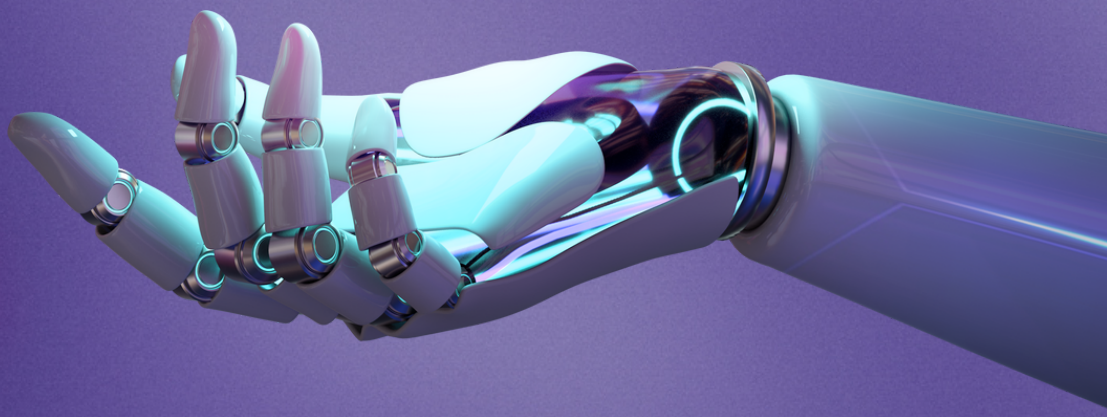
MODUL

SISTEM CERDAS



PENULIS

FATHORAZI NUR FAJRI

YAYASAN NURUL JADID PAITON
FAKULTAS TEKNIK
UNIVERSITAS NURUL JADID
PROBOLINGGO JAWA TIMUR

PP. Nurul Jadid
Karanganyar Paiton
Probolinggo 67291
0335 771732
ft@unuja.ac.id

LEMBAR PENGESAHAN

MODUL KULIAH

SISTEM CERDAS

Oleh

Fathorazi Nur Fajri, M.Kom

NIDN. 0713039301

Modul Ini Disusun Sebagai Pedoman Dan Acuan Dalam Pelaksanaan Kuliah

Sistem Cerdas

Program Studi Sistem Informasi Fakultas Teknik Universitas Nurul Jadid

DINYATAKAN DAPAT DIGUNAKAN

Disahkan Pada tanggal 15 September 2022

Mengetahui	
Ketua Program Studi	Dosen Pengampu
Sistem Informasi	Pemrograman Perangkat Bergerak
Abu Tholib, M.Kom	**Fathorazi Nur Fajri, M.Kom**

Kata pengantar

Modul ini membahas tentang semua topik yang berkaitan dengan kecerdasan buatan dan pembelajaran mesin dalam mendukung proses kegiatan belajar mengajar pada matakuliah sistem cerdas.
Topik yang Dicakup:

- Konsep Kecerdasan Buatan
- Mencari
- Teori Keputusan
- Pembelajaran Penguatan
- Jaringan Syaraf Tiruan
- Perbanyakan balik
- Ekstraksi Fitur
- Pembelajaran Mendalam
- Jaringan Syaraf Konvolusional
- Pembelajaran Penguatan Mendalam
- Pembelajaran Terdistribusi
- Pustaka pembelajaran mendalam Python/Matlab

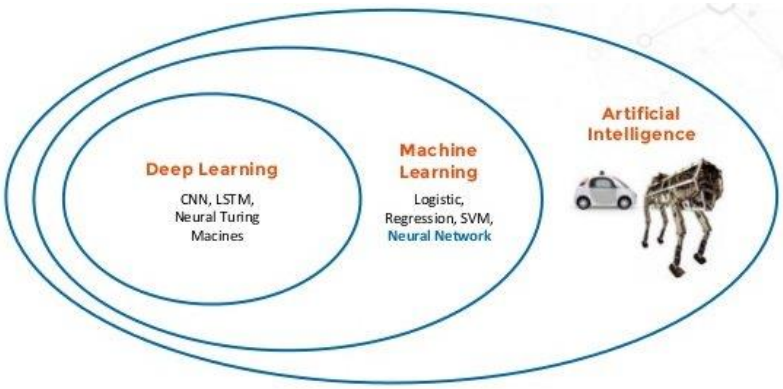
Daftar isi

Perkenalan

Buku ini akan membahas dasar-dasar yang diperlukan untuk mengimplementasikan dan memahami perpustakaan Kecerdasan Buatan dan Pembelajaran Mesin Anda sendiri. Semua rumus dan konsep akan disajikan dengan kode di Matlab dan Python.

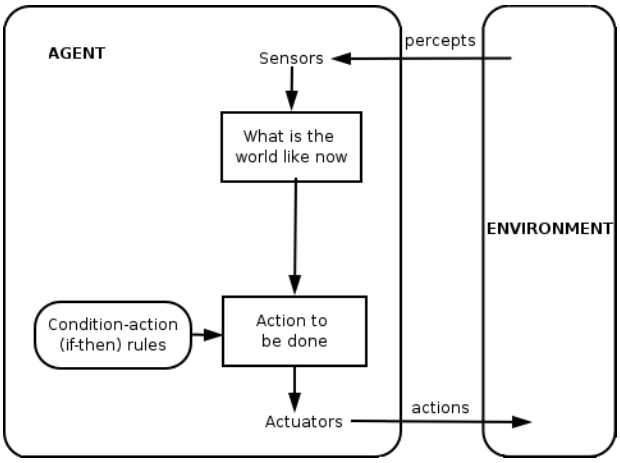
Kecerdasan Buatan dan Pembelajaran Mendalam

Hari ini kami memiliki banyak kebingungan seputar kecerdasan buatan, pembelajaran mesin, dan pembelajaran mendalam. Sebenarnya istilah-istilah tersebut hanyalah bagian dari Artificial Intelligence.



Apa itu Kecerdasan Buatan

Bidang studi yang mempelajari bagaimana membuat sistem komputasi yang mampu berperilaku cerdas. Beberapa teks lain mendefinisikan sebagai studi / desain agen cerdas. Di sini agen adalah sistem (Perangkat Lunak/Perangkat Keras) yang memahami lingkungannya dan mengambil tindakan yang memaksimalkan peluang keberhasilannya.



Definisi intelijen

Untuk ruang lingkup buku ini, agen cerdas adalah agen yang memecahkan masalah secara optimal, artinya sistem akan mencari tahu sendiri tindakan terbaik yang harus diambil.

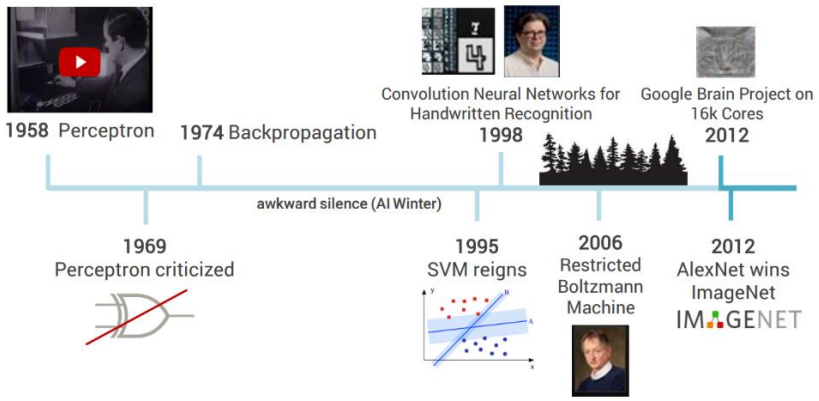
Cara untuk mengambil keputusan yang cerdas

- Lacak semua tindakan Anda dan periksa apakah itu baik atau buruk, lalu bandingkan tindakan baru dengan salah satunya.
- Sebelum mengambil tindakan, simulasikan semua hasil yang mungkin (baik atau buruknya tindakan) lalu pilih yang kurang buruk. Jadi Anda membutuhkan abstraksi (model) dunia, ingatlah bahwa model dunia bukanlah dunia.

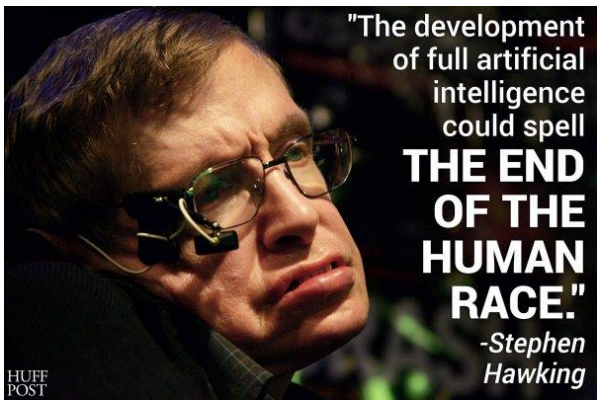
Fakta lucu tentang kecerdasan buatan adalah bahwa setelah masalah diselesaikan sepenuhnya, itu tidak disebut cerdas lagi ... (yaitu: Membuat komputer bermain catur adalah tampilan kecerdasan tertinggi, sekarang orang tidak lagi menganggap itu)

Sejarah

Pada dasarnya melalui sejarah kecerdasan buatan kami mengalami beberapa periode kejutan/harapan dan kekecewaan.



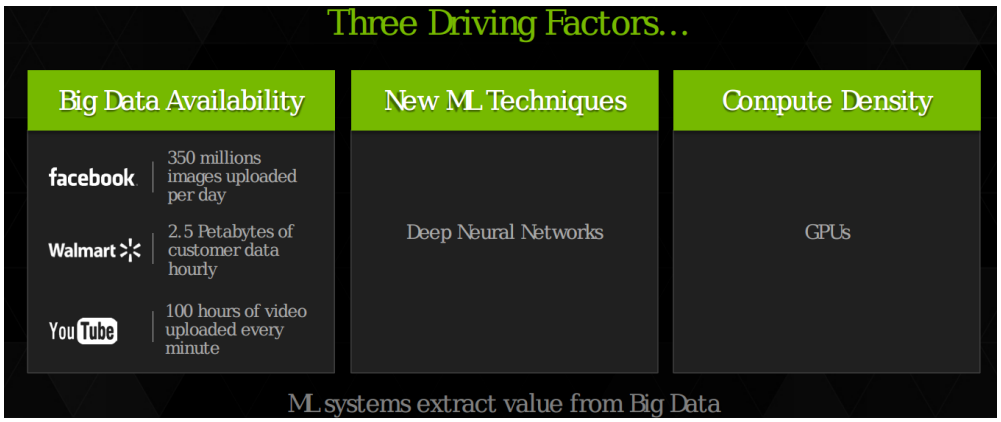
Fakta lucunya adalah bahwa sekarang kita berada dalam campuran antara hype/fear



Kenapa sekarang

Algoritme pembelajaran mesin (Bahkan yang dalam) sudah ada selama beberapa dekade, jadi mengapa sekarang kita memiliki kata kunci ini?

Pada dasarnya karena kemajuan daya komputasi melalui (GPU, sistem CPU multi-core, dan FPGA) dan ketersediaan data (Big data) melalui internet. Juga jumlah data yang perlu diklasifikasikan saat ini menjadi terlalu besar untuk ditangani secara manual, sehingga perusahaan besar Google, Microsoft, Facebook, mulai berinvestasi besar-besaran pada subjek tersebut.



Batch Size	Training Time CPU	Training Time GPU	GPU Speed Up
64 images	64 s	7.5 s	8.5X
128 images	124 s	14.5 s	8.5X
256 images	257 s	28.5 s	9.0X

Hype baru

Tahun-tahun terakhir 2013/2016 kecerdasan buatan (Pembelajaran mesin) mengejutkan orang dengan hasil yang lebih dekat atau terkadang lebih baik daripada manusia. Misalnya:

- Pemrosesan ucapan dan bahasa alami
- Pengenalan Wajah
- Klasifikasi Gambar, deteksi objek
- Mengemudi Mobil
- Memainkan game yang rumit (Alpha Go)
- Strategi kontrol (Rekayasa kontrol)

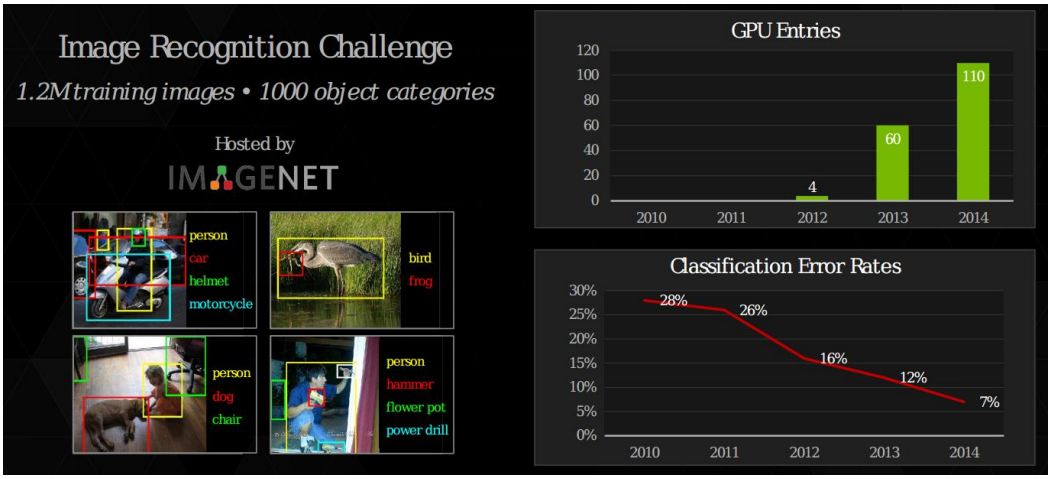
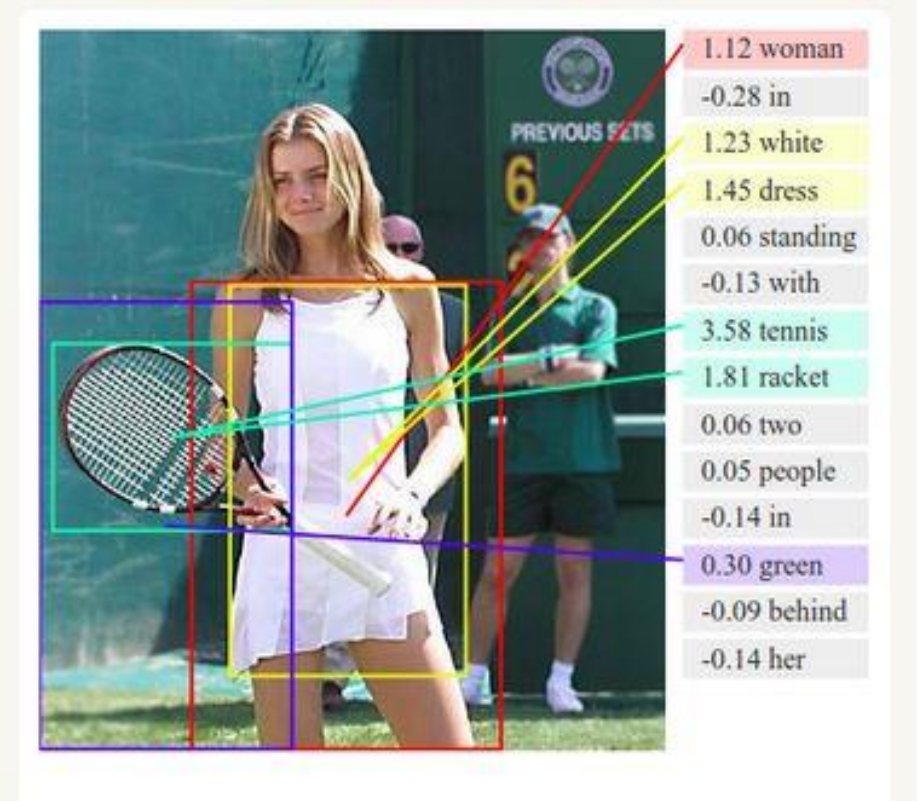
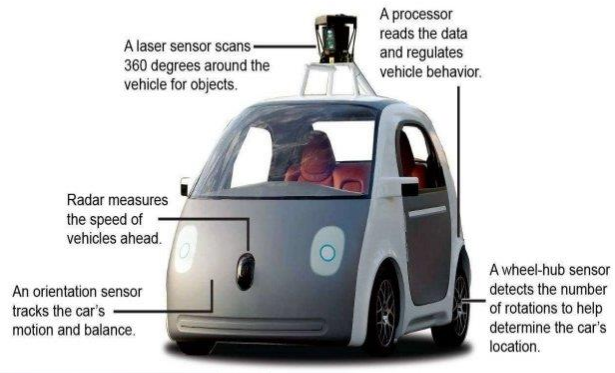


Image Recognition Challenge
1.2M training images • 1000 object categories
Hosted by
IMAGENET
person
car
helmet
motorcycle
bird
frog
person
dog
chair
person
hammer
flower pot
power drill
GPU Entries
120
100
80
60
40
20
0
2010
2011
2012
2013
2014
4
60
110
Classification Error Rates
30%
25%
20%
15%
10%
5%
0%
28%
26%
16%
12%
7%


1.12 woman
-0.28 in
1.23 white
1.45 dress
0.06 standing
-0.13 with
3.58 tennis
1.81 racket
0.06 two
0.05 people
-0.14 in
0.30 green
-0.09 behind
-0.14 her


A laser sensor scans 360 degrees around the vehicle for objects.
A processor reads the data and regulates vehicle behavior.
Radar measures the speed of vehicles ahead.
An orientation sensor tracks the car's motion and balance.
A wheel-hub sensor detects the number of rotations to help determine the car's location.

Jadi pada dasarnya orang mulai takut kehilangan pekerjaan dan beberapa server kecerdasan buatan mengambil alih dunia.

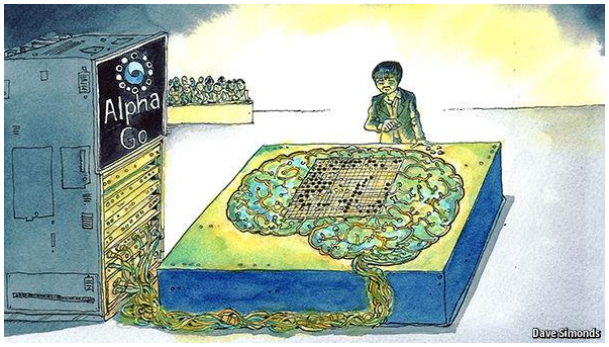
Perbandingan daya komputasi

Pada tabel di bawah ini kami menyajikan tabel dengan jumlah kemungkinan operasi per detik dan biaya beberapa platform perangkat keras

Type	Name	Flops	Cost
Mobile	Raspberry Pi 1st Gen, 700 Mhz	0,04 Gflops	$35
Mobile	Apple A8	1,4 Gflops	$700 (in iPhone 6)
CPU	Intel Core i7-4930K (Ivy Bridge), 3.7 GHz	140 Gflops	$700
CPU	Intel Core i7-5960X (Haswell), 3.0 GHz	350 Gflops	$1300
GPU	NVidia GTX 980	4612 Gflops (single precision), 144 Gflops (double precision)	$600 + cost of PC (~$1000)
GPU	NVidia Tesla K80	8740 Gflops (single precision), 2910 Gflops (double precision)	$4500 + cost of PC (~1500)

Perangkat keras Deepmind

Sekadar ilustrasi, gambar di bawah adalah perangkat keras yang digunakan untuk bermain melawan salah satu pemain Go terbaik di dunia.



Pembelajaran mesin

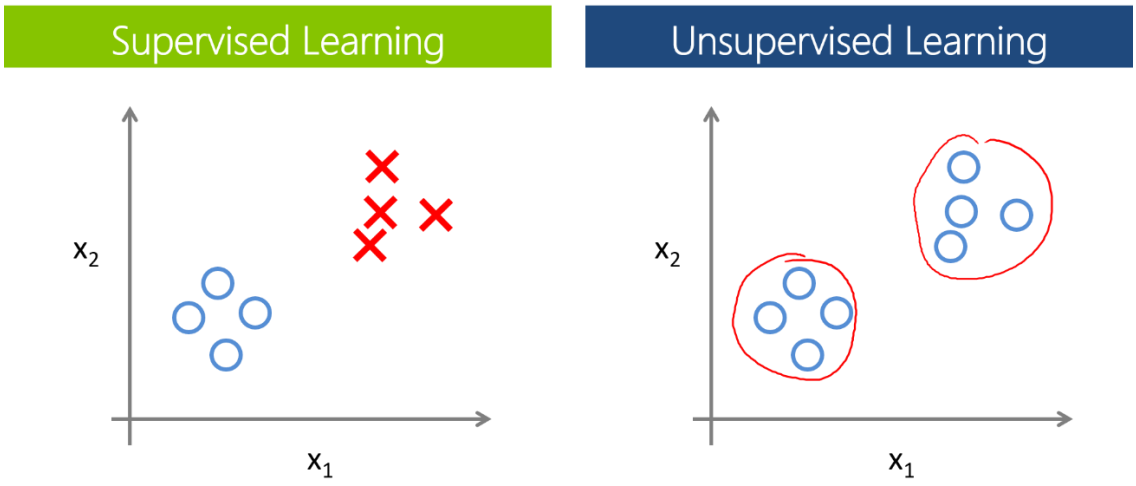
Pembelajaran mesin adalah tentang menggunakan komputer Anda untuk "belajar" bagaimana menghadapinya masalah tanpa "pemrograman". (Ini adalah cabang kecerdasan buatan)

Kami mengambil beberapa data, melatih model pada data tersebut, dan menggunakan model yang dilatih untuk membuat prediksi pada data baru. Pada dasarnya adalah cara untuk membuat komputer membuat sebuah program yang memberikan beberapa output dengan input yang diketahui dan yang terakhir memberikan yang cerdas output ke input yang berbeda tetapi serupa.

Kami membutuhkan pembelajaran mesin pada kasus-kasus yang akan sulit diprogram dengan tangan semua varian yang mungkin dari masalah klasifikasi/prediksi

Ide dasar Machine Learning adalah membuat komputer mempelajari sesuatu dari data. Pembelajaran mesin hadir dalam dua rasa:

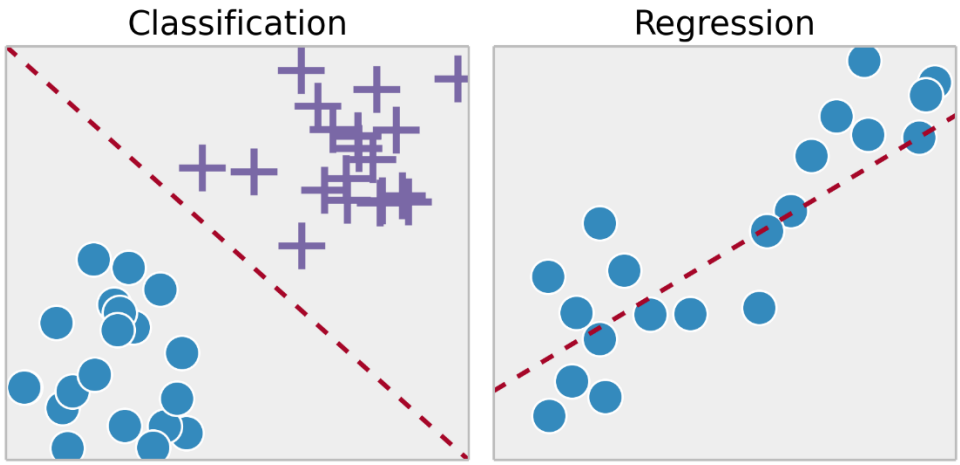
- Pembelajaran yang Diawasi: Anda memberi komputer beberapa pasang input/output, jadi di masa mendatang baru ketika input baru disajikan, Anda memiliki output yang cerdas.
- Pembelajaran Tanpa Pengawasan: Anda membiarkan komputer belajar dari data itu sendiri tanpa menunjukkan hasil yang diharapkan.



Contoh Pembelajaran yang Diawasi

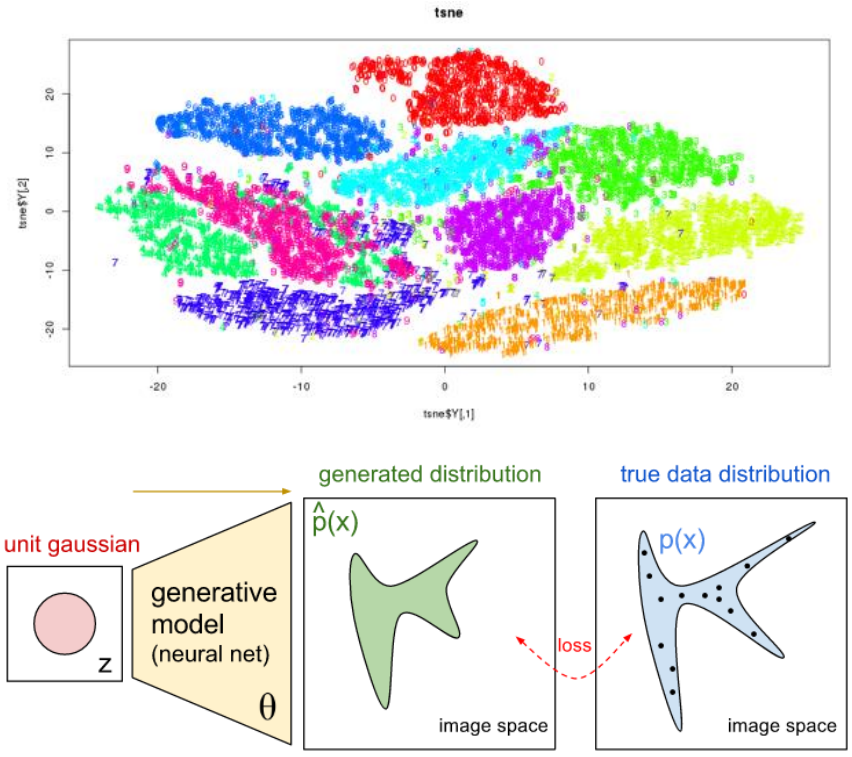
1. Klasifikasi Gambar: Kereta Anda dengan gambar/label. Kemudian di masa mendatang Anda memberikan gambar baru dengan harapan komputer akan mengenali objek baru tersebut (Klasifikasi)

2. Prediksi Pasar: Anda melatih komputer dengan data pasar historis dan meminta komputer memprediksi harga baru di masa depan (Regresi)



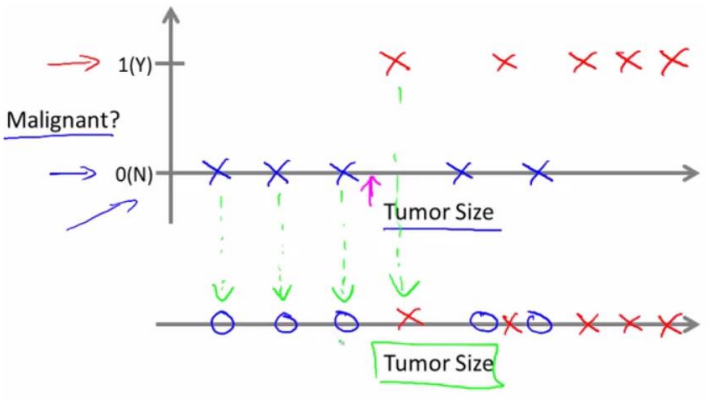
Contoh Pembelajaran Tanpa Pengawasan

1. Pengelompokan: Anda meminta komputer untuk memisahkan data serupa ke dalam kelompok, ini penting dalam penelitian dan sains.
2. Visualisasi Dimensi Tinggi: Gunakan komputer untuk membantu kami memvisualisasikan data dimensi tinggi.
3. Model Generatif: Setelah model menangkap distribusi probabilitas dari data input Anda, model tersebut akan dapat menghasilkan lebih banyak data. Ini bisa sangat berguna untuk membuat classifier Anda lebih kuat.

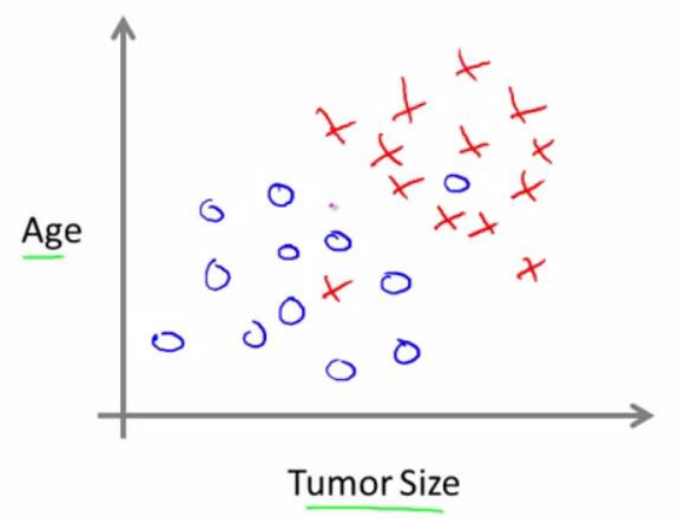


Fitur

Bayangkan masalah berikut, Anda sedang mengerjakan sistem yang harus mengklasifikasikan apakah tumor itu jinak atau ganas, pada awalnya satu-satunya informasi yang Anda miliki untuk membuat keputusan adalah ukuran tumornya. Distribusi data pelatihan untuk contoh ini dapat kita lihat di bawah ini. Perhatikan bahwa karakteristik (atau ciri) ukuran tumor tampaknya bukan merupakan indikator yang baik untuk memutuskan apakah tumor itu ganas atau jinak.



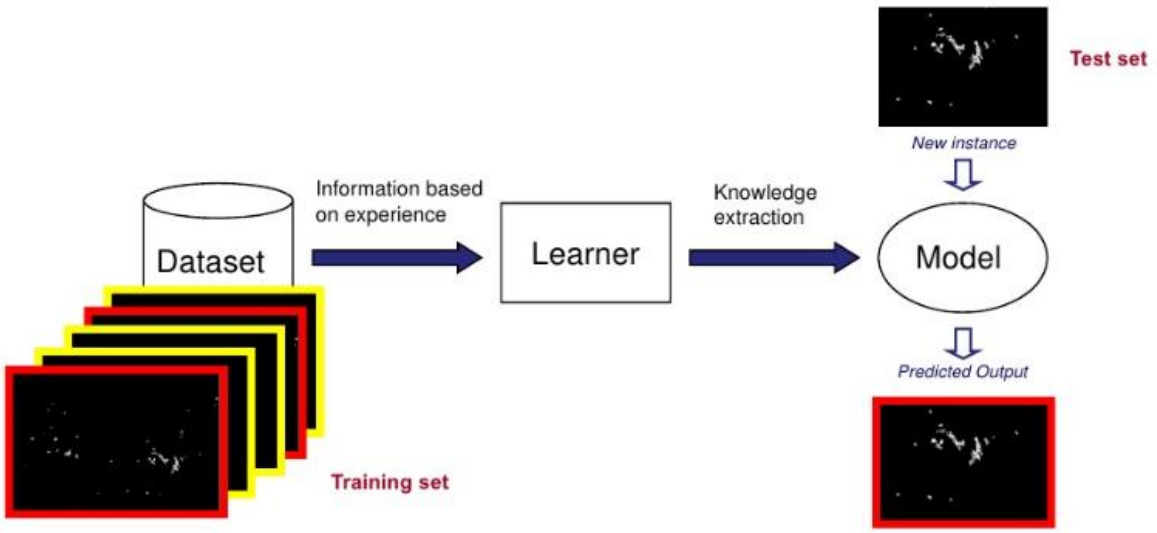
Sekarang pertimbangkan bahwa kami menambahkan satu fitur lagi ke masalah (Usia).



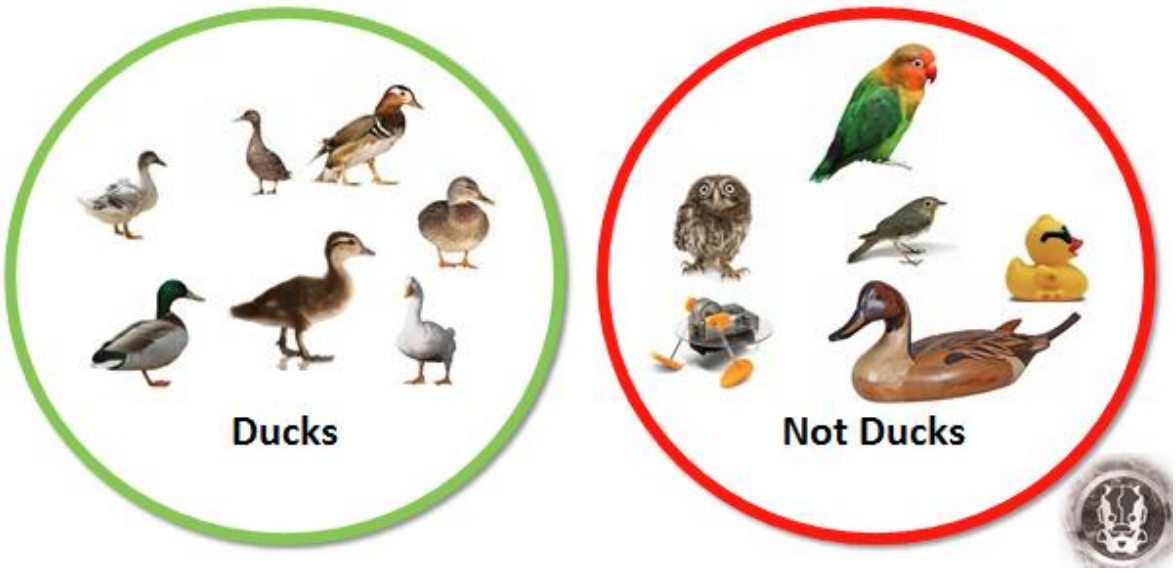
Intuisinya adalah dengan menambahkan lebih banyak fitur yang relevan dengan masalah yang ingin Anda selesaikan, Anda akan membuat sistem Anda lebih kuat. Sistem kompleks seperti ini bisa memiliki hingga ribuan fitur. Satu pertanyaan yang mungkin Anda tanyakan adalah bagaimana cara menentukan fitur apa yang relevan dengan masalah saya. Selain itu, algoritme mana yang paling baik digunakan untuk mengatasi jumlah fitur yang tak terbatas, misalnya Support Vector Machines memiliki beberapa trik matematis yang memungkinkan Anda menggunakan fitur dalam jumlah yang sangat besar.

Pelatihan

Idenya adalah untuk memberikan satu set input dan output yang diharapkan, jadi setelah pelatihan kita akan memiliki model (hipotesis) yang kemudian akan memetakan data baru ke salah satu kategori yang dilatih.

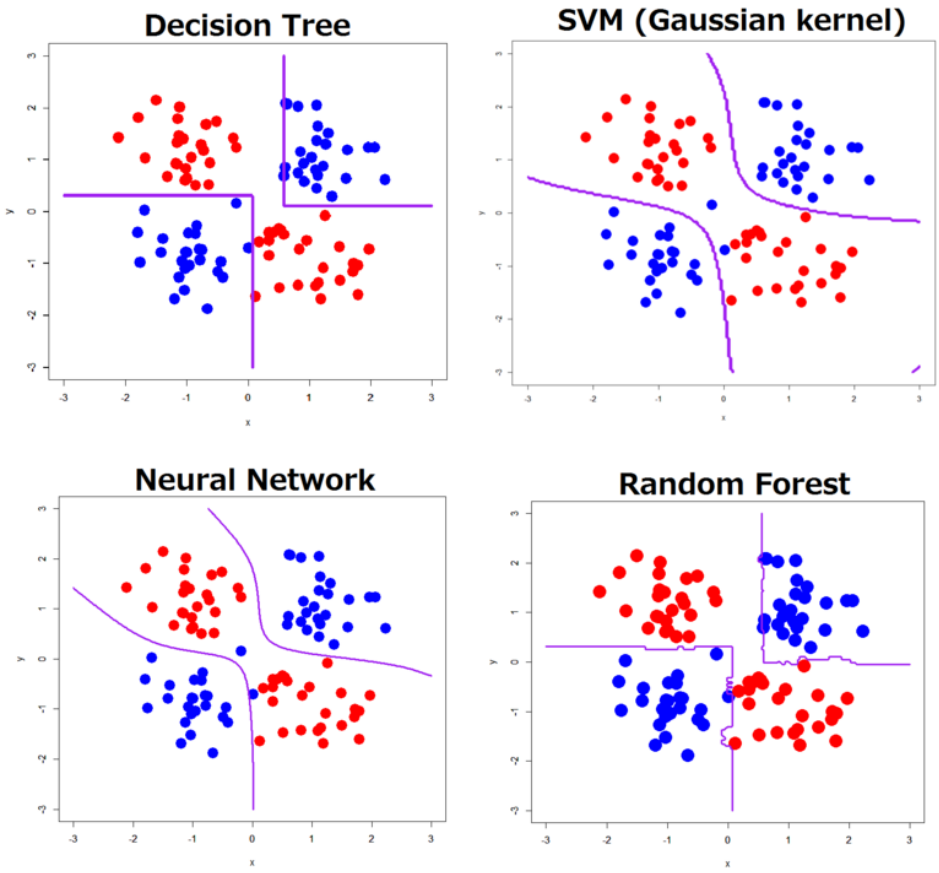


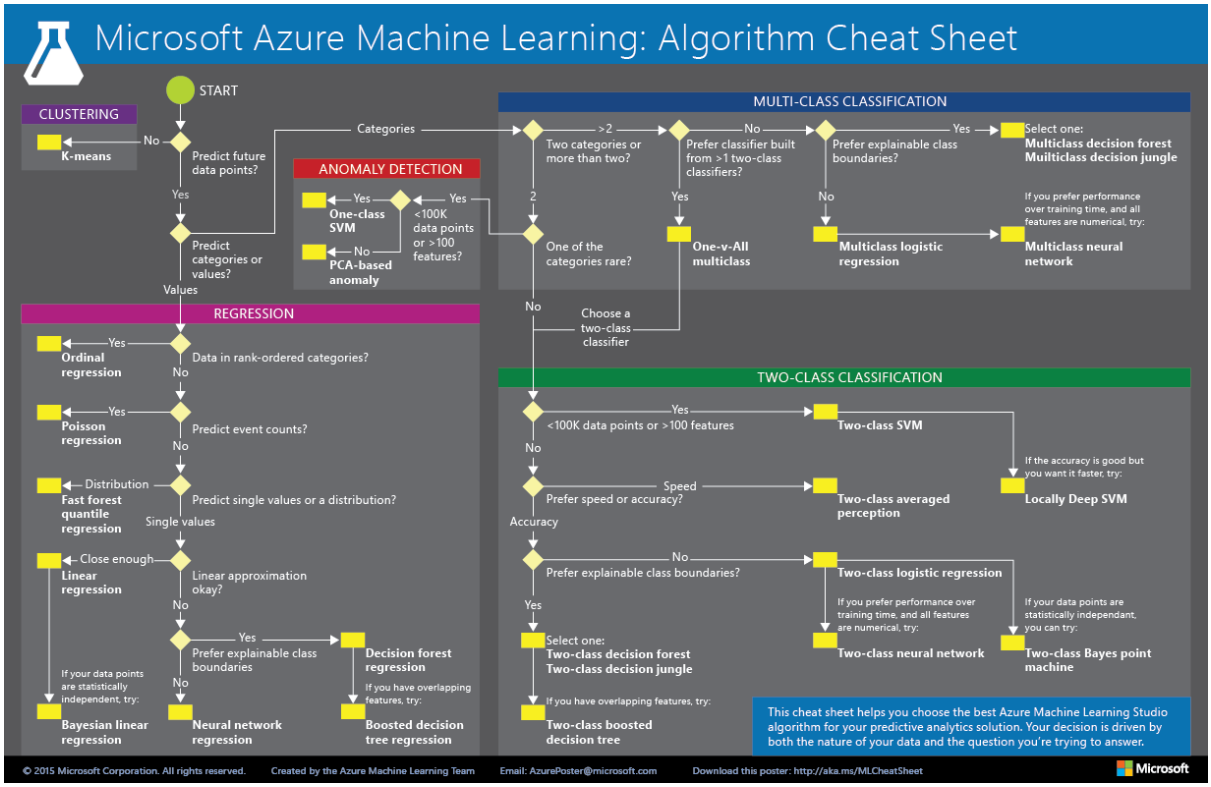
Mis: Bayangkan Anda memberikan satu set gambar dan kategori berikut, bebek atau bukan bebek, idenya adalah setelah pelatihan Anda bisa mendapatkan gambar bebek dari internet dan model tersebut akan memberi tahu Anda bahwa itu adalah "bebek".



Kantong trik

Ada banyak algoritme pembelajaran mesin yang berbeda, dalam buku ini kita akan lebih berkonsentrasi pada jaringan saraf, tetapi tidak ada satu pun algoritme terbaik, semuanya tergantung pada masalah yang perlu Anda selesaikan dan jumlah data yang tersedia.


Decision Tree
SVM (Gaussian kernel)
Neural Network
Random Forest

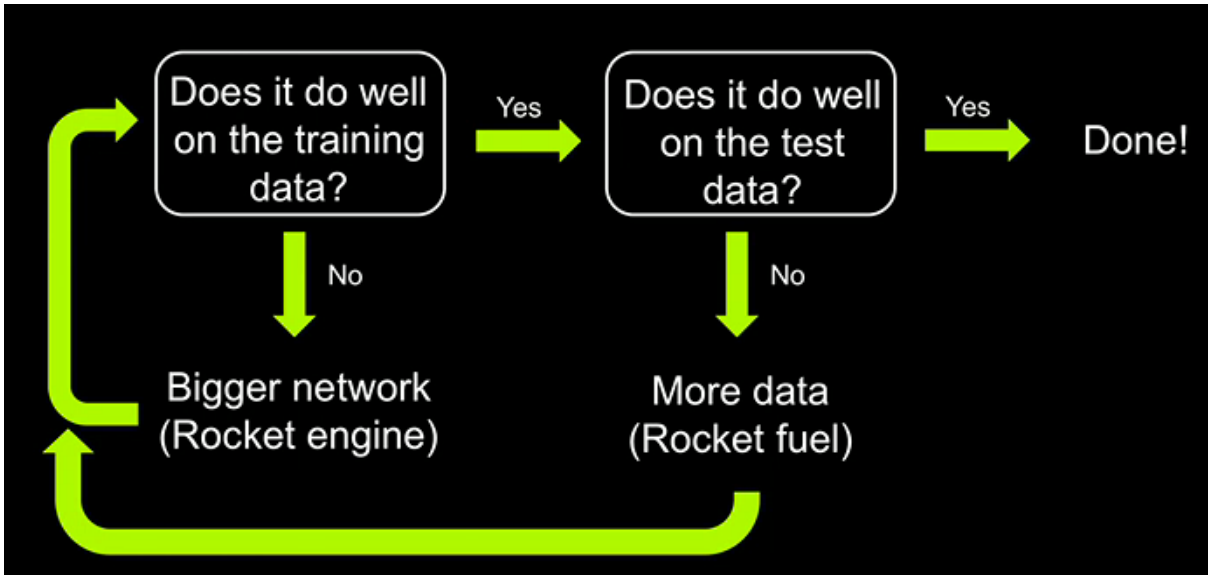

Microsoft Azure Machine Learning: Algorithm Cheat Sheet
START
CLUSTERING
K-means
No
Predict future data points?
Yes
Predict categories or values?
Values
Categories
ANOMALY DETECTION
Yes
One-class SVM
Yes
<100K data points or >100 features?
No
PCA-based anomaly
REGRESSION
Yes
Ordinal regression
Data in rank-ordered categories?
No
Yes
Poisson regression
Predict event counts?
No
Distribution
Fast forest quantile regression
Predict single values or a distribution?
Single values
Close enough
Linear regression
Linear approximation okay?
No
Yes
Prefer explainable class boundaries
Decision forest regression
If you have overlapping features, try:
Boosted decision tree regression
No
If your data points are statistically independent, try:
Bayesian linear regression
Neural network regression
MULTI-CLASS CLASSIFICATION
>2
Two categories or more than two?
No
Prefer classifier built from >1 two-class classifiers?
Yes
Prefer explainable class boundaries?
Yes
Select one:
Multiclass decision forest
Multiclass decision jungle
2
One of the categories rare?
Yes
One-v-All multiclass
No
Multiclass logistic regression
If you prefer performance over training time, and all features are numerical, try:
Multiclass neural network
No
Choose a two-class classifier
TWO-CLASS CLASSIFICATION
Yes
<100K data points or >100 features
Two-class SVM
No
Prefer speed or accuracy?
Speed
Accuracy
Two-class averaged perception
If the accuracy is good but you want it faster, try:
Locally Deep SVM
No
Prefer explainable class boundaries?
Two-class logistic regression
Yes
Select one:
Two-class decision forest
Two-class decision jungle
If you prefer performance over training time, and all features are numerical, try:
Two-class neural network
If your data points are statistically independant, you can try:
Two-class Bayes point machine
If you have overlapping features, try:
Two-class boosted decision tree
This cheat sheet helps you choose the best Azure Machine Learning Studio algorithm for your predictive analytics solution. Your decision is driven by both the nature of your data and the question you're trying to answer.
© 2015 Microsoft Corporation. All rights reserved.
Created by the Azure Machine Learning Team
Email: AzurePoster@microsoft.com
Download this poster: http://aka.ms/MLCheatSheet
Microsoft

Resep Dasar

Ini adalah resep yang sangat sederhana (mungkin mencakup 50% dari kemungkinan kasus), kami akan menjelaskan "bagaimana" nanti, tetapi ini memberikan beberapa petunjuk tentang cara berpikir saat menangani masalah pembelajaran mesin.

- Pertama periksa apakah model Anda berfungsi dengan baik pada data pelatihan, dan jika tidak, buat model lebih kompleks (Lebih dalam, atau lebih banyak neuron)
- Jika ya maka tes pada data “tes”, jika tidak Anda overfit, dan cara yang paling dapat diandalkan untuk menyembuhkan overfit adalah dengan mendapatkan lebih banyak data (Memasukkan data tes ke dalam data pelatihan tidak dihitung)

Omong-omong, kumpulan data gambar publik terbesar (imagenet) tidak cukup besar untuk kompetisi imagenet 1000 kelas

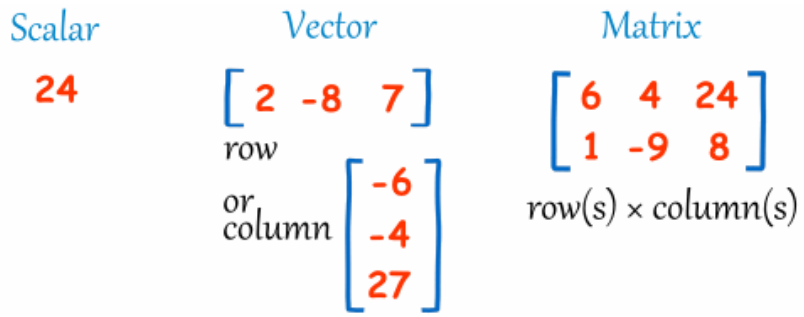


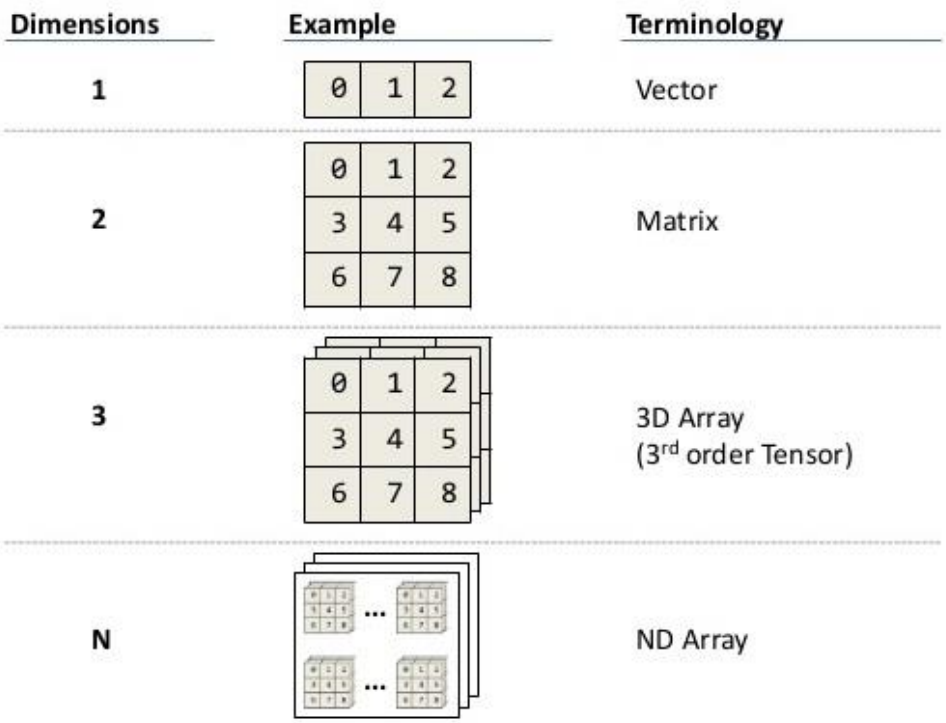
Aljabar linier

Aljabar Linear adalah topik penting untuk dipahami, banyak algoritma pembelajaran mendalam menggunakannya, jadi bab ini akan mengajarkan topik yang diperlukan untuk memahami apa yang akan terjadi selanjutnya.

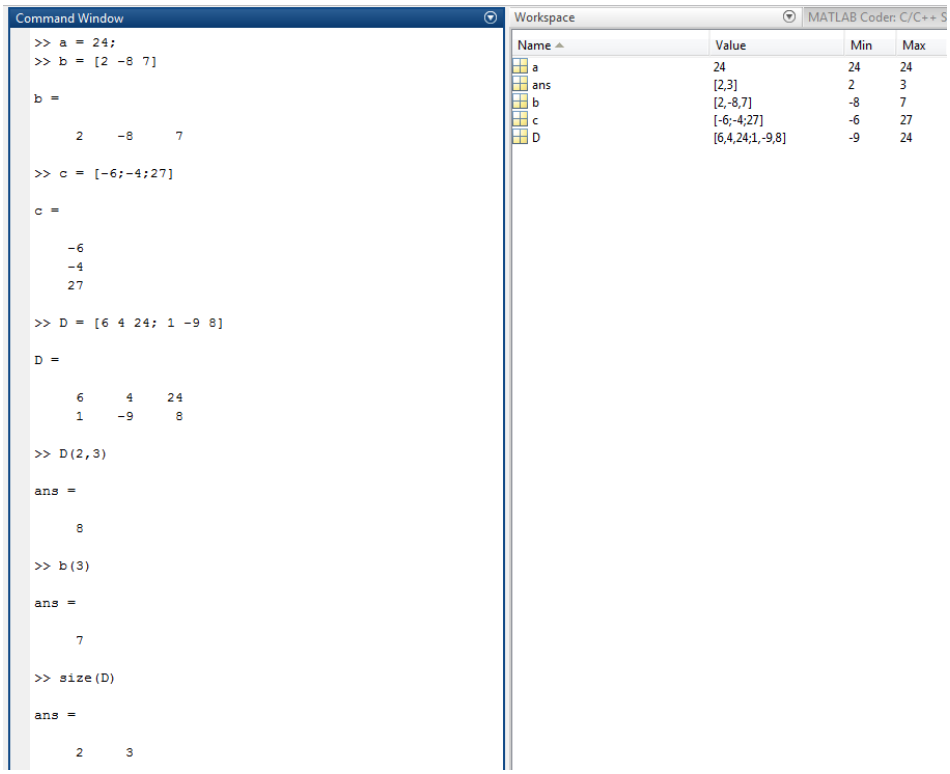
Skalar, Vektor dan Matriks

- Skalar: Angka tunggal
- Vektor: Array angka 1D, di mana setiap elemen diidentifikasi oleh satu indeks
- Matriks: Susunan angka 2D, di bawah ini kita memiliki matriks (2-baris)X(3-kolom). Dalam matriks, satu elemen diidentifikasi oleh dua indeks, bukan satu.





Di sini kami menunjukkan cara membuatnya di matlab dan python (numpy)



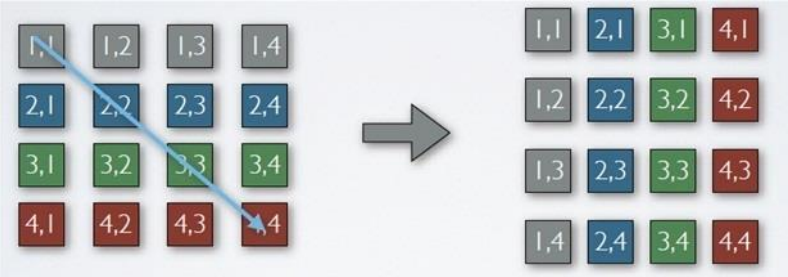

```
leo@monsterpc2:~$ ipython
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
Type "copyright", "credits" or "license" for more information.

IPython 2.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import numpy as np

In [2]: a = 24

In [3]: b = np.array([[2, -8, 7]])

In [4]: c = np.array([[-6],[-4],[27]])

In [5]: D = np.array([[6,4,24],[1,-9,8]])

In [6]: print(b)
[[ 2 -8  7]]

In [7]: print(c)
[[-6]
 [-4]
 [27]]

In [8]: print(D)
[[ 6  4 24]
 [ 1 -9  8]]

In [9]: D[1,2]
Out[9]: 8

In [10]: D.shape
Out[10]: (2, 3)
```

Operasi Matriks

Di sini kami akan menunjukkan beberapa operasi matriks penting.

Mengubah urutan

Jika Anda memiliki gambar (matriks 2D) dan mengalikannya dengan matriks rotasi, Anda akan mendapatkan gambar yang diputar. Sekarang jika Anda mengalikan gambar yang diputar ini dengan transpos matriks rotasi, gambar tersebut akan "tidak diputar" Pada dasarnya untuk mentranspos matriks berarti menukar baris dan kolomnya. Atau dengan kata lain memutar matriks di sekitar diagonal utamanya.



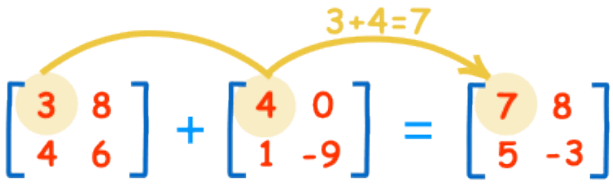
$$A$$

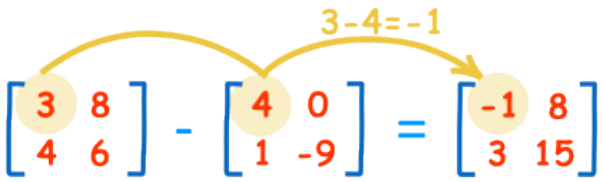
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

- $\begin{bmatrix} 1 & 2 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$
- $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$
- $\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$

Penjumlahan/Pengurangan

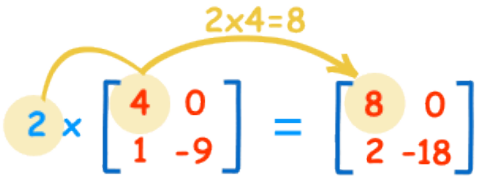
Pada dasarnya kita menjumlahkan 2 matriks dengan menjumlahkan setiap elemen dengan elemen lainnya. Kedua matriks harus memiliki dimensi yang sama





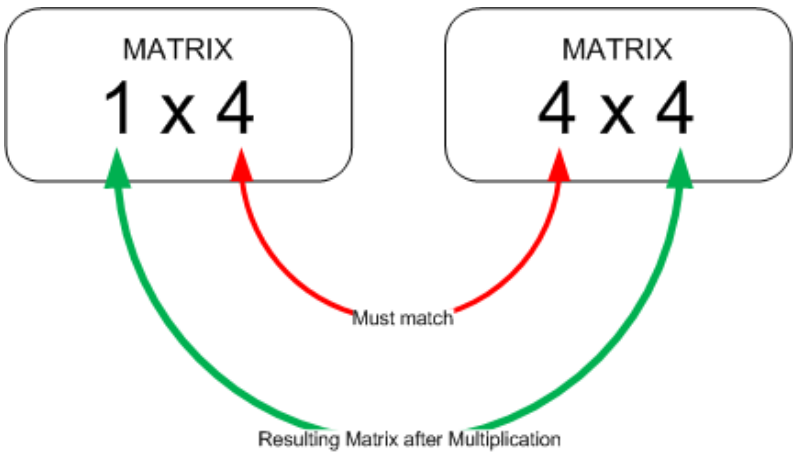
Kalikan dengan skalar

Kalikan semua elemen matriks dengan skalar

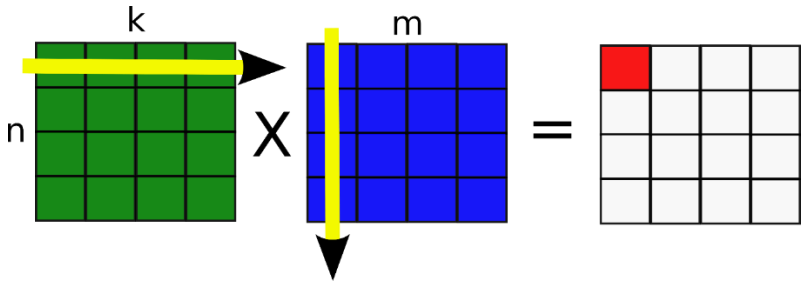


Perkalian Matriks

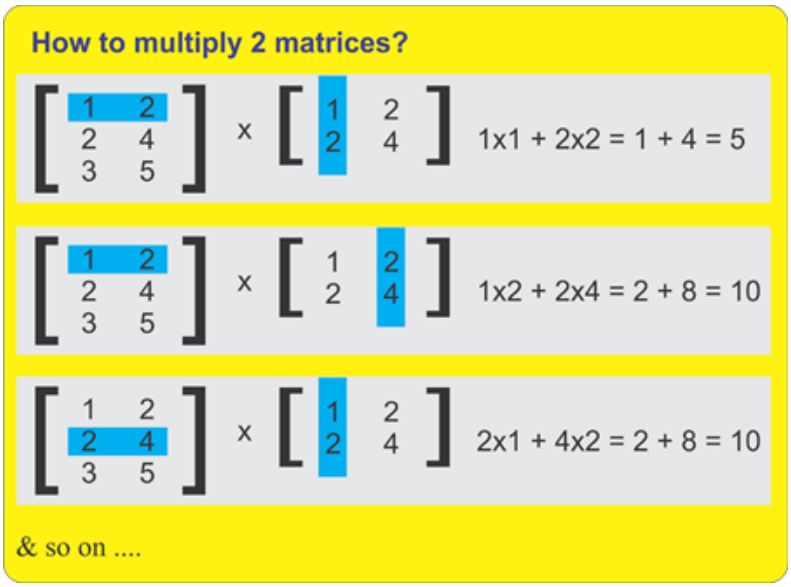
Produk matriks dari matriks n×m dengan matriks m×ℓ adalah matriks n×ℓ. Entri (i,j) dari hasil kali matriks AB adalah perkalian titik dari baris ke-i dari A dengan kolom ke-j dari B. Jumlah kolom pada matriks pertama harus sama dengan jumlah baris pada matriks kedua. Hasilnya adalah matriks lain atau skalar dengan dimensi yang ditentukan oleh baris matriks pertama dan kolom matriks kedua.

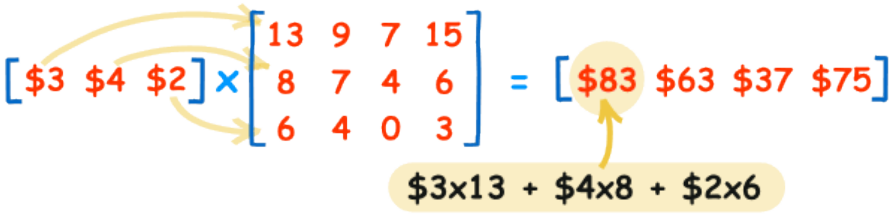


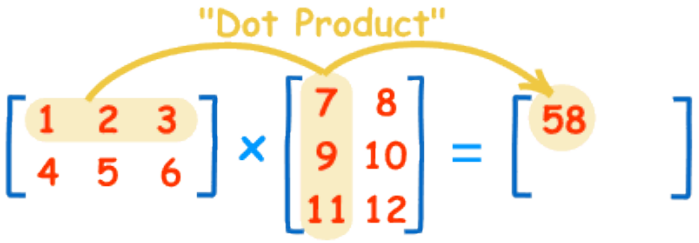
Pada dasarnya operasinya adalah "kalikan titik" setiap baris dari matriks pertama (k) dengan setiap kolom dari matriks kedua (m).

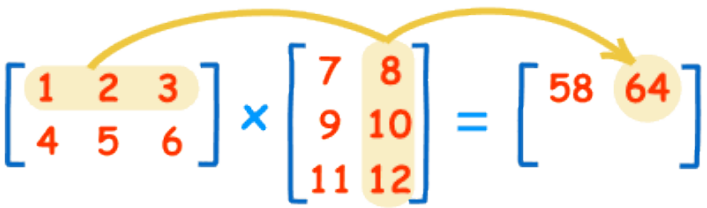


Beberapa contoh









Sifat komutatif

Perkalian matriks tidak selalu bersifat komutatif $A.B \neq B.A$, tetapi perkalian titik antara 2 vektor bersifat komutatif, $x^T.y = y^T.x$.

.Jenis Matriks

Ada beberapa matriks khusus yang menarik untuk diketahui.

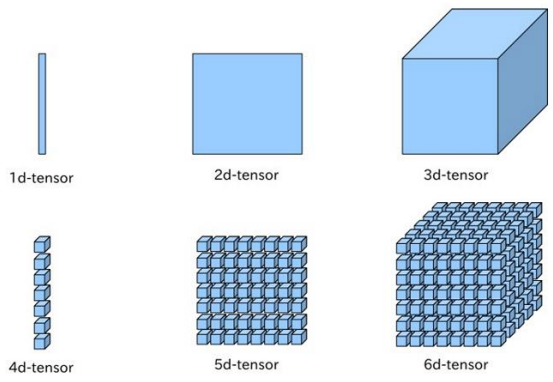
- Identitas: Diagonal matriks identitas diisi dengan satu, sisanya nol. Jika Anda mengalikan matriks B dengan matriks identitas, Anda akan mendapatkan matriks B sebagai hasilnya,

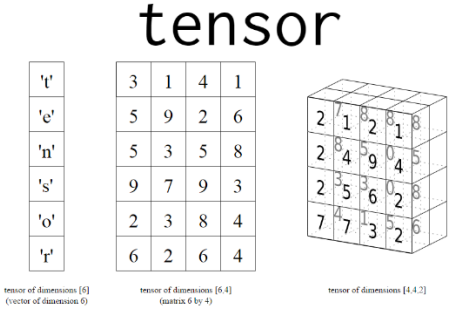
$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Pembalikan: Digunakan pada pembagian matriks dan untuk menyelesaikan sistem linier.

Tensor

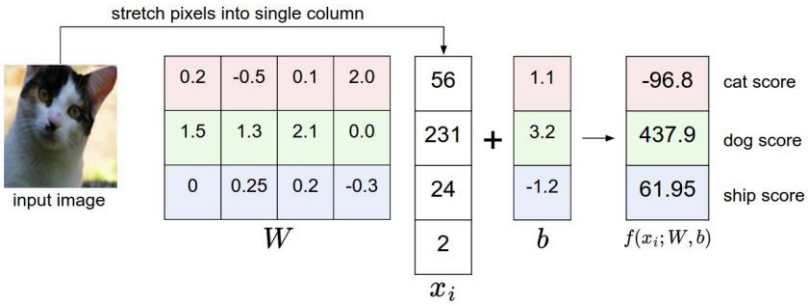
Terkadang kita perlu mengatur informasi dengan lebih dari 2 dimensi, kita menyebut tensor sebagai array n-dimensi. Misalnya tensor 1D adalah vektor, tensor 2D adalah matriks, tensor 3D adalah kubus, dan tensor 4D adalah vektor kubus, tensor 5D adalah matriks kubus.



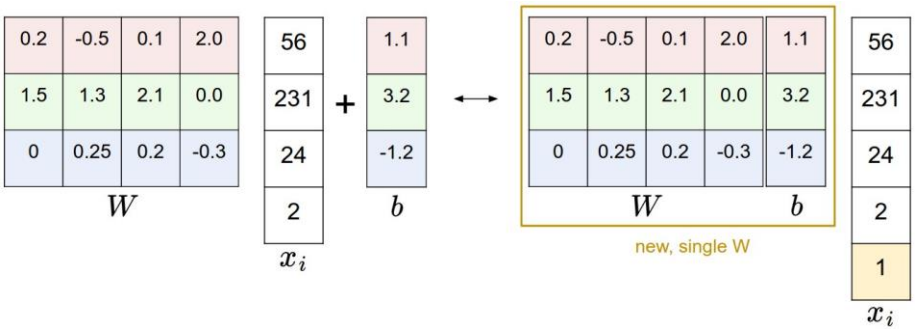


Contoh praktis

Di sini kami akan menunjukkan cara menggunakan perkalian matriks untuk mengimplementasikan pengklasifikasi linier. Kami tidak peduli sekarang tentang apa yang dilakukan pengklasifikasi linier, perhatikan saja bahwa kami menggunakan aljabar linier kami untuk menyelesaikannya.



Kita dapat menggabungkan bobot dan bias (disebut trik bias) untuk menyelesaikan klasifikasi linier sebagai perkalian matriks tunggal



Di Matlab

```
>> W_b = [0.2 -0.5 0.1 2 1.1; 1.5 1.3 2.1 0 3.2; 0 0.25 0.2 -0.3 -1.2]

W_b =

    0.2000   -0.5000    0.1000    2.0000    1.1000
    1.5000    1.3000    2.1000         0    3.2000
         0    0.2500    0.2000   -0.3000   -1.2000

>> x = [56 231 24 2 1]'

x =

    56
   231
    24
     2
     1

>> W_b*x

ans =

  -96.8000
  437.9000
   60.7500
```

Name	Value	Min	Max
ans	[-96.8000;437.9000;60...	-96.80...	437.90...
W_b	*3x5 double*	-1.2000	3.2000
x	[56;231;24;2;1]	1	231

Dengan Python

```
IPython 2.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: import numpy as np

In [2]: W_b = np.array([ [0.2,-0.5,0.1,2,1.1],[1.5,1.3,2.1,0,3.2],[0,0.25,0.2,-0.3,-1.2] ])

In [3]: x = np.array([56,231,24,2,1])

In [4]: scores = W_b.dot(x)

In [5]: print scores
[ -96.8   437.9    60.75]
```

Pembelajaran yang Diawasi

Dalam bab ini kita akan belajar tentang Supervised learning serta berbicara sedikit tentang Cost Functions dan Gradient descent. Juga kita akan belajar tentang 2 algoritma sederhana:

- Regresi Linear (untuk Regresi)
- Regresi Logistik (untuk Klasifikasi)

Hal pertama yang harus dipelajari tentang pembelajaran terawasi adalah bahwa setiap titik data sampel x memiliki keluaran atau label y yang diharapkan, dengan kata lain, pelatihan Anda terdiri dari $(x^{(i)}, y^{(i)})$ berpasangan.

Sebagai contoh perhatikan tabel di bawah ini:

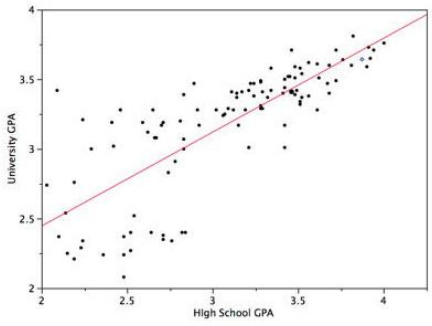
Ukuran kaki (x)	Mahal)
2104	460
1416	232
1534	315
852	178

Tabel ini (atau set pelatihan) menunjukkan ukuran rumah beserta harganya. Jadi rumah berukuran 2104 kaki berharga 460.

Idenya adalah kita dapat menggunakan data seperti ini untuk membuat model yang dapat memprediksi beberapa hasil (yaitu: Harga) dari input yang berbeda (yaitu: Ukuran dalam satuan kaki).

Regresi linier

Regresi adalah tentang mengembalikan angka skalar kontinu dari beberapa input. Model atau hipotesis yang akan kita pelajari akan memprediksi nilai mengikuti aturan linier. Namun terkadang model linier tidak cukup untuk menangkap sifat dasar data Anda.



Hipotesa:

Pada dasarnya regresi linier mencoba membuat garis $f(x) = \beta + \alpha.x$, yang cocok dengan data pelatihan, misalnya

$$h_{\theta_i}(x) = \theta_0 + \theta_1.x$$

Di mana $\theta_i \in \{\theta_0, \theta_1\}$

Seluruh gagasan pembelajaran yang diawasi adalah bahwa kami mencoba mempelajari parameter terbaik (theta dalam hal ini) dari set pelatihan kami.

Fungsi Biaya

Sebelum kita berbicara tentang cara mempelajari parameter (juga disebut bobot) dari hipotesis kita, kita perlu mengetahui cara mengevaluasi apakah rangkaian bobot kita saat ini sudah berfungsi dengan baik. Fungsi yang melakukan pekerjaan ini disebut fungsi Kerugian atau Biaya. Pada dasarnya ini akan mengembalikan nilai skalar antara 0 (Tidak ada kesalahan) dan tak terhingga (Sangat buruk).

Contoh dari fungsi seperti itu diberikan di bawah ini:

$$J(\theta_i) = \frac{1}{2.m}.\sum_{i=1}^{m}[h_{\theta_i}(x^{(i)}) - y^{(i)}]^2$$

Di mana

- m: Jumlah item dalam kumpulan data Anda
- (i): elemen ke-i dari kumpulan data Anda
- y: Label(nilai yang diharapkan) dalam kumpulan data

Fungsi biaya khusus ini disebut mean-squared error loss, dan sebenarnya sangat berguna untuk masalah regresi.

Selama pelatihan kami ingin meminimalkan kerugian kami dengan terus mengubah parameter theta. Fitur bagus lainnya dari fungsi khusus ini adalah bahwa ini adalah fungsi cembung sehingga dijamin tidak memiliki lebih dari satu minimum, yang juga akan menjadi minimum globalnya. Ini memudahkan kita untuk mengoptimalkan.

Tugas kita adalah untuk menemukan:

$$\min_\theta J(\theta_i)$$

Penurunan gradien

Penurunan gradien adalah algoritme sederhana yang akan mencoba menemukan minimum lokal suatu fungsi. Kami menggunakan penurunan gradien untuk meminimalkan fungsi kerugian kami. Salah satu fitur penting untuk diamati pada gradient descent adalah bahwa ia akan lebih sering terjebak ke minimum lokal pertama yang ditemuinya. Namun tidak ada jaminan bahwa minimum lokal yang ditemukannya adalah yang terbaik (global).

Algorithm 1 Gradient Descent

Input: Differentiable function $\mathbf{f(x)}$ where $\mathbf{f(x)} : \mathbf{R^n} \rightarrow \mathbf{R}$
Start point $\mathbf{x_{old}}$
Output: The local minima $\mathbf{x}^*$ that minimize $\mathbf{f(x)}$

while TRUE **do**
  tmpDelta $\leftarrow \mathbf{x_{old}} - \alpha.(\nabla \mathbf{f}(\mathbf{x_{old}}))$
  if abs(tmpDelta $- \mathbf{x_{old}}) <$ **CRITERIA then**
    break
  end if
  $\mathbf{x_{old}} \leftarrow$ **tmpDelta**
end while

Turunan gradien membutuhkan turunan pertama dari fungsi yang ingin Anda minimalkan. Jadi jika kita ingin meminimalkan beberapa fungsi dengan mengubah parameter, Anda perlu menurunkan fungsi ini sehubungan dengan parameter tersebut.

Satu hal yang perlu diperhatikan adalah karena algoritme ini akan dijalankan pada semua sampel di set pelatihan Anda, algoritme ini tidak dapat diskalakan dengan baik untuk set data yang lebih besar.

Contoh sederhana

Di bawah ini kami memiliki beberapa implementasi sederhana di matlab yang menggunakan gradient descent untuk meminimalkan fungsi berikut:

$$f(x) = x^4 - 3.x^3 + 2$$

Turunannya terhadap x adalah:

$$\frac{\partial f(x)}{\partial x} = 4.x^3 - 9.x^2$$

Kode untuk mengetahui pada titik mana minimum lokal kita terjadi adalah sebagai berikut:

```
% Some tests on Gradient descent
%% Define parameters start at 3.5
x_old=3.5; alpha=0.01; precision=0.0001;

%% Define function
x_input = [-1:0.01:3.5];
f = @(x) x.^4 - 3*x.^3 + 2;
df = @(x) 4*x.^3 - 9*x.^2;
y_output = f(x_input);
plot(x_input, y_output);

%% Gradient descent algorithm
% Keep repeating until convergence
while 1
    % Evalulate gradients
    tmpDelta = x_old - alpha*(df(x_old));
    % Check Convergence
    diffOldTmp = abs(tmpDelta - x_old);
    if diffOldTmp < precision
        break;
    end
    % Update parameters
    x_old = tmpDelta;
end
fprintf('The local minimum is at %d\n', x_old);
```

Penurunan gradien untuk regresi linier

Untuk menggunakan penurunan gradien untuk regresi linier, Anda perlu menghitung turunan dari kerugiannya (berarti kesalahan kuadrat) sehubungan dengan parameternya.

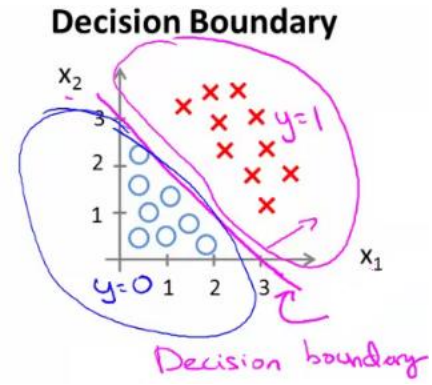
Turunan ini akan menjadi:

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} \cdot \sum_{i=1}^{m} (h_{\theta_i}(x^{(i)}) - y^{(i)}).x_j^{(i)}$$

Regresi logistik

Namanya mungkin terdengar membingungkan tetapi sebenarnya Regresi Logistik hanyalah tentang klasifikasi. Misalnya perhatikan contoh di bawah ini di mana kita ingin mengklasifikasikan 2 kelas: x dan o. Sekarang keluaran kita y akan memiliki dua kemungkinan nilai [0,1].

Biasanya kami tidak menggunakan Logistic Regression jika kami memiliki banyak fitur (misalnya lebih dari 100 fitur).



Hipotesa

Hipotesis kami hampir sama dibandingkan dengan regresi linier namun perbedaannya adalah sekarang kami menggunakan fungsi yang akan memaksa keluaran kami untuk memberikan

$$y \in [0, 1]$$

$$h_\theta(x) = g(\theta^T.x)$$

Di mana

$g(z) = \frac{1}{1+e^{-z}}$ adalah fungsi logistik atau sigmoid, oleh karena itu

$$h_\theta(x) = \frac{1}{1+e^{(\theta^T.x)}}$$

Di sini fungsi sigmoid akan mengubah bilangan skalar menjadi probabilitas antara 0 dan 1.

Fungsi biaya untuk klasifikasi

Saat berhadapan dengan masalah klasifikasi, kita harus menggunakan fungsi biaya/kerugian yang berbeda. Kandidat yang baik untuk klasifikasi adalah fungsi biaya lintas-entropi. Omong-omong, fungsi ini dapat ditemukan dengan menggunakan metode estimasi Maximum Likelihood.

Fungsi biaya lintas-entropi adalah:

$$J(\theta) = \frac{1}{m}.\sum_{i=1}^{m}[y^{(i)}.log(h_\theta(x^{(i)})) + (1-y^{(i)}).log(1-h_\theta(x^{(i)}))]$$

Sekali lagi tujuan kami adalah untuk menemukan theta parameter terbaik yang meminimalkan fungsi ini, jadi untuk menggunakan penurunan gradien kita perlu menghitung turunan dari fungsi ini sehubungan dengan theta

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m}.\sum_{i=1}^{m}(h_{\theta_i}(x^{(i)}) - y^{(i)}).x_j^{(i)}$$

Satu hal keren yang perlu diperhatikan adalah turunannya sama dengan turunan dari regresi linier.

Overfitting (Variance) dan Underfitting (Bias)

Gagasan utama pelatihan dalam pembelajaran mesin adalah membiarkan komputer mempelajari struktur dasar dari rangkaian pelatihan dan bukan hanya rangkaian pelatihan spesifik yang dilihatnya. Jika tidak mempelajari struktur yang mendasarinya dan malah hanya mempelajari struktur sampel pelatihan, maka kita katakan model kita overfit.

Kami dapat mengetahui bahwa overfitting telah terjadi ketika model kami memiliki akurasi yang sangat baik pada set pelatihan (misalnya: 99,99%) tetapi tidak bekerja dengan baik pada set pengujian (misalnya: 60%).

Hal ini pada dasarnya terjadi ketika model Anda terlalu rumit terkait dengan data yang tersedia, dan/atau Anda tidak memiliki cukup data untuk menangkap pola data yang mendasarinya.

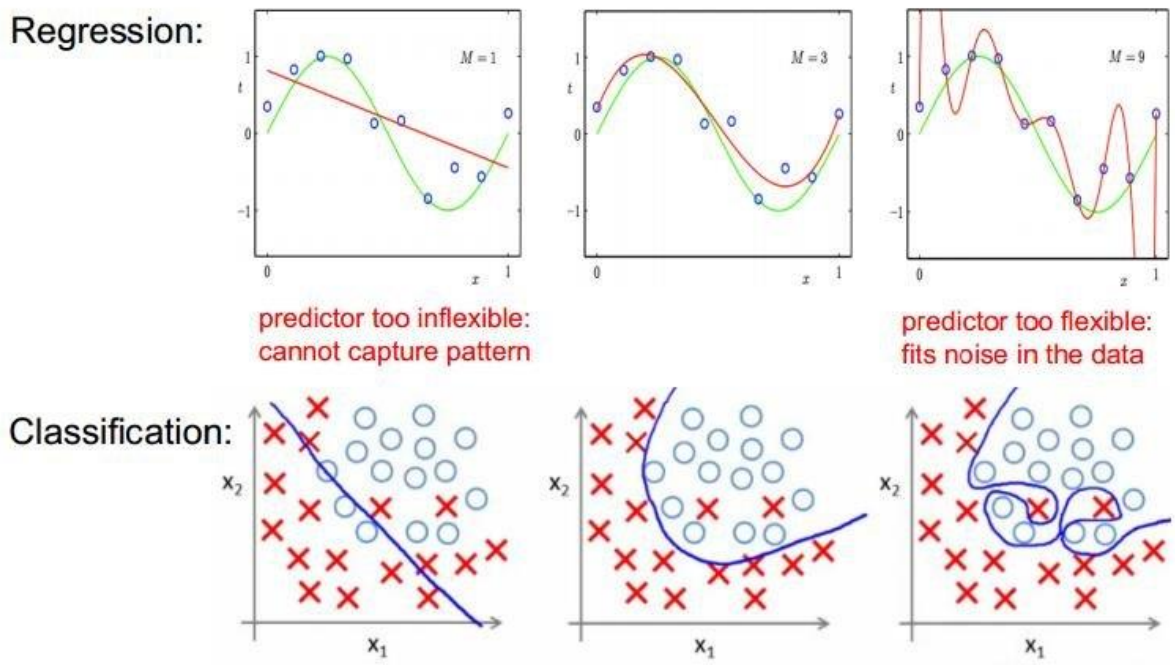
Kebalikan dari masalah ini disebut Underfitting, pada dasarnya ini terjadi ketika model Anda terlalu sederhana untuk menangkap konsep yang diberikan dalam set pelatihan.

Beberapa contoh grafis dari masalah ini ditunjukkan di bawah ini:

Kiri: Pakaian dalam

Tengah: Sempurna

Kanan: Pakaian luar



Mengatasi Overfitting

- Dapatkan lebih banyak data
- Gunakan regularisasi
- Periksa arsitektur model lainnya

Memecahkan Underfitting

- Tambahkan lebih banyak lapisan atau lebih banyak parameter
- Periksa arsitektur lainnya

Regularisasi

Ini adalah metode yang membantu overfitting dengan memaksa parameter hipotesis Anda memiliki nilai kecil yang bagus dan memaksa model Anda untuk menggunakan semua parameter yang tersedia. Untuk menggunakan regularisasi, Anda hanya perlu menambahkan istilah tambahan ke fungsi biaya/kerugian selama pelatihan. Di bawah

ini kami memiliki contoh fungsi kerugian kesalahan kuadrat rata-rata dengan istilah regularisasi tambahan.

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} . \sum_{i=1}^{m} \left(h_{\theta_i}(x^{(i)}) - y^{(i)} \right) . x_j^{(i)}$$

Juga versi regular dari fungsi cross-entropy loss

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \ \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \ \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^{n} \theta_j^2$$

Istilah ini pada dasarnya akan dikalikan dengan λ semua parameter θ dari hipotesis Anda

Biasanya kita tidak perlu mengatur istilah bias dari hipotesis kita.

Selama penurunan gradien, Anda juga perlu menghitung turunan dari istilah tersebut.

Intuisi

Anda dapat membayangkan bahwa selain memaksa bobot untuk memiliki nilai yang lebih rendah, regularisasi juga akan menyebarkan "konsep" ke lebih banyak bobot. Salah satu cara untuk memikirkan tentang apa yang terjadi ketika kita melakukan regularisasi ditunjukkan di bawah ini.

Untuk input x kami, kami dapat memiliki 2 vektor bobot w1 atau w2. Ketika kita mengalikan input kita dengan vektor bobot kita, kita akan mendapatkan output yang sama yaitu 1 untuk masing-masingnya. Namun vektor bobot w2 adalah pilihan vektor bobot yang lebih baik karena ini akan melihat lebih banyak masukan kita x dibandingkan dengan w1 yang hanya melihat satu nilai x.

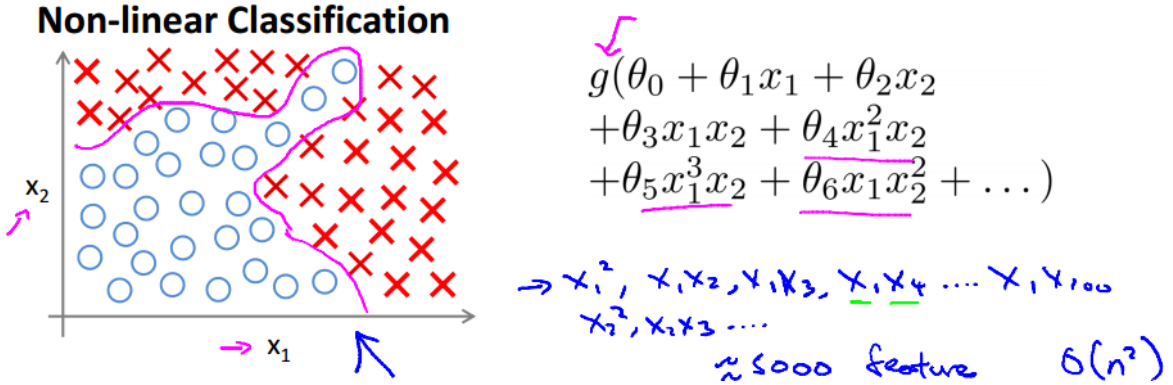
$$x = [1, 1, 1, 1]$$
$$w_1 = [1, 0, 0, 0]$$
$$w_2 = [0.25, 0.25, 0.25, 0.25]$$
$$\therefore w_1^T . x = w_2^T . x = 1$$

Dalam praktiknya, hal ini dapat mengurangi performa pada set pelatihan kami, tetapi akan meningkatkan generalisasi dan performa saat pengujian.

Hipotesis Non-Linear

Terkadang hipotesis kita perlu memiliki istilah non-linear untuk dapat memprediksi masalah klasifikasi/regresi yang lebih kompleks. Misalnya, kita dapat menggunakan Regresi Logistik dengan istilah kuadrat untuk menangkap kompleksitas ini. Seperti

yang disebutkan sebelumnya, ini tidak menskalakan dengan baik untuk sejumlah besar fitur.



Misalnya, jika kami ingin mengklasifikasikan gambar skala abu-abu 100x100 menggunakan regresi logistik, kami memerlukan 50 juta parameter untuk menyertakan istilah kuadrat. Pelatihan dengan fitur sebanyak ini akan membutuhkan setidaknya 10x lebih banyak gambar (500 juta) untuk menghindari overfitting. Juga biaya komputasi akan sangat tinggi.

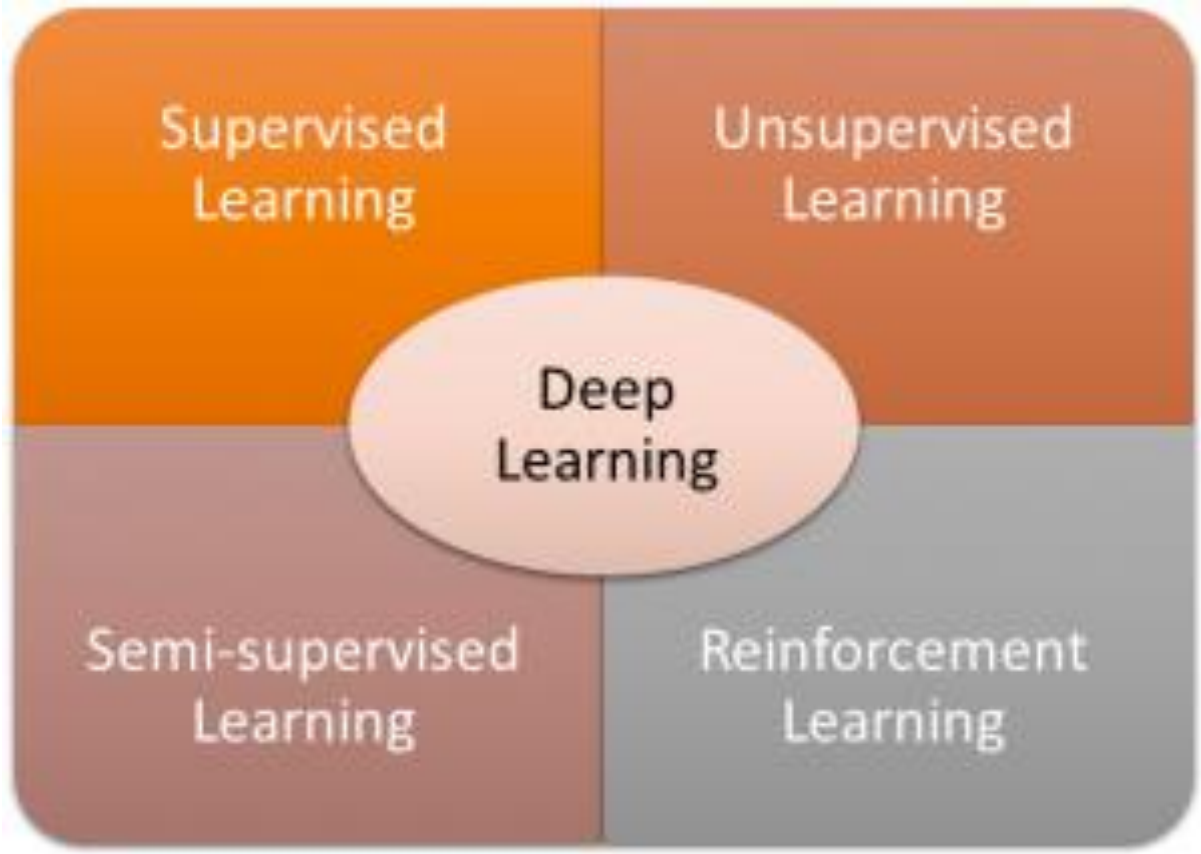
Di bab selanjutnya kita akan mempelajari algoritme lain yang menskalakan lebih baik untuk jumlah fitur yang lebih banyak.

Minim lokal

Saat kami menambah jumlah parameter pada model kami, fungsi kerugian kami akan mulai menemukan beberapa minima lokal. Ini bisa menjadi masalah untuk pelatihan karena penurunan gradien dapat macet di salah satunya.

Sebenarnya beberapa makalah baru-baru ini juga mencoba untuk membuktikan bahwa beberapa metode yang digunakan saat ini (yaitu: jaringan Deep Neural) local-minima sebenarnya sangat dekat dengan minima global, jadi Anda tidak perlu peduli menggunakan Convex loss atau terjebak dalam local-minima.

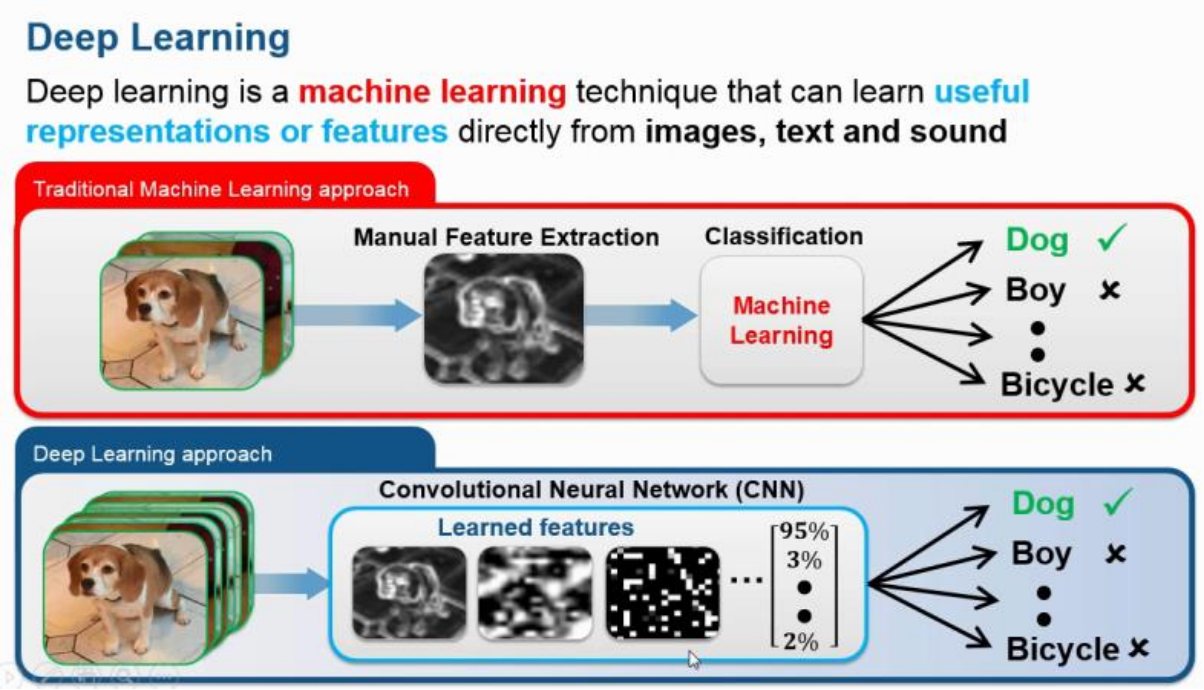
Pembelajaran Mendalam



Pembelajaran mendalam adalah cabang pembelajaran mesin berdasarkan sekumpulan algoritme yang belajar untuk merepresentasikan data. Di bawah ini kami daftar yang paling populer.

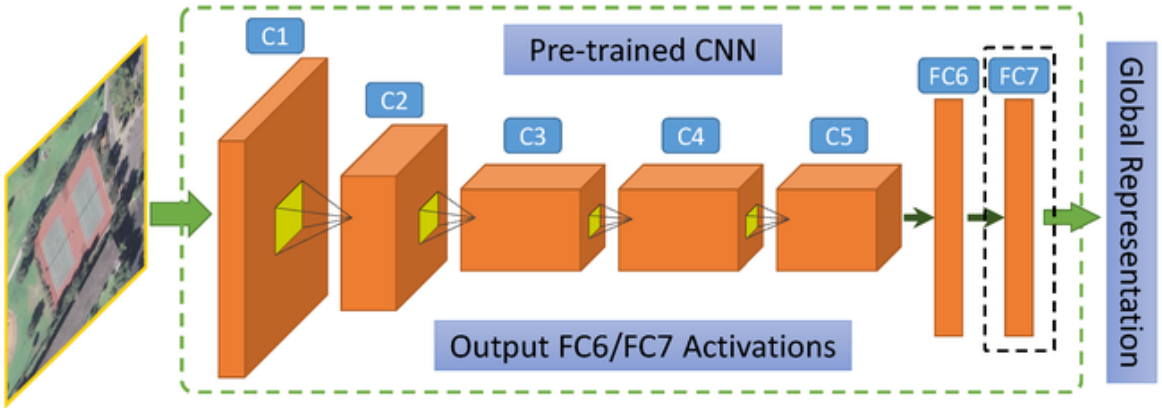
- Jaringan Syaraf Konvolusional
- Jaringan Keyakinan Mendalam
- Deep Auto-Encoder
- Jaringan Syaraf Berulang (RNN/LSTM/GRU)
- Jaringan Musuh Generatif (GAN)

Salah satu janji pembelajaran mendalam adalah bahwa mereka akan menggantikan ekstraksi fitur kerajinan tangan. Idenya adalah mereka akan "mempelajari" fitur terbaik yang diperlukan untuk mewakili data yang diberikan.



Lapisan dan lapisan

Model pembelajaran mendalam dibentuk oleh banyak lapisan. Dalam konteks jaringan saraf tiruan, multi layer perceptron (MLP) dengan lebih dari 2 lapisan tersembunyi sudah menjadi Deep Model. Sebagai patokan, model yang lebih dalam memiliki potensi untuk tampil lebih baik daripada model yang dangkal. Masalahnya adalah semakin dalam Anda masuk, semakin banyak data yang Anda perlukan untuk menghindari pemasangan yang berlebihan.



Jenis lapisan

Di sini kami mencantumkan beberapa lapisan yang paling banyak digunakan 1. Lapisan Konvolusi 2. Lapisan Penyatuan Maks/Rata-Rata 3. Lapisan Putus Sekolah 4. Lapisan Normalisasi Batch 5. Lapisan Terhubung Sepenuhnya (Affine). 6. Relu, Tanh, Lapisan Sigmoid (Lapisan Non-Linearitas) 7. Softmax, Cross Entropy, SVM, Euclidean (Loss Layers)

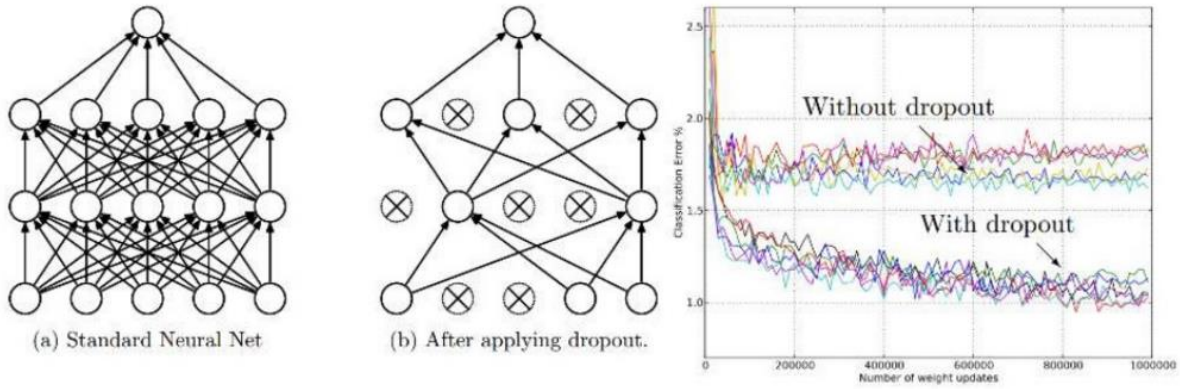
Hindari over-fitting (regularisasi)

Selain mendapatkan lebih banyak data, ada beberapa teknik yang digunakan untuk melawan over-fitting, berikut adalah daftar yang paling umum:

- Keluar
- Regularisasi L2
- Augmentasi Data

Keluar

Ini adalah teknik yang secara acak mematikan beberapa neuron dari lapisan yang terhubung sepenuhnya selama pelatihan.



Putus sekolah memaksa lapisan yang terhubung sepenuhnya untuk mempelajari konsep yang sama dengan cara yang berbeda

Regularisasi L2

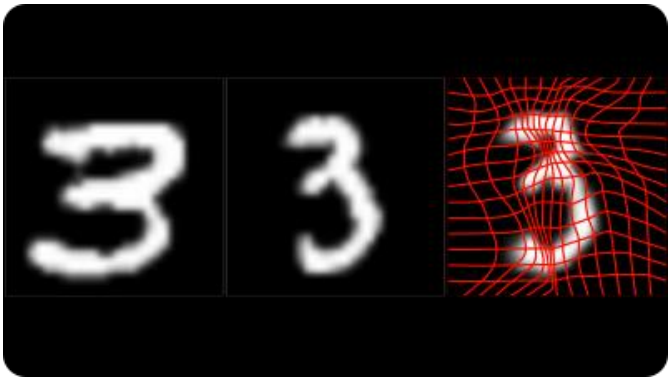
Bentuk regularisasi yang paling umum adalah regularisasi L2. Dalam hal ini kami menambahkan istilah ke fungsi kerugian kami yang menghukum nilai kuadrat dari semua bobot/parameter yang kami optimalkan. Untuk setiap berat *w* di jaringan saraf kami, kami menambahkan istilah $0{,}5\,\lambda w2$ ke fungsi kerugian/tujuan. *Λ* adalah parameter kekuatan regularisasi. Separuh digunakan hanya untuk mempermudah saat kita menghitung turunan untuk perambatan balik karena ia hanya akan meniadakan.

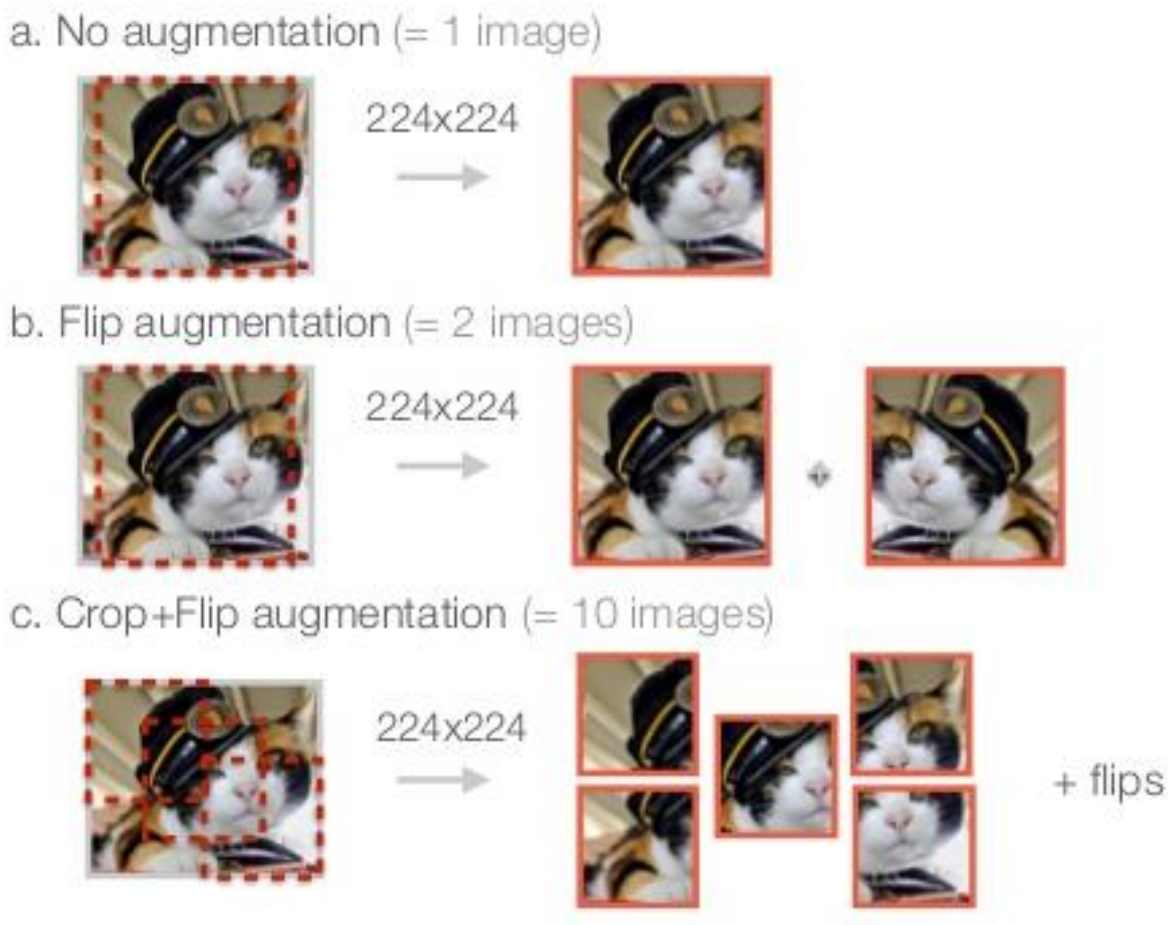
Sebagai hasil dari penggunaan regularisasi ini bobot nilai yang sangat tinggi akan dikenakan sanksi berat. Ini mendorong model kami untuk memilih bahwa semua masukan ke lapisan digunakan sedikit daripada beberapa masukan yang banyak digunakan. Properti ini secara intuitif cukup bagus untuk dimiliki karena model kami akan digunakan secara maksimal dan kami memiliki lebih sedikit bobot yang tidak digunakan.

Selain regularisasi L2, ada regularisasi L1 dan Max Norm, tetapi hal ini tidak dibahas di sini karena L2 umumnya berkinerja lebih baik.

Augmentasi Data

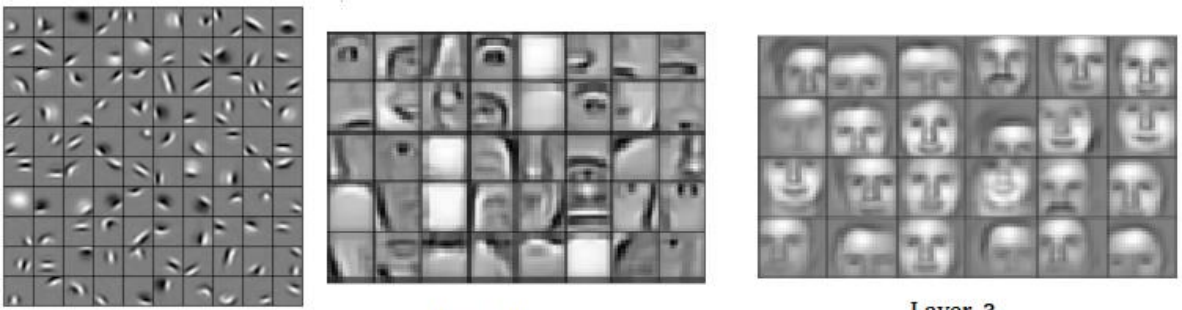
Dimungkinkan untuk membuat contoh pelatihan baru secara sintetik dengan menerapkan beberapa transformasi pada data input. Misalnya membalik gambar atau mengubah nilai RGB secara acak. Selama Kompetisi Imagenet 2012, Alex Krizhevesky (Alexnet) menggunakan augmentasi data dengan faktor 2048 yang berarti bahwa kumpulan data yang digunakan untuk melatih modelnya secara efektif 2048 kali lebih besar daripada di awal dan memberikan peningkatan pada generalisasi daripada tidak menggunakannya.



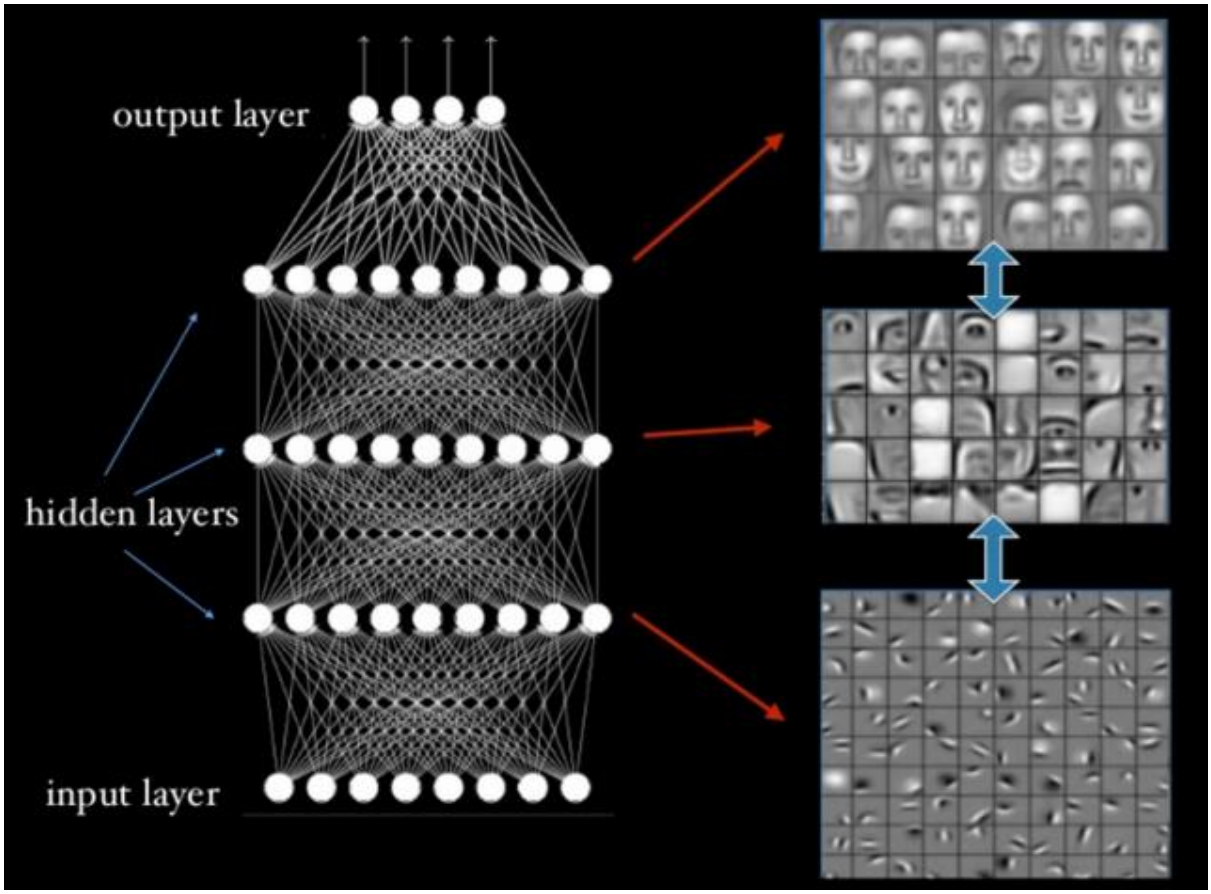


Representasi hierarkis otomatis

Idenya adalah membiarkan algoritme pembelajaran menemukan representasi terbaik yang dapat dilakukannya untuk setiap lapisan mulai dari input hingga yang lebih dalam. Lapisan dangkal belajar merepresentasikan data dalam bentuk yang lebih sederhana dan lapisan terdalam belajar merepresentasikan data dengan konsep yang dipelajari dari yang sebelumnya.



Layer 1 Layer 2 Layer 3



Lama vs Baru

Sebenarnya satu-satunya hal baru adalah penggunaan model yang akan mempelajari cara terbaik untuk merepresentasikan data Anda (pemilihan fitur) secara otomatis dan berdasarkan kumpulan data yang diberikan padanya. Ini berbeda dengan di masa lalu di mana fitur kerajinan tangan, seperti HOG (Histogram of Oriented Gradients), digunakan. Memikirkan fitur-fitur baru ini dapat memakan waktu lama dan tidak dijamin akan optimal untuk setiap kumpulan data.

Keuntungan terbesar dari cara baru ini adalah jika masalah Anda menjadi lebih kompleks, Anda cukup membuat model Anda "lebih dalam" dan mendapatkan lebih banyak data (banyak) untuk melatih masalah baru Anda dan model tersebut kemudian akan mempelajari fitur terbaik untuk tugas khusus Anda.

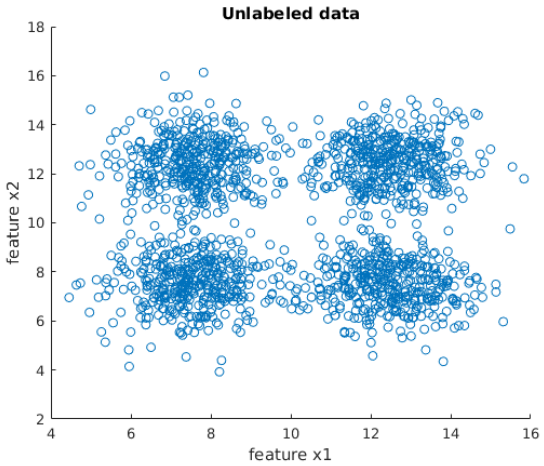
Beberapa orang dari Deep Learning

Pembelajaran Tanpa Pengawasan

Seperti disebutkan pada bab-bab sebelumnya, pembelajaran tanpa pengawasan adalah tentang mempelajari informasi tanpa label informasi. Di sini istilah informasi berarti, "struktur" misalnya Anda ingin tahu berapa banyak grup yang ada di kumpulan data Anda, bahkan jika Anda tidak tahu apa arti grup itu. Kami juga menggunakan pembelajaran tanpa pengawasan untuk memvisualisasikan kumpulan data Anda, untuk mencoba mempelajari beberapa wawasan dari data tersebut.

Contoh data tanpa label

Perhatikan kumpulan data berikut $X \in R2$ (X memiliki 2 fitur)



Salah satu jenis algoritme pembelajaran tanpa pengawasan yang disebut "pengelompokan" digunakan untuk menyimpulkan berapa banyak grup berbeda yang ada di kumpulan data Anda.

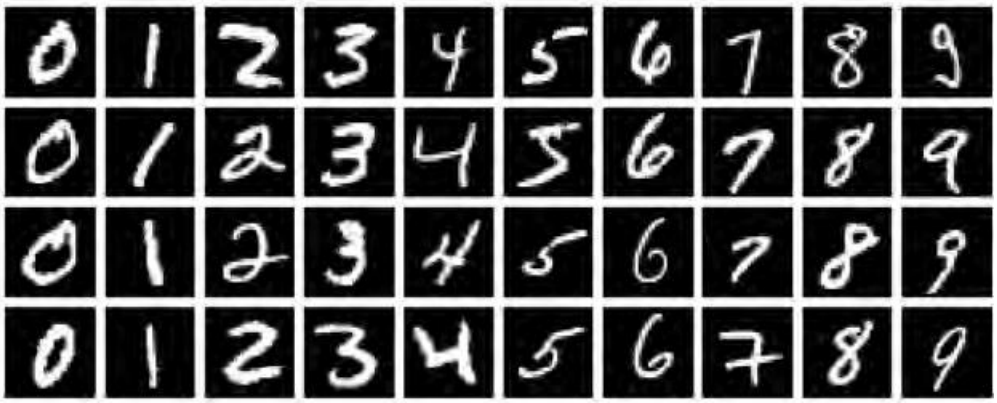
Di sini kita masih belum tahu apa arti kelompok-kelompok itu, tetapi kita tahu bahwa ada 4 kelompok yang kelihatannya sangat berbeda. Dalam hal ini kami memilih dataset

berdimensi rendah *R*2 tapi di kehidupan nyata bisa jadi ribuan dimensi, yaitu *R*784 untuk gambar 28x28 skala abu-abu.

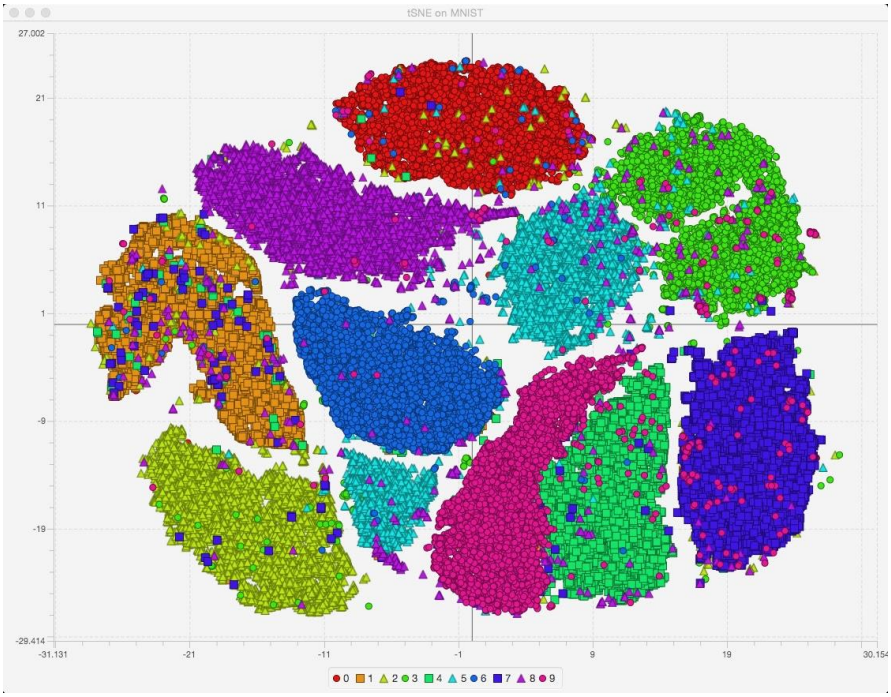
Pengurangan Dimensi

Untuk meningkatkan waktu respons klasifikasi (bukan kinerja prediksi) dan terkadang untuk memvisualisasikan dataset dimensi tinggi Anda (2D, 3D), kami menggunakan teknik pengurangan dimensi (yaitu: PCA, T-Sne).

Misalnya kumpulan data MNIST terdiri dari 60.000 contoh pelatihan dari (0..9) digit, masing-masing dengan 784 dimensi. Dimensi tinggi ini disebabkan fakta bahwa setiap digit adalah gambar skala abu-abu 28x28.



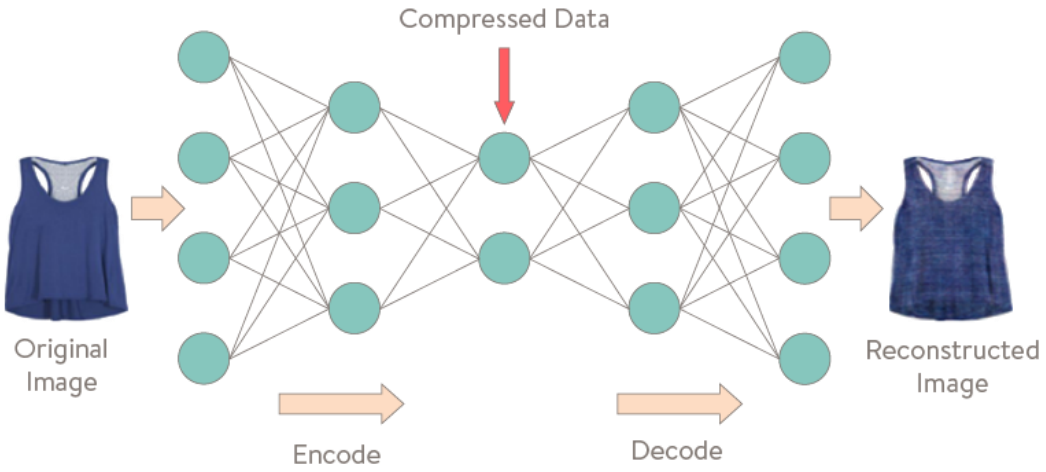
Akan sulit untuk memvisualisasikan kumpulan data ini, jadi salah satu opsinya adalah mengurangi dimensinya menjadi sesuatu yang terlihat di monitor (2D,3D).



Di sini mudah untuk mengamati bahwa pengklasifikasi dapat mengalami masalah untuk membedakan angka 1 dan 7. Keuntungan lain adalah ini memberi kami beberapa petunjuk tentang seberapa bagus rangkaian fitur kami saat ini.

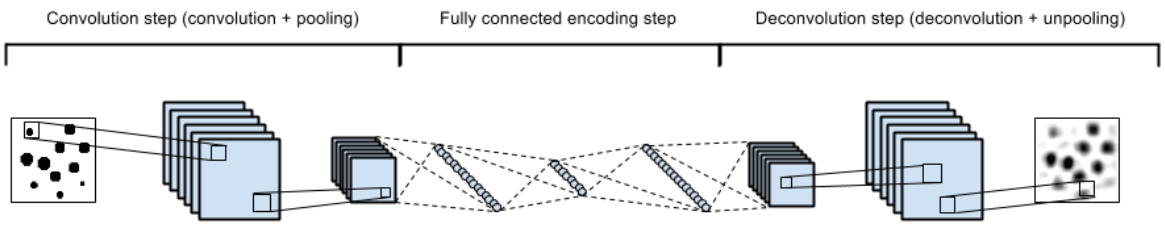
Autoencoder

Kita juga dapat menggunakan jaringan saraf untuk melakukan pengurangan dimensi. Idenya adalah kita memiliki topologi jaringan saraf yang mendekati input pada lapisan output. Di tengah autoencoder memiliki lapisan yang lebih kecil. Setelah pelatihan, lapisan tengah memiliki versi input yang dikompresi (lossy).



Konvolusi Pra-pelatihan jaringan saraf

Karena kami tidak memerlukan informasi label untuk melatih autoencoder, kami dapat menggunakannya sebagai prapelatih untuk jaringan neural konvolusi kami. Jadi di masa mendatang kami dapat memulai pelatihan Anda dengan bobot yang diinisialisasi dari pelatihan tanpa pengawasan.



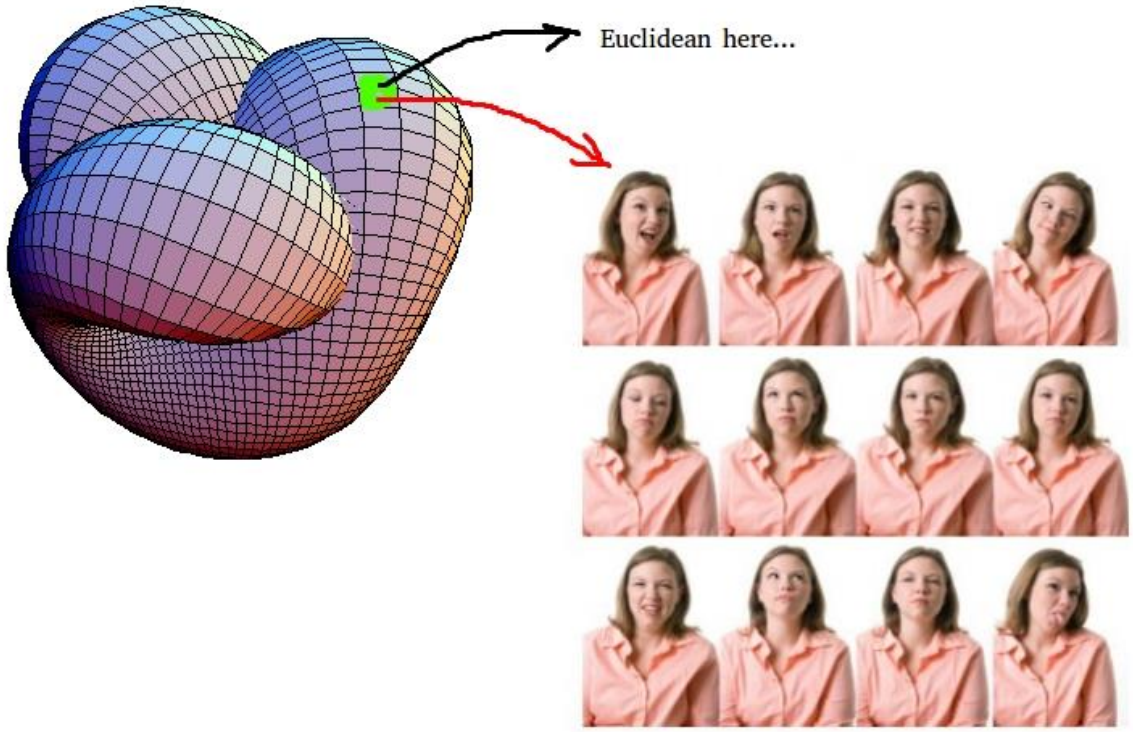
Beberapa contoh teknik ini dapat ditemukan di sini:

- Dengan Piton
- Dengan Obor

Manifold Data

Manifold Learning mengejar tujuan untuk menyematkan data yang awalnya terletak di ruang berdimensi tinggi di ruang berdimensi lebih rendah, sambil mempertahankan sifat karakteristik. Hal ini dimungkinkan karena agar data berdimensi tinggi menjadi menarik, data tersebut harus berdimensi rendah secara intrinsik. Misalnya, gambar wajah mungkin direpresentasikan sebagai titik dalam ruang dimensi tinggi (misalkan kamera Anda memiliki 5MP -- jadi gambar Anda, dengan mempertimbangkan setiap piksel terdiri dari tiga nilai [r,g,b], terletak pada ruang dimensi 15M), tetapi tidak setiap gambar 5MP adalah wajah. Wajah terletak pada sub-manifold di ruang berdimensi tinggi ini. Sub-manifold adalah Euclidean lokal, yaitu jika Anda mengambil dua titik yang sangat mirip, misalnya dua gambar kembar identik, mereka akan berdekatan di ruang euclidian

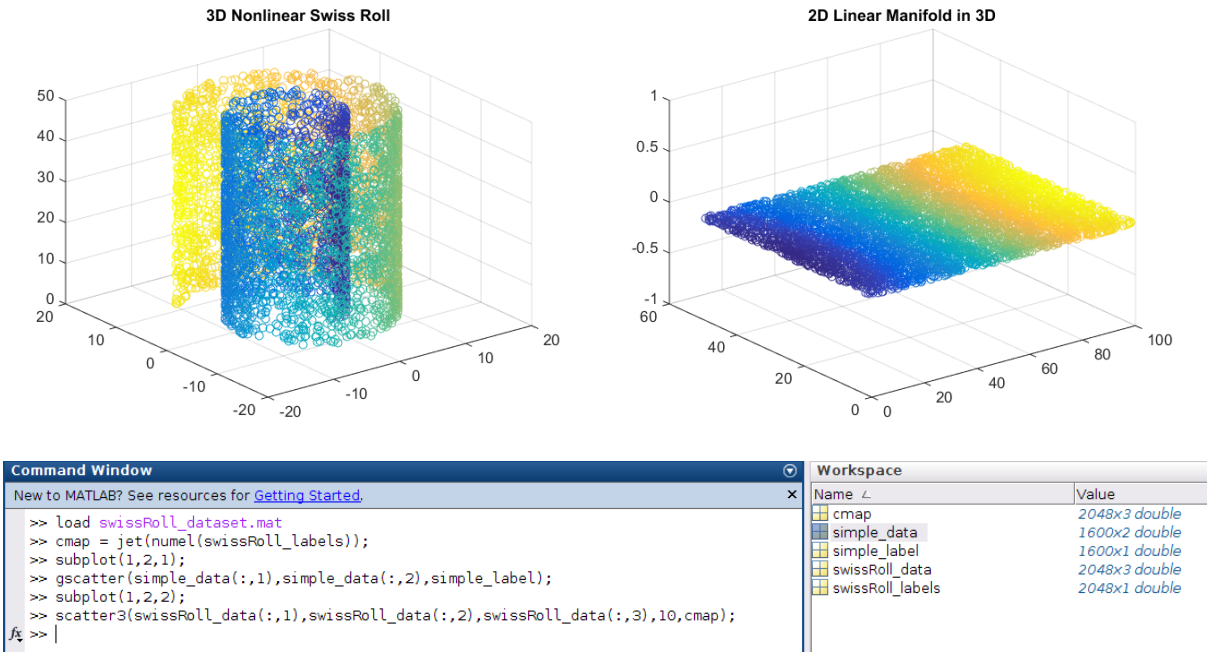
s



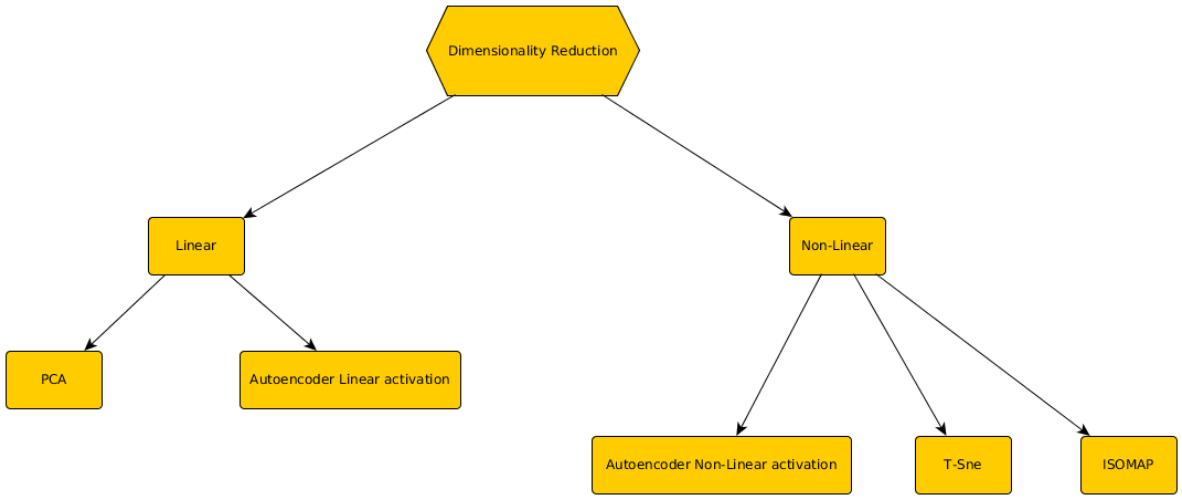
Misalnya pada dataset di atas kita memiliki manifold dimensi tinggi, tetapi permukaannya berada pada ruang dimensi yang jauh lebih rendah (hampir euclidian). Jadi pada subruang ini hal-hal seperti jarak memiliki arti.

Dengan penambahan lebih banyak fitur, distribusi data tidak akan linier, jadi teknik linier yang lebih sederhana (mis: PCA) tidak akan berguna untuk pengurangan dimensi. Pada kasus tersebut kami memerlukan hal-hal lain seperti T-Sne, Autoencoder, dll..

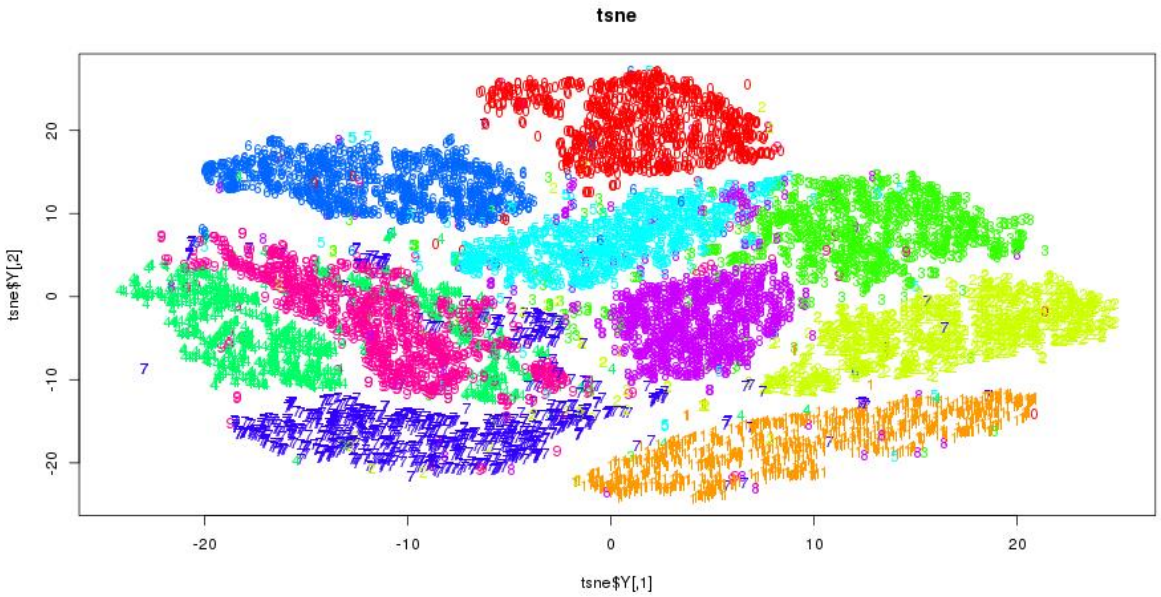
Omong-omong, pengurangan dimensi pada manifold non-linear kadang-kadang disebut pembelajaran manifold.

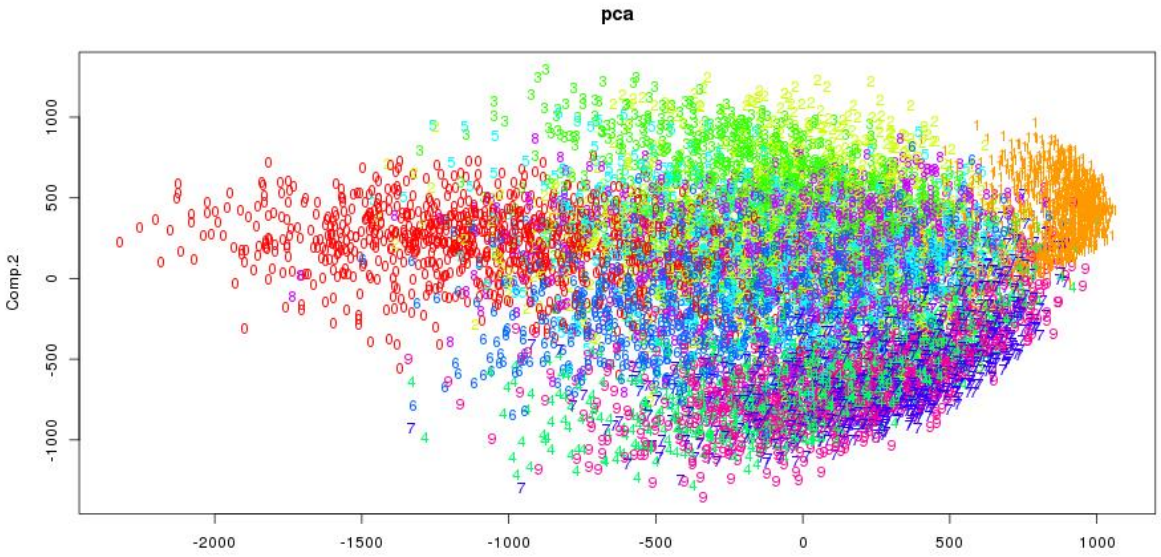


Di bawah ini kami memiliki diagram yang memandu Anda tergantung pada jenis masalahnya:



Berikut adalah perbandingan metode T-SNE dengan PCA pada dataset MNIST


tsne
tsne$Y[,2]
tsne$Y[,1]
-20
-10
0
10
20

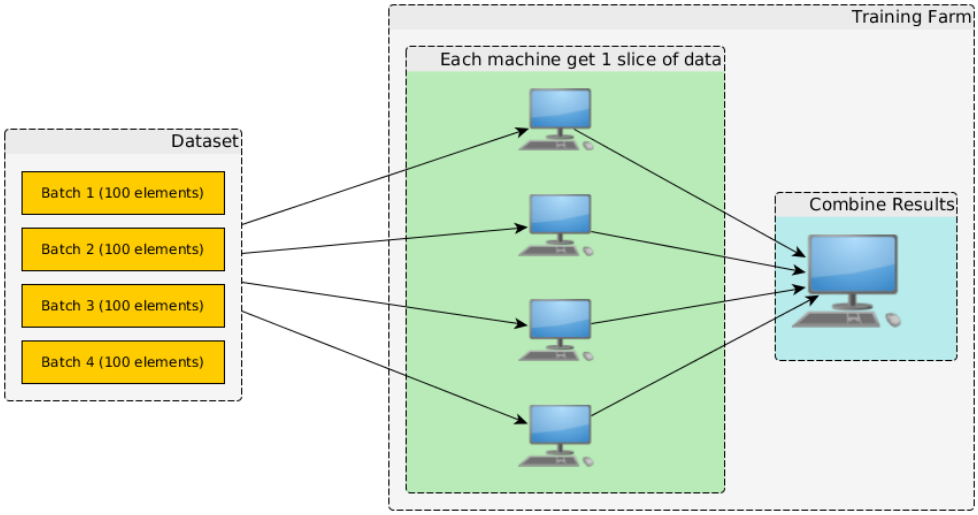

pca
Comp.2
-2000
-1500
-1000
-500
0
500
1000

Pembelajaran Terdistribusi

Pelajari bagaimana pelatihan model mendalam dapat didistribusikan ke beberapa mesin.

Pengurangan Peta

Pengurangan Peta dapat dijelaskan pada langkah-langkah berikut: 1. Pisahkan set pelatihan Anda, dalam beberapa kelompok (mis: bagi dengan jumlah pekerja di pertanian Anda: 4) 2. Berikan setiap mesin pertanian Anda 1/4 dari data 3. Lakukan propagasi Maju/Mundur, pada setiap node komputer (Semua node berbagi model yang sama) 4. Gabungkan hasil dari setiap mesin dan lakukan penurunan gradien 5. Perbarui versi model di semua node.



Contoh Model Regresi Linear

Pertimbangkan rumus penurunan gradien batch, yang merupakan penurunan gradien yang diterapkan pada semua set pelatihan:

$$\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)} - y^{(i)}) x_j^{(i)}$$

Setiap mesin akan menangani 100 elemen (Setelah membagi kumpulan data), menghitung

$temp_j^{1..4}$, Kemudian:

$$temp_j^1 = \sum_{i=1}^{100} (h_\theta(x^{(i)} - y^{(i)}) x_j^{(i)}$$
$$temp_j^2 = \sum_{i=101}^{200} (h_\theta(x^{(i)} - y^{(i)}) x_j^{(i)}$$
$$temp_j^3 = \sum_{i=201}^{300} (h_\theta(x^{(i)} - y^{(i)}) x_j^{(i)}$$
$$temp_j^4 = \sum_{i=301}^{400} (h_\theta(x^{(i)} - y^{(i)}) x_j^{(i)}$$

Setiap mesin menghitung propagasi balik dan kesalahan untuk pembagian datanya sendiri. Ingatlah bahwa semua mesin memiliki salinan model yang sama. Setelah masing-masing mesin dihitung masing-masing $temp_j^{\text{machine}}$. Mesin lain akan menggabungkan gradien tersebut, menghitung bobot baru, dan memperbarui model di semua mesin.

$$\theta_j := \theta_j - \alpha \frac{1}{400}(temp_j^1 + temp_j^2 + temp_j^3 + temp_j^4)$$

Inti dari prosedur ini adalah untuk memeriksa apakah kita dapat menggabungkan perhitungan semua node dan masih masuk akal, dalam hal perhitungan akhir.

Siapa yang menggunakan pendekatan ini

- Kafe
- Obor (Lapisan paralel)

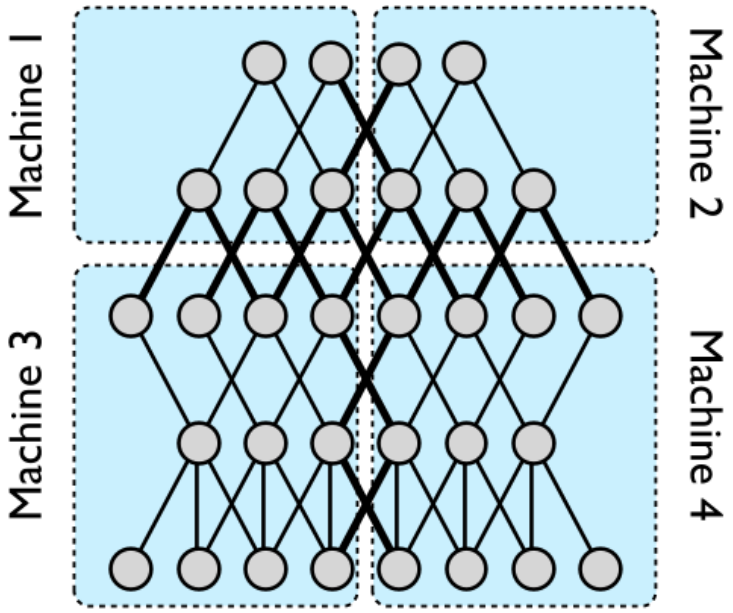
Masalah

Pendekatan ini memiliki beberapa masalah:

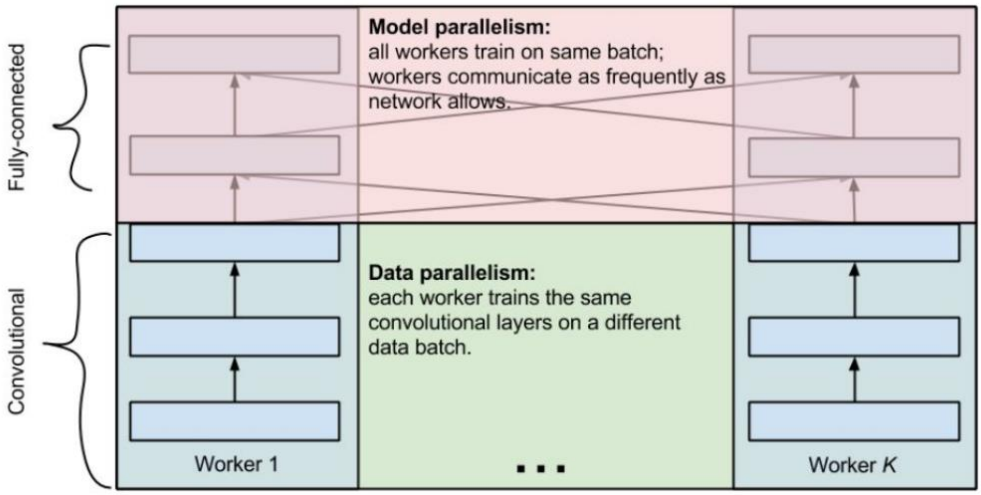
- Model lengkap harus pas di setiap mesin
- Jika modelnya terlalu besar, perlu waktu untuk memperbarui semua mesin dengan model yang sama

Membagi bobot

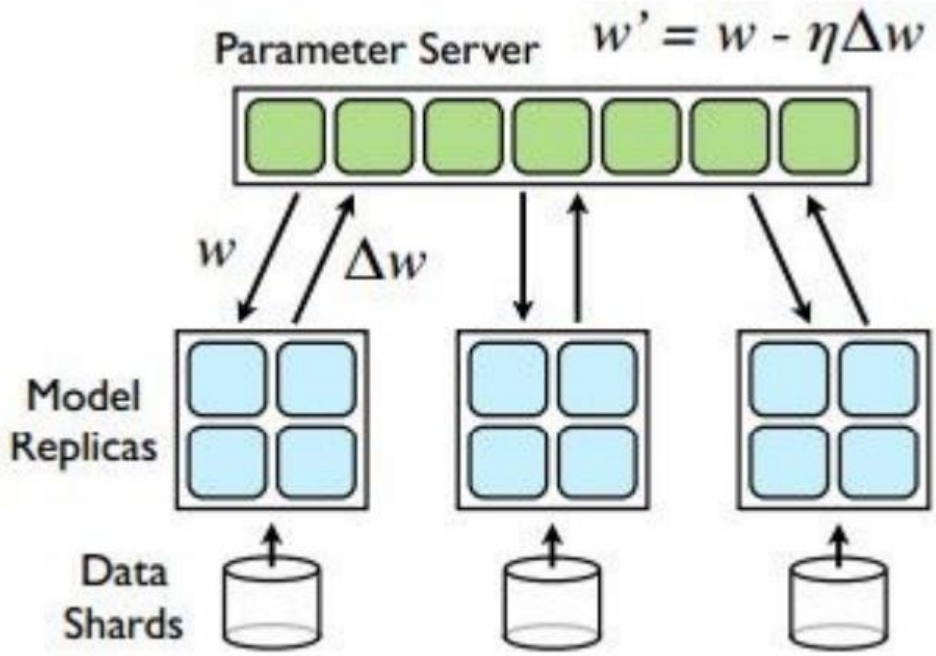
Pendekatan lain yang digunakan pada proyek google DistBelief di mana mereka menggunakan model jaringan saraf normal dengan bobot yang dipisahkan antara beberapa mesin.



Pada pendekatan ini hanya bobot (tepi tebal) yang melintasi mesin perlu disinkronkan antar pekerja. Teknik ini hanya dapat digunakan pada lapisan yang terhubung sepenuhnya. Jika Anda mencampur kedua teknik (referensi pada Alexnet) kertas, Anda melakukan pemrosesan yang terhubung sepenuhnya bersama ini (Hanya perkalian matriks), kemudian ketika Anda perlu ke bagian konvolusi, setiap lapisan konvolusi mendapatkan satu bagian dari kumpulan.



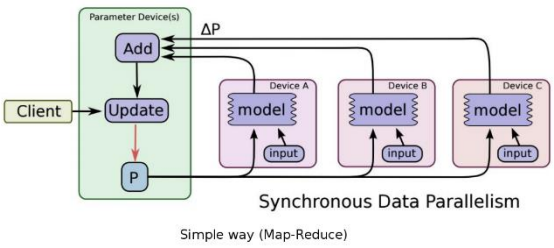
Pendekatan Google (lama)

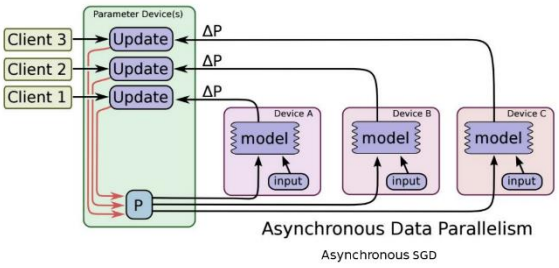


Di sini setiap replika model dilatih secara independen dengan potongan data dan server parameter yang menyinkronkan parameter antara pekerja.

Pendekatan baru Google

Sekarang google menawarkan pada Tensorflow beberapa otomatisasi untuk memilih strategi mana yang akan diikuti tergantung pada pekerjaan Anda.





Metodologi untuk penggunaan

Bab ini akan memberikan aresep tentang cara mengatasi masalah Machine learning

Proses 3 langkah

1. Tentukan apa yang perlu Anda lakukan dan bagaimana mengukur (metrik) seberapa baik/buruknya Anda.
2. Mulailah dengan model yang sangat sederhana (Beberapa lapisan)
3. Tambahkan lebih banyak kerumitan saat dibutuhkan.

Tentukan tujuan

Biasanya dengan menjadi sedikit lebih baik dari kinerja manusia (dalam hal akurasi atau waktu prediksi), sudah cukup untuk sebuah produk yang bagus.

Metrik

- Akurasi: % dari contoh yang benar
- Cakupan: Jumlah contoh yang diproses per satuan waktu
- Presisi: Jumlah deteksi yang benar
- Kesalahan: Jumlah kesalahan

Mulailah dengan model paling sederhana

Lihat apakah tersedia metode canggih untuk memecahkan masalah itu. Jika tidak tersedia gunakan resep berikut.

1. Banyak kebisingan dan sekarang banyak struktur (mis: Harga rumah dari fitur seperti jumlah kamar, ukuran dapur, dll...): Jangan gunakan pembelajaran mendalam
2. Sedikit kebisingan tetapi banyak struktur (mis: Gambar, video, teks): Gunakan pembelajaran mendalam

Contoh untuk metode Non-dalam:

- Regresi logistik
- SVM
- Pohon keputusan yang ditingkatkan (Algoritma "default" favorit sebelumnya), banyak digunakan dalam robotika

Sedalam apa

- Sedikit struktur: Gunakan hanya lapisan 2-3 lapisan yang Terhubung Sepenuhnya (Relu, Putus Sekolah, SGD+Momentum). Butuh setidaknya beberapa ribu contoh per kelas.
- Struktur spasial: Gunakan lapisan CONV (Inception/Residual, Relu, Droput, Batch-Norm, SGD+Momentum).
- Struktur berurutan (teks, pergerakan pasar): Gunakan jaringan Berulang (LSTM, SGD, kliping Gradien).

Saat menggunakan jaringan Residual/Inception, mulailah dengan contoh yang paling dangkal.

Memecahkan kesalahan kereta tinggi

Lakukan tindakan berikut pada pesanan ini:

1. Periksa untuk cacat pada data. (Perlu campur tangan manusia)
2. Periksa bug perangkat lunak di perpustakaan Anda. (Gunakan pemeriksaan gradien, itu mungkin kesalahan backprop)
3. Tune learning rate (Buat lebih kecil)
4. Membuat jaringan lebih dalam. (Anda harus mulai dengan jaringan yang dangkal di awal)

Memecahkan kesalahan tes tinggi

Lakukan tindakan berikut pada pesanan ini:

1. Lakukan lebih banyak augmentasi data (Coba juga model generatif untuk membuat lebih banyak data)
2. Tambahkan dropout dan batch-norm
3. Dapatkan lebih banyak data (Lebih banyak data memiliki pengaruh lebih besar pada Akurasi daripada yang lainnya)

Beberapa tren

Menyusul akhir 2016 Andrew Ng kuliah ini adalah topik yang perlu kita perhatikan.

1. Skalabilitas

Miliki sistem komputasi yang menskalakan dengan baik untuk lebih banyak data dan lebih banyak kompleksitas model.

2. Tim

Miliki tim Anda dibagi dengan orang-orang AI dan HPC (Cuda, OpenCl, dll...).

3. Data dulu

Data lebih penting daripada model Anda, usahakan selalu untuk mendapatkan data yang lebih berkualitas sebelum mencoba mengubah model Anda.

4. Augmentasi Data

Gunakan teknik augmentasi data normal plus model Generatif (Tanpa pengawasan).

5. Pastikan Validation Set dan Test set berasal dari distribusi yang sama

Ini akan menghindari tes atau set validasi yang tidak menceritakan kenyataan. Juga membantu untuk memeriksa apakah pelatihan Anda valid.

6. Memiliki metrik kinerja tingkat Manusia

Miliki tim ahli untuk membandingkan kinerja sistem Anda saat ini. Juga mendorong keputusan antara mendapatkan lebih banyak data, atau membuat model lebih kompleks.

7. Dataserver

Memiliki gudang data terpadu. Semua tim harus memiliki akses data, dengan kecepatan akses kualitas SSD.

8. Menggunakan Game

Menggunakan game itu keren untuk membantu menambah dataset, tetapi perhatian karena game tidak memiliki varian yang sama dari kelas yang sama dengan kehidupan nyata. Misalnya GTA tidak memiliki model mobil yang cukup dibandingkan dengan kehidupan nyata.

9. Ensemble selalu membantu

Melatih jaringan yang berbeda secara terpisah dan rata-rata hasil akhirnya selalu memberikan akurasi ekstra 2%. (Hasil terbaik Imagenet 2016 adalah ansambel sederhana)

10. Apa yang harus dilakukan jika Anda memiliki lebih dari 1000 kelas

Gunakan Softmax hierarkis untuk meningkatkan kinerja.

11. Berapa banyak sampel per kelas yang kita butuhkan untuk mendapatkan hasil yang baik

Jika melatih dari awal, gunakan jumlah parameter yang sama. Misalnya Model memiliki 1000 parameter, jadi gunakan 1000 sampel per kelas.

Jika melakukan pembelajaran transfer jauh lebih sedikit (Banyak yang belum ditentukan, lebih banyak lebih baik).

Kumpulan Data Tidak Seimbang/Hilang

Bab ini akan mengatasi beberapa masalah umum dengan kumpulan data yang mungkin Anda temui.

Data Hilang

Bayangkan Anda memiliki kumpulan data yang beberapa kolomnya mungkin memiliki data yang hilang. Apa yang harus dilakukan pada kasus ini? Menurut pengalaman saya, menyelesaikan pengabaian bukanlah pilihan yang baik. Salah satu hal yang dapat Anda lakukan adalah menggunakan statistik seluruh kumpulan data (misalnya nilai rata-rata) dan mengisi kolom tersebut dengan data tersebut. Juga beberapa algoritme menangani kasus tersebut secara otomatis (ex Naive Bayes)

Data Tidak Seimbang

Bayangkan masalah mengemudi mobil, sebagian besar waktu Anda memiliki pengemudi yang memegang setir dan sudut 0, sehingga sebagian besar waktu data Anda akan bernilai 0. Jika Anda menggunakan algoritme berbasis Neural Network, ini berarti bahwa jika Anda memprediksi menghasilkan nol, fungsi kerugian Anda akan mengembalikan nilai kecil dan propagasi balik tidak akan memperbarui bobot Anda dengan benar (Atau setidaknya akan memakan banyak waktu untuk menemukan kasus di mana kita sebenarnya mengarahkan)

Perlu juga diperhatikan bahwa beberapa algoritme tidak peduli dengan Data Tidak Seimbang (mis: Hutan Acak)

Kecerdasan buatan

Dalam ilmu komputer, mesin "cerdas" yang ideal adalah agen rasional yang fleksibel yang memahami lingkungannya dan mengambil tindakan yang memaksimalkan peluang keberhasilannya pada suatu tujuan.

Apa artinya rasional

Berikut akan kami coba jelaskan apa yang dimaksud dengan bertindak rasional:

- Maksimal mencapai tujuan yang telah ditentukan sebelumnya. (Utilitas yang diharapkan)
- Sasaran dinyatakan dalam istilah utilitas (nilai skalar non dimensi yang mewakili "kebahagiaan")
- Bersikap rasional berarti memaksimalkan utilitas yang Anda harapkan

Memiliki keputusan yang rasional (bagi kita yang cerdas), hanya terkait dengan kualitas keputusan (utilitas), bukan proses yang mengarah pada keputusan tersebut, misalnya:

- Anda memberikan jawaban yang benar karena Anda memaksa semua kemungkinan hasil
- Anda memberikan jawaban yang benar karena Anda menggunakan metode canggih yang canggih

Pada dasarnya Anda tidak peduli dengan metodenya, hanya keputusannya.

Saat bidang AI berkembang apa yang dianggap cerdas, tidak dianggap lagi setelah seseorang menemukan cara melakukannya.

Suatu sistem bersifat rasional jika ia melakukan hal yang benar dengan informasi yang tersedia.

Mengapa kita hanya peduli untuk bersikap rasional

Pada dasarnya kita tidak mengerti bagaimana otak kita bekerja, bahkan dengan kemajuan ilmu saraf saat ini, dan kemajuan daya komputasi. Kita tidak tahu bagaimana otak bekerja dan mengambil keputusan. Satu-satunya hal keren yang kita ketahui tentang otak adalah bahwa untuk membuat keputusan yang baik, **kita memerlukan ingatan dan simulasi** .

Masalah AI sentral:

1. Pemikiran
2. Pengetahuan
3. Perencanaan: Buat prediksi tentang tindakan mereka, dan pilih yang salah
4. Sedang belajar
5. Persepsi
6. Pemrosesan Bahasa Alami (NLP)

Agen

Ini adalah sistem (yaitu: program perangkat lunak) yang mengamati dunia melalui sensor dan bertindak berdasarkan lingkungan menggunakan aktuator. Ini mengarahkan aktivitasnya untuk mencapai tujuan. Agen cerdas juga dapat belajar atau menggunakan pengetahuan untuk mencapai tujuan mereka.

Jenis agen

Ada banyak jenis agen, tetapi di sini kita akan memisahkannya dalam 2 kelas

- Agen Refleks: Tidak peduli dengan efek masa depan dari tindakannya (Cukup gunakan if-table)
- Agen Perencanaan: Simulasikan konsekuensi tindakan sebelum melakukannya, menggunakan model dunia.

Agen refleks dan perencanaan bisa rasional, sekali lagi kita hanya peduli pada hasil tindakan, jika tindakan tersebut memaksimalkan utilitas yang diharapkan, maka itu rasional. kita perlu mengidentifikasi jenis agen mana yang diperlukan untuk memiliki perilaku rasional. Agen refleks atau perencanaan bisa jadi kurang optimal, tetapi biasanya perencanaan adalah ide yang bagus untuk diikuti.

Pada buku ini kita akan melihat banyak alat yang digunakan untuk menangani perencanaan. Misalnya pencarian adalah sejenis alat untuk perencanaan.

Masalah pencarian

Masalah pencarian menemukan solusi yang merupakan urutan tindakan (rencana) yang mengubah keadaan awal menjadi keadaan tujuan.

Jenis pencarian:

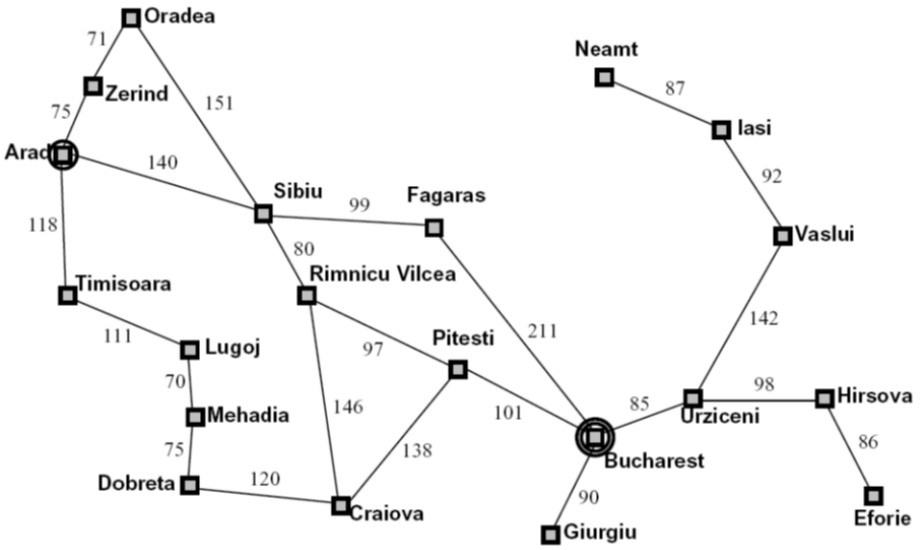
- Pencarian Tanpa Informasi: Terus mencari di mana saja sampai solusi ditemukan
- Pencarian Informasi: Memiliki semacam "informasi" yang menyatakan apakah kita sudah dekat atau tidak dengan solusi (Heuristik)

Masalah pencarian terdiri dari hal-hal berikut:

1. Ruang Negara: Apakah negara bagian perlu melakukan perencanaan
2. Fungsi penerus: Untuk setiap status x, kembalikan sekumpulan status yang dapat dijangkau dari x dengan satu tindakan
3. Tes keadaan awal dan tujuan: Memberikan poin awal dan cara memeriksa kapan perencanaan selesai

Contoh tujuan kami adalah melakukan perjalanan dari Arad, ke Bucharest

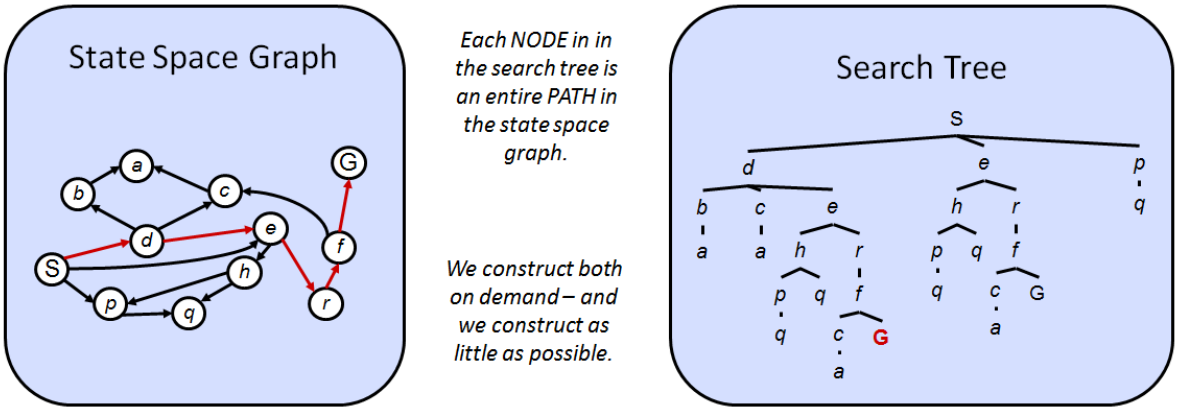
Di atas Anda memiliki keadaan dunia, kami tidak memerlukan begitu banyak detail, kami hanya membutuhkan kota, bagaimana koneksi dan jaraknya.



Pada masalah pencarian ini kami mendeteksi properti berikut:

- State space: Kota-kota (Satu-satunya variabel yang berhubungan dengan masalah pencarian ini)
- Status awal: Arad
- Fungsi Penerus: Pergi ke kota terdekat dengan biaya sebagai jarak.

Pertimbangkan peta di atas sebagai grafik, berisi node, yang tidak berulang dan bagaimana mereka terhubung bersama dengan biaya koneksinya.

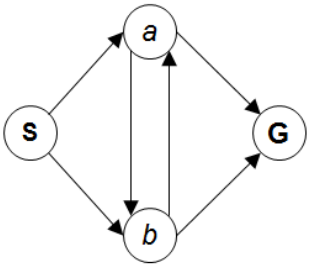


Salah satu cara untuk melakukan perencanaan adalah mengubah state space graph menjadi Search Tree, kemudian menggunakan beberapa algoritma yang mencari goal state pada tree tersebut. Di sini kami mengamati hal-hal berikut:

- Status awal akan menjadi simpul akar pohon
- Anak-anak simpul mewakili hasil yang mungkin untuk setiap keadaan.

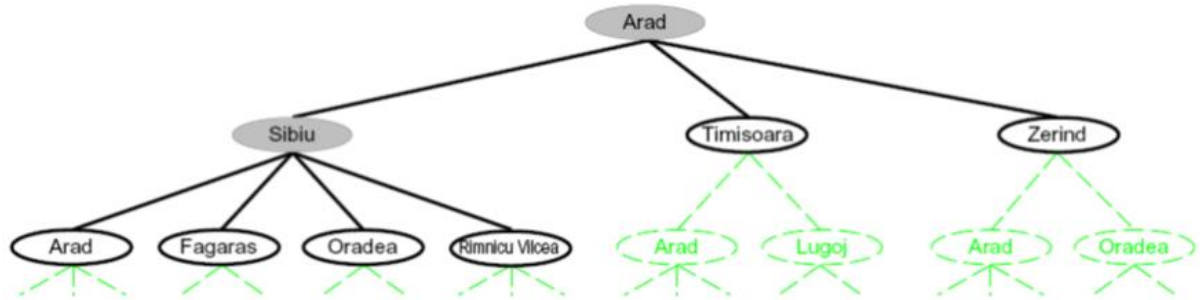
Masalahnya adalah bahwa Pohon Pencarian, atau grafik ruang keadaan bisa terlalu besar untuk muat di dalam komputer, misalnya grafik ruang keadaan berikut memiliki pohon pencarian tak terbatas.

Consider this 4-state graph:



How big is its search tree (from S)?

Apa yang harus dilakukan pada kasus-kasus itu, pada dasarnya Anda tidak menyimpan semua solusi yang mungkin dari pohon atau grafik di memori, Anda menavigasi pohon untuk kedalaman yang terbatas.



Misalnya, lihat grafik negara bagian Rumania, kita mulai di Arad (Start state)

- Arad memiliki 3 simpul anak yang mungkin, Sibiu, Timisoara dan Zerind
- Kami memilih simpul anak paling kiri Sibiu
- Kemudian kami memilih simpul anak paling kiri Arad, yang buruk, jadi kami mencoba Fagaras, lalu Oradea, dll...

Masalahnya adalah jika salah satu cabang pohon tidak terbatas, atau terlalu besar dan tidak ada solusinya, kita akan terus mencari cabangnya sampai akhir.

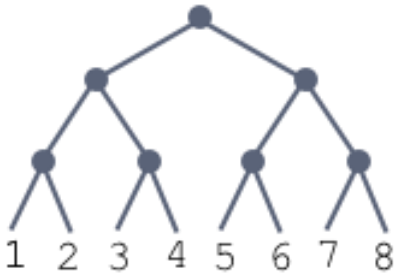
Pada saat kita memilih Sibiu pada langkah pertama, kita perlu menyimpan node lain yang mungkin (Timisoara, dan Zerind) yang disebut pinggiran pohon.

Gagasan penting yang perlu diingat dalam pencarian pohon:

- Pertama-tama pencarian pohon adalah mekanisme yang digunakan untuk perencanaan
- Perencanaan berarti Anda memiliki model dunia, jika modelnya buruk solusi Anda juga akan buruk
- Pinggiran: Atau cache dari solusi lain yang mungkin
- Bagaimana menjelajahi cabang saat ini

Faktor Percabangan

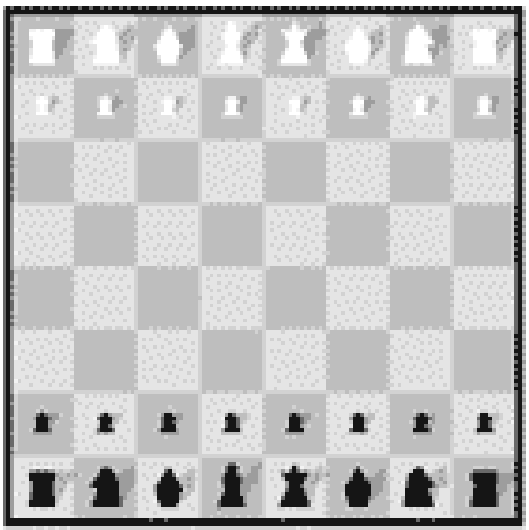
Faktor percabangan adalah jumlah anak pada setiap node pada sebuah pohon.



$b = 2$ $n = 8$

$b = 8$ $n = 8$

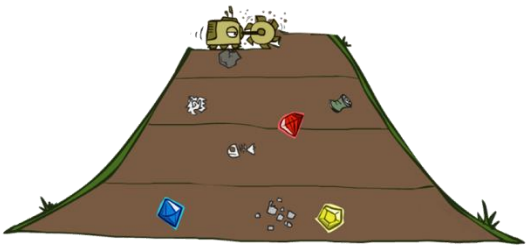
Masalah lama seperti tic-tac-toe atau masalah sederhana lainnya dapat diselesaikan dengan pohon pencarian atau semacam algoritma pohon yang dioptimalkan. Tetapi game seperti catur atau go memiliki faktor percabangan yang sangat besar, sehingga Anda tidak dapat memprosesnya secara real-time. Pada animasi di bawah ini kami membandingkan catur dan faktor percabangan.



Pencarian Pohon

Perkenalan

Pada bab ini kita akan belajar tentang beberapa cara untuk melakukan pencarian pohon. Sekadar mengingatkan dari pengenalan tree search adalah salah satu mekanisme untuk melakukan perencanaan. Perencanaan berarti agen akan mensimulasikan kemungkinan tindakan pada model kata, dan memilih salah satu yang akan memaksimalkan kegunaannya.



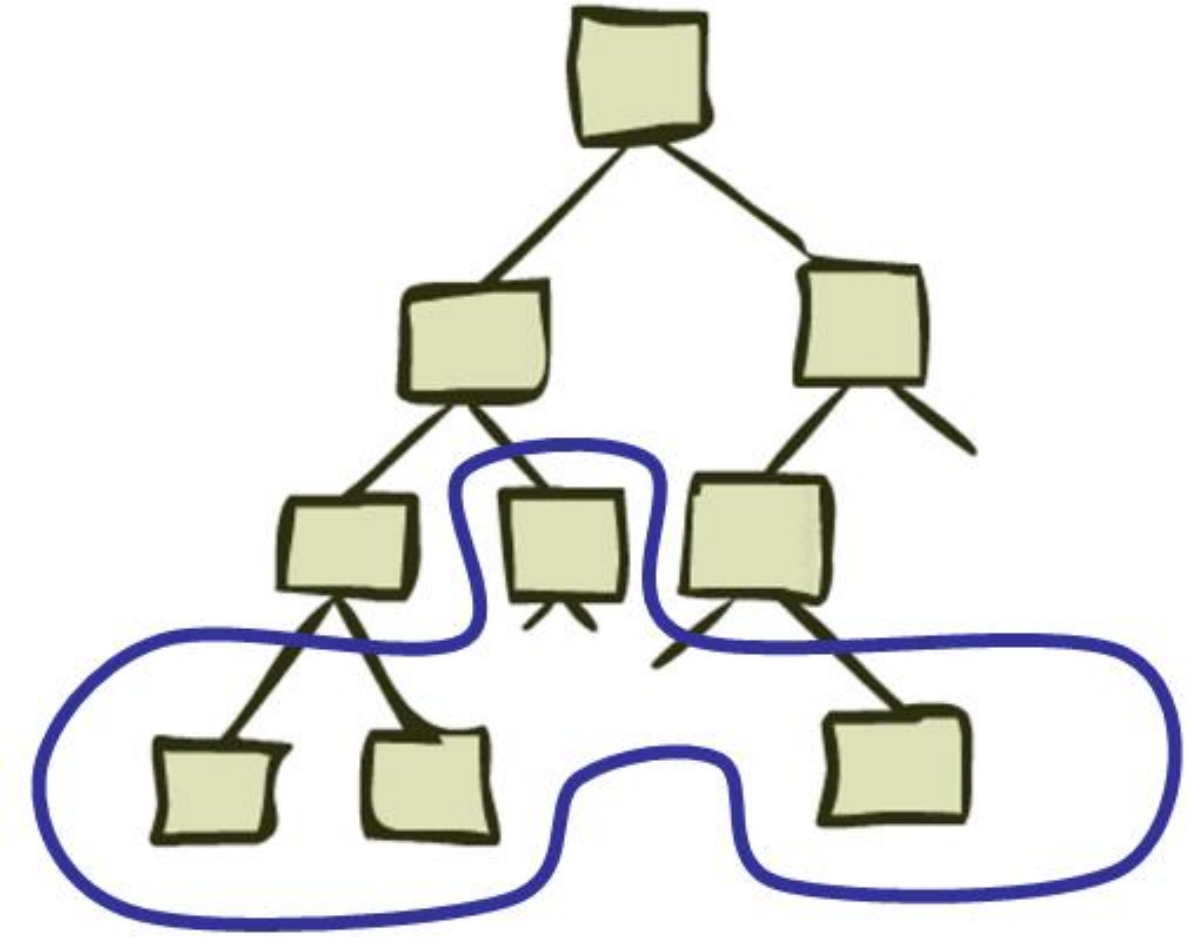
Pada bab ini kita akan mempelajari teknik pencarian pohon berikut ini

- Kedalaman pencarian pertama
- Pencarian Luas-Pertama
- Pencarian Biaya Seragam
- Pencarian serakah
- Pencarian A-star A*

Seperti yang disebutkan sebelumnya, kami tidak dapat menyimpan seluruh pohon di memori, jadi yang kami lakukan adalah memperluas pohon hanya saat Anda membutuhkannya dan melacak opsi lain yang belum Anda jelajahi.

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
   initialize the search tree using the initial state of problem
   loop do
      if there are no candidates for expansion then return failure
      choose a leaf node for expansion according to strategy
      if the node contains a goal state then return the corresponding solution
      else expand the node and add the resulting nodes to the search tree
   end
```

Untuk bagian-bagian yang masih ada di memori tetapi belum diperluas kami sebut pinggiran.



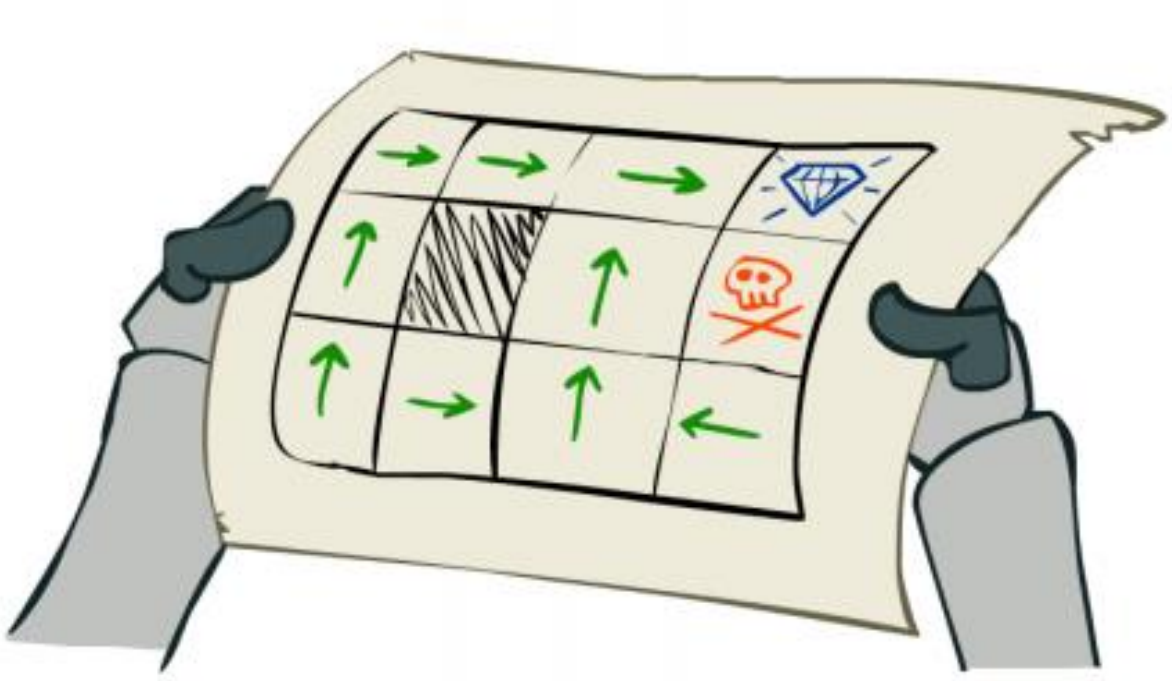
Kedalaman pencarian pertama

Proses Keputusan Markov

Perkenalan

Proses Keputusan Markov (MDP) adalah kerangka kerja yang digunakan untuk membantu membuat keputusan pada lingkungan stokastik. Tujuan kami adalah menemukan kebijakan, yang merupakan peta yang memberi kami semua tindakan optimal pada setiap keadaan di lingkungan kami.

MDP entah bagaimana lebih kuat daripada perencanaan sederhana, karena kebijakan Anda akan memungkinkan Anda untuk melakukan tindakan optimal bahkan jika terjadi kesalahan di sepanjang jalan. Perencanaan sederhana ikuti saja rencana setelah Anda menemukan strategi terbaik.



Apa itu Status

Pertimbangkan status sebagai ringkasan (kemudian disebut ruang-status) dari semua informasi yang diperlukan untuk menentukan apa yang terjadi selanjutnya. Ada 2 jenis ruang keadaan:

- World-state: Biasanya besar, dan tidak tersedia untuk agen.
- Agent-State: Lebih kecil, memiliki semua variabel yang diperlukan untuk membuat keputusan terkait dengan utilitas yang diharapkan agen.

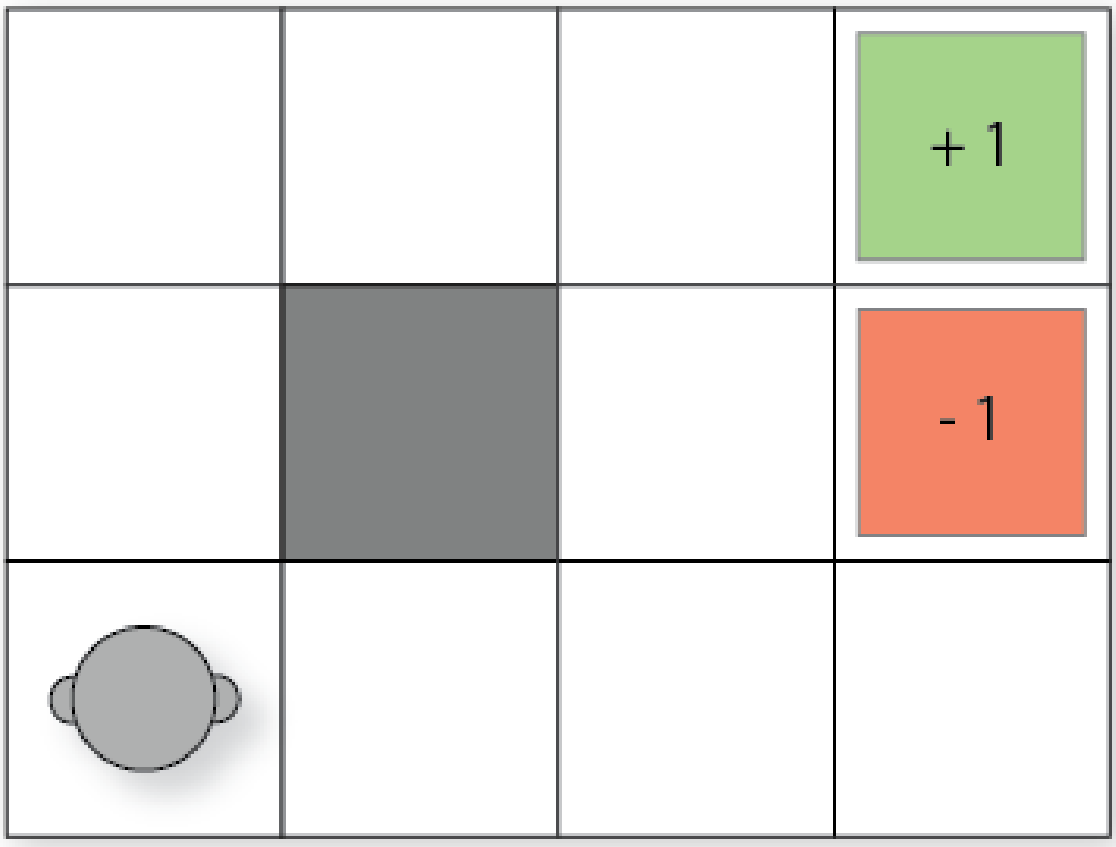
Properti Markovian

Pada dasarnya Anda tidak memerlukan keadaan sebelumnya untuk melakukan keputusan yang optimal, yang Anda butuhkan hanyalah keadaan saat ini *S*. Ini karena Anda dapat menyandikan pada keadaan Anda saat ini semua yang Anda butuhkan dari masa lalu untuk membuat keputusan yang baik. Tetap saja sejarah itu penting...

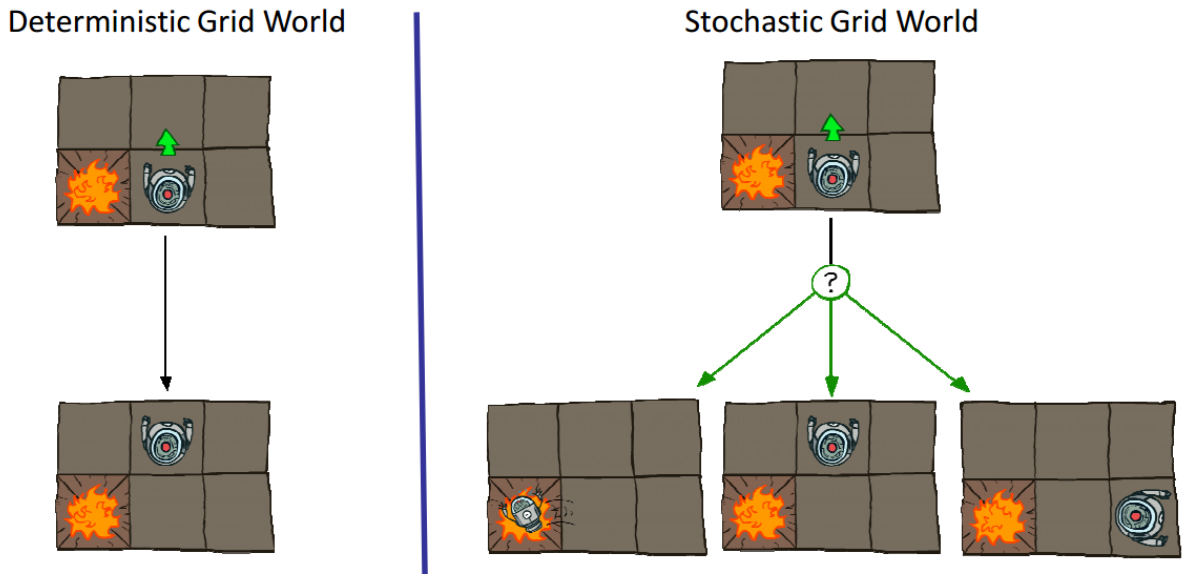
Lingkungan

Untuk menyederhanakan alam semesta kita bayangkan dunia grid, di sini tujuan agen Anda adalah tiba di blok hijau, dan menghindari blok merah. Tindakan Anda yang tersedia adalah:

↑,↓,←,→↑,↓,←,→



Masalahnya adalah kita tidak hidup di dunia deterministik yang sempurna, jadi tindakan kita bisa memiliki hasil yang berbeda:



Misalnya ketika kita memilih tindakan ke atas, kita memiliki probabilitas 80% untuk benar-benar naik, dan 10% untuk ke kiri atau ke kanan. Juga jika Anda memilih untuk pergi ke kiri atau ke kanan, Anda memiliki 80% kemungkinan untuk pergi ke kiri dan 10% untuk naik atau turun.

Berikut adalah bagian yang paling penting:

- Serikat: Satu set kemungkinan negara S
- Model: $T(s, sebuah, S') \therefore P(s' | s, a)$ Probabilitas untuk pergi ke negara bagian S' ketika Anda melakukan Tindakan A saat Anda berada di status bagian S disebut juga model transisi.
- Tindakan: $A(s)$, hal-hal yang dapat Anda lakukan pada keadaan tertentu S
- Hadiah: $R(s)$, nilai skalar yang Anda peroleh selama berada dalam keadaan.
- Kebijakan: $\Pi(s) \rightarrow A$, tujuan kami, adalah peta yang menunjukkan tindakan optimal untuk setiap negara bagian
- Kebijakan optimal: $\Pi*(s) \rightarrow A$, adalah kebijakan yang memaksimalkan imbalan yang Anda harapkan $R(s)$

Dalam pembelajaran penguatan kita akan mempelajari kebijakan yang optimal dengan coba-coba.

Memecahkan MDP dengan Pemrograman Dinamis

Seperti yang dinyatakan sebelumnya, MDP adalah alat untuk memodelkan masalah keputusan, tetapi bagaimana kita menyelesaikannya? Untuk menyelesaikan MDP kita membutuhkan Pemrograman Dinamis, lebih khusus lagi persamaan Bellman.

Tapi pertama-tama apa itu pemrograman dinamis? Pada dasarnya ini adalah metode yang membagi masalah menjadi sub-masalah yang lebih sederhana yang lebih mudah dipecahkan, itu hanya strategi bagi dan taklukkan.

Pemrograman dinamis adalah metode pengoptimalan matematis dan metode pemrograman komputer, tetapi keduanya mengikuti mekanisme pembagian dan penaklukan ini. Namun dalam matematika sering digunakan sebagai alat pengoptimalan. Pada pemrograman sering diimplementasikan dengan rekursi dan digunakan pada masalah seperti menemukan jalur terpendek pada grafik dan urutan pembangkitan.

Anda juga mungkin menemukan istilah yang disebut memoisasi yang merupakan sesuatu yang digunakan orang komputer untuk meningkatkan kinerja algoritme pembagian dan penaklukan tersebut dengan menghafal sub masalah yang telah dihitung.

Pembelajaran Penguatan

Perkenalan

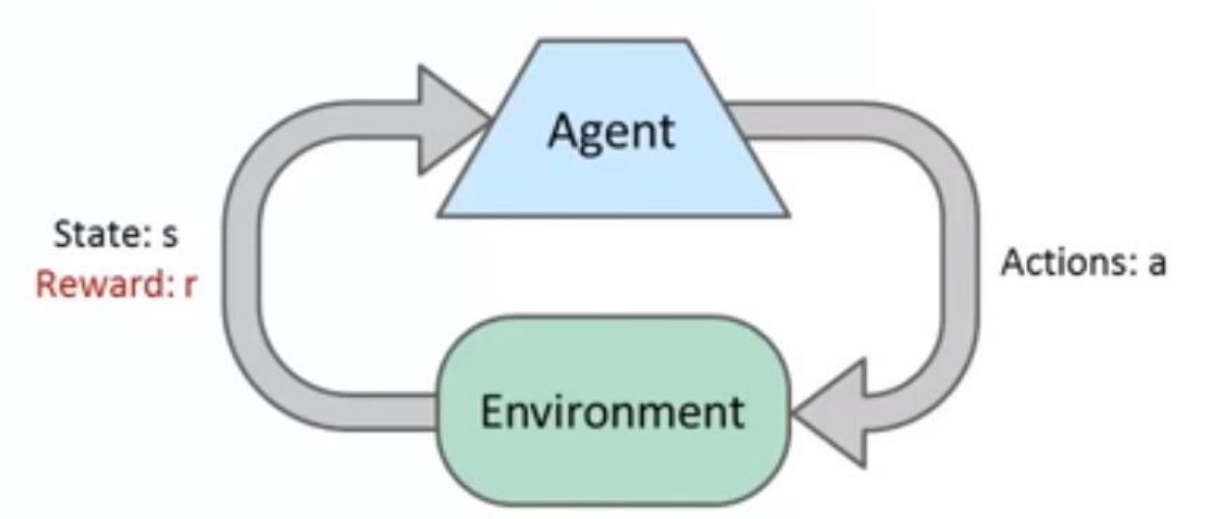
Pada bab ini kita akan mempelajari dasar-dasar Reinforcement learning (Rl), yang merupakan cabang dari pembelajaran mesin yang berkaitan dengan mengambil urutan tindakan untuk memaksimalkan beberapa hadiah.

Pada dasarnya seorang RL tidak tahu apa-apa tentang lingkungan, ia mempelajari apa yang harus dilakukan dengan menjelajahi lingkungan. Itu menggunakan tindakan, dan menerima status dan hadiah. Agen hanya dapat mengubah lingkungan Anda melalui tindakan.

Salah satu kesulitan besar Rl adalah bahwa beberapa tindakan membutuhkan waktu untuk menghasilkan hadiah, dan mempelajari dinamika ini bisa jadi menantang. Juga imbalan yang diterima lingkungan tidak terkait dengan tindakan terakhir, tetapi beberapa tindakan di masa lalu.

Beberapa konsep:

- Agen mengambil tindakan di lingkungan dan menerima status dan hadiah Tujuannya adalah untuk menemukan kebijakan $(s)=action$ yang memaksimalkan fungsi utilitasnya $U(s)$
- Terinspirasi oleh penelitian tentang psikologi dan pembelajaran hewan



Disini kita tidak mengetahui tindakan mana yang akan menghasilkan reward, juga kita tidak mengetahui kapan suatu tindakan akan menghasilkan reward, terkadang Anda melakukan suatu tindakan yang akan memakan waktu untuk menghasilkan reward.

Pada dasarnya semua dipelajari dengan interaksi dengan lingkungan.

Komponen pembelajaran penguatan:

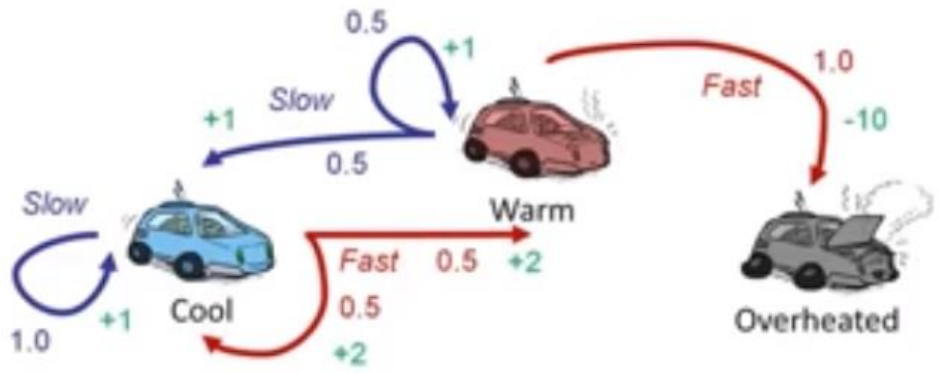
- Agen: Robot kami
- Lingkungan: Game, atau tempat tinggal agen.
- Satu set negara bagian $S \in S$
- Kebijakan: Petakan antara negara bagian ke tindakan
- Fungsi Hadiah $R(s, sebuah, S')$: Memberikan hadiah langsung untuk setiap negara bagian
- Fungsi Nilai: Memberikan jumlah total hadiah yang dapat diharapkan agen dari status tertentu ke semua kemungkinan status dari status tersebut. Dengan fungsi nilai Anda dapat menemukan kebijakan.
- Model $T(s, sebuah, S')$ (Opsional): Digunakan untuk melakukan perencanaan, bukan pendekatan trial-and-error sederhana yang umum untuk pembelajaran Reinforcement.

  Di Sini S' berarti keadaan yang mungkin terjadi setelah kita melakukan suatu Tindakan A pada negara bagian S

Ada varian pembelajaran Reinforcement yang disebut Deep Reinforcement Learning di mana Anda menggunakan Neural Networks sebagai penaksir fungsi untuk hal-hal berikut:

- Kebijakan (Pilih tindakan selanjutnya saat Anda berada di beberapa negara tertentu)
- Nilai-Fungsi (Ukur seberapa baik keadaan atau pasangan keadaan-aksi saat ini)
- Seluruh dinamika Model/Dunia, sehingga Anda dapat memprediksi status dan hadiah berikutnya.

Dalam benak kita masih akan berpikir bahwa ada proses keputusan Markov (MDP), yang memiliki:

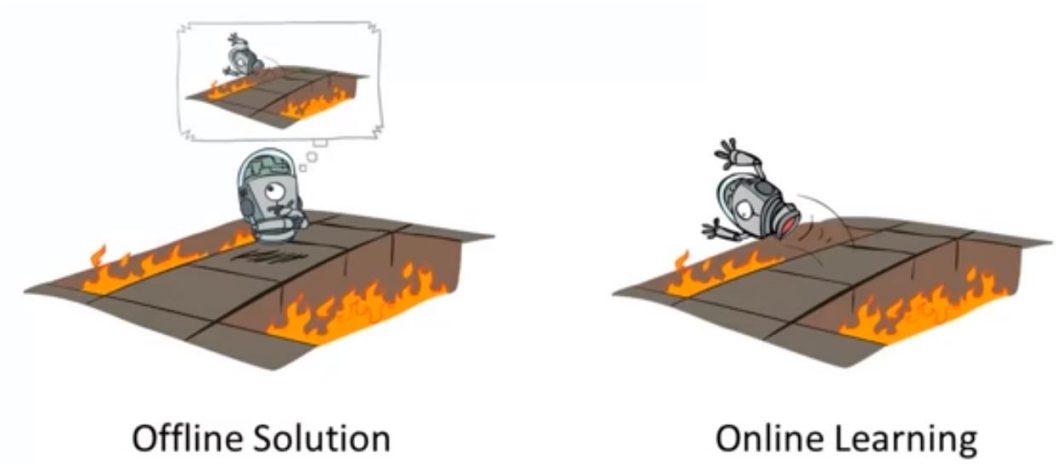


Kami sedang mencari kebijakan $\pi(s)$, yang berarti peta yang memberi kita tindakan optimal untuk setiap keadaan Satu-satunya masalah adalah kita tidak memilikinya sekarang secara eksplisit $T(s, sebuah, S')$ atau $R(s, sebuah, S')$, jadi kita tidak tahu keadaan mana yang baik atau tindakan apa yang dilakukan. Satu-satunya cara untuk mempelajari hal-hal itu adalah dengan mencobanya dan belajar dari sampel kami.

Pada pembelajaran Reinforcement kita tahu bahwa kita bisa bergerak cepat atau lambat (Tindakan) dan jika kita dingin, hangat atau kepanasan (keadaan). Tapi kita tidak tahu apa yang dilakukan tindakan kita dalam kaitannya dengan bagaimana mereka mengubah keadaan.

Offline (MDP) vs Online (RL)

Perbedaan lainnya adalah bahwa sebagai agen perencanaan MDP normal, temukan solusi optimal, dengan cara pencarian dan simulasi (Perencanaan). Seorang agen Rl belajar dari coba-coba, sehingga ia akan melakukan sesuatu yang buruk sebelum mengetahui bahwa ia seharusnya tidak melakukannya. Juga untuk mengetahui bahwa sesuatu itu benar-benar buruk atau baik, agen akan mengulanginya berkali-kali.



Bagaimana itu bekerja

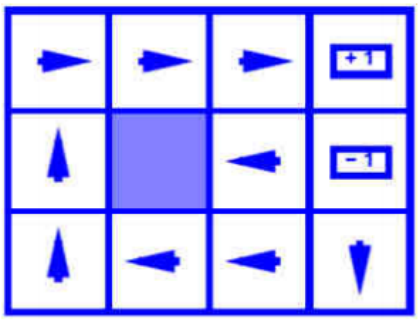
Kita akan belajar perkiraan tentang apa yang dilakukan tindakan dan penghargaan apa yang kita dapatkan dari pengalaman. Misalnya kita bisa melakukan tindakan secara acak

Hadiah terlambat

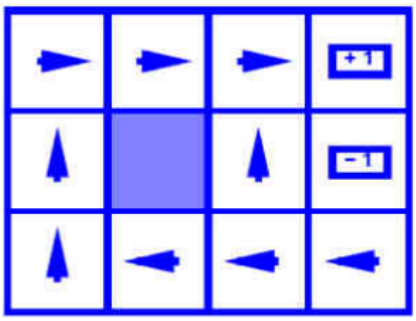
Kami memaksa MDP untuk mendapatkan hadiah yang bagus sesegera mungkin dengan memberikan beberapa diskon γ

imbalan dari waktu ke waktu. Pada dasarnya Anda memodulasi seberapa terburu-buru agen Anda dengan memberikan lebih banyak nilai negative $R(s)$ lembur.

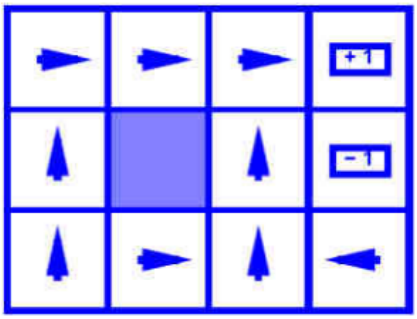
Anda juga dapat mengubah perilaku agen Anda dengan memberikan jumlah waktu yang dimiliki agen Anda.



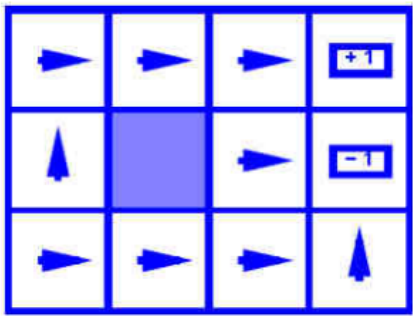
R(s) = -0.01



R(s) = -0.03



R(s) = -0.4



R(s) = -2.0

Eksplorasi dan Eksploitasi

Salah satu masalah pembelajaran Penguatan adalah dilema eksplorasi vs eksploitasi.

- Eksploitasi: Buat keputusan terbaik dengan pengetahuan yang sudah kita ketahui (mis: Pilih tindakan dengan nilai Q terbesar)
- Eksplorasi: Kumpulkan lebih banyak informasi dengan melakukan berbagai tindakan (stokastik) dari keadaan yang diketahui.

Contoh: Restoran

- Eksploitasi: Pergi ke restoran favorit, saat Anda lapar (berikan hadiah yang diketahui)
- Eksplorasi: Mencoba restoran baru (Bisa memberi lebih banyak hadiah)

Salah satu teknik untuk terus mengeksplorasi sedikit adalah penggunaan $\epsilon\text{–}greedy$ eksplorasi di mana sebelum kita mengambil tindakan kita menambahkan beberapa faktor acak.

Pemrosesan Bahasa Alami

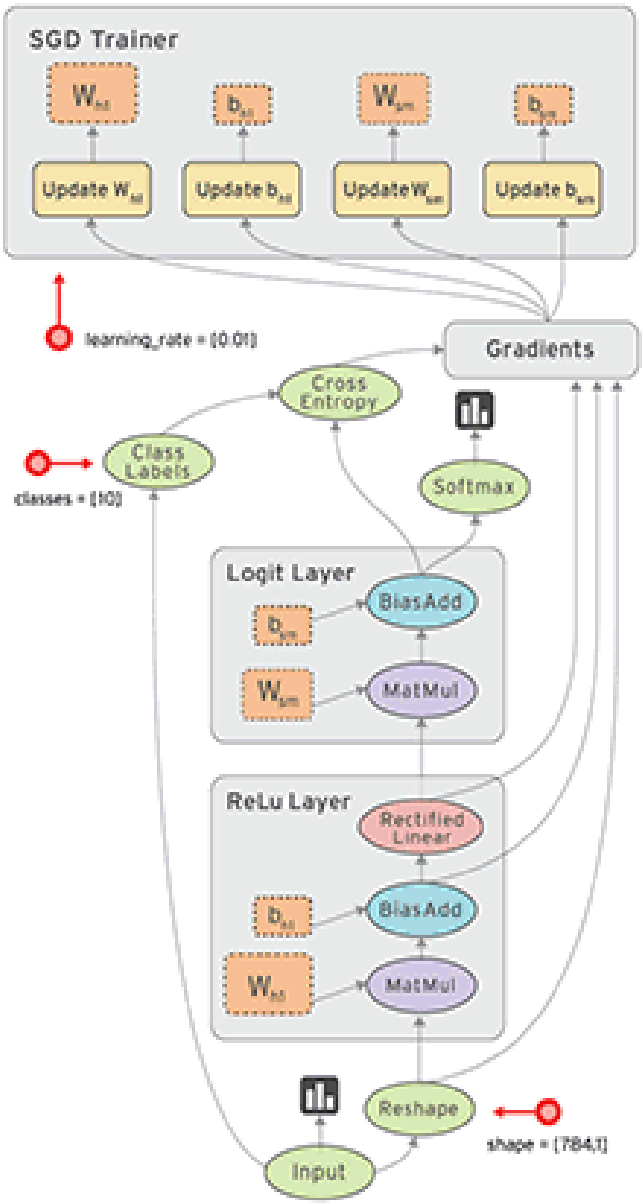
Perkenalan

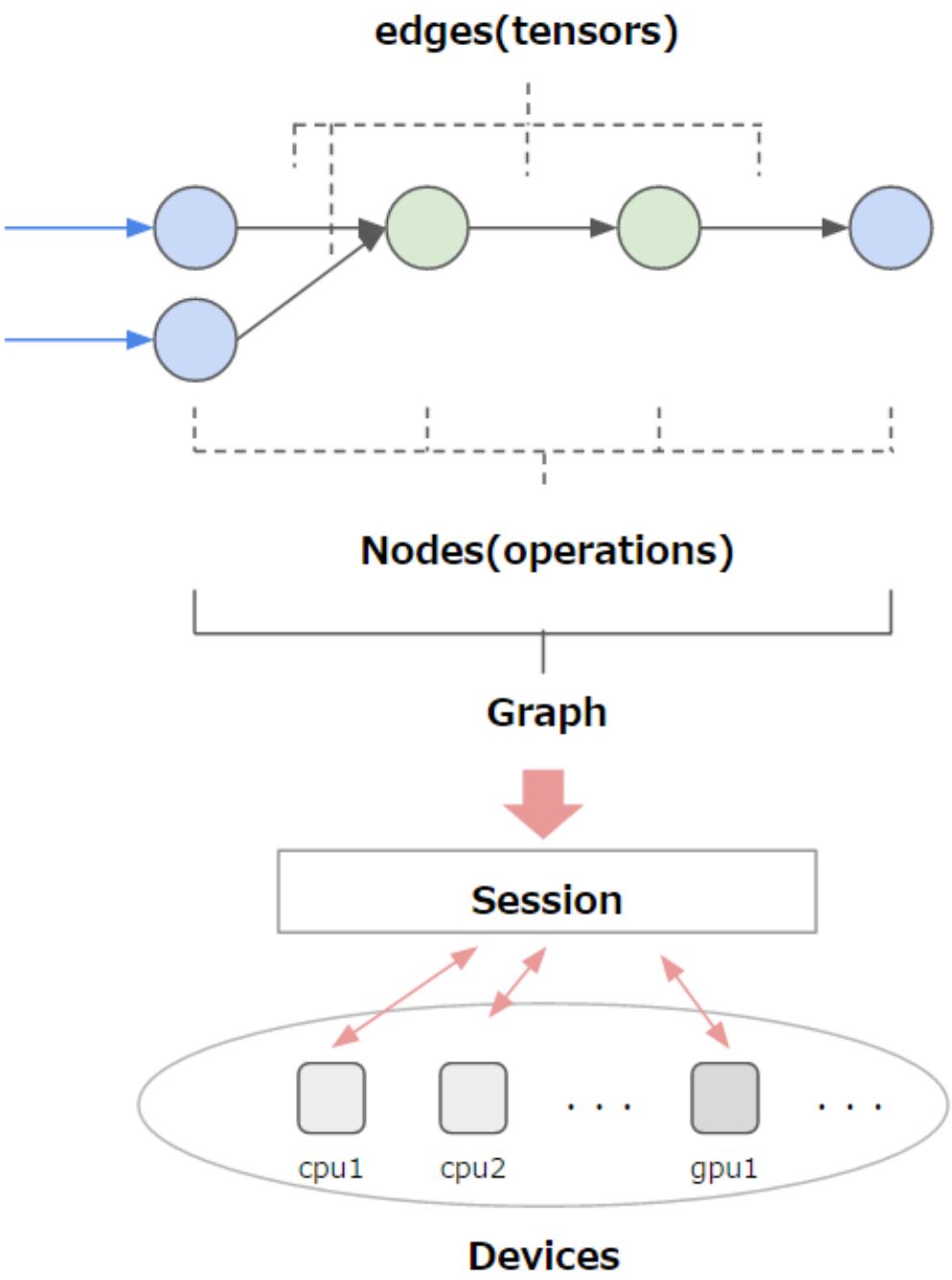
Dalam bab ini kita akan membahas topik pembelajaran mesin tentang Pemrosesan Bahasa Alami

Tensorflow

Perkenalan

Pada bab ini kita akan belajar tentang tensorflow, yang merupakan perpustakaan google untuk pembelajaran mesin. Dengan kata sederhana itu adalah perpustakaan untuk perhitungan numerik yang menggunakan grafik, pada grafik ini node adalah operasinya, sedangkan tepi grafik ini adalah tensor. Hanya untuk mengingat tensor, adalah matriks multidimensi, yang akan mengalir pada grafik aliran tensor.





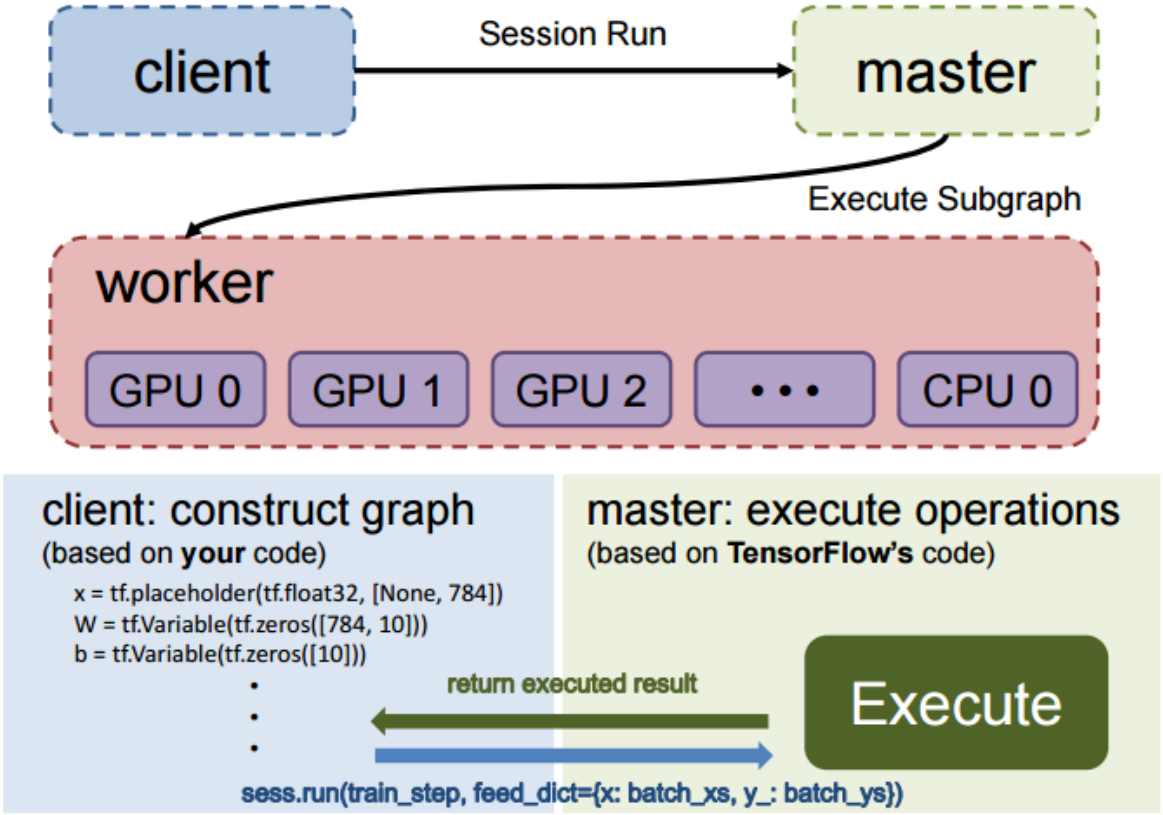
Setelah grafik komputasi ini dibuat, akan dibuat sesi yang dapat dijalankan oleh banyak CPU, GPU terdistribusi atau tidak. Berikut adalah komponen utama tensorflow:

1. Variabel: Pertahankan nilai di antara sesi, gunakan untuk bobot/bias
2. Node: Operasi
3. Tensor: Sinyal yang lewat dari/ke node
4. Placeholder: Digunakan untuk mengirim data antara program Anda dan grafik tensorflow
5. Sesi: Tempatkan saat grafik dieksekusi.

Implementasi TensorFlow menerjemahkan definisi grafik menjadi operasi yang dapat dijalankan yang didistribusikan ke seluruh sumber daya komputasi yang tersedia, seperti CPU atau salah satu kartu GPU komputer Anda. Secara umum Anda tidak perlu menentukan CPU atau GPU secara eksplisit. TensorFlow menggunakan GPU pertama Anda, jika ada, untuk sebanyak mungkin operasi.

Tugas Anda sebagai "klien" adalah membuat grafik ini secara simbolis menggunakan kode (C/C++ atau python), dan meminta tensorflow untuk mengeksekusi grafik ini. Seperti yang Anda bayangkan, kode tensorflow untuk "node eksekusi" tersebut adalah beberapa kode kinerja tinggi C/C++, CUDA. (Juga sulit dimengerti).

Misalnya, biasanya membuat grafik untuk merepresentasikan dan melatih jaringan saraf dalam fase konstruksi, lalu berulang kali menjalankan serangkaian operasi pelatihan dalam grafik dalam fase eksekusi.



Menginstal

Jika Anda sudah memiliki mesin dengan python (anaconda 3.5) dan driver nvidia cuda diinstal (7.5), instal tensorflow sederhana

```
export
TF_BINARY_URL=https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow-
0.10.0rc0-cp35-cp35m-linux_x86_64.whl
```

```
sudo pip3 install --ignore-installed --upgrade $TF_BINARY_URL
```

Jika Anda masih perlu menginstal beberapa driver cuda, lihat di sini untuk petunjuknya

Contoh sederhana

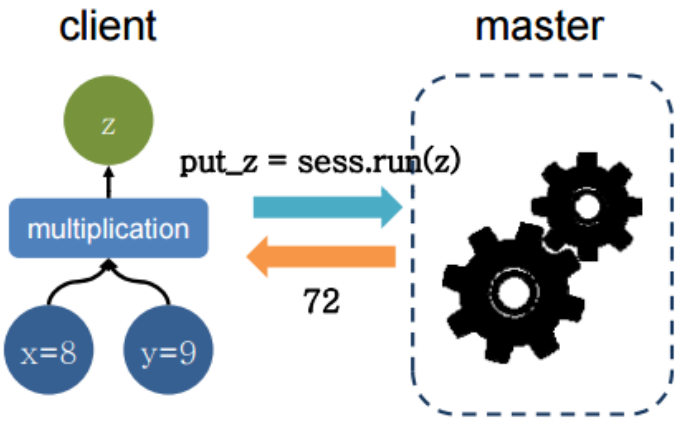
Sama seperti halo dunia mari kita buat grafik yang hanya mengalikan 2 angka. Di sini perhatikan beberapa bagian kode.

- Impor perpustakaan tensorflow
- Bangun grafiknya
- Buat sesi
- Jalankan sesi

Perhatikan juga bahwa pada contoh ini kita meneruskan ke model kita beberapa nilai konstan sehingga tidak begitu berguna dalam kehidupan nyata.

source code

```
import tensorflow as tf

x = tf.constant(8)
y = tf.constant(9)
z = tf.mul(x,y)

sess = tf.Session()

out_z = sess.run(z)

print('out_z: %d' % out_z)
```



output

```
out_z: 72
```

Bertukar data

Tensorflow memungkinkan pertukaran data dengan variabel grafik Anda melalui "placeholder". Placeholder tersebut dapat ditetapkan saat kami meminta sesi untuk dijalankan. Bayangkan placeholder sebagai cara untuk mengirim data ke grafik Anda saat Anda menjalankan sesi "session.run"

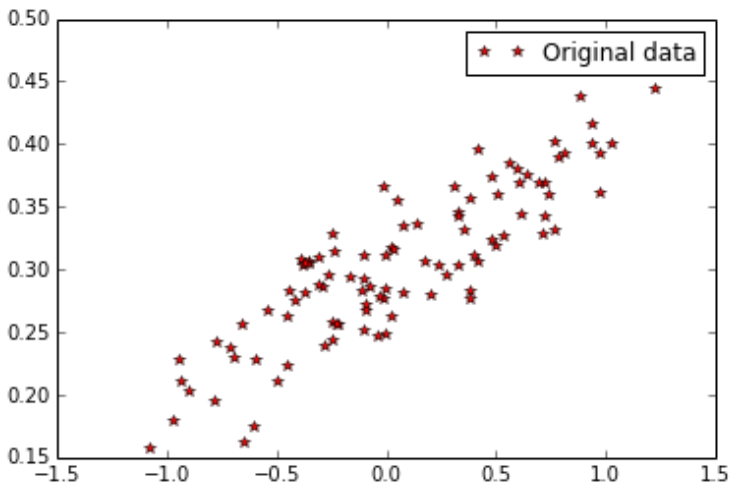
```
# Import tensorflow
import tensorflow as tf

# Build graph
a = tf.placeholder('float')
b = tf.placeholder('float')

# Graph
y = tf.mul(a,b)

# Create session passing the graph
session = tf.Session()
# Put the values 3,4 on the placeholders a,b
print session.run(y,feed_dict={a: 3, b:4})
```

Regresi Linier pada aliran tensor

Sekarang kita akan melihat cara membuat sistem regresi linier pada tensorflow

```
# Import libraries (Numpy, Tensorflow, matplotlib)
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
get_ipython().magic(u'matplotlib inline')

# Create 100 points following a function y=0.1 * x + 0.3 with some normal random distribution
num_points = 100
vectors_set = []
for i in xrange(num_points):
    x1 = np.random.normal(0.0, 0.55)
```

```
    y1 = x1 * 0.1 + 0.3 + np.random.normal(0.0, 0.03)
    vectors_set.append([x1, y1])

x_data = [v[0] for v in vectors_set]
y_data = [v[1] for v in vectors_set]

# Plot data
plt.plot(x_data, y_data, 'r*', label='Original data')
plt.legend()
plt.show()
```



Sekarang kita akan mengimplementasikan grafik dengan fungsi $y=W*X_{data}+B$, fungsi kerugian $loss=mean[(y-y_{data})^2]$. Fungsi kerugian akan mengembalikan nilai skalar dengan rata-rata semua jarak antara data kami, dan prediksi model.

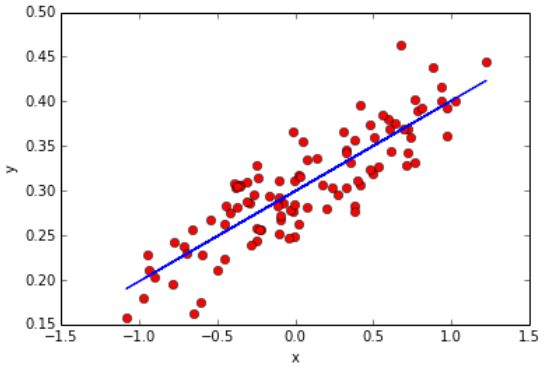
```
# Create our linear regression model
# Variables resides internally inside the graph memory
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1.0]))
y = W * x_data + b
```

```
# Define a loss function that take into account the distance between
# the prediction and our dataset
loss = tf.reduce_mean(tf.square(y-y_data))

# Create an optimizer for our loss function (With gradient descent)
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
```

Dengan grafik yang dibuat, tugas kita adalah membuat sesi untuk menginisialisasi semua variabel grafik kita, yang dalam hal ini adalah parameter model kita. Kemudian kita juga perlu memanggil sesi x kali untuk melatih model kita.

```
# Run session
# Initialize all graph variables
init = tf.initialize_all_variables()
# Create a session and initialize the graph variables (Will acutally run now...)
session = tf.Session()
session.run(init)

# Train on 8 steps
for step in xrange(8):
    # Optimize one step
    session.run(train)
    # Get access to graph variables(just read) with session.run(varName)
    print("Step=%d, loss=%f, [W=%f b=%f]") %
(step,session.run(loss),session.run(W),session.run(b))

# Just plot the set of weights and bias with less loss (last)
plt.plot(x_data, y_data, 'ro')
plt.plot(x_data, session.run(W) * x_data + session.run(b))
plt.xlabel('x')
```

```
plt.ylabel('y')

plt.legend()

plt.show()


# Close the Session when we're done.

session.close()
```



Memuat data

Hampir sepenuhnya tergantung pada Anda untuk memuat data pada tensorflow, yang berarti Anda harus mem-parsing datanya sendiri. Misalnya, salah satu opsi untuk klasifikasi gambar adalah memiliki file teks dengan semua nama file gambar, diikuti oleh kelasnya. Misalnya:

pelatihanFile.txt

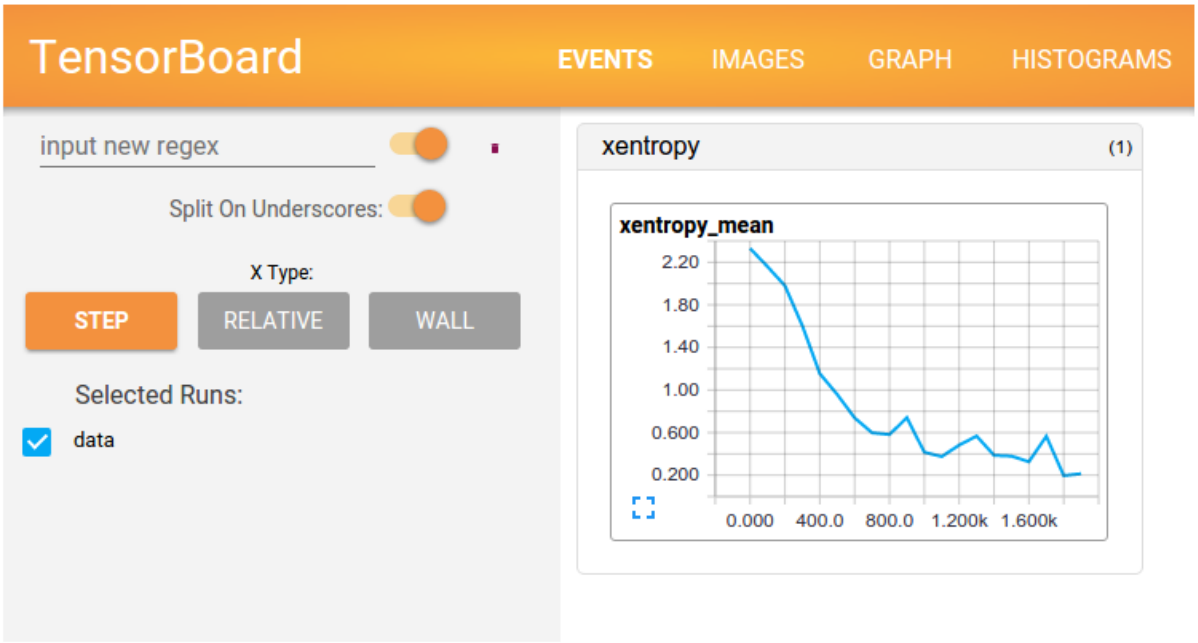
```
image1.png 0

image2.png 0

image3.png 1

image4.png 1

image5.png 2

image6.png 2
```

API umum untuk memuat data akan menjadi seperti ini.

```
train_data, train_label = getDataFromFile('trainingFile.txt')

val_data, val_label = getDataFromFile('validationFile.txt')


## Give to your graph through placeholders...
```

Papan tensor

Tensorflow menawarkan solusi untuk membantu memvisualisasikan apa yang terjadi pada grafik Anda. Alat ini disebut Tensorboard, pada dasarnya adalah halaman web tempat Anda dapat men-debug grafik Anda, dengan memeriksa variabelnya, koneksi node, dll...



Untuk menggunakan papan tensor, Anda perlu memberi anotasi pada grafik Anda, dengan variabel yang ingin Anda periksa, yaitu: nilai kerugian. Maka Anda perlu membuat semua ringkasan, menggunakan fungsi tf.merge_all_summaries().

Secara opsional, Anda juga dapat menggunakan fungsi "tf.name_scope" untuk mengelompokkan node pada grafik.

Setelah semua variabel dianotasi dan Anda mengonfigurasi ringkasan, Anda dapat membuka konsol dan memanggil:

```
tensorboard --logdir=/home/leo/test
```

Mempertimbangkan contoh sebelumnya di sini adalah perubahan yang diperlukan untuk menambahkan informasi ke papan tensor.

1) Pertama, kami membubuhi keterangan informasi pada grafik bahwa Anda tertarik untuk memeriksa tahap pembangunan. Kemudian panggil merge_all_summaries(). Dalam kasus kami, kami mencatat kerugian (skalar) dan W, b (histogram)

```
# Create our linear regression model

# Variables resides internally inside the graph memory
```

```
#tf.name_scope organize things on the tensorboard graphview
with tf.name_scope("LinearReg") as scope:
    W = tf.Variable(tf.random_uniform([1], -1.0, 1.0), name="Weights")
    b = tf.Variable(tf.zeros([1.0]), name="Bias")
    y = W * x_data + b

# Define a loss function that take into account the distance between
# the prediction and our dataset
with tf.name_scope("LossFunc") as scope:
    loss = tf.reduce_mean(tf.square(y-y_data))

# Create an optimizer for our loss function
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

#### Tensorboard stuff
# Annotate loss, weights and bias (Needed for tensorboard)
loss_summary = tf.scalar_summary("loss", loss)
w_h = tf.histogram_summary("W", W)
b_h = tf.histogram_summary("b", b)

# Merge all the summaries
merged_op = tf.merge_all_summaries()
```

2) Selama pembuatan sesi, kita perlu menambahkan panggilan ke "tf.train.SummaryWriter" untuk membuat penulis. Anda harus melewati direktori tempat tensorflow akan menyimpan ringkasan.

```
# Initialize all graph variables
init = tf.initialize_all_variables()
```

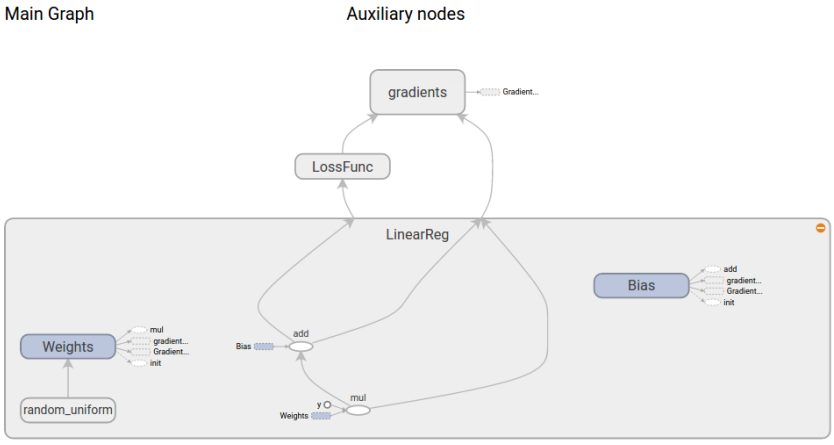
```
# Create a session and initialize the graph variables (Will acutally run now...)
session = tf.Session()
session.run(init)

# Writer for tensorboard (Directory)
writer_tensorboard = tf.train.SummaryWriter('/home/leo/test', session.graph_def)
```

3) Kemudian ketika kita mengeksekusi grafik kita, misalnya saat pelatihan kita dapat meminta tensorflow untuk membuat ringkasan. Tentu saja memanggil ini setiap saat akan berdampak pada kinerja. Untuk mengelola ini, Anda dapat menyebutnya di akhir setiap zaman.

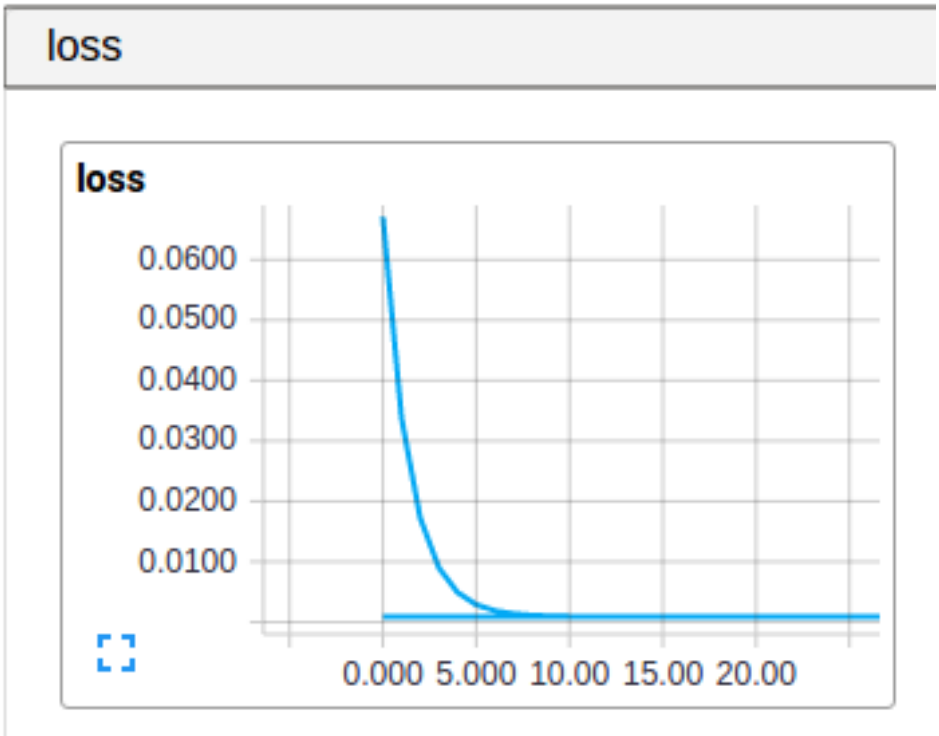
```
for step in xrange(1000):
    # Optimize one step
    session.run(train)

    # Add summary (Everytime could be to much....)
    result_summary = session.run(merged_op)
    writer_tensorboard.add_summary(result_summary, step)
```

Hasil pada papan tensor

Di sini kita dapat melihat model regresi linier kita sebagai grafik komputasi.



Di bawah ini kita dapat melihat bagaimana kerugian berkembang pada setiap iterasi.



Terkadang ipython menyimpan versi grafik Anda yang menimbulkan masalah saat menggunakan tensorboard, salah satu opsinya adalah memulai ulang kernel, jika Anda mengalami masalah.

Menggunakan GPU

Tensorflow juga memungkinkan Anda menggunakan GPU untuk mengeksekusi grafik atau bagian tertentu dari grafik Anda.

Pada sistem pembelajaran mesin umum Anda akan memiliki satu CPU multi-core, dengan satu atau lebih GPU, tensorflow mewakilinya sebagai berikut

- "/cpu:0": CPU Multicore
- "/gpu0": GPU Pertama
- "/ gpu1": GPU kedua

Sayangnya tensorflow tidak memiliki fungsi resmi untuk mencantumkan perangkat yang tersedia di sistem Anda, tetapi ada cara tidak resmi.

```
from tensorflow.python.client import device_lib

def get_devices_available():
    local_device_protos = device_lib.list_local_devices()
    return [x.name for x in local_device_protos]

print(get_devices_available())
```

```
['/cpu:0', '/gpu:0', '/gpu:1']
```

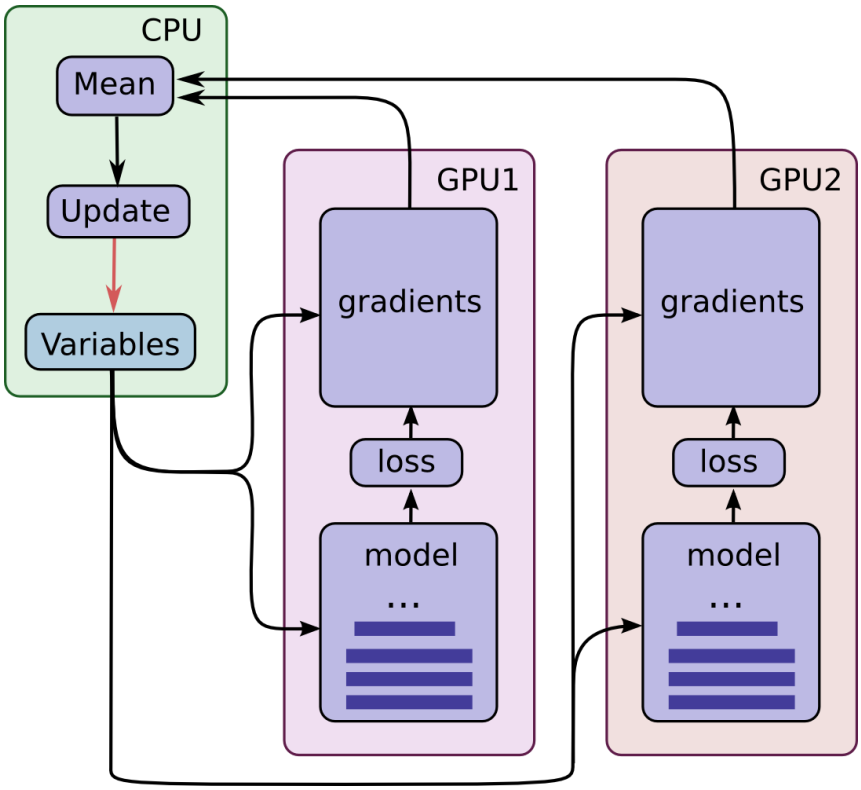
Perbaiki grafik ke perangkat

Gunakan pernyataan "with tf.device('/gpu:0')" pada python untuk mengunci semua node pada blok grafik ini ke gpu tertentu.

```
import tensorflow as tf


# Creates a graph.
with tf.device('/gpu:0'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
    c = tf.matmul(a, b)


# Creates a session with log_device_placement set to True, this will dump
# on the log how tensorflow is mapprint the operations on devices
sess = tf.Session(config=tf.ConfigProto(log_device_placement=True))
# Runs the op.
print(sess.run(c))
sess.close()
[[ 22.  28.]
 [ 49.  64.]]
```

Beberapa GPU dan pelatihan

Sekarang kami akan menjelaskan bagaimana pelatihan adalah satu di beberapa sistem GPU.



Pada dasarnya langkah-langkah untuk beberapa pelatihan gpu adalah ini:

1. Pisahkan data pelatihan Anda secara berkelompok seperti biasa
2. Buat salinan model di setiap gpu
3. Mendistribusikan batch yang berbeda untuk setiap gpu
4. Setiap gpu akan meneruskan batch dan menghitung gradiennya
5. Setiap gpu akan mengirimkan gradien ke cpu
6. CPU akan menghitung rata-rata setiap gradien, dan melakukan penurunan gradien. Parameter model diperbarui dengan rata-rata gradien di semua replika model.
7. CPU akan mendistribusikan model baru ke semua GPU
8. loop proses lagi sampai semua pelatihan selesai

PyTorch

Perkenalan

PyTorch adalah perpustakaan pembelajaran mendalam lainnya yang sebenarnya merupakan fork dari Chainer (Perpustakaan pembelajaran mendalam sepenuhnya dengan python) dengan kemampuan obor. Pada dasarnya ini adalah solusi facebook untuk menggabungkan obor dengan python.

Beberapa keuntungan

- Mudah untuk Debug dan pahami kodenya
- Memiliki jenis layer sebanyak Torch (Unpool, CONV 1,2,3D, LSTM, Grus)
- Banyak fungsi kerugian
- Dapat dianggap sebagai ekstensi Numpy untuk GPU
- Lebih cepat dari pustaka "define-by-run" lainnya, seperti chainer dan dynet
- Izinkan untuk membangun jaringan yang strukturnya bergantung pada perhitungan itu sendiri (Berguna untuk pembelajaran penguatan)

Komponen PyTorch

Kemasan	Keterangan
obor	Numpy like library dengan dukungan GPU
torch.autograd	Berikan dukungan diferensiasi untuk semua operasi obor
obor.nn	Perpustakaan jaringan saraf terintegrasi dengan autograd
torch.optim	Optimasi untuk torch.nn (ADAM, SGD, RMSPROP, dll...)
torch.multiprocessing	Berbagi memori antara tensor
torch.utils	DataLoader, Pelatihan, dan fungsi utilitas lainnya
torch.legacy	Kode lama diporting dari Torch

Apa bedanya dengan Tensorflow/Theano

Perbedaan utama dari Tensorflow adalah bahwa metodologi PyTorch dianggap "define-by-run" sementara Tensorflow dianggap "defined-and-run", jadi pada PyTorch Anda misalnya dapat mengubah model Anda pada waktu proses, debug dengan mudah dengan debugger python apa pun, sementara tensorflow selalu memiliki definisi/bangun grafik. Anda dapat mempertimbangkan tensorflow sebagai alat produksi yang lebih banyak sementara PyTorch lebih merupakan alat penelitian.

Dasar Pytorch:

Di sini kita akan melihat cara membuat tensor, dan melakukan beberapa manipulasi:

```
import torch
import numpy as np

# Create a tensor on torch
a = torch.rand(3, 3)

# Create a matrix on numpy and conver to PyTorch
b_npy = np.array([[1,2,3],[4,5,6],[7,8,9]])
# Convert from numpy to torch
b = torch.from_numpy(b_npy)

print(a)
print(b)

# Get a specific element
print(b[1,1])

# Get a range of elements
print(b[1:None,1:None])

# Set elements on array
a[1:None,1:None] = 0
print(a)
```

```
 0.4001  0.3895  0.1464
 0.2150  0.4885  0.9369
 0.7743  0.3803  0.5384
[torch.FloatTensor of size 3x3]


 1  2  3
 4  5  6
 7  8  9
[torch.LongTensor of size 3x3]

5

 5  6
 8  9
[torch.LongTensor of size 2x2]


 0.4001  0.3895  0.1464
 0.2150  0.0000  0.0000
 0.7743  0.0000  0.0000
[torch.FloatTensor of size 3x3]
```

Buat tensor yang diisi dengan beberapa nilai

```
import torch


a = torch.ones(2,3)

b = torch.zeros(3,2)

print(a)

print(b)
```

```
 1  1  1
 1  1  1
[torch.FloatTensor of size 2x3]


 0  0
 0  0
 0  0
[torch.FloatTensor of size 3x2]
```

Sekarang kita akan melakukan perhitungan pada GPU

```
import torch
import numpy as np

# Define tensors on the GPU
a = torch.rand(2, 3).cuda()
b = torch.rand(2, 3).cuda()

# Define some operation (will execute on the GPU)
c = (a + b) * 2

# Print "c" contents and shape(size)
print(c)
print(c.size())
```

```
 2.8274  2.9363  1.7493
 0.4575  2.1084  3.1521
[torch.cuda.FloatTensor of size 2x3 (GPU 0)]

torch.Size([2, 3])
```

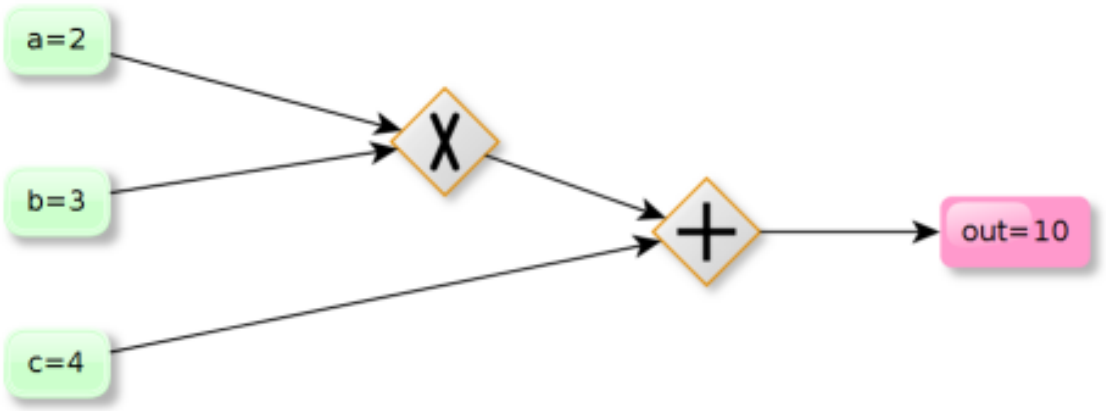
Autograd dan variabel

Autograd di PyTorch adalah komponen yang bertanggung jawab untuk melakukan perambatan balik, karena pada Tensorflow Anda hanya perlu menentukan perambatan maju. Autograd PyTorch sangat mirip dengan TensorFlow: di kedua framework kami mendefinisikan grafik komputasi, dan menggunakan diferensiasi otomatis untuk menghitung gradien.

Kita hanya perlu membungkus tensor dengan objek Variabel, Variabel mewakili simpul dalam grafik komputasi. Mereka tidak seperti placeholder tensorflow, pada PyTorch

Anda menempatkan nilai langsung pada model. Sekali lagi untuk memasukkan tensor pada grafik, bungkus dengan variabel.

Perhatikan grafik sederhana berikut ini:




```
import torch
from torch.autograd import Variable

# Define scalar a=2, b=3
a = Variable(torch.ones(1, 1) * 2, requires_grad=True)
b = Variable(torch.ones(1, 1) * 3, requires_grad=True)
c = Variable(torch.ones(1, 1) * 4, requires_grad=True)

# Define the function "out" having 2 parameters a,b
out = (a*b)+c
#c = torch.mul(a,b)+c
print('Value out:',out)

# Do the backpropagation
out.backward()

# Get dout/da (Derivative of out w.r.t to a)
```

```
print('Derivative of out w.r.t to a:',a.grad)

print('Derivative of out w.r.t to b:',b.grad)

print('Derivative of out w.r.t to c:',c.grad)
```

```
Value out: Variable containing:
 10
[torch.FloatTensor of size 1x1]

Derivative of out w.r.t to a: Variable containing:
 3
[torch.FloatTensor of size 1x1]

Derivative of out w.r.t to b: Variable containing:
 2
[torch.FloatTensor of size 1x1]

Derivative of out w.r.t to c: Variable containing:
 1
[torch.FloatTensor of size 1x1]
```

Contoh lengkap

Di sini kami menggabungkan konsep dan menunjukkan cara melatih kumpulan data MNIST menggunakan CNN

```
# Import libraries

import torch

from torch.autograd import Variable

import torchvision.datasets as dsets

import torchvision.transforms as transforms

import torch.nn as nn

import torch.nn.functional as F


# Hyper Parameters

num_epochs = 5

batch_size = 50

learning_rate = 0.001
```

```
# MNIST Dataset
train_dataset = dsets.MNIST(root='../data/',
                            train=True,
                            transform=transforms.ToTensor(),
                            download=True)

test_dataset = dsets.MNIST(root='../data/',
                           train=False,
                           transform=transforms.ToTensor())


# Data Loader (Input Pipeline)
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)


# CNN Model (2 conv layer) nn.Module is the base class to all neural networks
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, padding=2),
```

```
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(2))
        self.fc = nn.Linear(7*7*32, 10)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out

cnn = CNN()
cnn.cuda()
print(cnn)

# Loss and Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(cnn.parameters(), lr=learning_rate)

# Train the Model
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
```

```
        images = Variable(images)
        labels = Variable(labels)

        images, labels = images.cuda(), labels.cuda()

        # Forward + Backward + Optimize
        optimizer.zero_grad()
        outputs = cnn(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        if (i+1) % 500 == 0:
            print ('Epoch [%d/%d], Iter [%d/%d] Loss: %.4f'
                   %(epoch+1, num_epochs, i+1, len(train_dataset)//batch_size, loss.data[0]))


# Test the Model
cnn.eval()  # Change model to 'eval' mode (BN uses moving mean/var).
correct = 0
total = 0
for images, labels in test_loader:
    images = Variable(images)
    images, labels = images.cuda(), labels.cuda()
    outputs = cnn(images)
    _, predicted = torch.max(outputs.data, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum()
```

```
print('Test Accuracy of the model on the 10000 test images: %d %%' % (100 * correct
/ total))


# Save the Trained Model
torch.save(cnn.state_dict(), 'cnn.pkl')
```