



DIGITAL  
TALENT  
SCHOLARSHIP



# THEMATIC ACADEMY

## Tema Pelatihan

Pertemuan #14- : Membangun Model Artificial Neural Network (ANN)



KOMINFO



**#JADIJAGOANDIGITAL**

Badan Penelitian dan Pengembangan Sumber Daya Manusia

# Course Definition

UK J.62DMI00.013.1 - Membangun Model (ANN)

- a. Menyiapkan parameter model
- b. Menggunakan tools pemodelan

Menjelaskan algoritma fully connected NN, CNN, RNN dan menjelaskan penggunaan library sklearn dan keras untuk model berbasis neural network

Durasi 6 JP (270 Menit)

Rasio : Praktek dan Teori 70 : 30

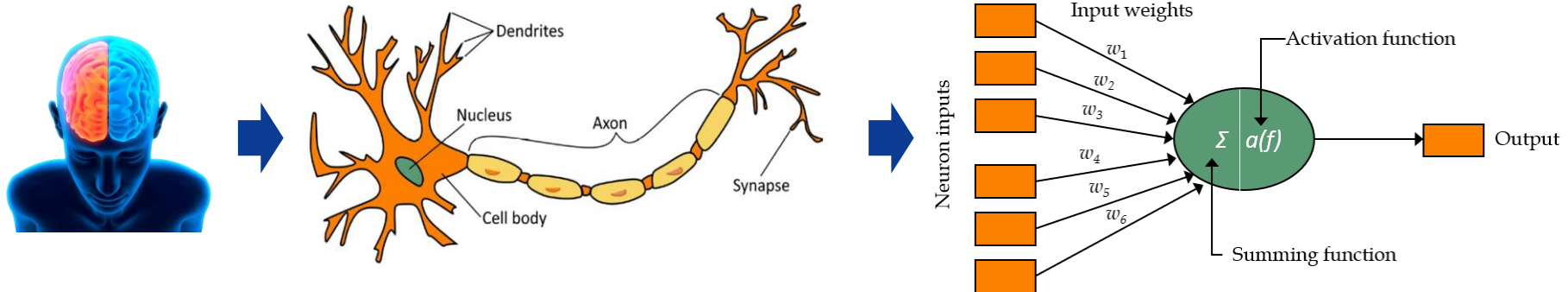
# Learning Objective

Peserta mampu melakukan proses pemodelan Artificial Neural Network (ANN)

# 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 |

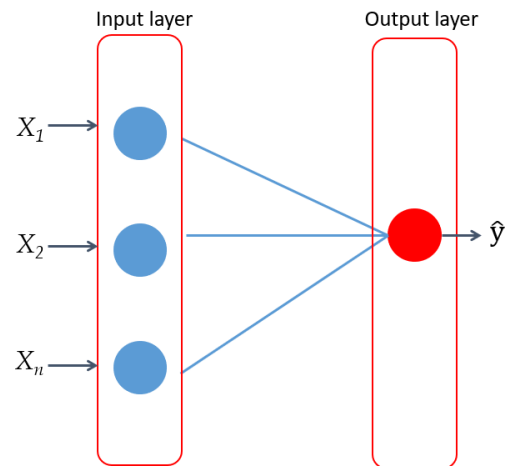
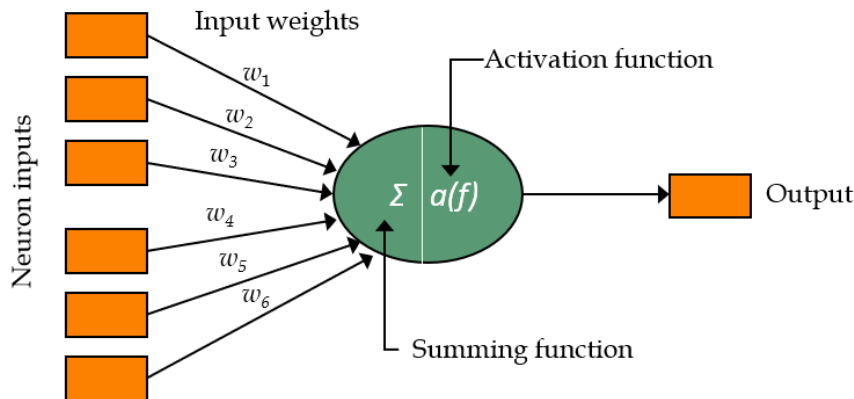


# Arsitektur Single-layer Perceptron

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 menghitung jumlah perkalian bobot terhadap input  $X$
- Fungsi aktivasi  $a(\cdot)$

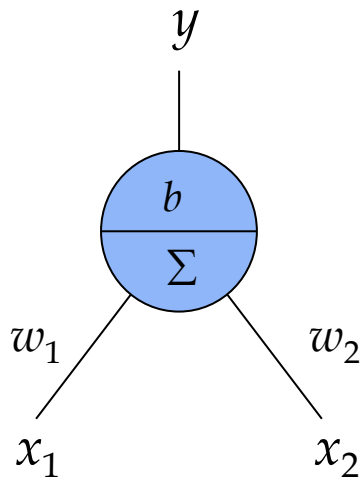


$$f = \sum_{i=1}^m w_i x_i + b$$
$$y = a(f)$$

# Apa yang bisa dilakukan sebuah Neuron ?

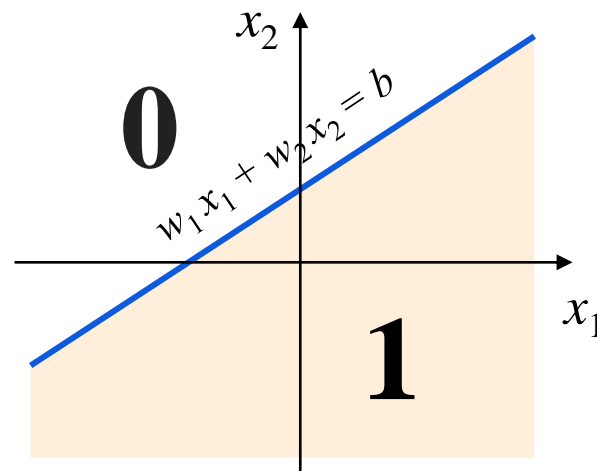
Sebuah neuron pada ANN dapat menyelesaikan permasalahan klasifikasi biner

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



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

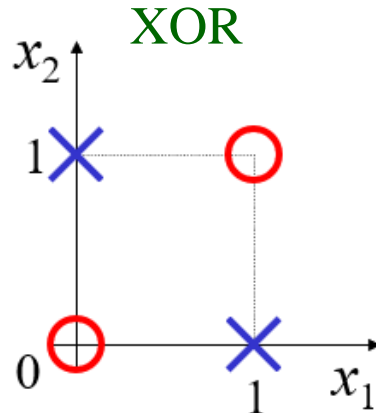
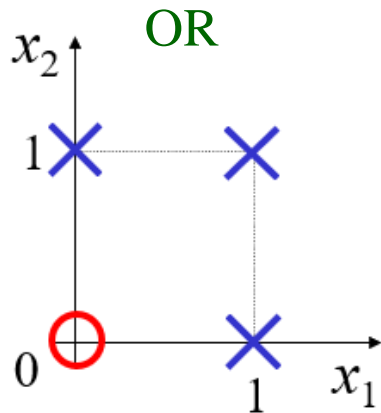
$$y = \begin{cases} 1 & f(x) \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



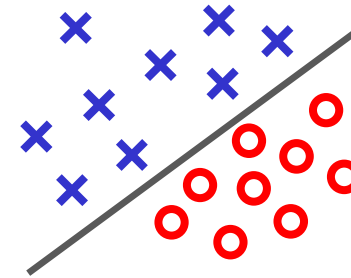
# Permasalahan Linear dan Non-Linear

Permasalahan klasifikasi dapat dikategorikan sebagai:

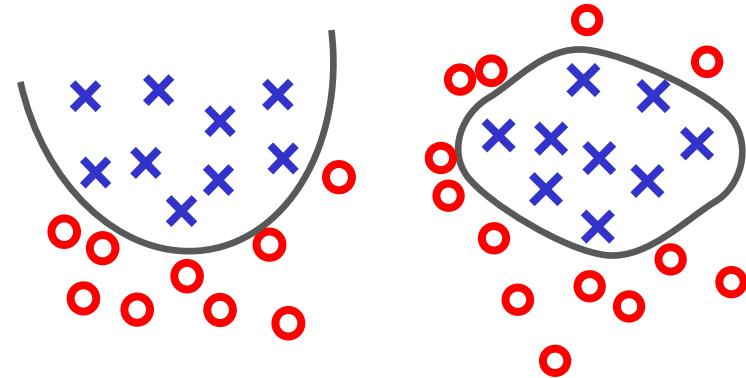
- Permasalahan Linear, misalnya fungsi OR dan AND
- Permasalahan Non-Linear, misalnya fungsi XOR



Linear



Non-linear

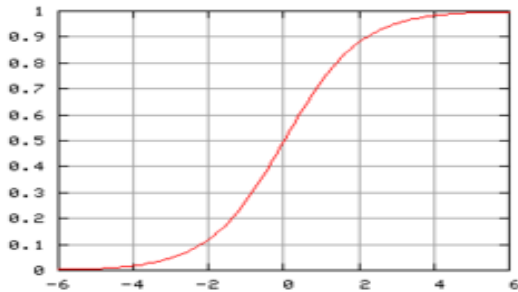


# Fungsi Aktivasi

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

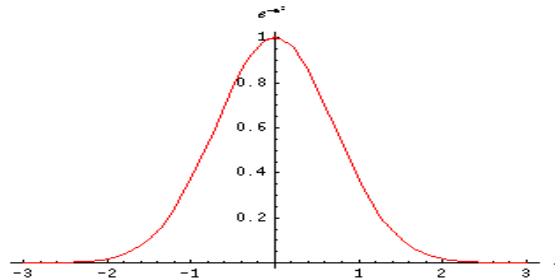
## Sigmoid function

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



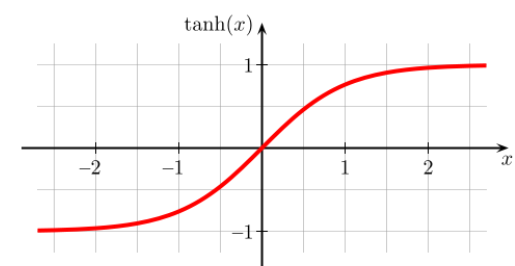
## Gaussian function

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



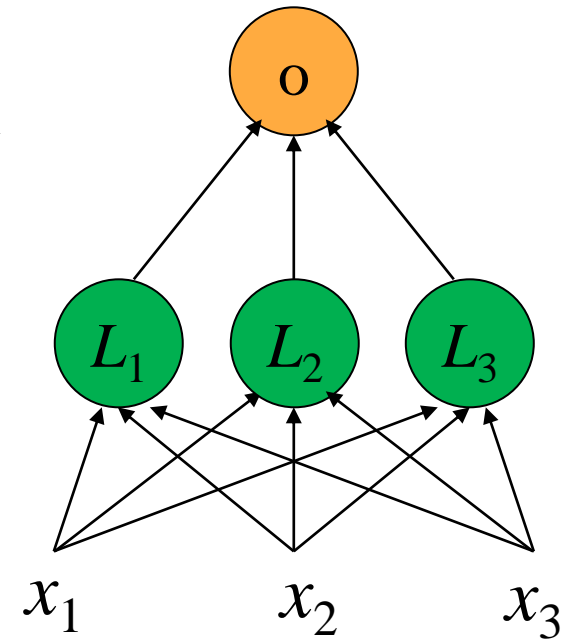
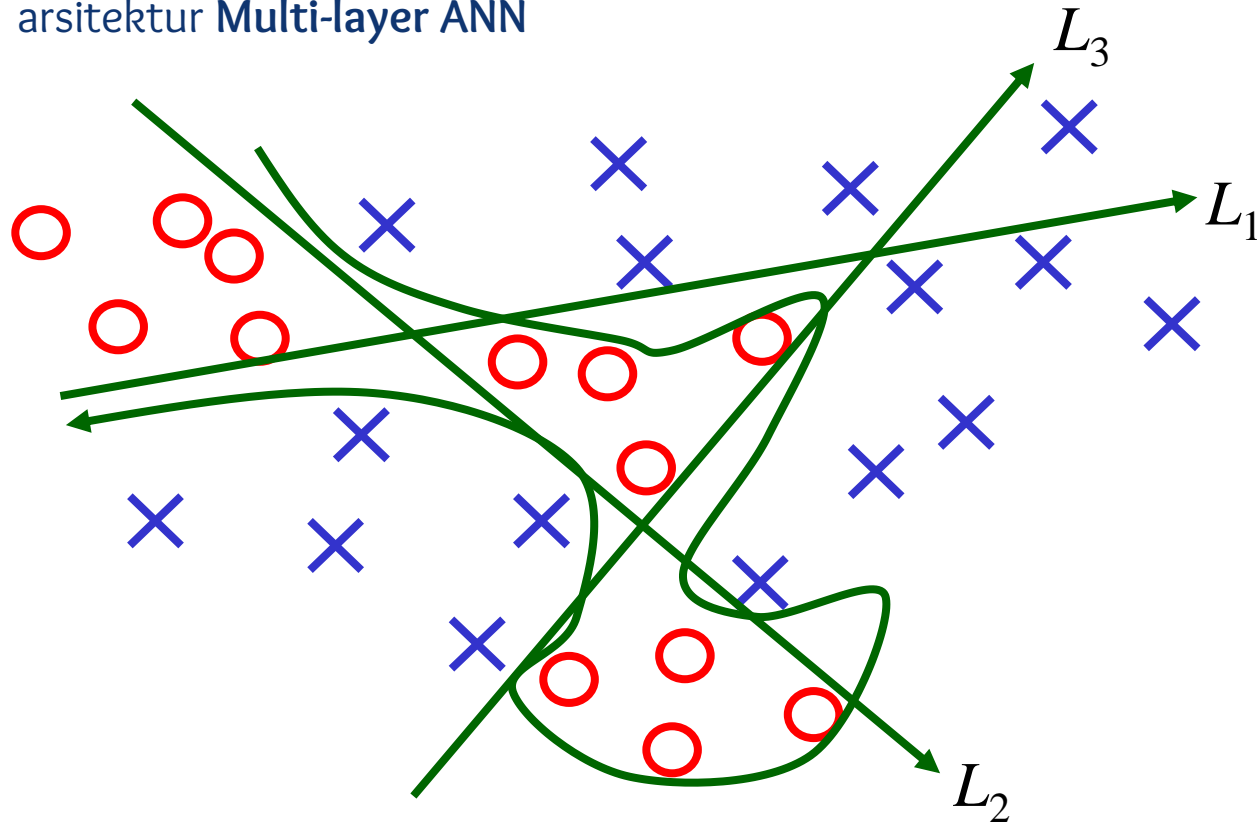
## Tangent Hyperbolic function

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



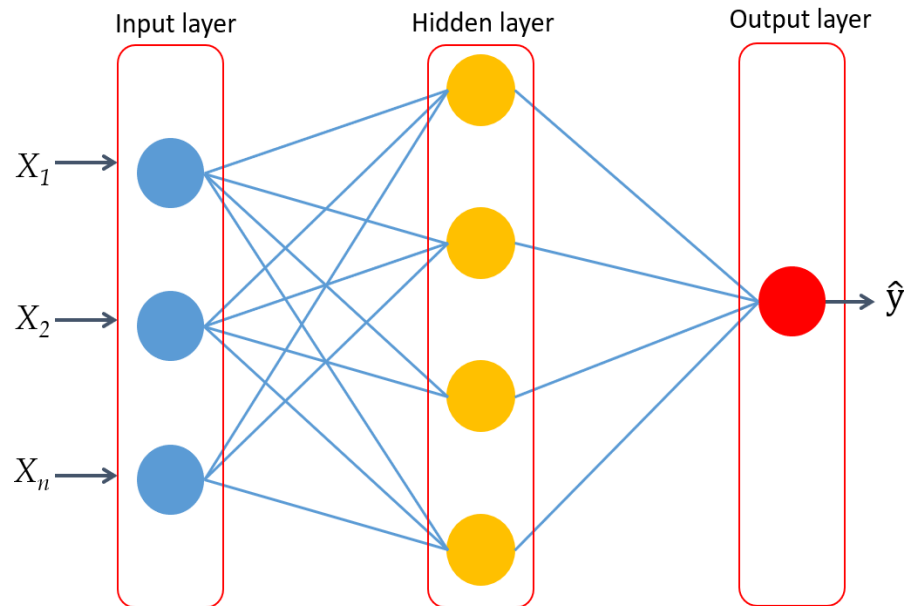


Pada permasalahan non-linear dan permasalahan yang lebih kompleks, menggunakan arsitektur **Multi-layer ANN**



# Arsitektur Multi-layer ANN

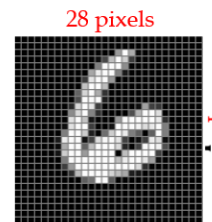
- Terdiri dari tiga layer yaitu:
  - input layer
  - hidden layer
  - output layer
- Hubungan antar neuron pada ANN merupakan **fully connected network (FCN)**
- **Jumlah hidden layer** sebaiknya disesuaikan dengan kompleksitas permasalahan
- **Jumlah neuron pada hidden layer** umumnya lebih banyak daripada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



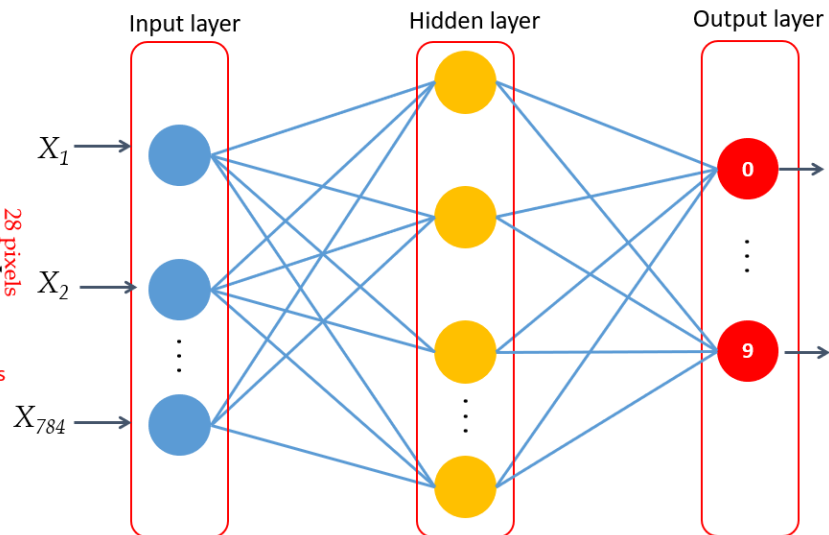
# Desain arsitektur ANN

## Penentuan jumlah *neuron* pada *input layer*

- Jumlah neuron sesuai dengan jumlah fitur pada data input



$28 \times 28 = 784$  pixels



## 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

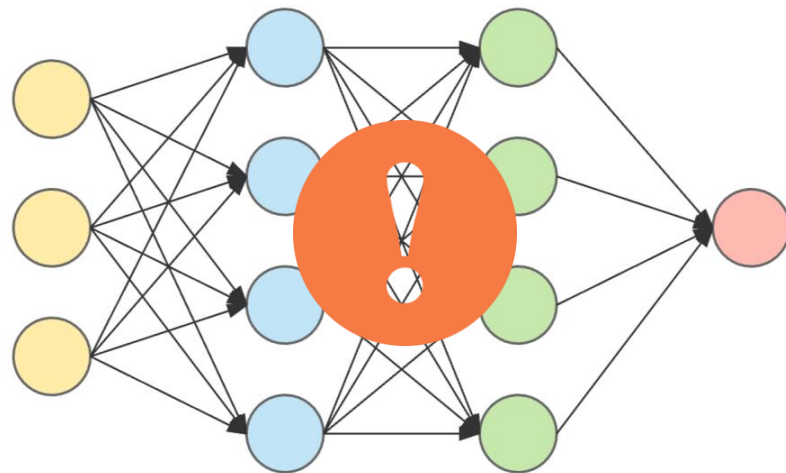
# Desain arsitektur ANN

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/>



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?



Ratio of training to test data: 50%

Noise: 0

Batch size: 10

REGENERATE

## FEATURES

Which properties do you want to feed in?

X<sup>1</sup>  
X<sup>2</sup>  
X<sup>12</sup>  
X<sup>22</sup>  
X<sup>1</sup>X<sup>2</sup>  
sin(X<sup>1</sup>)

+ - 2 HIDDEN LAYERS

+ -  
4 neurons

+ -  
2 neurons

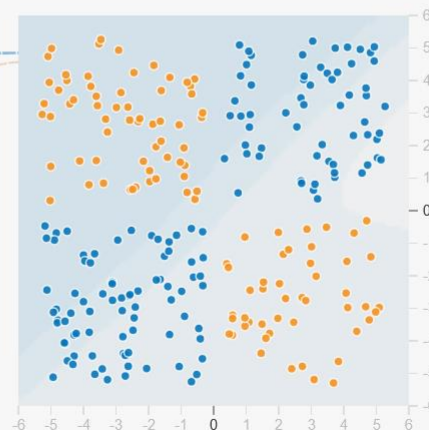


This is the output from one neuron. Hover to see it larger.

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

## OUTPUT

Test loss 0.519  
Training loss 0.501



Colors shown

# Tahapan ANN: *Feed Forward*

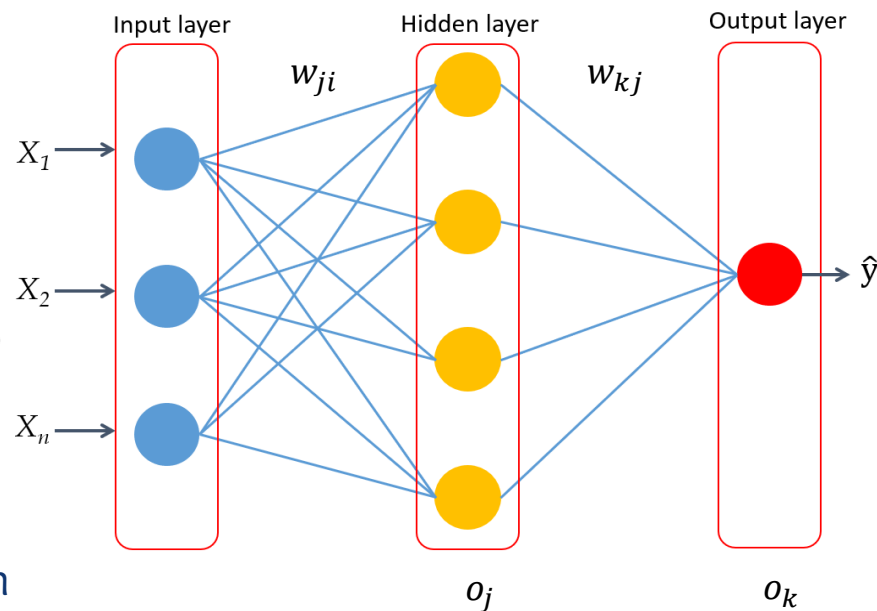
1. Masukkan vektor  $X$  ke *input layer*
2. Hitung output setiap neuron di *hidden layer* dan *output layer*

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

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

3. Hitung fungsi aktivasi pada layer output  $\hat{y} = a(o_k)$



# Tahapan ANN: Pembelajaran (*Learning*)

1. Inisialisasi bobot  $W$
2. Update bobot sehingga output dari ANN adalah konsisten dengan label kelas pada data latih

a. Fungsi obyektif (fungsi loss): 
$$J(W) = \frac{1}{n} \sum_{i=1}^n \left( y^{(i)} - f(x^{(i)}; W) \right)^2$$

$y$  – target

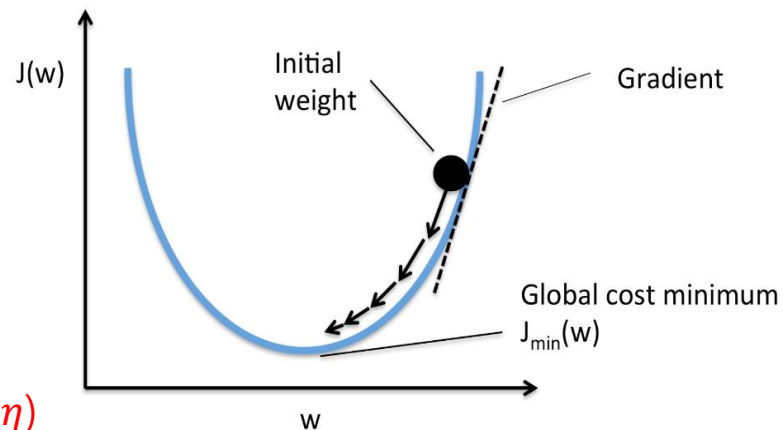
$f$  – prediksi

- 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 ( $\eta$ )**



## 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

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

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

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

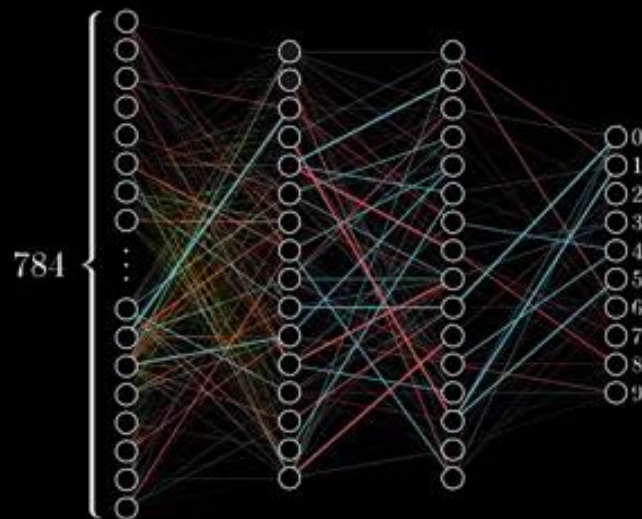
# Tahapan Pembelajaran Multi-layer Perceptron ANN

- 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)

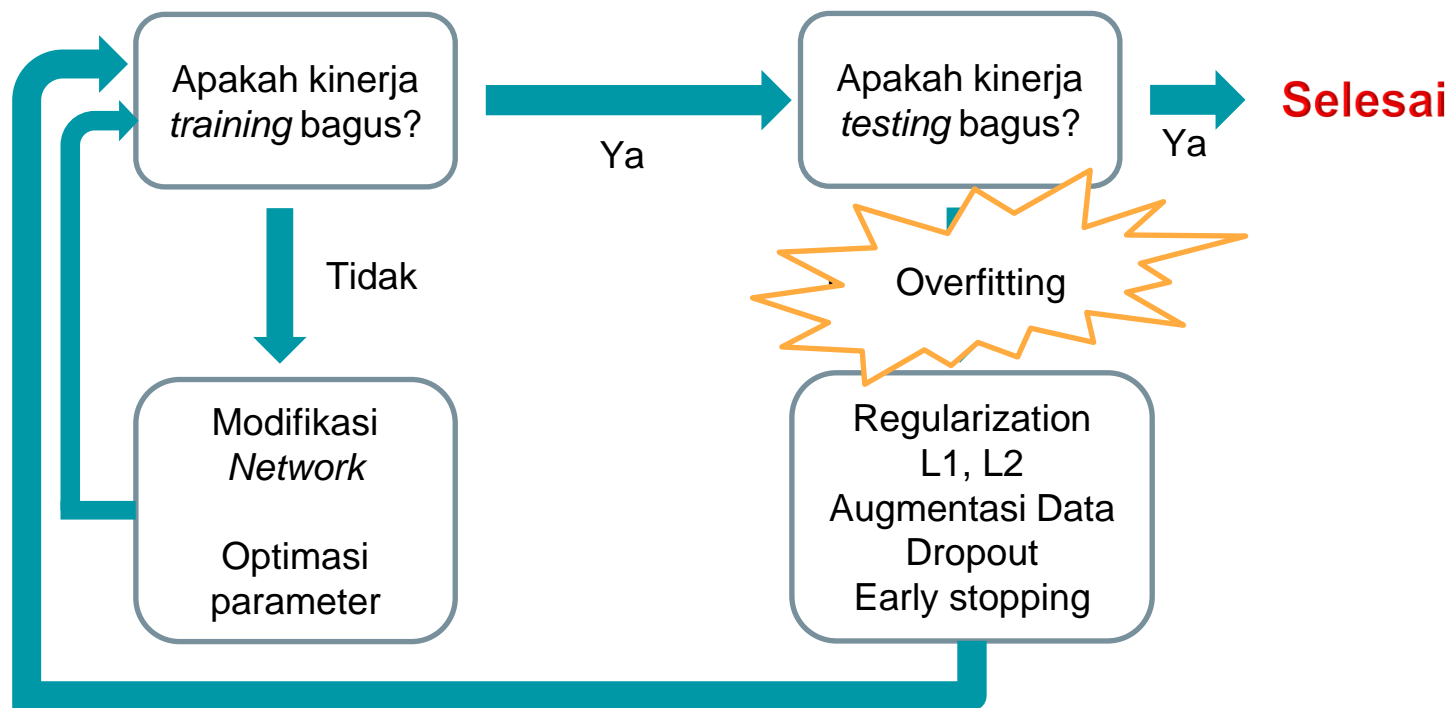
<https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/>

## Contoh Proses Pembelajaran ANN

Training in  
progress...

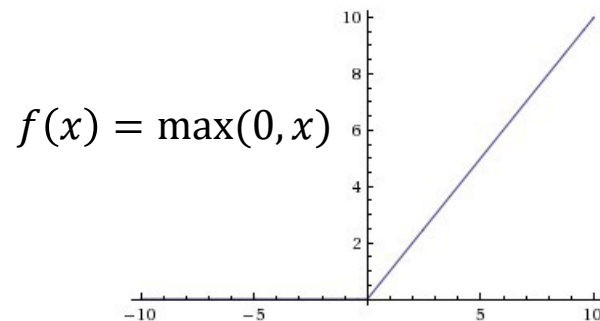


# Strategi Proses Pembelajaran



## 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*

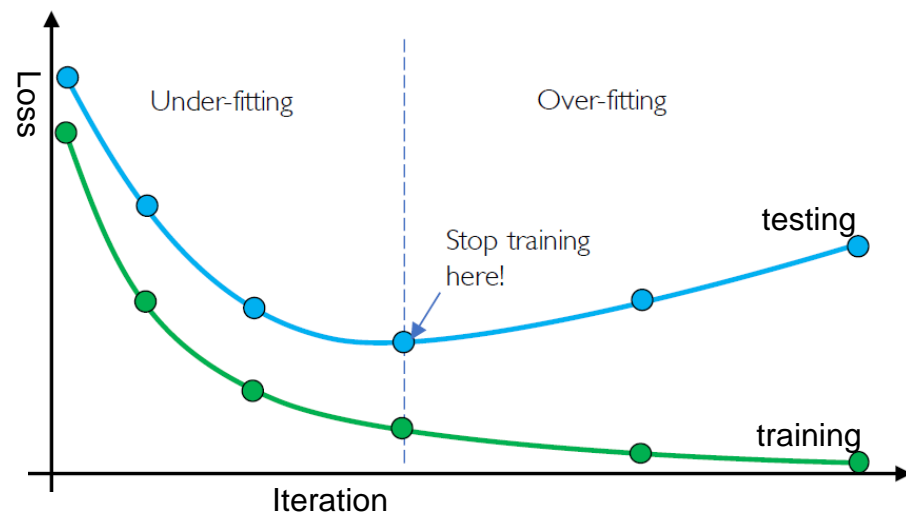
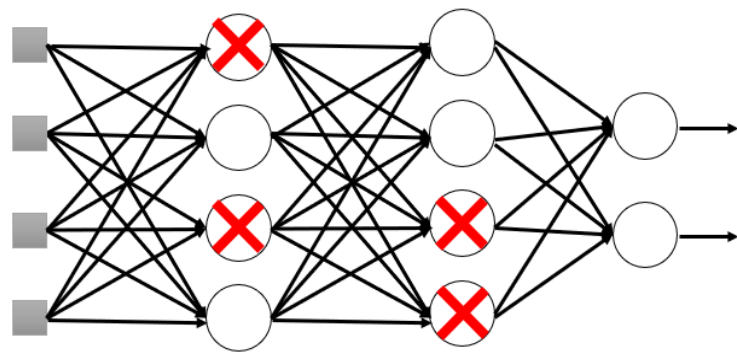
Regularisasi dilakukan untuk mengurangi *generalization error* dengan mencegah model lebih kompleks. Penerapan regularisasi dengan cara menambahkan *regularization term* pada semua parameter (bobot) ke fungsi obyektif.

- **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`

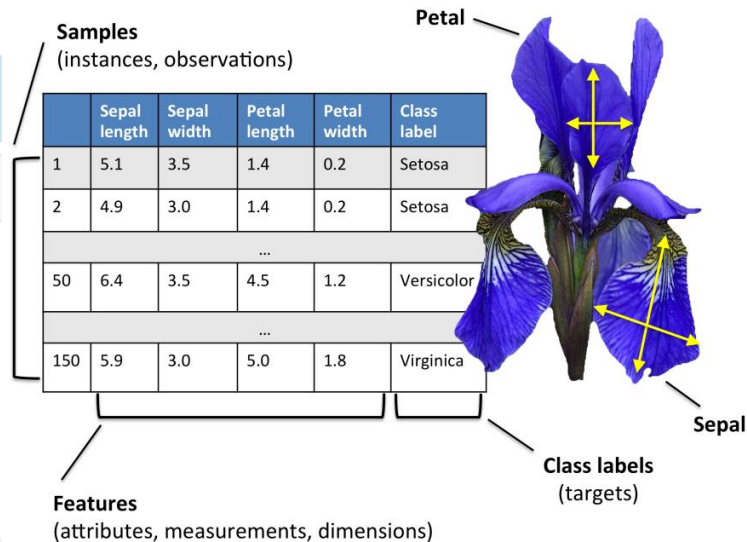
```
sklearn.datasets.load_iris(*, return_X_y=False, as_frame=False)
```

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](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]

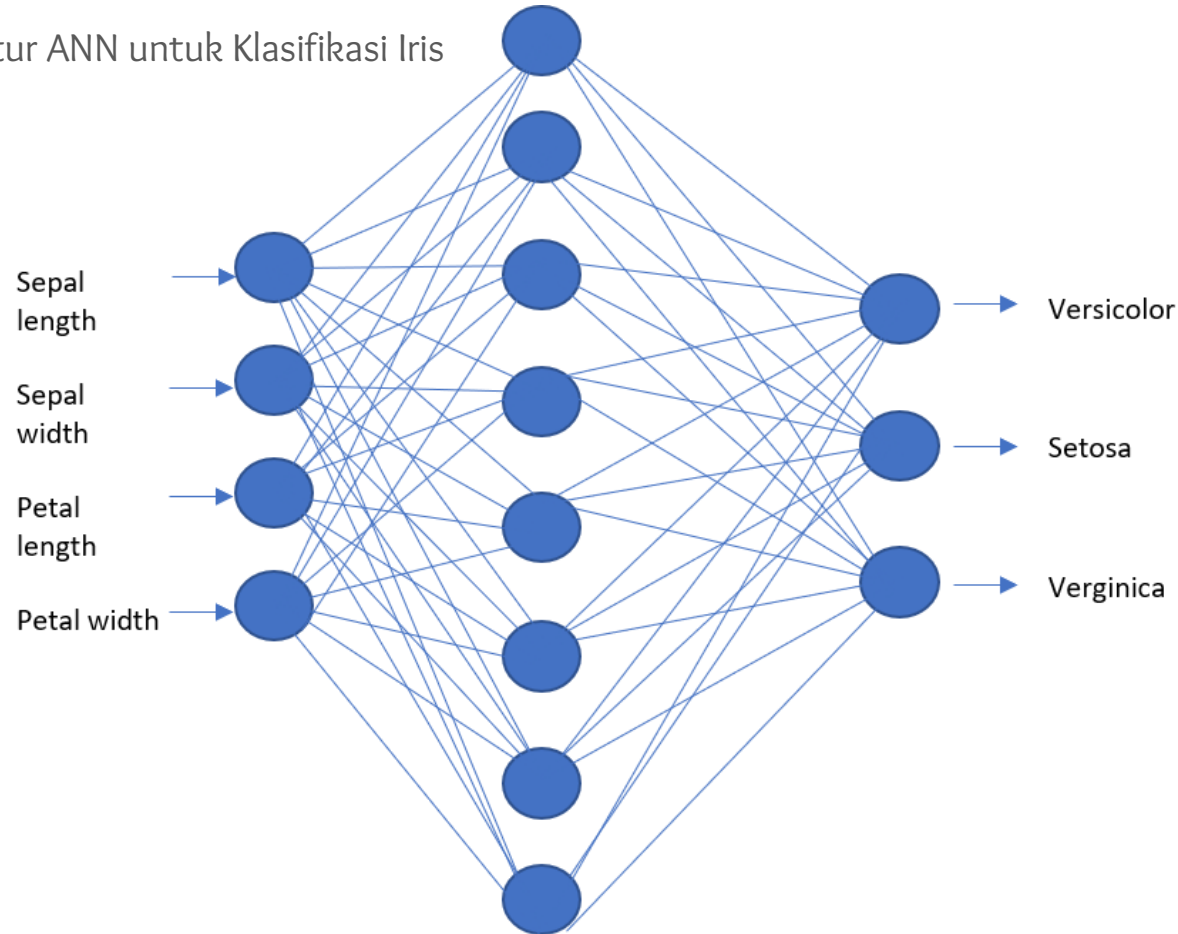
```
from sklearn.model_selection import train_test_split

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
```

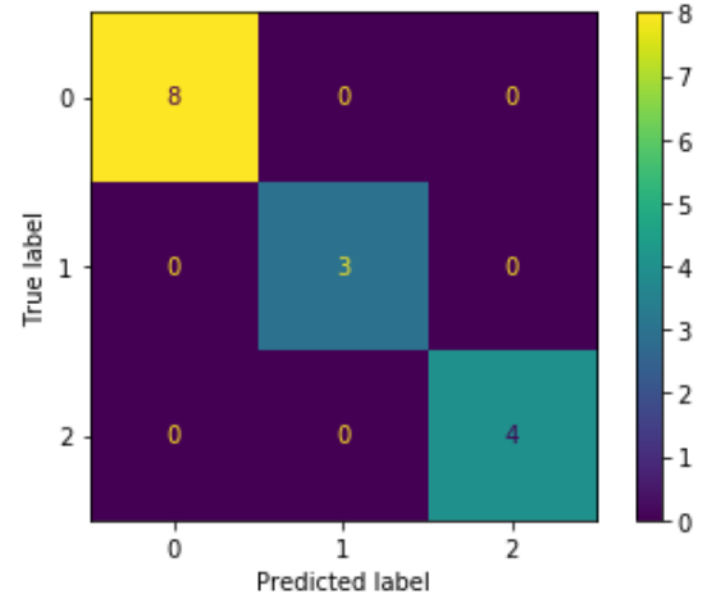
```
prediksi_test = mlp.predict(X_test)
```

```
acc_test = accuracy_score(Y_test, prediksi_test)
```

```
print('Akurasi Testing ANN:', acc_test)
```

```
plot_confusion_matrix(mlp, X_test, Y_test)
```

Akurasi Testing ANN: 1.0



## 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/getting_started/)

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



# Tools / Lab Online

- Keras Model Sequential dan Layers

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

[https://keras.io/guides/sequential\\_model/](https://keras.io/guides/sequential_model/)

<https://keras.io/api/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))
```

# 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

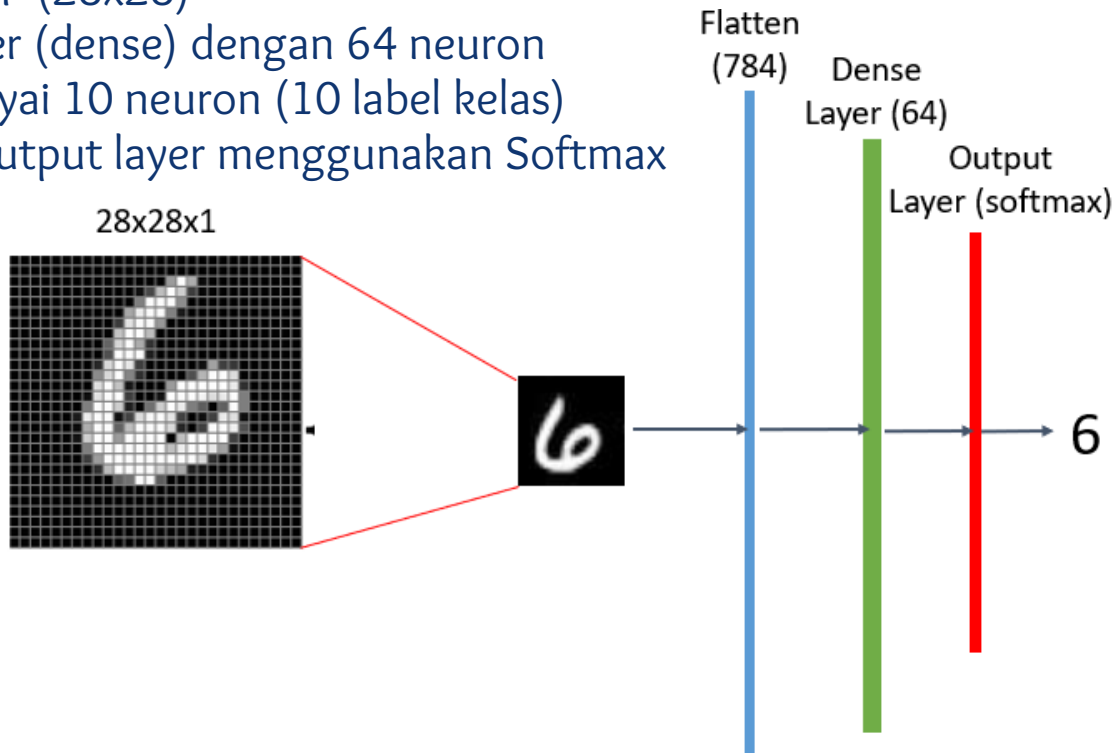
Setiap citra berukuran  $28 \times 28$  pixels dan label kelas dirubah menjadi *one hot encoded*

|  |  |  |  |  |   |                       |
|--|--|--|--|--|---|-----------------------|
| label = 5<br>   | label = 0<br>   | label = 4<br>   | label = 1<br>   | label = 9<br>   | 0 | [1 0 0 0 0 0 0 0 0 0] |
| label = 2<br>   | label = 1<br>   | label = 3<br>   | label = 1<br>   | label = 4<br>   | 1 | [0 1 0 0 0 0 0 0 0 0] |
| label = 3<br>   | label = 5<br>   | label = 3<br>   | label = 6<br>   | label = 1<br>   | 2 | [0 0 1 0 0 0 0 0 0 0] |
| label = 3<br>  | label = 5<br>  | label = 3<br>  | label = 6<br>  | label = 1<br>  | 3 | [0 0 0 1 0 0 0 0 0 0] |
| label = 3<br> | label = 5<br> | label = 3<br> | label = 6<br> | label = 1<br> | 4 | [0 0 0 0 1 0 0 0 0 0] |
| label = 3<br> | label = 5<br> | label = 3<br> | label = 6<br> | label = 1<br> | 5 | [0 0 0 0 0 1 0 0 0 0] |
| label = 3<br> | label = 5<br> | label = 3<br> | label = 6<br> | label = 1<br> | 6 | [0 0 0 0 0 0 1 0 0 0] |
| label = 3<br> | label = 5<br> | label = 3<br> | label = 6<br> | label = 1<br> | 7 | [0 0 0 0 0 0 0 1 0 0] |

## 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 # |
|-------------------|--------------|---------|
| =====             |              |         |
| flatten (Flatten) | (None, 784)  | 0       |
| dense (Dense)     | (None, 64)   | 50240   |
| dense_1 (Dense)   | (None, 10)   | 650     |
| =====             |              |         |

Total params: 50,890

Trainable params: 50,890

Non-trainable params: 0

# Contoh implementasi arsitektur *fully connected layer* pada pengenalan angka

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

```
model1.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['acc'])  
history =  
model1.fit(X_train,y_train,epochs=10,batch_size=100,validation_data=(X_test,y_test))  
model1.save('my_model1.h5')  
model1.evaluate(X_test,y_test)
```

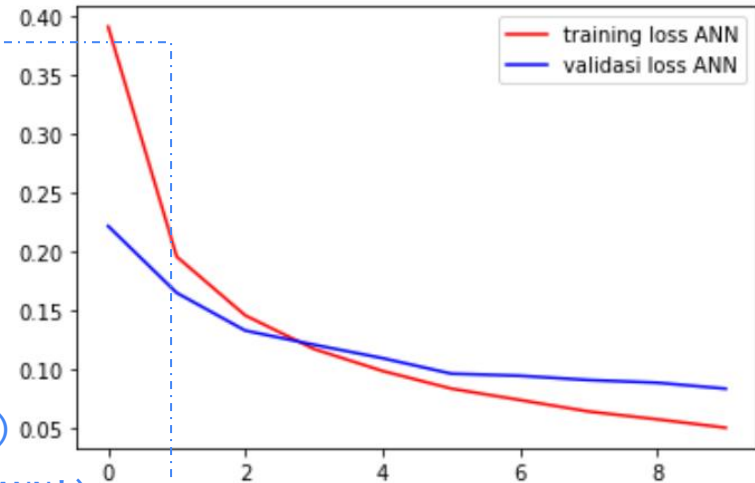
```
313/313 [=====] - 0s 1ms/step - loss: 0.0839 - acc: 0.9761  
[0.08388985693454742, 0.9761000275611877]
```

# Contoh implementasi arsitektur *fully connected layer* pada pengenalan angka

## 3. Visualiasasi 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
```

# Pengantar Deep Learning

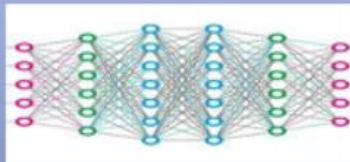
Artificial Intelligence



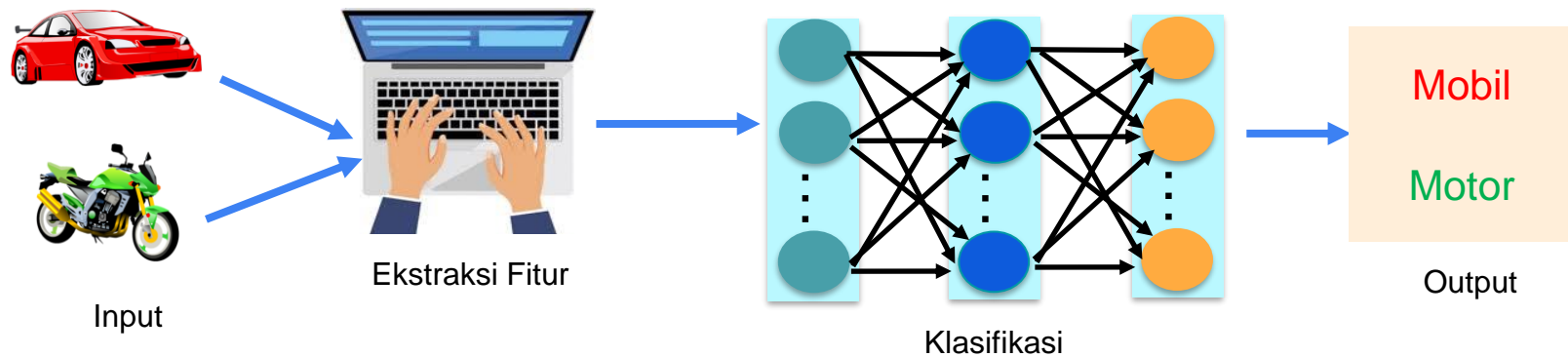
Machine Learning



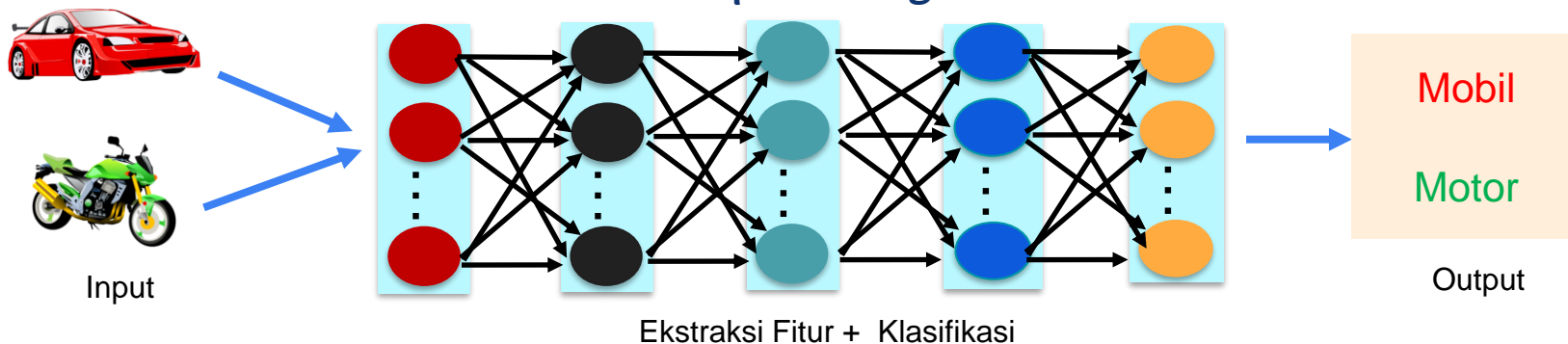
Deep Learning



## Machine Learning (Konvensional)



## Deep Learning



# Pengantar Deep Learning

Pendekatan klasifikasi secara konvensional umumnya melakukan ekstraksi fitur secara terpisah kemudian dilanjutkan proses pembelajaran menggunakan metode klasifikasi konvensional

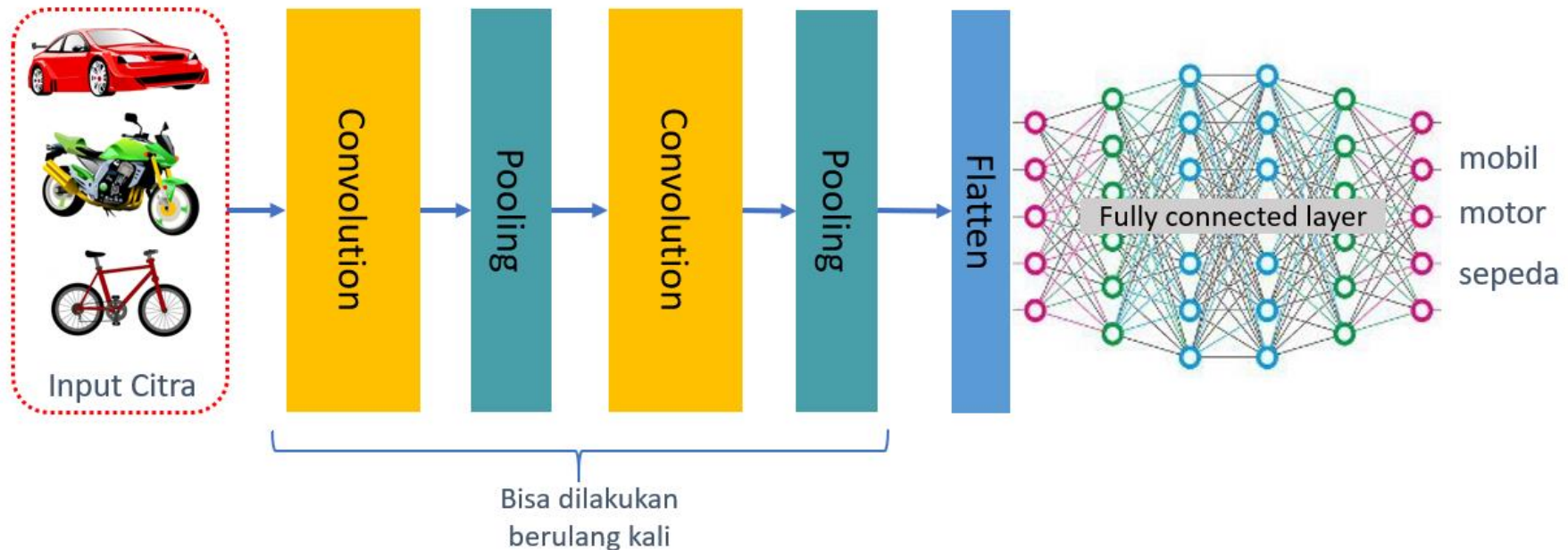
Kelemahan pendekatan konvensional:

- Memerlukan waktu dan pengetahuan lebih untuk ekstraksi fitur
- Sangat tergantung pada satu domain permasalahan saja sehingga tidak berlaku general

Pendekatan klasifikasi berbasis Deep learning mempelajari representasi hirarki (pola fitur) secara otomatis melalui beberapa tahapan proses *feature learning*

# Convolutional Neural Network (CNN)

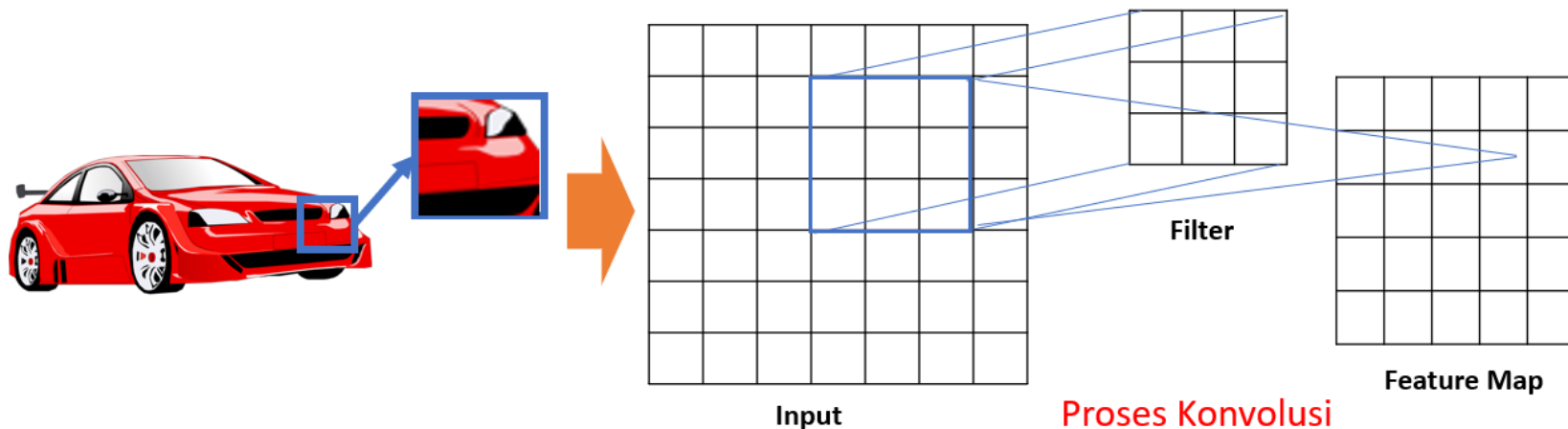
- CNN merupakan metode Deep Learning yang merupakan salah satu jenis arsitektur ANN
- Ada tiga layer utama yaitu *convolutional layer*, *pooling layer*, dan *fully connected layer*





# Convolutional Layer

- *Convolutional layer* merupakan proses konvolusi citra input dengan filter yang menghasilkan *feature map*
- Ukuran matrik citra dan ukuran matrik filter akan mempengaruhi ukuran matrik *feature map*



# Convolutional Layer

- Proses konvolusi citra dengan filter dilakukan sliding filter mulai dari kiri atas dari matirck citra sampai kanan bawah
- Rumus konvolusi dari citra  $I$  dengan filter  $K$  sebagai berikut:

$$(I * K)(i, j) == \sum_m \sum_n I(m, n) K(i + m, j + n)$$

Citra  $I$

|    |    |    |   |   |   |
|----|----|----|---|---|---|
| 30 | 30 | 30 | 0 | 0 | 0 |
| 30 | 30 | 30 | 0 | 0 | 0 |
| 30 | 30 | 30 | 0 | 0 | 0 |
| 30 | 30 | 30 | 0 | 0 | 0 |
| 30 | 30 | 30 | 0 | 0 | 0 |
| 30 | 30 | 30 | 0 | 0 | 0 |

Filter  $K$

|   |   |    |
|---|---|----|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| 1 | 0 | -1 |

\*

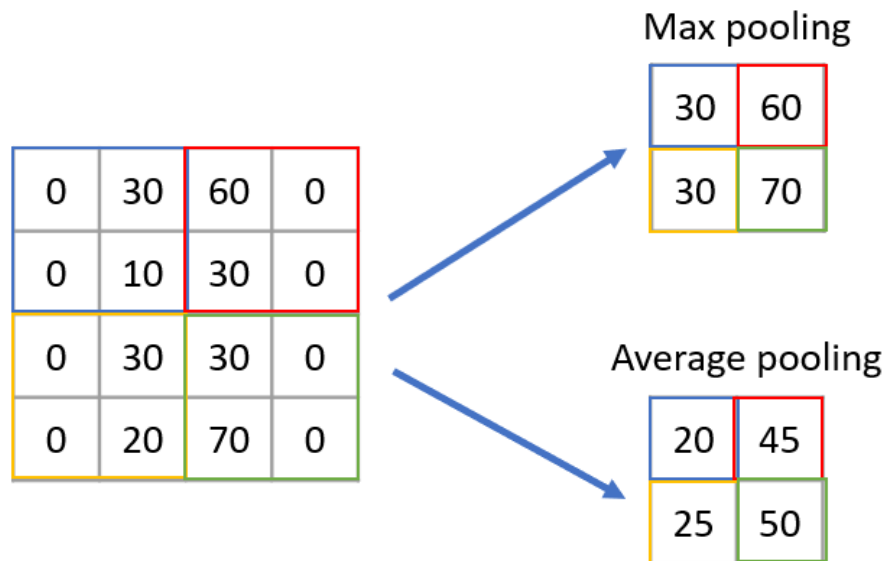
=

Feature Map

|   |    |    |   |
|---|----|----|---|
| 0 | 30 | 30 | 0 |
| 0 | 30 | 30 | 0 |
| 0 | 30 | 30 | 0 |
| 0 | 30 | 30 | 0 |

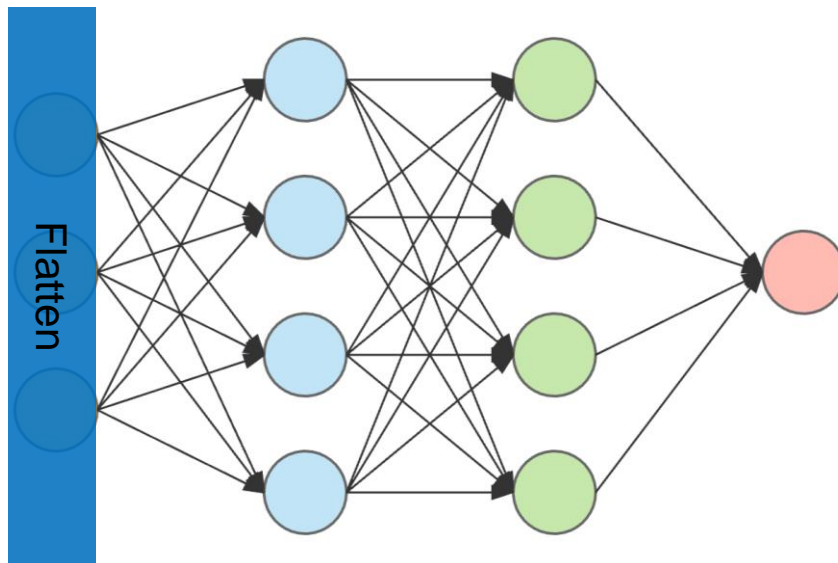
# Pooling Layer

- *Pooling layer* digunakan untuk mengurangi ukuran gambar menjadi lebih kecil (*down sample*) dan mengekstrak *salient features*
- *Pooling layer* yang umum digunakan adalah *Maximum pooling* dan *Average pooling*



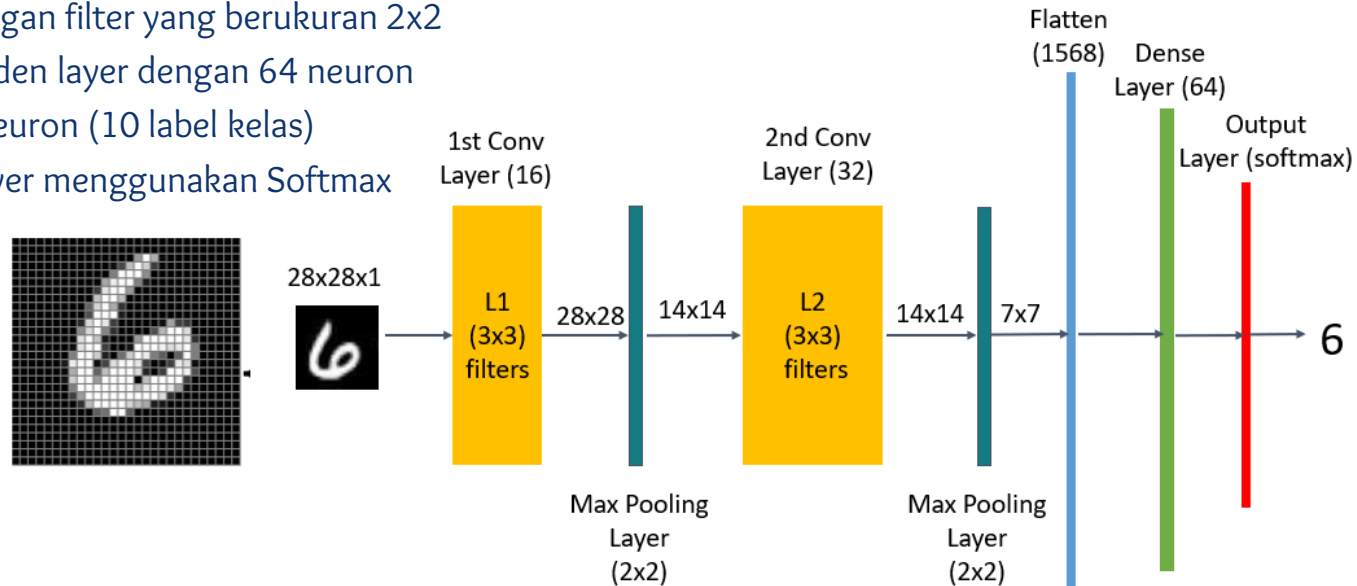
# Fully Connected Layer

- *Fully connected layer* merupakan arsitektur *Multi-layer ANN*
- *Feature map* hasil dari proses konvolusi dan pooling, selanjutnya dilakukan proses *flatten* yaitu merubah matrix menjadi vektor sebagai inputan *fully connected layer*



# Contoh Implementasi arsitektur CNN pada pengenalan angka

- Citra input 28x28
- Layer pertama konvolusi dengan 16 filter yang berukuran 3x3
- Layer kedua Max pooling dengan filter yang berukuran 2x2
- Layer ketiga konvolusi dengan 32 filter yang berukuran 3x3
- Layer keempat Max pooling dengan filter yang berukuran 2x2
- Layer Flatten dilanjutkan 1 hidden layer dengan 64 neuron
- Output layer mempunyai 10 neuron (10 label kelas)
- Fungsi aktivasi pada output layer menggunakan Softmax



# Contoh implementasi arsitektur CNN pada pengenalan angka

## 1. Define Model CNN

```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model2 = Sequential()
model2.add(Conv2D(16,(3,3),activation='relu',input_shape=(28,28,1),padding='same'))
model2.add(MaxPooling2D(2,2))
model2.add(Conv2D(32,(3,3),activation='relu',padding='same'))
model2.add(MaxPooling2D(2,2))
model2.add(Flatten())
model2.add(Dense(64,activation='relu'))
model2.add(Dense(10,activation='softmax'))
model2.summary()
```

Model: "sequential\_1"

| Layer (type)                   | Output Shape       | Param # |
|--------------------------------|--------------------|---------|
| conv2d (Conv2D)                | (None, 28, 28, 16) | 160     |
| max_pooling2d (MaxPooling2D)   | (None, 14, 14, 16) | 0       |
| conv2d_1 (Conv2D)              | (None, 14, 14, 32) | 4640    |
| max_pooling2d_1 (MaxPooling2D) | (None, 7, 7, 32)   | 0       |
| flatten_1 (Flatten)            | (None, 1568)       | 0       |
| dense_2 (Dense)                | (None, 64)         | 100416  |
| dense_3 (Dense)                | (None, 10)         | 650     |

Total params: 105,866

Trainable params: 105,866

Non-trainable params: 0

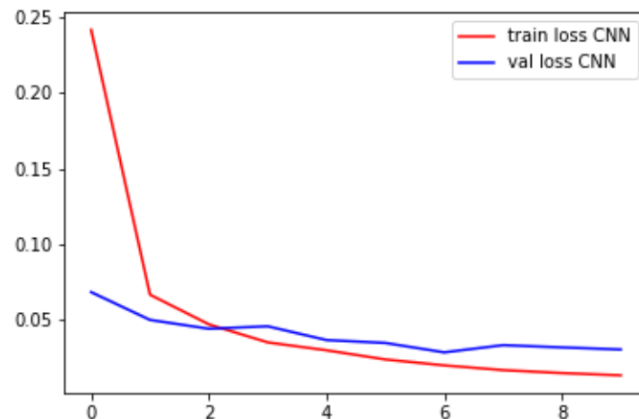
# Contoh implementasi arsitektur CNN pada pengenalan angka

## 2. Compile Model, Fit Model, Save Model, dan Evaluasi Model CNN

```
model. Compile(optimizer='adam',loss='categorical_crossentropy',metrics=['acc'])  
history =  
model.fit(X_train,y_train,epochs=10,batch_size=100,validation_data=(X_test,y_test))  
model.save('my_model2.h5')  
model.evaluate(X_test,y_test)
```

313/313 [=====] - 1s 3ms/step - loss: 0.0304 - acc: 0.9897

[0.03035075031220913, 0.9897000193595886]



# Contoh implementasi arsitektur CNN pada pengenalan angka

## 3. Load Model CNN dan Prediction

```
import numpy as np
from keras.models import load_model

model_simpan2 = load_model('my_model2.h5')
pred = model_simpan2.predict(X_test)
print('label actual:', np.argmax(y_test[30]))
print('label prediction:', np.argmax(pred[30]))
```

---

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



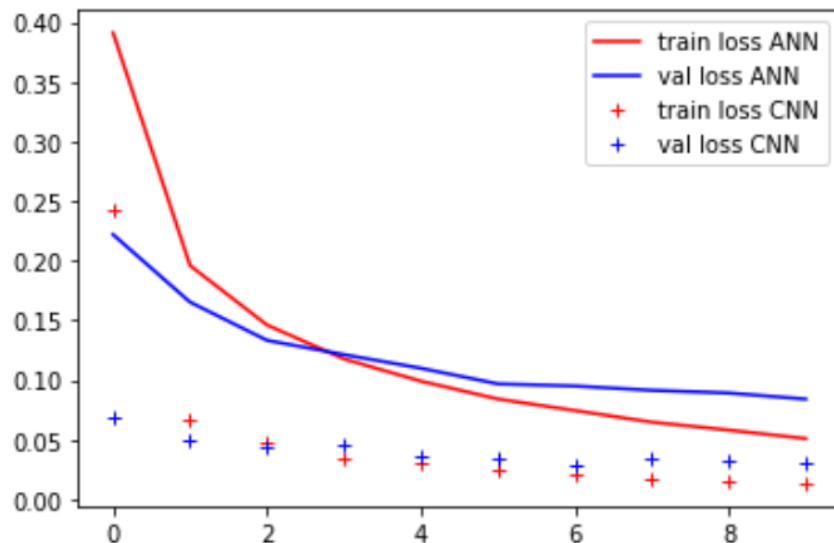
## Contoh Implementasi pada Pengenalan Angka

```
import matplotlib.pyplot as plt

epochs = range(10)
loss1 = history1.history['loss']
val_loss1 = history1.history['val_loss']
plt.plot(epochs, loss1, 'r', label='train loss ANN')
plt.plot(epochs, val_loss1, 'b', label='val loss ANN')

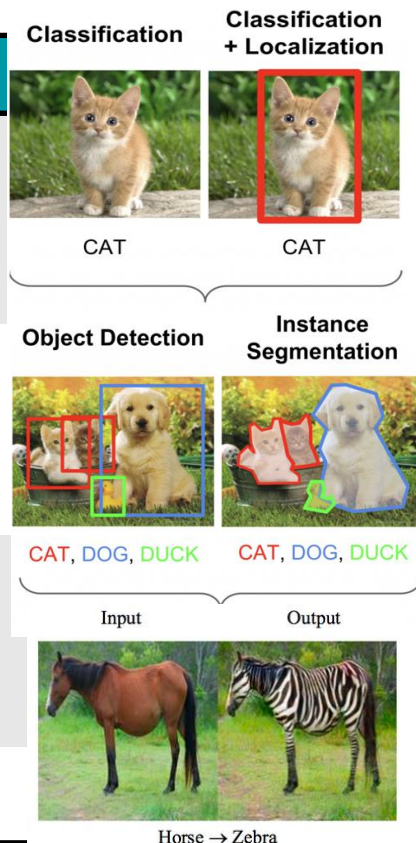
loss2 = history2.history['loss']
val_loss2 = history2.history['val_loss']
plt.plot(epochs, loss2, 'r+', label='train loss CNN')
plt.plot(epochs, val_loss2, 'b+', label='val loss CNN')
plt.legend()
```

Perbandingan Loss dari Model CNN dan model ANN



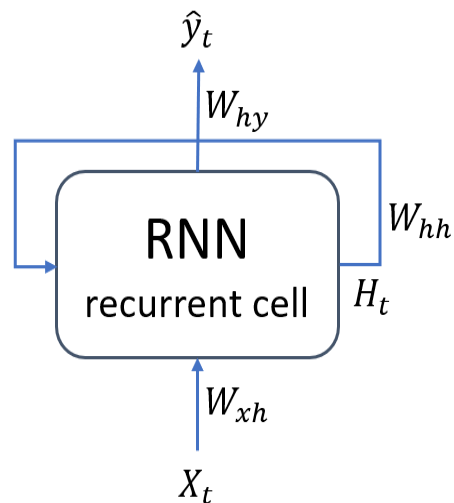
# Varian dari Arsitektur CNN dan Tipe Aplikasinya

| Aplikasi                         | Arsitektur CNN   |
|----------------------------------|--|
| Image Classification             | <ul style="list-style-type: none"> <li>LeNet-5 (1998)</li> <li>AlexNet (2012)</li> <li>GoogLeNet/Inception (2014)</li> <li>VGGNet (2014)</li> <li>ResNet (2015)</li> </ul>   |
| Object Detection                 | <ul style="list-style-type: none"> <li>R-CNN (2013)</li> <li>Fast R-CNN (2014)</li> <li>Faster R-CNN (2015)</li> <li>Single Shot Detector (SSD) (2016)</li> <li>YOLO (2016), YOLOv3 (2018), YOLOv4 (2020), YOLOv5 (2020)</li> </ul>                    |
| Semantic (Instance) Segmentation | <ul style="list-style-type: none"> <li>Fully Convolutional Network (FCN) (2015)</li> <li>U-Net (2015)</li> <li>Feature Pyramid Network (FPN) (2016)</li> <li>Mask R-CNN (2017)</li> <li>DeepLab (2016), DeepLabv3 (2017), DeepLabv3+ (2018)</li> </ul> |
| Generative model                 | <ul style="list-style-type: none"> <li>Autoencoders, Variational Autoencoders (VAE)</li> <li>Generative Adversarial Network (GAN)</li> </ul>   |



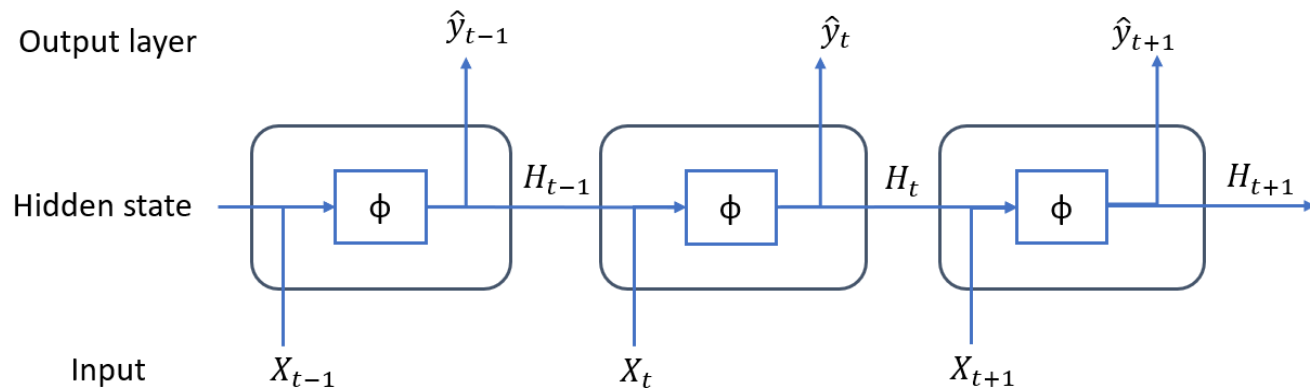
# Recurrent Neural Network

Recurrent Neural Network (RNN) adalah salah satu arsitektur ANN yang mampu merepresentasikan data *sequential* misalnya teks, dna, suara, *time series*, dan sebagainya

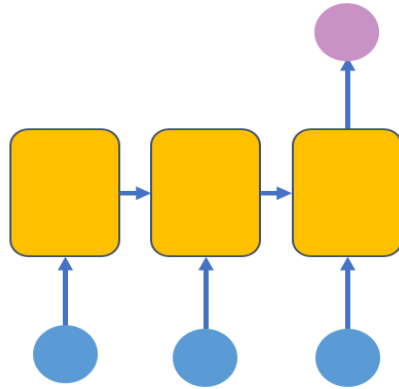


$$\hat{y}_t = H_t W_{hy} + b_y$$

$$H_t = \phi(X_t W_{xh} + H_{t-1} W_{hh} + b_h)$$



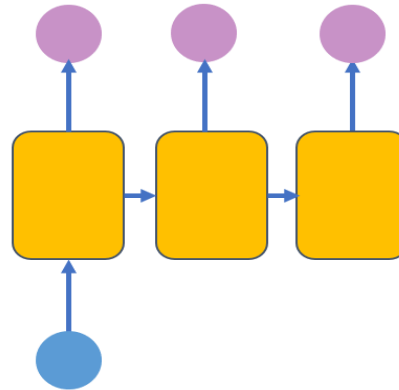
# Tipe arsitektur RNN dan aplikasinya



Many to One

Applications:

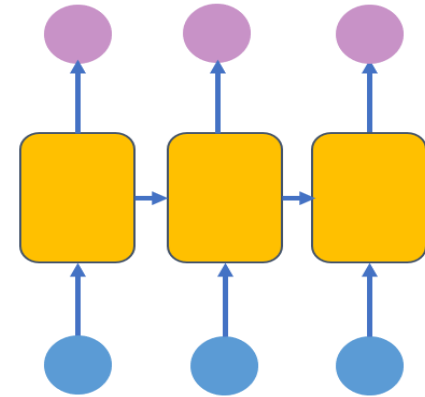
- Sentiment classification
- Opinion mining
- Speech recognition
- Automated answer scoring



One to Many

Applications:

- Image captioning
- Text generation

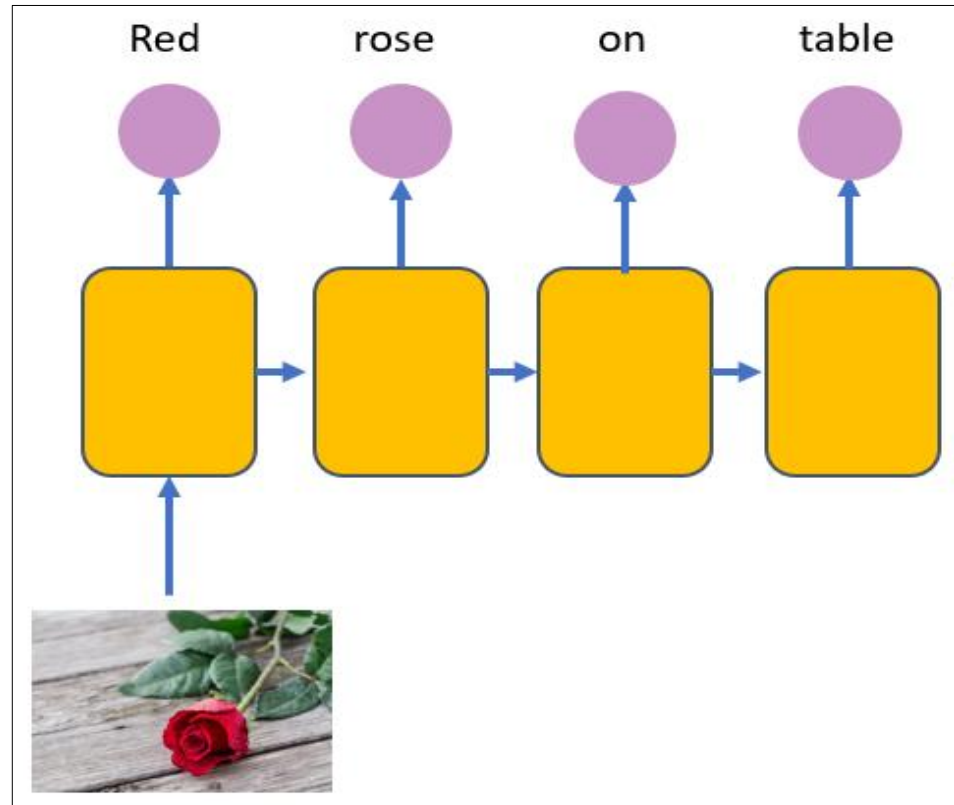
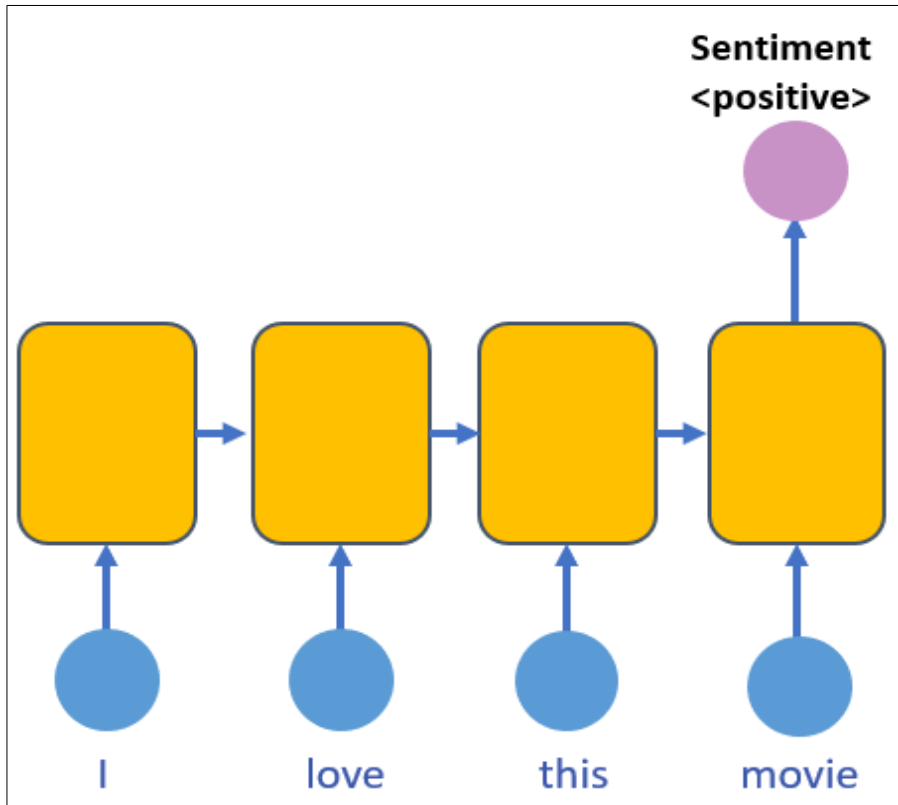


Many to Many

Applications:

- Translation
- Forecasting
- Chatbot
- Music generation

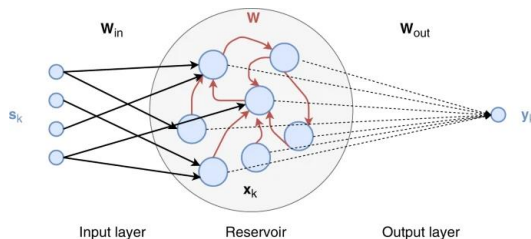
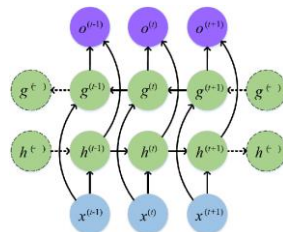
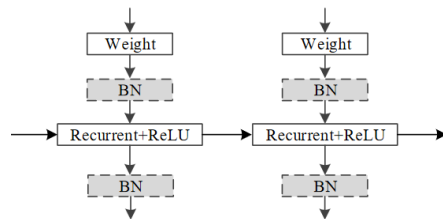
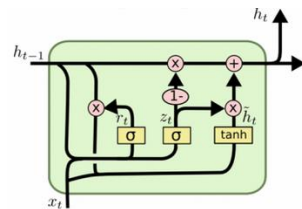
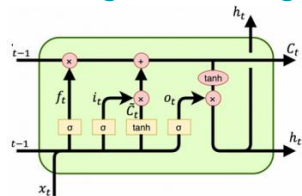
# Contoh Aplikasi: *Sentiment classification & Image captioning*



# Varian Arsitektur RNN

<http://dprogrammer.org/rnn-lstm-gru>

- **Long Short-Term Memory (LSTM)**: merupakan salah satu jenis arsitektur RNN yang terdiri dari beberapa unit yaitu input gate, output gate, dan forget gate
- **Gate Recurrent Unit (GRU)**: merupakan simplifikasi dari arsitektur LSTM dengan menggabungkan input gate dan forget gate sehingga jumlah parameter lebih sedikit
- **Independently RNN (IndRNN)**: arsitektur RNN dimana setiap neuron dalam satu layer independen dari yang lain
- **Bi-directional RNN**: merupakan arsitektur RNN menghubungkan dua hidden layer dari arah yang berlawanan ke output yang sama.
- **Echo State Network (ESN)**: ide dasar ESN adalah untuk membuat jaringan berulang yang terhubung secara acak, yang disebut reservoir



# Summary

## FCN

Data numerik

Jumlah hidden layer  
sesuai kompleksitas  
permasalahan

Klasifikasi dan regresi

## CNN

Data citra, video

Convolution & Pooling  
layer

Klasifikasi, deteksi obyek,  
*instance segmentation*,  
generate citra sintetis

## RNN

Data text, signal, suara,  
time-series

Konsep recurrent dan  
memperhatikan urutan  
input

Klasifikasi, regresi,  
*generate text, translation*

# 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



# Assignments

- Implementasi metode *Artificial Neural Network* (ANN) pada permasalahan klasifikasi
  - a. Download dataset Hepatitis C pada link berikut  
<https://archive.ics.uci.edu/ml/datasets/HCV+data>
  - b. Membagi dataset menjadi 3 bagian yaitu data train, data validasi, dan data uji
  - c. Merancang dan membangun model ANN
  - d. Lakukan tuning parameter agar menghasilkan model yang terbaik
  - e. Menampilkan grafik loss train dan validasi dari hasil pembangunan model
  - f. Menampilkan hasil confusion matrix dan akurasi dari data uji

# Pembuat Modul

Dr. Eng. Chastine Fatichah, S.Kom, M.Kom  
Institut Teknologi Sepuluh Nopember  
email: [chastine@if.its.ac.id](mailto:chastine@if.its.ac.id)

#JADIJAGOANDIGITAL  
**TERIMA KASIH**



digitalent.kominfo



DTS\_kominfo



digitalent.kominfo



digital talent scholarship