Menjadi Powerful Dengan Tensor(flow)

Part 1

Aditya Firman Ihsan • Tensorflow Developer Certified
KK Data Science Telkom University

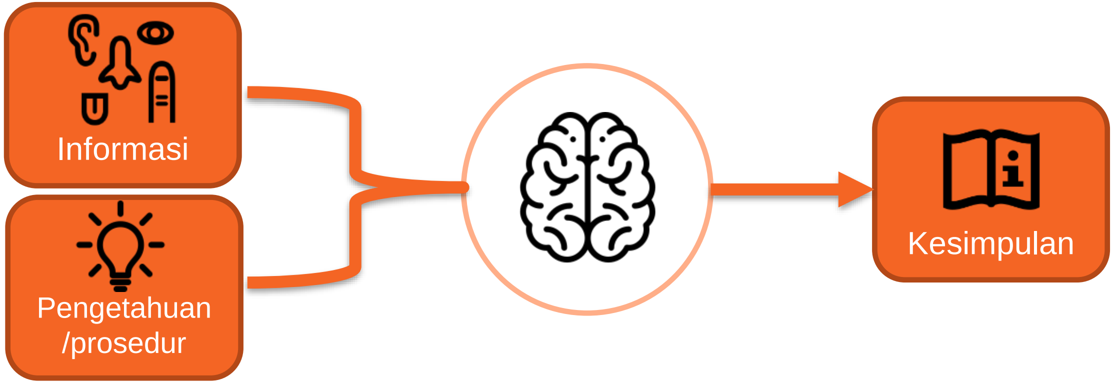
Telkom University

“Artificial Intelligence, deep learning, machine learning – whatever you’re doing, if you don’t understand it, learn it. Because otherwise you’re going to be a dinosaur within 3 years”

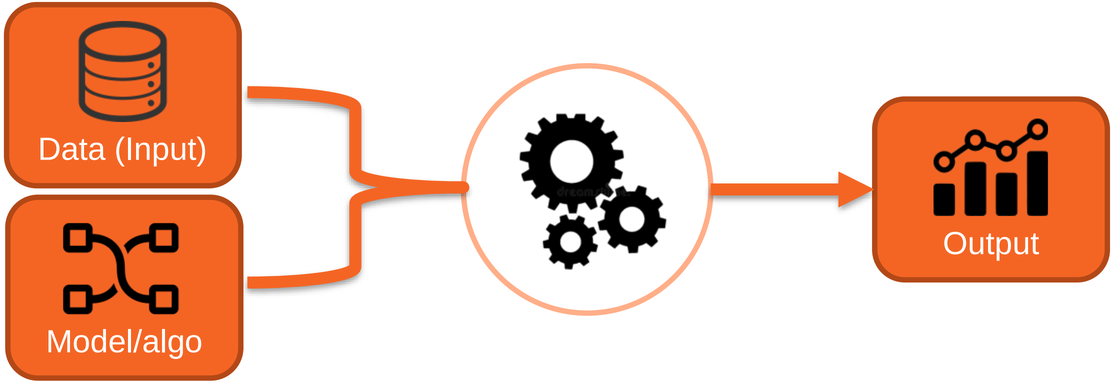
Mark Cuban

Mesin yang belajar?

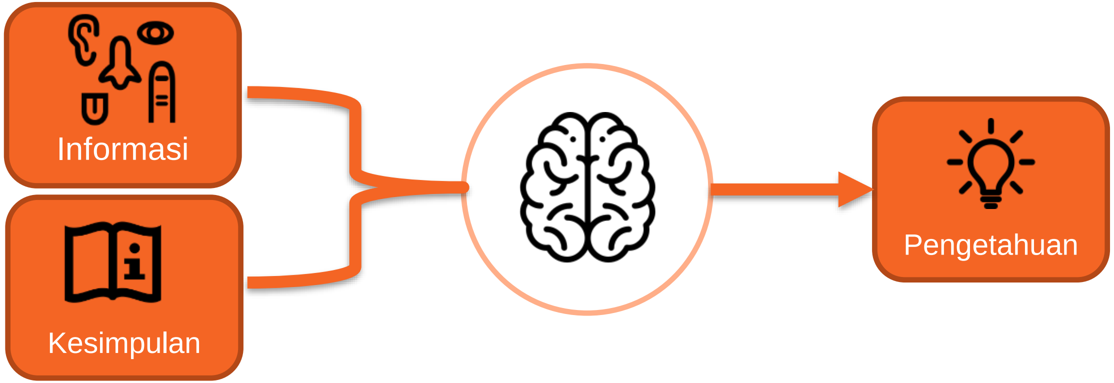
Bagaimana kita belajar?
Seperti ini?



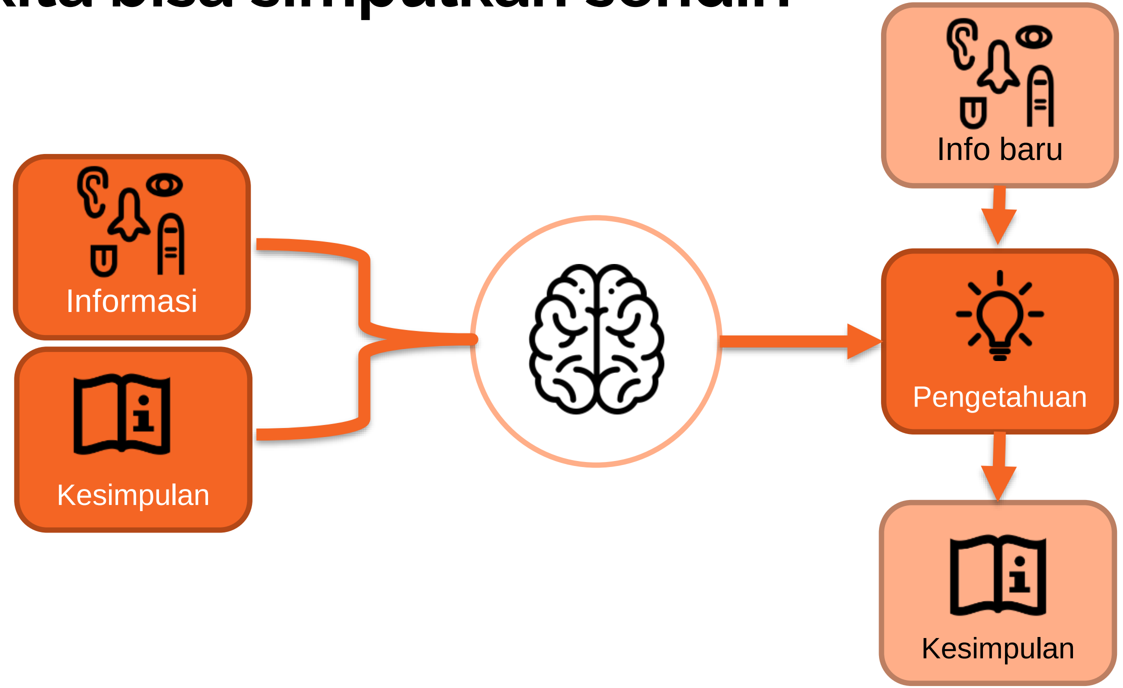
Bedanya dengan mesin?



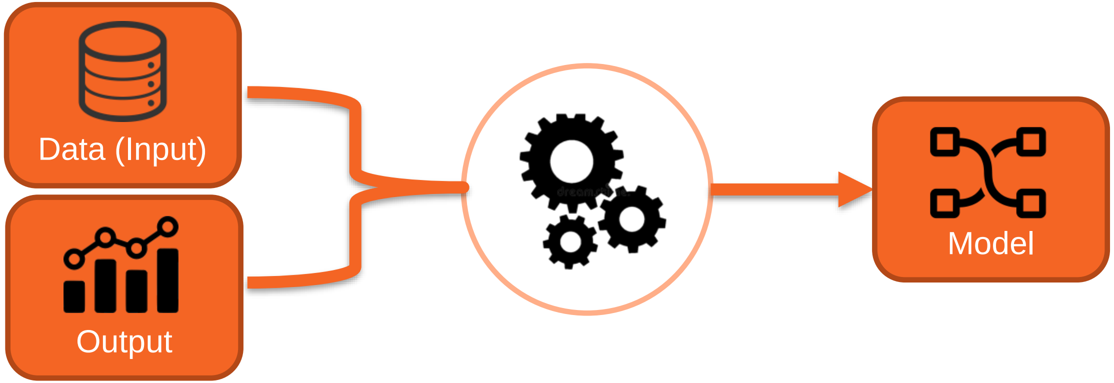
Ataukah seperti ini kita belajar?



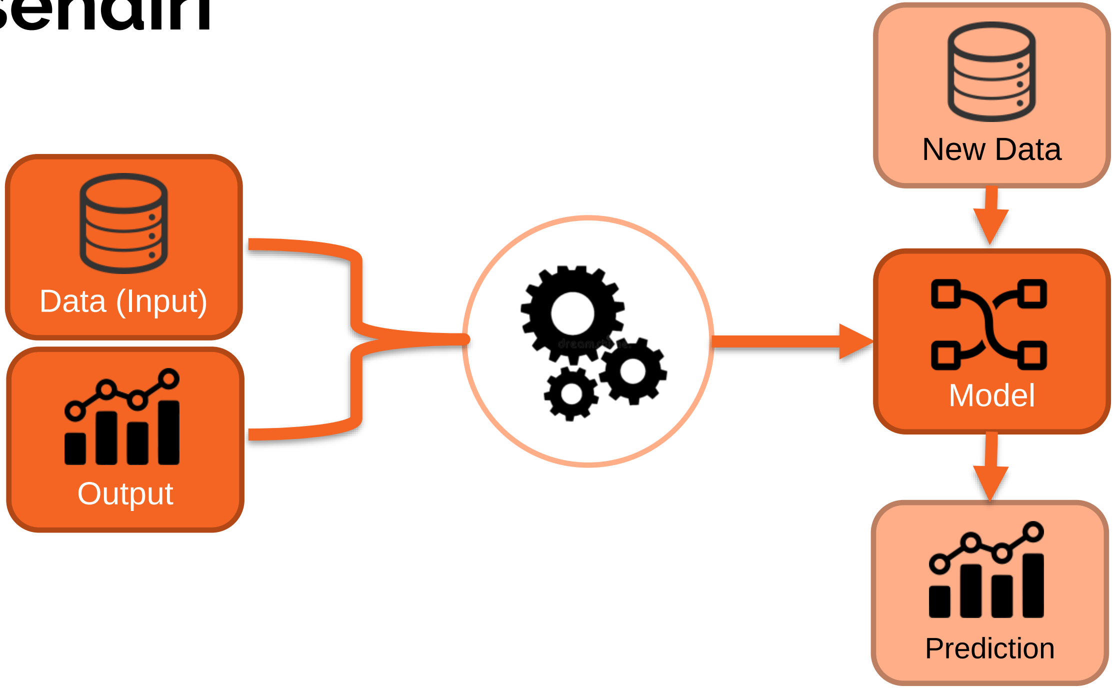
Yap, karena ketika diberi informasi baru, kita bisa simpulkan sendiri



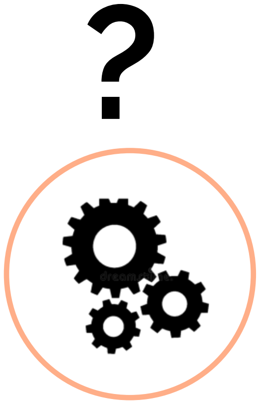
Demikian juga bila kita ingin buat mesin “belajar”



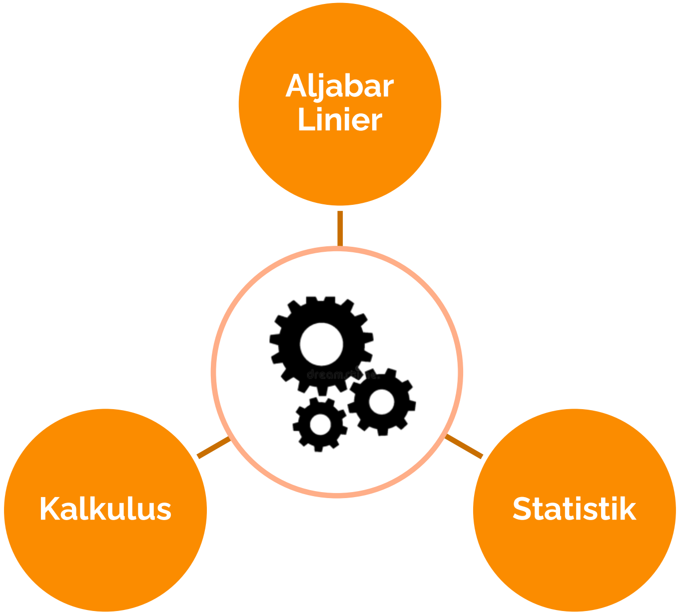
Sehingga, mesin itu bisa simpulkan sendiri



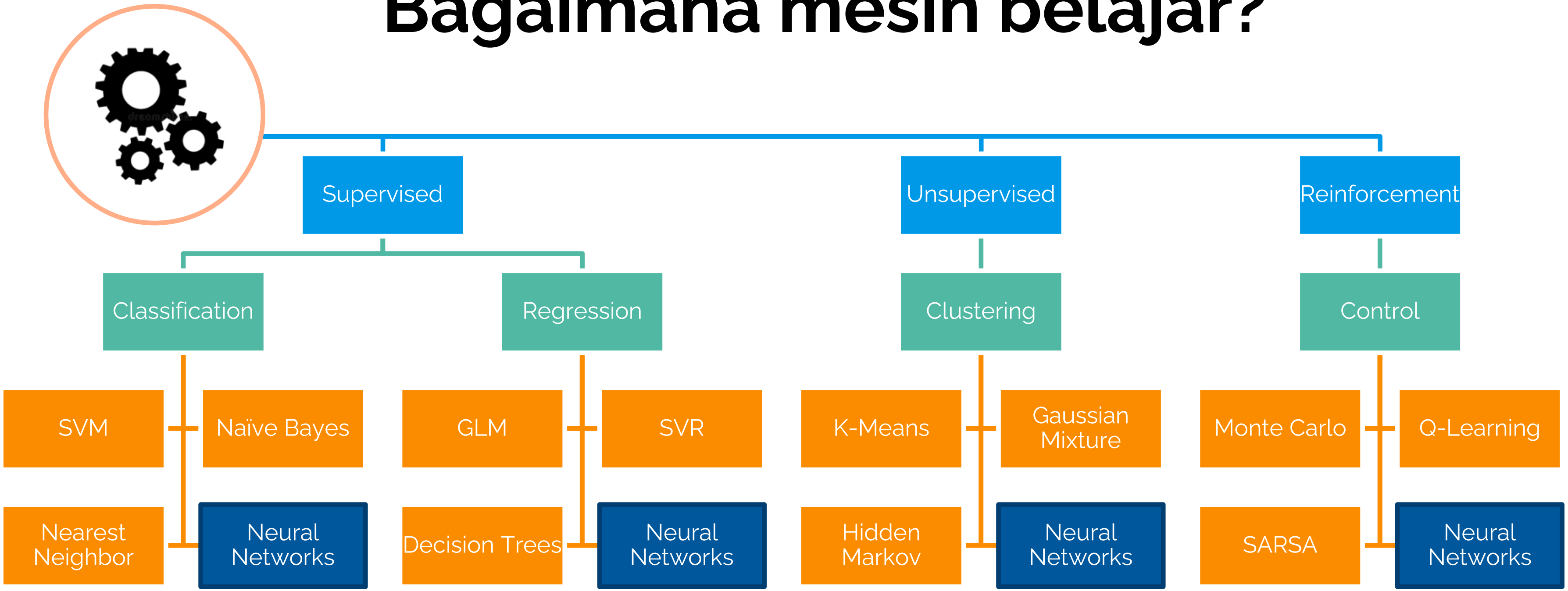
Bagaimana mesin belajar?



Bagaimana mesin belajar?

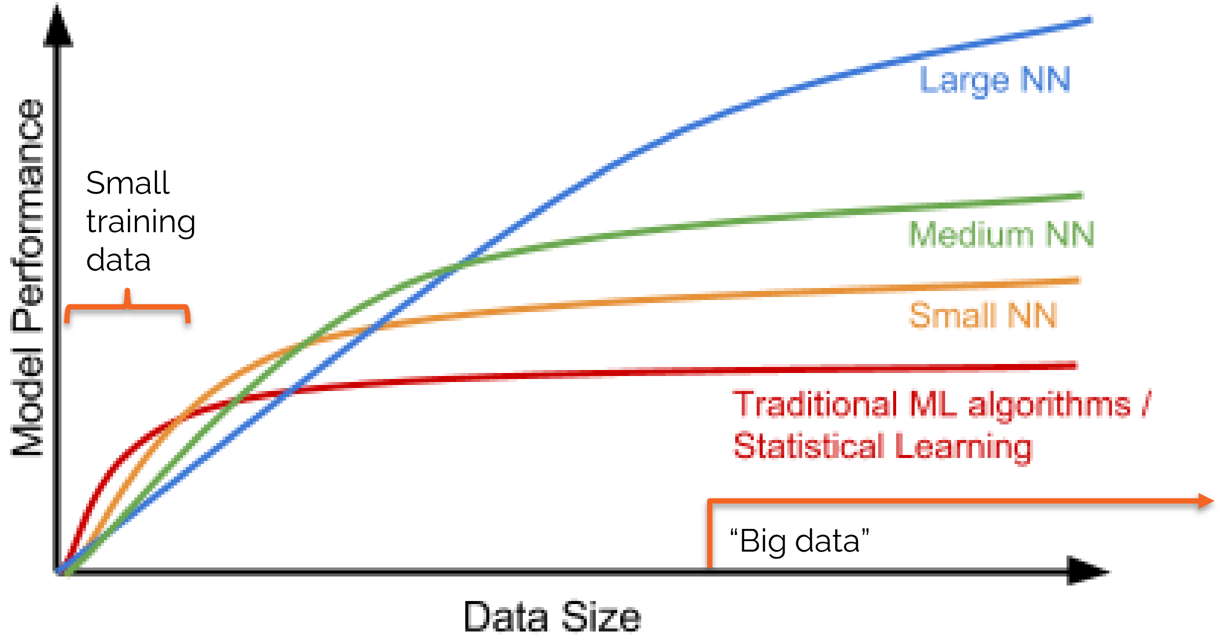


Bagaimana mesin belajar?

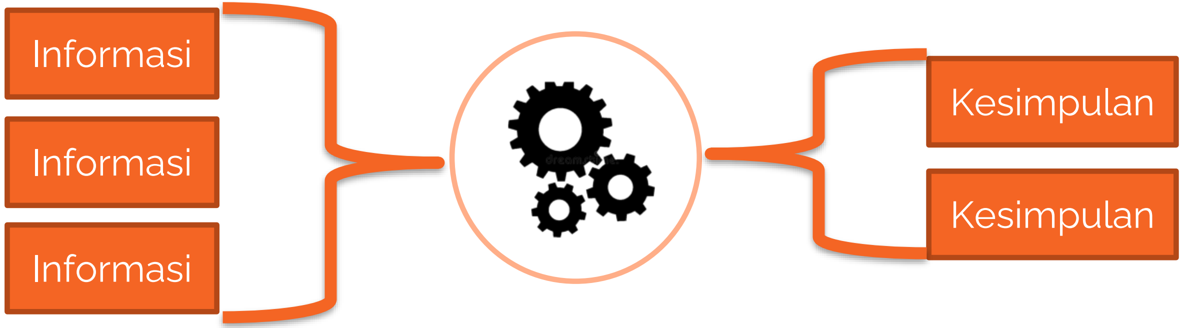


Apapun paradigma ML-nya, neural network solusinya

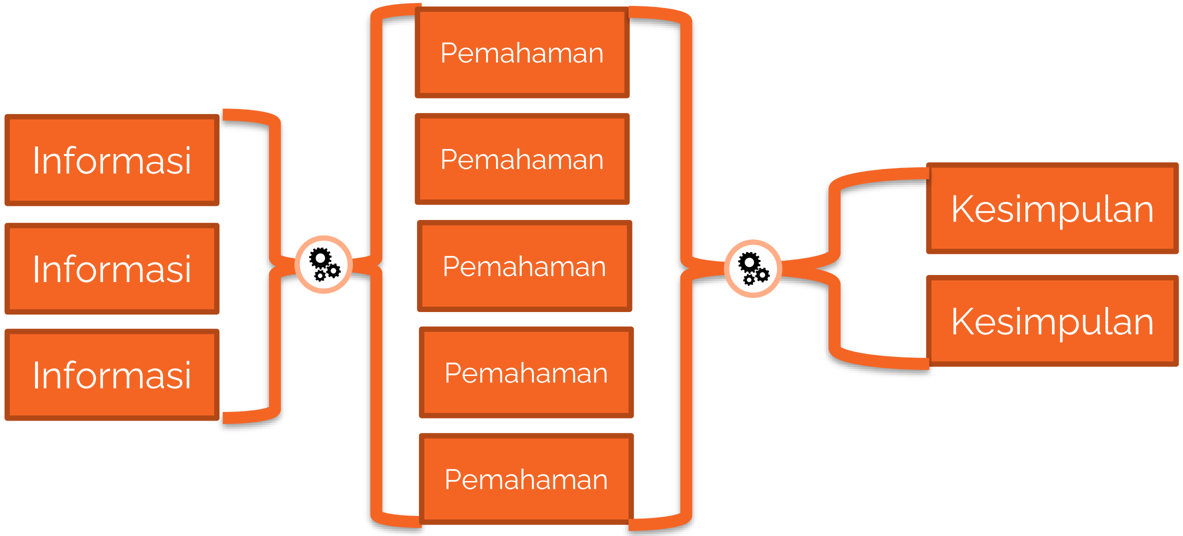
Kenapa Neural Network (NN)?



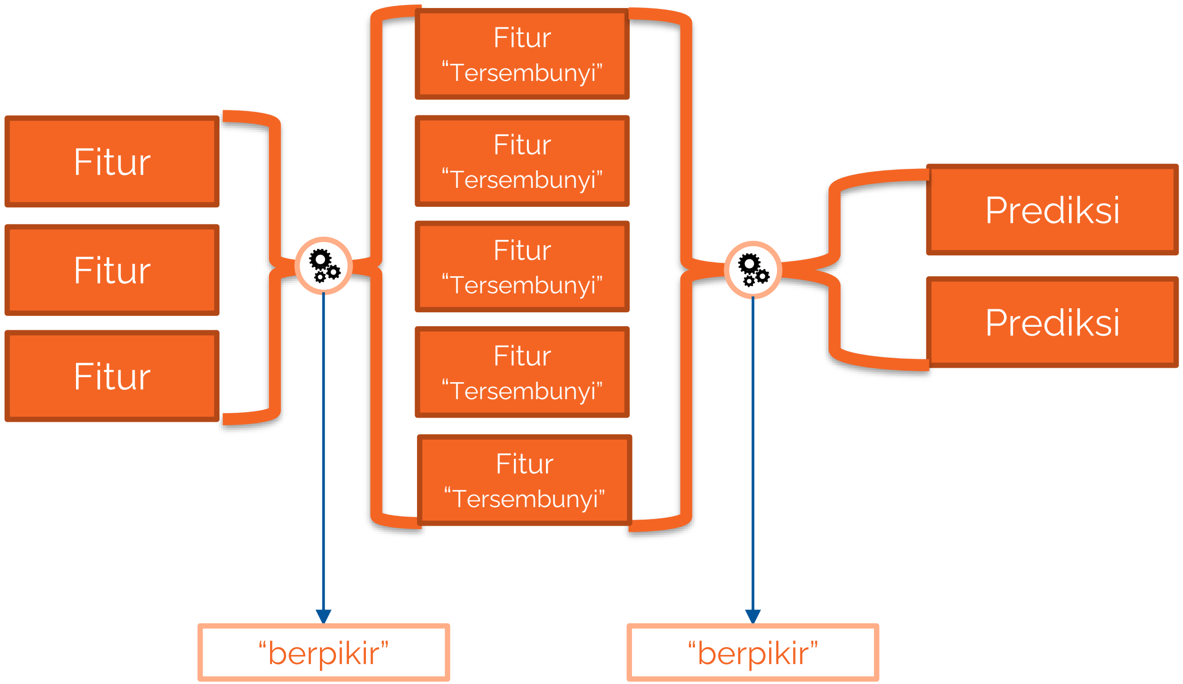
Apa yang dilakukan NN?



Apa yang dilakukan NN?

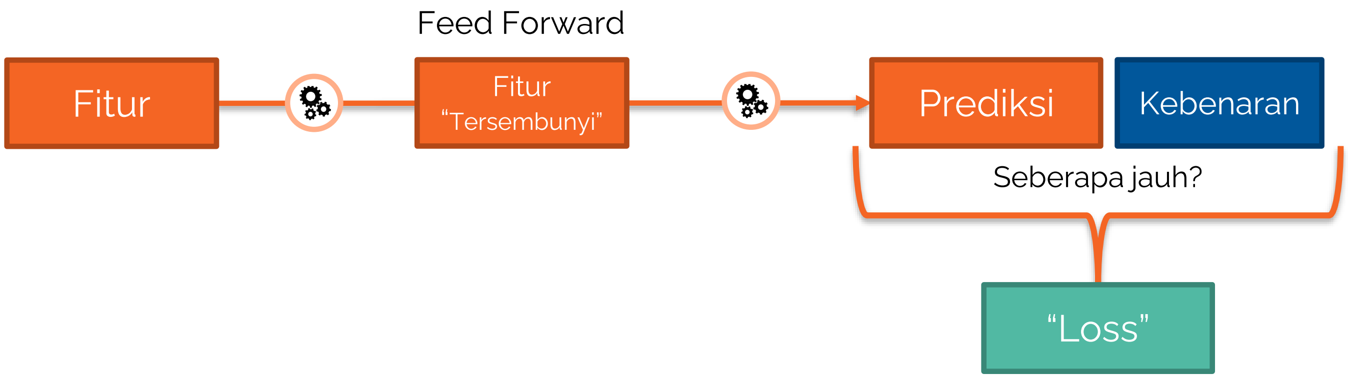


Apa yang dilakukan NN?

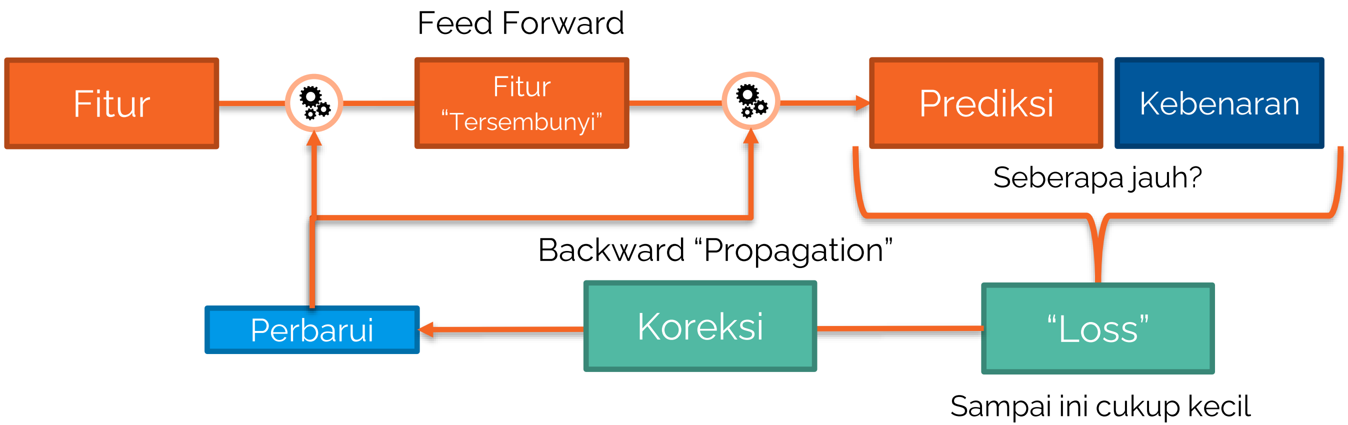


Apa yang dilakukan NN?

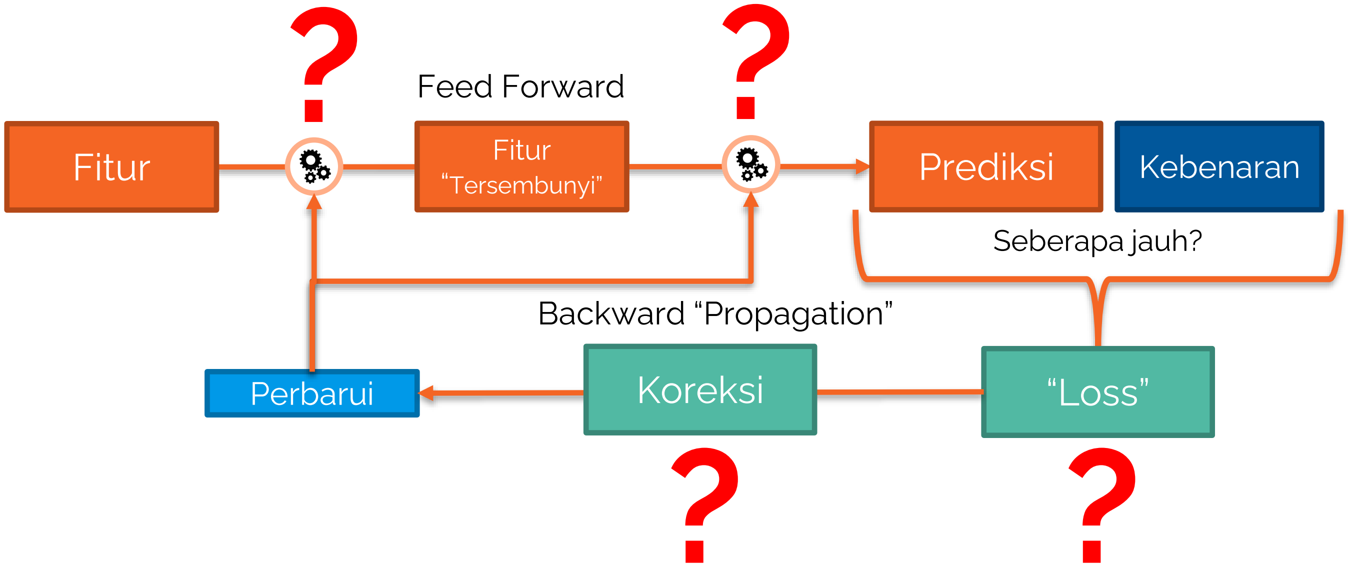
Apa yang dilakukan NN?



Apa yang dilakukan NN?

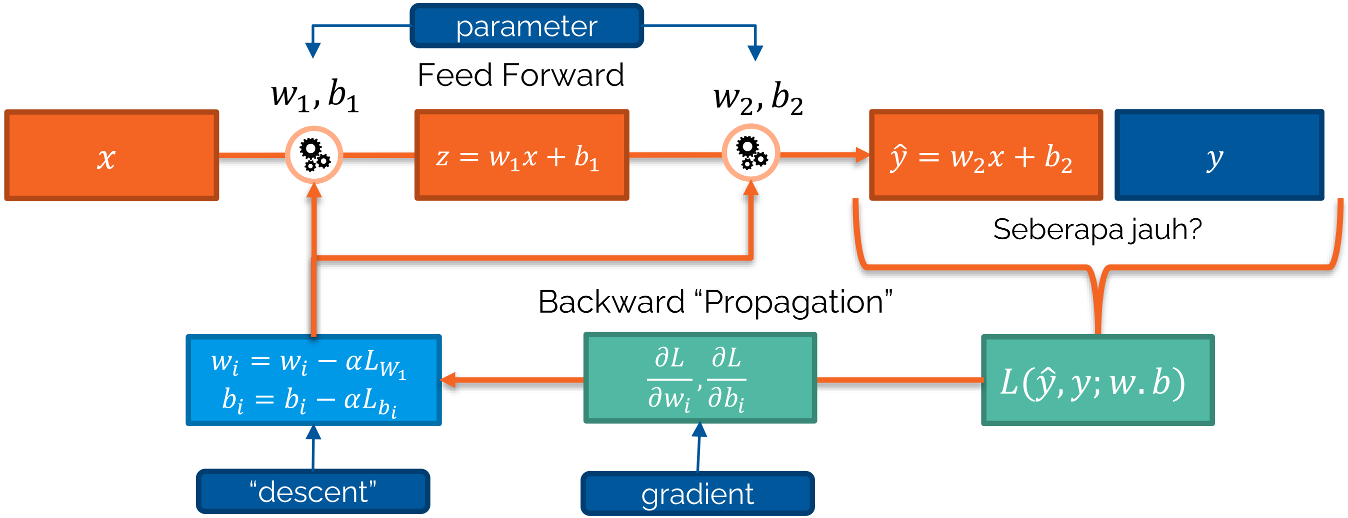


Apa sebenarnya benda-benda ini?

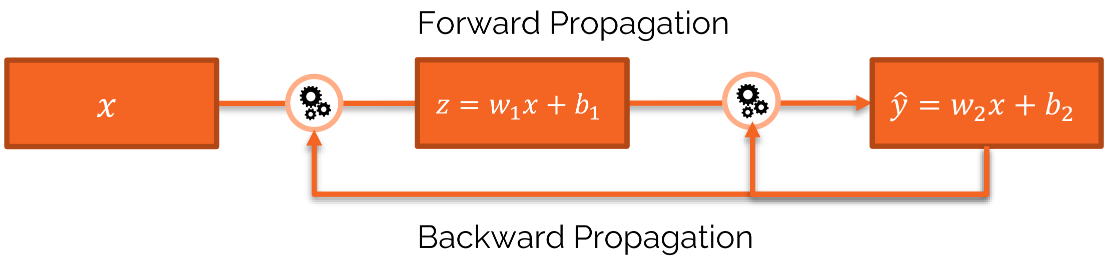


Semuanya hanya persamaan matematika

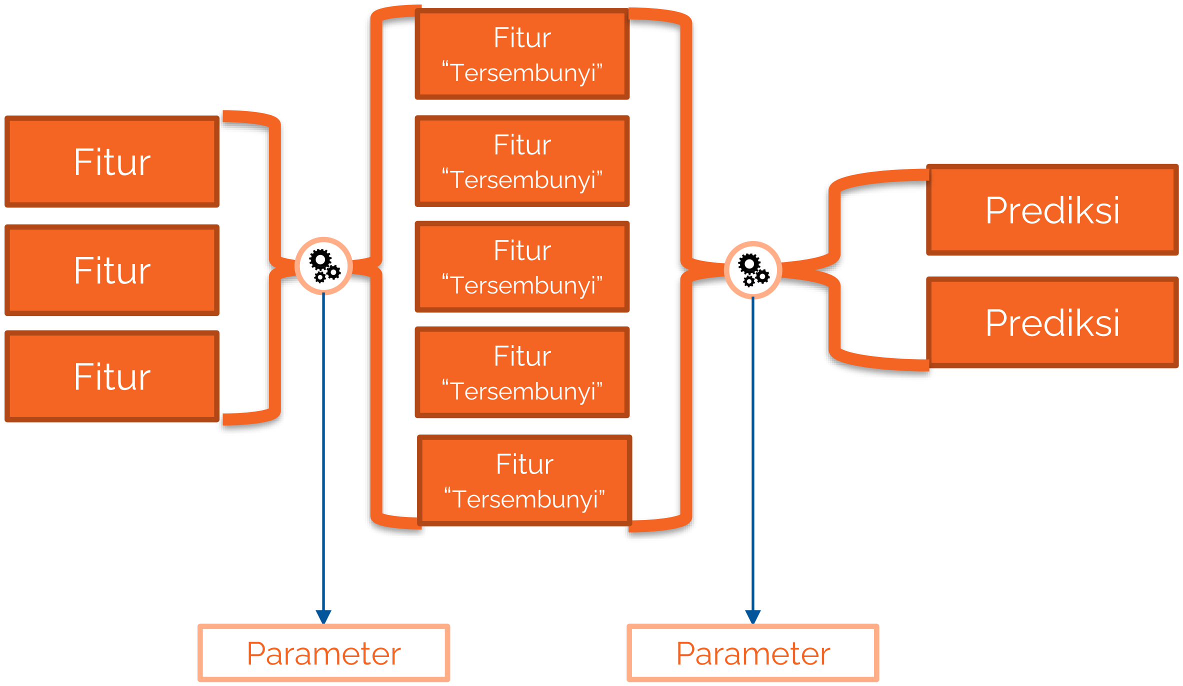
Apa yang dilakukan NN?



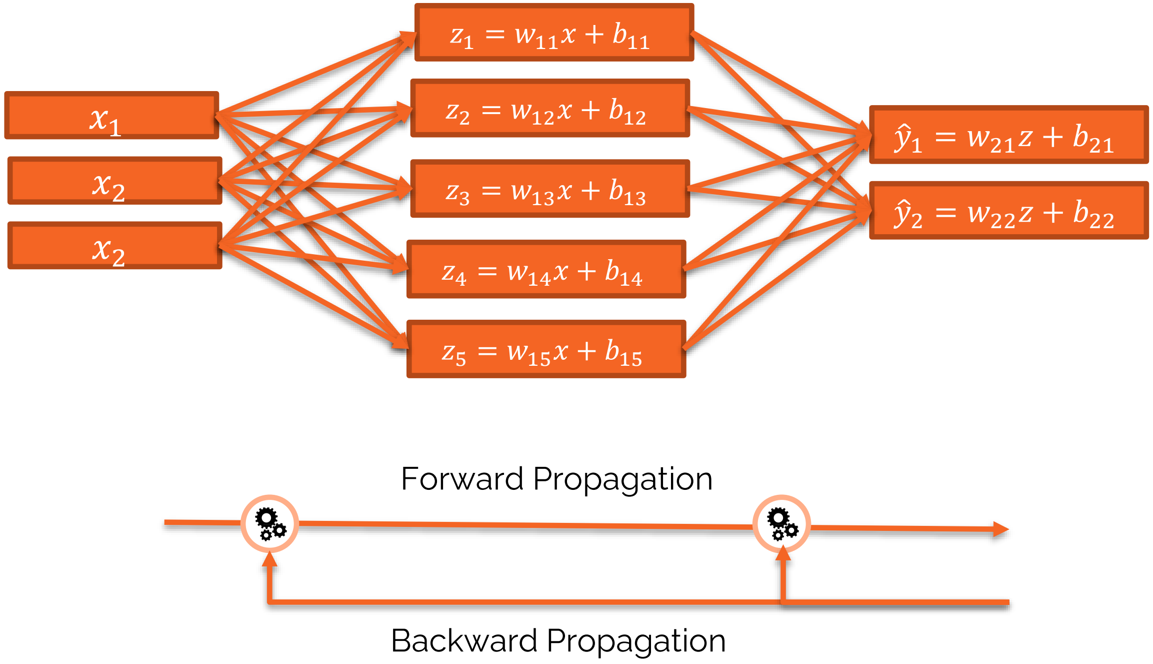
Apa yang dilakukan NN?



Apa yang dilakukan NN?



Apa yang dilakukan NN?

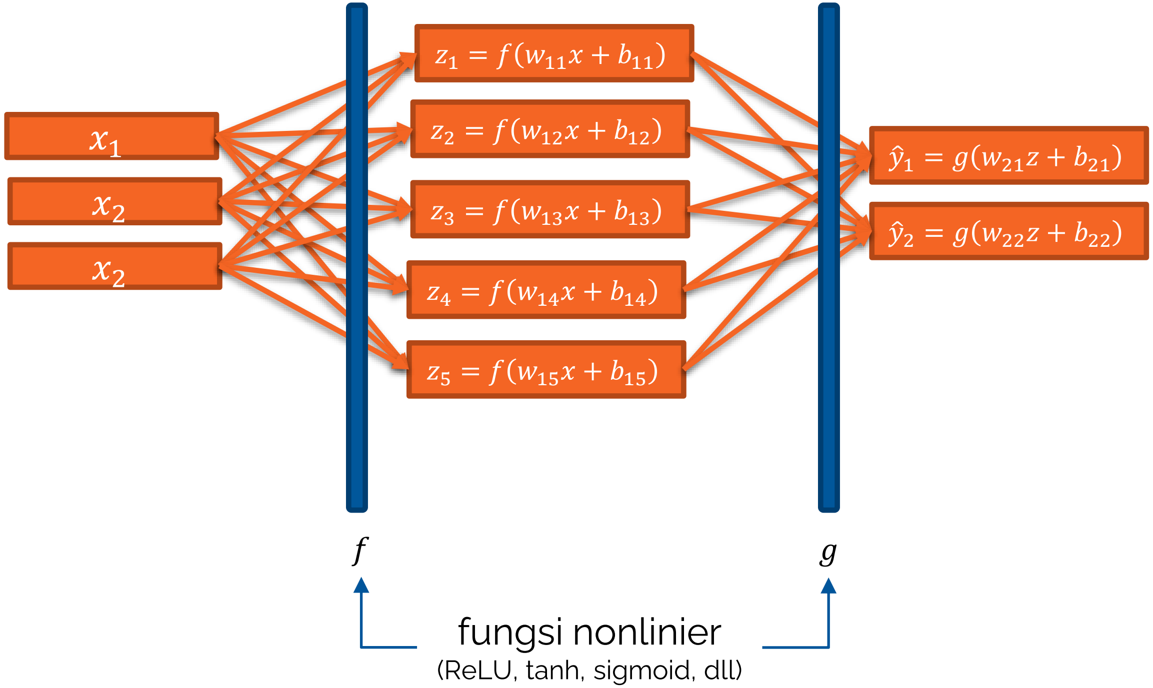


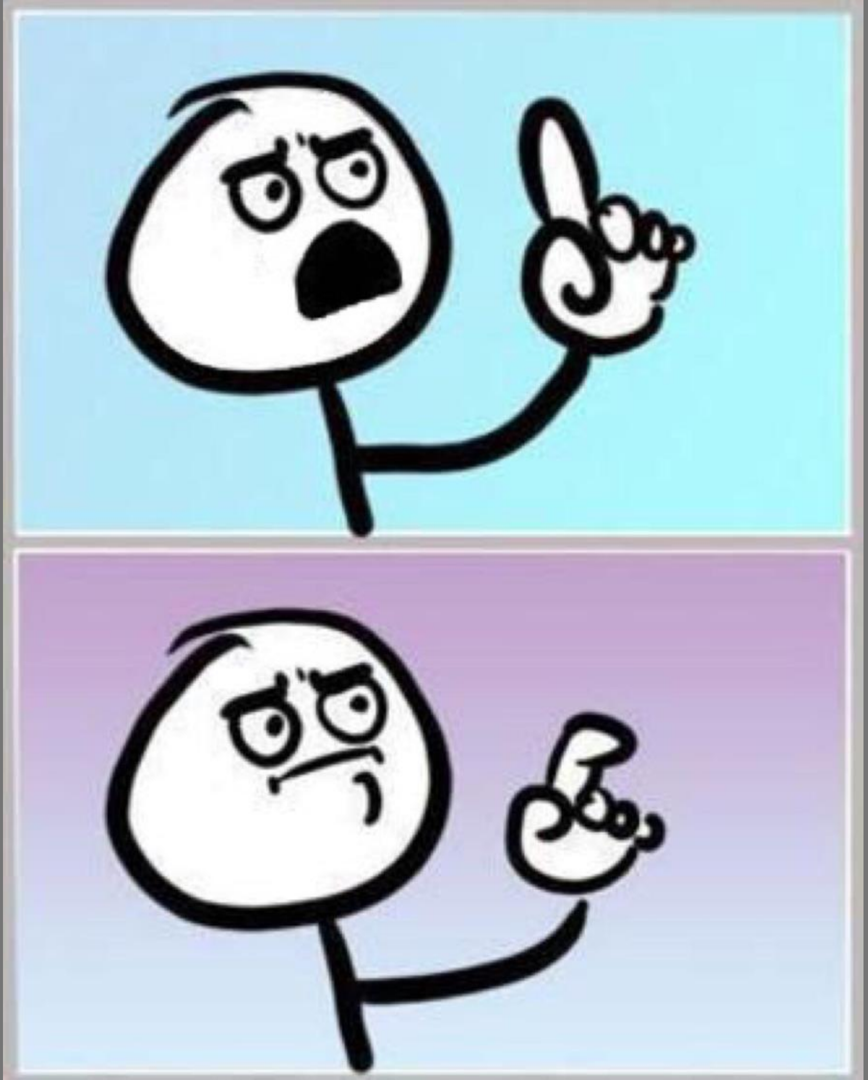
Tapi, tumpukan linier akan tetap linier

$$z = w_1 x + b_1$$

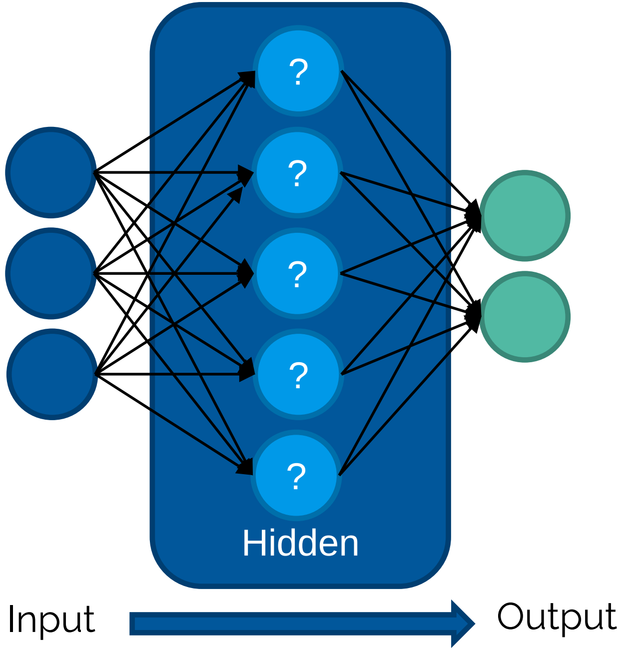
$$\hat{y} = w_2 z + b_2 = w_2(w_1 x + b_1) + b_2$$
$$= (w_2 w_1)x + (w_2 b_1 + b_2)$$

Apa yang dilakukan NN?

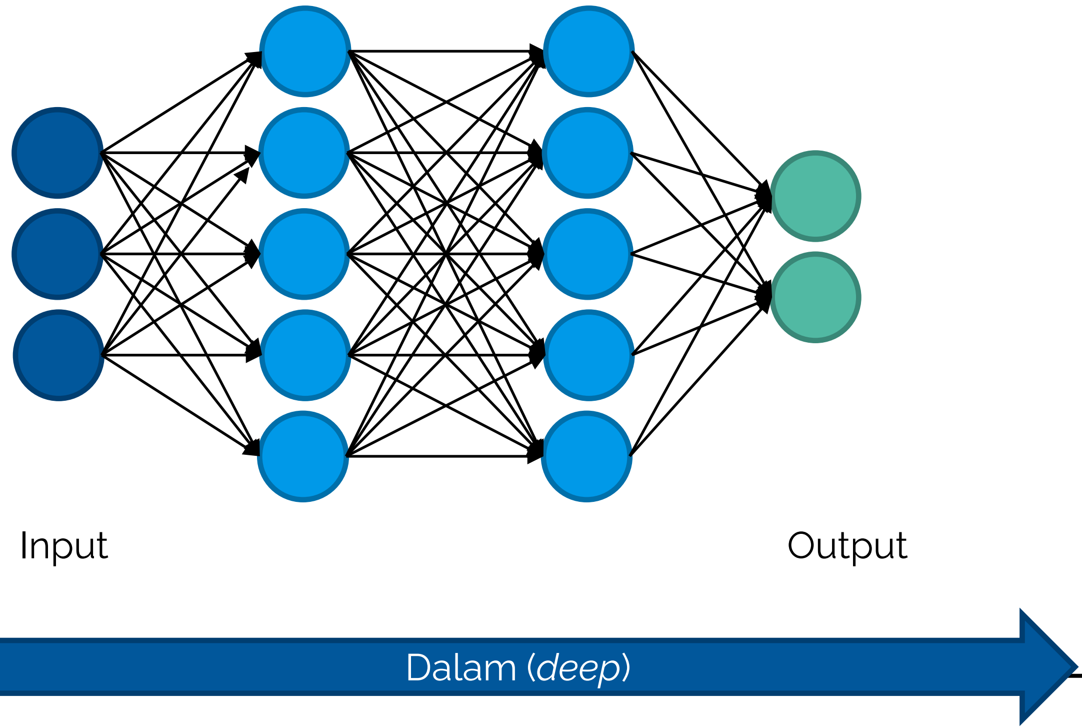




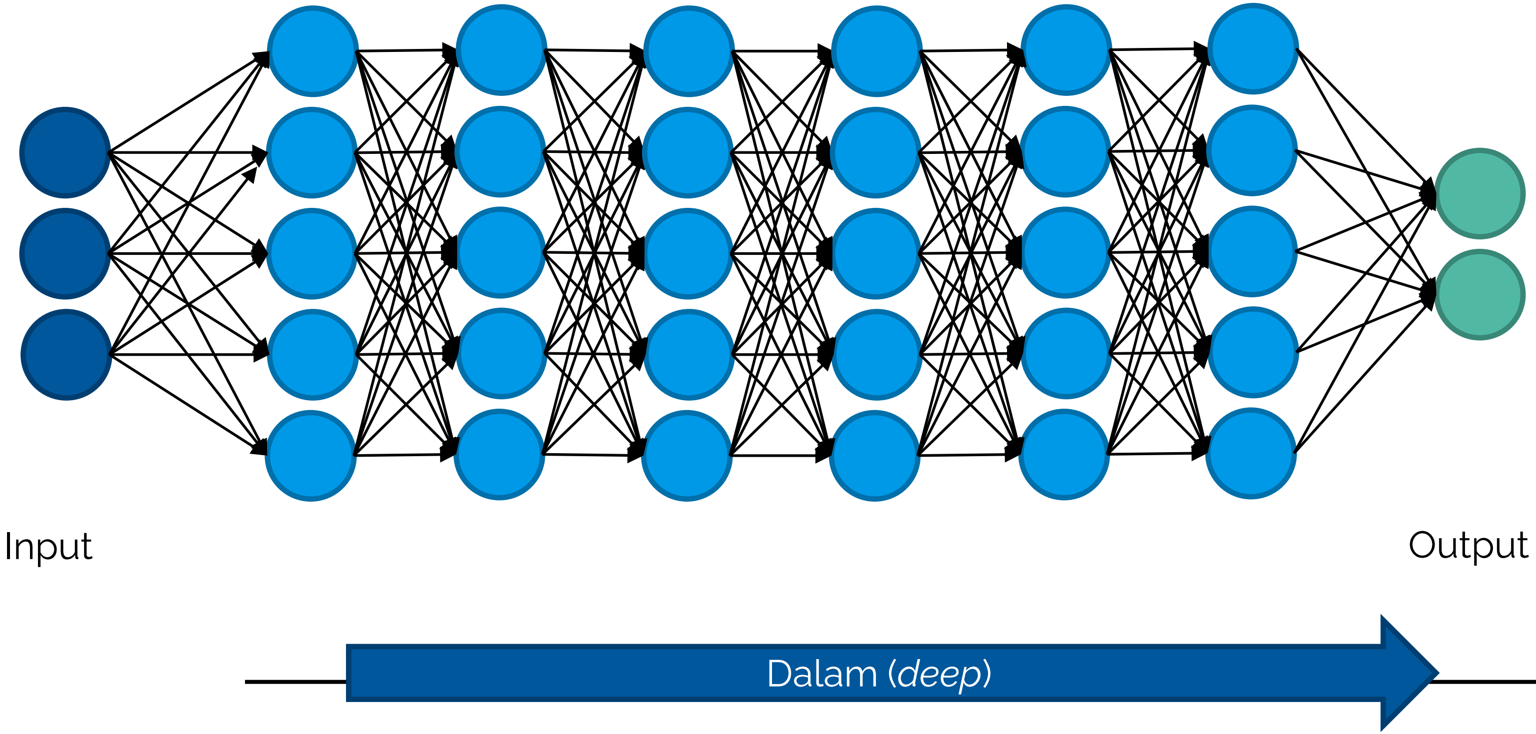
Ah, terlalu rumit, NN adalah berikut aja



Bagaimana kalau informasi abstraknya dipelajari bertahap?



Seberapa dalam mesin harus belajar?



Sedalam-dalam model NN, masih mudah diprogram pakai *library* standar

Kira-kira seperti ini lah (hanya pakai numpy)

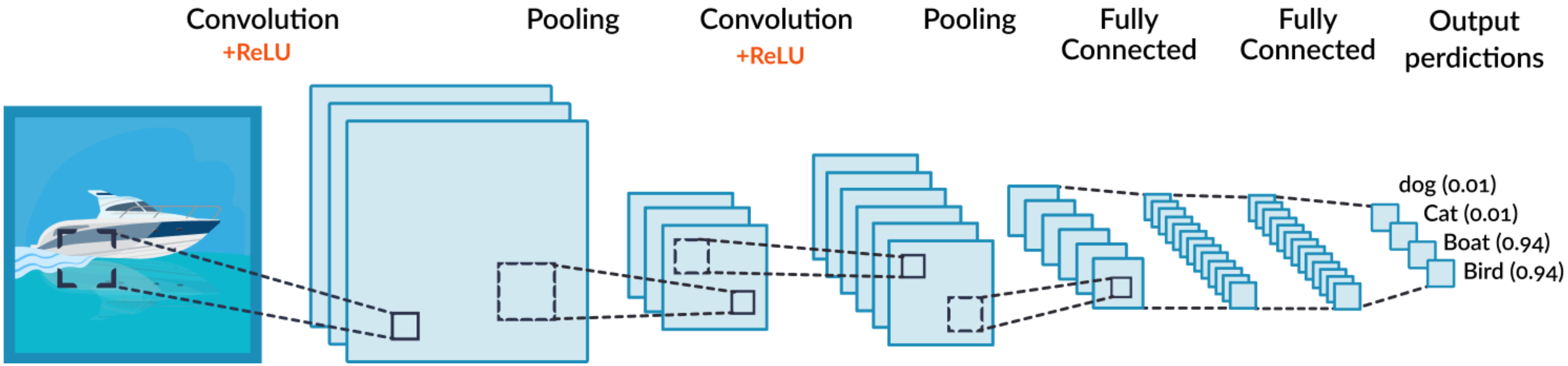
```
class DeepNeuralClassifier(object):
    def __init__(self, neurons):
        self.layers = neurons
        self.grads = {}

    def reset_state(self):
        self.Zl = []
        self.Al = []

    def initialize_parameters(self, X):
        parameters = {}
        layer_dims = [X.shape[0]] + self.layers
        L = len(layer_dims)
        for l in range(1, L):
            current, prev = layer_dims[l], layer_dims[l-1]
            scaler = np.sqrt(prev)
            randomized = np.random.randn((current, prev))
            parameters['W'+str(l)] = randomized/scaler
            parameters['b'+str(l)] = np.zeros((current,1))
        self.parameters = parameters

    def activate(self, Z, mode):
        if mode == "sigmoid":
            return 1/(1+np.exp(-Z))
        else:
            return np.maximum(0,Z)

    def activate_backward(self, dA, Z, mode):
        if mode =='sigmoid':
            s = 1/(1+np.exp(-Z))
            return dA * s * (1-s)
        else:
            dZ = np.array(dA, copy=True)
            dZ[Z <= 0] = 0
            return dZ
```

```
def forward_layer(self, A_prev, l, activation):
    W = self.parameters['W' + str(l)]
    b = self.parameters['b' + str(l)]

    Z = np.dot(W, A_prev) + b
    A = self.activate(Z, activation)
    return A, Z

def forward_propagation(self, X):
    self.Al.append(X)
    L = len(self.parameters) // 2

    for l in range(1, L+1):
        A_prev = self.Al[l-1]
        if l < L:
            activation = 'relu'
        else:
            activation = 'sigmoid'
        A, Z = self.forward_layer(A_prev, l, activation)
        self.Zl.append(Z)
        self.Al.append(A)

def backward_layer(self, dA, l, activation):
    A_prev = self.Al[l]
    wl = self.parameters['W' + str(l+1)]
    dZ = self.activate_backward(dA, self.Zl[l], activation)

    m = A_prev.shape[1]

    dW = np.dot(dZ, A_prev.T)/m
    db = np.sum(dZ, axis=1, keepdims=True)/m
    dA_prev = np.dot(wl.T, dZ)

    return dA_prev, dW, db

def backward_propagation(self, Y):
    L = len(self.parameters) // 2
    A = self.Al[-1]
    m = A.shape[1]
    Y = Y.reshape(A.shape)
    dA_prev = - (np.divide(Y, A) - np.divide(1-Y, 1-A))

    for l in reversed(range(L)):
        if l == L-1:
            activation = "sigmoid"
        else:
            activation = "relu"
        dA_prev, dW, db = self.backward_layer(dA_prev, l, activatio
        self.grads["dA" + str(l)] = dA_prev
        self.grads["dW" + str(l + 1)] = dW
        self.grads["db" + str(l + 1)] = db
```

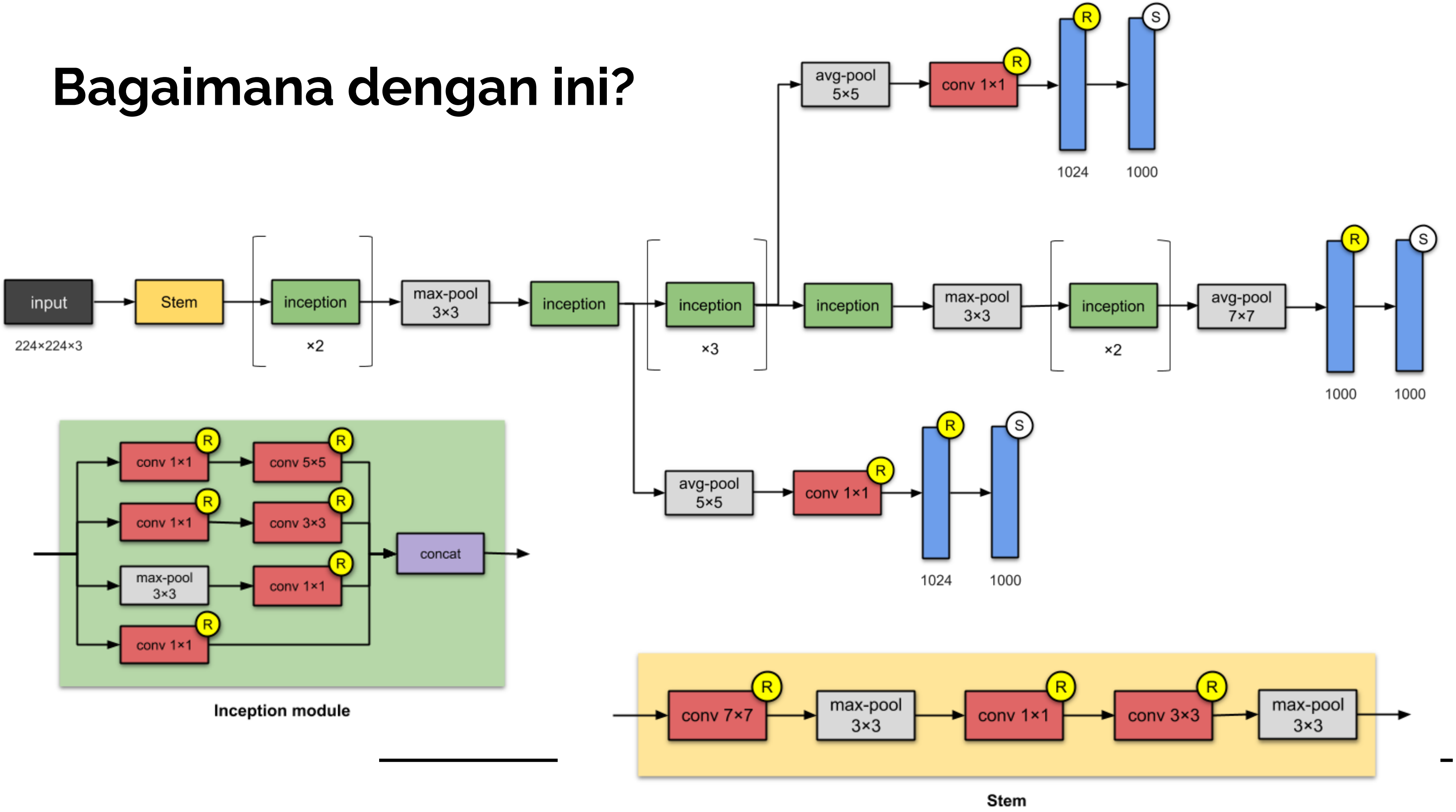
```
def compute_cost(self, Y):
    def logAY(a, y):
        return np.multiply(np.log(a), y)
    m = Y.shape[1]
    AL = self.Al[-1]
    cost = -np.sum(logAY(AL,Y)+logAY(1-AL,1-Y))/m
    cost = np.squeeze(cost)
    return cost

def update_parameters(self, lr):
    L = len(self.parameters) // 2
    for l in range(L):
        self.parameters["W"+str(l+1)] -= lr*self.grads["dW"+str(l+1)]
        self.parameters["b"+str(l+1)] -= lr*self.grads["db"+str(l+1)]
    return parameters

def train(self, X, Y, lr, num_iterations = 3000, print_cost=False):
    costs = []
    self.initialize_parameters(X)
    for i in range(0, num_iterations):
        self.reset_state()
        self.forward_propagation(X)
        cost = self.compute_cost(Y)
        self.backward_propagation(Y)
        self.update_parameters(lr)
        if print_cost and i % 100 == 0:
            print ("Cost setelah iterasi ke-%i: %f" %(i, cost))
            costs.append(cost)
    return costs

def predict(self, X, y):
    m = X.shape[1]
    p = np.zeros((1,m))
    probas = self.forward_propagation(X)[0][-1]
    for i in range(0, probas.shape[1]):
        if probas[0,i] > 0.5:
            p[0,i] = 1
        else:
            p[0,i] = 0
    print ("predictions: " + str(p))
    print ("true labels: " + str(y))
    print("Accuracy: "  + str(100*np.sum((p == y)/m)))
    return p
```

Tapi, itu pun baru jaringan sederhana...

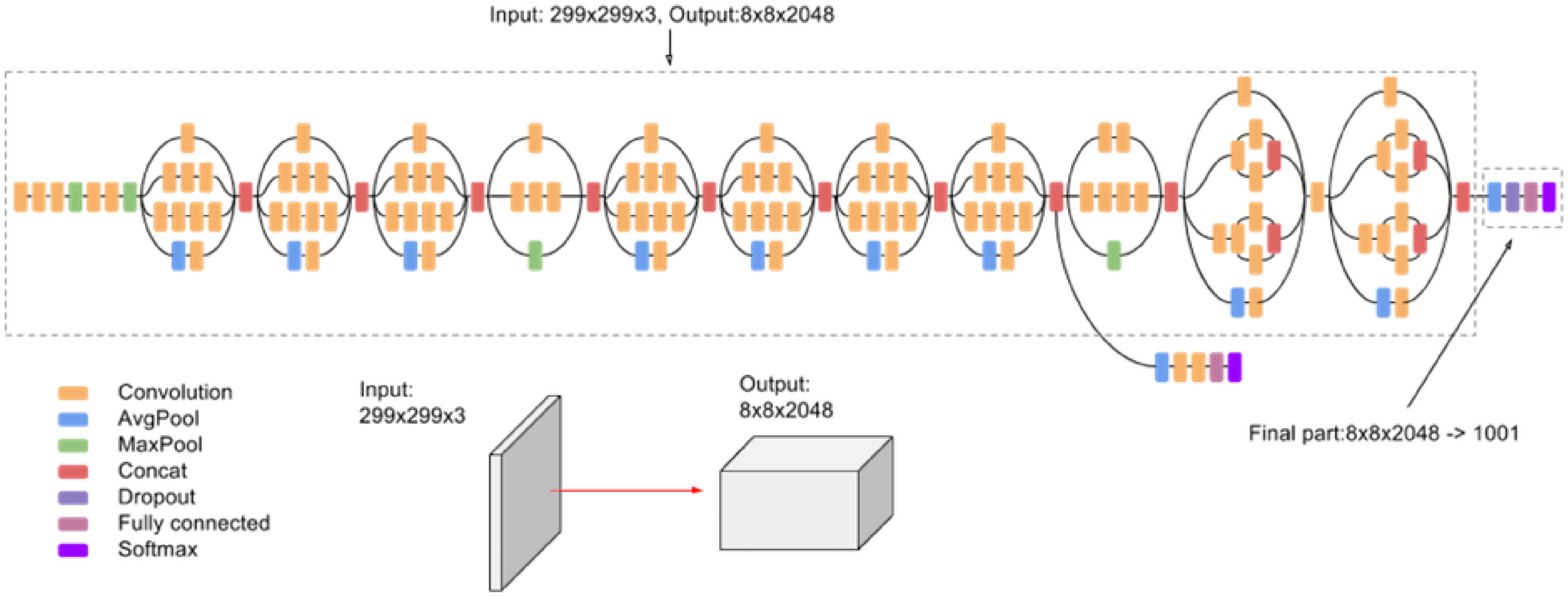
Model konvolusi?



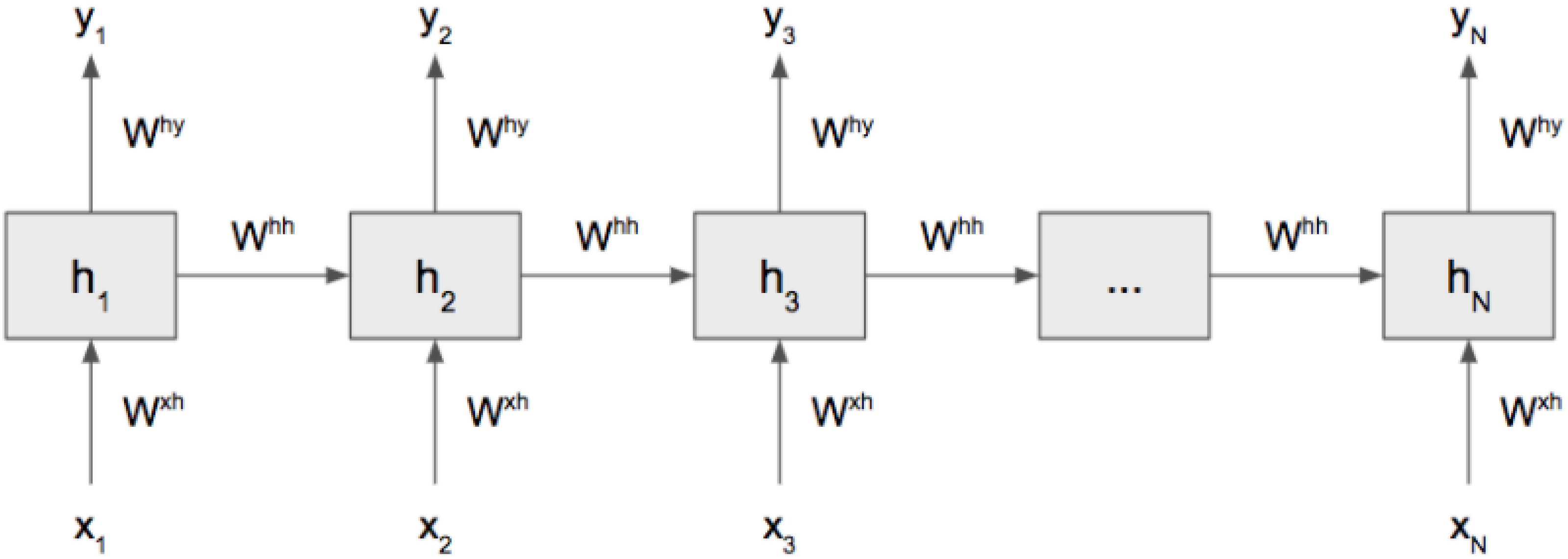
Bagaimana dengan ini?



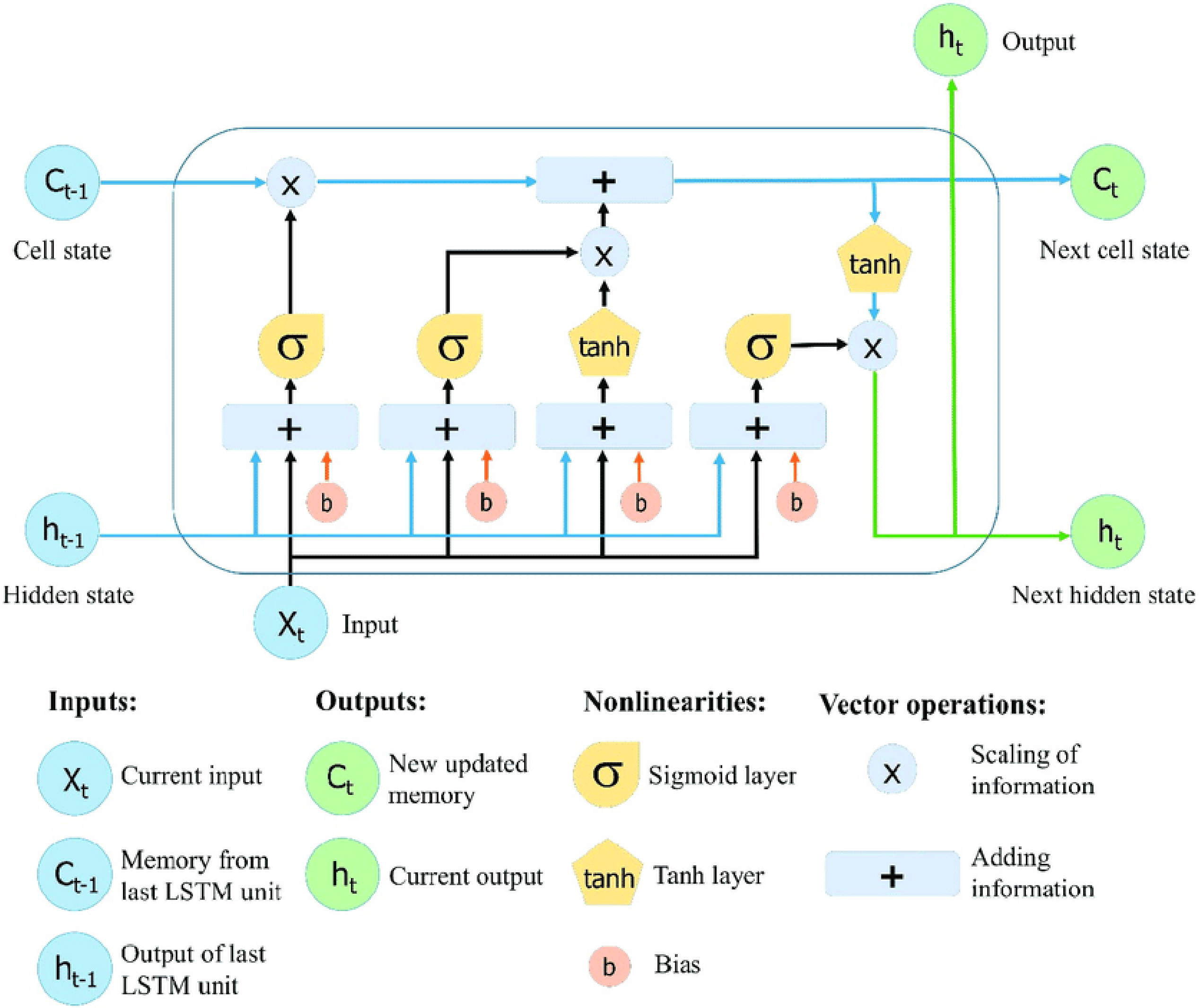
Atau ini?


Input: 299x299x3, Output:8x8x2048
Convolution
AvgPool
MaxPool
Concat
Dropout
Fully connected
Softmax
Input:
299x299x3
Output:
8x8x2048
Final part:8x8x2048 -> 1001

Belum lagi kalau rekuren

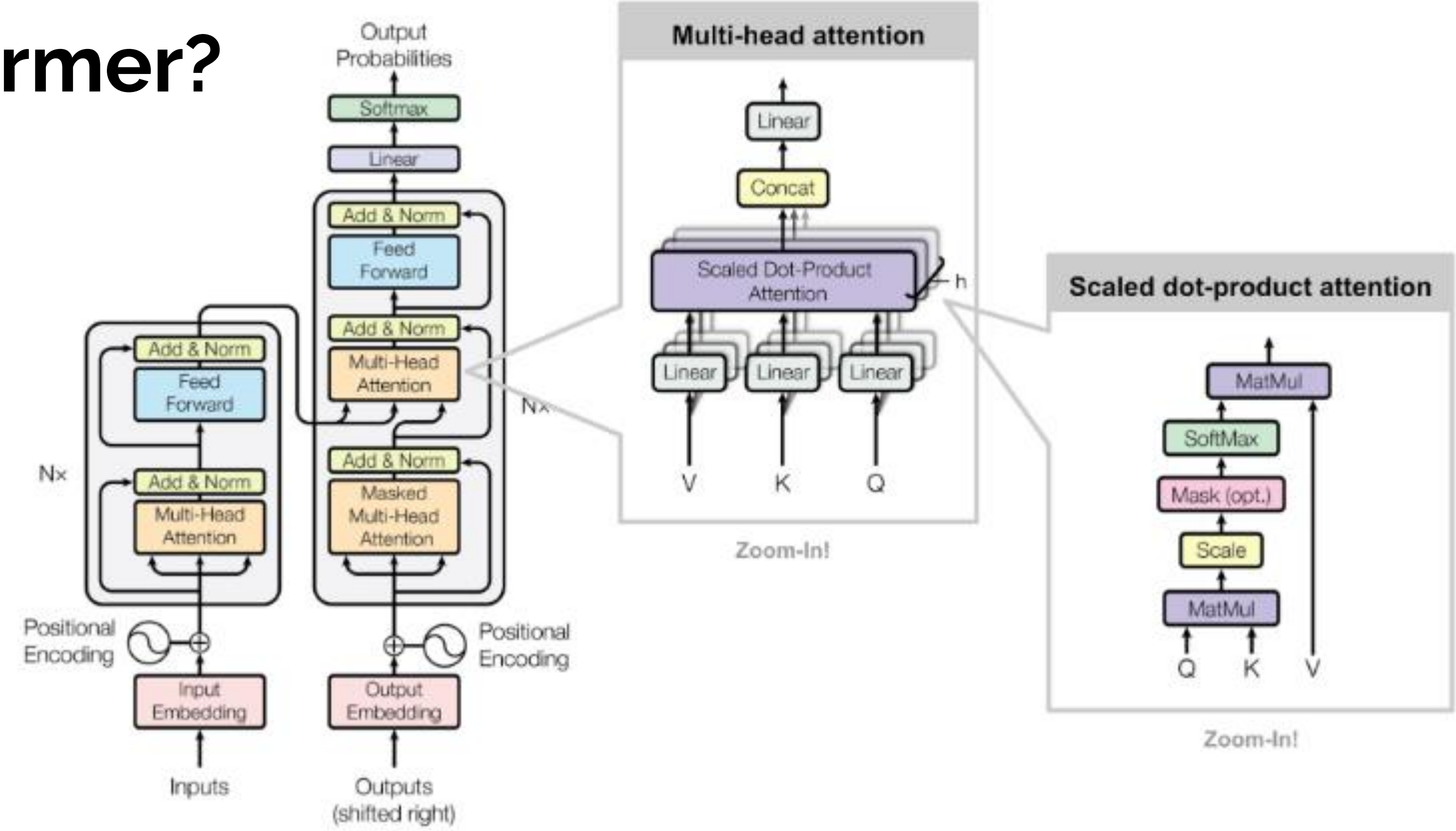


LSTM?



Output
Cell state
Next cell state
Hidden state
Input
Next hidden state
Inputs:
Current input
Memory from last LSTM unit
Output of last LSTM unit
Outputs:
New updated memory
Current output
Nonlinearities:
Sigmoid layer
Tanh layer
Bias
Vector operations:
Scaling of information
Adding information

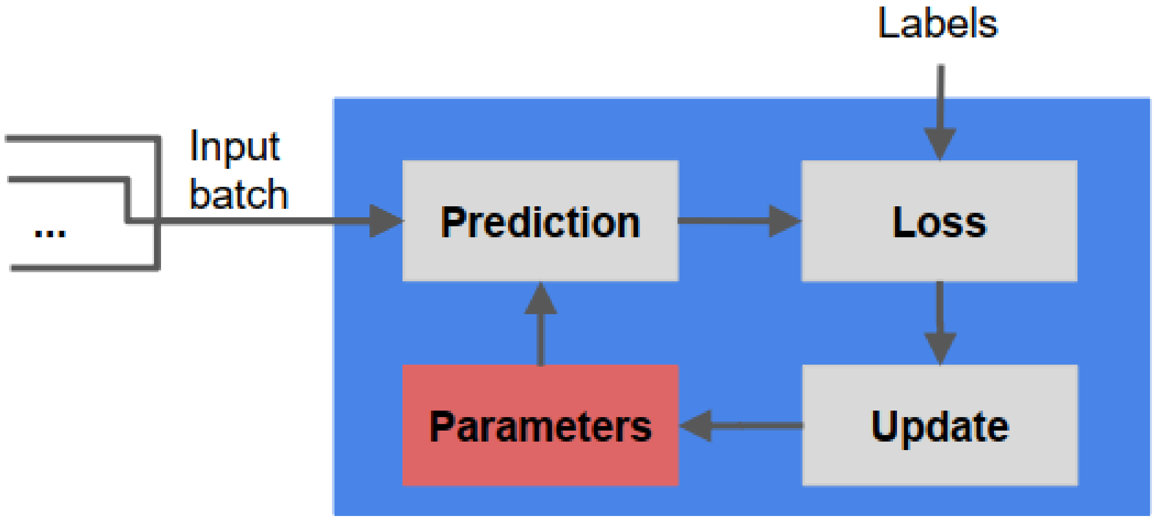
Transformer?



Cukup!

Intinya, NN akan terus semakin kompleks

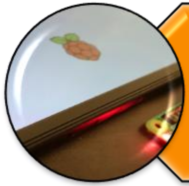
Padahal, di setiap modelnya,



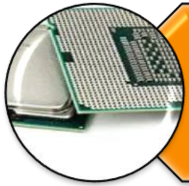
Dan tidak ada gunanya semua itu kalau hanya bisa dipakai di komputer gede

Portability is a requirement

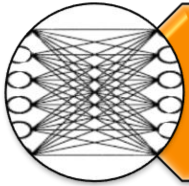
Tiga tantangan



Sistem yang heterogen



Daya komputasi



Kompleksitas Model

Tensor

A different framework of computing is needed

Apa itu tensor?

Sederhananya, tensor adalah *array* n-dimensi

Tensor

Apa itu tensor?

Komponen dasar tensor adalah dimensi dan tipe datanya.

Beberapa tipe data tensor:

int, uint, qint, float, complex, string, bool, variant

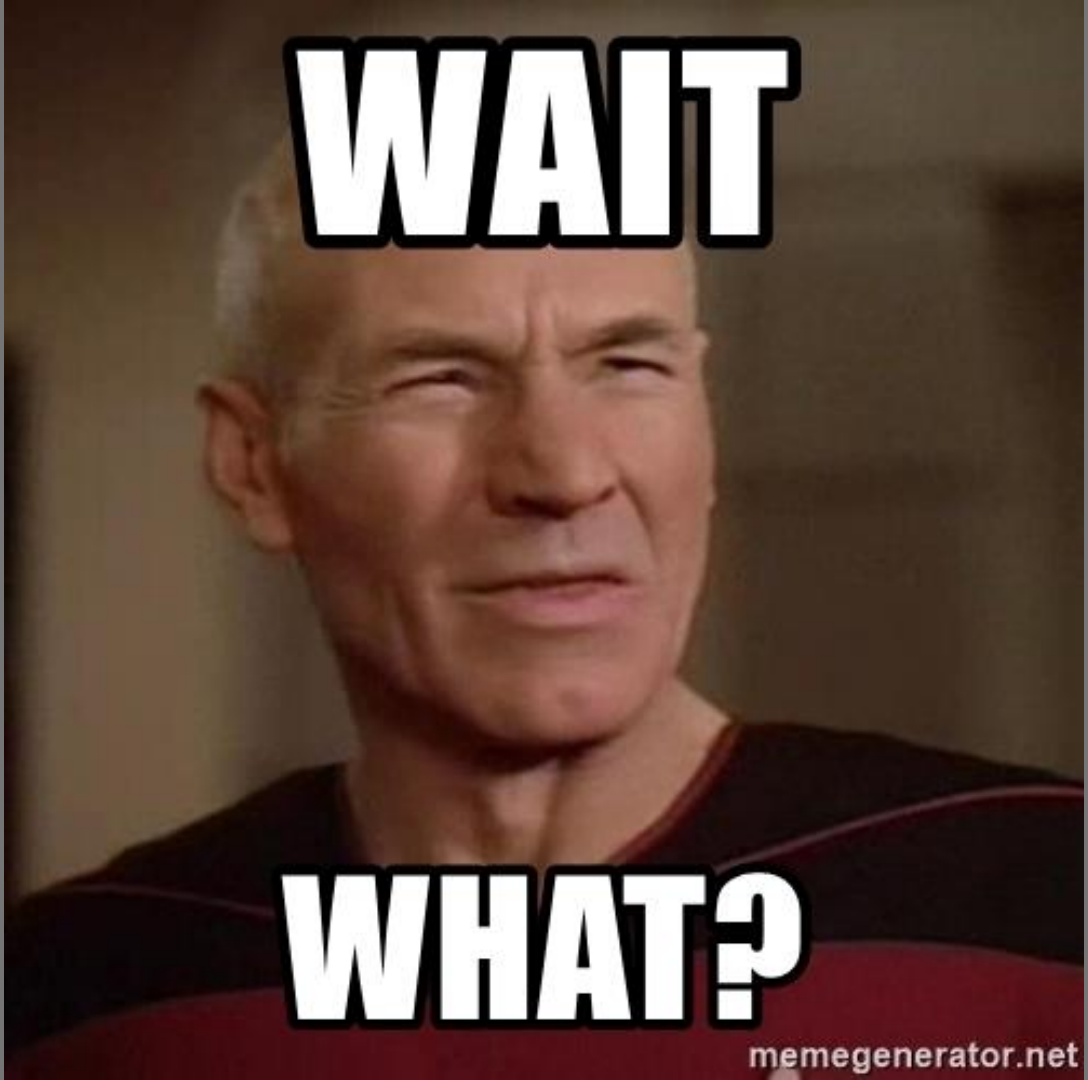
Kan sudah ada *array*

Kenapa harus tensor?

Kenapa harus tensor?

Tensor merupakan array yang menerapkan paradigma *differentiable programming*

Apa itu?
Sederhananya, konsep program dimana perhitungan numerik dapat dihitung turunannya melalui graf komputasi yang dibangun


WAIT
WHAT?
memegenerator.net

Kenapa harus tensor?

Tensor tidak sesederhana “penyimpan data”, ia “penyimpan komputasi”.
Maksudnya apa? Perhatikan 2 kode berikut

```
import numpy as np
np.array([[1,2],[3,4]])**2
```

```
array([[ 1,  4],
       [ 9, 16]])
```

```
import tensorflow as tf
t = tf.constant([[1,2],[3,4]])
t**2
```

```
<tf.Tensor 'pow_1:0' shape=(2, 2) dtype=int32>
```

Loh kok yang tensor tidak dihitung kuadratnya?

Kenapa harus tensor?

Bukan tidak dihitung, tapi operasinya “disimpan” dulu, sebagai bagian dari tensornya.

Untuk dapatkan hasilnya, ia perlu dikompilasi dan dijalankan dalam suatu sesi

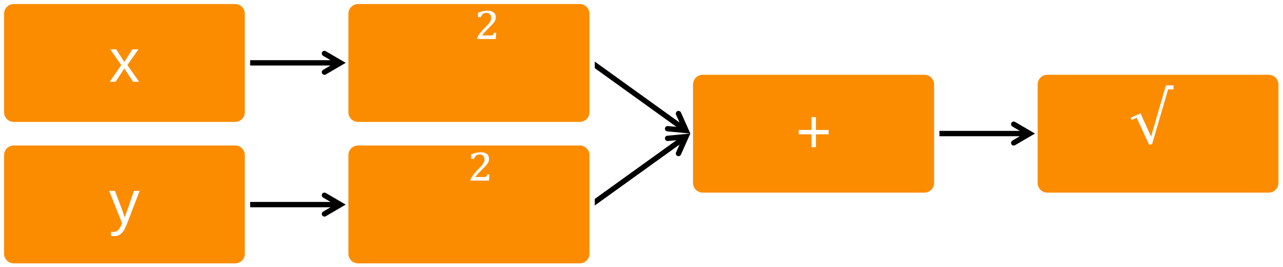
```
sess = tf.compat.v1.Session()
sess.run(t)
```

```
array([[1, 2],
       [3, 4]], dtype=int32)
```

Apa itu tadi sebutannya?

Graf Komputasi?

Ya, graf komputasi, atau *computation graph*, yakni graf asiklik berarah (DAG) dimana setiap *node*-nya merupakan sebuah operasi elementer. Misal, apa sebenarnya yang dilakukan Ketika menghitung $\sqrt{x^2 + y^2}$?

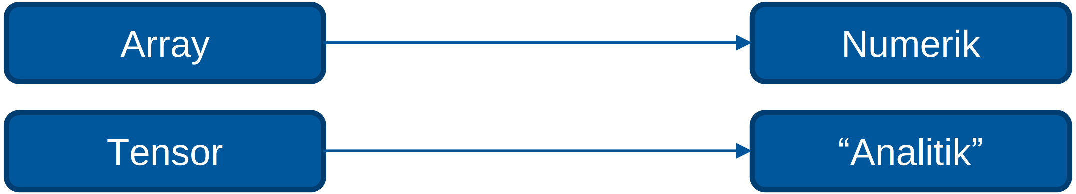


Graf Komputasi

Tensor sebagai model *graf* itu sendiri

"Kalau cuma butuh hitung data, Tensor itu *overkill*"

Tensor dibandingkan dengan array seperti variabel dibandingkan dengan konstanta. Variabel membaca semua operasi yang dilakukan padanya tanpa harus menghitung nilainya.



Graf Komputasi

Manfaatnya apa?

(1) Menghitung turunan.

Misal $L = \sqrt{x^2 + y^2}$, maka dengan aturan rantai sederhana

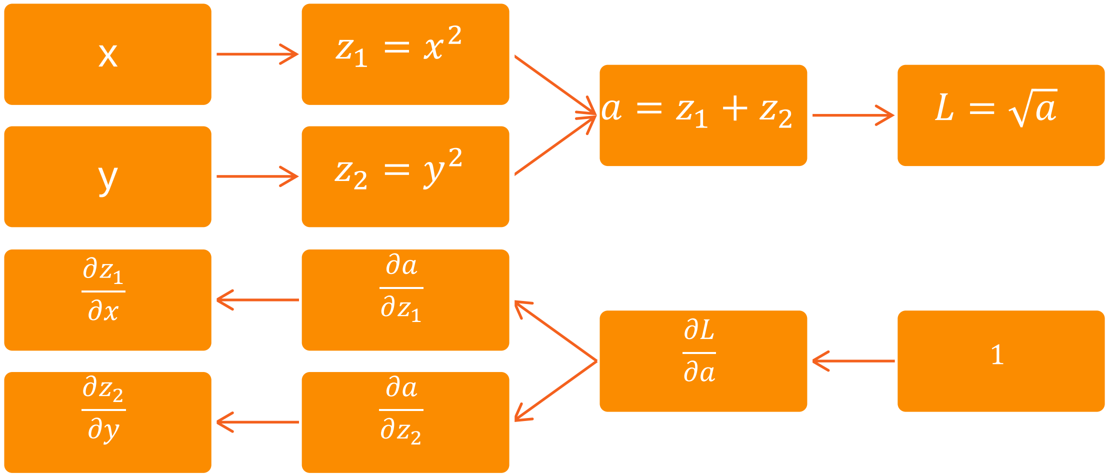
$$\frac{\partial L}{\partial x} = \left(\frac{\partial L}{\partial (x^2 + y^2)}\right)\left(\frac{\partial (x^2 + y^2)}{\partial x^2}\right)\left(\frac{\partial x^2}{\partial x}\right)$$

Graf Komputasi

Manfaatnya apa?

(1) Menghitung turunan.

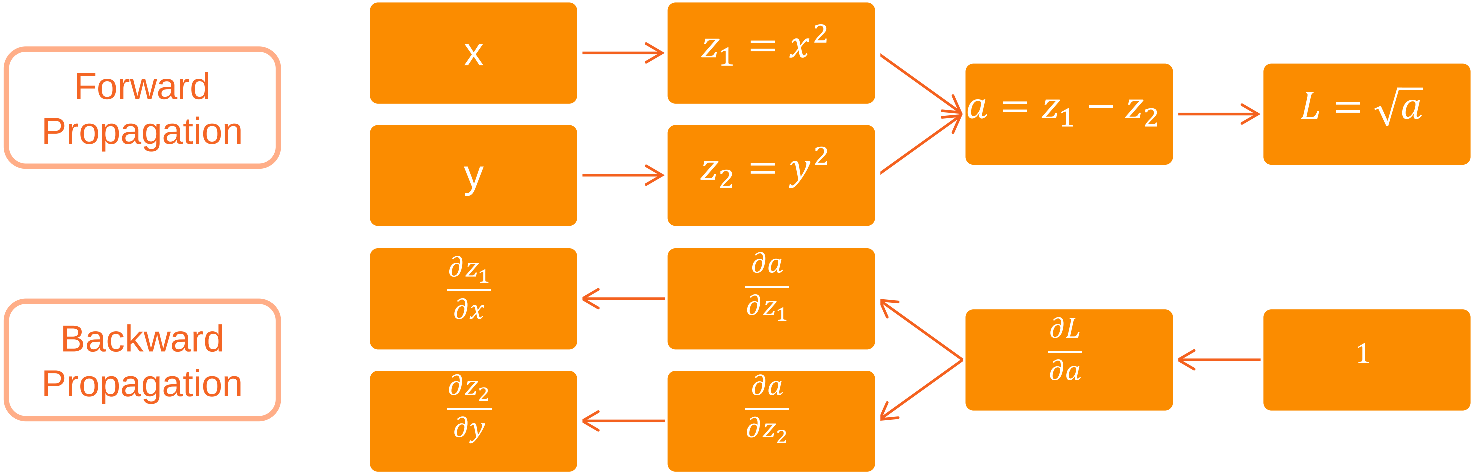
Dengan graf komputasi, jadi seperti berikut



Graf Komputasi

Manfaatnya apa?

Konsepnya hanya aturan rantai. Tapi bagaimana kalau fungsinya kompleks, NN-nya sangat dalam?



```python
x = tf.Variable(5.)
y = tf.Variable(7.)
```

Mendefinisikan tensornya

```python
with tf.GradientTape(persistent=True) as t1:
  with tf.GradientTape(persistent=True) as t2:
    z1 = tf.square(x)
    z2 = tf.square(y)
    a = tf.add(z1, z2)
    L = tf.sqrt(a)
```

x^2

y^2

Melakukan komputasi
Di dalam lingkup gradient tape
agar “direkam” turunan dari
graf komputasinya

```python
    dL_dx = t1.gradient(L, x)
    dL_dy = t1.gradient(L, y)
```

Menghitung
Turunan pertama

```python
d2L_dx2 = t2.gradient(dL_dx, x)
d2L_dxy = t2.gradient(dL_dx, y)
d2L_dyx = t2.gradient(dL_dy, x)
d2L_dy2 = t2.gradient(dL_dy, y)
```

Menghitung
Turunan kedua

```python
print("Turunan pertama")
print("L_x:",dL_dx,"\nL_y:", dL_dy)
print("\nTurunan Kedua")
print("L_xx:",d2L_dx2,"\nL_xy:",d2L_dxy,"\nL_yx:",d2L_dyx,"\nL_yy:",d2L_dy2)
```

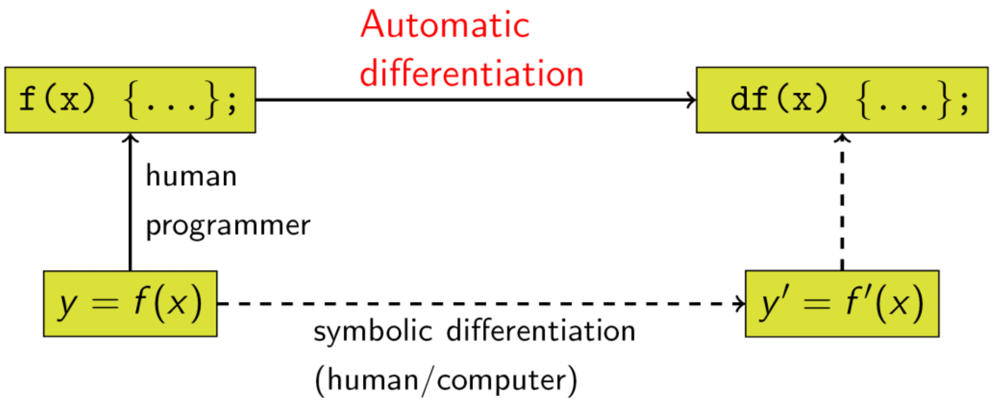
```
Turunan pertama
L_x: tf.Tensor(0.5812382, shape=(), dtype=float32)
L_y: tf.Tensor(0.81373346, shape=(), dtype=float32)

Turunan Kedua
L_xx: tf.Tensor(0.07697479, shape=(), dtype=float32)
L_xy: tf.Tensor(-0.054981995, shape=(), dtype=float32)
L_yx: tf.Tensor(-0.054981988, shape=(), dtype=float32)
L_yy: tf.Tensor(0.03927286, shape=(), dtype=float32)
```

Graf Komputasi

Manfaatnya apa?

Menghitung turunan melalui graf komputasi ini disebut *automatic differentiation* atau *algorithmic differentiation* atau *computational differentiation*. Singkatnya *auto-diff*.

Ini inti dari *differentiable programming*

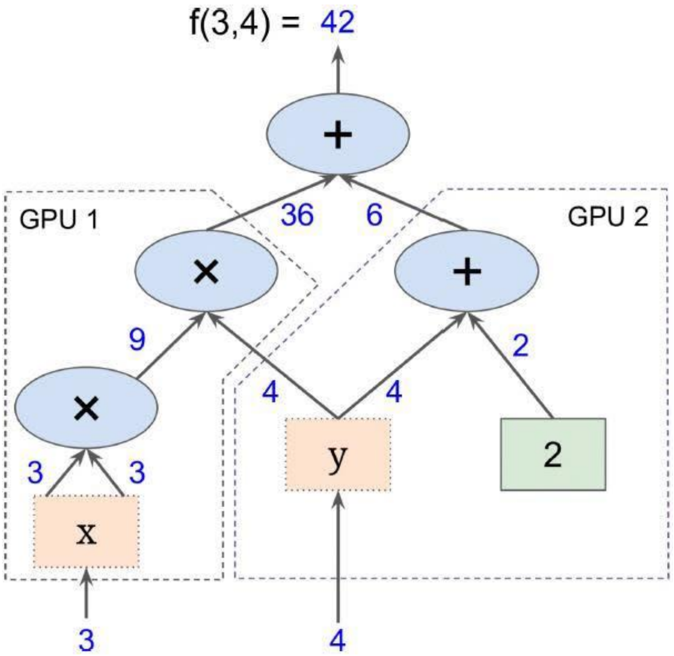


Graf Komputasi

Manfaatnya apa?

(2) Komputasi pararel.

Setiap subgraf bisa didistribusikan
Ke mesin/prosesor yang berbeda



Graf Komputasi

Perhatikan lagi kode sebelumnya

```
import tensorflow as tf
t = tf.constant([[1,2],[3,4]])
t**2
```

```
<tf.Tensor 'pow_1:0' shape=(2, 2) dtype=int32>
```

```
sess = tf.compat.v1.Session()
sess.run(t)
```

```
array([[1, 2],
       [3, 4]], dtype=int32)
```

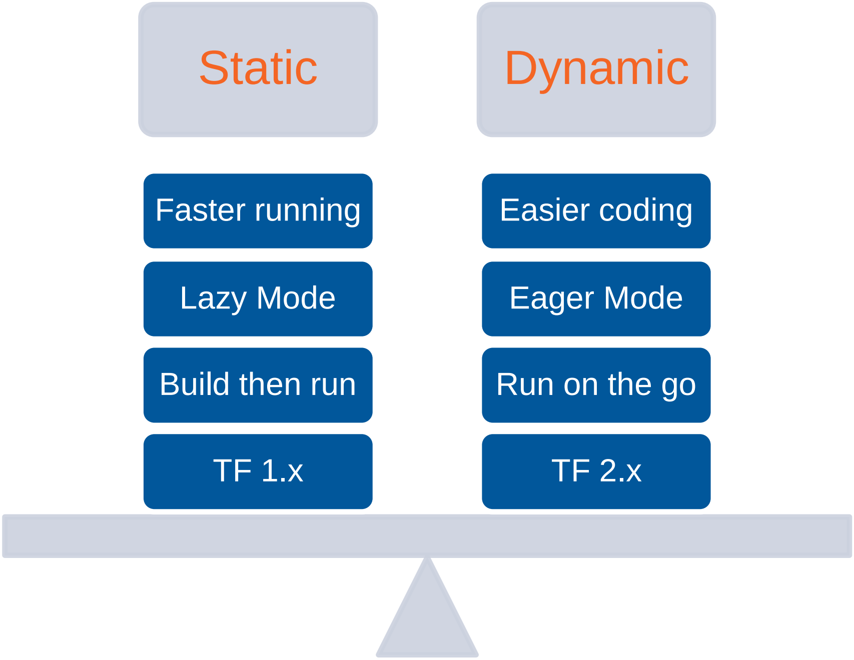
Cukup repot kan kita hanya bisa lihat nilai hasilnya kalau dijalankan di sesi

Graf Statis dan Graf Dinamis

Arsitektur graf yang harus dihitung melalui sesi disebut sebagai *static graph*, lebih **rumit** namun lebih **cepat**

Di TF versi 2.x, dikembangkan mode dinamis agar setiap kali graf bisa “dieksekusi” setiap saat tanpa harus kompilasi

Graf Statis dan Graf Dinamis



Graf Statis dan Graf Dinamis

Eager Mode

```
x = tf.constant([1,2,3], dtype=float)
tf.square(x)
```

```
<tf.Tensor: shape=(3,), dtype=float32, numpy=array([1., 4., 9.], dtype=float32)>
```

```
z1 = z.numpy()
z1
```

```
array([1., 4., 9.], dtype=float32)
```

Lazy Mode

```
x = tf.constant([1,2,3], dtype=float)
tf.square(x)
```

```
<tf.Tensor 'Square_3:0' shape=(3,) dtype=float32>
```

```
with tf.compat.v1.Session() as s:
  z1 = s.run(z)
z1
```

```
array([1., 4., 9.], dtype=float32)
```

Tensorflow = alat untuk menggunakan tensor?

Ya Iya sih

Tensorflow sendiri merupakan sebuah framework *Deep Learning*, yang memanfaatkan konsep tensor untuk efektivitas komputasi

Tapi...

penggunaan konsep tensor tidak hanya oleh Tensorflow, sudah sering jga dipakai beberapa *framework* NN (kadang dengan istilah yang berbeda).

Yang populer adalah dua ini

FB

PyTorch

G

TensorFlow

K

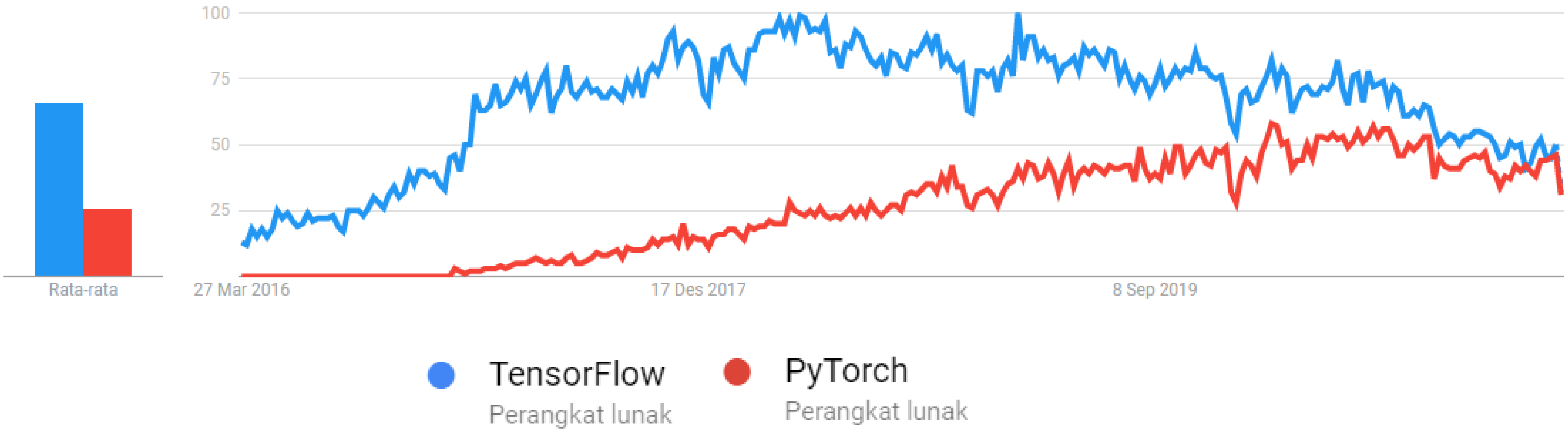
Microsoft Cognitive Toolkit

mxnet

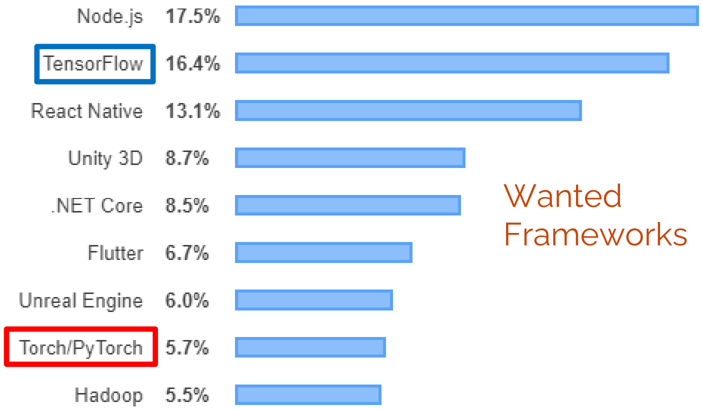
Caffe

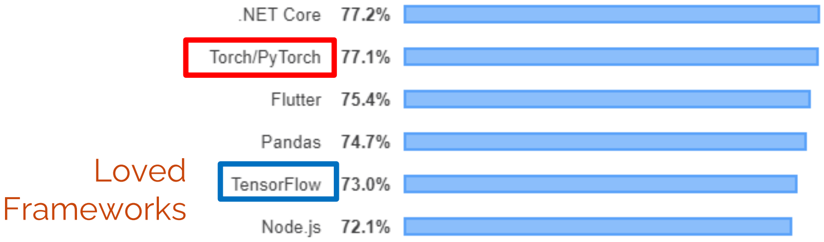
Pilih yang mana?

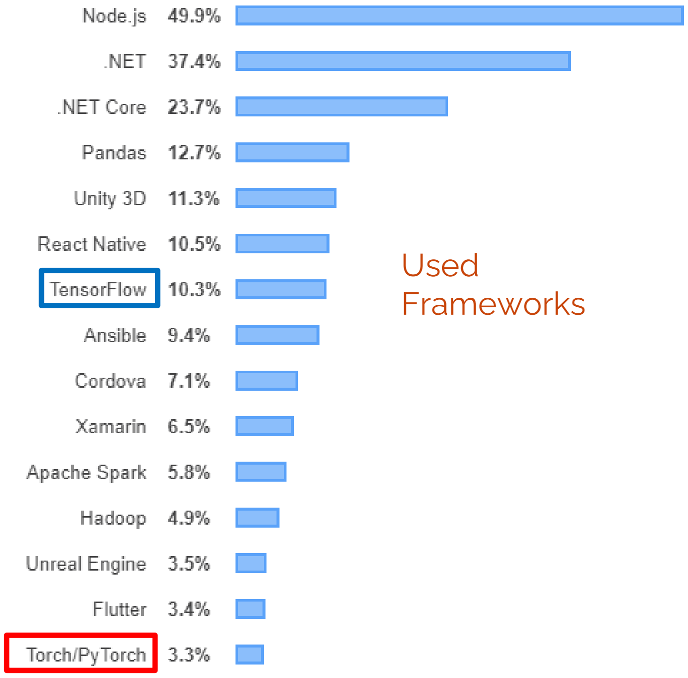
Demistifikasi (1) – Google Trends



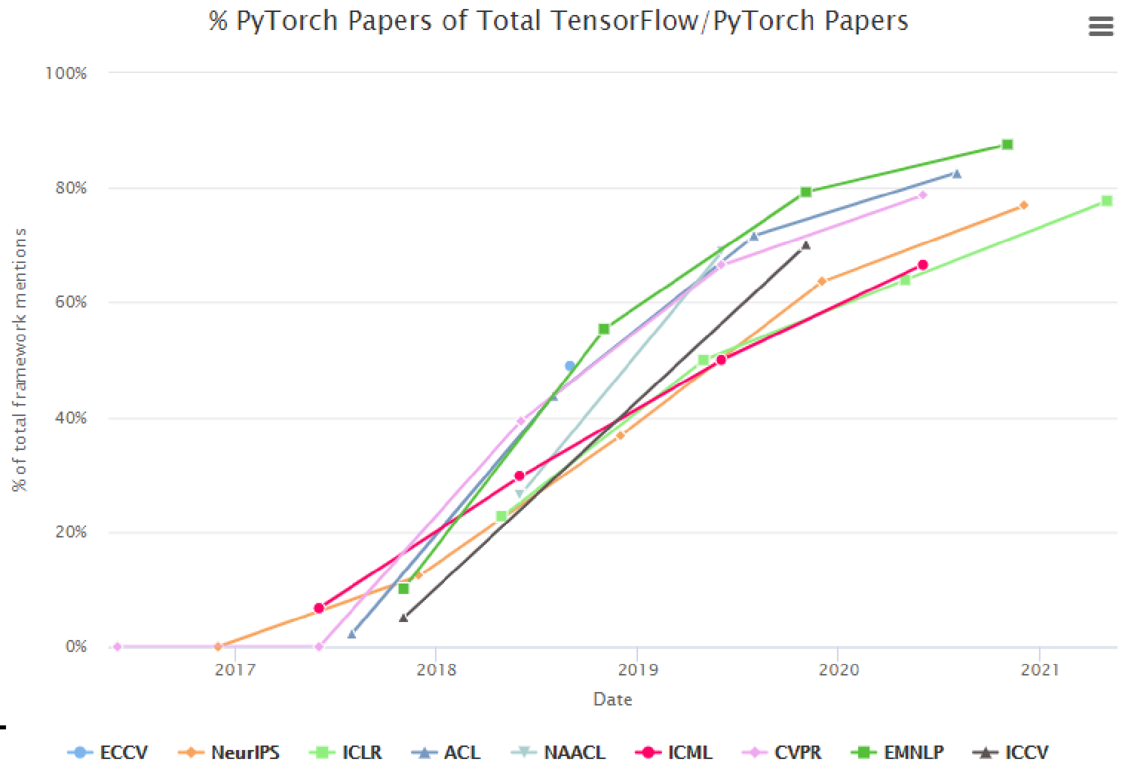
Demistifikasi (2) – Stack Overflow Developer Survey 2019







Demistifikasi (3) – Research Papers



Loh? Jadi?

TensorFlow vs PyTorch

- Tensorflow v1 rilis Nov 2015 dengan *Static Graph*
- PyTorch rilis Sept 2016 dengan *Dynamic Graph*
- *Dynamic Graph* lebih mudah digunakan dan dipahami -> PyTorch semakin disukai, terutama di kalangan peneliti
- Integrasi Tensorflow ke berbagai sistem berkembang membuat industry banyak memakai TF
- Tensorflow v2 rilis Jan 2019 dengan *Dynamic Graph*, tapi para peneliti sudah terlanjur pakai PyTorch
- Sekarang keduanya hampir sama dari segi arsitektur


Soumith Chintala
@soumithchintala
Debates on PyTorch vs TensorFlow were fun in 2017.
There was healthy competition to innovate, and philosophical differences like Theano vs Torch, Emacs vs vim, or android vs iOS.
Now both products look exactly the same, the debates are nonsense and boring. Please stop.
10:27 PM · May 22, 2020

Saya pribadi?

Saya belajar keduanya, tapi prefer Tensorflow

Kenapa Tensorflow?

Kenapa Tensorflow

Ada banyak alasan, diantaranya

(1) Large supporting communities

Github Repository Data (24/3/21):

	Forks	Contributors	Commits	Stars	Users
TensorFlow	84.3rb	2939	107.5rb	154rb	125rb
PyTorch	12.5rb	1788	34.7rb	47.1rb	64.7rb

Kenapa Tensorflow

Ada banyak alasan, diantaranya

(1) Large supporting communities

TFHub dan TF Model Garden -> kumpulan model dari peneliti, pengembang, praktisi seluruh dunia untuk saling mengembangkan

TensorFlow Model Garden

TensorFlow Hub

TensorFlow Hub

Search for models, collections & publishers

Send feedback

Filters

Clear all

Problem domain

Model format

TF.js | TFLite | Coral

TF Version

TF1 | TF2

Fine tunable

Architecture

Publisher

Dataset

Text embedding

universal-sentence-encoder-multilingual

Publisher: Google Updated: 03/01/2021 255.5k

16 languages (Arabic, Chinese-simplified, Chinese-traditional, English, French, German, Italian, Japanese, Korean, Dutch, Polish, Portuguese, Spanish, Thai, Turkish, Russian) text encoder.

Architecture: CNN

Text embedding

universal-sentence-encoder

Publisher: Google Updated: 03/01/2021 733.2k

Encoder of greater-than-word length text trained on a variety of data.

Architecture: DAN

Text embedding

universal-sentence-encoder-large

Publisher: Google Updated: 03/01/2021 821.9k

Encoder of greater-than-word length text trained on a variety of data.

Architecture: Transformer

Image feature vector .JS

imagenet/resnet_v2_50/feature_vector

Publisher: Google Updated: 03/01/2021 92.5k

Feature vectors of images with ResNet V2 50 trained on ImageNet (ILSVRC-2012-CLS).

Architecture: ResNet V2 50 | Dataset: ImageNet (ILSVRC-2012-CLS)

TensorFlow Model Garden

Welcome to the Model Garden for TensorFlow

The TensorFlow Model Garden is a repository with a number of different implementations of state-of-the-art (SOTA) models and modeling solutions for TensorFlow users. We aim to demonstrate the best practices for modeling so that TensorFlow users can take full advantage of TensorFlow for their research and product development.

Directory	Description
official	• A collection of example implementations for SOTA models using the latest TensorFlow 2's high-level APIs • Officially maintained, supported, and kept up to date with the latest TensorFlow 2 APIs by TensorFlow • Reasonably optimized for fast performance while still being easy to read
research	• A collection of research model implementations in TensorFlow 1 or 2 by researchers • Maintained and supported by researchers
community	• A curated list of the GitHub repositories with machine learning models and implementations powered by TensorFlow 2
orbit	• A flexible and lightweight library that users can easily use or fork when writing customized training loop code in TensorFlow 2.x. It seamlessly integrates with `tf.distribute` and supports running on different device types (CPU, GPU, and TPU).

Kenapa Tensorflow

Ada banyak alasan, diantaranya

(2) Banyak fitur pendukung (terutama dalam hal deployment)

TensorFlow Quantum

TensorFlow.js

TensorFlow Lite

TensorBoard

TensorFlow Extended

TensorFlow Recommenders

Kenapa Tensorflow

Ada banyak alasan, diantaranya

(2) Banyak fitur pendukung

TFDS (Tensorflow Dataset)
- Kumpulan dataset siap pakai sebagai tensor

Tensorboard
- Platform untuk analisis dan visualisasi

TF Lite
- Versi ringan dari tensorflow untuk Edge atau Mobile

Tensorflow.js
- Versi javascript dari tensorflow, untuk web deployment

Tensorflow Extended (TFX)
- Framework untuk membangun end-to-end pipeline deployment

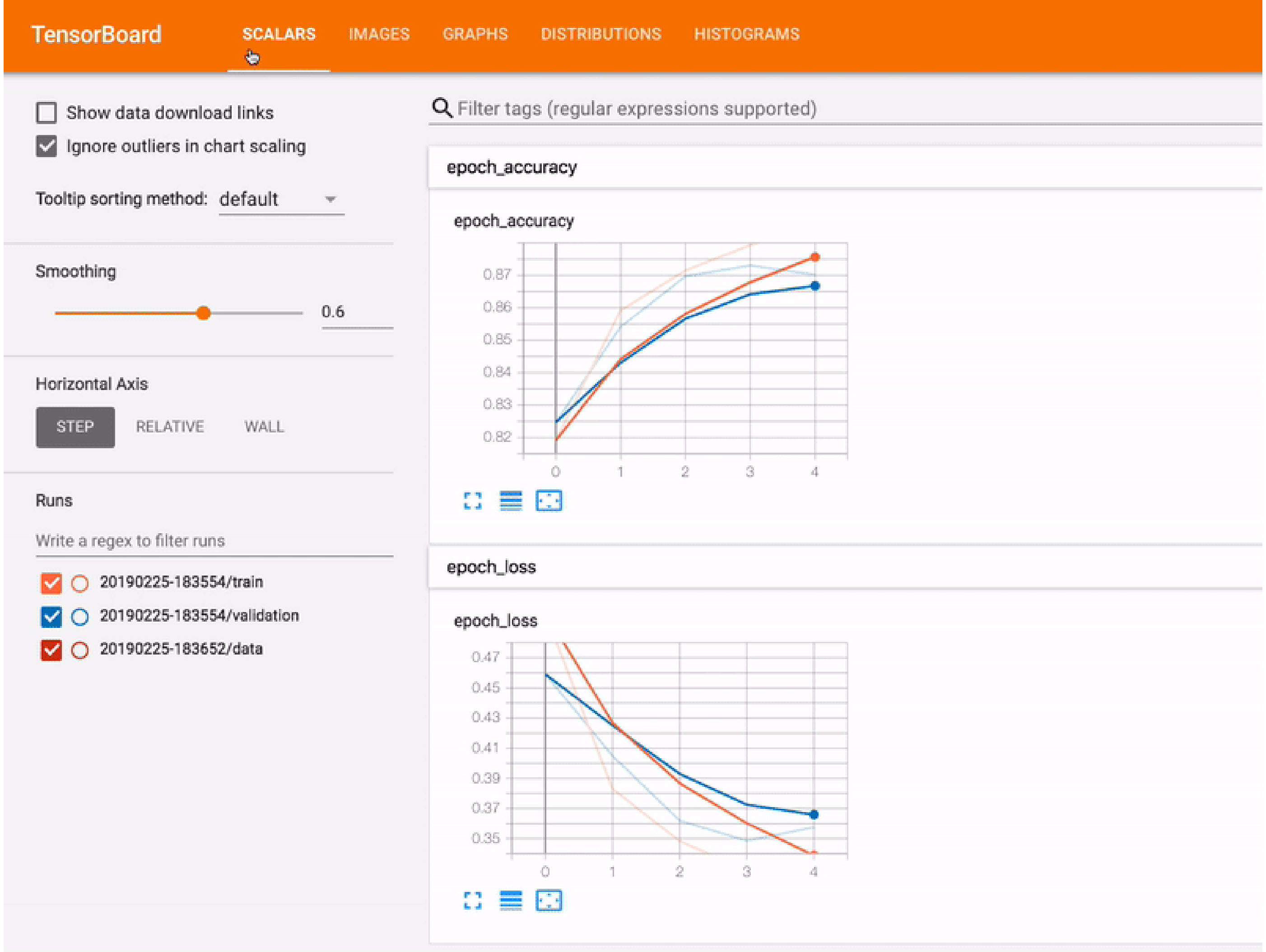
Tensorflow Agent
- Library tensorflow khusus untuk Reinforcement Learning

Tensorflow Graphic
- Library Tensorflow khusus untuk fungsi-fungsi terkait grafik computer

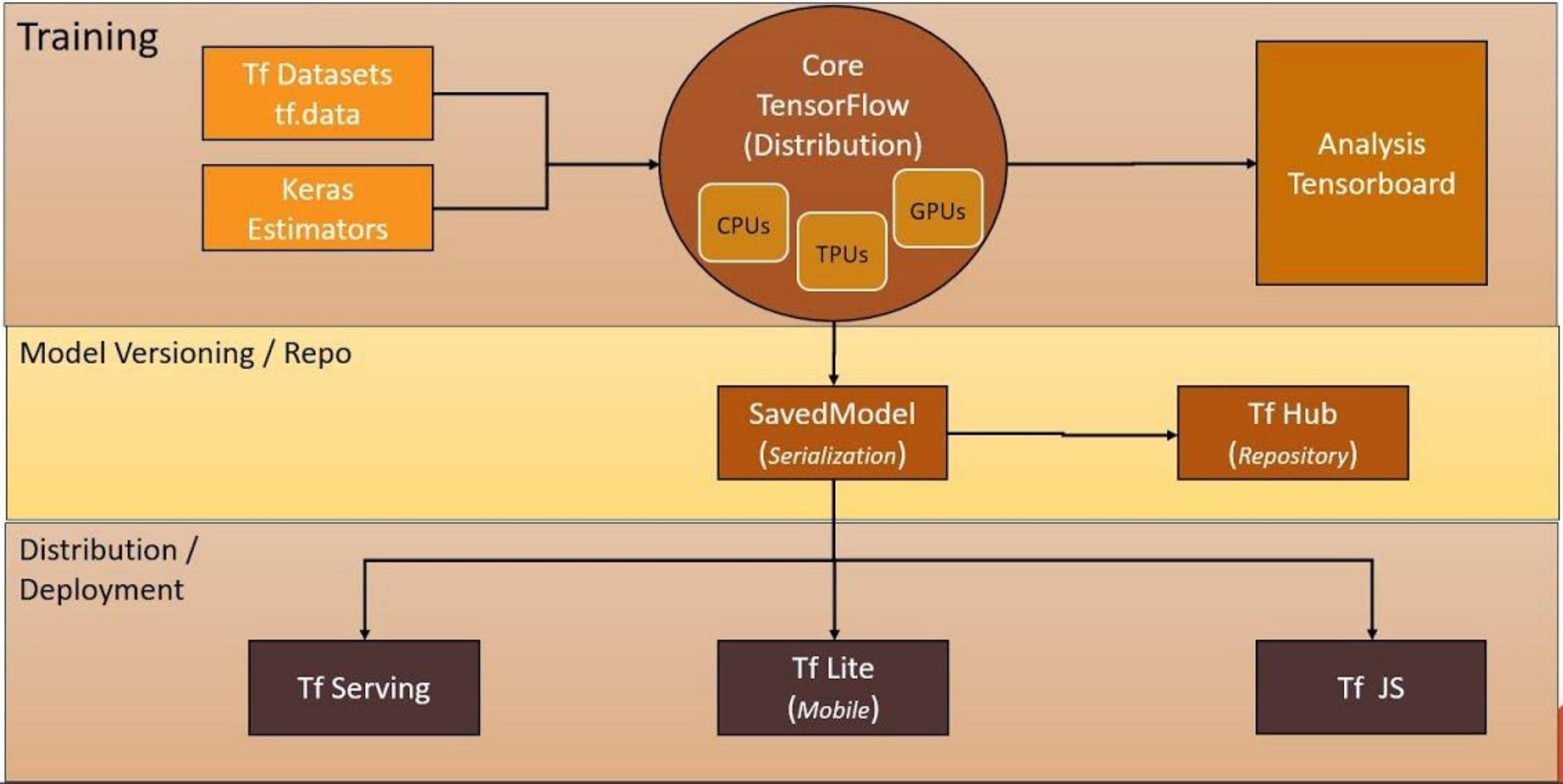
Tensorflow Federated (TFF)
- Framework tensorflow untuk desentralisasi data

Tensorflow Quantum (TFQ)
- Library tensorflow untuk utilisasi quantum computing

Tensorflow Recommenders (TFRS)
- Library tensoflow khusus untuk membangun system rekomendasi


TensorBoard
SCALARS
IMAGES
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
Show data download links
Ignore outliers in chart scaling
Tooltip sorting method: default
Smoothing
0.6
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
20190225-183554/train
20190225-183554/validation
20190225-183652/data
Filter tags (regular expressions supported)
epoch_accuracy
epoch_accuracy
0.87
0.86
0.85
0.84
0.83
0.82
0
1
2
3
4
epoch_loss
epoch_loss
0.47
0.45
0.43
0.41
0.39
0.37
0.35
0
1
2
3
4

TensorFlow 2.0



Kenapa Tensorflow

Ada banyak alasan, diantaranya

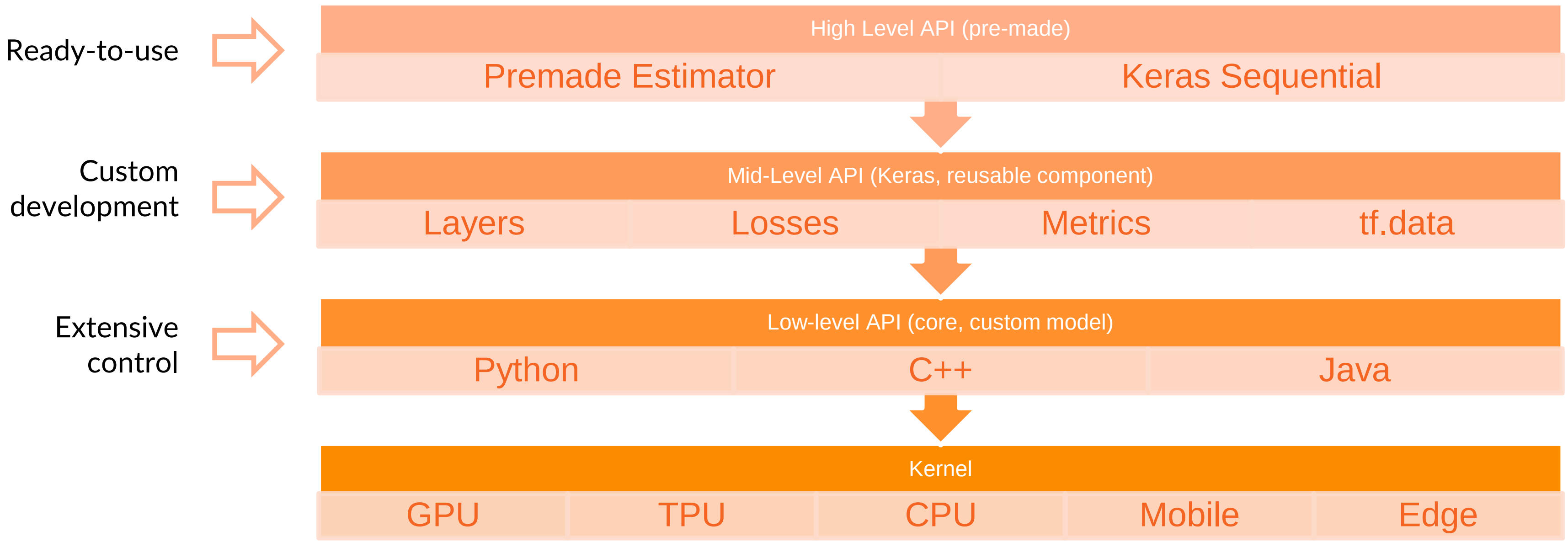
(3) Beragam strategi distribusi untuk komputasi

- Mirrored Strategy
- Single Device Strategy
- Multi Worker Mirrored Strategy
- TPU Strategy
- Custom Strategy

Kenapa Tensorflow

Ada banyak alasan, diantaranya

(4) Abstraksi yang fleksibel



Low-level API

```
class Model(object):
  def __init__(self):
    self.w = tf.Variable(2.0)
    self.b = tf.Variable(1.0)
  def __call__(self, x):
    return self.w * x + self.b

model = Model()
learning_rate = 0.1
for epoch in range(15):
  with tf.GradientTape() as t:
    y_predicted = model(xs)
    current_loss = tf.reduce_mean(tf.square(y_predicted - ys))
  dw, db = t.gradient(current_loss, [model.w, model.b])
  model.w.assign_sub(learning_rate * dw)
  model.b.assign_sub(learning_rate * db)
  print('Epoch {}/{}: loss = {}'.format(epoch, str(15), current_loss))
```

```
Epoch 0/15: loss = 2.001018524169922
Epoch 1/15: loss = 1.2801655530929565
Epoch 2/15: loss = 0.8190550804138184
Epoch 3/15: loss = 0.5240727663040161
Epoch 4/15: loss = 0.3353520631790161
Epoch 5/15: loss = 0.2146054059267044
Epoch 6/15: loss = 0.13734421133995056
Epoch 7/15: loss = 0.0879041776061058
Epoch 8/15: loss = 0.05626486986875534
Epoch 9/15: loss = 0.036015864461660385
Epoch 10/15: loss = 0.023055678233504295
Epoch 11/15: loss = 0.014760131947696209
Epoch 12/15: loss = 0.009449931792914867
Epoch 13/15: loss = 0.006050529424101114
Epoch 14/15: loss = 0.003874219721183181
```

High-level API

```
model = tf.keras.Sequential([tf.keras.layers.Dense(1)])
model.compile(optimizer='sgd', loss='mse')
model.fit(xs, ys, epochs=15)
```

```
Epoch 1/15
32/32 [==============================] - 0s 919us/step - loss: 19.4583
Epoch 2/15
32/32 [==============================] - 0s 1ms/step - loss: 4.9657
Epoch 3/15
32/32 [==============================] - 0s 824us/step - loss: 1.2787
Epoch 4/15
32/32 [==============================] - 0s 1ms/step - loss: 0.3759
Epoch 5/15
32/32 [==============================] - 0s 1ms/step - loss: 0.0917
Epoch 6/15
32/32 [==============================] - 0s 2ms/step - loss: 0.0257
Epoch 7/15
32/32 [==============================] - 0s 928us/step - loss: 0.0069
Epoch 8/15
32/32 [==============================] - 0s 1ms/step - loss: 0.0017
Epoch 9/15
32/32 [==============================] - 0s 1ms/step - loss: 5.0533e-04
Epoch 10/15
32/32 [==============================] - 0s 1ms/step - loss: 1.3377e-04
Epoch 11/15
32/32 [==============================] - 0s 971us/step - loss: 3.3871e-05
Epoch 12/15
32/32 [==============================] - 0s 1ms/step - loss: 9.5703e-06
Epoch 13/15
32/32 [==============================] - 0s 793us/step - loss: 2.8279e-06
Epoch 14/15
32/32 [==============================] - 0s 1ms/step - loss: 7.5005e-07
Epoch 15/15
32/32 [==============================] - 0s 1ms/step - loss: 2.1679e-07
```

Bagaimana memakainya?

To be continued...

Any Question?

TensorFlow™