FACULTY OF ENGINEERING
AT SRI RACHA
DEPARMENT OF COMPUTER ENGINEERING

03603351 วิทยาศาสตร์ข้อมูลเบื้องต้น

Introduction to Data Science

เครือข่ายประสาทเทียม

(Artificial Neural Network)

ผศ.ดร. กุลวดี สมบูรณ์วิวัฒน์

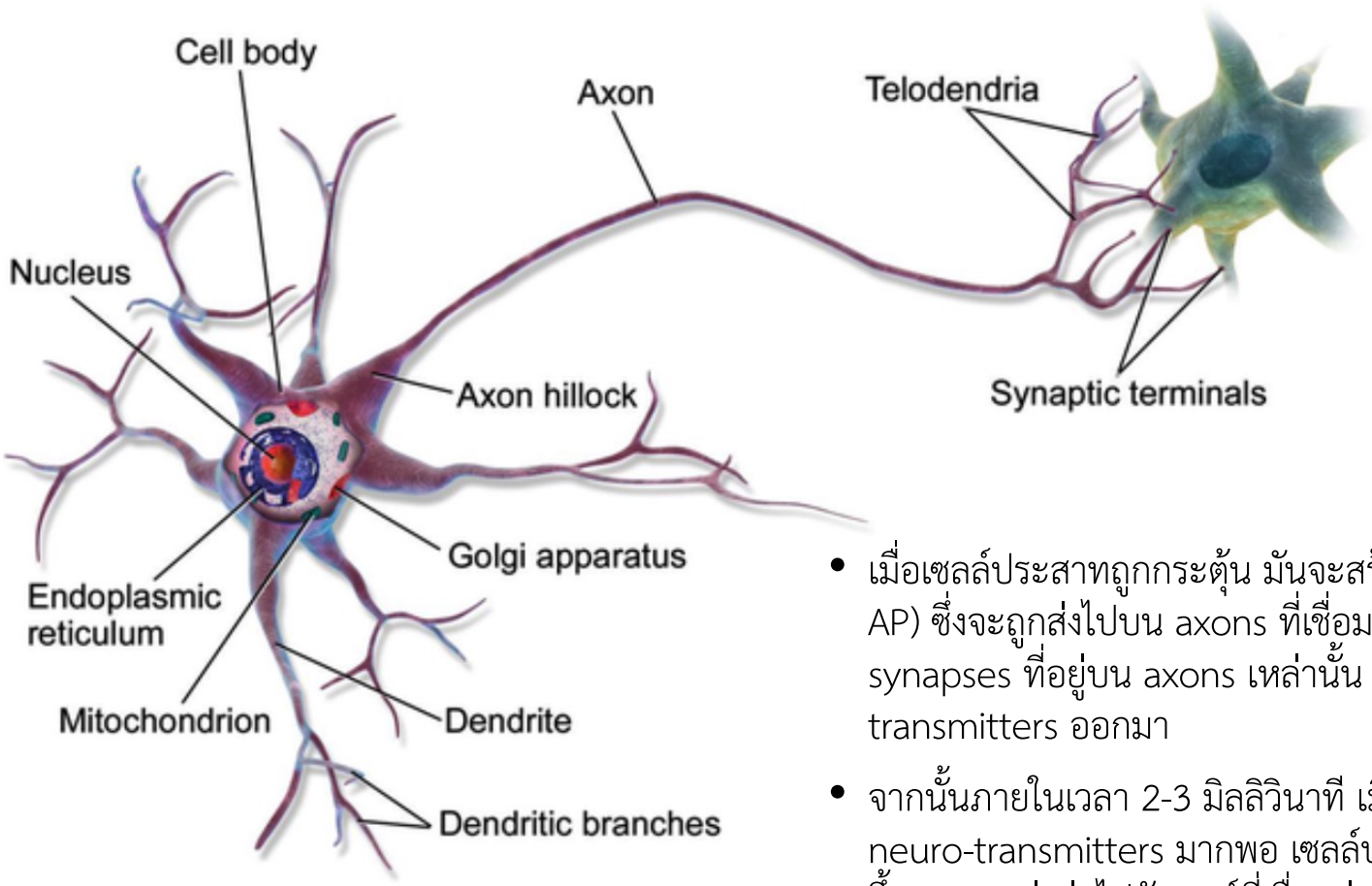
kulwadee@eng.src.ku.ac.th

ภาคการศึกษาที่ 1 ปี 2565

เครือข่ายประสาทเทียม (artificial neural networks: ANN)

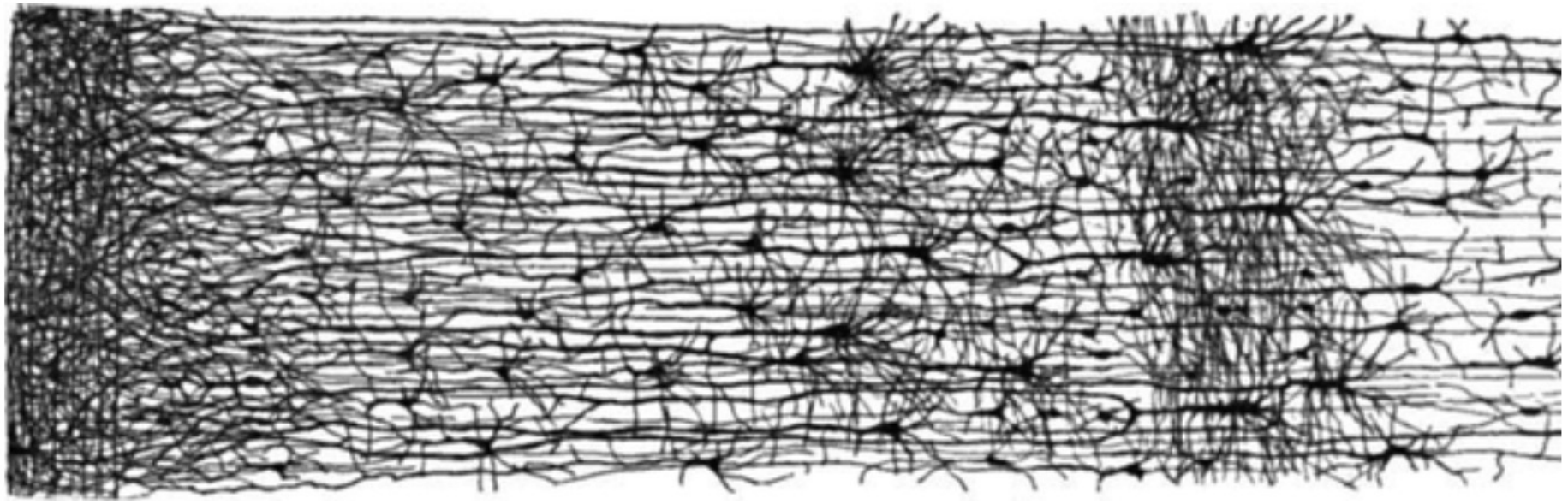
- สถาปัตยกรรมเครือข่ายประสาทเทียมถูกเสนอครั้งแรกในปี 1943 โดย McCulloch และ Pitts
- ในยุค 1960s หลังจากที่นักวิจัยไม่สามารถสร้างเครือข่ายประสาทเทียมที่ชาญฉลาดได้ตามที่ถูกคาดหวังไว้ การพัฒนาของเครือข่ายประสาทเทียมจึงหยุดชะงัก (AI winter)
- จนกระทั่งมาถึงยุค 2000s ความสนใจต่อเครือข่ายประสาทเทียมก็กลับมาอีกครั้ง เนื่องจาก
  - ความก้าวหน้าของอัลกอริทึมสำหรับการเรียนรู้เชิงลึก
  - ข้อมูลปริมาณมหาศาล (Big Data) สำหรับใช้ในการเทรน ANN
  - การประมวลผลสมรรถนะสูง ด้วย GPUs และ Cloud computing

เซลล์ประสาท (Biological Neurons)



- เมื่อเซลล์ประสาทถูกกระตุ้น มันจะสร้างสัญญาณไฟฟ้า (action pontentials: AP) ซึ่งจะถูกส่งไปบน axons ที่เชื่อมต่อไปยังเซลล์ประสาทอื่น ๆ ทำให้ synapses ที่อยู่บน axons เหล่านั้น ปล่อยสัญญาณเคมี เรียกว่า neuro-transmitters ออกมา
- จากนั้นภายในเวลา 2-3 มิลลิวินาที เมื่อเซลล์ประสาทปลายทางได้รับปริมาณ neuro-transmitters มากพอ เซลล์ประสาทปลายทางก็จะสร้างสัญญาณ AP ขึ้นมา และส่งต่อไปยังเซลล์ที่เชื่อมต่ออยู่กับมันต่อไป

เซลล์ประสาท (Biological Neurons)



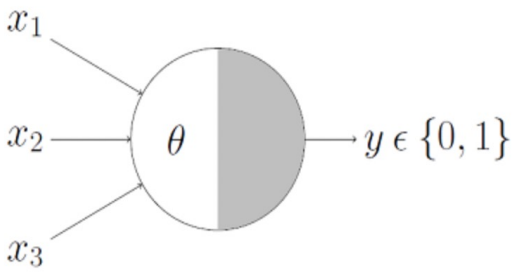
Drawing of a cortical lamination by S. Ramon y Cajal
(https://en.wikipedia.org/wiki/Cerebral_cortex)

- การทำงานของเซลล์ประสาทแต่ละเซลล์ไม่สลับซับซ้อน แต่เนื่องจากในสมองของสิ่งมีชีวิต มีเซลล์ประสาทเหล่านี้อยู่กว่าพันล้านเซลล์ แต่ละเซลล์เชื่อมต่อกับเซลล์ประสาทอื่นอีกกว่าพันเซลล์ เกิดเป็นเครือข่ายเซลล์ประสาท ซึ่งสามารถทำงานที่สลับซับซ้อนได้

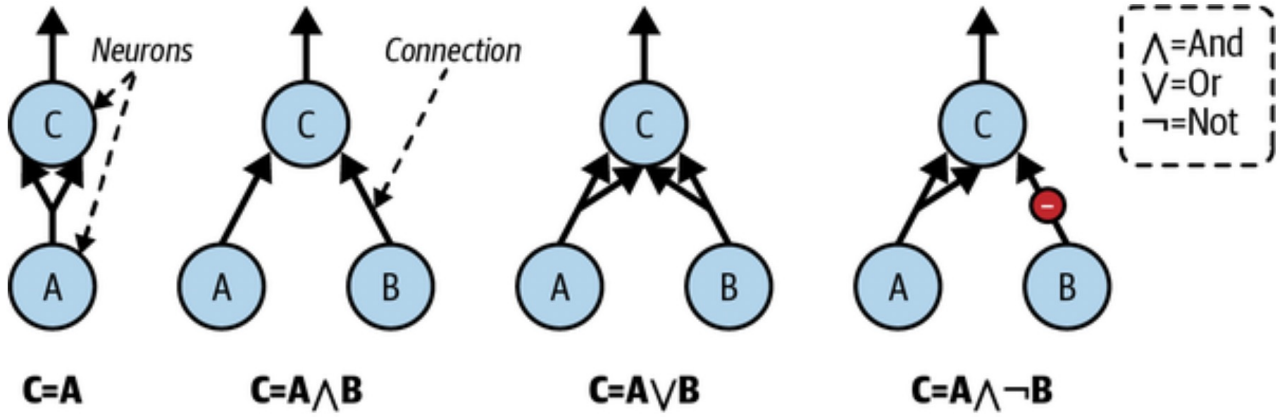
เครือข่ายประสาทเทียม ที่สร้างจาก M-P Neuron

- ในปี 1943, McCulloch และ Pitts ได้แสดงว่า เราสามารถสร้างเครือข่ายประสาทเทียมที่สามารถทำการคำนวณเชิงตรรกะ เช่น IDENTITY, AND, OR ได้

องค์ประกอบพื้นฐาน คือ
นิวรอนเทียม M-P Neuron



เครือข่ายประสาทเทียมสำหรับคำนวณฟังก์ชันเชิงตรรกะ



เครือข่ายประสาทเทียม Perceptron

- คิดค้นขึ้นในปี 1957 โดย Frank Rosenblatt

องค์ประกอบพื้นฐาน คือ นิวรอนเทียม Threshold Logic Unit (TLU)

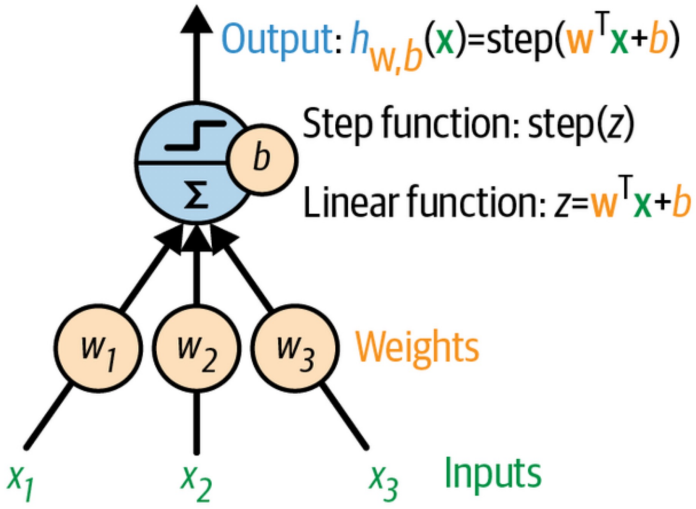


Figure 10-4. TLU: an artificial neuron that computes a weighted sum of its inputs $\mathbf{w}^T \mathbf{x}$, plus a bias term b, then applies a step function

$$\text{heaviside}\,(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}\,(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

เครือข่ายประสาทเทียม Perceptron

- Perceptron ประกอบด้วย TLUs ตั้งแต่หนึ่งหน่วยขึ้นไป จัดเรียงกันแบบลำดับชั้น (layer) จำนวน 1 เลเยอร์
- TLUs ทุกตัวจะเชื่อมต่อกับสัญญาณอินพุทจากอินพุทเลเยอร์ (input layer)
- เลเยอร์ของ TLUs เรียกว่า เอาท์พุทเลเยอร์ (output layer)

องค์ประกอบพื้นฐาน คือ นิวรอนเทียม Threshold Logic Unit (TLU)

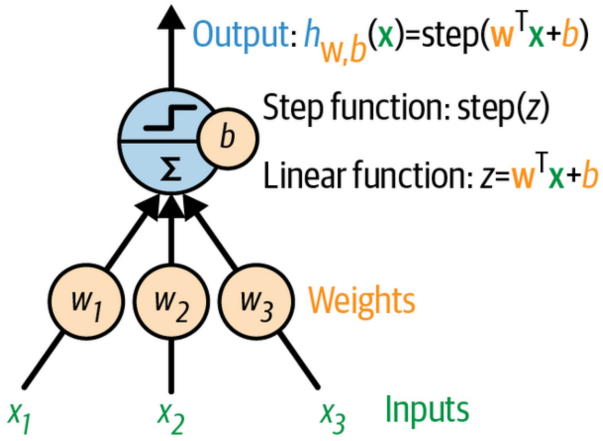
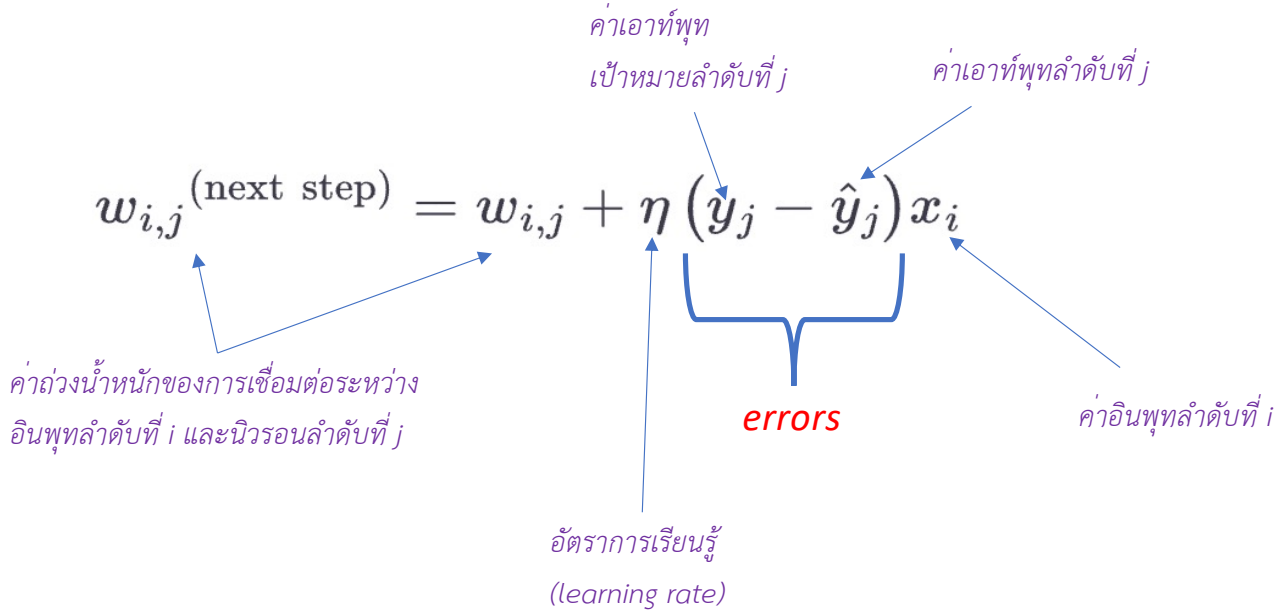


Figure 10-4. TLU: an artificial neuron that computes a weighted sum of its inputs $\mathbf{w}^T \mathbf{x}$, plus a bias term b, then applies a step function

$$\text{heaviside}\,(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases} \qquad \text{sgn}\,(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

เครือข่ายประสาทเทียม Perceptron

- Rosenblatt เสนออัลกอริทึมสำหรับเทรน Perceptron ซึ่งได้รับแรงบันดาลใจมาจาก Hebb's rule (The Organization of Behavior, 1949) ซึ่งกล่าวไว้ว่าการเชื่อมต่อระหว่างนิวรอนสองนิวรอน จะเข้มแข็งมากขึ้นเมื่อมันถูกกระตุ้นพร้อมกันบ่อย ๆ
- Perceptron Learning Rule: เสริมกำลังของการเชื่อมต่อระหว่างนิวรอนที่ช่วยลด error ของเอาท์พุท

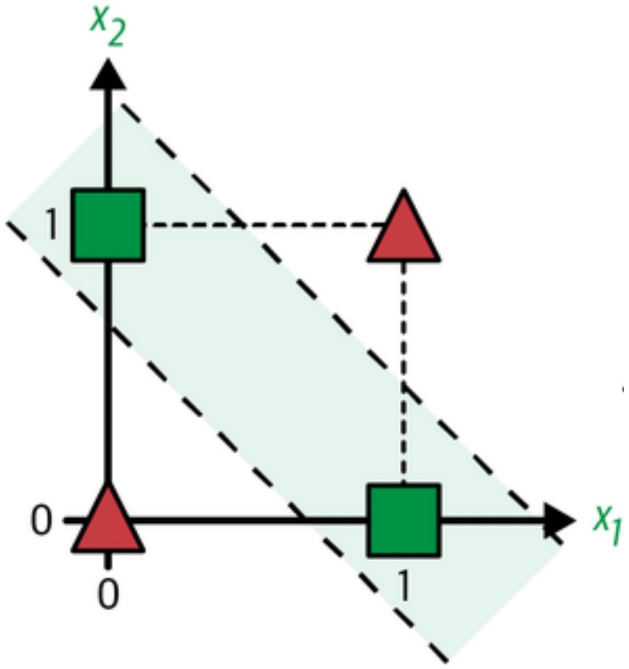


สร้าง Perceptron ด้วย scikit-learn

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris(as_frame=True)
X = iris.data[["petal length (cm)", "petal width (cm)"]].values
y = (iris.target == 0)

per_clf = Perceptron(random_state=42)
per_clf.fit(X, y)

X_new = [[2, 0.5], [3, 1]]
y_pred = per_clf.predict(X_new)
print(y_pred)
```

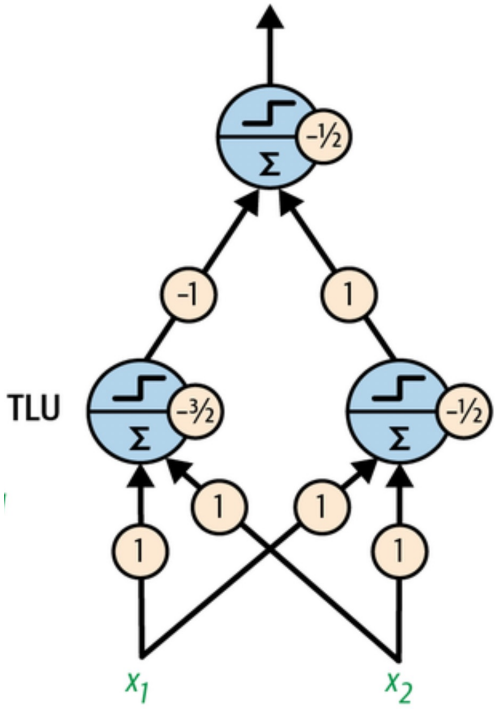
เครือข่ายประสาทเทียม Perceptron

- ในปี 1969 Marvin Minsky และ Seymour Papert ได้ชี้ถึงข้อจำกัดหลายอย่างของ Perceptron โดยเฉพาะอย่างยิ่ง การที่ Perceptron ไม่สามารถแก้ปัญหาการจำแนกประเภทที่ง่ายอย่าง XOR ได้
- ข้อจำกัดดังกล่าวทำให้นักวิจัยปัญญาประดิษฐ์ส่วนใหญ่ หันไปให้ความสนใจกับแนวทางการแก้ปัญหาด้วยวิธีการทางตรรกะ และการค้นหาแทน



เครือข่ายประสาทเทียม Multilayer Perceptron (MLP)

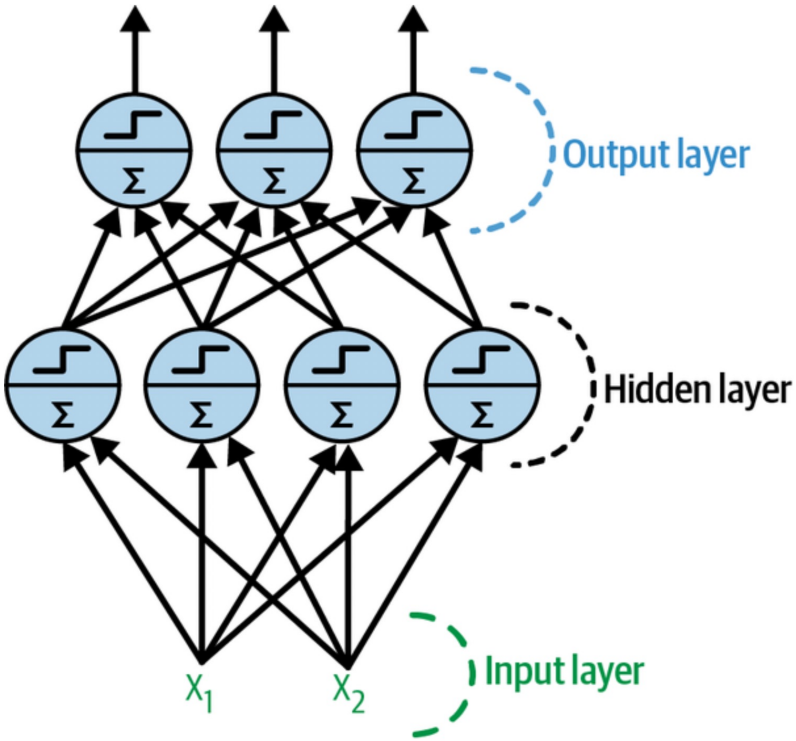
- ต่อมานักวิจัยพบว่าการนำ perceptron มาเรียงกันเป็นลำดับชั้นหลาย ๆ เลเยอร์ จะทำให้เครือข่ายประสาทเทียมที่ได้มีความสามารถมากขึ้น และสามารถแก้ปัญหา XOR classification problem ได้



เครือข่ายประสาทเทียม Multilayer Perceptron (MLP)

MLP ประกอบด้วย

- input layer จำนวน 1 เลเยอร์
- hidden layer ตั้งแต่ 1 เลเยอร์ขึ้นไป (สร้างขึ้นจาก TLUs)
    - จากจำนวน hidden layer มีความลึกมาก ๆ จะเรียกเครือข่ายประสาทเทียมที่ได้ว่า **Deep Neural Network (DNN)**
- output layer จำนวน 1 เลเยอร์ (สร้างขึ้นจาก TLUs) 2



Backpropagation Algorithm

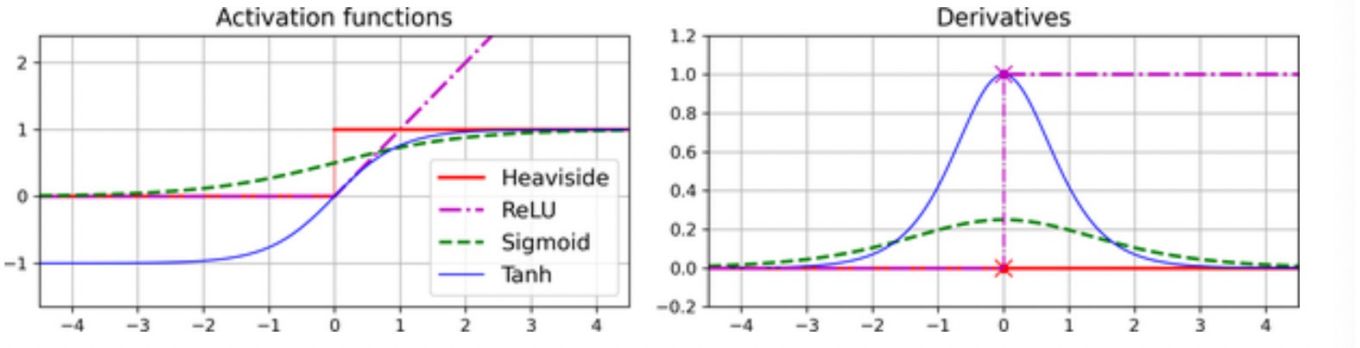
อัลกอริทึมสำหรับเทรน MLP ที่เป็นที่นิยมในปัจจุบัน คือ **backpropagation algorithm (DE Rumelhart, GE Hinton, RJ Williams, 1985)**

1. กำหนดค่าตั้งต้น (เช่น 0) ให้กับค่าถ่วงน้ำหนักของ connection ทุกเส้นในเครือข่าย
2. วนรอบเป็นจำนวน *e* ครั้ง (แต่ละรอบเรียกว่า Epoch) เพื่อเทรนเครือข่าย MLP
   - สลับตำแหน่ง (shuffle) ของข้อมูลในชุดข้อมูล
   - วนรอบเป็นจำนวน *m* ครั้งเพื่อสร้าง mini-batch
     - สุ่มเลือกตัวอย่างข้อมูลแบบไม่ใส่คืน (random sampling without replacement) ขึ้นมาจำนวน 1 ตัวอย่าง เพื่อรวมเข้าไว้ใน mini-batch ของ Epoch ปัจจุบัน
   - <u>Forward Pass:</u>
     - ป้อนอินพุทให้กับเครือข่าย MLP แล้วคำนวณเอาท์พุทของแต่ละเลเยอร์ ทุกเลเยอร์ในเครือข่าย
     - คำนวณค่า error ของเอาท์พุท โดยใช้ loss function
   - <u>Backward Pass:</u>
     - คำนวณปริมาณความคลาดเคลื่อน (error gradients) ที่เกิดขึ้นจาก connection แต่ละเส้นในเครือข่าย MLP โดยใช้ chain rule โดยเริ่มจาก output layer ย้อนกลับไปจนถึง input layer
   - <u>Gradient Descent:</u>
     - อัพเดตค่าถ่วงน้ำหนักของ connection ทุกเส้นในเครือข่ายโดยใช้ error gradients ที่ได้คำนวณไว้

Ref: https://sebastianraschka.com/faq/docs/sgd-methods.html

Activation Functions กับ Backpropagation Algorithm

- เนื่องจาก Backpropagation algorithm ต้องคำนวณหาค่า error gradients โดยใช้กฎ Chain rule ของแคลคูลัส activation function ของนิวรอนจะต้องเป็นฟังก์ชันที่สามารถหาค่า derivative ได้ และควรเป็นฟังก์ชันที่ให้ค่า derivate มากกว่าศูนย์เสมอ
- คณะนักวิจัยนำโดย Rumelhart จึงได้เปลี่ยน สถาปัตยกรรมของ MLP โดยแทนที่ step function ด้วย logistic function (เรียกอีกชื่อว่า sigmoid function)
- นอกเหนือจากฟังก์ชันซิกมอยด์ ดังกล่าว activation function ที่นิยมใช้ในเครือข่ายประสาทเทียม มีหลายฟังก์ชัน เช่น hyperbolic tangent function (tanh) และ rectified linear unit (ReLU)



Regression MLP ด้วย Scikit-learn

```
from sklearn.datasets import fetch_california_housing
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

# Prepare california housing dataset
housing = fetch_california_housing()
X_train_full, X_test, y_train_full, y_test = train_test_split(
    housing.data, housing.target, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(
    X_train_full, y_train_full, random_state=42)

# create pipeline
mlp_reg = MLPRegressor(
             hidden_layer_sizes=[50, 50, 50],
             activation='relu',
             solver='sgd',
             max_iter=200,
             verbose=True,
             random_state=42)
pipeline = make_pipeline(StandardScaler(), mlp_reg)
pipeline.fit(X_train, y_train)

# test the model
y_pred = pipeline.predict(X_test)
rmse = mean_squared_error(y_test, y_pred, squared=False)

print(rmse)
```

ClassificationMLP ด้วย Scikit-learn

```
from sklearn.datasets import fetch_openml
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split

X, y = fetch_openml("mnist_784", version=1, return_X_y=True, as_frame=False)
X = X / 255.0

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

mlp = MLPClassifier(
    hidden_layer_sizes=(36),
    max_iter=200,
    solver='sgd',
    verbose=True,
    random_state=42,
    learning_rate_init=0.2,
)

mlp.fit(X_train, y_train)

print("Training set score: %.1f" % mlp.score(X_train, y_train))
print("Test set score: %.1f" % mlp.score(X_test, y_test))
```

สร้าง Neural Network ด้วย Keras

- Neural Network ที่สร้างขึ้นด้วย Scikit-learn แต่มีฟีเจอร์ที่จำกัด เช่น ไม่รองรับการประมวลผลด้วย GPU และการเรียนรู้เชิงลึก
- Keras คือ high-level deep learning API ที่รวมอยู่ในไลบารี่ TensorFlow
- การติดตั้ง Keras

```
# Requires the latest pip
$ pip install --upgrade pip

# Current stable release for CPU and GPU
$ pip install tensorflow
```

Regression Deep Neural Network on California Housing Price Dataset

```
import tensorflow as tf
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split

tf.random.set_seed(42)

# Prepare california housing dataset
housing = fetch_california_housing()
X_train_full, X_test, y_train_full, y_test = train_test_split(
    housing.data, housing.target, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(
    X_train_full, y_train_full, random_state=42)

# Create the DNN
model = tf.keras.Sequential()
norm_layer = tf.keras.layers.Normalization(input_shape=X_train.shape[1:])
norm_layer.adapt(X_train)
model.add(norm_layer)
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(1))

model.summary()

# Compile the model
model.compile(loss='mse',
              optimizer='sgd',
              metrics=['RootMeanSquaredError'])

# Train the model
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_valid, y_valid))

# Evaluate the model
mse_test, rmse_test = model.evaluate(X_test, y_test)
print("MSE on test set: %.2f" % mse_test)
print("RMSE on test set: %.2f" % rmse_test)

# Use the model to make predictions
X_new = X_test[:3]
y_pred = model.predict(X_new)
for y, y_hat in zip(y_test[:3], y_pred):
    print("True price = %f, Prediction = %f" % (y, y_hat))
```

Classification Deep Neural Network on Fashion MNIST dataset

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

tf.random.set_seed(42)

fashion_mnist = tf.keras.datasets.fashion_mnist.load_data()
# 55_000 samples for training
#  5_000 samples for validation
# 10_000 samples for test set
(X_train_full, y_train_full), (X_test, y_test) = fashion_mnist
X_train, y_train = X_train_full[:-5000], y_train_full[:-5000]
X_valid, y_valid = X_train_full[-5000:], y_train_full[-5000:]

# scale the pixel intensity to the 0-1 range
X_train, X_valid, X_test = X_train / 255. , X_valid / 255., X_test / 255.

class_names = np.array(["T-shirt/top", "Trouser", "Pullover", "Dress", "Coat",
               "Sandal", "Shirt", "Sneaker", "Bag", "Ankle boot"])

# Create the DNN
model = tf.keras.Sequential()
model.add(tf.keras.layers.Input(shape=[28, 28]))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(300, activation='relu'))
model.add(tf.keras.layers.Dense(100, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))

model.summary()

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

```
# Train the model
history = model.fit(X_train, y_train, epochs=30, validation_data=(X_valid, y_valid))

# Plot the learning curve
pd.DataFrame(history.history).plot(
    figsize=(8, 5), xlim=[0, 29], ylim=[0, 1], grid=True,
    xlabel='EPoch', ylabel='Loss',
    style=['r--', 'r--.', 'b-', 'b-*']
)
plt.savefig('fashion_mnist_learning_curve.png')
plt.show()

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print("loss on test set: %.2f" % loss)
print("accuracy on test set: %.2f" % accuracy)

# Use the model to make predictions
X_new = X_test[:3]
y_proba = model.predict(X_new)
y_pred = y_proba.argmax(axis=-1)

y_test_classnames = class_names[y_test[:3]]
y_pred_classnames = class_names[y_pred]
for y, y_hat in zip(y_test_classnames, y_pred_classnames):
    print("True class = %s, Prediction = %s" % (y, y_hat))
```

การบันทึกและการโหลดโมเดล Keras

```
model.save('my_keras_model', save_format='tf')
```

```
model = tf.keras.models.load_model("my_keras_model")

y_pred = model.predict(X_new)
```

สรุป

- เครือข่ายประสาทเทียม เป็นโมเดล machine learning ที่ได้รับแรงบันดาลใจจากการทำงานของสมองของสิ่งมีชีวิต
- สถาปัตยกรรมเครือข่ายประสาทเทียม ประกอบด้วยหน่วยประมวลผล (artificial neurons) ซึ่งถูกจัดเรียงเป็นเลเยอร์
  - นิวรอนแต่ละตัวจะคำนวณค่าเอาท์พุท โดยรวมสัญญาณอินพุทแบบถ่วงน้ำหนัก แล้วนำผลรวมที่ได้ส่งไปยัง activation function เช่น sigmoid function, ReLU, tanh
  - แต่ละเลเยอร์ประกอบด้วยนิวรอนหลาย ๆ ตัว นิวรอนแต่ละตัวจะรับสัญญาณอินพุทจากเลเยอร์ก่อนหน้า และส่งเอาท์พุทของมันไปยังเลเยอร์ถัดไป
  - เลเยอร์ในเครือข่ายประสาทเทียมมี 3 ชนิด คือ input layer, hidden layer, และ output layer
  - หากเครือข่ายประสาทเทียมมีจำนวน hidden layer จำนวนมาก จะเรียกเครือข่ายดังกล่าวว่า deep neural network (DNN)
    - ไม่มีการกำหนดจำนวนไว้ชัดเจน แต่มักถือว่าเป็น DNN เมื่อเครือข่ายมีจำนวน hidden layer ตั้งแต่ 3 เลเยอร์ขึ้นไป
- การเทรนเครือข่ายประสาทเทียมทำได้โดยใช้อัลกอริทึม Backpropagation

สรุป

- Scikit-learn มีคลาสสำหรับสร้าง MLP ชื่อ MLPRegressor และ MLPClassifier
- Keras เป็นไลบารี่สำหรับสร้าง deep neural network
  - ออกแบบโครงสร้างของ DNN โดยใช้ Sequential API
  - คอมไพล์โมเดล โดยกำหนด loss, optimizer, และ metrics
  - เทรนโมเดล โดยเรียกใช้ฟังก์ชัน fit(X_train, y_train)
  - ประเมินประสิทธิภาพ โดยเรียกใช้ฟังก์ชัน evaluate(X_test, y_test