

# Neural Networks

อ. ปรัชญ์ ปิยะวงศ์วิศาล

Pratch Piyawongwisal

# Today

- Decision Tree Homework
- Artificial Neural Networks
  - Perceptron
  - Backpropagation
  - Activation Functions
  - Feedforward Neural Networks
  - Lab: MNIST with TF/Keras

# Homework – Decision Tree

จะ apply decision tree กับชุดข้อมูล student\_major.csv เพื่อทำนาย major ของนักศึกษา

1. แบ่งข้อมูลออกเป็นชุด train, test ในอัตราส่วน 75:25 โดยใช้ `train_test_split` (ตัวอย่างโค้ดในสไลด์ถัดไป)
2. ใช้ข้อมูลชุด train (`X_train, y_train`) ในการ `train/validate` โมเดล decision tree
  - ใช้ `GridSearchCV` เพื่อค้นหาค่า hyperparameter ของ model (เช่น `max_depth, criterion`) ที่ให้ค่า accuracy สูงที่สุด
3. นำ model ที่ประสิทธิภาพสูงที่สุดไปใช้ทำนาย major ของนักศึกษา กับข้อมูลชุดทดสอบ (`X_test, y_test`) และแสดงผลการทำนายทั้งหมด
4. ประเมินประสิทธิภาพของโมเดลที่ได้โดยใช้ `metric` ต่อไปนี้
  - `accuracy score`
  - `confusion matrix (optional)`

\*ใน Jupyter พยายามให้ code ทั้งหมดอยู่ใน cell เดียว (หากจำเป็น สามารถใช้ได้ 4 cell max)

# Homework – Decision Tree

5. จากการทดลองในข้อ 4. ให้

- แสดง line plot ของ accuracy v.s. max\_depth จากการทดลอง ในกรณีที่ใช้ Gini criterion
- แสดง line plot ของ accuracy v.s. max\_depth จากการทดลอง ในกรณีที่ใช้ entropy criterion โดยที่ค่า accuracy ของแต่ละค่า hyperparameter จะถูกเก็บอยู่ใน GridSearchCV.cv\_results\_[“mean\_test\_score”]

6. แสดงแผนผัง decision tree ของโมเดลที่ดีที่สุด

- นำผลจาก export\_graphviz ของโมเดลที่ดีที่สุดไปแสดงบนเว็บ webgraphviz
- capture screen ภาพ decision tree เช่นในไฟล์เดอร์ของ Jupyter
- ใน Jupyter สร้าง cell ล่างสุดให้เป็นแบบ markup แล้วแทรกรูปดังกล่าวด้วย
  - ![caption](path/to/image.png)

# Homework – Decision Tree

วิธีการทำ `train_test_split`

```
from sklearn.model_selection import train_test_split  
  
X = ...  
y = ...  
X_train, X_test, y_train, y_test = train_test_split(X, y,  
test_size=0.25, random_state=33)
```

จากนั้นสามารถนำ `X_train, y_train` ไปใช้ในการฝึก และนำ `X_test, y_test` ไปใช้ในการทำนาย เพื่อวัดประสิทธิภาพได้

# Homework – Decision Tree

## วิธีส่งงาน

- ใน Jupyter Notebook ให้เขียนโค้ดทำ decision tree โดยพยายามให้ห้องทดลองใน cell เดียว (หากจำเป็น สามารถใช้ได้ 4 cell max)
- ทำการ export html โดยไปที่ File -> Download as -> HTML
- rename ชื่อไฟล์เป็น ชื่อ สกุล ภาษาไทย ไม่มีคำนำหน้า
- ส่งไฟล์ html ไปที่ <https://www.dropbox.com/request/e8rvQXyDbJfX323iuljW>
- กรอกชื่อ สกุล ภาษาไทย ไม่ใส่คำนำหน้าชื่อ นาย/นางสาว

# Artificial Neural Networks

# Artificial Neural Network (โกรงข่ายประสาทเทียม)

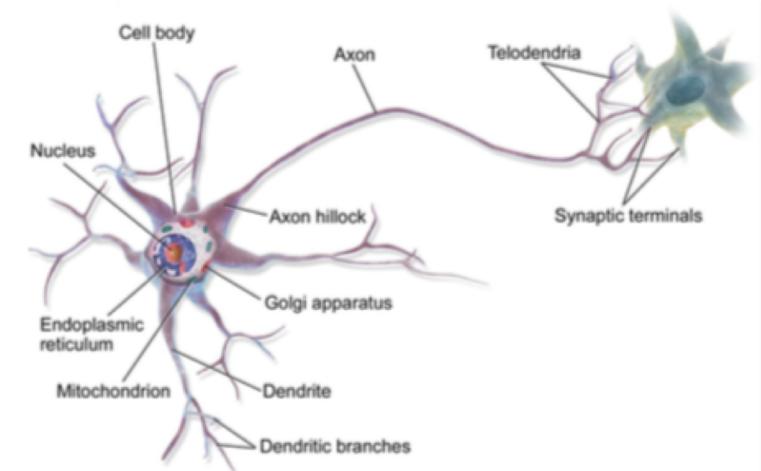
- เป็น supervised learning ใช้ทำ classification เป็นหลัก
- ข้อดี
  - ความแม่นยำสูงมาก
  - สามารถนำไปประยุกต์ใช้กับปัญหาที่มีความซับซ้อนได้มากmany
  - เหมาะกับงานประมวลผลภาพ
- ข้อเสีย
  - ต้องการข้อมูลจำนวนมหาศาล
  - cost function ไม่ convex (มี minima ให้หลายจุด) ดังนั้นการเลือก initialize ค่าพารามิเตอร์เริ่มต้นจึงมีผลต่อค่าตอบสนองท้ายของ gradient descent
  - เป็น black box model ตีความ/อธิบายที่มาที่ไปได้ยาก (ตรงข้ามกับ decision tree)

# Artificial Neural Network (ໂຄຮງຂ່າຍປະສາທເທື່ອນ)

- ตัวอย่าง applications ของ NN
    - Image Classification การจำแนกประเภทรูปภาพ
      - AlexNet: <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
    - Speech Recognition การรู้จำเสียงพูด
      - Apple's Siri: <https://machinelearning.apple.com/2017/10/01/hey-siri.html>
    - Video Recommendation ระบบแนะนำวิดีโอ
      - YouTube: <https://ai.google/research/pubs/pub45530>
    - Games
      - DeepMind's AlphaGo: [https://deepmind.com/documents/119/agz\\_unformatted\\_nature.pdf](https://deepmind.com/documents/119/agz_unformatted_nature.pdf)
      - Marl/O: <https://www.youtube.com/watch?v=qv6UVOQ0F44>
      - Atari game: <https://www.youtube.com/watch?v=V1eYniJ0Rnk>

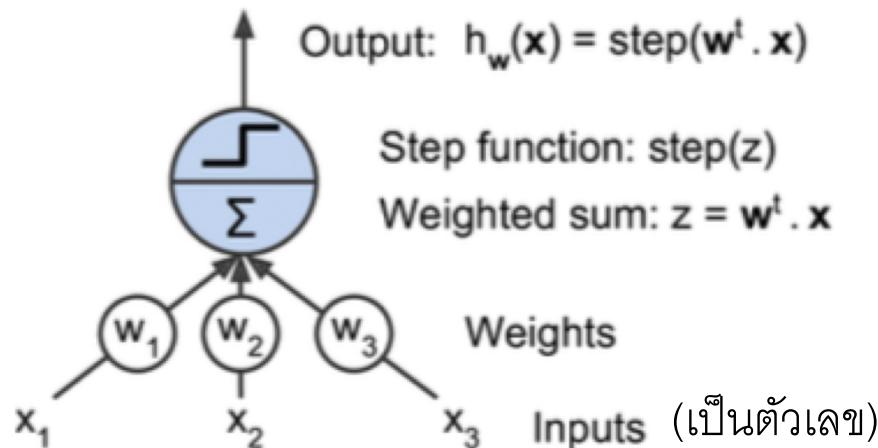
# Origin of Artificial Neural Network

- ได้แรงบันดาลใจจาก **Biological neurons**
- ในช่วงปี 1960s เชื่อกันว่าการสร้าง ANN เพื่อจำลองการคำนวณที่ซับซ้อนของสมองจะนำไปสู่ AI ที่คาดจริง (**Connectionism**)
- แต่ก็ล้มเหลวไปด้วยข้อจำกัดเหล่านี้
  - Lack of computation power
  - Lack of data
  - Lack of efficient training algorithms
- ปัจจุบัน ANN กลับมา **dominate** สาขา Machine Learning อีกรึ้ง เนื่องจากเกิด
  - Fast CPU, GPU
  - Big Data
  - Algorithms: Backpropagation, SGD, dropout, Adam



# Perceptron

- เป็น ANN ที่เรียนง่าย คิดค้นโดย Rosenblatt ปี 1957
- ประกอบด้วย Neuron ที่เป็น Linear Threshold Unit (LTU) ดังรูป



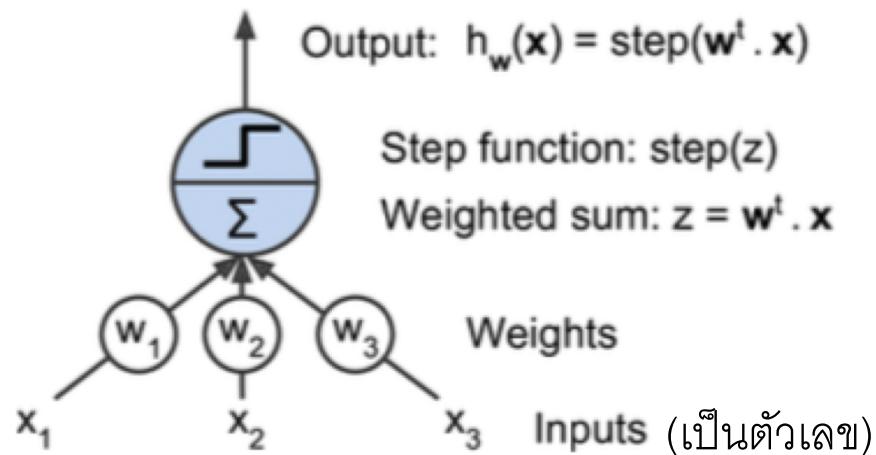
$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

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

- สำหรับ step function อาจใช้เป็นฟังก์ชัน heaviside หรือ sign

# Perceptron

- Single LTU สามารถใช้ทำ **binary classification** (นิ่กถึง Iris) ได้
- สังเกตว่าโมเดลของ LTU คือการหา **linear combination** ของ **input** และนำมา **threshold** เพื่อให้ **output** ทำงานอย่างมากว่าเป็นคลาส + positive หรือ – negative
- คล้าย Logistic Regression

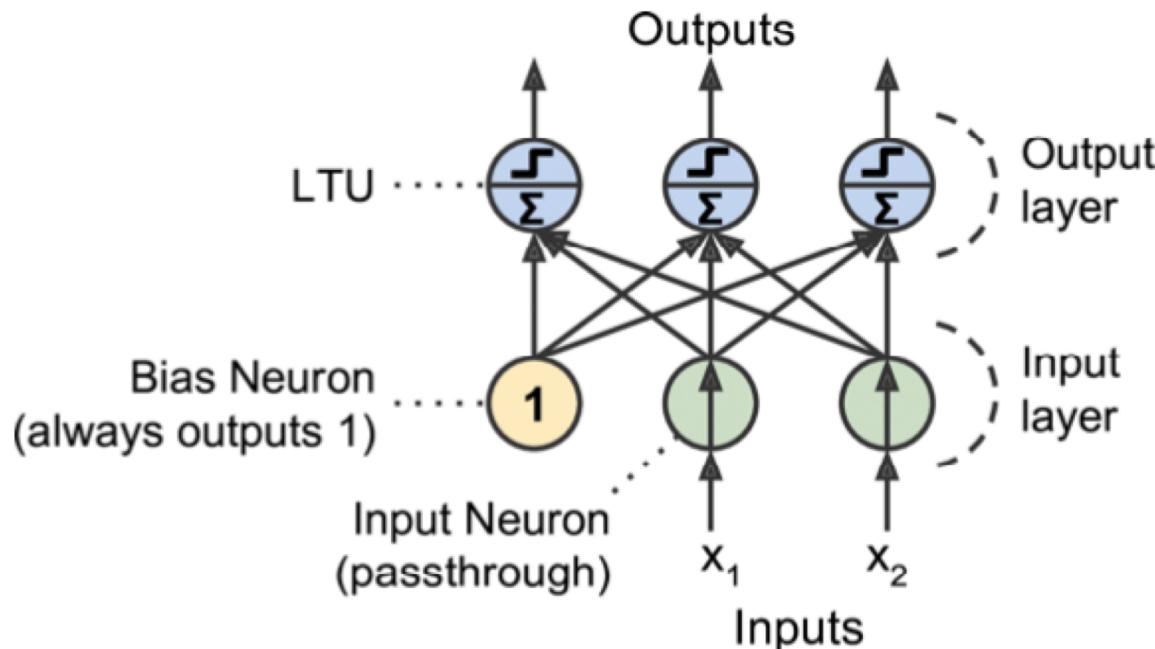


$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

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

# Perceptron

- Perceptron ประกอบด้วย layer ของ LTU เพียง 1 layer
- มี Bias input node ที่มีค่าเป็น 1 เสมอ  
(เพื่อให้ decision boundary ไม่จำเป็นต้องผ่านจุด origin)
- ในภาพเป็น Perceptron ที่มี 3 LTU สามารถใช้ classify ได้ 3 คลาส



# Perceptron - Training

- การ train Perceptron คือการหาค่า **weight** ที่ทำให้โมเดลสามารถทำนายได้แม่นที่สุด
- Rosenblatt ใช้ Hebb's rule: "Cells that fire together, wire together."
- กล่าวคือ ต้องการให้ **weight** มีค่าสูง หาก neuron คุณนั้น **output** ค่าเดียวกันบ่อย ๆ
- จึงเกิดเป็น Perceptron learning rule ดังนี้

*Equation 10-2. Perceptron learning rule (weight update)*

$$w_{i,j}^{(\text{next step})} = w_{i,j} + \eta(\hat{y}_j - y_j)x_i$$

- สังเกตว่า **weight** จะถูก update ก็ต่อเมื่อ  $\hat{y} \neq y$  (ทำนายพลาด) เท่านั้น

# Perceptron Code

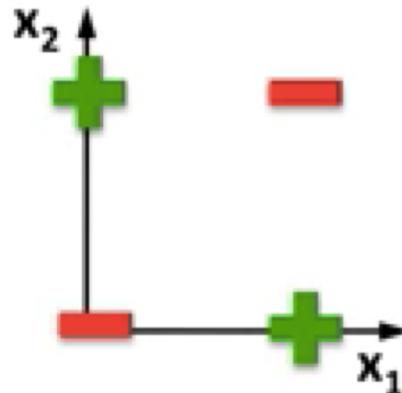
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron

iris = load_iris()
X = iris.data[:, (2, 3)] # petal length, petal width
y = (iris.target == 0).astype(np.int) # Iris Setosa?
per_clf = Perceptron(random_state=42)
per_clf.fit(X, y)
y_pred = per_clf.predict([[2, 0.5]])
```

# Perceptron - Limitations

- ข้อจำกัดของ Perceptron คือไม่สามารถทำ non-linear classification ได้
  - training data ต้องสามารถถูกแบ่งแทนด้วย hyperplane ตรงได้เท่านั้น (linearly separable)
  - เช่น ไม่สามารถแก้ XOR problem ได้

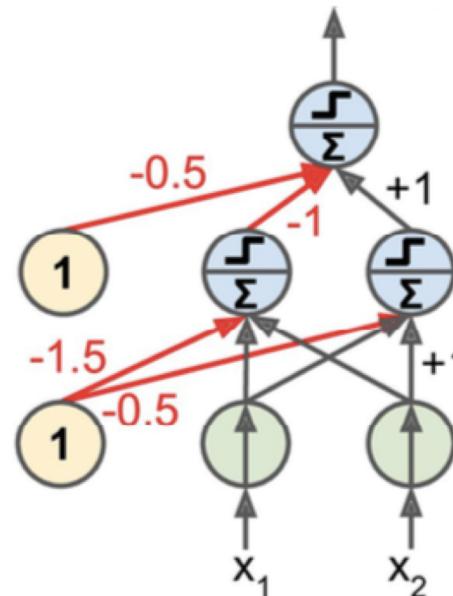
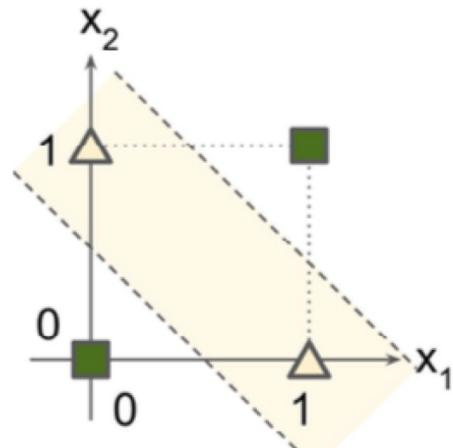
Linear classifiers  
cannot solve this



# Perceptron - Limitations

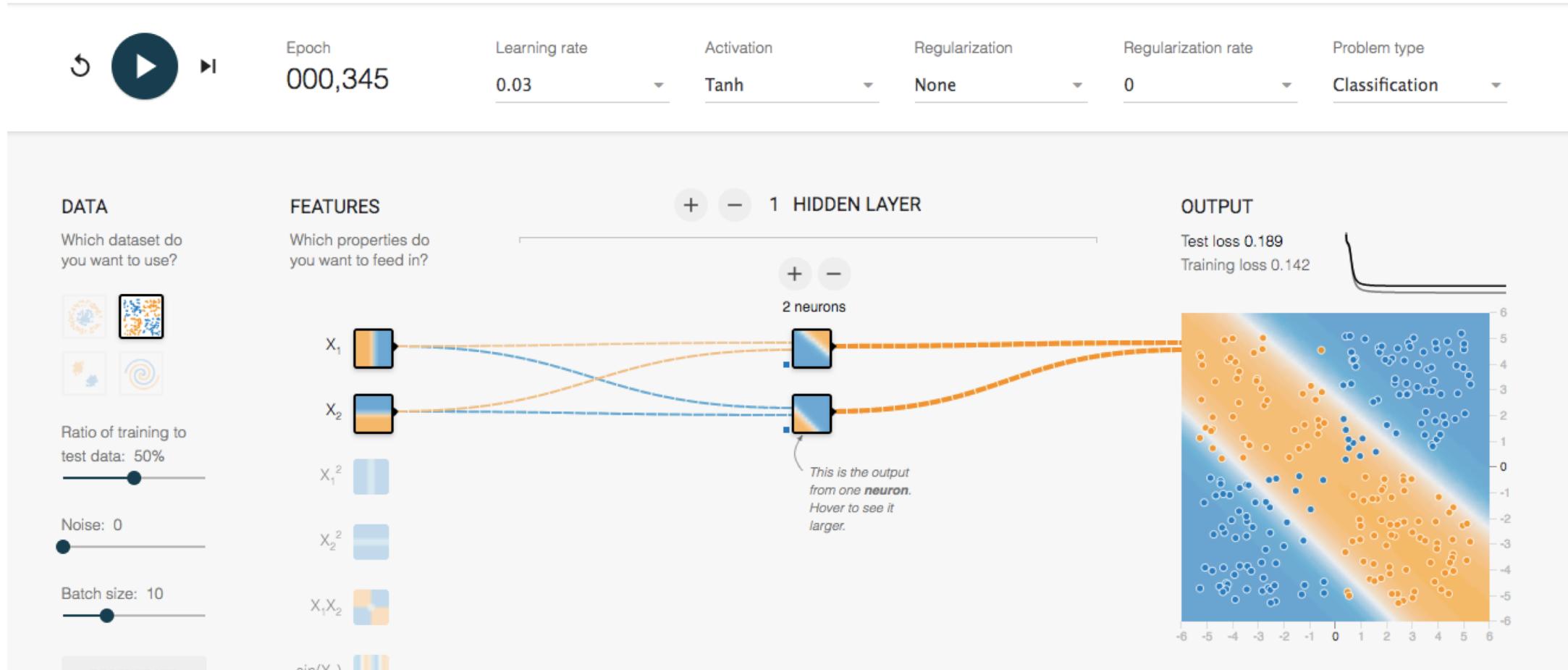
- ข้อจำกัดของ Perceptron คือไม่สามารถทำ non-linear classification ได้
  - training data ต้องสามารถถูกแบ่งแทนด้วย hyperplane ตรงได้เท่านั้น (linearly separable)
  - เช่น ไม่สามารถแก้ XOR problem ได้

**solution:** เพิ่มจำนวน layer (Multilayer Perceptron)



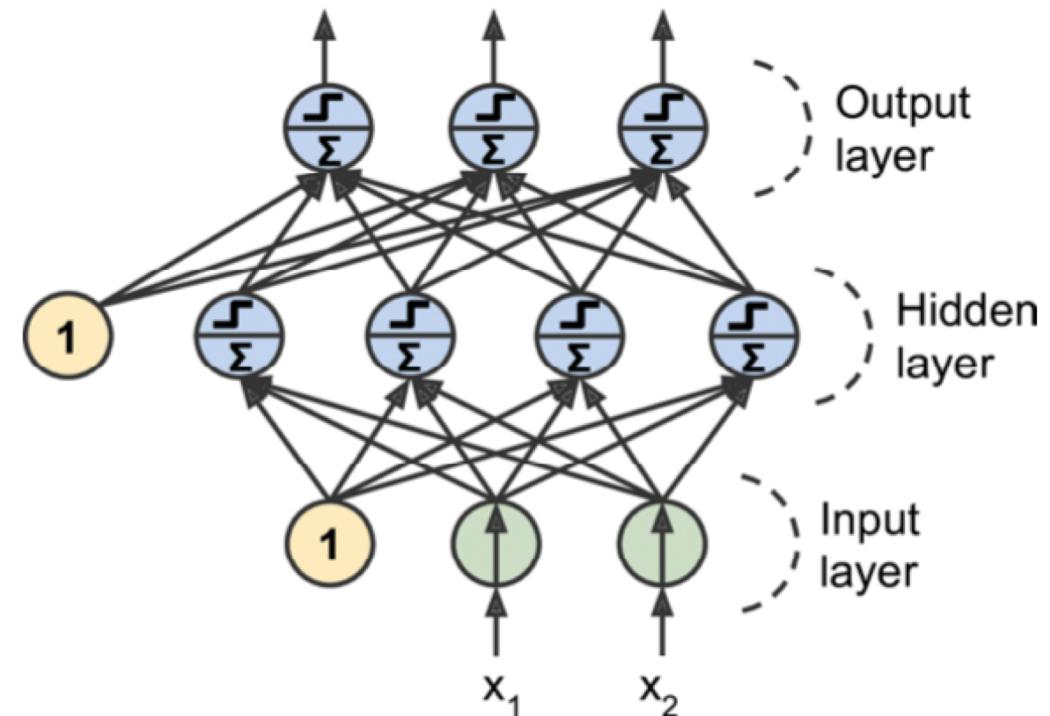
# Visualizing Perceptrons with Tensorflow Playground

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



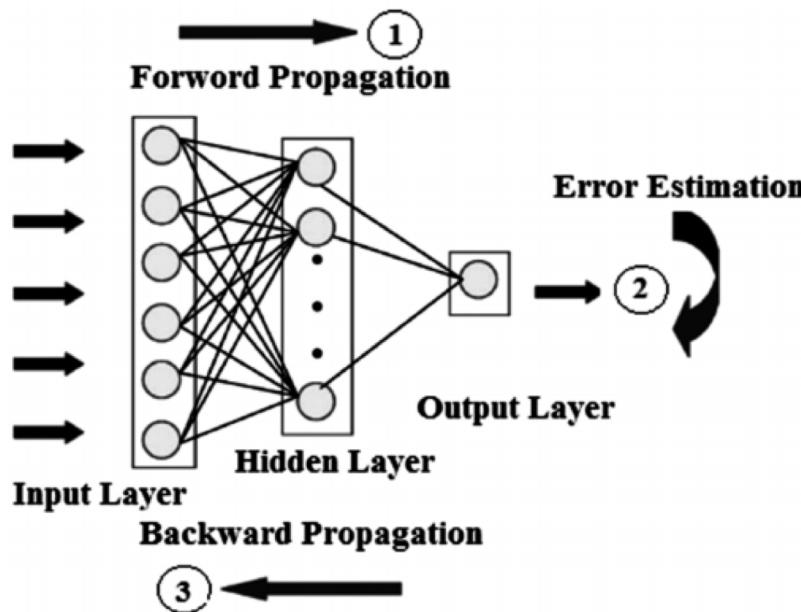
# Multilayer Perceptron (MLP)

- เราเรียก ANN มี 2 hidden layer ขึ้นไปว่า Deep Neural Network
- ในยุคหนึ่ง นักวิจัยตั้งกับการหาวิธี train MLP อยู่หลายปี
- จนกระทั่งปี 1986 Rumelhart ได้ตีพิมพ์อัลกอริทึมที่ปฏิวัติวงการ ชื่อ **Backpropagation**



# Backpropagation

- Backpropagation ทำให้เราสามารถหา gradient ของค่า error/cost ซึ่งประกอบด้วย partial derivative ของ error ต่อแต่ละ weight ได้
  - $\nabla_w Error \rightarrow$  matrix ของ  $\frac{\partial Error}{\partial w_i}$  สำหรับทุก  $w_i$
  - เมื่อคำนวณ gradient ได้ ก็สามารถ train ด้วยวิธี gradient descent ได้ 😊



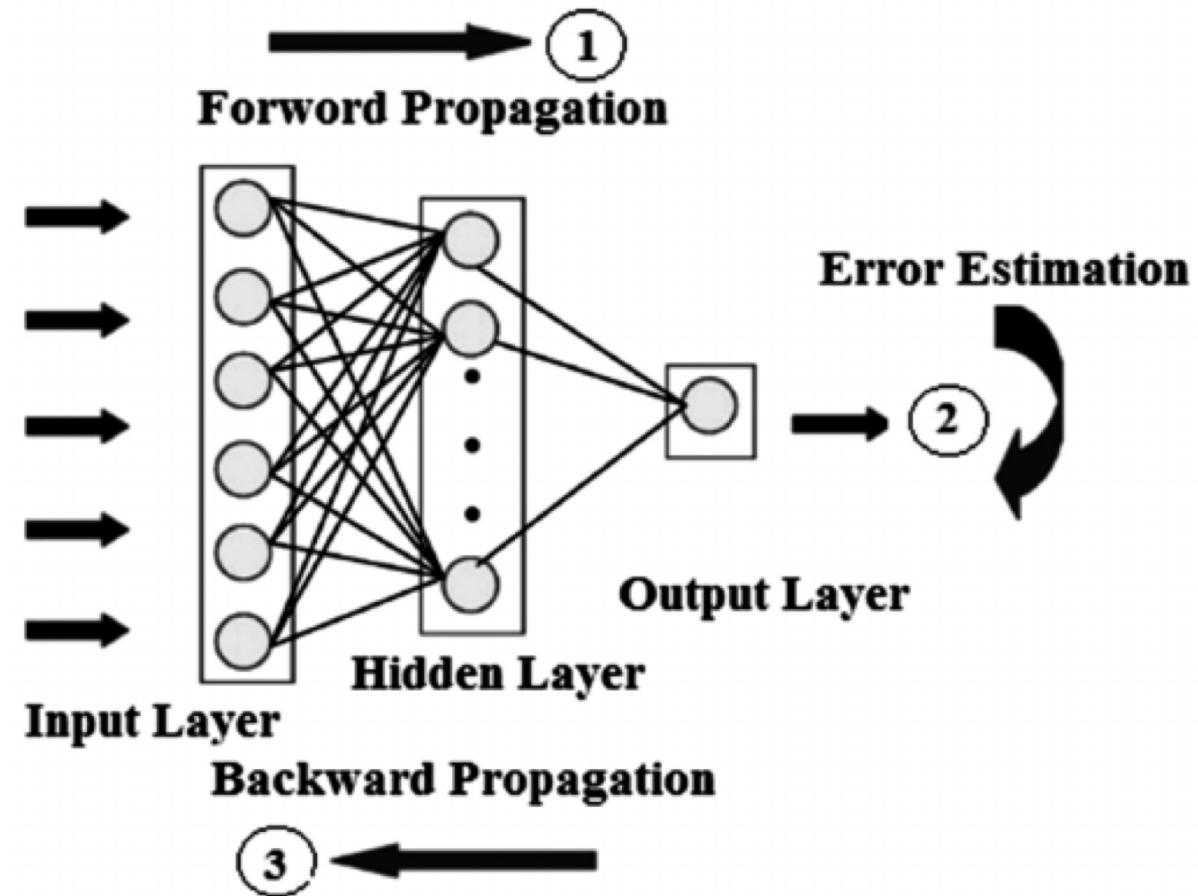
$$\theta^{(\text{next step})} = \theta - \eta \nabla_{\theta} \text{MSE}(\theta)$$

ทบทวน gradient descent ของ Linear Regression

# Backpropagation

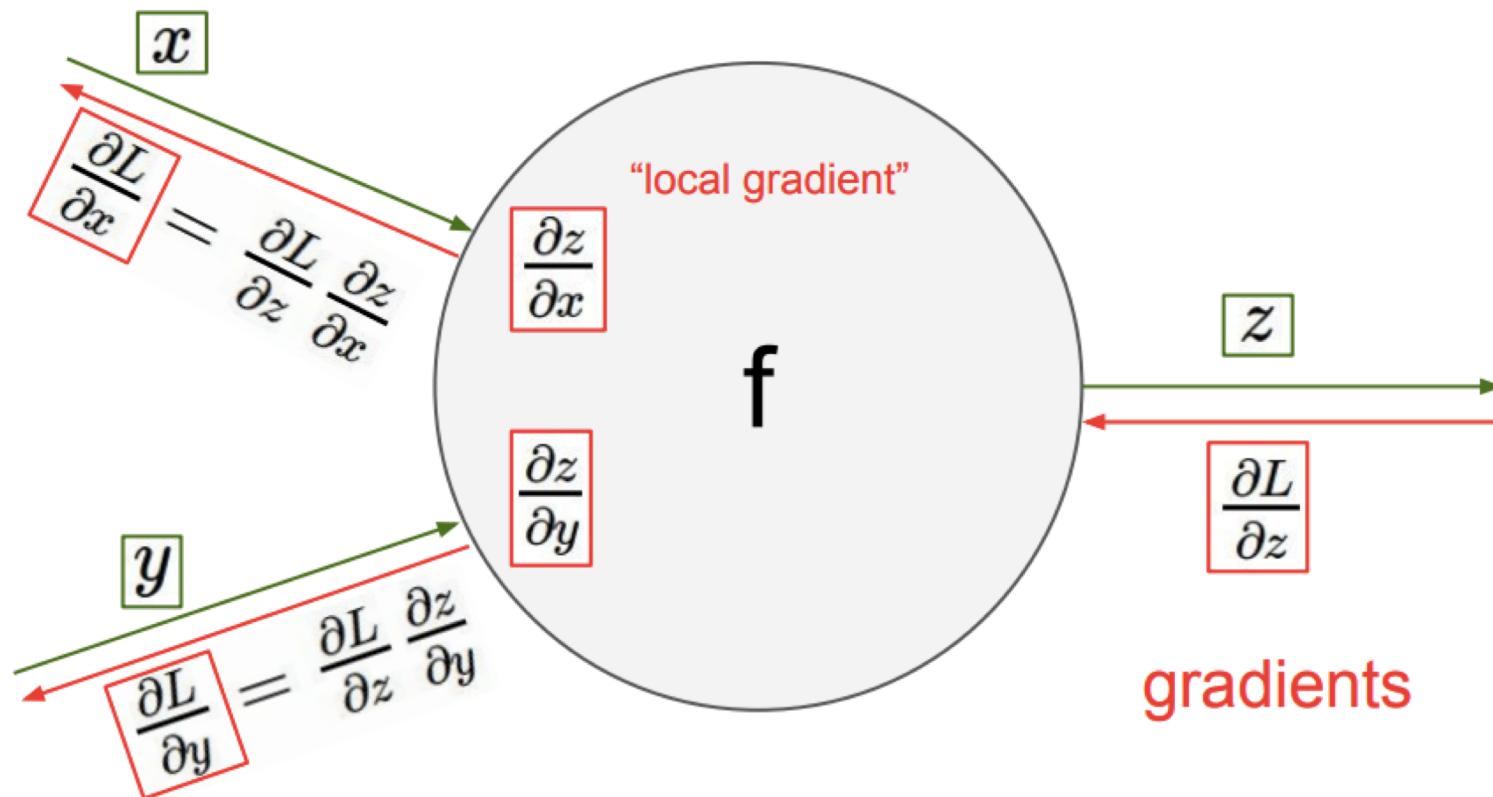
ขั้นตอนวิธีคร่าว ๆ

- for each training instance,
  - make predictions  
*(forward pass)*
  - measure errors
  - propagate  $\frac{\partial \text{Error}}{\partial w_i}$  in reverse  
*(backward pass)*
  - use the computed  $\frac{\partial \text{Error}}{\partial w_i}$  to adjust  $w_i$   
*(gradient descent)*



# Backpropagation

- การคำนวณ gradient ใน backward pass ด้วย chain rule



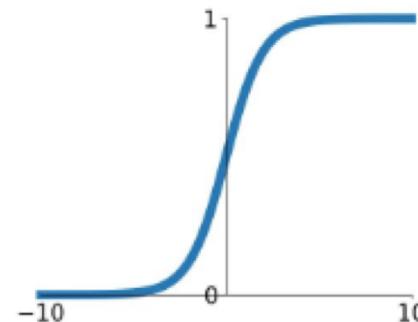
# Activation functions

- เมื่อจากใน **backprop** เราต้องคำนวณ  $\frac{\partial Error}{\partial w_i}$  จึงจำเป็นต้องเปลี่ยนไปใช้ **step function** ที่สามารถหา **derivative** ได้ เราเรียกฟังก์ชันนี้ว่า **activation function**
- ตัวอย่าง **activation function** ที่ควรรู้
  - Logistic (Sigmoid)
  - Hyperbolic tangent (tanh)
  - Rectified Linear Unit (ReLU)
  - Leaky ReLU
  - Exponential Linear Unit (ELU)

# Activation Functions

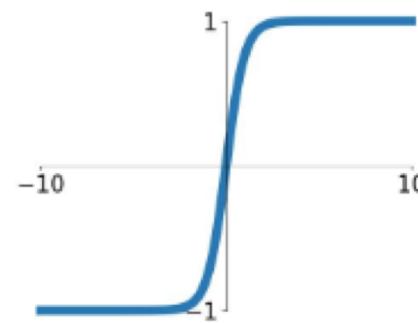
## Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



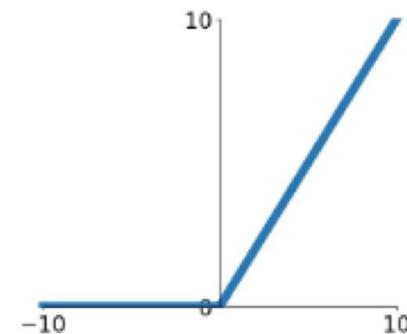
## tanh

$$\tanh(x)$$



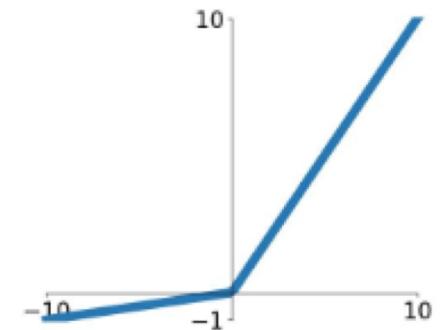
## ReLU

$$\max(0, x)$$



## Leaky ReLU

$$\max(0.1x, x)$$

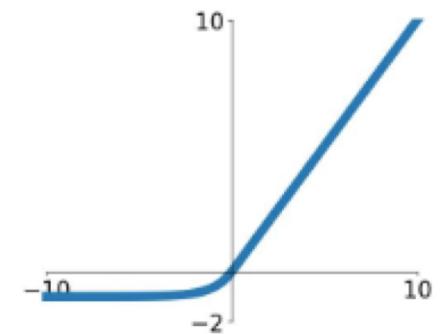


## Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

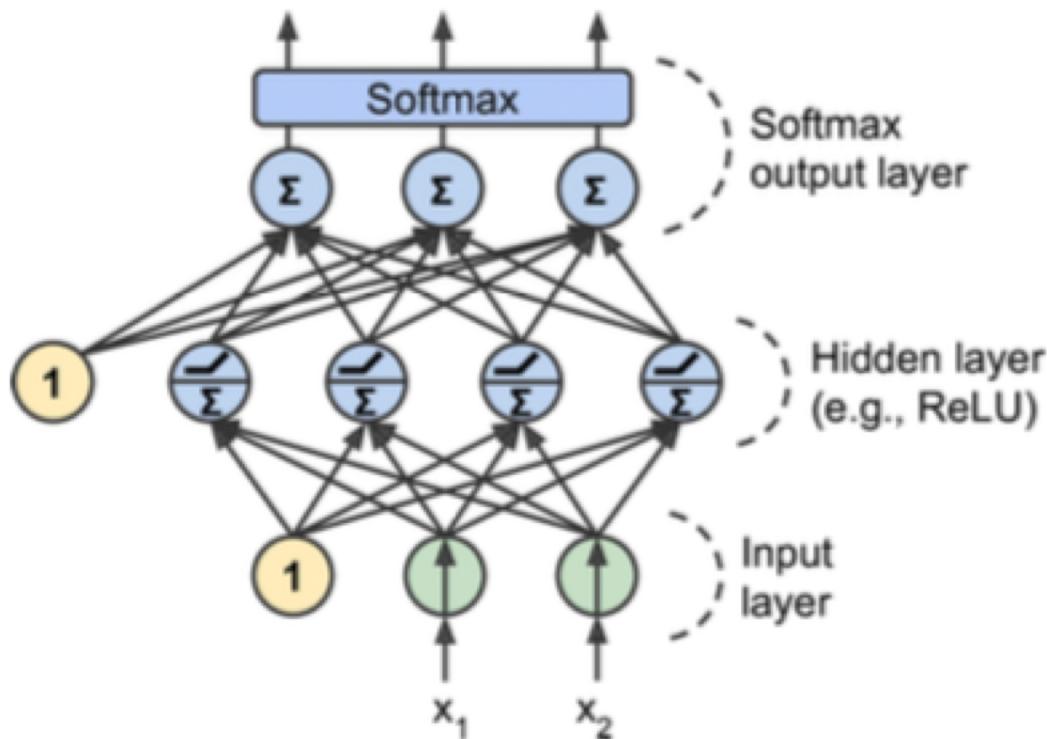
## ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

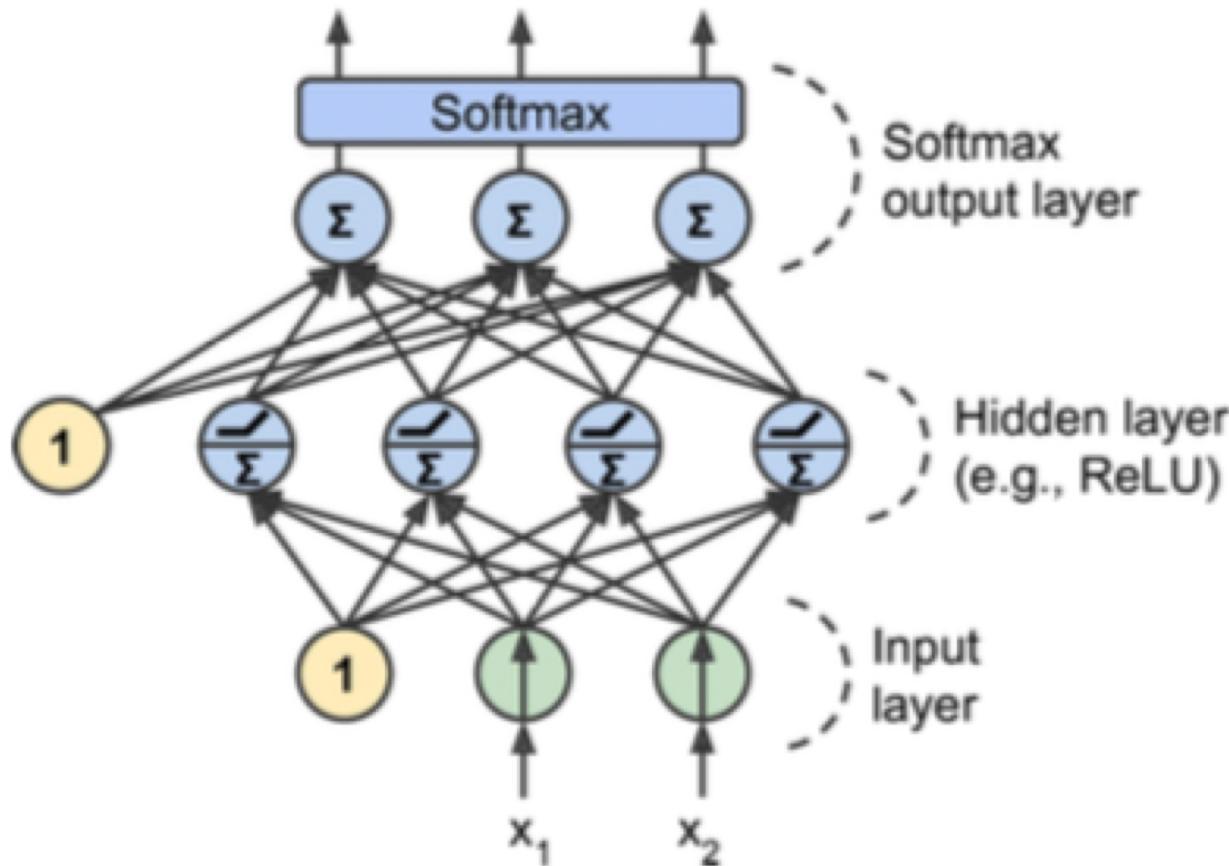


# Softmax Layer

- สำหรับงาน classification เพื่อให้ output ของ network คำนวณค่าความน่าจะเป็นในช่วง 0-1 ของแต่ละคลาส จึงนิยมให้ layer สุดท้ายเป็นแบบ Softmax (generalized/multiclass sigmoid)



# Feedforward Neural Network



เราเรียก architecture ของ ANN ที่ไม่ไปในทางเดียว  
จาก **input** ไปยัง **output** ดังภาพนี้ว่า

**"Feedforward Neural Network"**

# Lab: MNIST with Neural Network

- Popular open-source deep learning libraries

*Table 9-1. Open source Deep Learning libraries (not an exhaustive list)*

Library	API	Platforms	Started by	Year
Caffe	Python, C++, Matlab	Linux, macOS, Windows	Y. Jia, UC Berkeley (BVLC)	2013
Deeplearning4j	Java, Scala, Clojure	Linux, macOS, Windows, Android	A. Gibson, J. Patterson	2014
H2O	Python, R	Linux, macOS, Windows	H2O.ai	2014
MXNet	Python, C++, others	Linux, macOS, Windows, iOS, Android	DMLC	2015
TensorFlow	Python, C++	Linux, macOS, Windows, iOS, Android	Google	2015
Theano	Python	Linux, macOS, iOS	University of Montreal	2010
Torch	C++, Lua	Linux, macOS, iOS, Android	R. Collobert, K. Kavukcuoglu, C. Farabet	2002

# Lab: MNIST with Neural Network

- Keras เป็น high-level API ที่สามารถนำไปรันบน tensorflow, pytorch ได้
- ใช้งานง่าย เหมาะสมกับการเรียนรู้ครั้งแรก
- <https://keras.io/>

# MNIST with Keras – Import dataset

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train,y_train),(x_test,y_test) = mnist.load_data()

x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)
```

# MNIST with Keras – Create NN model

```
model = tf.keras.models.Sequential()  
model.add(tf.keras.layers.Flatten())  
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))  
model.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))  
model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

# MNIST with Keras – Fit/Evaluate model

```
model.compile(optimizer="adam",
loss="sparse_categorical_crossentropy", metrics=['accuracy'])
model.fit(x_train, y_train, epoch=3)
val_loss, val_acc = model.evaluate(x_test, y_test)

model.predict([x_test])
```

# More on NN

- Vanishing Gradient Problem
- Weight Initialization
- Learning Rate Scheduling
- Dealing with overfitting: Dropout, Batch Norm
- More advanced NNs: CNN, RNN
- See Aj. Ekapol's lecture:  
<https://www.youtube.com/watch?v=gaBS1SP6Eiw>