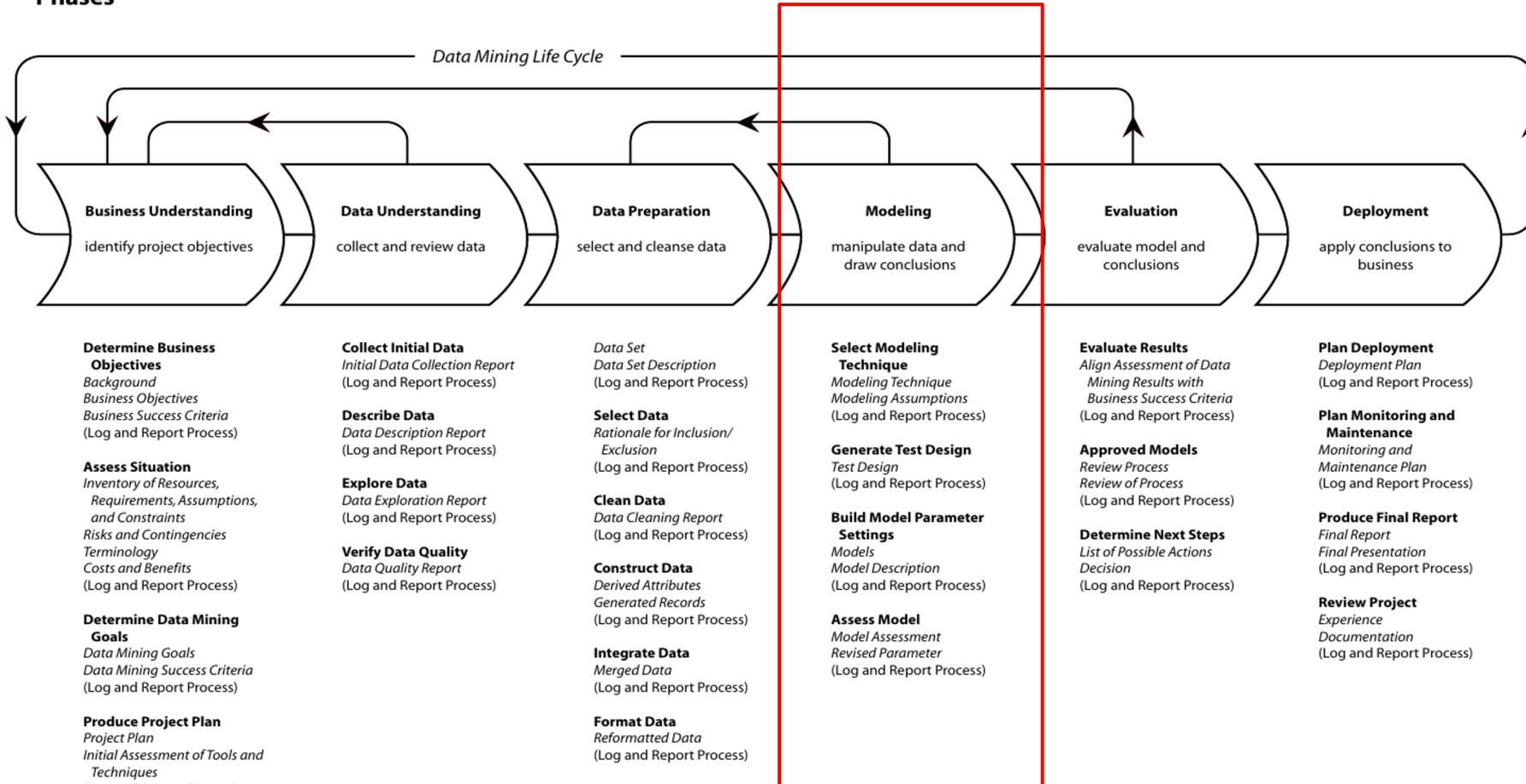


Dasar ANN, Perceptron, Back Propagation

Machine Learning

Phases



a visual guide to CRISP-DM methodology

Generic Tasks

Specialized Tasks
(Process Instances)

SOURCE CRISP-DM 1.0

<http://www.crisp-dm.org/download.htm>

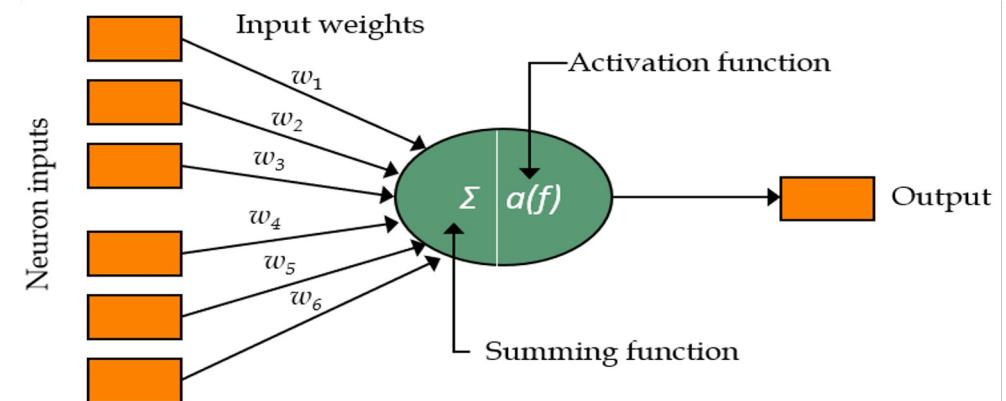
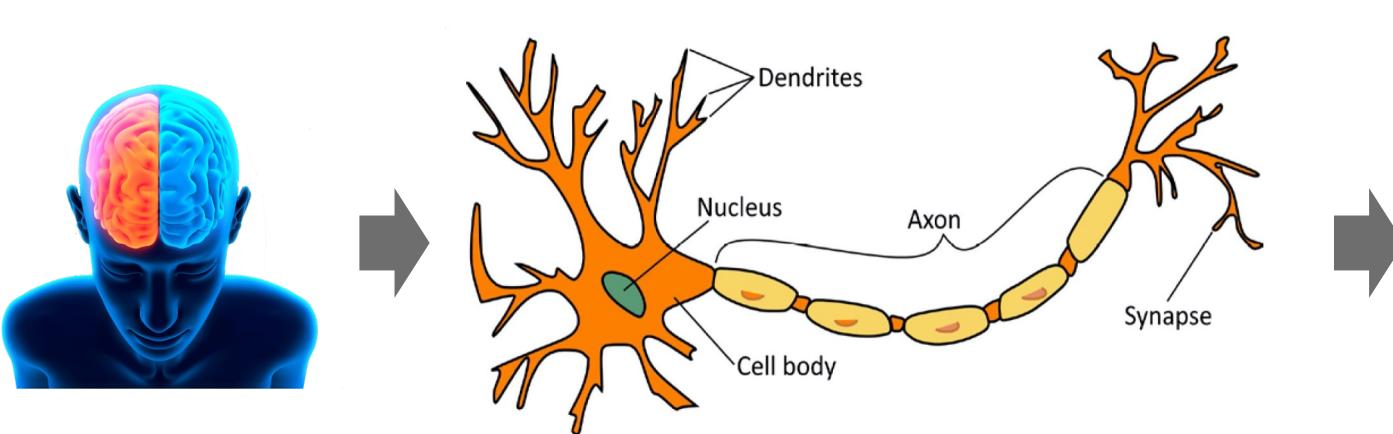
DESIGN Nicole Leaper

<http://www.nicoleleaper.com>



Artificial Neural Network (ANN)

- Salah satu metode mesin pembelajaran yang terinspirasi oleh cara kerja jaringan saraf biologis di otak manusia
- Merupakan jaringan dari unit pemroses kecil yang saling terhubung, yang dimodelkan berdasar sistem saraf manusia
- Konsep ANN bermula pada artikel dari Waffen McCulloch dan Walter Pitts pada tahun 1943 yaitu mencoba untuk memformulasikan model matematis sel-sel otak manusia



Jaringan saraf biologis	ANN
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight

Artificial Neural Network (ANN)

Model matematis dari ANN memiliki beberapa asumsi sebagai berikut:

- Neuron merupakan terjadinya pemrosesan informasi.
- Sinyal akan dikirimkan diantara neuron-neuron tersebut melalui penghubung dendrit dan akson.
- Di antara elemen-elemen tersebut memiliki penghubung. Setiap penghubung memiliki bobot yang akan ditambah atau dikurangi nilai sinyalnya.
- Terdapat fungsi aktivasi pada setiap neuron yang dikenakan pada jumlah semua inputnya untuk menentukan output.

Artificial Neural Network (ANN)

Suatu model ANN dipengaruhi oleh:

Arsitektur jaringan, yaitu arsitektur yang menentukan pola jaringan di antara neuron.

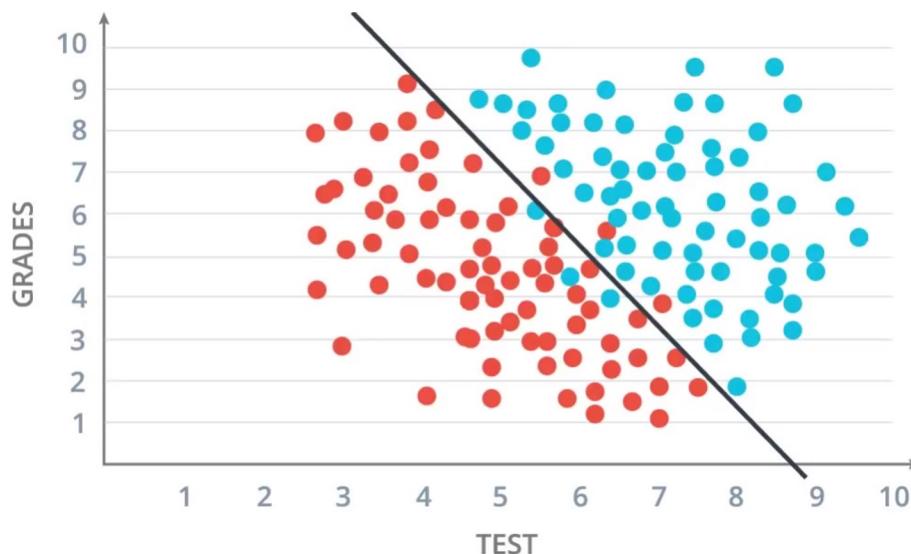
Metode pembelajaran, yaitu metode pembelajaran digunakan untuk melakukan pengadaptasian nilai-nilai yang menggambarkan koneksi bobot, menentukan dan mengubah nilai bobot.

Fungsi aktivasi, yaitu dapat berupa fungsi sigmoid dengan parameter tertentu, atau fungsi non linear.

Kelebihan Artificial Neural Network (ANN)

1. **Belajar adaptive (beradaptasi)**, yaitu mekanisme proses pembelajaran bagaimana melakukan pekerjaan berdasarkan data yang diberikan dalam proses pelatihan atau pengalaman sebelumnya.
1. **Self-organization**, yaitu kemampuan jaringan saraf tiruan untuk dapat belajar dan membuat organisasi sendiri dan melakukan representasi dari informasi yang diterimanya selama proses belajar.
1. **Real time operation**, yaitu kemampuan jaringan saraf tiruan yang dilakukan secara parallel, sehingga dapat menggunakan perangkat keras yang dirancang dan diproduksi khusus untuk dapat mengambil keuntungan dari proses ini.
1. ANN dapat berfungsi sebagai Universal Function Approximator

Kasus dari Sudut Pandang LR



Diketahui Persamaan Garis

$$2x_1 + x_2 - 18 = 0$$

dimana x_1 = Test

:

x_2 = Grade

Apakah Siswa 3 diterima?



Siswa 3

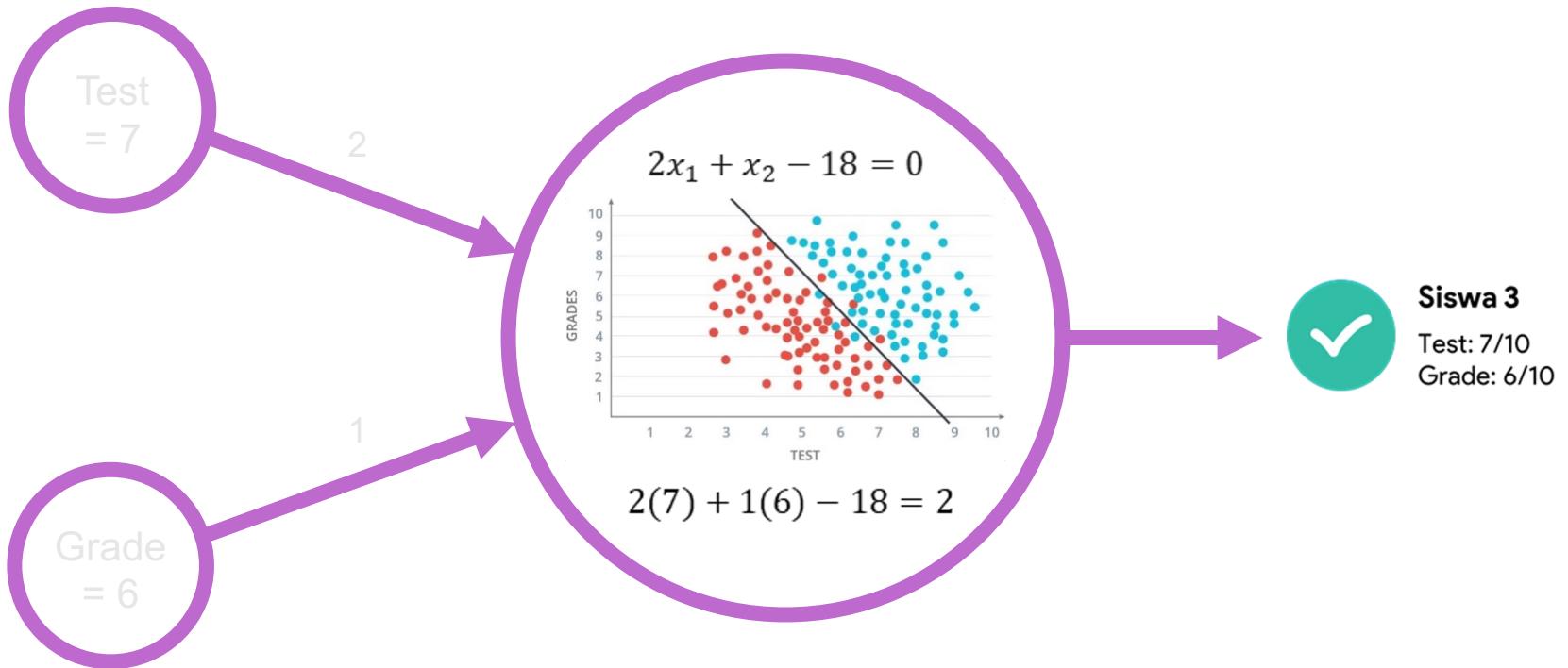
Test: 7/10

Grade: 6/10

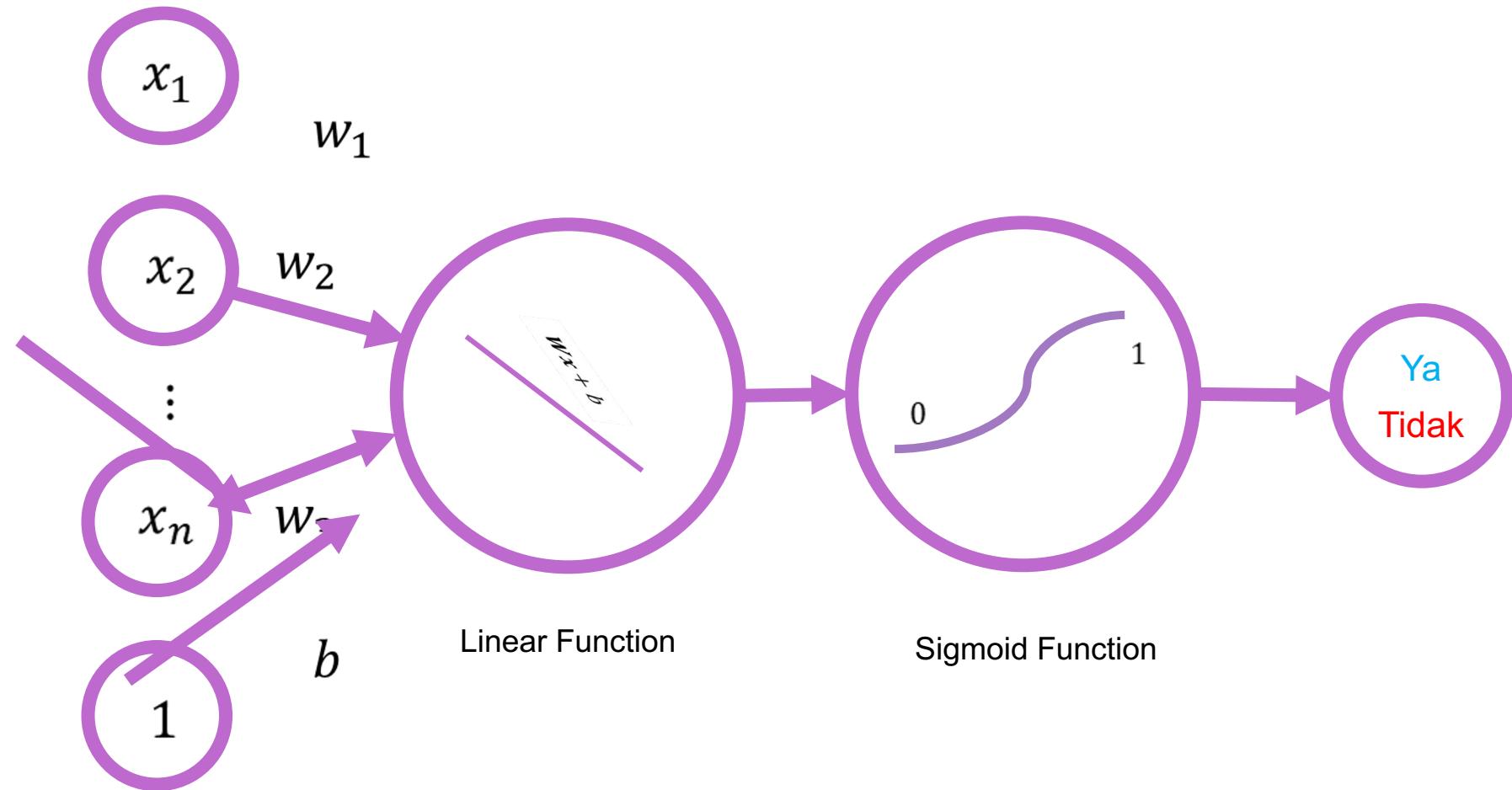
Prediksi

$$\hat{y} = \begin{cases} 1, & 2x_1 + x_2 - 18 \geq 0 \\ 0, & 2x_1 + x_2 - 18 < 0 \end{cases}$$

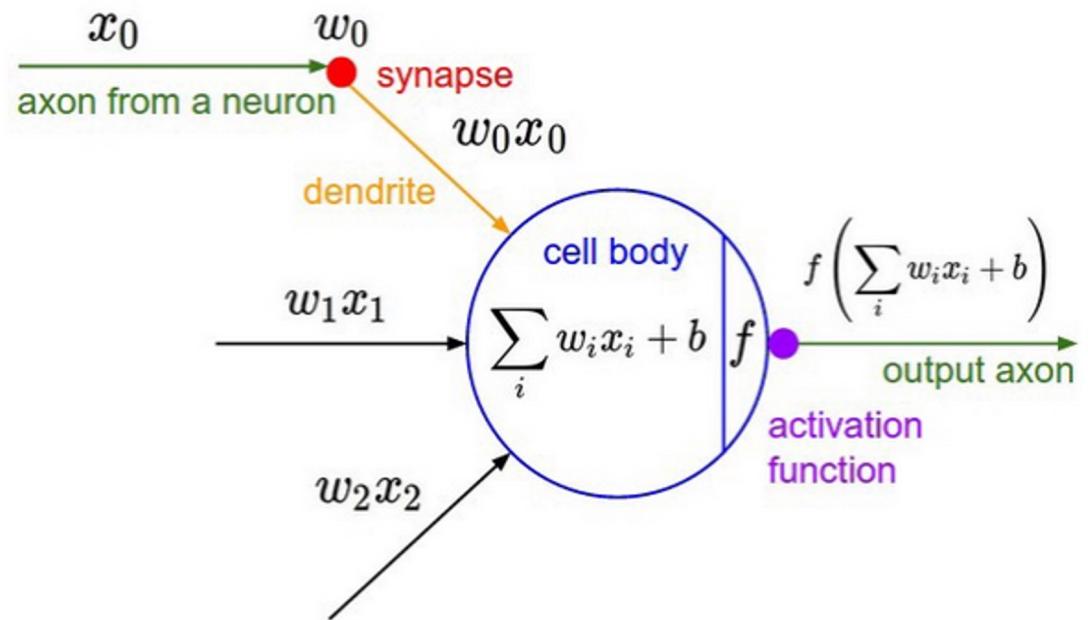
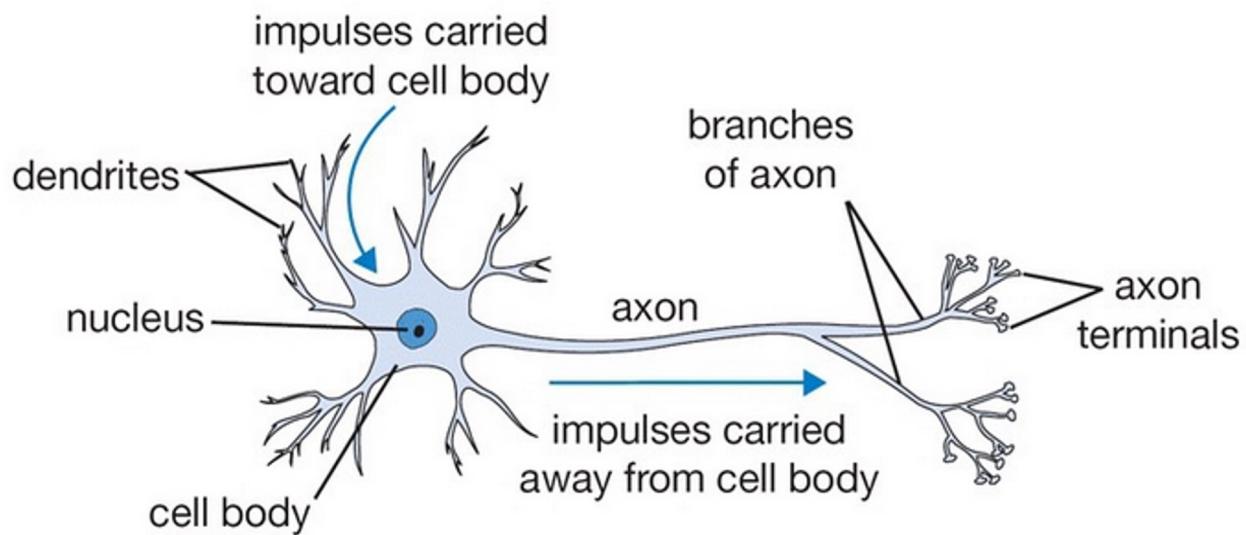
Neuron



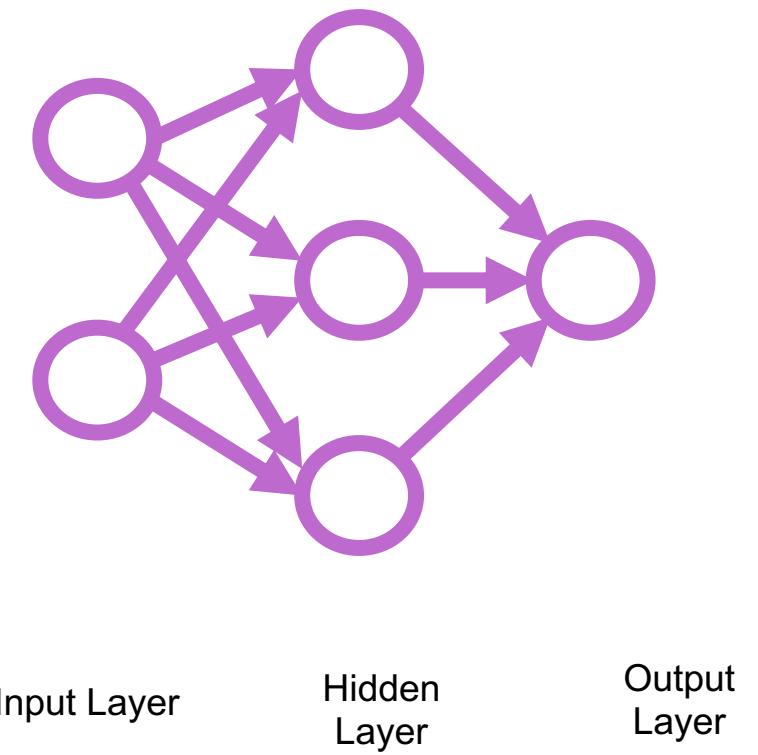
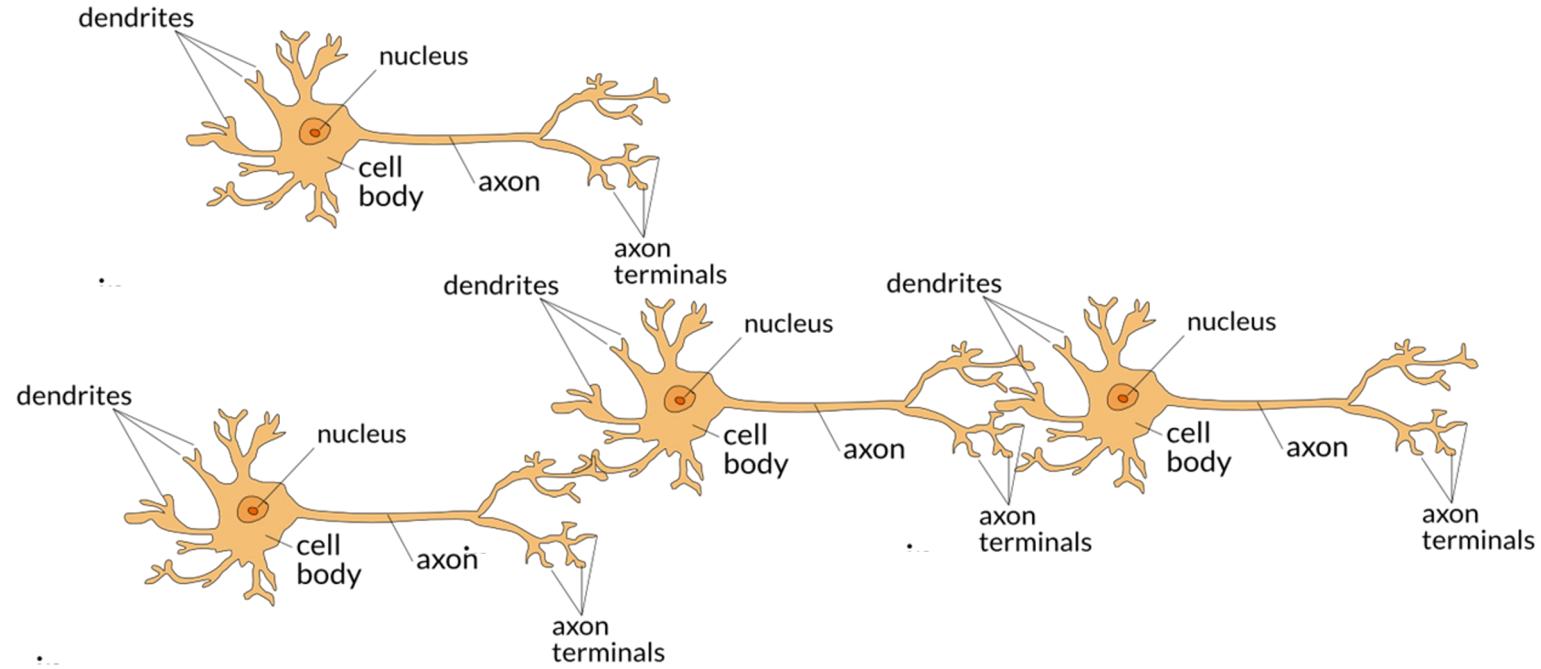
Neuron



Neuron



Neuron

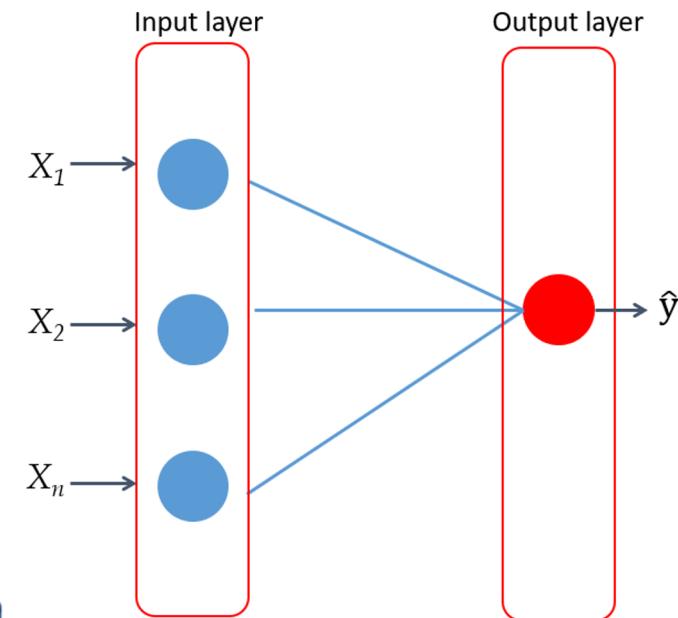
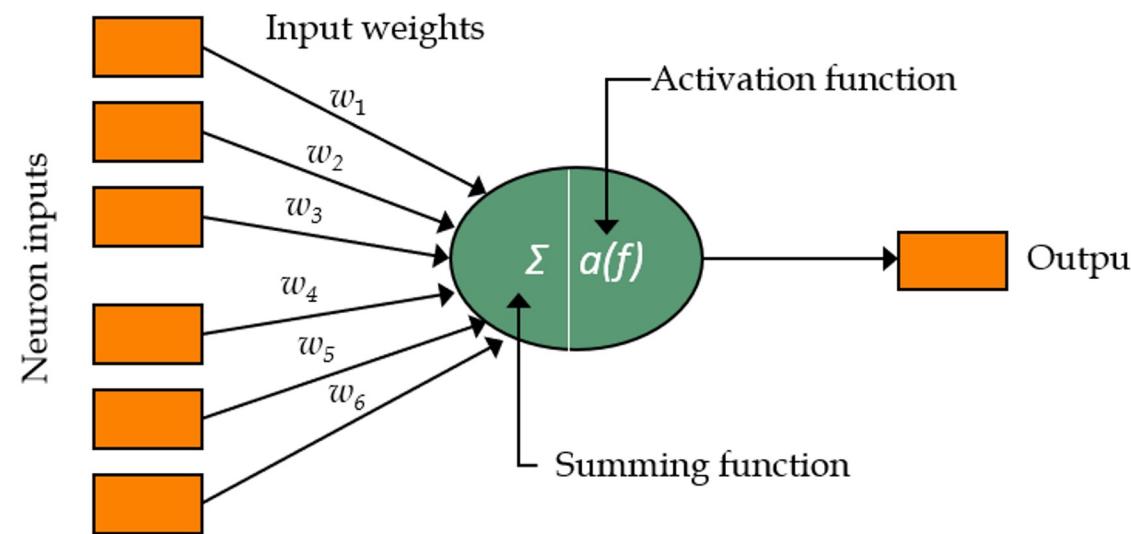


Arsitektur Single-layer Perceptron

Arsitektur single-layer ANN hanya terdiri dari input layer dan output layer

Unit pemrosesan informasi pada ANN sebagai berikut:

- Satu set link berupa neuron dan bobot w
- Fungsi penambah (penggabung linear) untuk mengitung jumlah perkalian bobot terhadap input X
- Fungsi aktivasi $a(\cdot)$

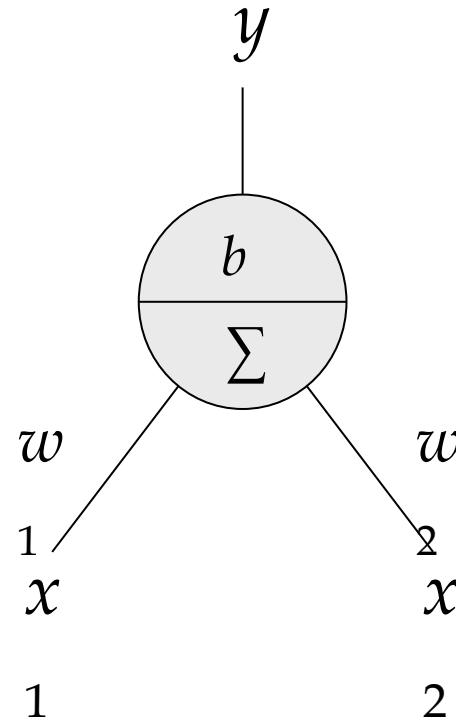


$$f = \sum_{i=1}^m w_i x_i + b$$
$$y = a(f)$$

Apa yang bisa dilakukan sebuah Neuron ?

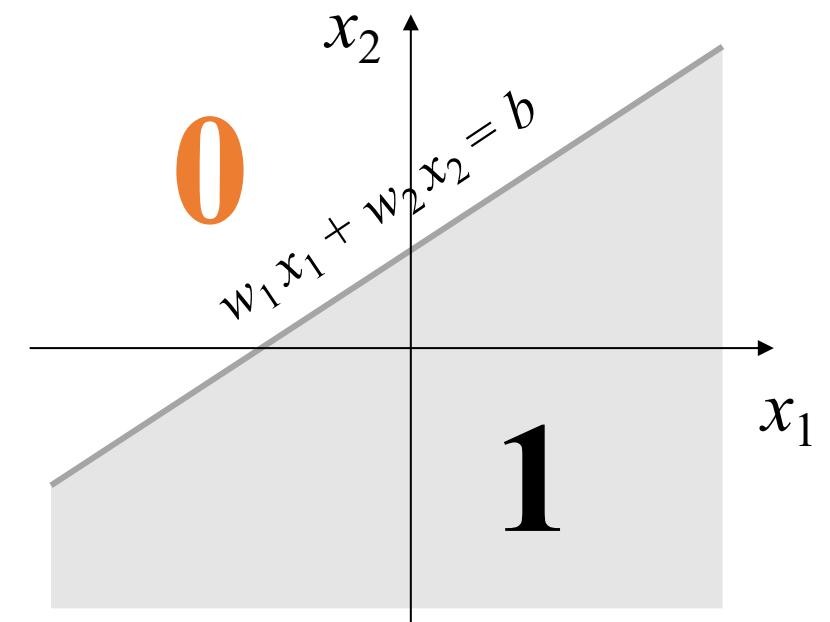
Sebuah neuron pada ANN dapat menyelesaikan permasalahan klasifikasi biner

- Sebagai fungsi pemisah (*hyperspace separation*)
- Sebagai *binary threshold*



$$f(x) = w_1x_1 + w_2x_2 - b$$

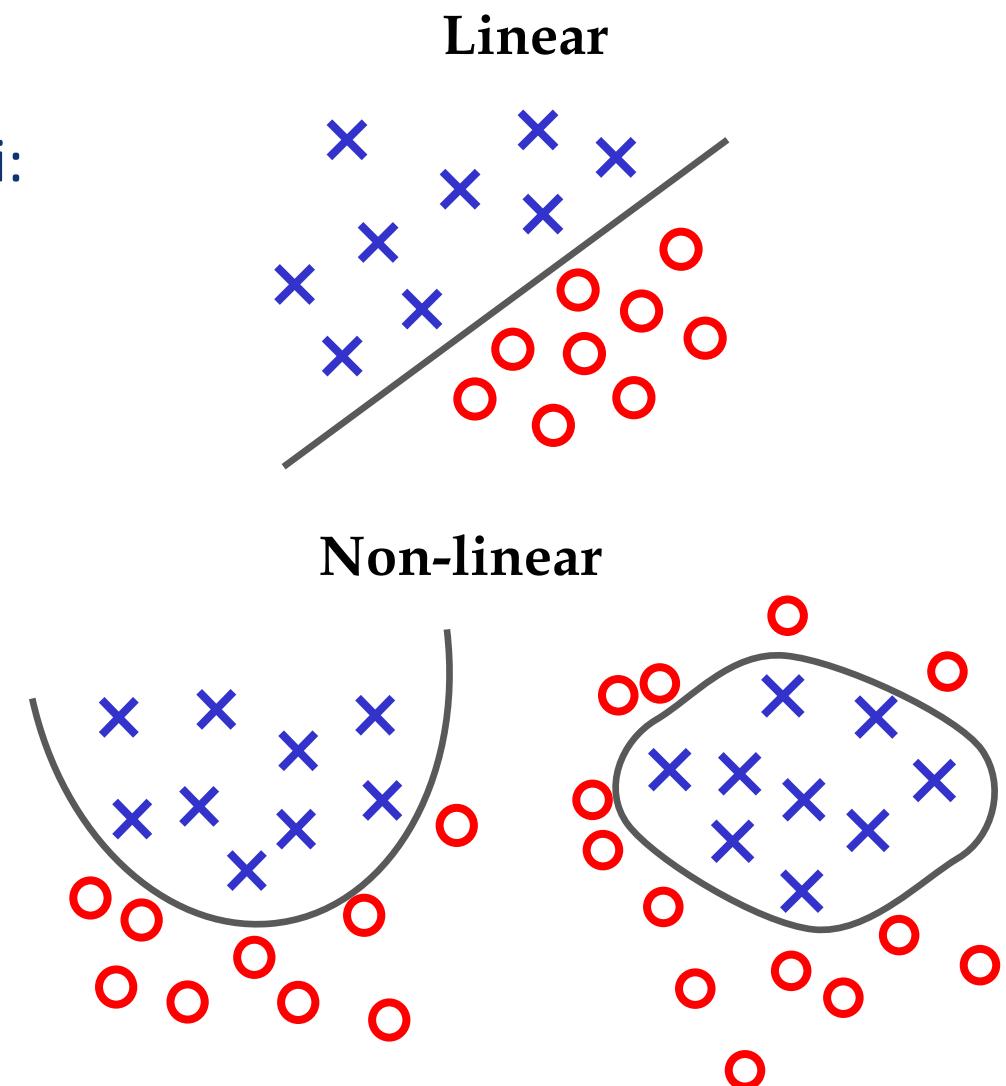
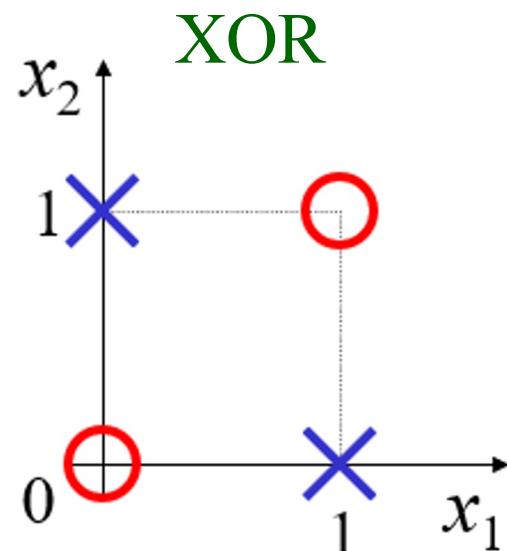
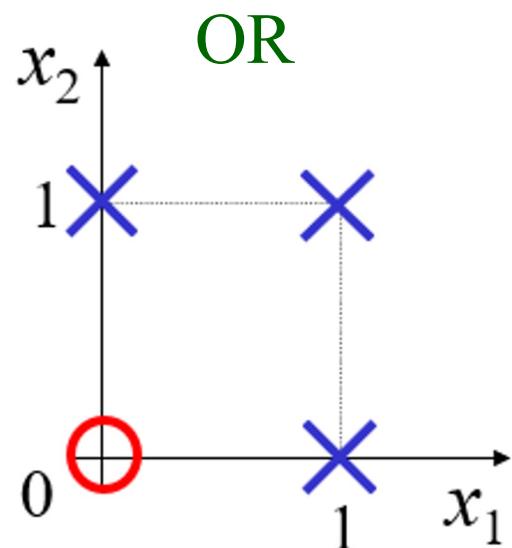
$$y = \begin{cases} 1 & f(x) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



Permasalahan Linear dan Non-Linear

Permasalahan klasifikasi dapat dikategorikan sebagai:

- Permasalahan Linear, misalnya fungsi OR dan AND
- Permasalahan Non-Linear, misalnya fungsi XOR



Fungsi Aktivasi

- Fungsi aktivasi merubah neuron menjadi non-linear
- Beberapa contoh fungsi aktivasi yang umum digunakan pada metode ANN sebagai berikut

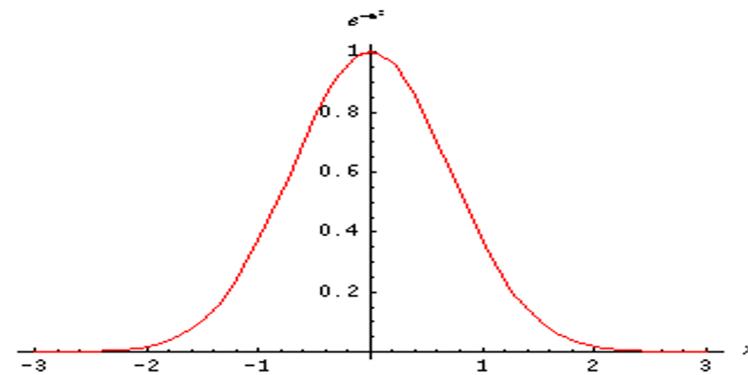
Sigmoid function

$$a(f) = \frac{1}{1 + \exp(-f)}$$



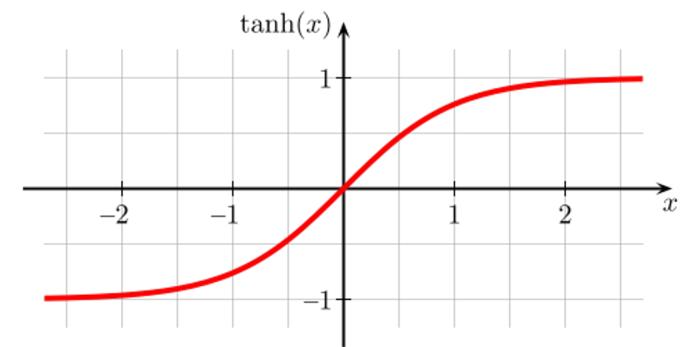
Gaussian function

$$a(f) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left(\frac{f-\mu}{\sigma}\right)^2\right)$$

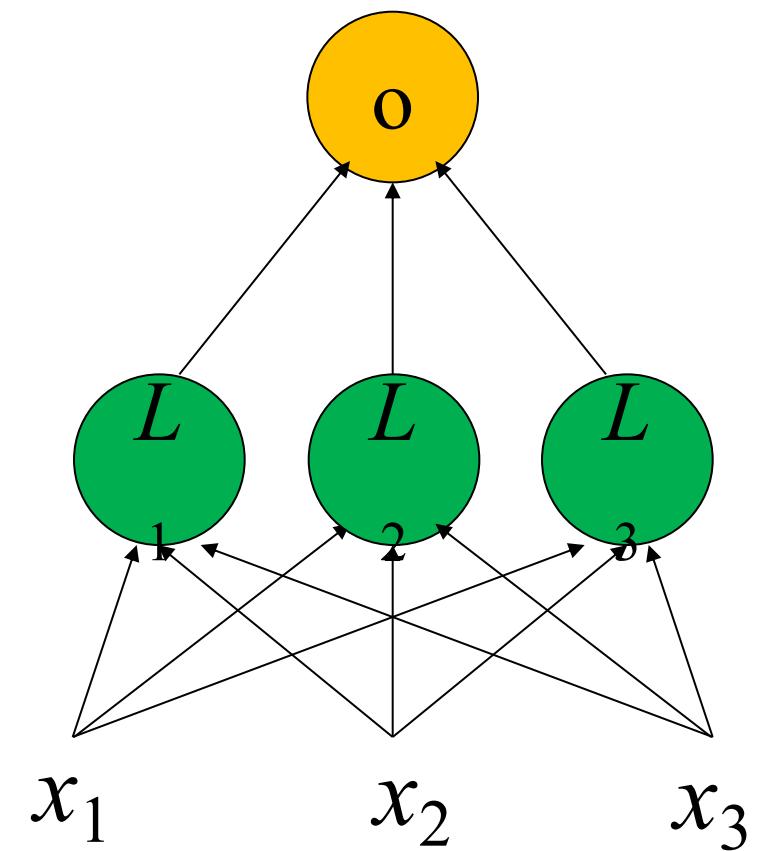
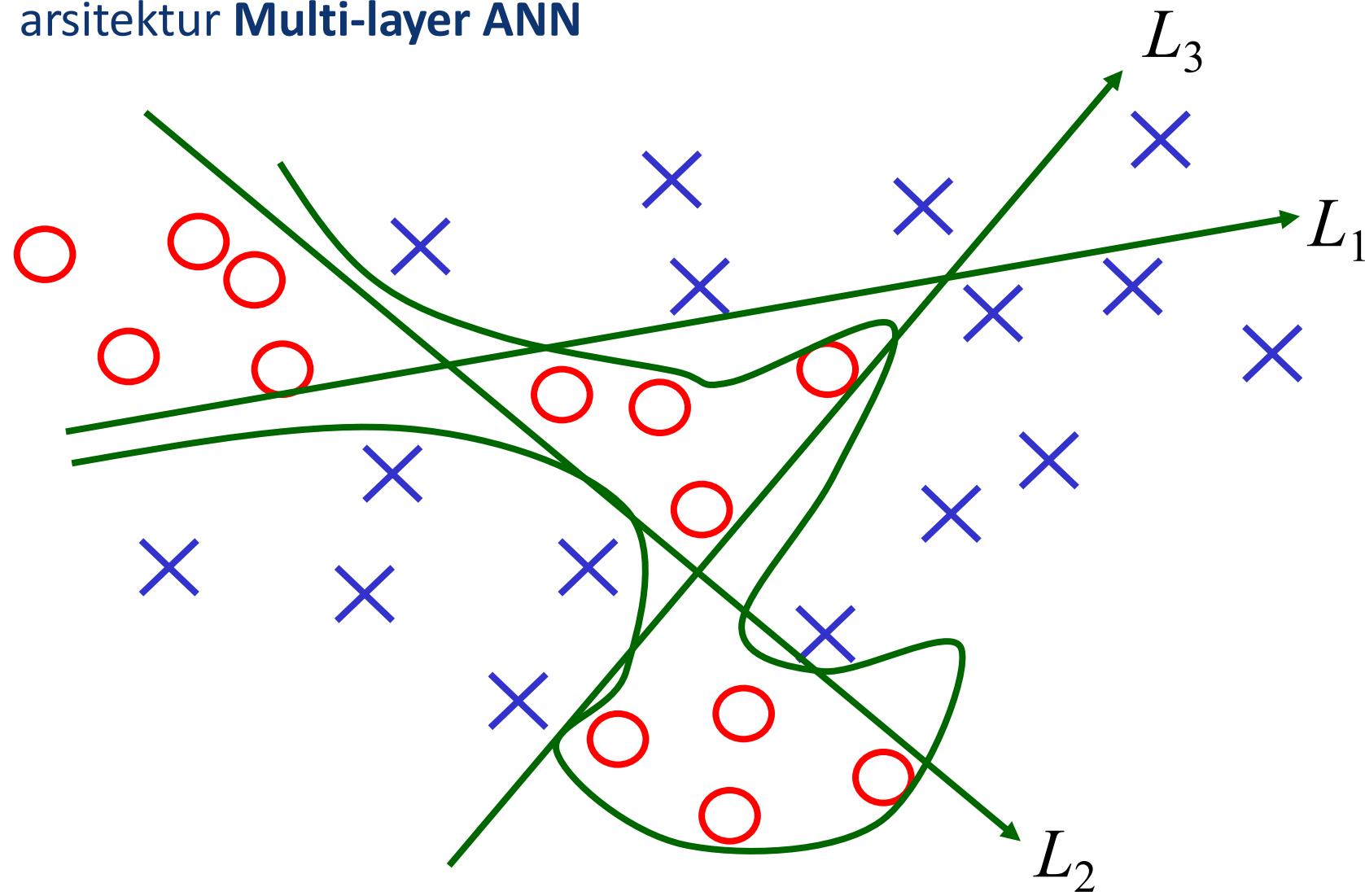


Tangent Hyperbolic function

$$\tanh(x) = 2\sigma(2x) - 1$$



Pada permasalahan non-linear dan permasalahan yang lebih kompleks, menggunakan arsitektur **Multi-layer ANN**



Linier Basis Function (LBF) vs Radial Basis Function (RBF)

Linier Basis Function (LBF)

Diberikan satu set input dataset dari N sampel $\{X_n\}$, di mana $n = 1, \dots, N$, serta nilai target yang sesuai $\{t_n\}$, tujuannya adalah untuk menyimpulkan nilai t untuk nilai baru x . Kumpulan-kumpulan data input bersama dengan nilai target yang sesuai t dikenal sebagai kumpulan data pelatihan.

$$y(x) = t$$

Model dasar dari LBF adalah untuk regresi dimana sebuah model terdiri dari sebuah linier kombinasi variable input:

$$y(w, x) = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_D x_D$$

$$\text{where } x = (x_1, x_2, \dots, x_D)^T$$

Jika kita berasumsi bahwa fungsi non-linier dari variabel input adalah (x) , maka kita dapat menulis ulang fungsi aslinya sebagai:

$$y(x, w) = w_0 + w_1 \varphi(x_1) + w_2 \varphi(x_2) + \dots + w_D \varphi(x_D)$$

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(x)$$

$\varphi(x)$ are known as *basis functions*

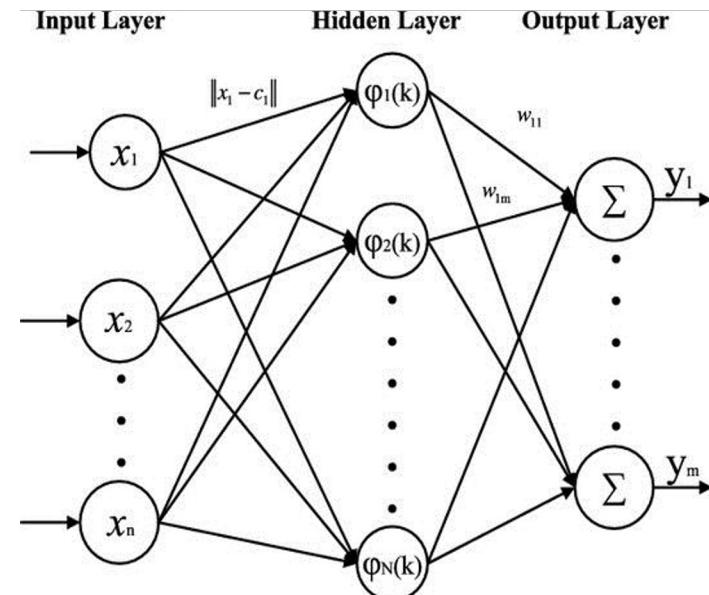
Linier Basis Function (LBF) vs Radial Basis Function (RBF)

Radial Basis Function (RBF)

RBF Radial Basis Function (RBF) networks adalah model linear dimana fungsi basis berupa radial basis function, yaitu fungsi yang tergantung pada jarak antara argumennya, yaitu $\phi(\|x - x_n\|)$, sehingga modelnya berbentuk:

$$y(x) = \sum_{n=1}^N w_n \phi(\|x - x_n\|)$$

RBF memiliki 2 pelatihan, tahap awal, parameter fungsi basis ditentukan secara cepat dengan menggunakan unsupervised method yang hanya memerlukan data input saja. Tahap kedua pelatihan ini adalah membawa hasil dari unit tersembunyi ke unit output secara linier. Fungsi basis radial merupakan fungsi yang bergantung pada jarak antara data dengan suatu pusat data. Fungsi basis radial yang digunakan umumnya nonlinier, dan Fungsi basis yang biasa digunakan adalah fungsi Gaussian.

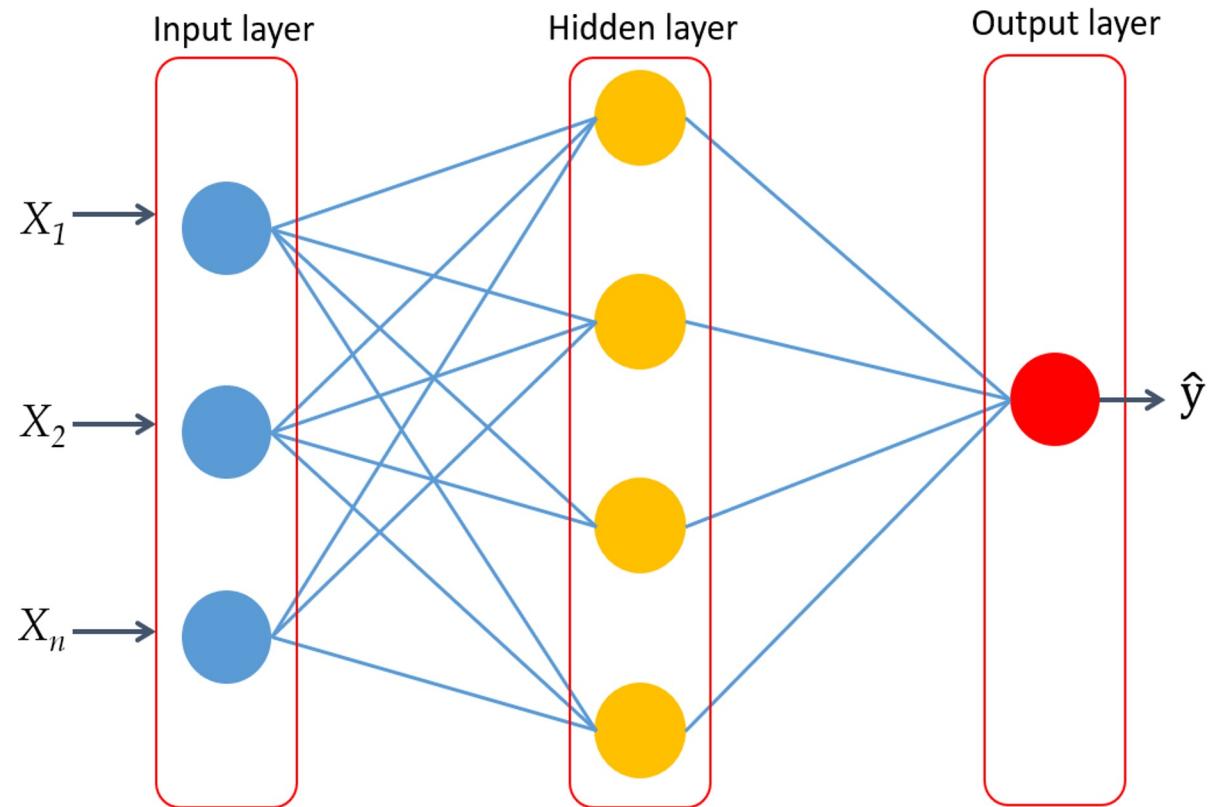


Multi-layer ANN

- Multilayer perceptron (MLP) adalah jaringan saraf tiruan feedforward yang menghasilkan serangkaian output dari serangkaian input. Ciri dari MLP adalah terdiri dari beberapa lapisan node input yang terhubung sebagai grafik terarah antara layer input dan output.
- MLP menggunakan backpropagation untuk melatih jaringan. MLP dapat dikatakan sebagai jaringan yang mendalam (deep learning).
- Multi layer perceptron memiliki sejumlah neuron atau saraf yang saling terhubung dengan neuron lainnya dengan neuron bobot penghubung. Dimana setiap neuron yang ada, merupakan sebuah unit yang memiliki tugas untuk memproses dan menghitung nilai aktivasi, mulai dari input hingga output, atau dari unit satu ke unit lainnya.

Arsitektur Multi-layer ANN

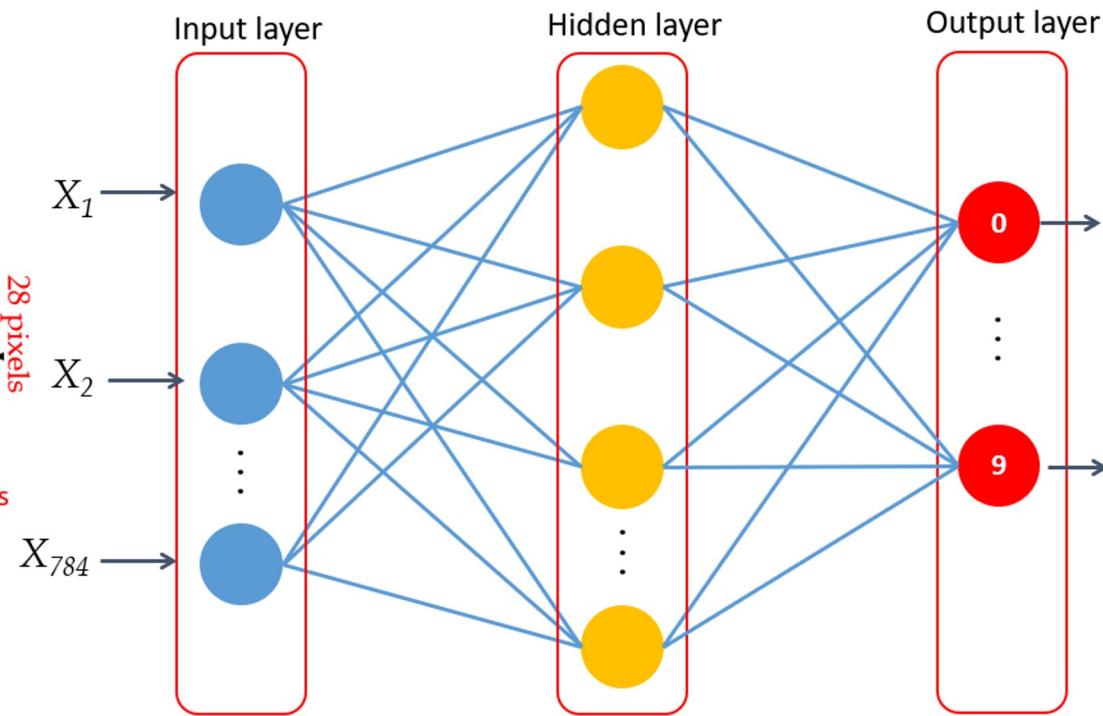
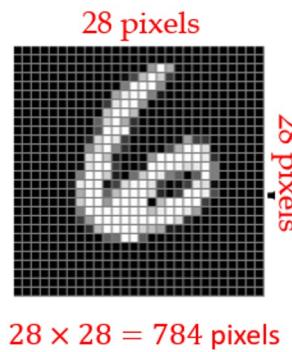
- Terdiri dari tiga layer yaitu:
 - input layer
 - hidden layer
 - output layer
- Hubungan antar neuron pada ANN merupakan ***fully connected network (FCN)***
- **Jumlah *hidden layer*** sebaiknya disesuaikan dengan kompleksitas permasalahan
- **Jumlah neuron pada *hidden layer*** umumnya lebih banyak daripada jumlah *neuron di output layer*



Desain arsitektur ANN

Penentuan jumlah *neuron* pada *input layer*

- Jumlah neuron sesuai dengan jumlah fitur pada data input



Penentuan jumlah *neuron* pada *output layer*

- Jumlah neuron sesuai dengan permasalahan
- Pada permasalahan klasifikasi biner dan regresi bisa menggunakan hanya satu *neuron*
- Pada permasalahan klasifikasi *multiclass* menggunakan jumlah *neuron* sesuai jumlah label kelasnya, misalnya: 10 neuron pada pengenalan angka

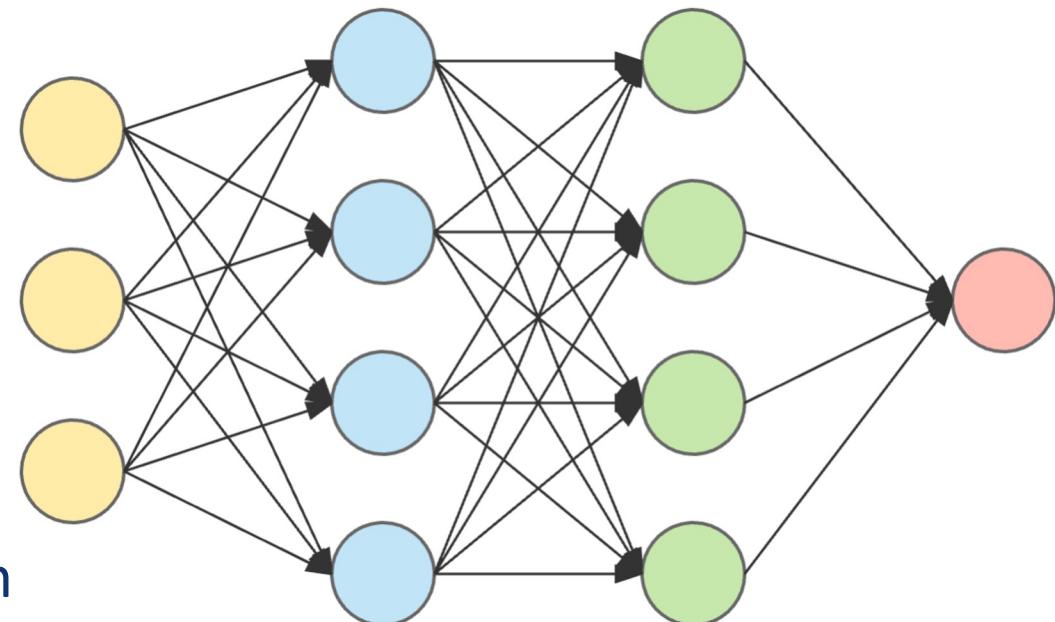
Desain arsitektur ANN

Penentuan jumlah *hidden layer*:

- Semakin banyak jumlah layer memerlukan komputasi waktu lebih lama
- Jumlah layer sebaiknya disesuaikan dengan kompleksitas permasalahan

Penentuan jumlah node (neuron) pada *hidden layer*:

- Semakin banyak jumlah node memungkinkan mempelajari pola yang lebih rumit
- Untuk mencegah *overfitting* sebaiknya menambah jumlah node secara bertahap



Desain arsitektur ANN

<https://playground.tensorflow.org/>

Epoch 000,000 Learning rate 0.03 Activation Tanh Regularization None Regularization rate 0 Problem type Classification

DATA

Which dataset do you want to use?

Which properties do you want to feed in?

Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

FEATURES

+ - 2 HIDDEN LAYERS

+ - 4 neurons

X₁ X₂ X₁₂ X₂₂ X_{1X2} sin(X₁)

The outputs are mixed with varying weights, shown by the thickness of the lines.

This is the output from one neuron. Hover to see it larger.

OUTPUT

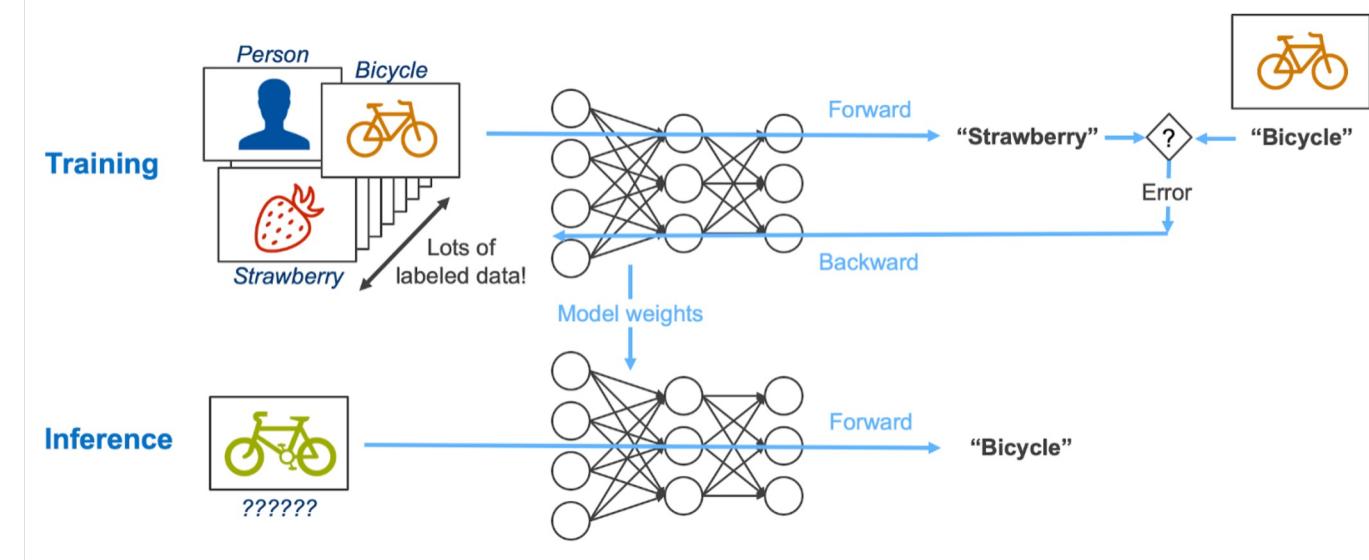
Test loss 0.519
Training loss 0.501

Colors shows

The screenshot displays the TensorFlow Playground web application. At the top, there are navigation icons (back, forward, search, etc.) and a URL bar with the address https://playground.tensorflow.org/. Below the URL bar, various model configuration parameters are set: Epoch at 000,000, Learning rate at 0.03, Activation function as Tanh, and Regularization set to None. The problem type is specified as Classification. In the middle section, under 'DATA', there are two dropdown menus for selecting datasets and features. A slider indicates a 50% ratio of training to test data. Below these are sliders for Noise and Batch size. Under 'FEATURES', a neural network diagram is shown with two hidden layers containing 4 and 2 neurons respectively. The diagram illustrates how multiple inputs (X1, X2, X12, X22, X1X2, sin(X1)) are processed through these neurons using varying weights. An annotation points to one neuron's output. To the right, a scatter plot shows the classification results with axes ranging from -6 to 6. The plot shows two distinct clusters of points, colored blue and orange, separated by a decision boundary. Loss values are displayed as 0.519 for test and 0.501 for training. At the bottom, there is a 'REGENERATE' button.

Mekanisme Pembelajaran (*Learning*)

1. Training: learning pada saat pembentukan model.
2. Inferensi: learning saat menggunakan model NN.



Mekanisme Pengujian Pembelajaran *(Learning)*

1. Stability: pengujian mekanisme learning dengan menguji akurasi terkait prediksi data yang pernah dilatih sebelumnya.
2. Plasticity: pengujian mekanisme learning dengan menguji akurasi terkait prediksi dari sebuah sistem terkait data baru yang belum pernah dilakukan.

Tahapan ANN: *Feed Forward*

1. Masukkan vektor X ke *input layer*
2. Hitung output setiap neuron di *hidden layer* dan *output layer*

$$o_j = \sum_{i=1}^n w_{ji}x_i + b \quad o_k = \sum_{j=1}^m w_{kj}o_j + b$$

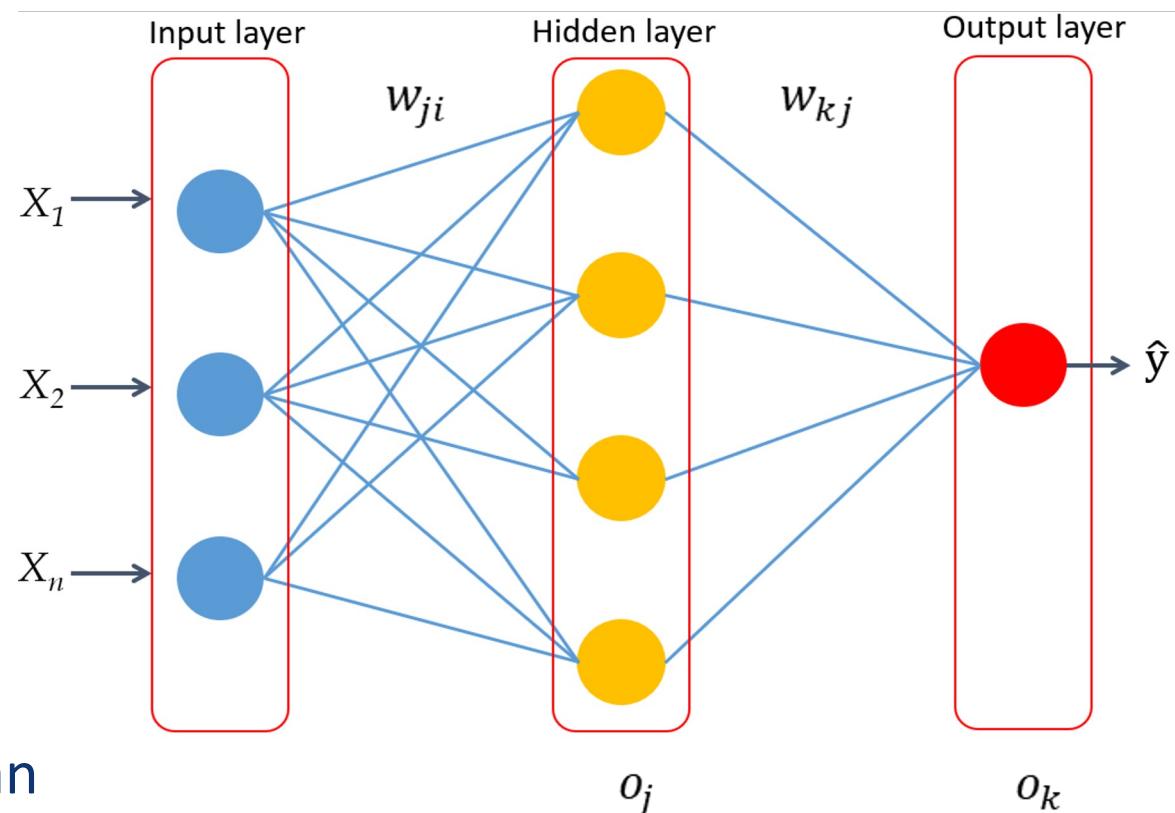
dimana x_i dan o_j adalah matrik input dan output neuron pada layer sebelumnya

w_{ji} dan w_{kj} adalah bobot yang menghubungkan antara neuron pada dua layer *berbeda*, dan b adalah

bias

n dan m adalah jumlah neuron di layer sebelumnya
 $y = a(o_k)$

3. Hitung fungsi aktivasi pada layer output



Tahapan ANN: Pembelajaran (*Learning*)

1. Inisialisasi bobot W
1. Update bobot sehingga output dari ANN adalah konsisten dengan label kelas pada data latih
 - a. Fungsi obyektif (fungsi loss):
$$J(W) = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - f(x^{(i)}; W))^2$$

y – target
 f – prediksi
 - b. Menemukan bobot baru dengan meminimalkan fungsi obyektif, contoh:
algoritma *backpropagation*

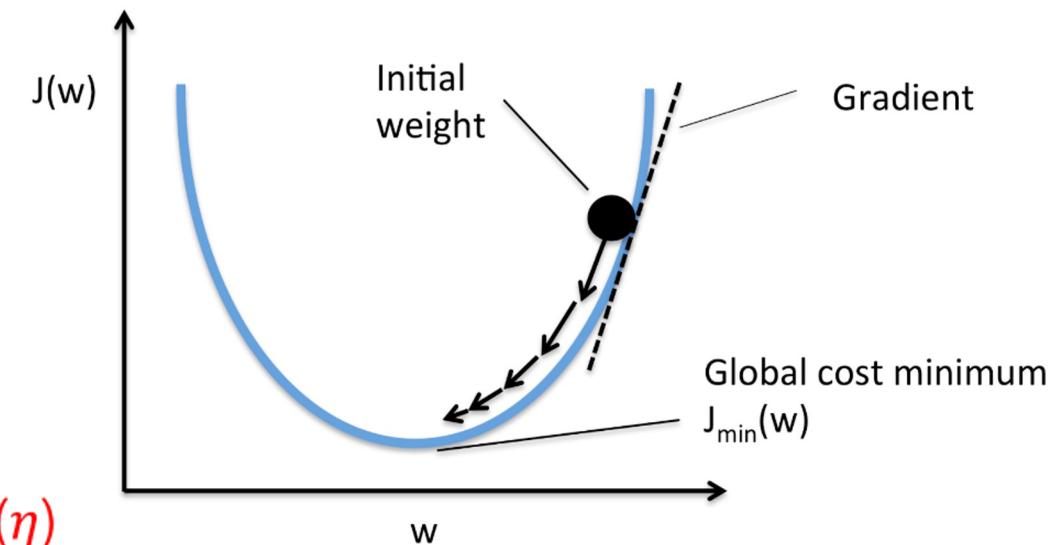
Algoritma Pembelajaran (*Learning*)

Untuk merancang algoritma pembelajaran, ada beberapa hal yang perlu diperhatikan:

1. Kriteria Iterasi berhenti? **konvergen, iterasi (epoch)**
2. Bagaimana direction? **gradient descent**
3. Berapa banyak (step) yang diperlukan? **nilai learning rate (η)**

Algoritma Backpropagation Gradient Descent

- Inisialisasi bobot secara random
- Melakukan iterasi sampai konvergen atau maksimum iterasi
 - Hitung gradient, $\frac{\partial J(W)}{\partial w}$
 - Update bobot, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial w}$
- Mengembalikan nilai bobot



$$W^* = \operatorname{argmin}_W J(W)$$

$$J(W) = \frac{1}{n} \sum_{i=1}^n \left(y^{(i)} - f(x^{(i)}; W) \right)^2$$

Nilai *learning rate* jika terlalu kecil memerlukan waktu lebih lama untuk konvergen, jika terlalu besar membuat model tidak stabil

Algoritma Pembelajaran (*Learning*)

Algoritma Stochastic Gradient Descent

- Inisialisasi bobot secara random
- Melakukan iterasi sampai konvergen atau maksimum iterasi
 - Baca **setiap** data poin i
 - Hitung gradient, $\frac{\partial J_i(W)}{\partial W}$
 - Update bobot, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
- Mengembalikan nilai bobot

- Inisialisasi bobot secara random
- Melakukan iterasi sampai konvergen atau maksimum iterasi
 - Baca **batch B** data poin
 - Hitung gradient,
$$\frac{\partial J(W)}{\partial W} = \frac{1}{B} \sum_{k=1}^B \frac{\partial J_k(W)}{\partial W}$$
 - Update bobot, $W \leftarrow W - \eta \frac{\partial J(W)}{\partial W}$
 - Mengembalikan nilai bobot

Tahapan Pembelajaran Multi-layer Perceptron ANN

Langkah 0 – Inisialisasi bobot, learning rate, maksimum iterasi

Langkah 1 – Membaca vektor input X

Langkah 2 – Lakukan iterasi (*epoch*)

Langkah 3 – Hitung luaran neuron di hidden layer dan output layer

Langkah 4 – Hitung *back propagate error* (pada output layer dan hidden layer)

Langkah 5 – Perbarui semua bobot (pada output layer dan hidden layer)

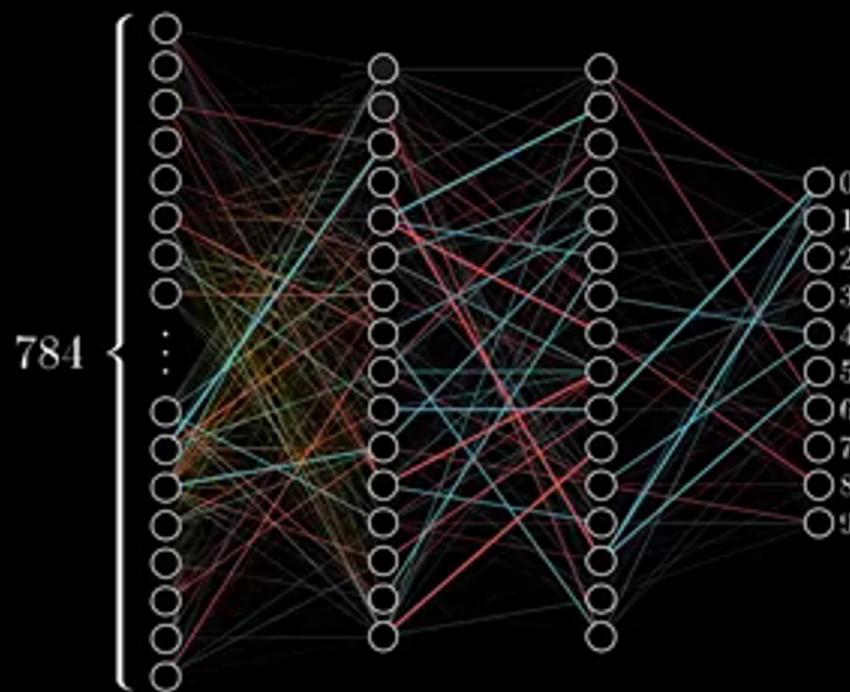
Langkah 6 – Ulangi langkah 3 – 5 hingga bobot konvergen atau maksimum iterasi

Langkah 7 – Luaran berupa matrik bobot (pada output layer dan hidden layer)

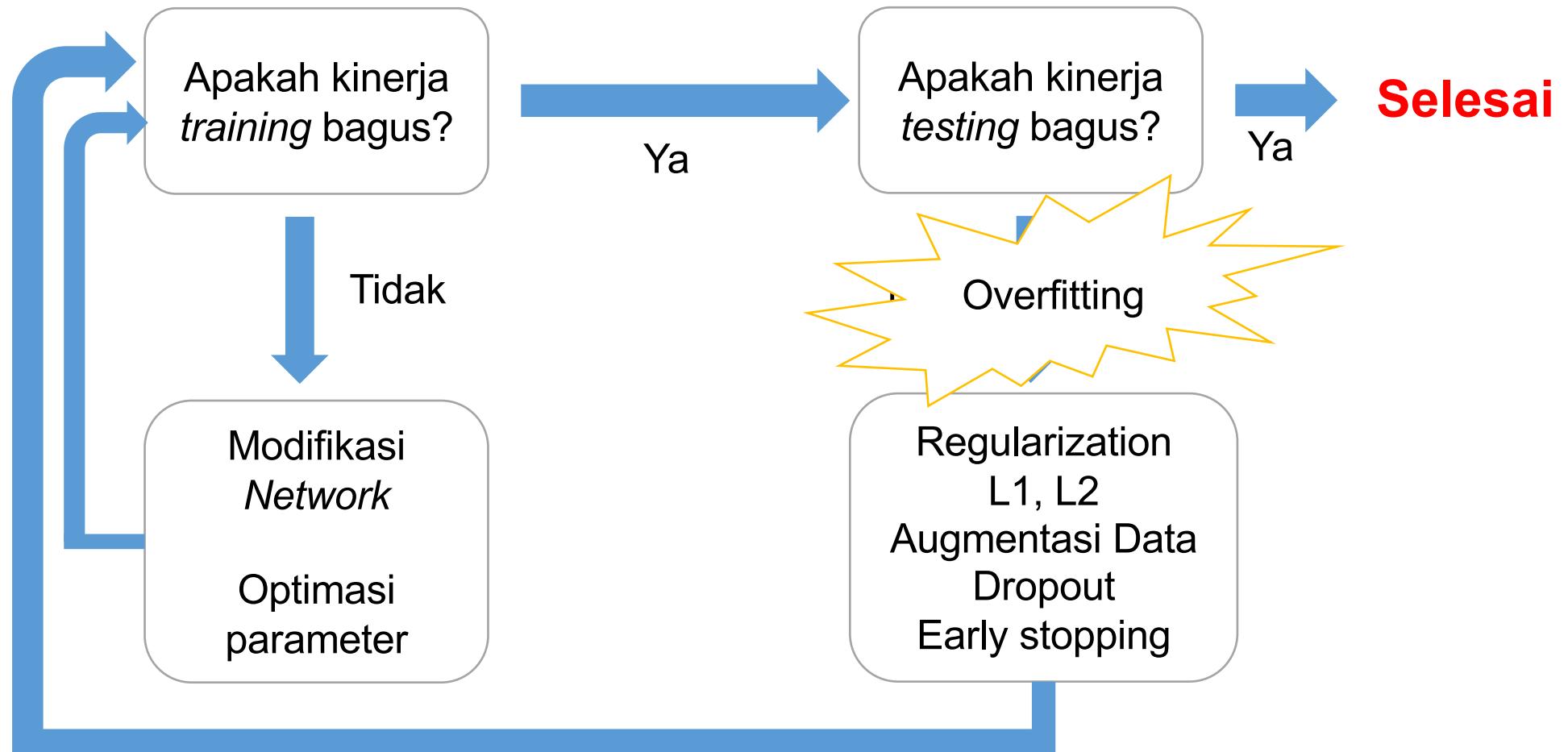
<https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/>

Contoh Proses Pembelajaran ANN

Training in
progress. . .

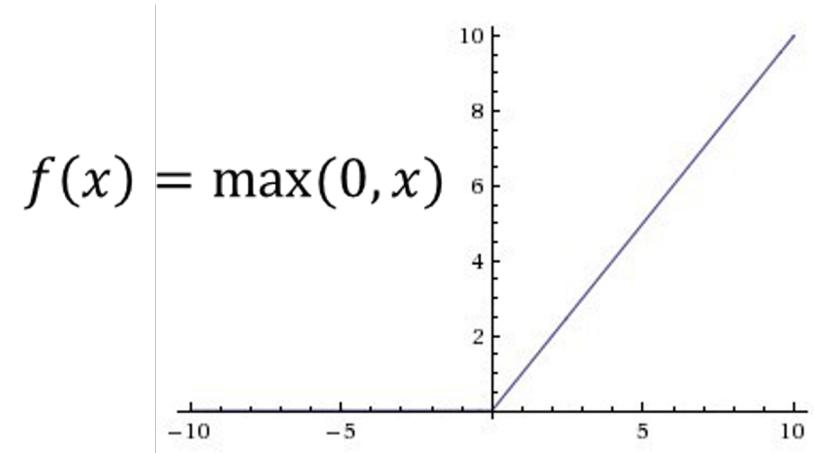


Strategi Proses Pembelajaran



Modifikasi Network

- Merubah arsitektur, misalnya menambah jumlah hidden layer, jumlah neuron, atau jenis arsitektur lain
- Merubah fungsi aktivasi, misalnya menggunakan ReLU



Rectified Linear Unit function (ReLU)

Optimasi parameter

Nilai *learning rate* berpengaruh pada perhitungan bobot baru, umumnya penggunaan *learning rate* yang menyesuaikan nilai gradien (*adaptive learning rate*) menunjukkan kinerja model yang lebih baik. Contoh *algoritma adaptive learning rate*:

- Adagrad [John Duchi, JMLR 2011]
- Adadelta [Matthew D. Zeiler, arXiv 2012]
- Adam [Diederik P. Kingma, ICLR 2015]
- AdaSecant [Caglar Gulcehre, arXiv 2014]
- RMSprop <https://www.youtube.com/watch?v=O3sxAc4hxZU>

Mencegah *Overfitting*

Regularization

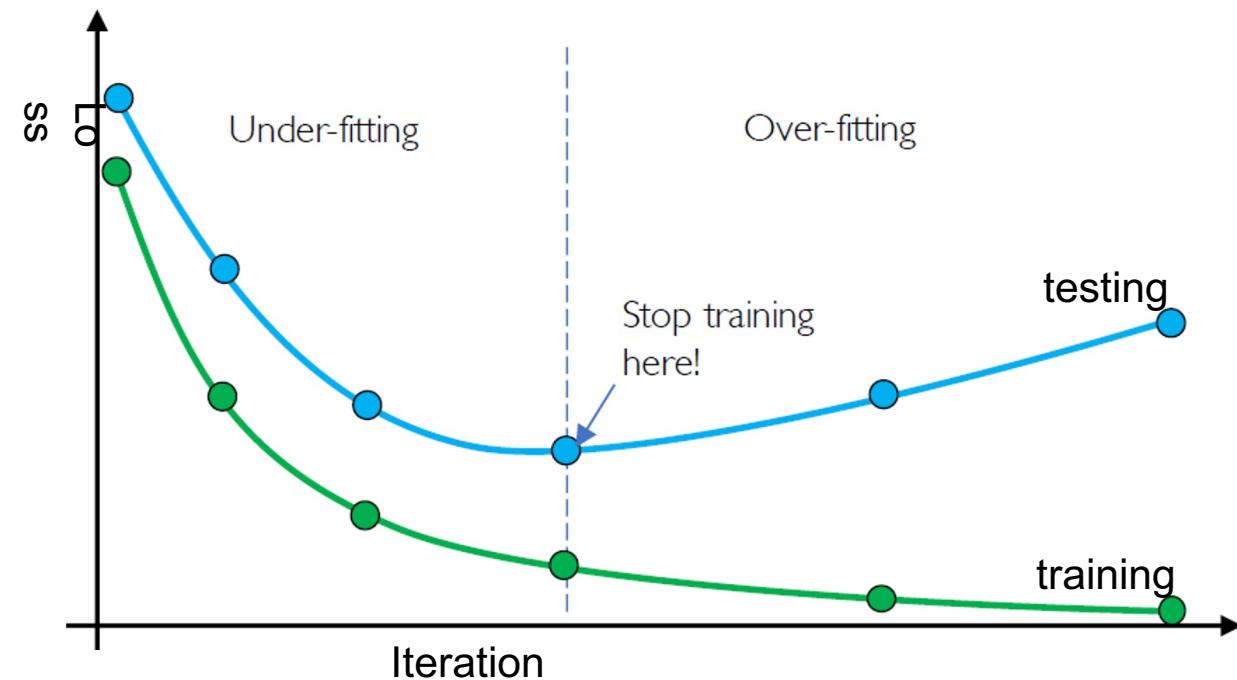
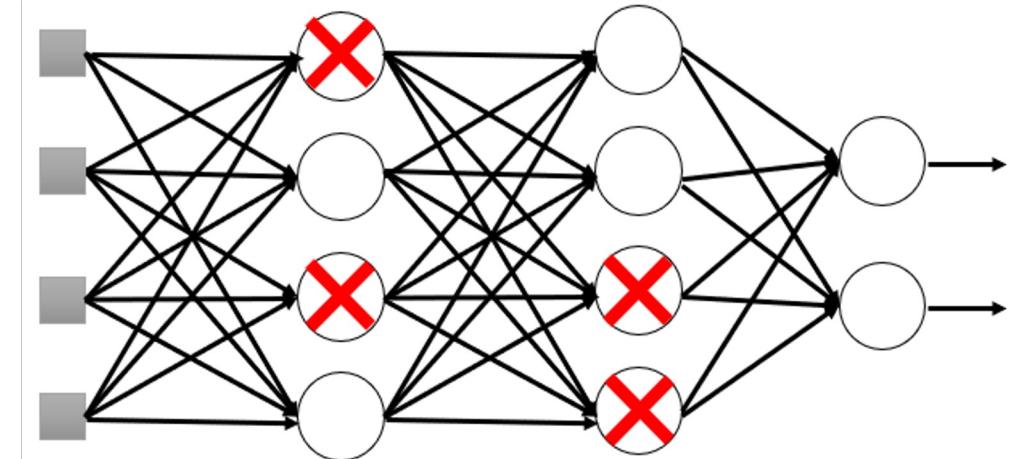
Regularisasi dilakukan untuk mengurangi *generalization error* dengan mencegah model lebih kompleks. Penerapan regularisasi dengan cara menambahkan *regularization term* pada semua parameter (bobot) ke fungsi obyektif.

- **Regularization L1 norm**
 - Menambahkan *sum of the absolute weights* sebagai *penalty term* ke fungsi obyektif
- **Regularization L2 norm (*weight decay*)**
 - Menambahkan *sum of the squared weights* sebagai *penalty term* ke fungsi obyektif

Mencegah *Overfitting*

Cara meregulasi parameter untuk menghindari *overfitting* sehingga model lebih general

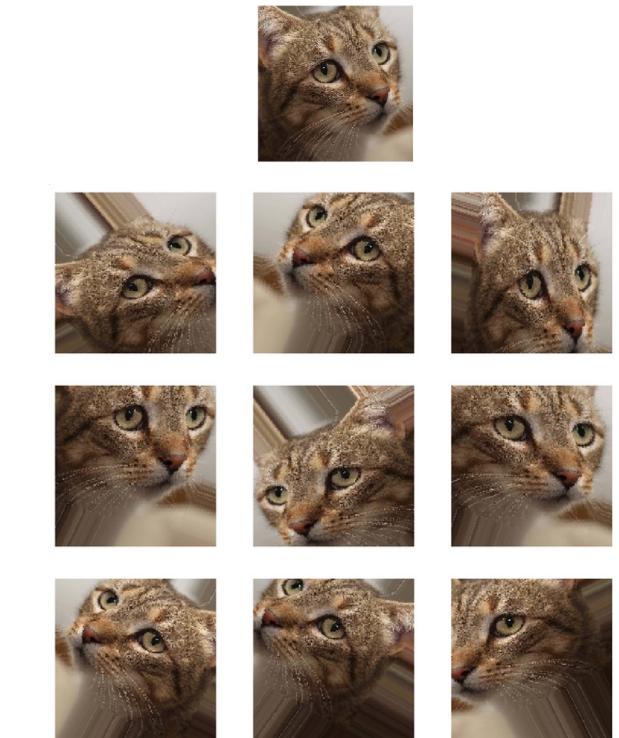
- ***Dropout***
 - Penentuan neuron yang diset tidak aktif sesuai prosentase *dropout* $p\%$
- ***Early stopping***
 - Iterasi pada saat training dihentikan jika *generalization error* mulai naik



Mencegah *Overfitting*

Menambah Data Latih (Augmentasi Data)

- Proses memperbanyak variasi data latih, sehingga model yang dihasilkan lebih baik dalam memprediksi data uji
- Metode augmentasi data yang digunakan tergantung dari jenis data input
- Metode *oversampling* data numerik: smote, adasyn, dan sebagainya
- Contoh augmentasi data citra: rotasi, translasi, flip, dan zoom



Tahapan implementasi ANN

- Load data: membaca file data input
- Split data: membagi data menjadi data latih, data validasi, data uji
- Define model: merancang arsitektur atau model ANN
- Compile model: menjalankan model ANN yang sudah dirancang
- Fit model: membangun model ANN berdasarkan data latih
- Evaluation model: mengevaluasi model ANN berdasarkan data validasi
- Save model: menyimpan model ANN
- Prediction: memprediksi output dari data uji menggunakan model ANN yang terbaik

Bisa digabung
menjadi satu

Tools / Lab Online

Scikit-learn

sklearn.neural_network.MLPClassifier

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=100, activation='relu', *, solver='adam', alpha=0.0001,  
batch_size='auto', learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_iter=200, shuffle=True,  
random_state=None, tol=0.0001, verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True,  
early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_no_change=10, max_fun=15000)
```

[source]

Parameter MLPClassifier:

- hidden_layer size: jumlah neuron di *hidden layer*
- activation: fungsi aktivasi yang digunakan di *hidden layer*
- solver: metode adaptive learning rate yang digunakan
- batch_size: ukuran batch
- learning_rate_init: inisialisasi *learning rate*
- max_iter: *maksimum iterasi*
- early_stopping: bernilai false jika tidak menerapkan *early stopping*

Contoh implementasi ANN menggunakan library Scikit-learn

- Load data

sklearn.datasets.load_iris

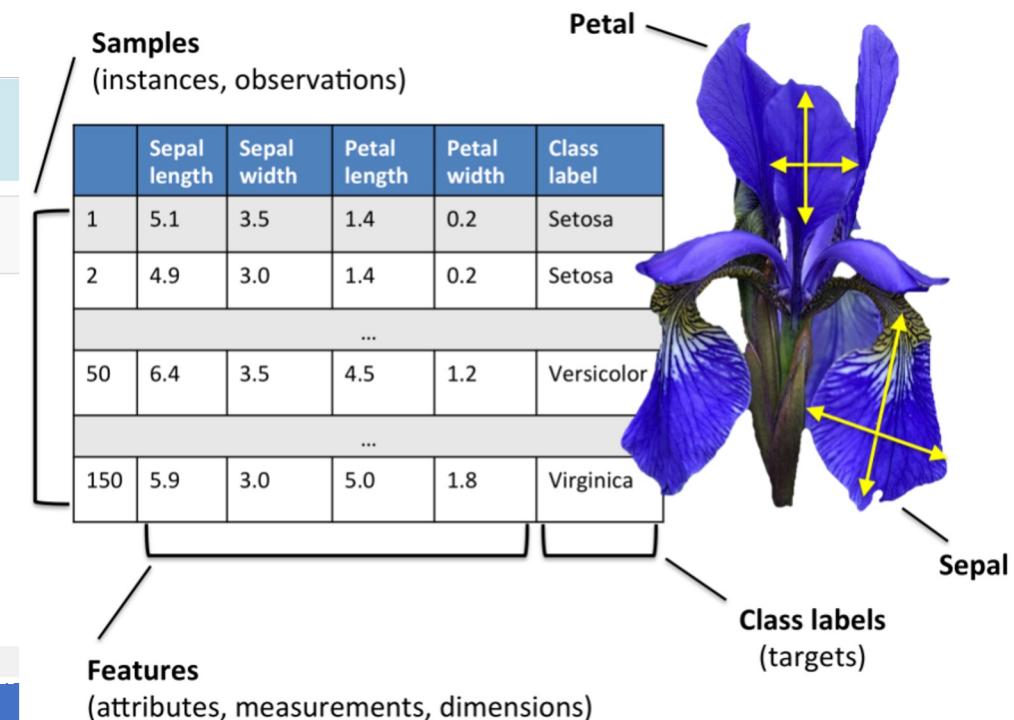
```
sklearn.datasets.load_iris(*, return_X_y=False, as_frame=False)
```

Load and return the iris dataset (classification).

The iris dataset is a classic and very easy multi-class classification dataset.

Classes	3
Samples per class	50
Samples total	150
Dimensionality	4
Features	real, positive

```
from sklearn import datasets  
  
iris = datasets.load_iris()  
  
X = iris.data  
  
y = iris.target
```



Contoh implementasi ANN menggunakan library Scikit-learn

- Split data https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

sklearn.model_selection.train_test_split

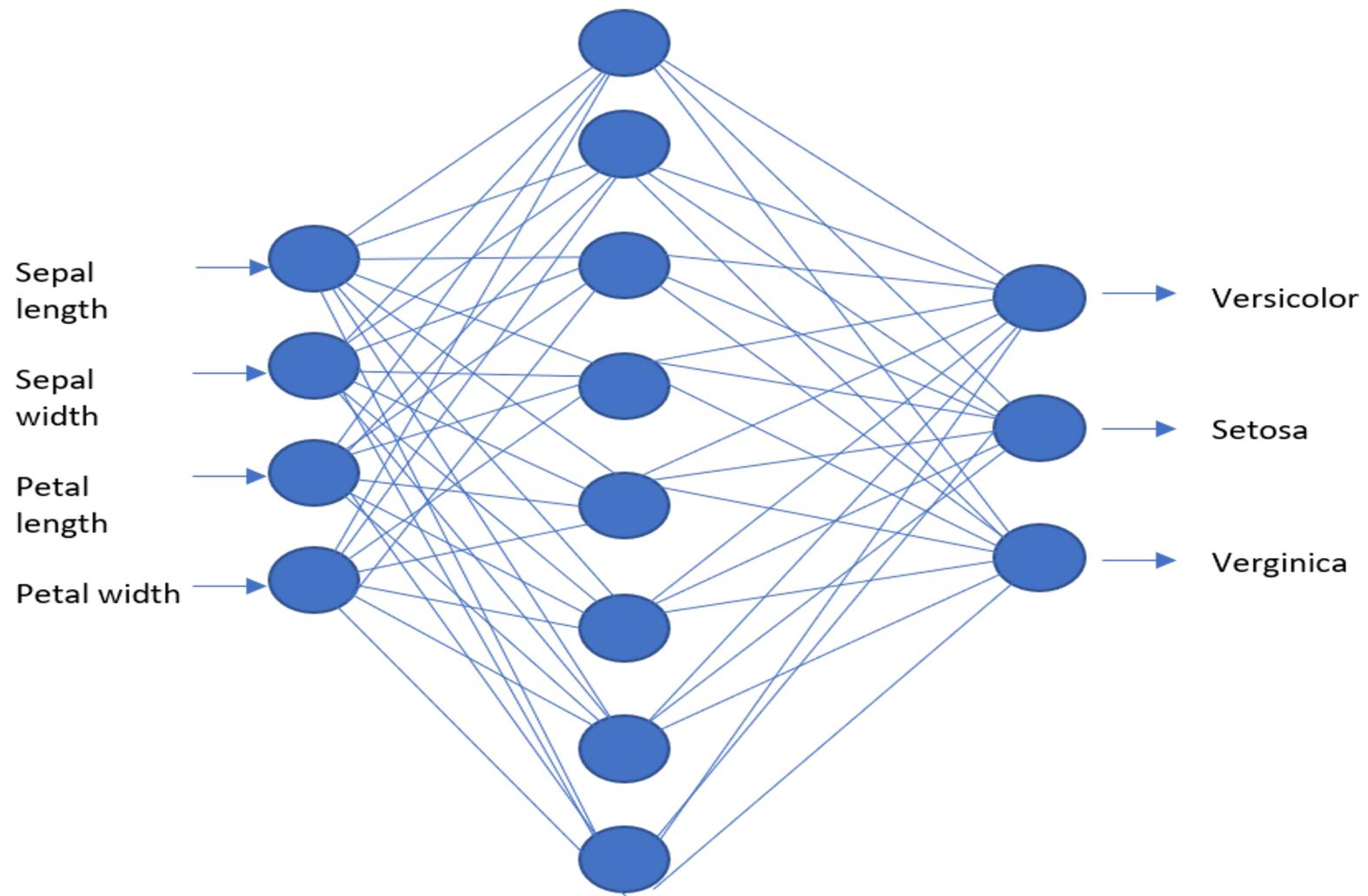
```
sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True,  
stratify=None) [source]
```

```
from sklearn.model_selection import train_test_split  
  
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=.10)  
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=.2)  
print('X_train', X_train.shape)  
print('X_val', X_val.shape)  
print('X_test', X_test.shape)
```

X_train (108, 4)
X_val (27, 4)
X_test (15, 4)

Contoh implementasi ANN menggunakan library Scikit-learn

Arsitektur ANN untuk Klasifikasi Iris



Iris Versicolor



Iris Setosa



Iris Virginica

Contoh implementasi ANN menggunakan library Scikit-learn

- Define and compile model

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(100, ), activation='logistic', max_iter= 800)
```

- Fit model and evaluation model

```
from sklearn.metrics import accuracy_score

mlp.fit(X_train, Y_train)
prediksi_val = mlp.predict(X_val)
acc_val = accuracy_score(Y_val, prediksi_val)
print('Akurasi Validasi Training ANN:', acc_val)
```

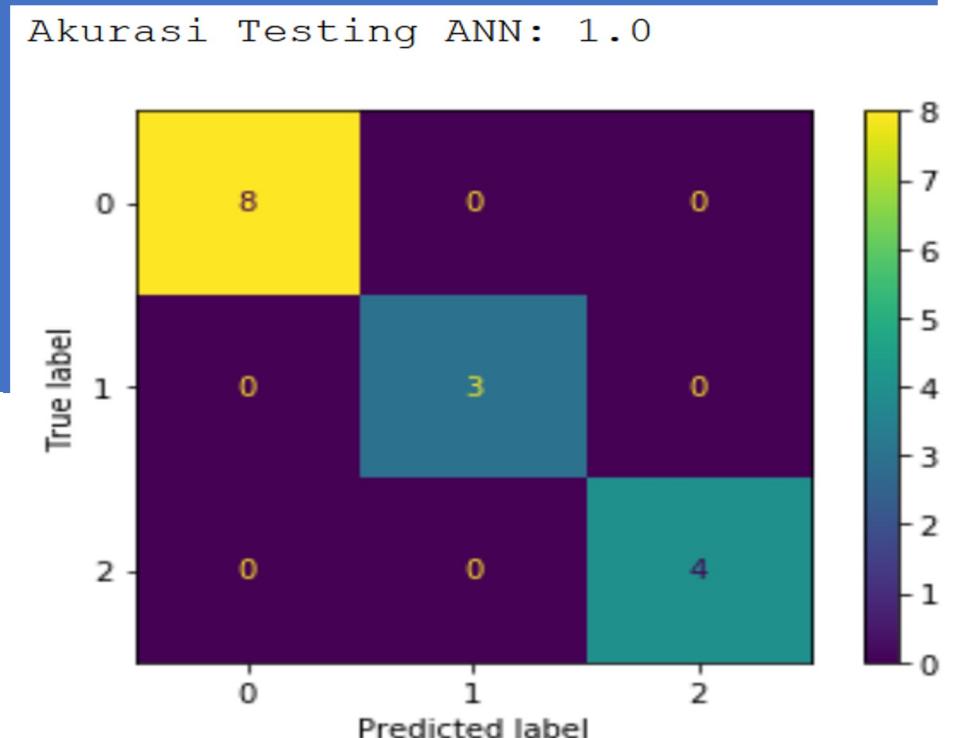
Akurasi Validasi Training ANN: 1.0

Contoh implementasi ANN menggunakan library Scikit-learn

- Prediction

```
from sklearn.metrics import accuracy_score, plot_confusion_matrix

prediksi_test = mlp.predict(X_test)
acc_test = accuracy_score(Y_test, prediksi_test)
print('Akurasi Testing ANN:', acc_test)
plot_confusion_matrix(mlp, X_test, Y_test)
```



Tools / Lab Online

- TensorFlow is an end-to-end open-source platform for machine learning
- Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research.



<https://www.tensorflow.org/overview/>

https://keras.io/getting_started/
<https://keras.io/examples/>

Tools / Lab Online

- Keras Model Sequential dan Layers

```
model = keras.Sequential(  
    [  
        layers.Dense(64, activation="relu", name="layer1"),  
        layers.Dense(32, activation="relu", name="layer2"),  
        layers.Dense(4, name="layer3"),  
    ]  
)
```

atau

```
model = keras.Sequential()  
model.add(layers.Dense(64, activation="relu"))  
model.add(layers.Dense(32, activation="relu"))  
model.add(layers.Dense(4))
```

<https://keras.io/api/models/>

https://keras.io/guides/sequential_model/

<https://keras.io/api/layers/>

Contoh implementasi ANN menggunakan library keras

- Load data

```
from sklearn import datasets  
  
from sklearn.model_selection import train_test_split  
  
from keras.utils import to_categorical  
  
iris = datasets.load_iris()  
  
X = iris.data  
  
y = iris.target  
  
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=.10)  
  
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=.2)  
  
print('X_train', X_train.shape)  
  
print('X_val', X_val.shape)  
  
print('X_test', X_test.shape)  
  
Y_train = to_categorical(Y_train,3)  
  
Y_val = to_categorical(Y_val,3)  
  
Y_test = to_categorical(Y_test,3)
```

Contoh implementasi ANN menggunakan library keras

- Define model dan compile model

```
from keras.models import Sequential  
from keras.layers import Flatten, Dense  
  
model = Sequential()  
model.add(Flatten())  
model.add(Dense(64, activation='relu'))  
model.add(Dense(3, activation='softmax'))  
  
model.compile(optimizer='adam', loss='categorical_crossentropy',  
metrics=['acc'])
```

Contoh implementasi ANN menggunakan library keras

- Fit model

```
model.fit(X_train,Y_train,epochs=64,batch_size=5,validation_data=(X_test,Y_test))  
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
flatten (Flatten)	(None, 4)	0
dense (Dense)	(None, 64)	320
dense_1 (Dense)	(None, 3)	195
=====		
Total params:	515	
Trainable params:	515	
Non-trainable params:	0	

Contoh implementasi ANN menggunakan library keras

- Evaluation model

```
loss, accuracy = model.evaluate(X_test, Y_test)
print('Akurasi Testing MLP:', accuracy)
```

```
1/1 [=====] - 0s 0s/step - loss: 0.0393 - acc: 1.0000
Akurasi Testing ANN: 1.0
```

Contoh implementasi arsitektur *fully connected layer* pada pengenalan angka

Dataset MNIST *Handwritten Digit* dibagi menjadi 3:

- 55,000 training data
- 10,000 test data
- 5,000 validation data

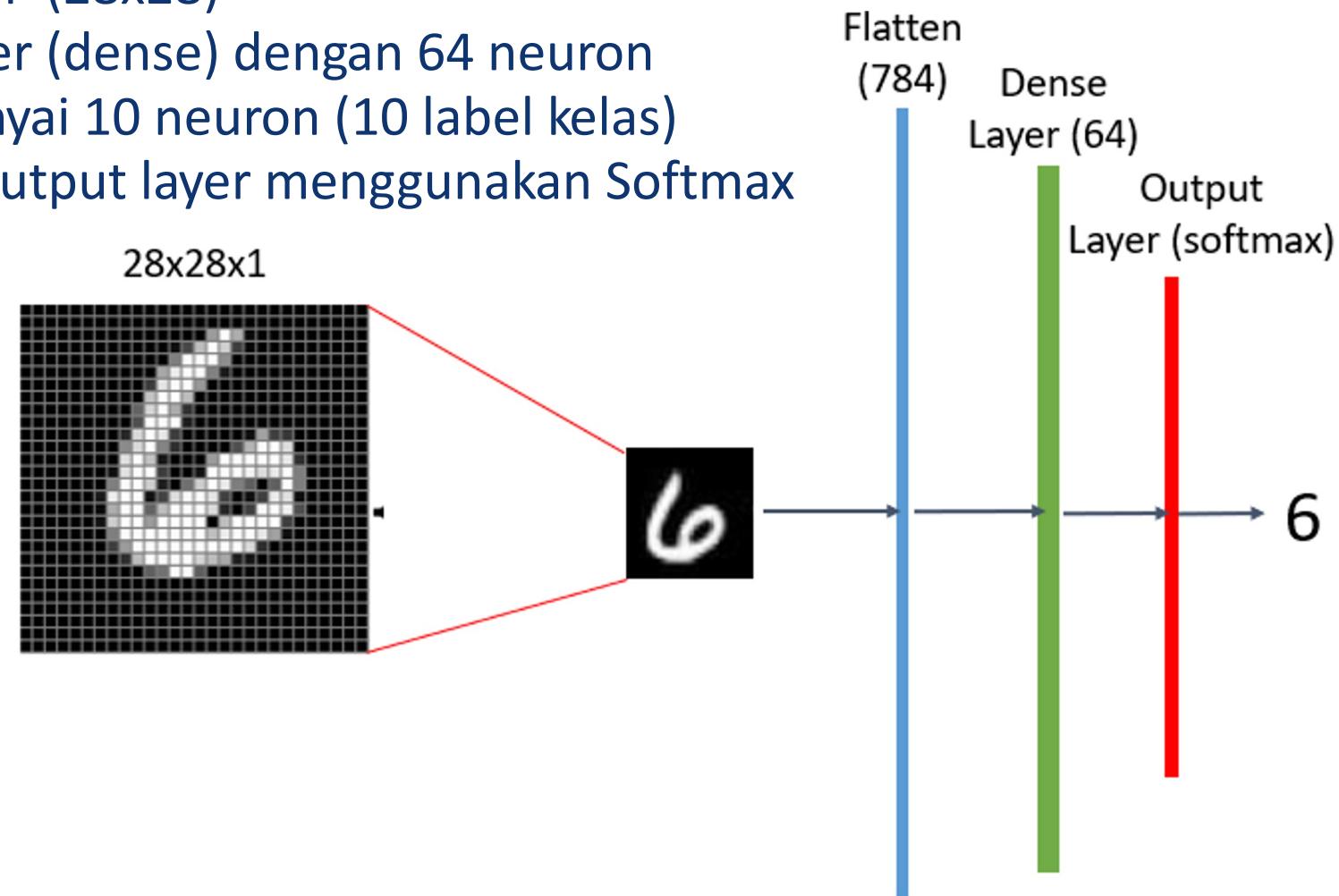
Setiap citra berukuran 28×28 pixels dan label kelas diubah menjadi *one hot encoded*

label = 5	label = 0	label = 4	label = 1	label = 9	0	[1 0 0 0 0 0 0 0]
2	0	4	1	9	1	[0 1 0 0 0 0 0 0]
2	1	3	1	4	2	[0 0 1 0 0 0 0 0]
3	5	3	6	1	3	[0 0 0 1 0 0 0 0]
7	2	8	6	9	4	[0 0 0 0 1 0 0 0]
					5	[0 0 0 0 0 1 0 0]
					6	[0 0 0 0 0 0 1 0]
					7	[0 0 0 0 0 0 0 1]

Contoh implementasi arsitektur *fully connected layer* pada pengenalan angka

Arsitektur ANN yang digunakan untuk pengenalan angka:

- Ukuran input layer 784 (28x28)
- Terdapat 1 hidden layer (dense) dengan 64 neuron
- Output layer mempunyai 10 neuron (10 label kelas)
- Fungsi aktivasi pada output layer menggunakan Softmax



Contoh implementasi arsitektur *fully connected layer* pada pengenalan angka

1. Load data

```
import keras

from keras.datasets import mnist

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

X_train = X_train.reshape(-1, 28,28,1)
X_test = X_test.reshape(-1, 28,28,1)

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

X_train /= 255
X_test /= 255

y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

Contoh implementasi arsitektur *fully connected layer* pada pengenalan angka

2. Define

Model

```
from keras.models import Sequential  
  
from keras.layers import Flatten, Dense  
  
model1 = Sequential()  
model1.add(Flatten())  
model1.add(Dense(64, activation='relu'))  
model1.add(Dense(10, activation='softmax'))
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 64)	50240
dense_1 (Dense)	(None, 10)	650
<hr/>		
Total params: 50,890		
Trainable params: 50,890		
Non-trainable params: 0		
<hr/>		

Contoh implementasi arsitektur *fully connected layer* pada pengenalan angka

3. Compile Model, Fit Model, Save Model, dan Evaluasi

Model

```
modell.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])  
  
history =  
modell.fit(X_train, y_train, epochs=10, batch_size=100, validation_data=(X_test, y_test))  
  
modell.save('my_modell.h5')  
  
modell.evaluate(X_test, y_test)
```

313/313 [=====] - 0s 1ms/step - loss: 0.0839 - acc: 0.9761

[0.08388985693454742, 0.9761000275611877]

Contoh implementasi arsitektur *fully connected layer* pada pengenalan angka

3. Visualisasi Evaluasi Model

```
import matplotlib.pyplot as plt

epochs = range(10)

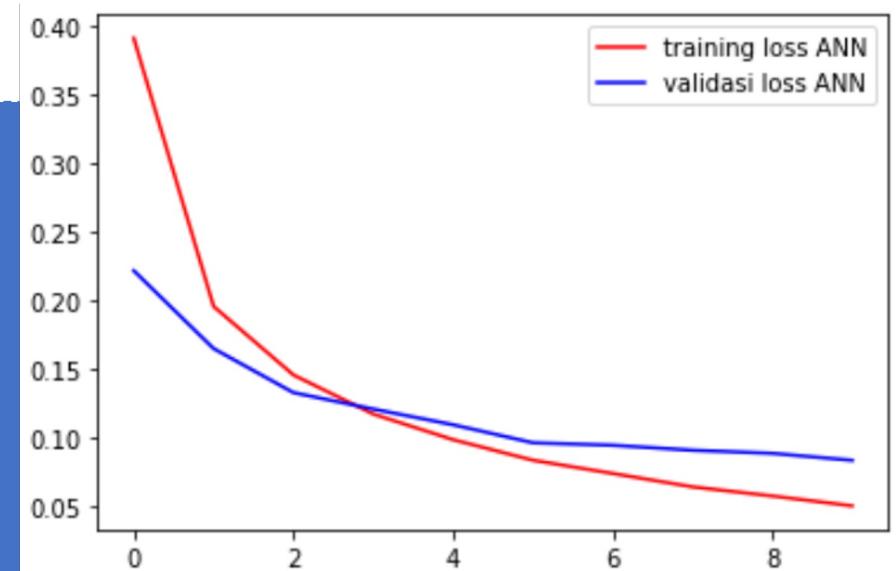
loss1 = history1.history['loss']

val_loss1 = history1.history['val_loss']

plt.plot(epochs, loss1, 'r', label='training loss ANN')

plt.plot(epochs, val_loss1, 'b', label='validasi loss ANN')

plt.legend()
```



Contoh implementasi arsitektur *fully connected layer* pada pengenalan angka

3. Load Model dan

Prediction

```
import numpy as np  
  
from keras.models import load_model  
  
  
model_simpan = load_model('my_model.h5')  
pred = model_simpan.predict(x_test)  
print('label actual:',np.argmax(y_test[30]))  
print('label prediction:',np.argmax(pred[30]))
```

```
label actual: 3  
label prediction: 3
```

Referensi

- 虞台文, Feed-Forward Neural Networks, Course slides presentation
- Andrew Ng, Machine Learning, Course slides presentation
- Michael Negnevitsky, Artificial Intelligence : A Guide to Intelligent Systems, Second Edition, Addison Wesley, 2005.
- Hung-yi Lee, Deep Learning Tutorial
- Alexander Amini, Intro to Deep Learning, MIT 6.S191, 2021