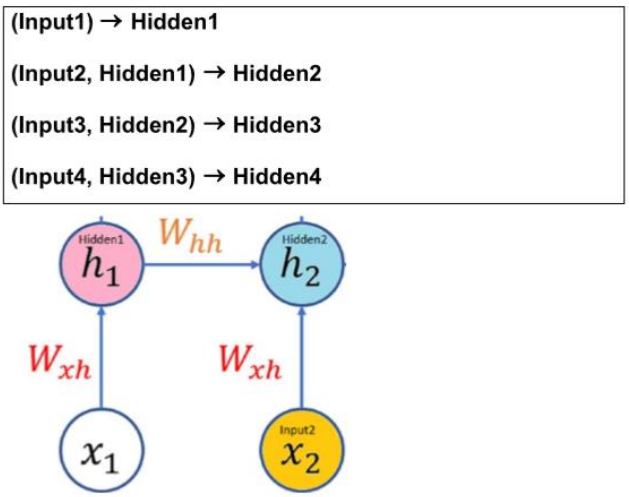
(Input2, Input1) → Output2

(Input3, Input2, Input1) → Output3

(Input4, Input3, Input2, Input1) → Output4

Gambar 9-2. Urutannya

Gambar 9-3 menunjukkan bagaimana node dari layer tersembunyi terhubung ke node dari layer input.



Gambar 9-3. Koneksi

Dalam RNN, jika urutannya cukup panjang, gradien (yang penting untuk menyetel bobot dan bias) dihitung selama pelatihannya (propagasi balik). Mereka baik menghilang (perkalian banyak nilai kecil kurang dari 1) atau meledak (perkalian banyak nilai besar lebih dari 1), menyebabkan model untuk melatih sangat lambat.

9.2 KONSEP LSTM

Memori jangka pendek panjang adalah arsitektur RNN yang dimodifikasi yang menangani masalah menghilangnya dan meledaknya gradien dan mengatasi masalah pelatihan pada urutan panjang dan mempertahankan memori. Semua RNN memiliki loop umpan balik di layer berulang. Putaran umpan balik membantu menjaga informasi dalam "memori" dari waktu ke waktu. Namun, mungkin sulit untuk melatih RNN standar untuk memecahkan masalah yang memerlukan pembelajaran dependensi temporal jangka panjang. Sejak gradien fungsi loss meluruh secara eksponensial dengan waktu (fenomena yang dikenal sebagai masalah gradien hilang), sulit untuk melatih RNN khas. Itulah sebabnya RNN dimodifikasi sedemikian rupa sehingga mencakup sel memori yang dapat menyimpan informasi dalam memori untuk jangka waktu yang lama. RNN yang dimodifikasi lebih dikenal sebagai LSTM. Di LSTM, satu set gerbang digunakan untuk mengontrol ketika informasi memasuki memori, yang memecahkan masalah gradien yang hilang atau meledak.

Koneksi berulang menambahkan status atau memori ke jaringan dan memungkinkannya untuk mempelajari dan memanfaatkan sifat pengamatan yang teratur dalam Input Sequence. Memori internal berarti output jaringan bergantung pada konteks terkini dalam Input Sequence, bukan apa yang baru saja disajikan sebagai input ke jaringan.

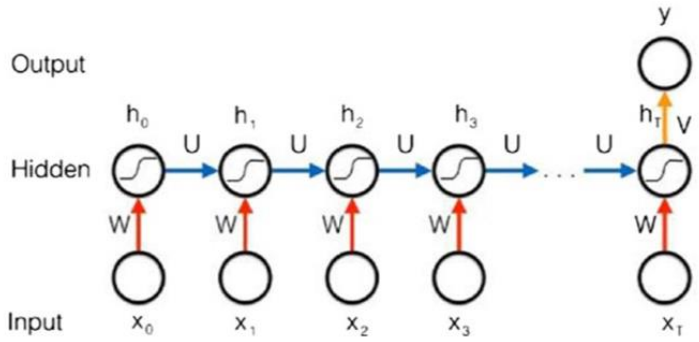
9.3 MODE LSTM

LSTM dapat memiliki salah satu mode berikut:

- Model satu lawan satu
- Model satu-ke-banyak
- Model banyak-ke-satu
- Model banyak ke banyak

Selain mode ini, model banyak-ke-banyak yang disinkronkan juga digunakan, terutama untuk klasifikasi video.

Gambar 9-4 menunjukkan LSTM banyak-ke-satu. Ini menyiratkan bahwa banyak input membuat satu output dalam model ini.



Gambar 9-4. LSTM banyak-ke-satu

9.4 PREDIKSI URUTAN

LSTM paling cocok untuk data urutan. LSTM dapat memprediksi, mengklasifikasikan, dan menghasilkan data urutan. Urutan berarti urutan pengamatan, bukan serangkaian pengamatan. Contoh urutan adalah rangkaian pengujian di mana stempel waktu dan nilai berada dalam urutan (kronologis) dari urutan. Contoh lain adalah video, yang dapat dianggap sebagai rangkaian gambar atau rangkaian klip audio.

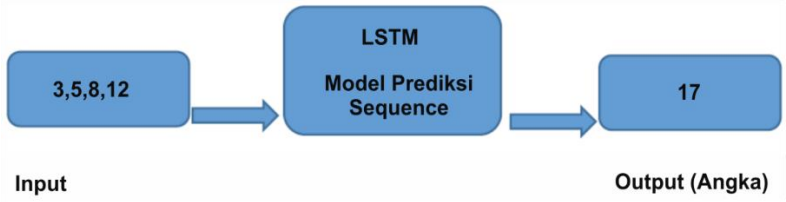
Prediksi berdasarkan urutan data disebut prediksi urutan. Prediksi urutan dikatakan memiliki empat jenis.

- Prediksi numerik urutan
- Klasifikasi urutan
- Pembuatan urutan
- Prediksi urutan-ke-urutan

Prediksi Numerik Urutan

Prediksi numerik urutan memprediksi nilai berikutnya untuk urutan yang diberikan. Kasus penggunaannya adalah prediksi pasar saham dan prediksi cuaca. Lihat contoh dibawah ini:

- *Input Sequence: 3,5,8,12*
- *Output: 17*

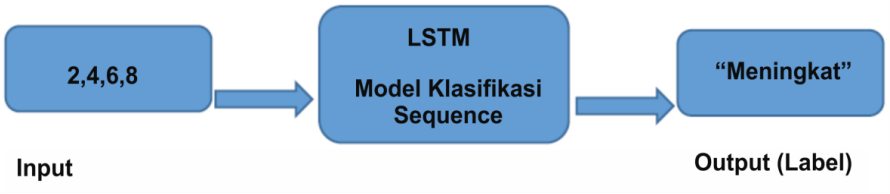


Gambar 9-5. Contoh Model Prediksi

Klasifikasi Urutan

Klasifikasi urutan memprediksi label kelas untuk urutan yang diberikan. Kasus penggunaannya adalah deteksi penipuan (yang menggunakan urutan transaksi sebagai input untuk mengklasifikasikan/memprediksi apakah suatu akun telah diretas atau tidak) dan klasifikasi siswa berdasarkan kinerja (urutan nilai ujian selama enam bulan terakhir secara kronologis). Lihat contoh dibawah ini:

- *Input Sequence: 2,4,6,8*
- *Output: "Meningkat"*

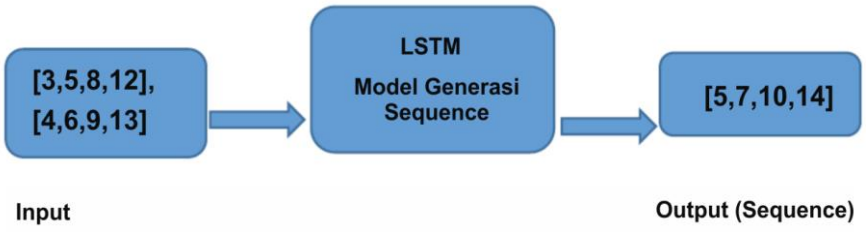


Gambar 9-6. Contoh Model Klasifikasi

Pembuatan Urutan

Pembuatan urutan adalah ketika Anda menghasilkan urutan output baru yang memiliki properti yang sama dengan Input Sequence dalam korpus input. Kasus penggunaannya adalah pembuatan teks (diberikan 100 baris blog, buat baris blog berikutnya) dan pembuatan musik (diberikan contoh musik, buat karya musik baru). Lihat contoh dibawah ini:

- Input Sequence: [3, 5,8,12], [4,6,9,13]
- Output: [5,7,10,14]

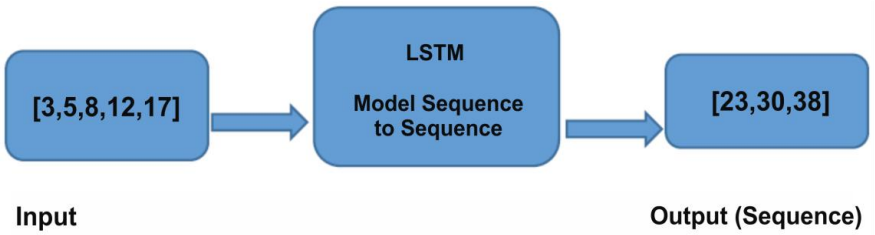


Gambar 9-7. Contoh Model Pengurutan

Prediksi Urutan-ke-Urutan

Prediksi urutan-ke-urutan adalah saat Anda memprediksi urutan berikutnya untuk urutan tertentu. Kasus penggunaannya adalah peringkasan dokumen dan prediksi deret waktu multilangkah (memprediksi urutan angka). Lihat contoh berikut:

- Input Sequence: [3, 5,8,12,17]
- Output: [23,30,38]



Gambar 9-8. Contoh Model Urutan ke Urutan

Seperti disebutkan, LSTM digunakan untuk prediksi deret waktu dalam bisnis. Mari kita lihat model LSTM. Asumsikan bahwa file CSV diberikan di mana kolom pertama adalah stempel waktu dan kolom kedua adalah nilai. Ini dapat mewakili data sensor (IoT). Mengingat data deret waktu, Anda harus memprediksi nilai untuk masa depan.

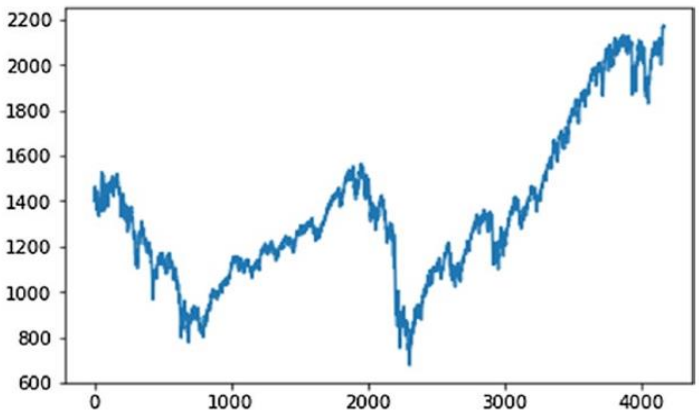
9.5 PREDIKSI DERET WAKTU DENGAN MODEL LSTM

Berikut adalah contoh lengkap prediksi deret waktu dengan LSTM:

```
# Simple LSTM for a time series data
import numpy as np
import matplotlib.pyplot as plt
from pandas import read_csv import math
```

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import pylab
# convert an array of values into a timeseries data
def create_timeseries(series, ts_lag=1):
        dataX = []
        dataY = []
        n_rows = len(series)-ts_lag
        for i in range(n_rows-1):
                a = series[i:(i+ts_lag), 0]
                dataX.append(a)
                dataY.append(series[i + ts_lag, 0])
                X, Y = np.array(dataX), np.array(dataY)
                return X, Y
        # fix random seed for reproducibility
        np.random.seed(230)
        # load dataset
        dataframe = read_csv('sp500.csv', usecols=[0])
        plt.plot(dataframe)
        plt.show()
```

Gambar 9-9 menunjukkan plot data.



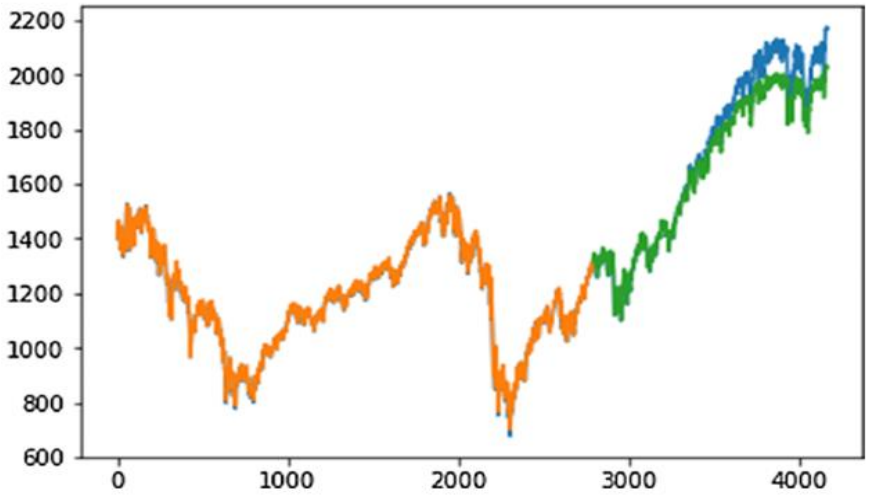
Gambar 9-9. Plot datanya

Berikut beberapa kode lagi:

```
        # Changing datatype to float32 type
        series = dataframe.values.astype('float32')
        # Normalize the dataset
        scaler = StandardScaler()
        series = scaler.fit_transform(series)
        # split the datasets into train and test sets
        train_size = int(len(series) * 0.75)
        test_size = len(series) - train_size
        train, test = series[0:train_size,:],
        series[train_ size:len(series),:]
```

```
# reshape the train and test dataset into X=t and Y=t+1
ts_lag = 1 trainX, trainY = create_timeseries(train, ts_lag)
testX, testY = create_timeseries(test, ts_lag)
# reshape input data to be [samples, time steps, features]
trainX = np.reshape(trainX, (trainX.shape[0], 1, trainX.
shape[1]))
testX = np.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
# Define the LSTM model
model = Sequential()
model.add(LSTM(10, input_shape=(1, ts_lag)))
model.add(Dense(1))
model.compile(loss='mean_squared_logarithmic_error',
optimizer='adagrad')
# fit the model
model.fit(trainX, trainY, epochs=500, batch_size=30)
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# rescale predicted values
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# calculate root mean squared error
trainScore = math.sqrt(mean_squared_error(trainY[0],
trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0],
testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# plot baseline and predictions
pylab.plot(trainPredictPlot)
pylab.plot(testPredictPlot)
pylab.show()
```

Pada Gambar 9-6, Anda dapat melihat plot deret waktu aktual versus yang diprediksi. Bagian yang berwarna jingga adalah data pelatihan, bagian yang berwarna biru adalah data uji, dan bagian yang berwarna hijau adalah hasil prediksi.



Gambar 9-10. Plot deret waktu aktual versus yang diprediksi

Sejauh ini, kita telah mempelajari konsep RNN, LSTM dan prediksi deret waktu dengan model LSTM. LSTM telah digunakan dalam klasifikasi teks. Kami menggunakan LSTM (vanilla LSTM atau LSTM dua arah) untuk membangun pengklasifikasi teks. Pertama, korpus teks diubah menjadi angka dengan menggunakan penyisipan kata (semantik) seperti word2vec atau glove. Kemudian, klasifikasi urutan dilakukan melalui LSTM. Pendekatan ini menawarkan akurasi yang jauh lebih tinggi daripada kumpulan kata-kata biasa atau tf-idf diikuti oleh pengklasifikasi ML seperti SVM, Random Forest. Di bab 11, kita dapat melihat bagaimana LSTM dapat digunakan untuk pengklasifikasi.

BAB 10
SPEECH TO TEXT (UCAPAN KE TEKS) DAN SEBALIKNYA

Dalam bab ini, Anda akan belajar tentang pentingnya konversi ucapan-ke-teks dan teks-ke-suara. Anda juga akan belajar tentang fungsi dan komponen yang diperlukan untuk melakukan konversi jenis ini. Secara khusus, saya akan membahas yang berikut:

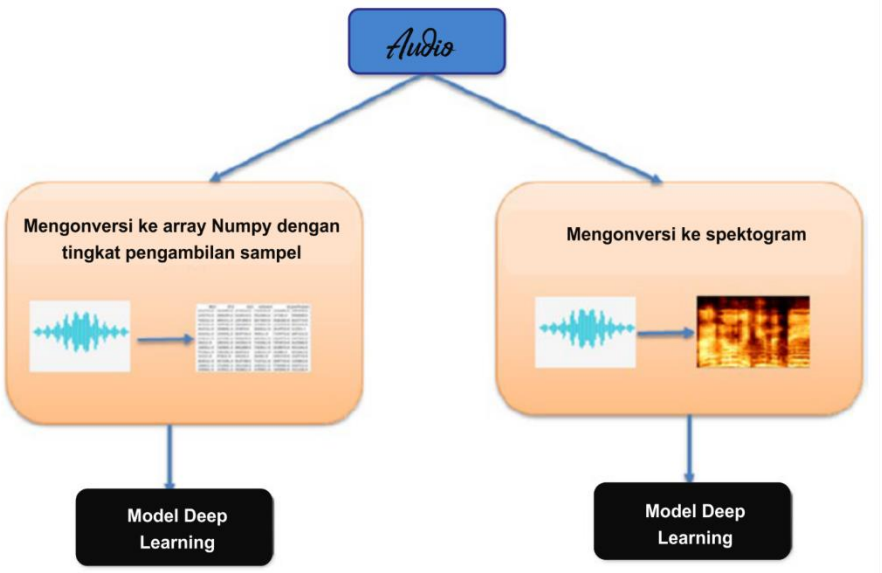
- Mengapa Anda ingin mengubah ucapan menjadi teks
- Ucapan sebagai data
- Fitur ucapan yang memetakan ucapan ke matriks
- Spektogram, yang memetakan ucapan ke gambar
- Membangun pengklasifikasi untuk pengenalan suara melalui fitur *mel-frequency cepstral coefficient* (MFCC)
- Membangun pengklasifikasi untuk pengenalan suara melalui spektogram
- Pendekatan sumber terbuka untuk pengenalan suara
- Penyedia layanan kognitif populer
- Masa depan Speech teks

10.1 KONVERSI UCAPAN-KE-TEKS / *SPEECH TO TEXT*

Konversi ucapan-ke-teks, dalam istilah awam, berarti aplikasi mengenali kata-kata yang diucapkan oleh seseorang dan mengubah suara menjadi teks tertulis. Ada banyak alasan Anda ingin menggunakan konversi *Speech-to-Text*.

- Orang buta atau cacat fisik dapat mengontrol perangkat yang berbeda hanya dengan menggunakan suara.
- Anda dapat menyimpan catatan rapat dan acara lainnya dengan mengubah percakapan lisan menjadi transkrip teks.
- Anda dapat mengonversi audio dalam file video dan audio untuk mendapatkan subtitle dari kata-kata yang diucapkan.
- Anda dapat menerjemahkan kata ke dalam bahasa lain dengan berbicara ke perangkat dalam satu bahasa dan mengubah teks menjadi ucapan dalam bahasa lain.

Ucapan sebagai Data



Gambar 10-1. Ucapan sebagai data

Langkah pertama untuk membuat sistem pengenalan suara otomatis adalah mendapatkan fitur-fiturnya. Dengan kata lain, Anda mengidentifikasi komponen gelombang audio yang berguna untuk mengenali konten linguistik dan menghapus semua fitur tidak berguna lainnya yang hanya suara latar.

Ucapan setiap orang disaring oleh bentuk saluran vokal mereka dan juga oleh lidah dan gigi. Suara apa yang keluar tergantung pada bentuk ini. Untuk mengidentifikasi fonem yang dihasilkan secara akurat, Anda perlu menentukan bentuk ini secara akurat. Dapat dikatakan bahwa bentuk dari saluran vokal memanifestasikan dirinya untuk membentuk selubung spektrum daya waktu pendek. Adalah tugas MFCC untuk mewakili amplop ini secara akurat. Ucapan juga dapat direpresentasikan sebagai data dengan mengubahnya menjadi spektogram (Gambar 10-1).

MFCC banyak digunakan dalam pengenalan suara dan pembicara otomatis. Skala mel menghubungkan frekuensi yang dirasakan, atau nada, dari nada murni dengan frekuensi terukur yang sebenarnya. Anda dapat mengonversi audio dalam skala frekuensi ke skala mel menggunakan rumus berikut:

$$M(f) = 1125\ In(1+f/700)$$

Untuk mengubahnya kembali menjadi frekuensi, gunakan rumus berikut:

$$M^{-1}(m) = 700(exp(m/1125)-1)$$

Berikut adalah fungsi untuk mengekstrak fitur MFCC dengan Python:

```
def mfcc (signal,samplerate=16000,winlen=0.025,winstep=0.01, numcep=13,
nfilt=26,nfft=512,lowfreq=0,highfreq=None, preemph=0.97,
ceplifter=22,appendEnergy=True)
```

Ini adalah parameter yang digunakan:

- signal: Ini adalah sinyal yang Anda perlukan untuk menghitung fitur MFCC. Itu harus berupa array N*1 (baca file WAV).
- samplerate: Ini adalah sample rate sinyal di mana Anda bekerja.
- winlen: Ini adalah panjang jendela analisis dalam detik. Secara default adalah 0,025 detik.
- winstep: Ini adalah langkah jendela yang berurutan. Secara default adalah 0,01 detik.
- numcep: Ini adalah jumlah ceptrum yang harus dikembalikan oleh fungsi. Secara default adalah 13.
- nfilt: Ini adalah jumlah filter di bank filter. Secara default adalah 26.
- nfft: Ini adalah ukuran dari *Fourier Fast Transformation* (FFT). Secara default adalah 512.
- lowfreq: Ini adalah tepi pita terendah, dalam hertz. Secara default adalah 0.
- highfreq: Ini adalah tepi pita tertinggi, dalam hertz. Secara default, ini adalah laju sampel dibagi 2.
- preemph: Ini menerapkan filter preemphasis dengan preemph sebagai koefisien. 0 berarti tidak ada filter. Secara default adalah 0,97.
- ceplifter: Ini menerapkan lifter ke koefisien cepstral akhir. 0 berarti tidak ada pengangkat. Secara default adalah 22.
- appendEnergy: Koefisien cepstral ke-nol diganti dengan log energi bingkai total, jika disetel ke true.

Fungsi ini mengembalikan array Numpy yang berisi fitur. Setiap baris berisi satu vektor fitur.

10.2 *SPEKTOGRAM:* MEMETAKAN UCAPAN KE SEBUAH GAMBAR

Spektogram adalah representasi fotografi atau elektronik dari spektrum. Idenya adalah untuk mengubah file audio menjadi gambar dan meneruskan gambar ke model *Deep learning* seperti CNN dan LSTM untuk analisis dan klasifikasi.

Spektogram dihitung sebagai urutan FFT segmen data berjendela. Format umum adalah grafik dengan dua dimensi geometris; satu sumbu mewakili waktu, dan sumbu lain mewakili frekuensi. Dimensi ketiga menggunakan warna atau ukuran titik untuk menunjukkan amplitudo frekuensi tertentu pada waktu tertentu. *Spektogram* biasanya dibuat dengan salah satu dari dua cara. Mereka dapat didekati sebagai bank filter yang dihasilkan dari serangkaian filter *band-pass.* Atau, dalam Python, ada fungsi langsung yang memetakan audio ke *spektogram*.

Membangun Pengklasifikasi untuk Pengenalan Ucapan Melalui Fitur MFCC

Untuk membangun pengklasifikasi untuk pengenalan suara, Anda harus menginstal paket Python python_speech_features. Anda dapat menggunakan perintah pip install python_speech_features untuk menginstal paket ini.

Fungsi mfcc membuat matriks fitur untuk file audio. Untuk membuat pengklasifikasi yang mengenali suara orang yang berbeda, Anda perlu mengumpulkan data ucapan mereka dalam format WAV. Kemudian Anda mengubah semua file audio menjadi matriks menggunakan fungsi mfcc. Kode untuk mengekstrak fitur dari file WAV ditampilkan di sini:

```
from python_speech_features import mfcc
from python_speech_features import delta
from python_speech_features import logfbank
import scipy.io.wavfile as wav

(samplerate,signal) = wav.read("audio.wav")
mfccfeatures = mfcc(signal,samplerate)
dmfccfeature = delta(mfccfeatures, 2)
fbankfeature = logfbank(signal,samplerate)

print(fbankfeature)
```

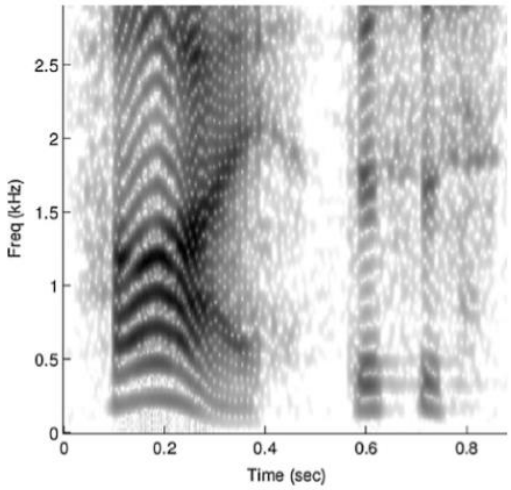
Jika Anda menjalankan kode sebelumnya, Anda akan mendapatkan output dalam bentuk berikut:

```
[[ 7.66608682  7.04137131  7.30715423 ...,  9.43362359  9.11932984   9.93454603]
 [ 4.9474559   4.97057377  6.90352236 ...,  8.6771281   8.86454547   9.7975147 ]
 [ 7.4795622   6.63821063  5.98854983 ...,  8.78622734  8.805521   9.83712966]
 ...,
 [ 7.8886269   6.57456605  6.47895433 ...,  8.62870034  8.79965464   9.67997298]
 [ 5.73028657  4.87985847  6.64977329 ...,  8.64089442  8.62887745   9.90470194]
 [ 8.8449656   6.67098127  7.09752316 ...,  8.84914694  8.97807983   9.45123015]]
```

Di sini, setiap baris mewakili satu vektor fitur. Kumpulkan sebanyak mungkin rekaman suara seseorang dan tambahkan matriks fitur dari setiap file audio dalam matriks ini. Ini akan bertindak sebagai kumpulan data pelatihan Anda. Ulangi langkah yang sama dengan semua kelas lainnya. Setelah kumpulan data disiapkan, Anda dapat memasukkan data ini ke dalam model Deep learning (yang digunakan untuk klasifikasi) untuk mengklasifikasikan suara orang yang berbeda.

Membangun Pengklasifikasi untuk Pengenalan Ucapan Melalui Spektrogram

Menggunakan pendekatan spektogram mengonversi semua file audio menjadi gambar (Gambar 10-2), jadi yang harus Anda lakukan hanyalah mengonversi semua file suara dalam data pelatihan menjadi gambar dan memasukkan gambar tersebut ke model Deep learning seperti yang Anda lakukan di sebuah CNN.



Gambar 10-2. Spektogram sampel *Speech*

Berikut adalah kode Python untuk mengonversi file audio menjadi spektogram:

```
import matplotlib.pyplot as plt
from scipy import signal
from scipy.io import wavfile

sample_rate, samples = wavfile.read('monoAudioFile.wav')
frequencies, times, spectogram = signal.spectrogram(samples, sample_rate)

plt.imshow(spectogram)
plt.ylabel('Freq(kHz)')
plt.xlabel('Time (sec)')
plt.show()
```

10.3 PENDEKATAN SUMBER TERBUKA

Ada paket *open source* yang tersedia untuk Python yang melakukan konversi ucapan-ke-teks dan teks-ke-suara. Berikut adalah beberapa API konversi *Speech-to-text open source*:

- PocketSphinx
- Google Speech
- Google Cloud Speech
- Wit.ai
- Houndify
- IBM Speech to Text API
- Microsoft Bing Speech

Setelah menggunakan semua ini, saya dapat mengatakan bahwa mereka bekerja dengan cukup baik; aksen Amerika sangat jelas. Jika Anda tertarik untuk mengevaluasi keakuratan konversi, Anda memerlukan satu metrik: tingkat kesalahan kata/*word error rate* (WER). Pada bagian selanjutnya, saya akan membahas masing-masing API yang disebutkan sebelumnya.

Contoh Menggunakan Setiap API

Mari kita lihat setiap API.

Menggunakan PocketSphinx

PocketSphinx adalah API open source yang digunakan untuk konversi ucapan-ke-teks. Ini adalah mesin pengenalan suara yang ringan, khusus disetel untuk perangkat genggam dan seluler, meskipun bekerja sama baiknya di desktop. Cukup gunakan perintah pip install PocketSphinx untuk menginstal paket.

import speech_recognition as sr

```
from os import path
AUDIO_FILE = "MyAudioFile.wav"
r = sr.Recognizer()
with sr.AudioFile(AUDIO_FILE) as source:
audio = r.record(source)
try:
print("Sphinx thinks you said " + r.recognize_sphinx(audio))
except sr.UnknownValueError:
print("Sphinx could not understand audio")
except sr.RequestError as e:
print("Sphinx error; {0}".format(e))
```

Menggunakan Google Speech API

Google menyediakan Speech API sendiri yang dapat diimplementasikan dalam kode Python dan dapat digunakan untuk membuat aplikasi yang berbeda.

```
# recognize speech using Google Speech Recognition
try:
    print("Google Speech Recognition thinks you said " +
    r.recognize_google(audio))
    except sr.UnknownValueError:
    print("Google Speech Recognition could not understand audio")
    except sr.RequestError as e:
    print("Could not request results from Google Speech
    Recognition service;{0}".format(e))
```

Menggunakan Google Cloud Speech API

Anda juga dapat menggunakan Google Cloud Speech API untuk konversi. Buat akun di Google Cloud dan salin kredensialnya.

```
GOOGLE_CLOUD_SPEECH_CREDENTIALS = r"INSERT THE CONTENTS OF THE
GOOGLE CLOUD SPEECH JSON CREDENTIALS FILE HERE" try:
    print("Google Cloud Speech thinks you said " +
    r.recognize_google_cloud(audio, credentials_json=GOOGLE_
    CLOUD_SPEECH_CREDENTIALS))
    except sr.UnknownValueError:
    print("Google Cloud Speech could not understand audio")
    except sr.RequestError as e:
    print("Could not request results from Google Cloud Speech
    service; {0}".format(e))
```

Menggunakan Wit.ai API

Wit.ai API memungkinkan Anda membuat konverter ucapan-ke-teks. Anda perlu membuat akun dan kemudian membuat proyek. Salin kunci Wit.ai Anda dan mulai coding.

```
#recognize speech using Wit.ai
WIT_AI_KEY = "INSERT WIT.AI API KEY HERE" # Wit.ai keys are
32-character uppercase alphanumeric strings
try:
    print("Wit.ai thinks you said " + r.recognize_wit(audio,
    key=WIT_AI_KEY))
    except sr.UnknownValueError:
    print("Wit.ai could not understand audio")
    except sr.RequestError as e:
```

```
        print("Could not request
        results from Wit.ai service;  {0}". format(e))
```

Menggunakan Houndify API

Mirip dengan API sebelumnya, Anda perlu membuat akun di Houndify dan mendapatkan ID klien dan kunci Anda. Ini memungkinkan Anda membuat aplikasi yang merespons suara.

```
# recognize speech using Houndify HOUNDIFY_CLIENT_ID = "INSERT
HOUNDIFY CLIENT ID HERE"
# Houndify client IDs are Base64-encoded strings
HOUNDIFY_CLIENT_KEY = "INSERT HOUNDIFY CLIENT KEY HERE"
# Houndify client keys are Base64-encoded strings
try:
        print("Houndify thinks you said " + r.recognize_
        houndify(audio, client_id=HOUNDIFY_CLIENT_ID, client_
        key=HOUNDIFY_CLIENT_KEY))
        except sr.UnknownValueError:
        print("Houndify could not understand audio")
        except sr.RequestError as e:
        print("Could not request results from Houndify service;
        {0}".format(e))
```

Menggunakan IBM Speech to Text API

IBM Speech to Text API memungkinkan Anda untuk menambahkan kemampuan pengenalan suara IBM ke aplikasi Anda. Masuk ke cloud IBM dan mulai proyek Anda untuk mendapatkan nama pengguna dan kata sandi IBM.

```
# IBM Speech to Text
# recognize speech using IBM Speech to Text
IBM_USERNAME = "INSERT IBM SPEECH TO TEXT USERNAME HERE"
# IBM Speech to Text usernames are strings of the form  XXXXXXXX-XXXX- XXXX-XXXX-
XXXXXXXXXXXX
IBM_PASSWORD = "INSERT IBM SPEECH TO TEXT PASSWORD HERE" # IBM
Speech to Text passwords are mixed-case alphanumeric strings
try:
        print("IBM Speech to Text thinks you said " + r.recognize_
        ibm(audio, username=IBM_USERNAME, password=IBM_PASSWORD))
        except sr.UnknownValueError:
        print("IBM Speech to Text could not understand audio")
        except sr.RequestError as e:
        print("Could not request results from IBM Speech to Text
        service; {0}".format(e))
```

Menggunakan Bing Voice Recognition API

API ini mengenali audio yang berasal dari mikrofon secara real time. Buat akun di Bing.com dan dapatkan kunci API Pengenalan Suara Bing.

```
# recognize speech using Microsoft Bing Voice Recognition
BING_KEY = "INSERT BING API KEY HERE" # Microsoft Bing Voice
Recognition API key is 32-character lowercase hexadecimal strings
try:
        print("Microsoft Bing Voice Recognition thinks you said " + r.recognize_bing(audio,
        key=BING_KEY))
        except sr.UnknownValueError:
```

```
print("Microsoft Bing Voice Recognition could not understand audio")
except sr.RequestError as e:
print("Could not request results from Microsoft Bing Voice Recognition service;
{0}".format(e))
```

Setelah Anda mengubah ucapan menjadi teks, Anda tidak dapat mengharapkan akurasi 100 persen. Untuk mengukur akurasi, Anda dapat menggunakan WER.

10.4 Konversi Text-to-Speech

Bagian bab ini berfokus pada mengonversi teks tertulis ke file audio.

Menggunakan pyttsx

Menggunakan paket Python yang disebut pyttsx, Anda dapat mengonversi teks menjadi audio.

Lakukan pip install pyttsx. Jika Anda menggunakan python 3.6 kemudian klik pip3 install pyttsx3.

```
import pyttsx
engine = pyttsx.init()
engine.say("Your Message")
engine.runAndWait()
```

Menggunakan SAPI

Anda juga dapat menggunakan SAPI untuk melakukan konversi text-to-speech dengan Python.

```
from win32com.client import constants, Dispatch
Msg = "Hi this is a test"
speaker = Dispatch("SAPI.SpVoice")
#Create SAPI SpVoice Object
speaker.Speak(Msg)
#Process TTS del speaker
```

Menggunakan SpeechLib

Anda dapat mengambil input dari file teks dan mengubahnya menjadi audio menggunakan SpeechLib, seperti yang ditunjukkan di sini:

```
from comtypes.client import CreateObject
engine = CreateObject("SAPI.SpVoice")
stream = CreateObject("SAPI.SpFileStream")
from comtypes.gen import SpeechLib
infile = "SHIVA.txt"
outfile = "SHIVA-audio.wav"
stream.Open(outfile, SpeechLib.SSFMCreateForWrite)
engine.AudioOutputStream = stream f = open(infile, 'r')
theText = f.read()
f.close()
engine.speak(theText)
stream.Close()
```

Sering kali, Anda harus mengedit audio sehingga Anda dapat menghapus suara dari file audio. Bagian selanjutnya menunjukkan caranya.

Kode Pemotongan Audio

Buat file CSV audio yang berisi nilai yang dipisahkan koma dari detail audio dan lakukan hal berikut menggunakan Python:

```
import wave
```

```
import sys
import os
import csv origAudio = wave.open('Howard.wav', 'r') #change path
frameRate = origAudio.getframerate()
nChannels = origAudio.getnchannels()
sampWidth = origAudio.getsampwidth()
nFrames   = origAudio.getnframes()
filename =  'result1.csv' #change path
exampleFile = open(filename)
exampleReader = csv.reader(exampleFile)
exampleData = list(exampleReader)
count = 0
for data in exampleData:
    #for selections in data:
        print('Selections ', data[4], data[5])
        count += 1
        if data[4] == 'startTime' and data[5] == 'endTime':
        print('Start time')
        else:
                start = float(data[4])       end = float(data[5])
                origAudio.setpos(start*frameRate)
                chunkData = origAudio.readframes(int((end- start)*frameRate))
                outputFilePath = 'C:/Users/Navin/outputFile{0}.wav'. format(count) # change
                path
                chunkAudio = wave.open(outputFilePath, 'w')
                chunkAudio.setnchannels(nChannels)
                chunkAudio.setsampwidth(sampWidth)
                chunkAudio.setframerate(frameRate)
                chunkAudio.writeframes(chunkData)
                chunkAudio.close()
```

10.5 PENYEDIA LAYANAN KOGNITIF

Mari kita lihat beberapa penyedia layanan kognitif yang membantu pemrosesan ucapan.

Microsoft Azure

Microsoft Azure menyediakan yang berikut ini:

- *Custom Speech Service:* Ini mengatasi hambatan pengenalan suara seperti gaya bicara, kosa kata, dan kebisingan latar belakang.
- *Translator Speech API:* Ini memungkinkan terjemahan ucapan waktu nyata.
- *Speaker Identification API*: Ini dapat mengidentifikasi pembicara berdasarkan sampel ucapan setiap pembicara dalam data audio yang diberikan.
- *Bing Speech API:* Ini mengubah audio menjadi teks, memahami maksud, dan mengonversi teks kembali menjadi ucapan untuk responsivitas alami.

Kognitif Amazon Service

Amazon Cognitive Services menyediakan Amazon Polly, layanan yang mengubah teks menjadi ucapan. Amazon Polly memungkinkan Anda membuat aplikasi yang berbicara, memungkinkan Anda membangun kategori produk yang sepenuhnya baru dengan kemampuan bicara.

- 47 suara dan 24 bahasa dapat digunakan, dan tersedia opsi bahasa Inggris India.
- Nada seperti berbisik, marah, dan sebagainya, dapat ditambahkan ke bagian tertentu dari Speech menggunakan efek Amazon.

Anda dapat menginstruksikan sistem cara mengucapkan frasa atau kata tertentu dengan cara yang berbeda. Misalnya, "W3C" diucapkan sebagai World Wide Web Consortium, tetapi Anda dapat mengubahnya untuk mengucapkan akronim saja. Anda juga dapat memberikan teks input dalam format SSML.

IBM Watson Service

Ada dua layanan dari IBM Watson.

- *Speech to text*: Inggris AS, Spanyol, dan Jepang
- *Text to speech*: Inggris AS, Inggris Inggris Raya, Spanyol, Prancis, Italia, dan Jerman

Masa Depan Speech Analytics

Teknologi pengenalan suara telah membuat kemajuan besar. Setiap tahun, sekitar 10 hingga 15 persen lebih akurat dari tahun sebelumnya. Di masa depan, ini akan menyediakan antarmuka paling interaktif untuk komputer. Ada banyak aplikasi yang akan segera Anda saksikan di pasar, termasuk buku interaktif, kontrol robot, dan antarmuka mobil self-driving. Data ucapan menawarkan beberapa kemungkinan baru yang menarik karena ini adalah masa depan industri.

Speech intelligence (Kecerdasan bicara) memungkinkan orang untuk mengirim pesan, menerima atau memberi perintah, mengajukan keluhan, dan melakukan pekerjaan apa pun di mana mereka biasa mengetik secara manual. Ini menawarkan pengalaman pelanggan yang luar biasa dan mungkin itulah sebabnya semua departemen dan bisnis yang menghadapi pelanggan cenderung menggunakan aplikasi ucapan dengan sangat berat. Saya dapat melihat masa depan yang cerah bagi para pengembang aplikasi pengucapan.

BAB 11
MENGEMBANGKAN CHATBOTS

Sistem kecerdasan buatan yang bertindak sebagai antarmuka untuk interaksi manusia dan mesin melalui teks atau suara disebut chatbots. Interaksi dengan chatbots dapat berupa langsung atau kompleks. Contoh interaksi langsung dapat berupa menanyakan berita terkini. Interaksi dapat menjadi lebih kompleks ketika berhubungan dengan pemecahan masalah, katakanlah, ponsel Android Anda. Istilah chatbots telah mendapatkan popularitas luar biasa dalam satu tahun terakhir dan telah berkembang menjadi platform yang paling disukai untuk interaksi dan keterlibatan pengguna. Bot, bentuk lanjutan dari chatbot, membantu mengotomatiskan tugas yang "dilakukan pengguna".

Bab tentang chatbots ini akan berfungsi sebagai panduan menyeluruh tentang apa, bagaimana, di mana, kapan, dan mengapa chatbots! Secara khusus, saya akan membahas yang berikut:

- Mengapa Anda ingin menggunakan chatbots
- Desain dan fungsi chatbots
- Langkah-langkah membuat chatbot
- Pengembangan chatbot menggunakan API
- Praktik terbaik chatbot

11.1 MENGAPA CHATBOT?

Penting bagi chatbot untuk memahami informasi apa yang dicari pengguna, yang disebut maksud. Misalkan pengguna ingin mengetahui restoran vegetarian terdekat; pengguna dapat mengajukan pertanyaan itu dalam banyak cara yang memungkinkan. Sebuah chatbot (khususnya intent classifier di dalam chatbot) harus dapat memahami intent tersebut karena pengguna ingin mendapatkan jawaban yang benar. Padahal, untuk memberikan jawaban yang tepat, chatbot harus bisa memahami konteks, maksud, entitas, dan sentimen.

Chatbot bahkan harus memperhitungkan apa pun yang dibahas dalam sesi. Misalnya, pengguna mungkin menanyakan pertanyaan *"Berapa harga ayam biryani di sana?"* Meskipun pengguna telah meminta harga, mesin obrolan dapat salah paham dan menganggap pengguna sedang mencari restoran. Jadi, sebagai tanggapan, chatbot dapat memberikan nama restoran.

11.2 DESAIN DAN FUNGSI CHATBOT

Chatbot merangsang percakapan cerdas dengan manusia melalui aplikasi AI. Antarmuka di mana percakapan berlangsung difasilitasi baik melalui teks lisan atau tertulis. Facebook Messenger, Slack, dan Telegram menggunakan platform perpesanan chatbot. Mereka melayani banyak tujuan, termasuk memesan produk secara online, berinvestasi dan mengelola keuangan, dan sebagainya. Aspek penting dari chatbots adalah mereka memungkinkan percakapan kontekstual.

Chatbots berkomunikasi dengan pengguna dengan cara yang mirip dengan bagaimana manusia berkomunikasi dalam kehidupan sehari-hari mereka. Meskipun chatbot memungkinkan untuk berkomunikasi secara kontekstual, mereka masih memiliki jalan panjang untuk berkomunikasi secara kontekstual dengan segala sesuatu dan apa saja. Tetapi antarmuka obrolan menggunakan bahasa untuk menghubungkan mesin ke manusia,

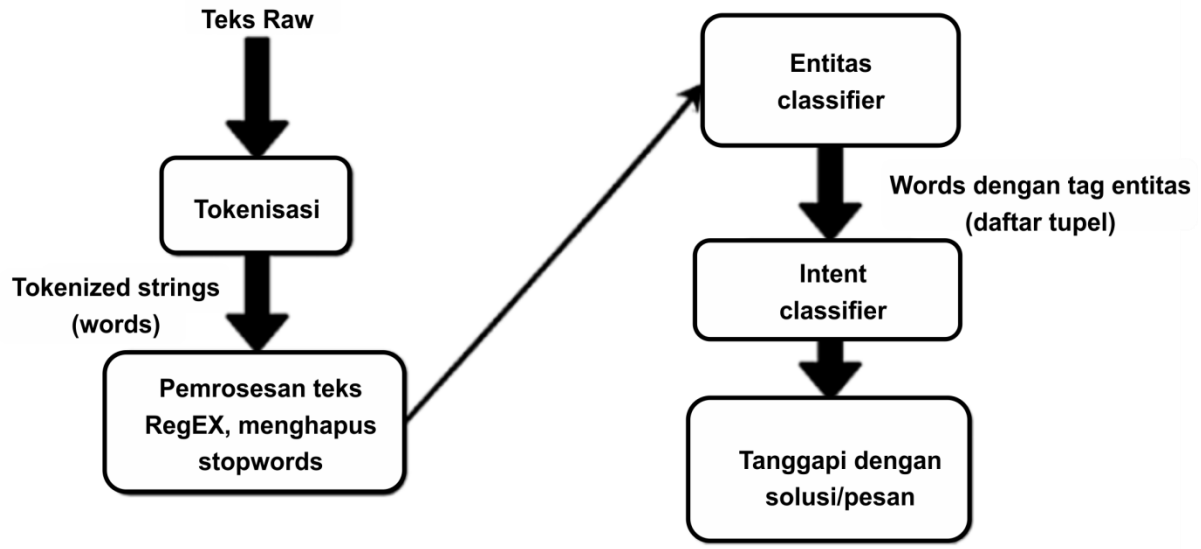
membantu orang menyelesaikan sesuatu dengan cara yang nyaman dengan memberikan informasi secara kontekstual.

Selain itu, chatbots mendefinisikan ulang cara bisnis dijalankan. Dari menjangkau konsumen hingga menyambut mereka di ekosistem bisnis hingga memberikan informasi kepada konsumen tentang berbagai produk dan fitur mereka, chatbots membantu semuanya. Mereka muncul sebagai cara paling nyaman untuk berurusan dengan konsumen secara tepat waktu dan memuaskan.

11.3 LANGKAH-LANGKAH UNTUK MEMBUAT CHATBOT

Chatbot dibangun untuk berkomunikasi dengan pengguna dan memberi mereka perasaan bahwa mereka berkomunikasi dengan manusia dan bukan bot. Namun ketika pengguna memberikan input, biasanya mereka tidak memberikan input dengan cara yang benar. Dengan kata lain, mereka mungkin memasukkan tanda baca yang tidak perlu, atau mungkin ada cara berbeda untuk menanyakan pertanyaan yang sama.

Misalnya, untuk *"Restoran di dekat saya?"* pengguna dapat memasukkan *"Restoran di samping saya?"* atau *"Temukan restoran terdekat".* Oleh karena itu, Anda perlu melakukan praproses data agar mesin chatbot dapat dengan mudah memahaminya. Gambar 11-1 menunjukkan prosesnya, yang dirinci di bagian berikut.



Gambar 11-1. Diagram alur untuk menunjukkan bagaimana mesin chatbot memproses string input dan memberikan balasan yang valid.

11.4 TEKS DAN PESAN PRAPEMROSESAN

Prapemrosesan teks dan pesan mencakup beberapa langkah, dibahas selanjutnya.

Tokenisasi

Memotong kalimat menjadi satu kata (disebut token) disebut tokenization. Dalam Python, umumnya string diberi token dan disimpan dalam daftar.

Misalnya, kalimat "Kecerdasan buatan adalah tentang penerapan matematika" menjadi sebagai berikut:

["Artificial", "intelligence", "is", "all", "about", "applying", "mathematics"]

Berikut adalah contoh kodenya:

```
from nltk.tokenize import TreebankWordTokenizer
I = "Artificial intelligence is all about applying mathematics"
token = TreebankWordTokenizer().tokenize(I)
print(token)
```

Menghapus Tanda Baca

Anda juga dapat menghapus tanda baca yang tidak perlu dalam kalimat.
Misalnya, kalimat "Bisakah saya mendapatkan daftar restoran, yang memberikan pengiriman ke rumah." menjadi sebagai berikut:
"Bisakah saya mendapatkan daftar restoran yang memberikan pengiriman ke rumah."
Berikut adalah contoh kodenya:

```
from nltk.tokenize import TreebankWordTokenizer
from nltk.corpus import stopwords
l = " Kecerdasan buatan adalah tentang menerapkan matematika!"
token = TreebankWordTokenizer().tokenize(l)
output = []
output = [k for k in token if k.isalpha()]
print(output)
```

Menghapus Stop Words

Stopwords adalah kata-kata yang ada dalam kalimat yang tidak membuat banyak perbedaan jika dihilangkan. Meskipun format kalimatnya berubah, ini banyak membantu dalam pemahaman bahasa alami/ *natural language understanding* (NLU).

Misalnya, kalimat “Kecerdasan buatan dapat mengubah gaya hidup masyarakat”. menjadi berikut setelah menghapus kata berhenti:

"Kecerdasan buatan mengubah gaya hidup orang."

Berikut adalah contoh kodenya:

```
from nltk.tokenize import TreebankWordTokenizer
from nltk.corpus import stopwords
l = " Kecerdasan buatan adalah tentang menerapkan matematika "
token = TreebankWordTokenizer().tokenize(l)
stop_words = set(stopwords.words('english'))
output= [] for k in token:
    if k not in stop_words:
        output.append(k)
print(output)
```

Kata-kata mana yang dianggap *sebagai stop word* bisa bermacam-macam. Ada beberapa set kata berhenti yang telah ditentukan sebelumnya yang disediakan oleh *Natural Language Toolkit* (NLTK), Google, dan banyak lagi.

Named Entity Recognition

Named Entity Recognition (NER), juga dikenal sebagai identifikasi entitas, adalah tugas mengklasifikasikan entitas dalam teks ke dalam kelas yang telah ditentukan seperti nama negara, nama seseorang, dan sebagainya. Anda juga dapat menentukan kelas Anda sendiri.

Misalnya, menerapkan NER pada kalimat “Pertandingan kriket Indonesia vs Australia hari ini sangat fantastis.” memberi Anda output berikut:
Pertandingan [Hari ini] Waktu [Indonesia] Negara vs [Australia] Negara [kriket] sangat fantastis.

Untuk menjalankan kode NER, Anda perlu mengunduh dan mengimpor paket yang diperlukan, seperti yang disebutkan dalam kode berikut.

Menggunakan Stanford NER

Untuk menjalankan kode, unduh file english.all.3class.dissim.crf.ser.gz dan stanford-ner.jar.

```
from nltk.tag import StanfordNERTagger
```

```
from nltk.tokenize import word_tokenize
StanfordNERTagger("stanford-ner/classifiers/english.all.3class. distsim.crf.ser.gz",
"stanford-ner/stanford-ner.jar")
text = " Agus adalah pendiri Universitas Stekom di Kota Semarang "
text = word_tokenize(text) ner_tags = ner_tagger.tag(text)
print(ner_tags)
```

Menggunakan MITIE NER (Terlatih)

Unduh file ner_model.dat dari MITIE untuk menjalankan kode.

```
from mitie.mitie import *
from nltk.tokenize import word_tokenize
print("loading NER model...")
ner = named_entity_extractor("mitie/MITIE-models/english/
ner_model.dat".encode("utf8"))
text = "Agus adalah pendiri Universitas Stekom di Kota Semarang".
encode("utf-8")
text = word_tokenize(text)
ner_tags = ner.extract_entities(text)
print("\nEntities found:", ner_tags)
for e in ner_tags:
    range = e[0]
    tag = e[1]
    entity_text = " ".join(text[i].decode() for i in range)
    print( str(tag) + " : " + entity_text)
```

Menggunakan MITIE NER (Self-Trained)

Unduh file total_word_feature_extractor.dat dari MITIE (https://github.com/mit-nlp/MITIE) untuk menjalankan kode.

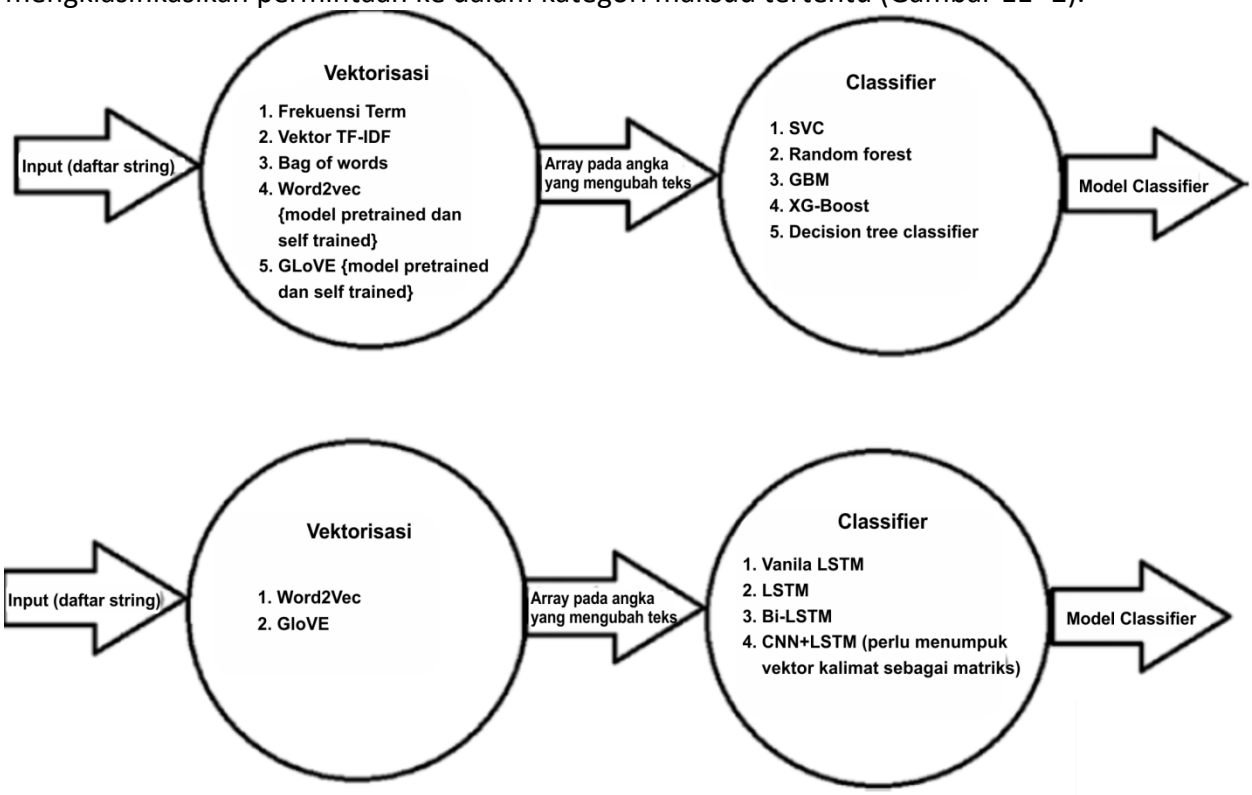
```
from mitie.mitie import *
sample = ner_training_instance([b"Agus", b"adalah", b"pendiri", b"Universitas", b"Stekom",
b"di", b”kota”, b"Semarang", b"."])
sample.add_entity(range(0, 1), "person".encode("utf-8"))
sample.add_entity(range(5, 7), "organization".encode("utf-8"))
sample.add_entity(range(8, 10), "Location".encode("utf-8"))
trainer = ner_trainer("mitie/MITIE-models/english/total_word_
feature_extractor.dat".encode("utf-8"))
trainer.add(sample)
ner = trainer.train()
tokens = [b"Josh", b"adalah", b"seorang", b"rektor", b"di", b"Universitas", b"Stekom", b"."]
entities = ner.extract_entities(tokens)
print ("\nEntities found:", entities)
for e in entities:
    range = e[0]
    tag = e[1]
    entity_text = " ".join(str(tokens[i]) for i in range)
    print ("    " + str(tag) + ": " + entity_text)
```

Klasifikasi Maksud

Klasifikasi maksud adalah langkah dalam NLU di mana Anda mencoba memahami apa yang diinginkan pengguna. Berikut adalah dua contoh input ke chatbot untuk menemukan tempat terdekat:

- "*Saya perlu membeli bahan makanan.":* Tujuannya adalah untuk mencari toko kelontong terdekat.
- "*Saya ingin makan makanan vegetarian.":* Tujuannya adalah untuk mencari restoran terdekat, idealnya yang vegetarian.

Pada dasarnya, Anda perlu memahami apa yang dicari pengguna dan dengan demikian mengklasifikasikan permintaan ke dalam kategori maksud tertentu (Gambar 11- 2).



Gambar 11-2. Aliran umum klasifikasi maksud, dari kalimat ke vektor ke model

Untuk melakukannya, Anda perlu melatih model untuk mengklasifikasikan permintaan ke dalam maksud menggunakan algoritme, mulai dari kalimat, vektor, hingga model.

Word Embedding

Word Embedding adalah teknik mengubah teks menjadi angka. Sulit untuk menerapkan algoritma apa pun dalam teks. Oleh karena itu, Anda harus mengubahnya menjadi angka. Berikut ini adalah berbagai jenis teknik penyisipan kata.

Count Vector

Misalkan Anda memiliki tiga dokumen (D1, D2, dan D3) dan ada N kata unik dalam grup dokumen. Anda membuat matriks (D×N), yang disebut C, yang dikenal sebagai vektor hitungan. Setiap entri matriks adalah frekuensi kata unik dalam dokumen itu.

Mari kita lihat ini menggunakan contoh.

D1: Agus sangat malas.

D2: Tapi dia cerdas.

D3: Dia jarang datang ke kelas.

Di sini, D=3 dan N=12. Kata-kata uniknya adalah sulit, malas, Tapi, untuk, Agus, dia, cerdas, datang, sangat, kelas, dan apa adanya. Oleh karena itu, vektor hitungan, C, adalah sebagai berikut:

	Sangat Rajin	**Sangat Malas**	**Tapi**	**ke**	**Agus**	**Dia**	**Cerdas**	**Datang**	**sangat**	**kelas**	**adalah**

D1	0	1	0	0	1	0	0	0	1	0	1
D2	0	0	1	0	0	1	1	0	0	0	1
D3	1	0	0	1	0	1	0	1	0	1	0

Term Frequency-Inverse Document Frequency (TF-IDF)

Untuk teknik ini, Anda memberi setiap kata dalam kalimat sebuah nomor tergantung pada berapa kali kata itu muncul dalam kalimat itu dan juga tergantung pada dokumennya. Kata-kata yang muncul berkali-kali dalam sebuah kalimat dan tidak terlalu sering dalam sebuah dokumen akan memiliki nilai yang tinggi.

Misalnya, pertimbangkan satu set kalimat:

- "Saya seorang laki-laki."
- "Saya seorang gadis."
- "Kamu tinggal di mana?"

TF-IDF mengubah set fitur untuk kalimat sebelumnya, seperti yang ditunjukkan di sini:

	Saya	Laki-laki	Gadis	dimana	kah	kamu	Tinggal
1.	0.60	0.80	0	0	0	0	0
2.	0.60	0	0.80	0	0	0	0
3.	0	0	0	0.5	0.5	0.5	0.5

Anda dapat mengimpor paket TFIDF dan menggunakannya untuk membuat tabel ini. Sekarang mari kita lihat beberapa contoh kode. Anda dapat menggunakan pengklasifikasi vektor dukungan pada fitur yang diubah TF-IDF dari string permintaan.

```
#import required packages
import pandas as pd
from random import sample
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score
# read csv file
data = pd.read_csv("intent1.csv")
print(data.sample(6))
```

Sebelum melanjutkan dengan kode, berikut adalah contoh kumpulan data:

Deskripsi (Pesan)	**intent_label (Target)**
Restoran non-sayuran yang bagus di dekat saya?	0
Saya mencari rumah sakit	1
Rumah sakit yang bagus untuk operasi jantung	1
Sekolah internasional untuk anak-anak	2
Restoran non-sayuran di sekitar saya	0
Sekolah untuk anak kecil	2

Dalam contoh ini, ini adalah nilai yang akan digunakan:

- 0 berarti mencari restoran.
- 1 berarti mencari rumah sakit.
- 2 berarti mencari sekolah.

Sekarang mari kita bekerja pada kumpulan data.

```
# split dataset into train and test.
X_train, X_test, Y_train, Y_test = train_test_split(data
["Description"], data["intent_label"], test_size=3)
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
# vectorize the input using
tfidf values. tfidf = TfidfVectorizer()
tfidf = tfidf.fit(X_train)
X_train = tfidf.transform(X_train)
X_test = tfidf.transform(X_test)
# penyandian label untuk berbagai kategori maksud
le = LabelEncoder().fit(Y_train)
Y_train = le.transform(Y_train)
Y_test = le.transform(Y_test)
# model lain seperti GBM, Random Forest juga dapat digunakan model = SVC ()
model = model.fit(X_train, Y_train)
p = model.predict(X_test)
# calculate the f1_score. average="micro" karena kita ingin menghitung skor untuk
multiclass.
# Setiap instance (bukan kelas (cari rata-rata makro))
berkontribusi sama terhadap penilaian.
print("f1_score:", f1_score( Y_test, p, average="micro"))
print("accuracy_score:",accuracy_score(Y_test, p))
```

Word2Vec

Ada beberapa metode berbeda untuk mendapatkan vektor kata untuk sebuah kalimat, tetapi teori utama di balik semua teknik ini adalah memberikan kata-kata yang serupa representasi vektor yang serupa. Jadi, kata-kata seperti laki-laki dan laki-laki dan perempuan akan memiliki vektor yang sama. Panjang setiap vektor dapat diatur. Contoh teknik Word2vec termasuk GloVe dan CBOW (n-gram dengan atau tanpa skip gram).

Anda dapat menggunakan Word2vec dengan melatihnya untuk kumpulan data Anda sendiri (jika Anda memiliki cukup data untuk masalah tersebut), atau Anda dapat menggunakan data yang telah dilatih sebelumnya. Word2vec tersedia di Internet. Model pra-pelatihan telah dilatih pada dokumen besar seperti data Wikipedia, tweet, dan sebagainya, dan hampir selalu baik untuk masalah tersebut.

Contoh beberapa teknik yang dapat Anda gunakan untuk melatih pengklasifikasi maksud Anda adalah dengan menggunakan 1D-CNN pada vektor kata dari kata-kata dalam sebuah kalimat, yang ditambahkan dalam daftar untuk setiap kalimat.

```
# import required packages
from gensim.models import Word2Vec
import pandas as pd
import numpy as np
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils.np_utils import to_categorical
from keras.layers import Dense, Input, Flatten
from keras.layers import Conv1D, MaxPooling1D, Embedding, Dropout
from keras.models import Model
from sklearn.preprocessing import LabelEncoder
```

from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, accuracy_score
 # read data
data = pd.read_csv("intent1.csv")
 # split data ke test dan train
X_train, X_test, Y_train, Y_test = train_test_split(data ["Description"], data["intent_label"], test_size=6)
 # penyandian label untuk berbagai kategori maksud
le = LabelEncoder().fit(Y_train)
Y_train = le.transform(Y_train)
Y_test = le.transform(Y_test)
 # dapatkan word_vectors untuk kata-kata dalam training set
X_train = [sent for sent in X_train]
X_test = [sent for sent in X_test]
 # secara default genism.Word2Vec menggunakan CBOW, untuk melatih kata ***vecs.***
Kita juga dapat menggunakan skipgram dengan itu
 # by setting the "sg" attribute to number of skips we want.
 # CBOW dan Skip gram untuk kalimat "Hai Agus bagaimana harimu?" menjadi:
 # Continuos bag of words: 3-grams {"Hi Agus Bagaimana", "Agus Bagaimana dengan", "Bagaimana dengan..." ...}
 # Skip-gram 1-skip 3-grams: {"Hi Agus bagaimana", "Hi Agus bagaimana...", "Hi Bagaimana", "Agus bagaimana...
 # your", ...}
 # See how: "Hi Agus sudah" lewati "bagaimana".
 # Skip-gram 2-skip 3-grams: {"Hi Agus bagaiman", "Hi Agus sudah", "Hi Agus Kamu", "Hi bagaimana
 # your", ...}
 # See how: "Hi Agus Kamu" skips over "Bagaimana Kamu".
 # Itulah pengertian umum dari CBOW dan skip gram. *word_vecs = Word2Vec(X_train)*
print("Word vectors trained")
 # pangkas setiap kalimat hingga maksimal 20 kata. max_sent_len = 20
 # tokenize input strings
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
sequences = tokenizer.texts_to_sequences(X_train)
sequences_test = tokenizer.texts_to_sequences(X_test)
word_index = tokenizer.word_index
vocab_size = len(word_index)
 # kalimat dengan kurang dari 20 kata, akan diisi dengan nol agar panjangnya 20
 # kalimat dengan lebih dari 20 kata, akan dipangkas menjadi 20.
x = pad_sequences(sequences, maxlen=max_sent_len)
X_test = pad_sequences(sequences_test, maxlen=max_sent_len)
 # 100 is the size of wordvec.
embedding_matrix = np.zeros((vocab_size + 1, 100))
 # buat matriks setiap kata dengan word_vectors-nya untuk model CNN.
 # jadi setiap baris matriks akan mewakili satu kata. Akan ada baris untuk setiap kata di
 # training set
Untuk kata,Hi dalam word_index.items():

```
        try:
                embedding_vector = word_vecs[word]
        except:
                embedding_vector = None
                        if embedding_vector is not None:
                                embedding_matrix[i] = embedding_vector
print("Embeddings done")
vocab_size = len(embedding_matrix)
# Model CNN membutuhkan label multikelas untuk diubah menjadi satu pengodean panas.
# yaitu setiap kolom mewakili label, dan akan ditandai satu
untuk label yang sesuai.
y = to_categorical(np.asarray(Y_train))
embedding_layer = Embedding(vocab_size,
                                                        100,
                                                        weights=[embedding_matrix],
                                                        input_length=max_sent_len,
                                                        trainable=True)
sequence_input = Input(shape=(max_sent_len,), dtype='int32')
# susun setiap kata dari kalimat dalam matriks. Jadi setiap matriks mewakili sebuah kalimat.
# Setiap baris dalam matriks adalah kata (Vektor Kata) dari sebuah kalimat.
embedded_sequences = embedding_layer(sequence_input)
# membangun model Konvolusi.
l_cov1 = Conv1D(128, 4, activation='relu')(embedded_sequences)
l_pool1 = MaxPooling1D(4)(l_cov1)
l_flat = Flatten()(l_pool1)
hidden = Dense(100, activation='relu')(l_flat)
preds = Dense(len(y[0]), activation='softmax')(hidden)
model = Model(sequence_input, preds)
model.compile(loss='binary_crossentropy',optimizer='Adam')
print("pemasangan model - jaringan saraf convolutional yang disederhanakan ")
model.summary()
# train the model model.fit(x, y, epochs=10, batch_size=128)
#get scores and predictions.
p = model.predict(X_test)
p = [np.argmax(i) for i in p]
score_cnn = f1_score(Y_test, p, average="micro")
print("accuracy_score:",accuracy_score(Y_test, p))
print("f1_score:", score_cnn)
```

Pemasangan model adalah jaringan saraf convolutional yang disederhanakan, seperti yang ditunjukkan di sini:

Layer (Type)	Output Shape	Param #
input_20 (InputLayer)	(None, 20)	0
embedding_20 (Embedding)	(None, 20, 100)	2800
conv1d_19 (Conv1D)	(None, 17, 128)	51328
max_pooling1d_19 (MaxPooling)	(None, 4, 128)	0
flatten_19 (Flatten)	(None, 512)	0
dense_35 (Dense)	(None, 100)	51300
dense_36 (Dense)	(None, 3)	303

Berikut adalah jumlah parameter:

- *Parameter total: 105.731*
- *Parameter yang dapat dilatih: 105.731*
- *Parameter yang tidak dapat dilatih: 0*

Berikut adalah beberapa fungsi penting Word2vec menggunakan paket Gensim:

- Beginilah cara Anda mengimpor Gensim dan memuat model yang telah dilatih sebelumnya:

```
import genism
# loading model pre-training sebelumnya
model = gensim.models.KeyedVectors.
load_word2vec_format('GoogleNews-vectorsnegative300.bin', binary=True)
```

- Ini adalah model pra-pelatihan dari Google untuk bahasa Inggris, dan berukuran 300 dimensi.
- Berikut adalah cara menemukan vektor kata dari sebuah kata dari model yang telah dilatih sebelumnya:

```
# mendapatkan vektor kata dari sebuah kata
lion = model['lion']
print(len(lion))
```

- Berikut adalah cara mencari indeks kesamaan antara dua kata:

```
# Menghitung indeks kesamaan
print(model.similarity('King', 'Queen'))
```

- Ini adalah cara menemukan yang ganjil dari kumpulan kata:

```
# Pilih yang ganjil
print(model.doesnt_match("Mango Grape Tiger
Banana Strawberry".split()))
```

- Inilah cara menemukan kata-kata yang paling mirip:

```
print(model.most_similar(positive=[Prince,
Girl], negative=[Boy]))
```

Fitur unik Word2vec adalah Anda bisa mendapatkan vektor, dari vektor lain menggunakan operasi vektor. Misalnya, vektor "Pangeran" dikurangi vektor "laki-laki" ditambah vektor "perempuan" akan hampir sama dengan vektor "Putri." Oleh karena itu, ketika Anda menghitung ini, Anda akan mendapatkan vektor "Putri."

Vec ("putri") – Vec("laki-laki") + Vec("putri")≈
Vec("Putri")

Ini hanya sebuah contoh. Kasus ini berlaku dalam banyak kasus lainnya. Ini adalah spesialisasi Word2vec dan berguna dalam memperkirakan kata yang mirip, kata berikutnya, generasi bahasa alami (NLG), dan sebagainya.

Tabel 11-1 menunjukkan model yang telah dilatih sebelumnya dengan parameter lain.

Model File	Jumlah Dimensi	Ukuran Korpus	Ukuran Kosakata	Arsitektur	Ukuran Context Window	Author
Google News	300	100B	3M	Word2Vec	BoW, ~5	Google
Freebase IDs	1000	100B	1.4M	Word2Vec, Skip-gram	BoW, ~10	Google
Freebase names	1000	100B	1.4M	Word2Vec, Skip-gram	BoW, ~10	Google
Wikipedia + Gigaword 5	50	6B	400,000	GloVe	10+10	GloVe
Wikipedia + Gigaword 5	100	6B	400,000	GloVe	10+10	GloVe
Wikipedia + Gigaword 5	200	6B	400,000	GloVe	10+10	GloVe
Wikipedia + Gigaword 5	300	6B	400,000	GloVe	10+10	GloVe
Common Crawl 42B	300	42B	1.9M	GloVe	AdaGrad	GloVe
Common Crawl 840B	300	840B	2.2M	GloVe	AdaGrad	GloVe
Wikipedia dependency	300	-	174,000	Word2Vec	Syntactic Dependencies	Levy & Goldberg
DBPedia vectors (wiki2vec)	1000	-	-	Word2Vec	BoW, 10	Idio

Tabel 11-1. Model Pralatihan Berbeda dengan Parameter Lain

11.5 MEMBANGUN RESPONS

Respons adalah bagian penting lain dari chatbots. Berdasarkan bagaimana chatbot membalas, pengguna mungkin tertarik padanya. Setiap kali chatbot dibuat, satu hal yang harus diingat adalah penggunanya. Anda perlu tahu siapa yang akan menggunakannya dan untuk tujuan apa akan digunakan. Misalnya, chatbot untuk situs web restoran hanya akan ditanya tentang restoran dan makanan.

Jadi, Anda tahu sedikit lebih banyak pertanyaan apa yang akan diajukan. Oleh karena itu, untuk setiap maksud, Anda menyimpan beberapa jawaban yang dapat digunakan setelah mengidentifikasi maksud sehingga pengguna tidak akan mendapatkan jawaban yang sama berulang kali. Anda juga dapat memiliki satu maksud untuk pertanyaan di luar konteks; maksud itu dapat memiliki banyak jawaban, dan memilih secara acak, chatbot dapat membalas.

Misalnya, jika maksudnya adalah "halo" atau “hai” atau “Hi”, Anda dapat memiliki beberapa balasan seperti "Halo! Apa kabar?" dan “Halo! Apa kabarmu?" dan “Hai! Apa yang bisa saya bantu?" Chatbot dapat memilih salah satu secara acak untuk balasan. Dalam kode contoh berikut, Anda mengambil input dari pengguna, tetapi di chatbot asli, maksud ditentukan oleh chatbot itu sendiri berdasarkan pertanyaan apa pun yang diajukan oleh pengguna.

```
import random
intent = input()
```

output = ["Halo! Apa Kabar","Halo! Apakabarmu","Hai! Apa yang bisa saya bantu",”Hi”, "Hai","Haiii","Hello! Bagaimana saya dapat membantu Anda?","Hey! What's up?"]
if(intent == "Hii"):
print(random.choice(output))

11.6 PENGEMBANGAN CHATBOT MENGGUNAKAN API

Membuat chatbot bukanlah tugas yang mudah. Anda membutuhkan perhatian terhadap detail dan pikiran yang tajam untuk membangun chatbot yang dapat dimanfaatkan dengan baik. Ada dua pendekatan untuk membangun chatbot.

- Pendekatan berbasis aturan
- Pendekatan pembelajaran mesin yang membuat sistem belajar sendiri dengan menyederhanakan data

Beberapa chatbot bersifat dasar, sementara yang lain lebih maju dengan otak AI. Chatbot yang dapat memahami bahasa alami dan meresponsnya menggunakan otak AI, dan penggemar teknologi memanfaatkan berbagai sumber seperti Api.ai untuk membuat chatbot yang kaya akan AI. Pemrogram memanfaatkan layanan berikut untuk membuat bot:

- Kerangka kerja bot Microsoft
- Wit.ai
- Api.ai
- Watson IBM

Penggemar pembuatan bot lain dengan keterampilan pemrograman terbatas atau tidak sama sekali memanfaatkan platform pengembangan bot seperti berikut ini untuk membangun chatbot:

- Chatfuel
- Texit.in
- Octane AI
- Motion.ai

Ada berbagai API yang menganalisis teks. Tiga raksasa besar tersebut adalah sebagai berikut:

- Layanan Kognitif Microsoft Azure
- Amazon Lex
- IBM Watson

Layanan Kognitif dari Microsoft Azure

Mari kita mulai dengan Microsoft Azure.

- *Language Understanding Intelligent Service (LUIS):* Ini menyediakan alat sederhana yang memungkinkan Anda untuk membangun model bahasa Anda sendiri (maksud/entitas) yang memungkinkan aplikasi/bot untuk memahami perintah Anda dan bertindak sesuai dengan itu.
- *Text Analytics API*: Ini mengevaluasi sentimen dan topik untuk memahami apa yang diinginkan pengguna.
- *Translator Text API:* Ini secara otomatis mengidentifikasi bahasa dan kemudian menerjemahkannya ke bahasa lain secara real time.
- *Web Language Model API:* Ini memasukkan spasi ke dalam rangkaian kata yang tidak memiliki spasi secara otomatis.
- *Bing Spell Check API*: Ini memungkinkan pengguna untuk memperbaiki kesalahan ejaan; mengenali perbedaan antara nama, nama merek, dan bahasa gaul; dan memahami homofon saat mereka mengetik.

- *Linguistic Analysis API*: Ini memungkinkan Anda untuk mengidentifikasi konsep dan tindakan dalam teks Anda dengan penandaan bagian ucapan dan menemukan frasa dan konsep menggunakan pengurai bahasa alami. Ini sangat berguna untuk menggali umpan balik pelanggan.

Amazon Lex

Amazon Lex adalah layanan untuk membangun antarmuka percakapan ke dalam aplikasi apa pun menggunakan suara dan teks. Sayangnya, tidak ada opsi sinonim, dan tidak ada ekstraksi entitas dan klasifikasi maksud yang tepat. Berikut ini adalah beberapa manfaat penting menggunakan Amazon Lex:

- Itu mudah. Ini memandu Anda dalam membuat chatbot.
- Memiliki algoritma pembelajaran yang mendalam. Algoritma seperti NLU dan NLP diimplementasikan untuk chatbot. Amazon telah memusatkan fungsi ini sehingga dapat dengan mudah digunakan.
- Memiliki fitur penyebaran dan penskalaan yang mudah.
- Memiliki integrasi bawaan dengan platform AWS.
- Hemat biaya.

IBM Watson

IBM menyediakan IBM Watson API untuk membuat chatbot Anda sendiri dengan cepat. Dalam pelaksanaannya, mendekati perjalanan sama pentingnya dengan perjalanan itu sendiri. Mendidik diri Anda sendiri tentang Watson Conversational AI untuk dasar-dasar perusahaan dari desain percakapan, dan dampaknya terhadap bisnis Anda, sangat penting dalam merumuskan rencana tindakan yang sukses. Persiapan ini akan memungkinkan Anda untuk berkomunikasi, belajar, dan memantau dengan standar, memungkinkan bisnis Anda membangun proyek yang siap untuk pelanggan dan sukses.

Desain percakapan adalah bagian terpenting dalam membangun chatbot. Hal pertama yang harus dipahami adalah siapa pengguna dan apa yang ingin mereka capai. IBM Watson memiliki banyak teknologi yang dapat Anda integrasikan dengan mudah di chatbot Anda; beberapa di antaranya adalah Watson Conversation, Watson Tone Analyzer, speech to text, dan masih banyak lagi.

Praktik Terbaik pada Pengembangan Chatbot

Saat membangun chatbot, penting untuk dipahami bahwa ada praktik terbaik tertentu yang dapat dimanfaatkan. Ini akan membantu dalam menciptakan bot ramah pengguna yang berhasil yang dapat memenuhi tujuannya untuk melakukan percakapan yang lancar dengan pengguna.

Salah satu hal terpenting dalam hubungan ini adalah mengenal target audiens dengan baik. Berikutnya adalah hal-hal lain seperti mengidentifikasi skenario kasus penggunaan, mengatur nada obrolan, dan mengidentifikasi platform pengiriman pesan. Dengan mengikuti praktik terbaik berikut, keinginan untuk memastikan percakapan yang lancar dengan pengguna dapat menjadi kenyataan.

Kenali Potensi Pengguna

Pemahaman menyeluruh tentang audiens target adalah langkah pertama dalam membangun bot yang sukses. Tahap selanjutnya adalah mengetahui tujuan dibuatnya bot tersebut. Berikut adalah beberapa poin yang perlu diingat:

- Ketahui apa tujuan dari bot tertentu. Bisa jadi bot untuk menghibur penonton, memudahkan pengguna bertransaksi, memberikan berita, atau berfungsi sebagai saluran layanan pelanggan.
- Jadikan bot lebih ramah pelanggan dengan mempelajari produk pelanggan.

Baca Sentimen Pengguna dan Buat Bot Memperkaya Emosi

Chatbot harus hangat dan ramah seperti manusia agar percakapan menjadi pengalaman yang menyenangkan. Itu harus cerdas membaca serta memahami sentimen pengguna untuk mempromosikan blok konten yang dapat mendorong pengguna untuk melanjutkan percakapan. Pengguna akan didorong untuk mengunjungi lagi jika pengalamannya kaya untuk pertama kalinya. Berikut adalah beberapa poin yang perlu diingat:

- Promosikan produk Anda atau ubah pengguna menjadi duta merek dengan memanfaatkan sentimen positif.
- Segera tanggapi komentar negatif agar tetap bertahan dalam permainan percakapan.
- Bila memungkinkan, gunakan bahasa yang ramah untuk membuat pengguna merasa seperti sedang berinteraksi dengan manusia yang sudah dikenalnya.
- Buat pengguna merasa nyaman dengan mengulang input dan memastikan bahwa mereka mampu memahami semua yang sedang dibahas.

BAB 12
DETEKSI DAN PENGENALAN WAJAH

Deteksi wajah adalah proses mendeteksi wajah dalam sebuah gambar atau video. Pengenalan wajah adalah proses mendeteksi wajah dalam sebuah gambar dan kemudian menggunakan algoritma untuk mengidentifikasi milik siapa wajah itu. Pengenalan wajah dengan demikian merupakan bentuk identifikasi orang.

Anda harus terlebih dahulu mengekstrak fitur dari gambar untuk melatih pengklasifikasi pembelajaran mesin guna mengidentifikasi wajah dalam gambar. Sistem ini tidak hanya nonsubjektif, tetapi juga otomatis—tidak diperlukan pelabelan fitur wajah dengan tangan. Anda cukup mengekstrak fitur dari wajah, melatih pengklasifikasi Anda, dan kemudian menggunakannya untuk mengidentifikasi wajah berikutnya. Karena untuk pengenalan wajah Anda harus terlebih dahulu mendeteksi wajah dari gambar, Anda dapat menganggap pengenalan wajah sebagai tahap dua fase.

- *Tahap 1:* Mendeteksi keberadaan wajah dalam aliran gambar atau video menggunakan metode seperti kaskade Haar, HOG + Linear SVM, Deep learning, atau algoritme lain yang dapat melokalisasi wajah.
- *Tahap 2*: Ambil setiap wajah yang terdeteksi selama fase pelokalan dan pelajari milik siapa wajah itu—inilah tempat Anda benar-benar menetapkan nama untuk sebuah wajah.

12.1 DETEKSI WAJAH, PENGENALAN WAJAH, DAN ANALISIS WAJAH

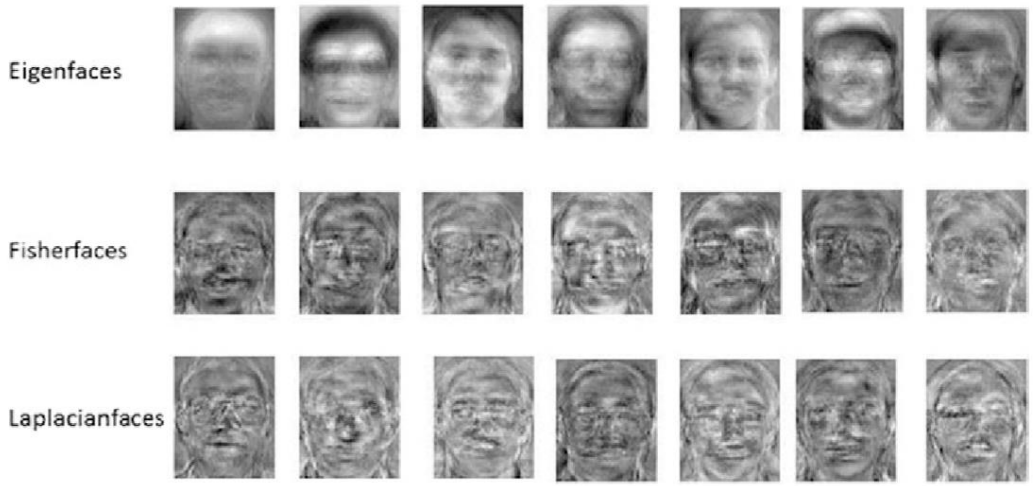
Ada perbedaan antara deteksi wajah, pengenalan wajah, dan analisis wajah.

- Deteksi wajah: Ini adalah teknik untuk menemukan semua wajah manusia dalam sebuah gambar.
- Pengenalan wajah: Ini adalah langkah berikutnya setelah deteksi wajah. Dalam pengenalan wajah, Anda mengidentifikasi wajah mana yang dimiliki orang yang menggunakan repositori gambar yang ada.
- Analisis wajah: Sebuah wajah diperiksa, dan beberapa kesimpulan diambil seperti usia, warna kulit, dan sebagainya.

12.2 OPENCV

OpenCV menyediakan tiga metode untuk pengenalan wajah (lihat Gambar 12-1):

- Eigenfaces
- Local binary pattern histograms /Histogram pola biner lokal (LBPH)
- Fisherfaces

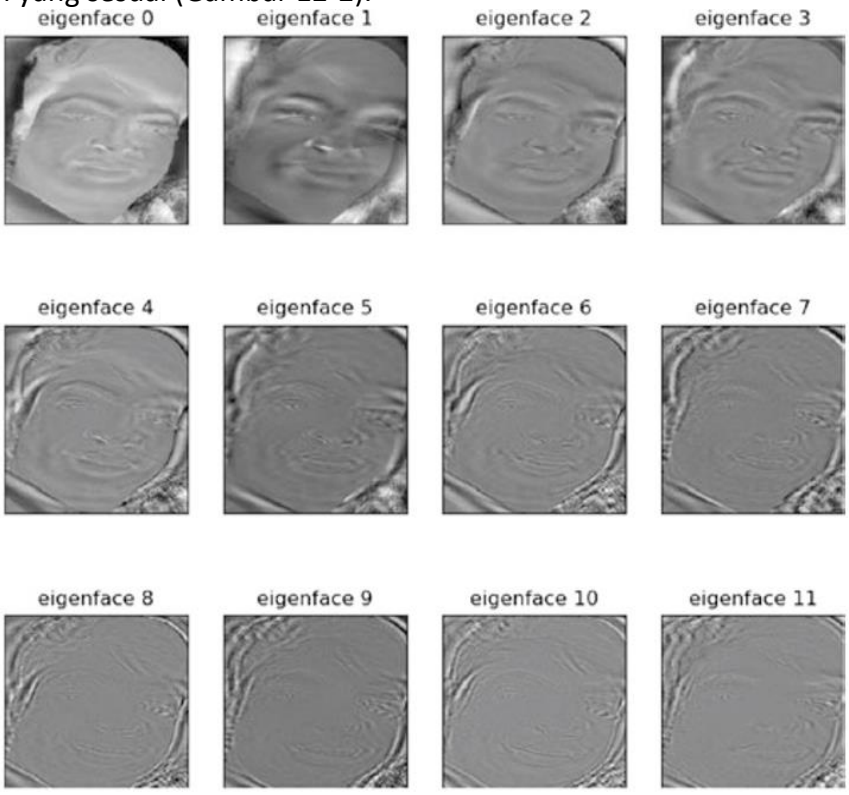


Gambar 12-1. Menerapkan metode OpenCV ke wajah

Ketiga metode mengenali wajah dengan membandingkan wajah dengan beberapa set pelatihan wajah yang dikenal. Untuk pelatihan, Anda menyediakan algoritma dengan wajah dan memberi label dengan orang yang memilikinya. Saat Anda menggunakan algoritme untuk mengenali beberapa wajah yang tidak dikenal, ia menggunakan model yang dilatih pada set pelatihan untuk membuat pengenalan. Masing-masing dari tiga metode yang disebutkan di atas menggunakan set pelatihan yang sedikit berbeda. Wajah Laplacian bisa menjadi cara lain untuk mengenali wajah.

Eigenfaces

Algoritma *eigenfaces* menggunakan analisis komponen utama untuk membuat representasi gambar wajah berdimensi rendah, yang akan Anda gunakan sebagai fitur untuk gambar wajah yang sesuai (Gambar 12-2).



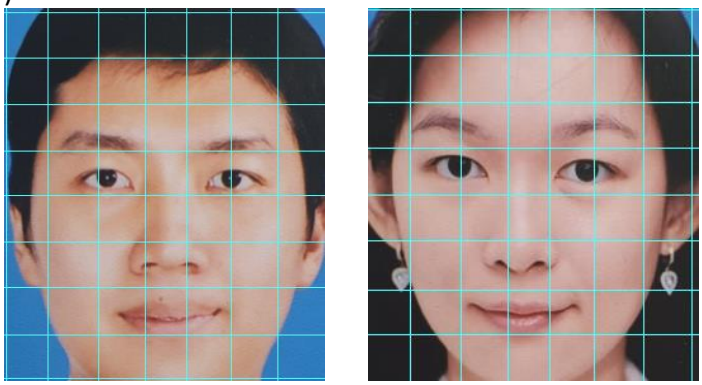
Gambar 12-2. Menerapkan dekomposisi nilai Eigen dan mengekstraksi 11 eigenface dengan magnitudo terbesar

Untuk ini, Anda mengumpulkan kumpulan data wajah dengan beberapa gambar wajah dari setiap orang yang ingin Anda kenali—seperti memiliki beberapa contoh pelatihan dari kelas gambar yang ingin Anda beri label dalam klasifikasi gambar. Dengan kumpulan data gambar wajah ini, yang dianggap memiliki lebar dan tinggi yang sama dan idealnya dengan mata dan struktur wajah yang sejajar pada koordinat (x, y) yang sama, Anda menerapkan dekomposisi nilai eigen dari kumpulan data, menjaga vektor eigen dengan nilai eigen terbesar yang sesuai. Mengingat vektor eigen ini, wajah kemudian dapat direpresentasikan sebagai kombinasi linier dari apa yang disebut Kirby dan Sirovich sebagai wajah eigen. Algoritma eigenfaces melihat seluruh kumpulan data.

LBPH

Anda dapat menganalisis setiap gambar secara independen di LBPH. Metode LBPH agak lebih sederhana, dalam arti bahwa Anda mengkarakterisasi setiap gambar dalam kumpulan data secara lokal; ketika gambar baru yang tidak dikenal disediakan, Anda melakukan analisis yang sama pada gambar tersebut dan membandingkan hasilnya dengan setiap gambar dalam kumpulan data. Cara Anda menganalisis gambar adalah dengan mengkarakterisasi pola lokal di setiap lokasi dalam gambar.

Sementara algoritma eigenfaces bergantung pada PCA untuk membangun representasi dimensi rendah dari gambar wajah, metode *local binary pattern/*pola biner lokal (LBP) bergantung pada, seperti namanya, ekstraksi fitur. Pertama kali diperkenalkan oleh Ahonen et al. dalam makalah tahun 2006 “Pengenalan Wajah dengan Pola Biner Lokal”, metode ini menyarankan untuk membagi gambar wajah menjadi kotak 7×7 sel berukuran sama (Gambar 12-3).



Gambar 12-3. Menerapkan LBPH untuk pengenalan wajah dimulai dengan membagi gambar wajah menjadi kotak 7x7 sel berukuran sama

Anda kemudian mengekstrak histogram pola biner lokal dari masing-masing 49 sel. Dengan membagi gambar ke dalam sel, Anda memasukkan lokalitas ke dalam vektor fitur akhir. Selanjutnya, sel-sel di tengah memiliki bobot lebih sehingga mereka berkontribusi lebih banyak pada representasi keseluruhan. Sel di sudut membawa lebih sedikit informasi identitas wajah dibandingkan dengan sel di tengah kisi (yang berisi struktur mata, hidung, dan bibir). Terakhir, Anda menggabungkan histogram LBP tertimbang ini dari 49 sel untuk membentuk vektor fitur akhir Anda.

Fisherface

*Principal Component Analysis (*PCA), yang merupakan inti dari metode Eigenfaces, menemukan kombinasi linear fitur yang memaksimalkan total varians dalam data. Meskipun ini jelas merupakan cara yang ampuh untuk merepresentasikan data, ini tidak mempertimbangkan kelas apa pun sehingga banyak informasi diskriminatif yang mungkin hilang saat membuang komponen. Bayangkan sebuah situasi di mana varians dalam data Anda dihasilkan oleh sumber eksternal, biarkan saja. Komponen yang diidentifikasi oleh PCA

tidak harus mengandung informasi diskriminatif sama sekali, sehingga sampel yang diproyeksikan dioleskan bersama dan klasifikasi menjadi tidak mungkin.

Analisis Diskriminan Linier melakukan pengurangan dimensi kelas khusus dan ditemukan oleh ahli statistik hebat Sir R. A. Fisher. Penggunaan beberapa pengukuran dalam masalah taksonomi. Untuk menemukan kombinasi fitur yang memisahkan terbaik antar kelas, Analisis Diskriminan Linier memaksimalkan rasio antar kelas terhadap pencar di dalam kelas, alih-alih memaksimalkan pencar secara keseluruhan. Idenya sederhana: kelas yang sama harus mengelompok bersama-sama, sementara kelas yang berbeda berada sejauh mungkin satu sama lain dalam representasi dimensi yang lebih rendah.

12.3 MENDETEKSI WAJAH

Fitur pertama yang Anda perlukan untuk melakukan pengenalan wajah adalah mendeteksi di mana dalam gambar saat ini ada wajah. Dengan Python Anda dapat menggunakan filter kaskade Haar dari Library OpenCV untuk melakukan ini secara efisien.

Untuk implementasi yang ditampilkan di sini, saya menggunakan Anaconda dengan Python 3.5, OpenCV 3.1.0, dan dlib 19.1.0. Untuk menggunakan kode berikut, pastikan Anda memiliki versi ini (atau yang lebih baru). Untuk melakukan deteksi wajah, beberapa inisialisasi harus dilakukan, seperti yang ditunjukkan di sini:

```
# Import the OpenCV library
import cv2
# Initialize a face cascade using the frontal face haar cascade provided
# with the OpenCV2 library. This will be required for face detection in an
# image.
faceCascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
# The desired output width and height, can be modified according to the needs.
OUTPUT_SIZE_WIDTH = 700
OUTPUT_SIZE_HEIGHT = 600

# Open the first webcam device
capture = cv2.VideoCapture(0)

# Create two opencv named windows for showing the input, output images.
cv2.namedWindow("base-image", cv2.WINDOW_AUTOSIZE)
cv2.namedWindow("result-image", cv2.WINDOW_AUTOSIZE)

# Position the windows next to each other
cv2.moveWindow("base-image", 20, 200)
cv2.moveWindow("result-image", 640, 200)

# Start the window thread for the two windows we are using
cv2.startWindowThread()

rectangleColor = (0, 100, 255)
```

Sisa kode akan menjadi loop tak terbatas yang terus mendapatkan gambar terbaru dari webcam, mendeteksi semua wajah dalam gambar yang diambil, menggambar persegi panjang di sekitar wajah terbesar yang terdeteksi, dan akhirnya menunjukkan input, output gambar di jendela (Gambar 12-4).

Gambar 12-4. Output sampel yang menunjukkan wajah yang terdeteksi

Anda dapat melakukan ini dengan kode berikut dalam infinite loop:

```
# Retrieve the latest image from the webcam
rc,fullSizeBaseImage = capture.read()
# Resize the image to 520x420
baseImage= cv2.resize(fullSizeBaseImage, (520, 420))

# Check if a key was pressed and if it was Q or q, then destroy all
# opencv windows and exit the application, stopping the infinite loop.
pressedKey = cv2.waitKey(2)
if (pressedKey == ord('Q')) | (pressedKey == ord('q')):
cv2.destroyAllWindows()
exit(0)
# Result image is the image we will show the user, which is a
# combination of the original image captured from the webcam with the
# overlayed rectangle detecting the largest face
resultImage = baseImage.copy()

# We will be using gray colored image for face detection.
# So we need to convert the baseImage captured by webcam to a gray-based image
gray_image = cv2.cvtColor(baseImage, cv2.COLOR_BGR2GRAY)
# Now use the haar cascade detector to find all faces in the
# image
faces = faceCascade.detectMultiScale(gray_image, 1.3, 5)
```

```
# As we are only interested in the 'largest' face, we need to
# calculate the largest area of the found rectangle.
# For this, first initialize the required variables to 0.
maxArea = 0
x = 0
y = 0
w = 0
h = 0

# Loop over all faces found in the image and check if the area for this face is
# the largest so far
for(_x, _y, _w, _h) in faces:
if _w * _h > maxArea:
        x = _x
        y = _y
        w = _w
        h = _h
    maxArea = w * h

# If any face is found, draw a rectangle around the
#  largest face present in the picture
If maxArea > 0:
cv2.rectangle(resultImage, (x-10, y-20),
(x + w+10, y + h+20), rectangleColor, 2)
# Since we want to show something larger on the screen than the
# original 520x420, we resize the image again
```

```
# Note that it would also be possible to keep the large version
# of the baseimage and make the result image a copy of this large
# base image and use the scaling factor to draw the rectangle
# at the right coordinates.
largeResult = cv2.resize(resultImage,
(OUTPUT_SIZE_WIDTH, OUTPUT_SIZE_HEIGHT))
# Finally, we show the images on the screen
cv2.imshow("base-image", baseImage)
cv2.imshow("result-image", largeResult)
```

12.4 MELACAK WAJAH

Kode deteksi wajah sebelumnya memiliki beberapa kelemahan.

- Kode mungkin mahal secara komputasi.
- Jika orang yang terdeteksi menoleh sedikit, kaskade Haar mungkin tidak mendeteksi wajah.
- Sulit untuk melacak wajah di antara bingkai.

Pendekatan yang lebih baik untuk ini adalah melakukan deteksi wajah sekali dan kemudian menggunakan pelacak korelasi dari Library dlib yang sangat baik untuk hanya melacak wajah dari bingkai ke bingkai.

Agar ini berfungsi, Anda perlu mengimpor Library lain dan menginisialisasi variabel tambahan.

```
import dlib

# Create the tracker we will use to recognize face in different frames
# we get from the webcam
tracker = dlib.correlation_tracker()

# The Boolean variable we use to keep track whether we are
# using dlib tracker, or not.
trackingFace = 0
```

Dalam *infinite for loop*, Anda sekarang akan menentukan apakah pelacak korelasi dlib saat ini melacak wilayah dalam gambar. Jika tidak demikian, Anda akan menggunakan kode yang sama seperti sebelumnya untuk menemukan wajah terbesar, tetapi alih-alih menggambar persegi panjang, Anda menggunakan koordinat yang ditemukan untuk menginisialisasi pelacak korelasi.

```
# If we are not tracking a face, then try to detect one using the above code itself.
if not trackingFace:

# We will be using gray colored image for face detection.
# So we need to convert the baseImage captured by webcam to a gray-based image
gray = cv2.cvtColor(baseImage, cv2.COLOR_BGR2GRAY)
# Now use the haar cascade detector to find all faces
# in the image
faces=faceCascade.detectMultiScale(gray,1.3,5)

# In the console we can show our this case of using the
# detector for a face, when we are detecting it for first time.
print("Using the cascade detector to detect face")


# As we are only interested in the 'largest' face, we need to
# calculate the largest area of the found rectangle.
# For this, first initialize the required variables to 0.
maxArea = 0
x = 0
y = 0
w = 0
h =0
```

```
# Loop over all faces and check if the area for this
# face is the largest so far
# We need to convert it to int here because dlib tracker
# needs an int as its argument. If we omit the cast to
# int here, you will get cast errors since the detector
# returns numpy.int32 and the tracker requires an int
for(_x, _y, _w, _h) in faces:
if _w * _h > maxArea:
            x = int(_x)
            y = int(_y)
            w = int(_w)
            h = int(_h)
    maxArea = w * h

# If any face is found, draw a rectangle around the
# largest face present in the picture
if maxArea > 0:

#Initialize the tracker
tracker.start_track(baseImage,
dlib.rectangle(x-10, y-20, x+w+10, y+h+20))

# Set the indicator variable such that we know the
# tracker is tracking a face in the image
trackingFace = 1
```

Sekarang bit terakhir dalam infinite loop adalah memeriksa kembali apakah pelacak korelasi secara aktif melacak wajah (yaitu, apakah itu hanya mendeteksi wajah dengan kode sebelumnya, trankingFace=1?). Jika pelacak secara aktif melacak wajah di gambar, Anda akan memperbarui pelacak. Bergantung pada kualitas pembaruan (yaitu, seberapa yakin pelacak tentang apakah itu masih melacak wajah yang sama), Anda dapat menggambar persegi panjang di sekitar wilayah yang ditunjukkan oleh pelacak atau menunjukkan bahwa Anda tidak melacak wajah lagi.

```
# Check if the tracker is actively tracking a face in the image
if trackingFace:

# Update the tracker and request information about the
# quality of the tracking update
trackingQuality = tracker.update(baseImage)

# If the tracking quality is good enough, determine the
# updated position of the tracked region and draw the
# rectangle
If trackingQuality >= 9.0:
tracked_position = tracker.get_position()

t_x = int(tracked_position.left())
t_y = int(tracked_position.top())
t_w = int(tracked_position.width())
t_h = int(tracked_position.height())
cv2.rectangle(resultImage, (t_x,t_y),
(t_x+t_w, t_y+t_h),
rectangleColor, 2)

else:
# If the quality of the tracking update is not good enough
# for us  (e.g. the face being tracked moved out of the
# screen) we stop the tracking of the face and in the
# next loop we will find the largest face in the image
# again
trackingFace = 0
```

Seperti yang Anda lihat dalam kode, Anda mencetak pesan ke konsol setiap kali Anda menggunakan detektor lagi. Jika Anda melihat output konsol saat menjalankan aplikasi ini, Anda akan melihat bahwa meskipun Anda sedikit bergerak di layar, pelacak cukup baik dalam mengikuti wajah setelah terdeteksi.

12.5 PENGENALAN WAJAH

Sistem pengenalan wajah mengidentifikasi nama orang yang ada dalam bingkai video dengan mencocokkan wajah di setiap bingkai video dengan gambar terlatih dan mengembalikan (dan menulis dalam file CSV) label jika wajah dalam bingkai berhasil dicocokkan. Anda sekarang akan melihat cara membuat sistem pengenalan wajah langkah demi langkah. Pertama, Anda mengimpor semua Library yang diperlukan. face_recognition adalah Library sederhana yang dibuat menggunakan pengenalan wajah canggih dlib yang juga dibuat dengan Deep learning.

```
import os
import re
import warnings
import scipy.misc
import cv2
import face_recognition
from PIL import Image
import argparse
import csv
import os
```

Argparse adalah Library Python yang memungkinkan Anda menambahkan argumen Anda sendiri ke file; itu kemudian dapat digunakan untuk memasukkan direktori gambar atau jalur file apa pun pada saat eksekusi.

```
parser = argparse.ArgumentParser()
parser.add_argument("-i", "--images-dir", help="image dir")
parser.add_argument("-v", "--video", help="video to recognize faces on")
parser.add_argument("-o", "--output-csv", help="Ouput csv file [Optional]")
parser.add_argument("-u", "--upsample-rate", help="How many times to upsample the image looking for faces. Higher numbers
                find smaller faces. [Optional]")
args = vars(parser.parse_args())
```

Dalam kode sebelumnya, saat menjalankan file Python ini, Anda harus menentukan yang berikut: direktori gambar input pelatihan, file video yang akan kita gunakan sebagai kumpulan data, dan file CSV output untuk menulis output pada setiap kerangka waktu.

```
#Check if argument values are valid
if args.get("images_dir", None) is None and os.path.exists(args.get("images_dir", None)):
    print("Please check the path to images folder")
    exit()
if args.get("video", None) is None and os.path.isfile(args.get("video", None)):
    print("Please check the path to video")
    exit()
if args.get("output_csv", None) is None:
    print("You haven't specified an output csv file. Nothing will be written.")
# By default upsample rate = 1
upsample_rate = args.get("upsample_rate", None)
if upsample_rate is None:
    upsample_rate = 1
```

```
# Helper functions
def image_files_in_folder(folder):
    return [os.path.join(folder, f) for f in os.listdir(folder) if re.match(r'.*\.(jpg|jpeg|png)', f, flags=re.I)]
```

Dengan menggunakan fungsi sebelumnya, semua file gambar dari folder yang ditentukan dapat dibaca. Fungsi berikut menguji frame input dengan gambar pelatihan yang diketahui:

```
def test_image(image_to_check, known_names, known_face_encodings, number_of_times_to_upsample=1):
    """
    Test if any face is recognized in unknown image by checking known images
    :paramimage_to_check: Numpy array of the image
    :paramknown_names: List containing known labels
    :paramknown_face_encodings: List containing training image labels
    :paramnumber_of_times_to_upsample: How many times to upsample the image looking for
    faces. Higher numbers find smaller faces.
    :return: A list of labels of known names
    """
    # unknown_image = face_recognition.load_image_file(image_to_check)
    unknown_image = image_to_check
    # Scale down the image to make it run faster
    if unknown_image.shape[1] > 1600:
        scale_factor = 1600 / unknown_image.shape[1]
        with warnings.catch_warnings():
            warnings.simplefilter("ignore")
        unknown_image = scipy.misc.imresize(unknown_image, scale_factor)
    face_locations = face_recognition.face_locations(unknown_image,  number_of_times_to_upsample)
    unknown_encodings = face_recognition.face_encodings(unknown_image, face_locations)
    result = []
    for unknown_encoding in unknown_encodings:
        result = face_recognition.compare_faces(known_face_encodings, unknown_encoding)
    result_encoding = []
    for nameIndex, is_match in enumerate(result):
        if is_match:
            result_encoding.append(known_names[nameIndex])

    return result_encoding
```

Sekarang Anda menentukan fungsi untuk mengekstrak label untuk gambar yang cocok dan dikenal.

```
def map_file_pattern_to_label(labels_with_pattern, labels_list):
    """
    Map file name pattern to full label
    :paramlabels_with_pattern: dict : { "file_name_pattern": "full_label" }
    :paramlabels_list: list : list of labels of file names got from test_image()
    :return: list of full labels
    """
    result_list = []
    for key, label in labels_with_pattern.items():
        for img_labels in labels_list:
            if str(key).lower() in str(img_labels).lower():
                if str(label) not in result_list:
                    result_list.append(str(label))
                # continue
    # result_list = [label for key, label in labels_with_pattern if str(key).lower() in labels_list]
    return result_list
```

Baca video input untuk mengekstrak frame uji.

```
cap = cv2.VideoCapture(args["video"])

#get the training images
training_encodings = []
training_labels = []
for file in image_files_in_folder(args['images_dir']):
    basename = os.path.splitext(os.path.basename(file))[0]
    img = face_recognition.load_image_file(file)
    encodings = face_recognition.face_encodings(img)

    if len(encodings) > 1:
        print("WARNING: More than one face found in {}. Only considering the first face.".format(file))

    if len(encodings) == 0:
        print("WARNING: No faces found in {}. Ignoring file.".format(file))
    if len(encodings):
        training_labels.append(basename)
        training_encodings.append(encodings[0])


csvfile = None
csvwriter = None
if args.get("output_csv", None) is not None:
    csvfile = open(args.get("output_csv"), 'w')
    csvwriter = csv.writer(csvfile, delimiter=',', quotechar='|', quoting=csv.QUOTE_MINIMAL)

ret, firstFrame = cap.read()
frameRate = cap.get(cv2.CAP_PROP_FPS)
```

Sekarang tentukan label set pelatihan Anda. Kemudian cocokkan frame yang diekstrak dari video input yang diberikan untuk mendapatkan hasil yang diinginkan.

```
# Labels with file pattern, edit this
label_pattern = {
    "shah": "Shahrukh Khan",
    "amir": "Amir Khan"
            }

# match each frame in video with our trained set of labeled images
while ret:
    curr_frame = cap.get(1)

    ret, frame = cap.read()

    result = test_image(frame, training_labels, training_encodings, upsample_rate)
    labels = map_file_pattern_to_label(label_pattern, result)
    curr_time = curr_frame / frameRate
    print("Time: {} faces: {}".format(curr_time, labels))
    if csvwriter:
        csvwriter.writerow([curr_time, labels])
    cv2.imshow('frame', frame)

    key = cv2.waitKey(1) & 0xFF
    if key == ord('q'):
        break
if csvfile:
    csvfile.close()
cap.release()
cv2.destroyAllWindows()
```

12.6 PENGENALAN WAJAH BERBASIS *DEEP LEARNING*

Impor paket yang diperlukan.

```
import cv2                # working with, mainly resizing, images
import numpy as np        # dealing with arrays
import os                 # dealing with directories
from random import shuffle # mixing up or currently ordered data that might lead our network astray in training.
from tqdm import tqdm
from scipy import misc
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression
import tensorflow as tf
import glob
import matplotlib.pyplot as plt
import dlib
```

Inisialisasi variabel.

```
from skimage import io
tf.reset_default_graph()
TRAIN_DIR ='resize_a/train'
TEST_DIR ='resize_a/test'
IMG_SIZE = 200
boxScale=1
LR = 1e-3
MODEL_NAME = 'quickest.model'.format(LR, '2conv-basic')
```

Fungsi label_img() digunakan untuk membuat larik label, dan fungsi detect_faces() mendeteksi bagian wajah pada gambar.

```
def label_img(img):
    word = img.split('(')[-2]
    word_label = word[0]
    if word_label == 'R': return [1,0]

    elif word_label == 'A': return [0,1]

def detect_faces(image):

    # Create a face detector
    face_detector = dlib.get_frontal_face_detector()

    # Run detector and get bounding boxes of the faces on image.
    detected_faces = face_detector(image, 1)
    face_frames = [(x.left(), x.top(),
                    x.right(), x.bottom()) for x in detected_faces]

    return face_frames
```

Fungsi create_train_data() digunakan untuk pra-pemrosesan data pelatihan.

```
def create_train_data():
    training_data = []
    for img in tqdm(os.listdir(TRAIN_DIR)):
        label = label_img(img)
        path = os.path.join(TRAIN_DIR,img)
        img= misc.imread(path)
        img = cv2.imread(path,cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
        detected_faces = detect_faces(img)
        for n, face_rect in enumerate(detected_faces):
            img = Image.fromarray(img).crop(face_rect)
            img = np.array(img)
        img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
# If any face is found, draw a rectangle around the
#  largest face present in the picture

        training_data.append([np.array(img),np.array(label)])
    shuffle(training_data)
    np.save('train_data.npy', training_data)
    return training_data
```

Fungsi process_test_data() digunakan untuk melakukan praproses data pengujian.

```
def process_test_data():
    testing_data = []
    for img in tqdm(os.listdir(TEST_DIR)):
        path = os.path.join(TEST_DIR,img)
        imgnum = img.split('.')[-2]
        img_num=get_num(imgnum)
        img= misc.imread(path)
        img = cv2.imread(path,cv2.IMREAD_GRAYSCALE)
        img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
        detected_faces = detect_faces(img)
        for n, face_rect in enumerate(detected_faces):
            img = Image.fromarray(img).crop(face_rect)
            img = np.array(img)
        img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
# If any face is found, draw a rectangle around the
#  largest face present in the picture

        testing_data.append([np.array(img), img_num])
```

Kemudian Anda membuat model dan memasukkan data pelatihan ke dalam model.

```
train_data= create_train_data()
train = train_data[:-2]
test = train_data[-2:]
X = np.array([i[0] for i in train]).reshape(-1,200,200,1)
Y = [i[1] for i in train]
test_x = np.array([i[0] for i in test]).reshape(-1,200,200,1)
test_y = [i[1] for i in test]
convnet = input_data(shape=[None, 200, 200, 1], name='input')

convnet = conv_2d(convnet, 4, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 5, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)

convnet = conv_2d(convnet, 8, 5, activation='relu')
convnet = max_pool_2d(convnet, 5)

convnet = fully_connected(convnet, 8, activation='relu')
convnet = dropout(convnet, 0.2)

convnet = fully_connected(convnet, 2, activation='softmax')
convnet = regression(convnet, optimizer='adam', learning_rate=LR, loss='categorical_crossentropy', name='targets')
model.fit({'input': X}, {'targets': Y}, n_epoch=1, validation_set=({'input': test_x}, {'targets': test_y}),
    snapshot_step=500, show_metric=True, run_id=MODEL_NAME)
```

Terakhir, Anda menyiapkan data uji dan memprediksi hasilnya.

```
test_data = process_test_data()

fig=plt.figure()

for num,data in enumerate(test_data[:12]):


    img_num = data[1]
    img_data = data[0]
    y = fig.add_subplot(3,4,num+1)
    orig = img_data
    data = img_data.reshape(IMG_SIZE,IMG_SIZE,1)
    #model_out = model.predict([data])[0]
    model_out = model.predict([data])[0]

    if np.argmax(model_out) == 0: str_label='Ronaldo'
    elif np.argmax(model_out) == 1: str_label='amitabh'

    y.imshow(orig,cmap='gray')
    plt.title(str_label)
    y.axes.get_xaxis().set_visible(False)
    y.axes.get_yaxis().set_visible(False)
plt.show()
```

12.7 *TRANSFER LEARNING*

Transfer learning memanfaatkan pengetahuan yang diperoleh saat memecahkan satu masalah dan menerapkannya pada masalah yang berbeda tetapi terkait. Di sini Anda akan melihat bagaimana Anda dapat menggunakan jaringan saraf dalam yang telah dilatih sebelumnya yang disebut model Inception v3 untuk mengklasifikasikan gambar. Model Inception cukup mampu mengekstraksi informasi yang berguna dari sebuah gambar.

Mengapa *Transfer Learning?*

Sudah diketahui bahwa jaringan convolutional membutuhkan sejumlah besar data dan sumber daya untuk dilatih. Sudah menjadi norma bagi para peneliti dan praktisi untuk menggunakan pembelajaran transfer dan penyesuaian (yaitu, mentransfer bobot jaringan yang dilatih pada proyek sebelumnya seperti ImageNet ke tugas baru).

Anda dapat mengambil dua pendekatan.

- *Transfer Learning*: Anda dapat mengambil CNN yang telah dilatih sebelumnya di ImageNet, menghapus layer terakhir yang terhubung sepenuhnya, dan kemudian memperlakukan CNN lainnya sebagai pengekstrak fitur untuk kumpulan data baru. Setelah mengekstrak fitur untuk semua gambar, Anda melatih pengklasifikasi untuk kumpulan data baru.
- *Fine-tuning:* Anda dapat mengganti dan melatih kembali pengklasifikasi di atas CNN dan juga menyempurnakan bobot jaringan pra-latihan melalui backpropagation.

Contoh *Transfer Learning*

Dalam contoh ini, pertama-tama Anda akan mencoba mengklasifikasikan gambar dengan memuat langsung model Inception v3. Impor semua Library yang diperlukan.

```
%matplotlib inline
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
import os

# Functions and classes for loading and using the Inception model.
import inception
```

Sekarang tentukan direktori penyimpanan untuk model dan kemudian unduh model Inception v3.

```
inception.data_dir = 'D:/'
```

```
inception.maybe_download()
```

Muat model yang telah dilatih sebelumnya dan tentukan fungsi untuk mengklasifikasikan gambar yang diberikan.

```
model = inception.Inception()
```

```
def classify(image_path):
    # Display the image.
    p = Image.open(image_path)
    p.show()

    # Use the Inception model to classify the image.
    pred = model.classify(image_path=image_path)

    # Print the scores and names for the top-10 predictions.
    model.print_scores(pred=pred, k=10, only_first_name=True)
```

Sekarang setelah model didefinisikan, mari kita periksa untuk beberapa gambar.

Gambar 12.5 Gambar Minuman Botol

Ini memberikan hasil yang benar 91,11 persen, tetapi sekarang jika Anda memeriksa seseorang, inilah yang Anda dapatkan seperti gambar di bawah ini :

Gambar 12.6 Gambar Model Pembanding

Ini bola tenis 48,50 persen! Sayangnya, model Inception sepertinya tidak bisa mengklasifikasikan gambar orang. Alasan untuk ini adalah kumpulan data yang digunakan untuk melatih model Inception, yang memiliki beberapa label teks yang membingungkan untuk kelas. Anda dapat menggunakan kembali model Inception yang telah dilatih sebelumnya dan hanya mengganti layer yang melakukan klasifikasi akhir. Ini disebut pembelajaran transfer. Pertama Anda memasukkan dan memproses gambar dengan model Inception. Tepat sebelum layer klasifikasi akhir dari model Inception, Anda menyimpan apa yang disebut nilai transfer ke file cache.

Alasan menggunakan file cache adalah untuk memproses gambar dengan model Inception membutuhkan waktu yang lama. Ketika semua gambar dalam kumpulan data baru telah diproses melalui model Inception dan nilai transfer yang dihasilkan disimpan ke file cache, maka Anda dapat menggunakan nilai transfer tersebut sebagai input ke jaringan saraf lainnya. Anda kemudian akan melatih jaringan saraf kedua menggunakan kelas dari kumpulan data baru, sehingga jaringan belajar bagaimana mengklasifikasikan gambar berdasarkan nilai transfer dari model Inception. Dengan cara ini, model Inception digunakan untuk mengekstrak informasi yang berguna dari gambar, dan jaringan saraf lain kemudian digunakan untuk klasifikasi yang sebenarnya.

12.8 MENGHITUNG NILAI TRANSFER *(TRANSFER_VALUE)*

Impor fungsi transfer_value_cache dari file Inception.

```
from inception import transfer_values_cache
```

```
file_path_cache_train = os.path.join(cifar10.data_path, 'inception_cifar10_train.pkl')
file_path_cache_test = os.path.join(cifar10.data_path, 'inception_cifar10_test.pkl')
```

```
print("Processing Inception transfer-values for training-images ...")

# Scale images because Inception needs pixels to be between 0 and 255,
# while the CIFAR-10 functions return pixels between 0.0 and 1.0
images_scaled = images_train * 255.0

# If transfer-values have already been calculated then reload them,
# otherwise calculate them and save them to a cache-file.
transfer_values_train = transfer_values_cache(cache_path=file_path_cache_train,
                                              images=images_scaled,
                                              model=model)
```

```
print("Processing Inception transfer-values for test-images ...")

# Scale images because Inception needs pixels to be between 0 and 255,
# while the CIFAR-10 functions return pixels between 0.0 and 1.0
images_scaled = images_test * 255.0

# If transfer-values have already been calculated then reload them,
# otherwise calculate them and save them to a cache-file.
transfer_values_test = transfer_values_cache(cache_path=file_path_cache_test,
                                             images=images_scaled,
                                             model=model)
```

Sampai sekarang, nilai transfer disimpan dalam file cache. Sekarang Anda akan membuat jaringan saraf baru. Tentukan jaringan.

```
# Wrap the transfer-values as a Pretty Tensor object.
x_pretty = pt.wrap(x)

with pt.defaults_scope(activation_fn=tf.nn.relu):
    y_pred, loss = x_pretty.\
        fully_connected(size=1024, name='layer_fc1').\
        softmax_classifier(num_classes=num_classes, labels=y_true)
```

Berikut cara optimasinya:

```
global_step = tf.Variable(initial_value=0,
                          name='global_step', trainable=False)
optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(loss, global_step)
```

Berikut adalah akurasi klasifikasi:

```
y_pred_cls = tf.argmax(y_pred, dimension=1)
correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

Inilah proses TensorFlow:

```
session = tf.Session()
session.run(tf.global_variables_initializer())
```

Berikut adalah fungsi helper untuk melakukan batch training:

```
def random_batch():
    # Number of images (transfer-values) in the training-set.
    num_images = len(transfer_values_train)

    # Create a random index.
    idx = np.random.choice(num_images,
                           size=train_batch_size,
                           replace=False)

    # Use the random index to select random x and y-values.
    # We use the transfer-values instead of images as x-values.
    x_batch = transfer_values_train[idx]
    y_batch = labels_train[idx]

    return x_batch, y_batch
```

Untuk pengoptimalan, berikut adalah kodenya:

```
def optimize(num_iterations):
    # Number of images (transfer-values) in the training-set.


    start_time = time.time()

    for i in range(num_iterations):
        # Get a batch of training examples.
        # x_batch now holds a batch of images (transfer-values) and
        # y_true_batch are the true labels for those images.
        x_batch, y_true_batch = random_batch()

        # Put the batch into a dict with the proper names
        # for placeholder variables in the TensorFlow graph.
        feed_dict_train = {x: x_batch,
                           y_true: y_true_batch}

        # Run the optimizer using this batch of training data.
        # TensorFlow assigns the variables in feed_dict_train
        # to the placeholder variables and then runs the optimizer.
        # We also want to retrieve the global_step counter.
        i_global, _ = session.run([global_step, optimizer],
                                  feed_dict=feed_dict_train)

        # Print status to screen every 100 iterations (and last).
        if (i_global % 100 == 0) or (i == num_iterations - 1):
            # Calculate the accuracy on the training-batch.
            batch_acc = session.run(accuracy,
                                    feed_dict=feed_dict_train)
```

```
            # Print status.
            msg = "Global Step: {0:>6}, Training Batch Accuracy: {1:>6.1%}"
            print(msg.format(i_global, batch_acc))

    # Ending time.
    end_time = time.time()

    # Difference between start and end-times.
    time_dif = end_time - start_time

    # Print the time-usage.
    print("Time usage: " + str(timedelta(seconds=int(round(time_dif)))))

    # Use the random index to select random x and y-values.
    # We use the transfer-values instead of images as x-values.
    x_batch = transfer_values_train[idx]
    y_batch = labels_train[idx]

    return x_batch, y_batch
```

Untuk memplot matriks konfusi, berikut adalah kodenya:

```
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(cls_pred):
    # This is called from print_test_accuracy() below.

    # cls_pred is an array of the predicted class-number for
    # all images in the test-set.

    # Get the confusion matrix using sklearn.
    cm = confusion_matrix(y_true=cls_test,  # True class for test-set.
                          y_pred=cls_pred)  # Predicted class.

    # Print the confusion matrix as text.
    for i in range(num_classes):
        # Append the class-name to each line.
        class_name = "({}) {}".format(i, class_names[i])
        print(cm[i, :], class_name)

    # Print the class-numbers for easy reference.
    class_numbers = [" ({0})".format(i) for i in range(num_classes)]
    print("".join(class_numbers))
```

Berikut adalah fungsi pembantu untuk menghitung klasifikasi:

```
# Split the data-set in batches of this size to limit RAM usage.
batch_size = 256

def predict_cls(transfer_values, labels, cls_true):
    # Number of images.
    num_images = len(transfer_values)

    # Allocate an array for the predicted classes which
    # will be calculated in batches and filled into this array.
    cls_pred = np.zeros(shape=num_images, dtype=np.int)

    # Now calculate the predicted classes for the batches.
    # We will just iterate through all the batches.
    # There might be a more clever and Pythonic way of doing this.

    # The starting index for the next batch is denoted i.
    i = 0

    while i < num_images:
        # The ending index for the next batch is denoted j.
        j = min(i + batch_size, num_images)

        # Create a feed-dict with the images and labels
        # between index i and j.
        feed_dict = {x: transfer_values[i:j],
                     y_true: labels[i:j]}

        # Calculate the predicted class using TensorFlow.
        cls_pred[i:j] = session.run(y_pred_cls, feed_dict=feed_dict)

        # Set the start-index for the next batch to the
        # end-index of the current batch.
        i = j

    # Create a boolean array whether each image is correctly classified.
    correct = (cls_true == cls_pred)

    return correct, cls_pred
```

```
def classification_accuracy(correct):
    # When averaging a boolean array, False means 0 and True means 1.
    # So we are calculating: number of True / len(correct) which is
    # the same as the classification accuracy.

    # Return the classification accuracy
    # and the number of correct classifications.
    return correct.mean(), correct.sum()
```

```
def predict_cls_test():
    return predict_cls(transfer_values = transfer_values_test,
                       labels = labels_test,
                       cls_true = cls_test)
```

Sekarang mari kita jalankan.

```
def print_test_accuracy(show_example_errors=False,
                        show_confusion_matrix=False):

    # For all the images in the test-set,
    # calculate the predicted classes and whether they are correct.
    correct, cls_pred = predict_cls_test()

    # Classification accuracy and the number of correct classifications.
    acc, num_correct = classification_accuracy(correct)

    # Number of images being classified.
    num_images = len(correct)

    # Print the accuracy.
    msg = "Accuracy on Test-Set: {0:.1%} ({1} / {2})"
    print(msg.format(acc, num_correct, num_images))

    # Plot some examples of mis-classifications, if desired.
    if show_example_errors:
        print("Example errors:")
        plot_example_errors(cls_pred=cls_pred, correct=correct)

    # Plot the confusion matrix, if desired.
    if show_confusion_matrix:
        print("Confusion Matrix:")
        plot_confusion_matrix(cls_pred=cls_pred)
```

from datetime import timedelta
optimize(num_iterations=1000)
Global Step: 13100, Training Batch Accuracy: 100.0%
Global Step: 13200, Training Batch Accuracy: 100.0%
Global Step: 13300, Training Batch Accuracy: 100.0%
Global Step: 13400, Training Batch Accuracy: 100.0%
Global Step: 13500, Training Batch Accuracy: 100.0%
Global Step: 13600, Training Batch Accuracy: 100.0%
Global Step: 13700, Training Batch Accuracy: 100.0%
Global Step: 13800, Training Batch Accuracy: 100.0%
Global Step: 13900, Training Batch Accuracy: 100.0%
Global Step: 14000, Training Batch Accuracy: 100.0%
Time usage: 0:00:36
print_test_accuracy(show_example_errors=True,
show_confusion_matrix=True)
Accuracy on Test-Set: 83.2% (277 / 333)
Example errors: Confusion Matrix:
[108 3 5] (0) Aamir Khan
[0 83 22] (1) Salman Khan
[4 22 86] (2) Shahrukh Khan
(0) (1) (2)

12.9 API

Banyak API yang mudah digunakan juga tersedia untuk tugas deteksi wajah dan pengenalan wajah. Berikut adalah beberapa contoh API deteksi wajah:

- PixLab
- Trueface.ai
- Kairos
- Microsoft Computer Vision

Berikut adalah beberapa contoh API pengenalan wajah:

- Face++
- LambdaLabs
- KeyLemon
- PixLab

Jika Anda menginginkan deteksi wajah, pengenalan wajah, dan analisis wajah dari satu penyedia, saat ini ada tiga raksasa besar yang memimpin di sini.

- Amazon's Amazon Recognition API
- Microsoft Azure's Face API
- IBM Watson's Visual Recognition API

Amazon Recognition API dapat melakukan empat jenis pengenalan.

- *Deteksi objek dan pemandangan*: Pengenalan mengidentifikasi berbagai objek menarik seperti kendaraan, hewan peliharaan, atau furnitur, dan memberikan skor kepercayaan.
- *Analisis wajah*: Anda dapat menemukan wajah di dalam gambar dan menganalisis atribut wajah, seperti apakah wajah tersenyum atau mata terbuka, dengan skor kepercayaan tertentu.
- *Perbandingan wajah*: Amazon Recognition API memungkinkan Anda mengukur kemungkinan bahwa wajah dalam dua gambar adalah orang yang sama. Sayangnya, ukuran kemiripan dua wajah orang yang sama bergantung pada usia saat foto. Juga, peningkatan lokal dalam iluminasi wajah mengubah hasil perbandingan wajah.
- *Pengenalan wajah*: API mengidentifikasi orang dalam gambar tertentu menggunakan repositori pribadi. Ini cepat dan akurat.

API Wajah Microsoft Azure akan mengembalikan skor kepercayaan untuk seberapa besar kemungkinan kedua wajah itu milik satu orang. Microsoft juga memiliki API lain seperti berikut ini:

- *Computer Vision API*: Fitur ini mengembalikan informasi tentang konten visual yang ditemukan dalam gambar. Itu dapat menggunakan penandaan, deskripsi, dan model khusus domain untuk mengidentifikasi konten dan memberi label dengan percaya diri.
- *Content Moderation API:* Ini mendeteksi gambar yang berpotensi menyinggung atau tidak diinginkan, teks dalam berbagai bahasa, dan konten video.
- Emotion API: Ini menganalisis wajah untuk mendeteksi berbagai perasaan dan mempersonalisasi respons aplikasi Anda.
- *Video API:* Ini menghasilkan output video yang stabil, mendeteksi gerakan, membuat thumbnail cerdas, dan mendeteksi serta melacak wajah.
- *Pengindeks Video:* Ini menemukan wawasan dalam video seperti entitas ucapan, polaritas sentimen ucapan, dan garis waktu audio.
- *Layanan Visi Kustom*: Ini menandai gambar baru berdasarkan model bawaan atau model yang dibuat melalui kumpulan data pelatihan yang Anda sediakan.

Visual Recognition API IBM Watson dapat melakukan beberapa deteksi spesifik seperti berikut ini:

- Dapat menentukan usia orang tersebut.
- Dapat menentukan jenis kelamin orang tersebut.
- Dapat menentukan lokasi kotak pembatas di sekitar wajah.
- Dapat mengembalikan informasi tentang selebriti yang terdeteksi dalam gambar. (Ini tidak dikembalikan ketika seorang selebriti tidak terdeteksi.)

BAB 13
LAMPIRAN

LAMPIRAN I: *Fungsi Keras untuk Pemrosesan Gambar*

Keras memiliki fungsi yang disebut ImageDataGenerator yang memberi Anda kumpulan data gambar tensor dengan augmentasi data waktu nyata. Data akan diulang dalam batch tanpa batas. Berikut adalah fungsinya:

```
keras.preprocessing.image.ImageDataGenerator
    (featurewise_center=False,
     samplewise_center=False,
     featurewise_std_normalization=False,
     samplewise_std_normalization=False,
     zca_whitening=False,
     zca_epsilon=1e-6,
     rotation_range=0.,
     width_shift_range=0.,
     height_shift_range=0.,
     shear_range=0.,
     zoom_range=0.,
     channel_shift_range=0.,
     fill_mode='nearest',
     cval=0.,
     horizontal_flip=False,
     vertical_flip=False,
     rescale=None,
     preprocessing_function=None,
     data_format=K.image_data_format())
```

Berikut adalah argumen fungsi:

- featurewise_center: Boolean tipe data. Tetapkan rata-rata input ke 0 di atas kumpulan data, dari segi fitur.
- samplewise_center: Boolean tipe data. Tetapkan rata-rata setiap sampel ke 0.
- featurewise_std_normalization: Tipe data boolean. Membagi input dengan std dari kumpulan data, berdasarkan fitur.
- samplewise_std_normalization: Tipe data boolean. Membagi setiap input dengan std-nya.
- zca_epsilon: Epsilon untuk pemutihan ZCA. Standarnya adalah 1e-6.
- zca_whitening: boolean. Menerapkan pemutih ZCA.
- rotation_range: int. Menetapkan derajat jangkauan untuk rotasi acak.
- width_shift_range: Tipe data float (fraksi dari total lebar). Menetapkan rentang untuk pergeseran horizontal acak.
- height_shift_range: Tipe data float (fraksi dari tinggi total). Menetapkan rentang untuk pergeseran vertikal acak.
- shear_range: Tipe data float. Mengatur intensitas geser (sudut geser berlawanan arah jarum jam sebagai radian).
- zoom_range: Tipe data float atau [Up, Down]. Mengatur rentang untuk zoom acak. Jika float, [down, up] = [1-zoom_range, 1+zoom_range].

- channel_shift_range: Tipe data float. Menetapkan rentang untuk pergeseran saluran acak.
- fill_mode: Salah satu dari {"constant", "terdekat", "reflect" atau "wrap"}. Poin di luar batas input diisi sesuai dengan mode yang diberikan.
- cval: Tipe data float atau int. Nilai digunakan untuk titik di luar batas saat fill_mode = "constant".
- horizontal_flip: Boolean tipe data. Secara acak membalik input secara horizontal.
- vertical_flip: Boolean tipe data. Secara acak membalik input secara vertikal.
- rescale: Faktor rescaling. Ini default ke Tidak Ada. Jika Tidak Ada atau 0, tidak ada penskalaan ulang yang diterapkan. Jika tidak, Anda mengalikan data dengan nilai yang diberikan (sebelum menerapkan transformasi lainnya).
- preprocessing_function: Fungsi yang akan diimplikasikan pada setiap input. Fungsi akan berjalan sebelum ada modifikasi lain di dalamnya. Fungsi harus mengambil satu argumen, sebuah gambar (tensor Numpy dengan peringkat 3), dan harus menampilkan tensor Numpy dengan bentuk yang sama.
- data_format: Salah satu dari {"channels_first", "channels_ last"}. Mode "channels_last" berarti bahwa gambar harus memiliki bentuk (sampel, tinggi, lebar, saluran). Mode "channels_first" berarti bahwa gambar harus memiliki bentuk (sampel, saluran, tinggi, lebar). Ini default ke nilai format image_data_ yang ditemukan di file konfigurasi Keras Anda di ~/.keras/keras.json. Jika Anda tidak mengaturnya, maka itu akan menjadi "channels_last".

Berikut metode-metodenya:

- fit(x): Menghitung statistik data internal yang terkait dengan transformasi yang bergantung pada data, berdasarkan larik data sampel. Ini diperlukan hanya jika itu adalah featurewise_ center atau featurewise_std_normalization atau zca_ whitening.
    - Berikut adalah argumen metode:
        - x: Data sampel. Ini harus memiliki peringkat 4. Dalam kasus data skala abu-abu, sumbu saluran harus memiliki nilai 1, dan dalam kasus data RGB, harus memiliki nilai 3.
        - augment: Tipe data boolean (default: False). Ini menetapkan apakah akan cocok dengan sampel yang ditambah secara acak.
        - rounds: Tipe data int (default: 1). Jika augment disetel, ini menetapkan berapa banyak augmentasi yang melewati data yang akan digunakan.
        - seed: Tipe data int (default: Tidak ada). Menetapkan benih acak.
- flow(x, y): Mengambil data Numpy dan melabeli array dan menghasilkan kumpulan data yang diperbesar/dinormalisasi. Menghasilkan batch tanpa batas, dalam loop tak terbatas.
    - Berikut argumennya:
        - x: Data. Ini harus memiliki peringkat 4. Dalam kasus data skala abu-abu, sumbu saluran harus memiliki nilai 1, dan dalam kasus data RGB, harus memiliki nilai 3.
        - y: Label.
        - batch_size: Tipe data int (default: 32).
        - shuffle: Tipe data boolean (default: True).
        - seed: Tipe data int (default: Tidak ada).
        - save_to_dir: Tidak ada atau str (default: Tidak ada). Ini memungkinkan Anda untuk secara optimal menentukan direktori tempat menyimpan

gambar tambahan yang dihasilkan (berguna untuk memvisualisasikan apa yang Anda lakukan).
- save_prefix: Tipe data str (default: ''). Ini adalah awalan yang digunakan untuk nama file gambar yang disimpan (hanya relevan jika save_to_dir diatur).
- save_format: Baik png atau jpeg (hanya relevan jika save_to_dir disetel). Standar: png.
- menghasilkan: Tuple dari (x, y) di mana x adalah array Numpy dari data gambar dan y adalah array Numpy dari label yang sesuai. Generator berputar tanpa batas

LAMPIRAN II: Beberapa Kumpulan Data Gambar Teratas yang Tersedia

- **MNIST**: Mungkin kumpulan data gambar paling terkenal yang tersedia untuk Anda, kumpulan data ini disusun oleh Yann LeCun dan tim. Kumpulan data ini digunakan hampir di mana-mana sebagai tutorial atau pengantar dalam visi komputer. Ini memiliki sekitar 60.000 gambar pelatihan dan sekitar 10.000 gambar uji.
- **CIFAR-10**: Kumpulan data ini dibuat sangat terkenal oleh tantangan ImageNet. Ini memiliki 60.000 gambar 32 × 32 dalam 10 kelas, dengan 6.000 gambar per kelas. Ada 50.000 gambar pelatihan dan 10.000 gambar uji.
- **ImageNet:** Database gambar objek berlabel ini digunakan dalam Tantangan Pengenalan Visual Skala Besar ImageNet. Ini termasuk objek berlabel, kotak pembatas, kata-kata deskriptif, dan fitur SIFT. Ada total 14.197.122 kasus.
- **MS COCO**: Kumpulan data Microsoft Common Objects in CONtext (MS COCO) berisi 91 kategori objek umum, dengan 82 di antaranya memiliki lebih dari 5.000 instans berlabel. Secara total, kumpulan data memiliki 2.500.000 instans berlabel dalam 328.000 gambar. Berbeda dengan kumpulan data ImageNet yang populer, COCO memiliki lebih sedikit kategori tetapi lebih banyak contoh per kategori. COCO adalah kumpulan data pendeteksian, segmentasi, dan keterangan objek berskala besar.
- **10k US Adult Faces:** Kumpulan data ini berisi 10.168 foto fase alami dan beberapa ukuran untuk 2.222 wajah, termasuk skor daya ingat, visi komputer dan atribut fisik, serta anotasi titik tengara.
- **Flickr 32/47 Brands Logos:** Ini terdiri dari gambar dunia nyata yang dikumpulkan dari Flickr logo perusahaan dalam berbagai keadaan. Muncul dalam dua versi: kumpulan data 32-merek dan kumpulan data 47-merek. Ada total 8.240 gambar.
- **YouTube Faces**: Ini adalah database video wajah yang dirancang untuk mempelajari masalah pengenalan wajah tanpa batas dalam video. Kumpulan data berisi 3.425 video dari 1.595 orang yang berbeda.
- **Caltech Pedestrian:** Kumpulan data Caltech Pedestrian terdiri dari sekitar 10 jam video 640x480 30Hz yang diambil dari kendaraan yang mengemudi melalui lalu lintas reguler di lingkungan perkotaan. Sekitar 250.000 bingkai (dalam 137 segmen berdurasi kira-kira satu menit) dengan total 350.000 kotak pembatas dan 2.300 pejalan kaki unik diberi anotasi.
- **PASCAL VOC**: Ini adalah kumpulan data yang sangat besar untuk tugas klasifikasi gambar. Ini memiliki 500.000 contoh data.
- **Microsoft Common Objects in Context (COCO):** Ini berisi adegan sehari-hari yang kompleks dari objek umum dalam konteks alaminya. Penyorotan, pelabelan, dan klasifikasi objek menjadi 91 jenis objek. Ini berisi 2.500.000 instance.

- **Caltech-256:** Ini adalah kumpulan data besar gambar untuk klasifikasi objek. Gambar dikategorikan dan diurutkan dengan tangan. Ada total 30.607 gambar.
- **FBI crime data set:** Kumpulan data kejahatan FBI luar biasa. Jika Anda tertarik dengan analisis data deret waktu, Anda dapat menggunakannya untuk merencanakan perubahan tingkat kejahatan di tingkat nasional selama periode 20 tahun.

LAMPIRAN III: Pencitraan Medis: Format File DICOM

Digital Imaging and Communication in Medicine (DICOM) adalah jenis format file yang digunakan dalam domain medis untuk menyimpan atau mentransfer gambar yang diambil selama berbagai tes pada beberapa pasien.

Mengapa DICOM?

MRI, CT scan, dan sinar-X dapat disimpan dalam format file normal, tetapi karena keunikan laporan medis, berbagai jenis data diperlukan untuk gambar tertentu.

Apa itu Format File DICOM?

Format file ini berisi header yang terdiri dari metadata gambar seperti nama pasien, ID, golongan darah, dan sebagainya. Ini juga berisi nilai piksel yang dipisahkan ruang dari gambar yang diambil selama berbagai tes medis.

Standar DICOM adalah format file kompleks yang dapat ditangani oleh paket-paket berikut:

- pydicom: Ini adalah paket untuk bekerja dengan gambar dengan Python. dicom adalah versi yang lebih lama dari paket ini. Pada tulisan ini, pydicom 1.x adalah versi terbaru.
- oro.dicom: Ini adalah paket untuk bekerja dengan gambar di R.

File DICOM direpresentasikan sebagai FileName.dcm

```
import dicom

ds = dicom.read_file("E:/datasciencebowl/stage1/00cba091fa4ad62cc3200a657aeb957e/0a291d1b12b86213d813e3796f14b329.dcm")
```

```
(0008, 0005) Specific Character Set             CS: 'ISO_IR 100'
(0008, 0016) SOP Class UID                      UI: CT Image Storage
(0008, 0018) SOP Instance UID                   UI: 1.2.840.113654.2.55.158283083714550104456272463610634335 9
(0008, 0060) Modality                           CS: 'CT'
(0008, 103e) Series Description                 LO: 'Axial'
(0010, 0010) Patient's Name                     PN: '00cba091fa4ad62cc3200a657aeb957e'
(0010, 0020) Patient ID                         LO: '00cba091fa4ad62cc3200a657aeb957e'
(0010, 0030) Patient's Birth Date               DA: '19000101'
(0018, 0060) KVP                                DS: ''
(0020, 000d) Study Instance UID                 UI: 2.25.862087301405397123827718905017727342779506923977090073054 73
(0020, 000e) Series Instance UID                UI: 2.25.115758773296352289258085968002699747408935194517846260466 14
(0020, 0011) Series Number                      IS: '3'
(0020, 0012) Acquisition Number                 IS: '1'
(0020, 0013) Instance Number                    IS: '88'
(0020, 0020) Patient Orientation                CS: ''
(0020, 0032) Image Position (Patient)           DS: ['-145.500000', '-158.199997', '-241.199997']
(0020, 0037) Image Orientation (Patient)        DS: ['1.000000', '0.000000', '0.000000', '0.000000', '1.000000', '0.000
']
(0020, 0052) Frame of Reference UID             UI: 2.25.830335096344416863856520734629838018401219166784177196696 50
(0020, 1040) Position Reference Indicator       LO: 'SN'
(0020, 1041) Slice Location                     DS: '-241.199997'
(0028, 0002) Samples per Pixel                  US: 1
(0028, 0004) Photometric Interpretation         CS: 'MONOCHROME2'
(0028, 0010) Rows                               US: 512
(0028, 0011) Columns                            US: 512
(0028, 0030) Pixel Spacing                      DS: ['0.597656', '0.597656']
(0028, 0100) Bits Allocated                     US: 16
```

DAFTAR PUSTAKA

Bejiga, M. B., Zeggada, A., Nouffidj, A., & Melgani, F. (2017). *A convolutional neural network approach for assisting avalanche search and rescue operations with UAV imagery.* Remote Sensing, 9(2). https://doi.org/10.3390/rs9020100

Devikar, P. 2016. Transfer Learning for Image Classification of Various Dog Breeds. *International Journal of Advanced Research in Computer Engineering and Technology* (IJARCET), Vol.5: 2707-2715.

Fahriza Azwar Muhammad, Rizky Arif Windiator,Yuridi Bintang Pratama, "pemrograman socket untuk koneksi Abtara Raspberry Pi dengan Referee Box",*Universitas Islam Indonesia*, 2016.

Fitri, Kiki Reski R, Ady Rahmansyah, dan Wahyuni. *Penggunaan Bahasa Pemrograman Python Sebagai Pusat Kendali Pada Robot 10-D*, 2017.

H. Abhirawa, Jondri, dan A. Arifianto, "Pengenalan wajah menggunakan convolutional neural network," Dalam *e-Proceeding of Engineering*, 2017.

K. P. Danukusumo, "*Implementasi deep learning menggunakan convolutional neural network untuk klasifikasi citra candi berbasis GPU*," Skripsi, Universitas Atma Jaya Yogyakarta, Yogyakarta, 2017.

Kadir, A. 2018. Dasar Logika Pemrograman Komputer. Cetakan Kedua. Elexmedia Komputindo.

Kim, J., Sangjun, O., Kim, Y., & Lee, M. (2016). Convolutional Neural Network with Biologically Inspired Retinal Structure. Procedia Computer Science, 88, 145–154. https://doi.org/10.1016/j.procs.2016.07.418

Krizhevsky, A., Sutskever I., & Hinton G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of NIPS.

Ravi, S. & Nayeem, S. (2013). A Study on Face Recognition Technique based on Eigenface. Foundation of Computer Science FCS, New York, USA Volume 5– No.4. International Journal of Applied Information Systems (IJAIS) – ISSN : 2249-0868.

S. R. Dewi, "Deep learning object detection pada video menggunakan tensorflow dan convolutional network," Skripsi, Universitas Islam Indonesia, Yogyakarta, 2018.

Suartika E. P, Arya Yudhi Wijaya Wijaya, dan Rully Soelaiman . "Klasifikasi Citra Menggunakan Convolutional Neural Network (Cnn) pada Caltech 101." 2016.

Wahyono, Teguh. Fundamental Of Python For Mechibe Learning. Yogyakarta: Gava Media, 2018.

Yuliza, IncomTech, Jurnal Telekomunikasi dan Komputer, vol.4, no.1,2013.

Zhi, T., Duan, L. Y., Wang, Y., & Huang, T. (2016). Two- stage pooling of deep convolutional features for image retrieval. In 2016 IEEE International Conference on Image Processing (ICIP) (hal. 2465– 2469). https://doi.org/10.1109/ICIP.2016.7532802

Zufar, M. & Setiyono B. (2016). Convolutional Neural Networks untuk Pengenalan Wajah Secara Real-Time. JURNAL SAINS DAN SENI ITS Vol. 5 No. 2 . A-72.