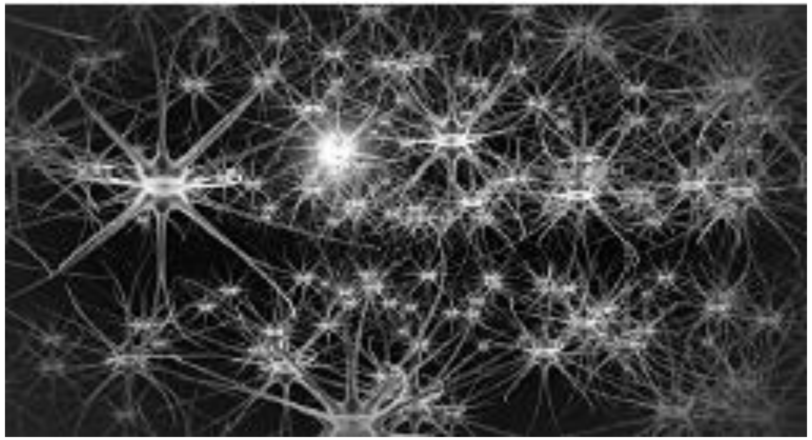
Deep Learning

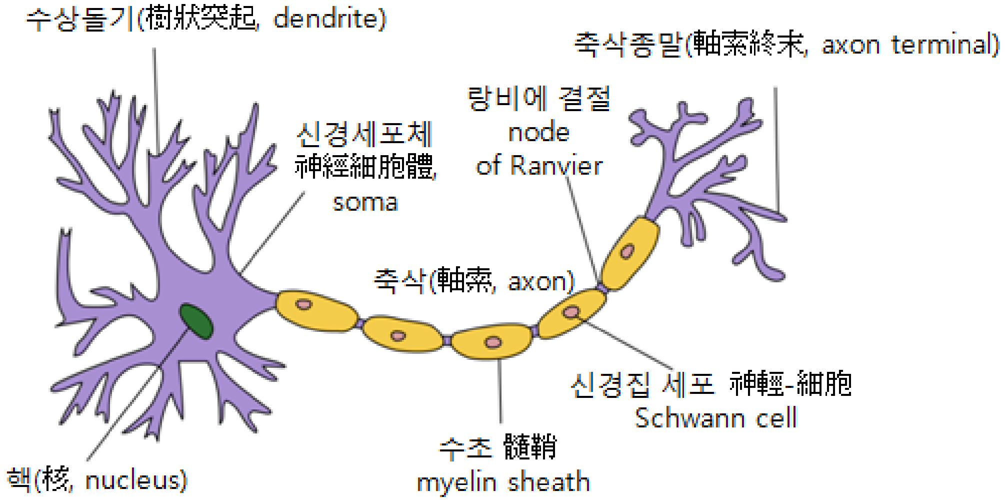
2021. 8

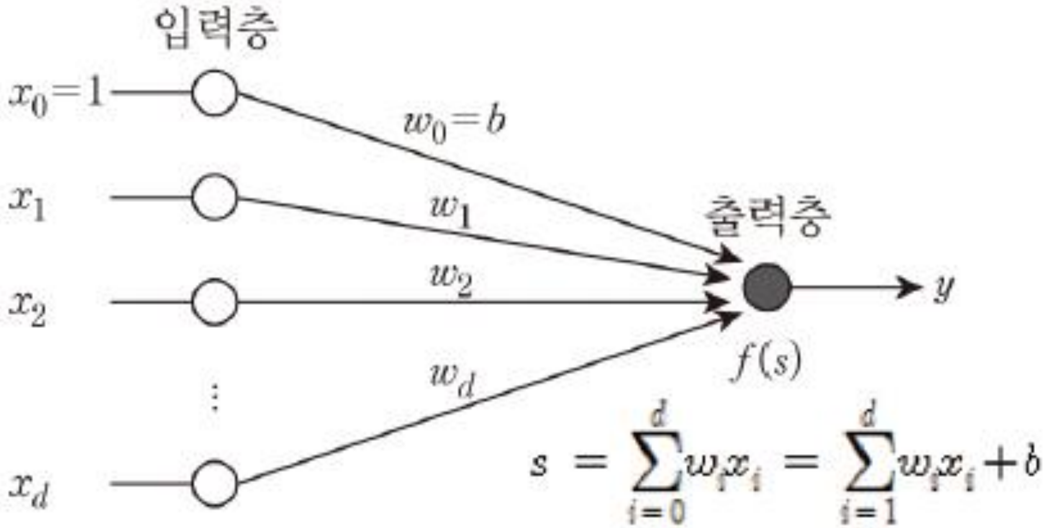
Yongjin Jeong, KwangWoon University

[참고] 본 자료에는 인터넷에서 다운받아 사용한 그림이나 수식들이 일부 있으니 다른 용도로 사용하거나 외부로 유출을 금해 주시기 바랍니다.

Artificial Neural Network







Deep learning Applications

◆ AlphaGo : 48 TPUs, 분산 컴퓨팅

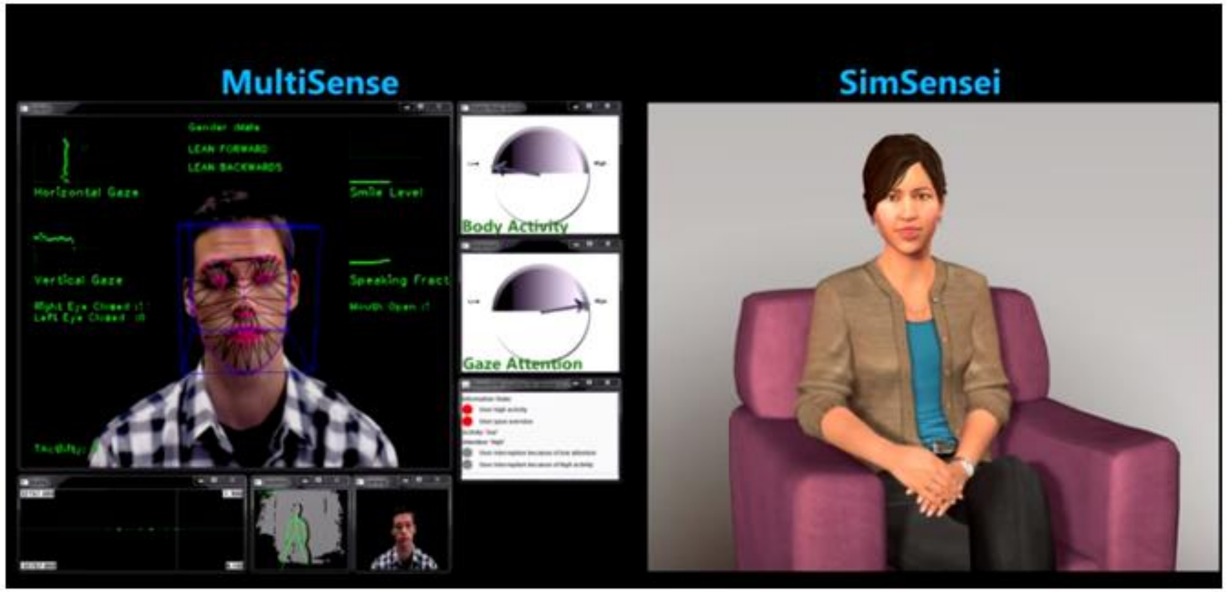
- Google DeepMind 개발
- 2016.3.9~3.15 총 5회의 대국에서 알파고가 4승 1패로 승리
- 기계학습과 병렬처리로 구현

◆ AlphaGo Zero (2017. 12) : 4 TPUs, 단일 서버

- 바둑, 체스, 일본장기와 대국에서 탁월한 성능

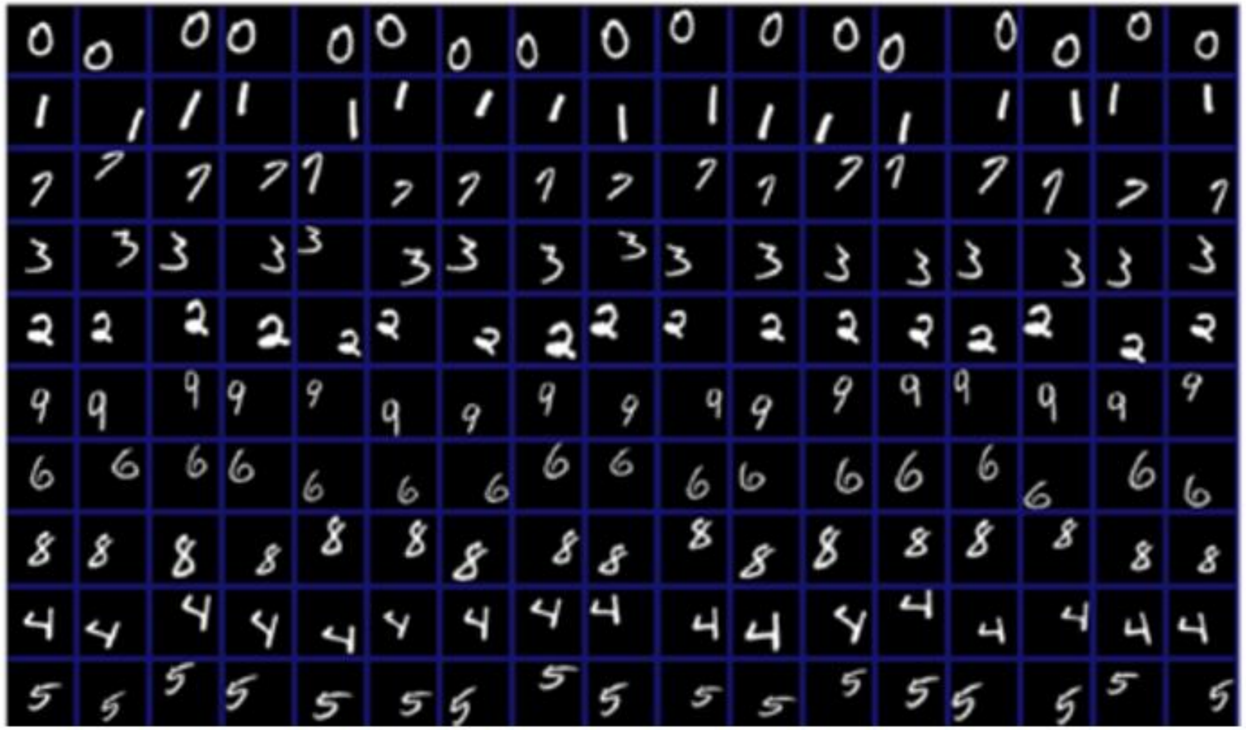
Deep learning Applications

- SimSensei : 안면인식 및 동작인식 기반 우울증 감지 시스템
  - 미, USC(Univ. of Southern California), Institute for Creative Technologies
  - 아바타 형태의 가상 치료 전문가(Therapist)가 일상적인 질문, 환자는 해당 질문에 응답
  - 환자의 얼굴 근육과 음성/패턴, 자세, 행동패턴 등을 파악해 심리상태를 분석 소프트웨어
  - 현재, 우울증 증세 파악 : 주로 설문에 기반한 환자의 응답
  - 설문에 나타나지 않는 환자의 표정, 움직임, 자세 등 66개의 특징 포인트의 관찰을 통해 보다 정확한 증세 파악 가능



Deep learning Applications

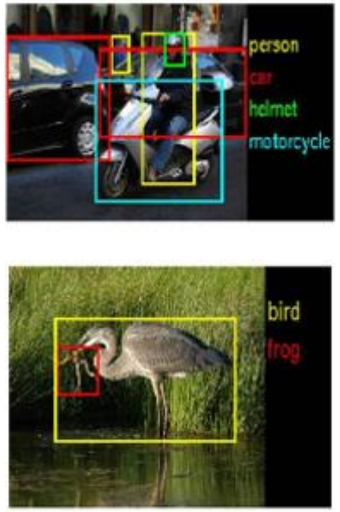
◆ **필기체 숫자 인식 : MNIST 데이터 집합에 대해 2012년 0.23% 오류률 달성**

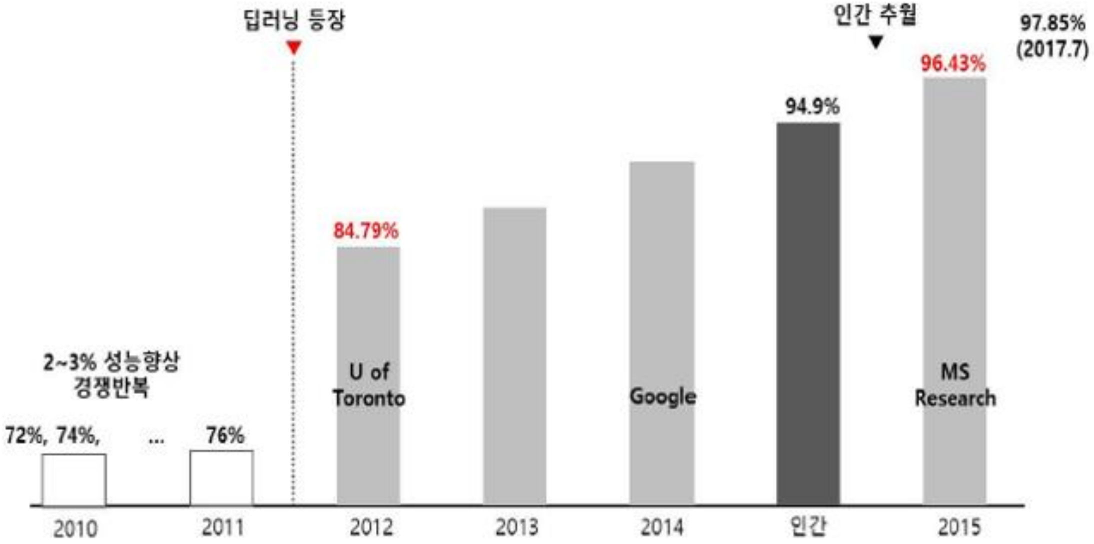


MNIST 필기체 숫자

Deep learning Applications

◆ ImageNet 경진대회(ILSVRC)

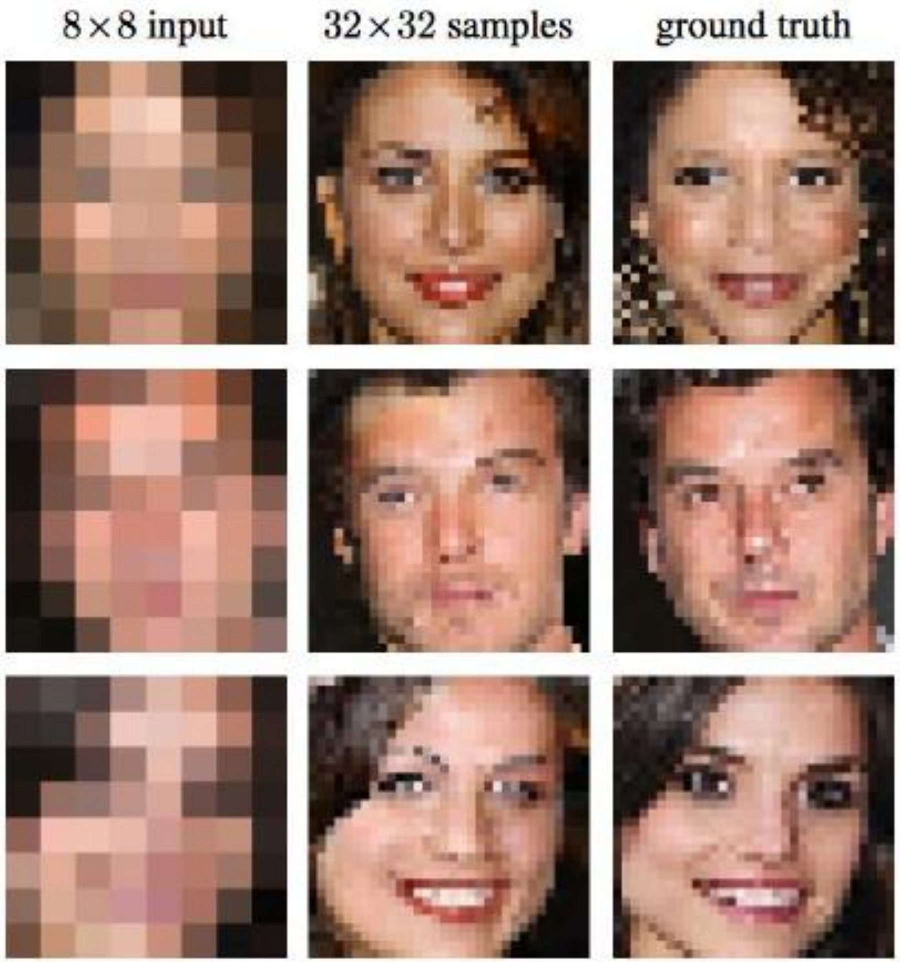




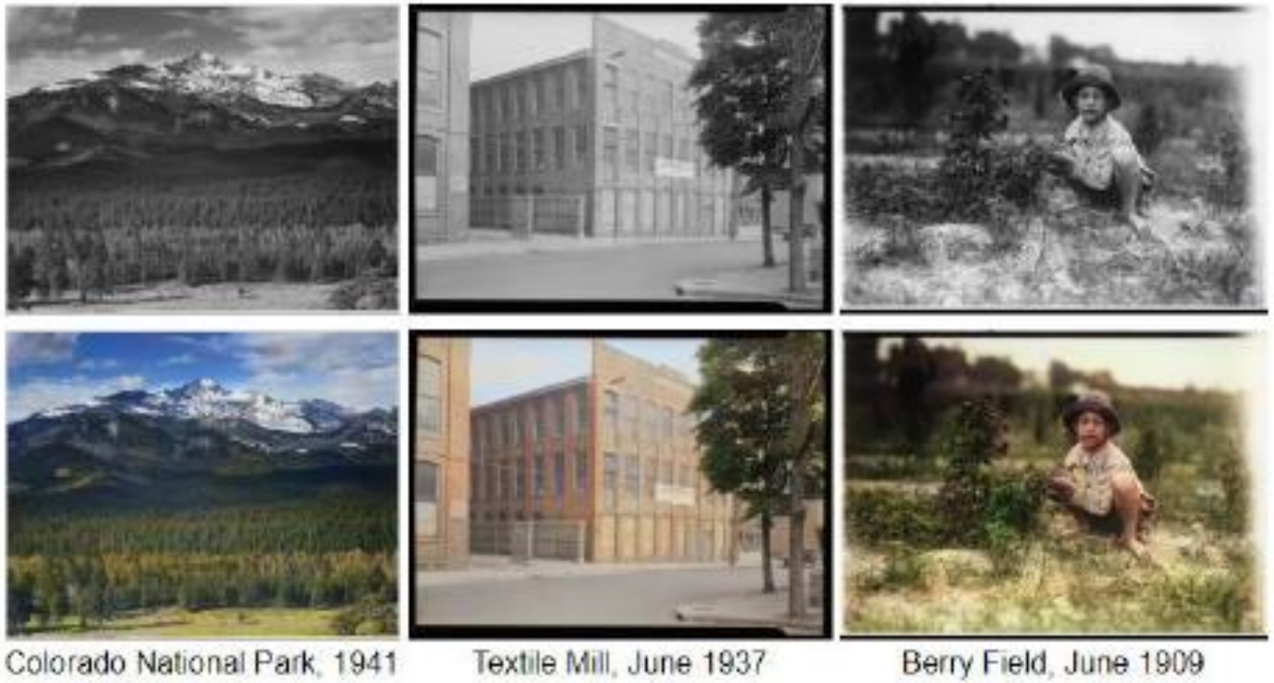
<자료> LG경제연구원, 2017. 10.

Deep learning Applications

- **Image Pixel Recovery**

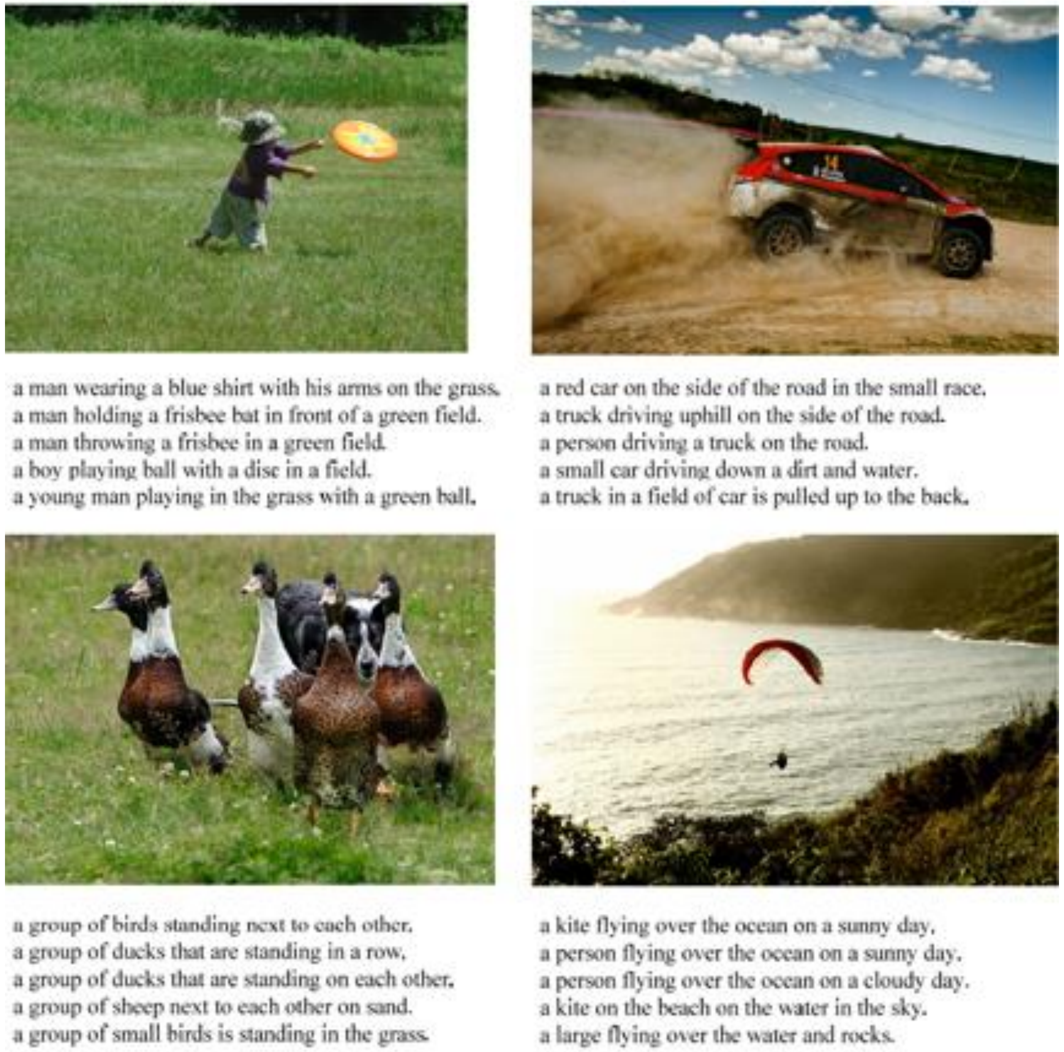


- **Image Color Recovery**



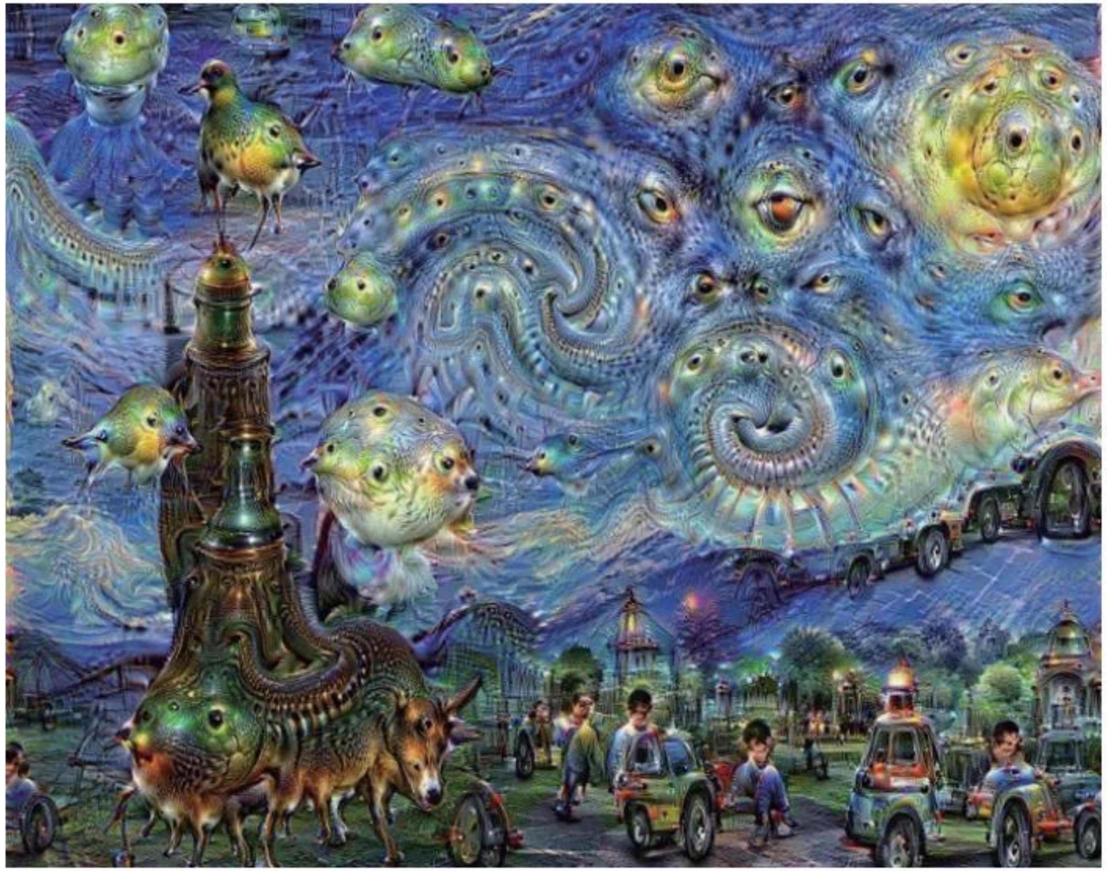
Deep learning Applications

- **Image Captioning**



Deep learning Applications

- **Deep Dream**

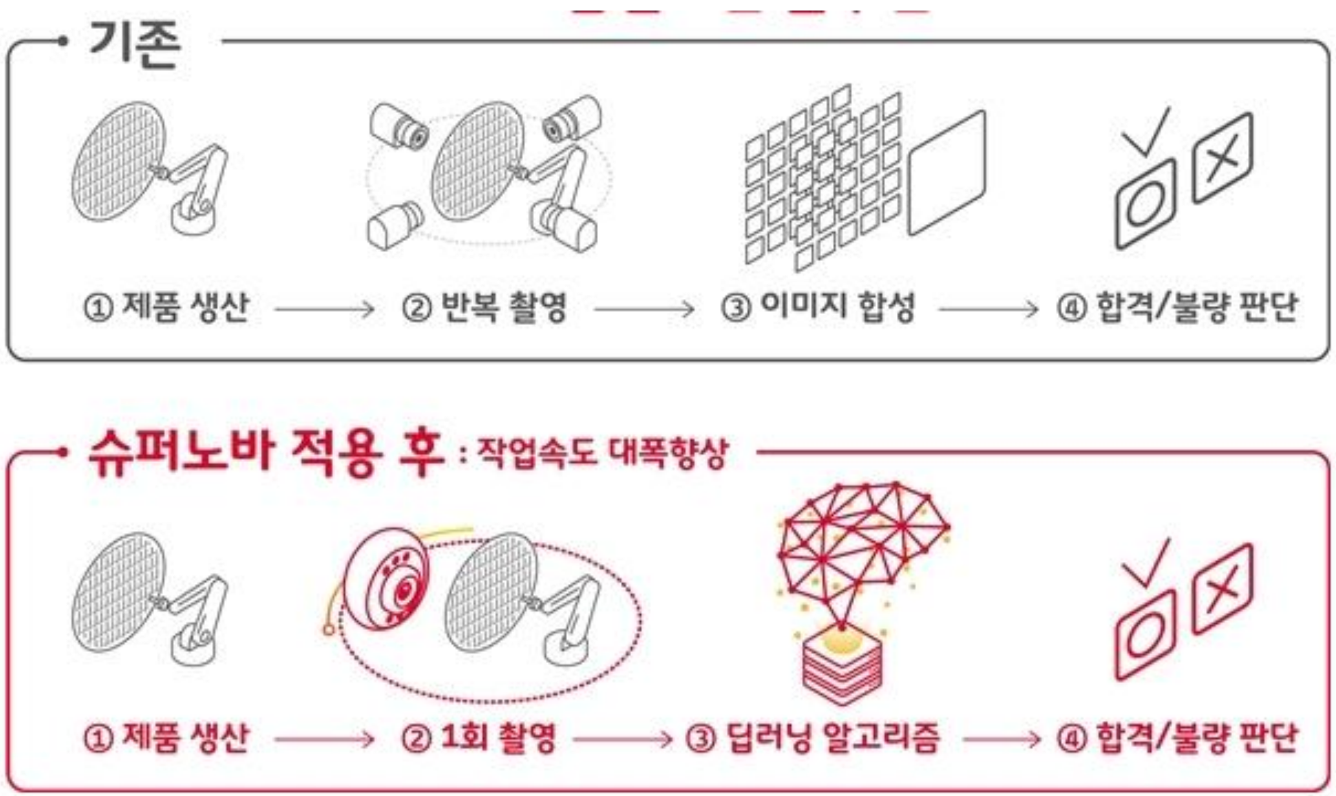


구글 딥드림의 최고가 8,000달러에 낙찰된 작품 @월스트리트저널

https://deepdreamgenerator.com/

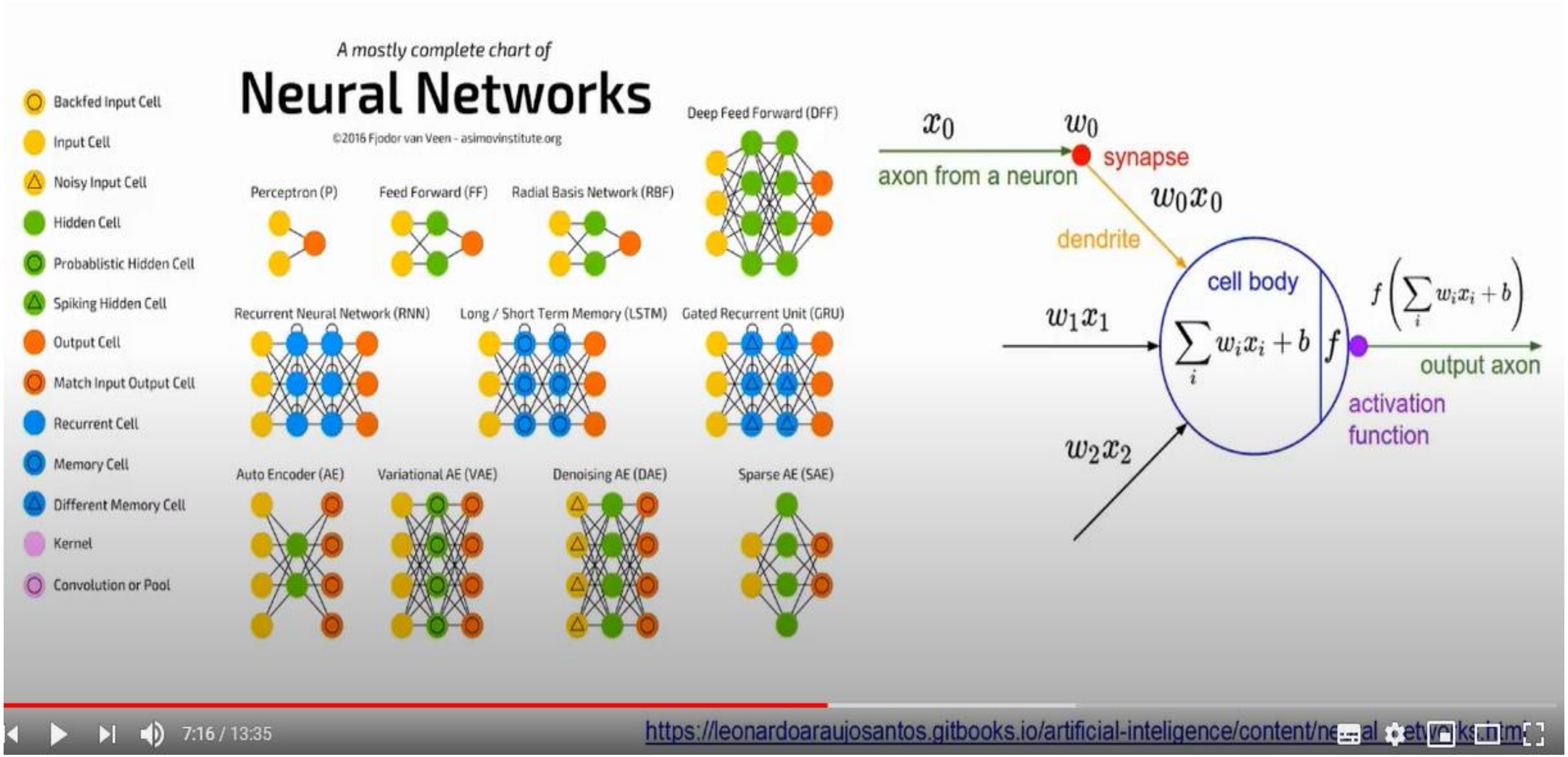
Deep learning Applications

SKT, AI 품질개선 솔루션 '슈퍼노바' MWC서 공개 (2019)

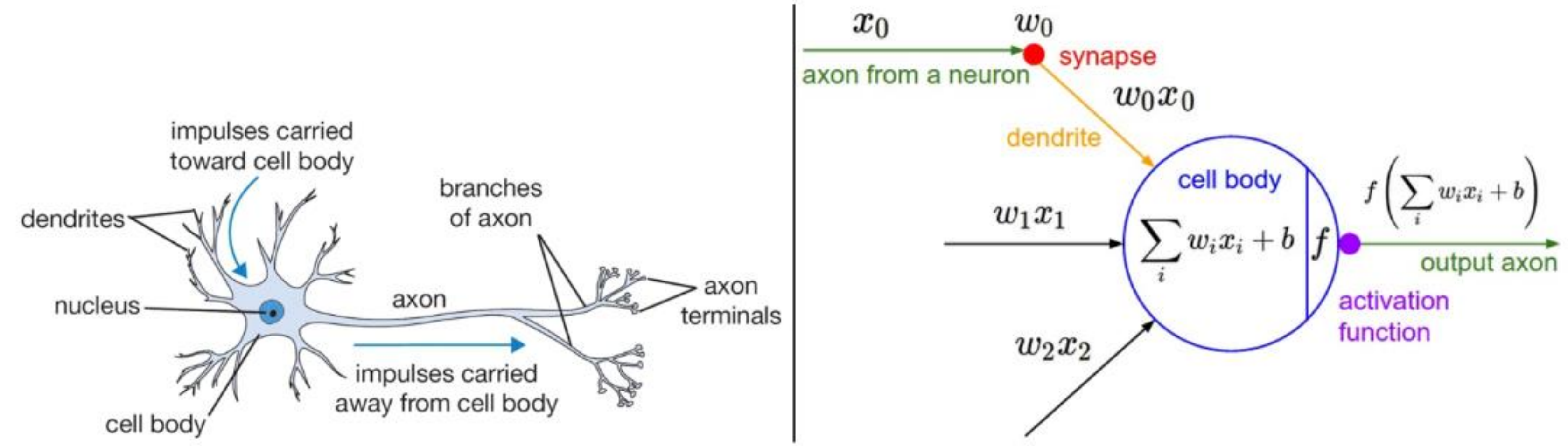


인포그래픽은 슈퍼노바 기술 개념도. SK텔레콤은 자사 전시관 전시관 5G 커넥티드 팩토리 부스에 '슈퍼노바'를 활용한 반도체 제조공정 혁신 모델을 전시한다. <SK텔레콤>

Neural-Net Chart



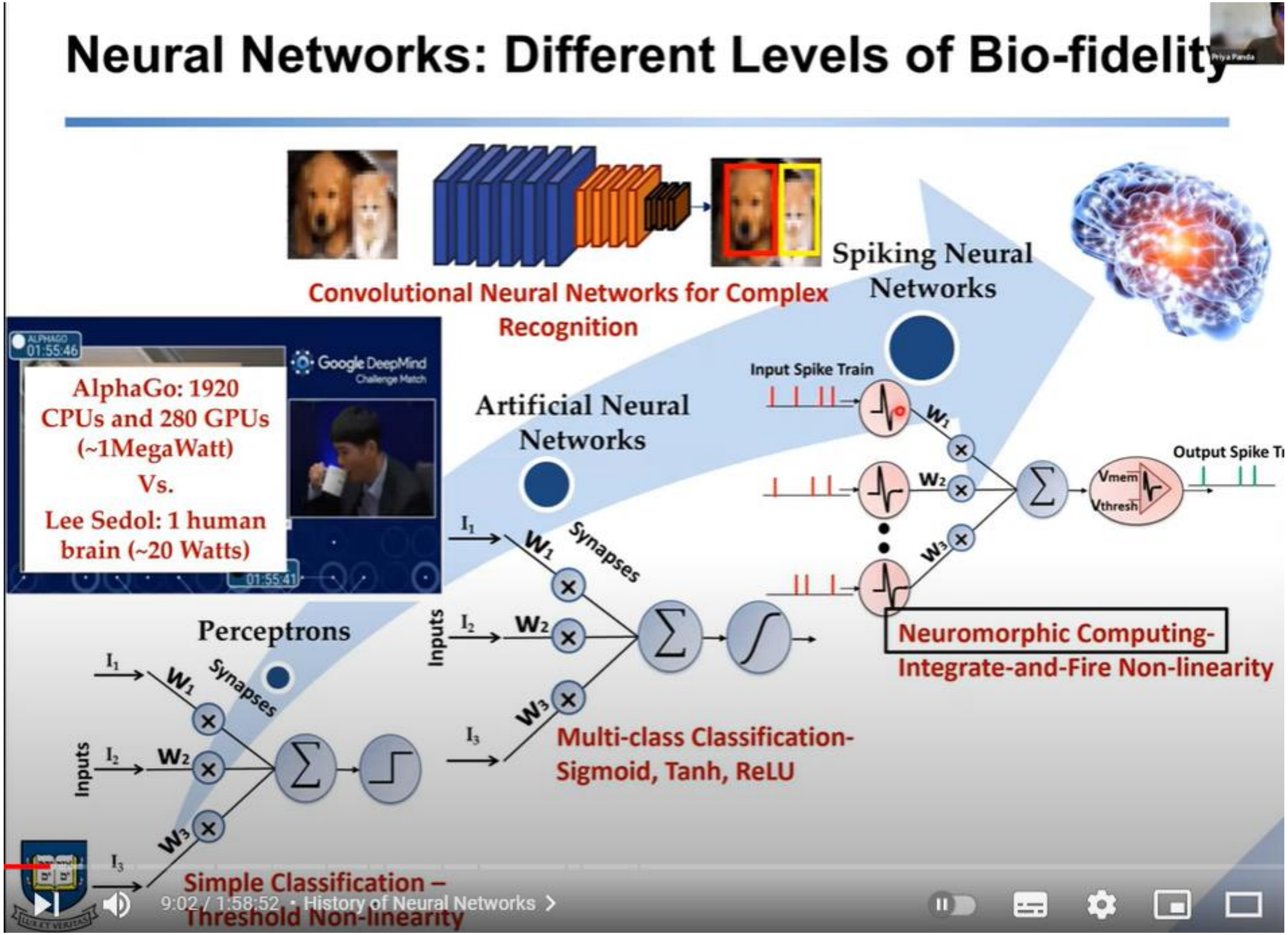
Mathematical model for Neurons



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

(출처: http://cs231n.github.io/neural-networks-1/)

- Axon (축삭돌기) : 뉴런에서 뻗어 나와 다른 뉴런의 수상돌기와 연결
- Dendrite (수상돌기) : 다른 뉴런의 축삭 돌기와 연결, 몸체에 나뭇가지 형태로 붙어 있다
- Synapse (시냅스) : 축삭돌기와 수상돌기가 연결된 지점, 여기서 한 뉴런이 다른 뉴런으로 신호가 전달



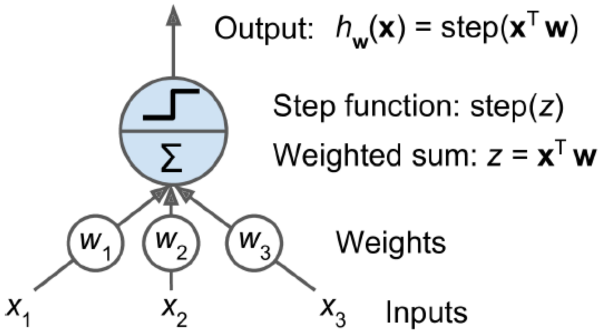
https://www.youtube.com/watch?v=7TybETlCslM (Professor Priya Panda at Yale University)

Perceptron

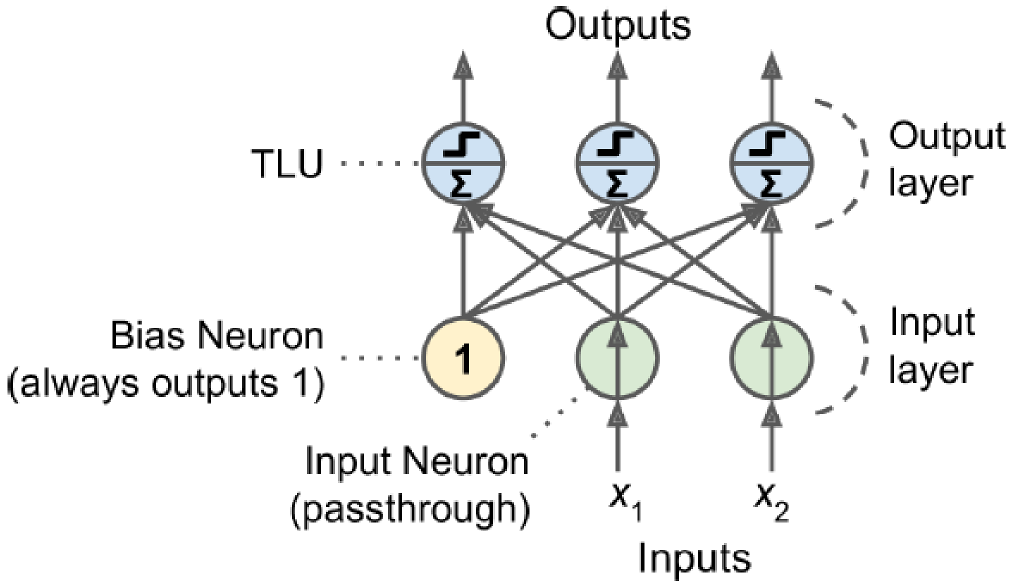
- **Perceptron**
  - A simplest ANN architecture based on threshold logic unit (TLU) or linear threshold unit (LTU) (1957)
  - A single layer of TLUs (fully connected or a dense layer)

$$\boxed{f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}}$$

Threshold logic unit

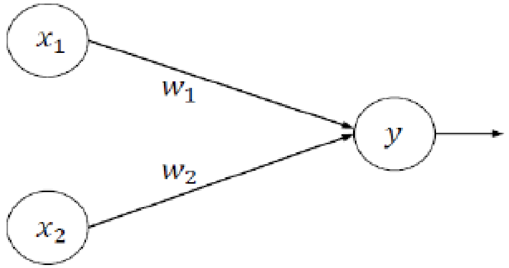


Perceptron diagram

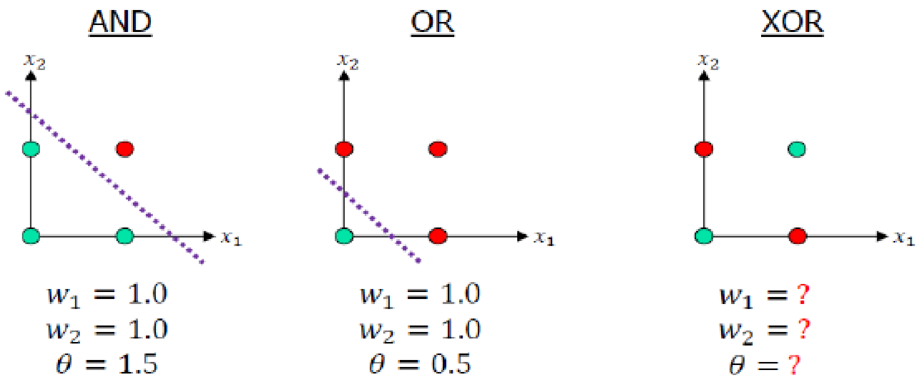


Perceptron

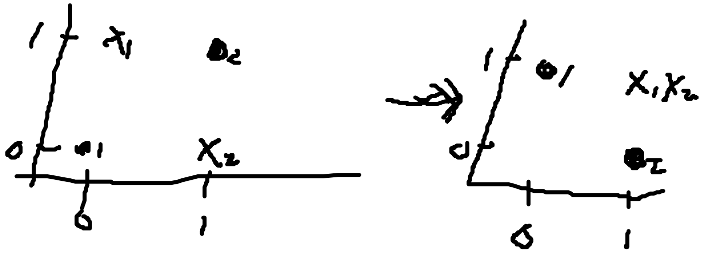
- **Perceptron**
  - Could not solve XOR problem (classifying two classes of XOR dataset)
  - But, later proved that Multilayer Perceptron (MLP) can solve the problem.



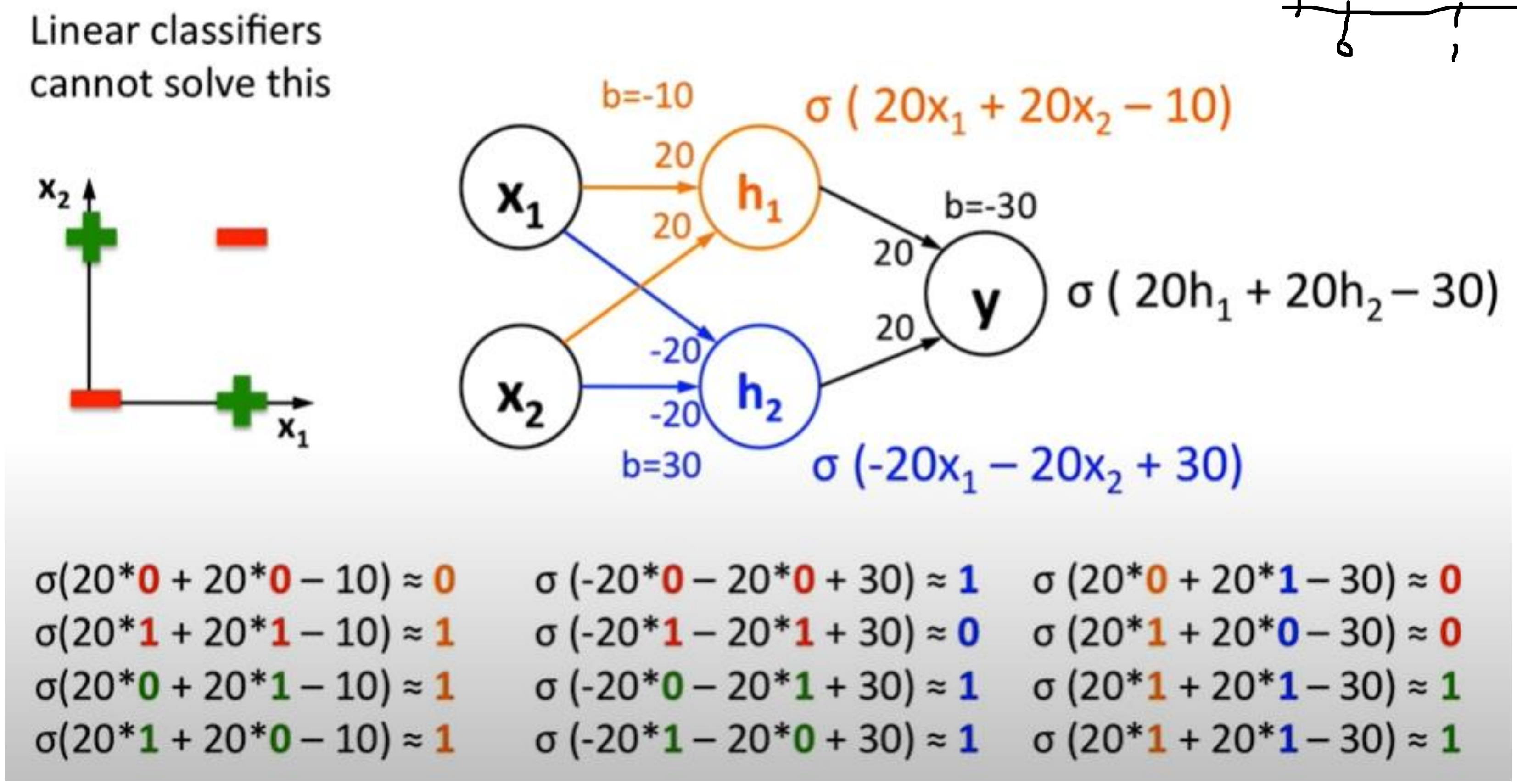
$$y = \begin{cases} 0 & (w_1 \cdot x_1 + w_2 \cdot x_2 \leq \theta) \\ 1 & (w_1 \cdot x_1 + w_2 \cdot x_2 > \theta) \end{cases}$$



Perceptron

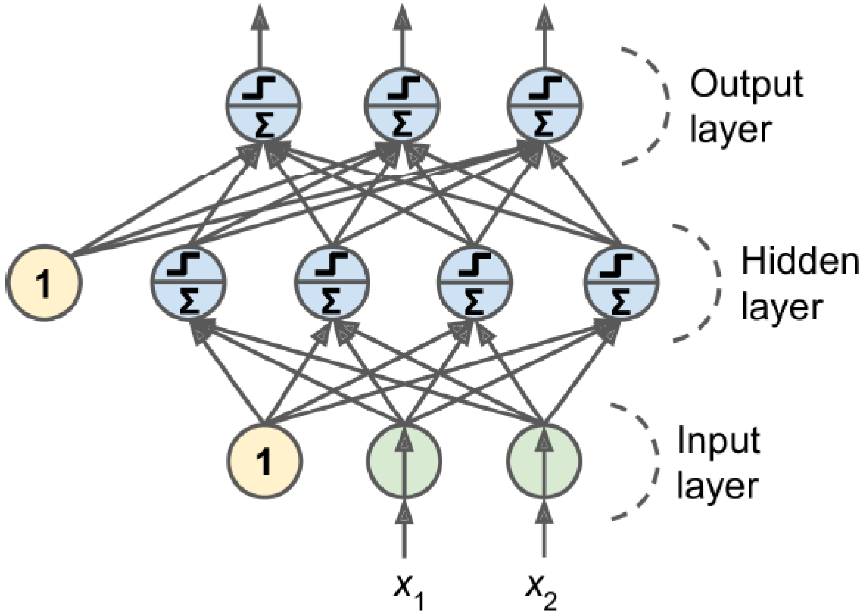


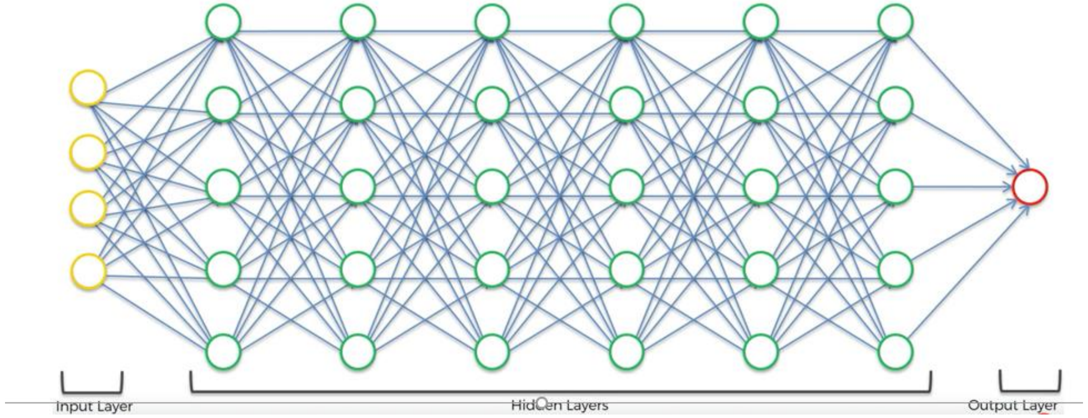
- **Solving XOR problem using MLP**



(Ref: https://www.youtube.com/watch?v=kNPGXgzxoHw)

Multi-Layer Perceptron (MLP)



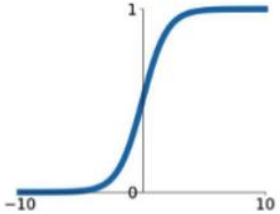


Activation Functions

- **Changed Activation function**
  - step function is flat, so there is no gradient.
  - There are several activation functions.

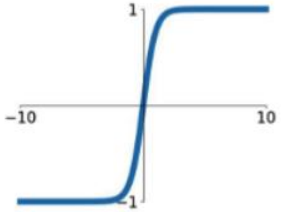
Sigmoid

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



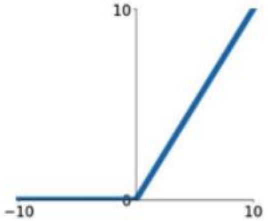
tanh

$\tanh(x)$



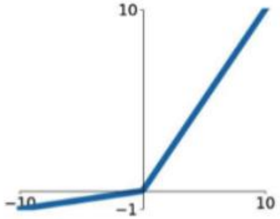
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

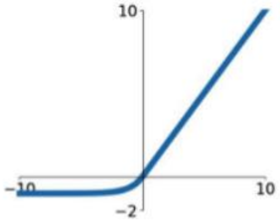


Maxout

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

ELU

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



Activation Functions

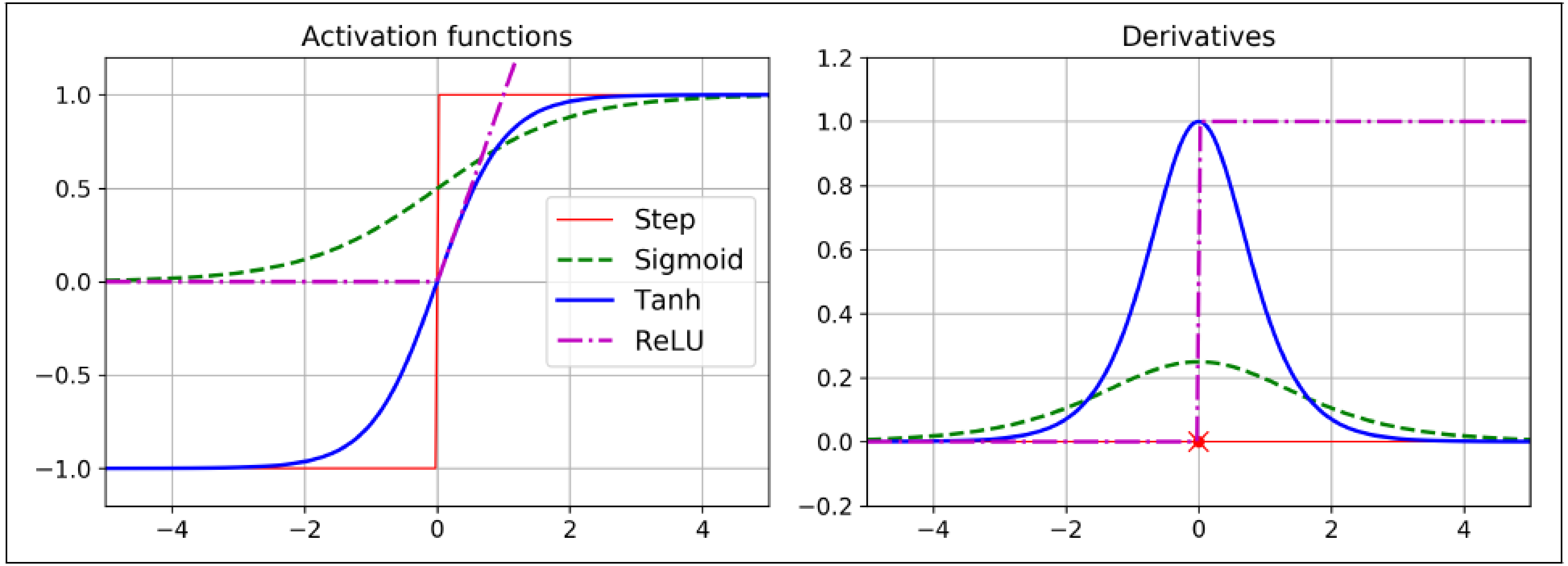


Figure 10-8. Activation functions and their derivatives

MLP for regression

Table 10-1. Typical Regression MLP Architecture

Hyperparameter	Typical Value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem. Typically 1 to 5.
# neurons per hidden layer	Depends on the problem. Typically 10 to 100.
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None or ReLU/Softplus (if positive outputs) or Logistic/Tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

MLP for classification

MLP (including ReLU and softmax) for classification

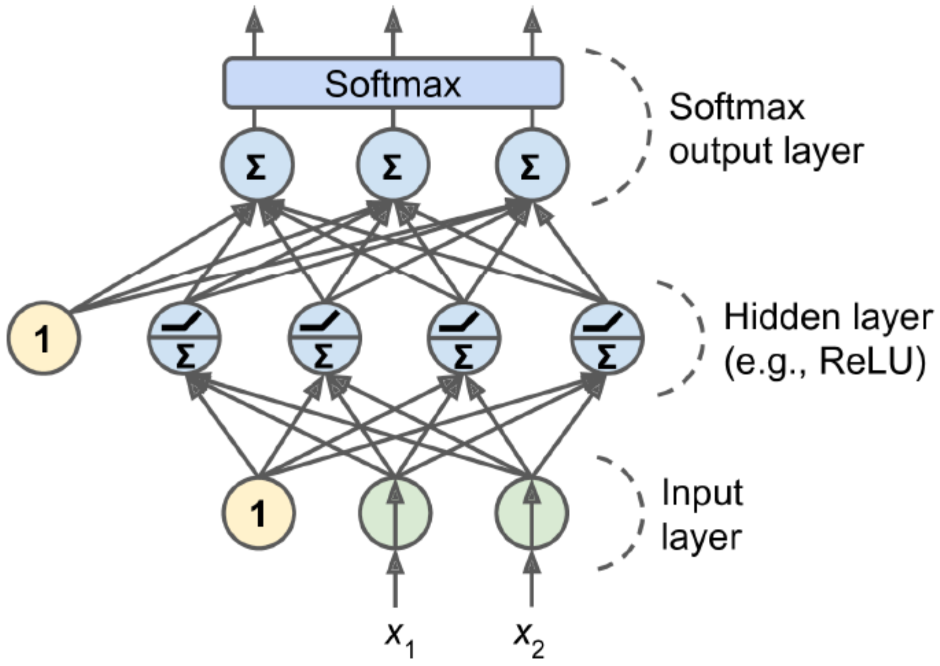


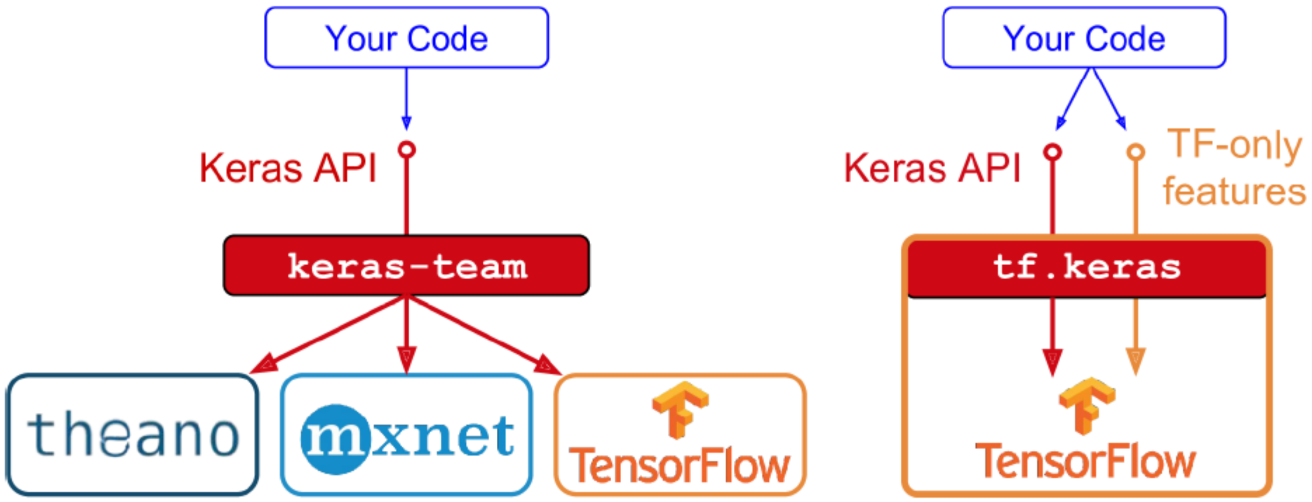
Table 10-2. Typical Classification MLP Architecture

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross-Entropy	Cross-Entropy	Cross-Entropy

multi-class: single label multi-class, (e.g. [0] or [1] or [2])
multi-label: may have more than one class label
(e.g. [1,0,1], [1,1,0])

Implementing MLP with Keras

- **Original Keras (https://keras.io) by Francois Chollet**
  - Can choose Backend from Tensorflow, MisroSoft (CNTK), or Theano
- **Tensorflow Keras**
  - Tensorflow 2.0 comes bundles with its own Keras (tf.keras)
  - Only support Tensorflow as the backend




```
>>> import tensorflow as tf
>>> from tensorflow import keras
>>> tf.__version__
'2.0.0'
>>> keras.__version__
'2.2.4-tf'
```

Building a Model with Keras

- **Sequential API**
  - Create the model

```
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28, 28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))
```

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])
```

  - Compile the Model (specify loss function and optimizer, [extra metrics])
  - Training and evaluating
  - Make prediction

Building a Model with Keras

- **Sequential API**

(*) sparse_categorical_crossentropy: when the target is integer (sparse) labeled (not one-hot encoded)

```
model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=["accuracy"])
```

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

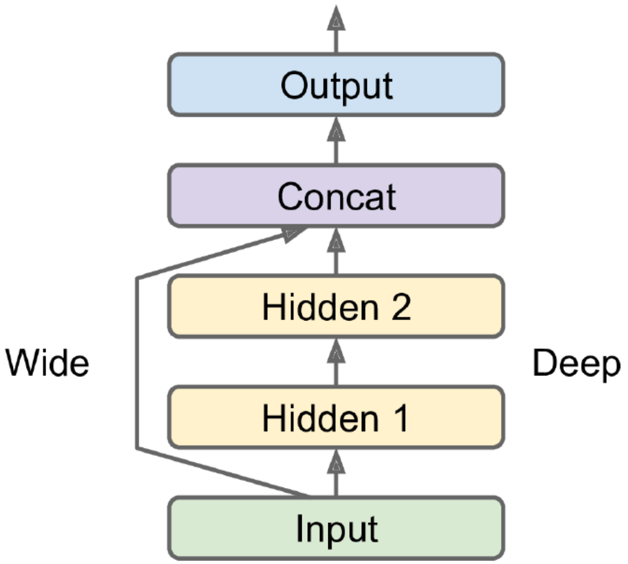
```
>>> model.evaluate(X_test, y_test)
8832/10000 [==========================] - ETA: 0s - loss: 0.4074 - acc: 0.8540
[0.40738476498126985, 0.854]
```

```
>>> X_new = X_test[:3]
>>> y_proba = model.predict(X_new)
>>> y_proba.round(2)
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.09, 0.  , 0.12, 0.  , 0.79],
       [0.  , 0.  , 0.94, 0.  , 0.02, 0.  , 0.04, 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ]],
      dtype=float32)
```

Building a Model with Keras

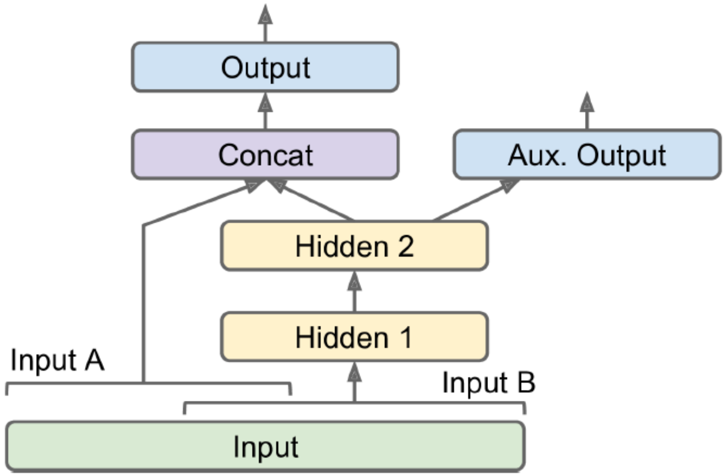
- **Building complex models using the Functional API**

Wide and deep Neural network




```python
input = keras.layers.Input(shape=X_train.shape[1:])
hidden1 = keras.layers.Dense(30, activation="relu")(input)
hidden2 = keras.layers.Dense(30, activation="relu")(hidden1)
concat = keras.layers.Concatenate()[input, hidden2])
output = keras.layers.Dense(1)(concat)
model = keras.models.Model(inputs=[input], outputs=[output])
```

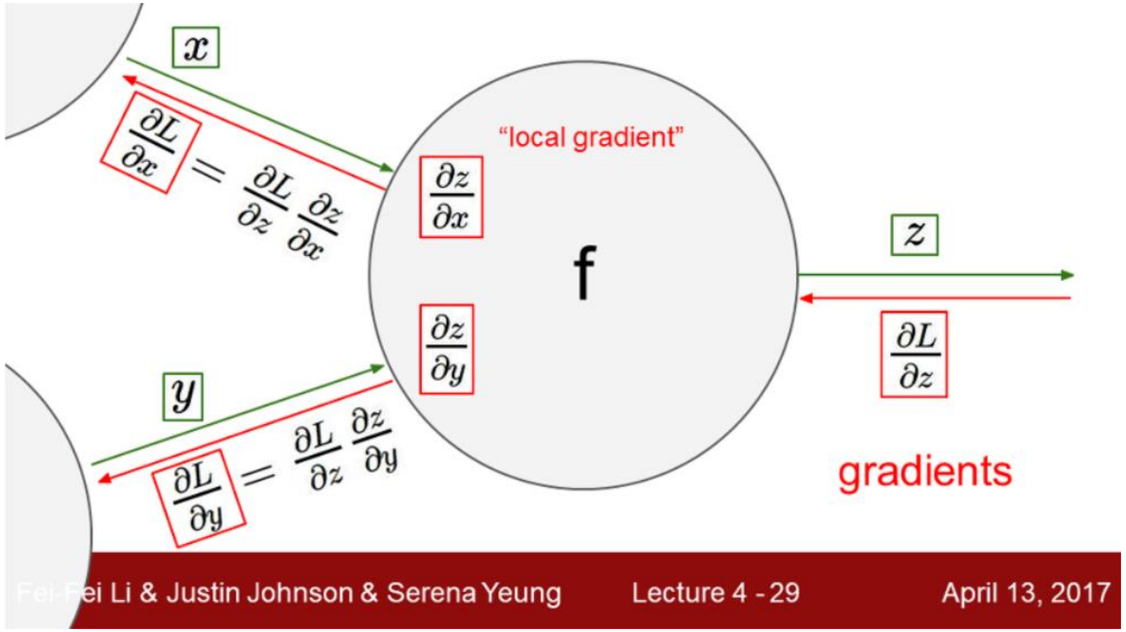
Handling multiple outputs

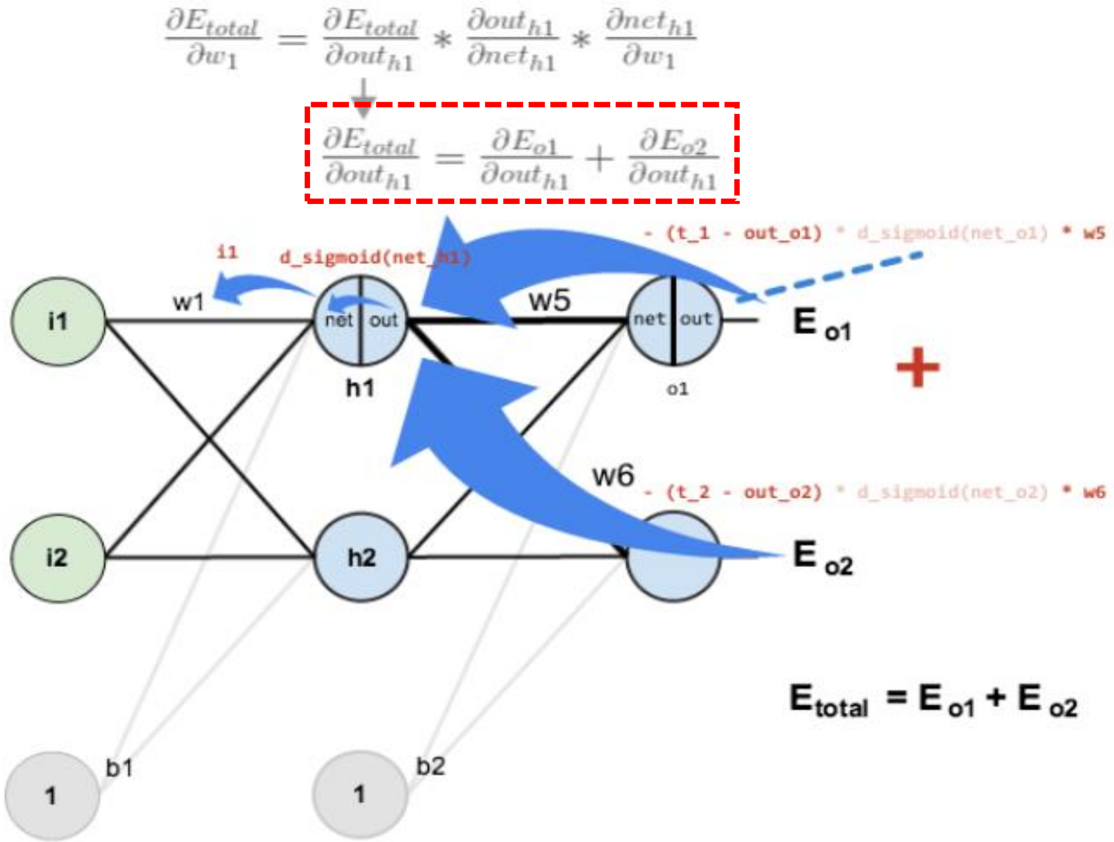



```python
[...] # Same as above, up to the main output layer
output = keras.layers.Dense(1)(concat)
aux_output = keras.layers.Dense(1)(hidden2)
model = keras.models.Model(inputs=[input_A, input_B],
                           outputs=[output, aux_output])
```

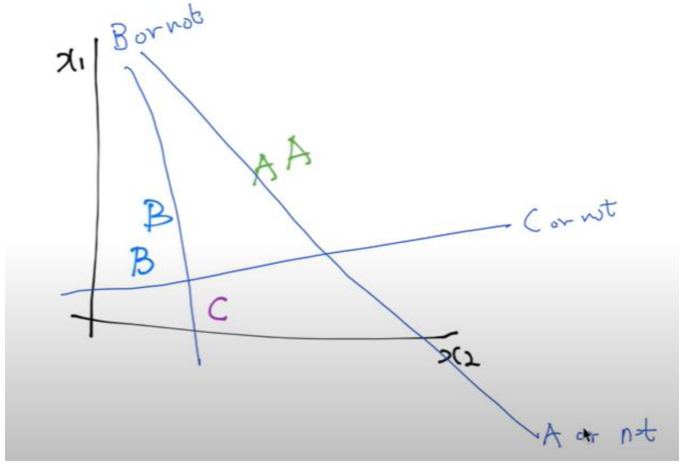
Back-Propagation

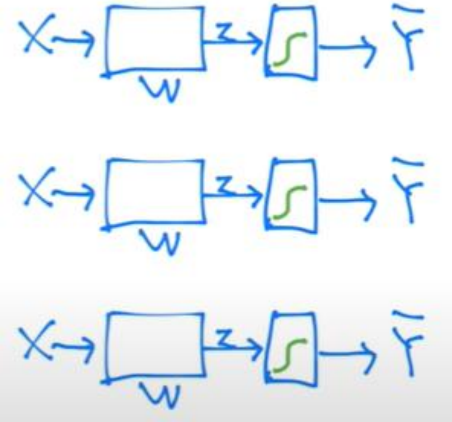
- **Use Chain-rule**

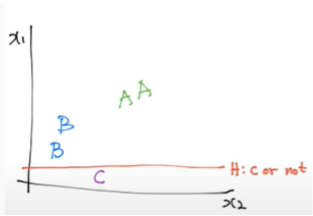


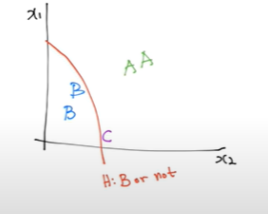


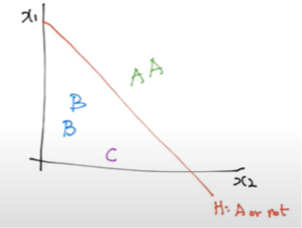
Multinomial (Multi-class) Classification









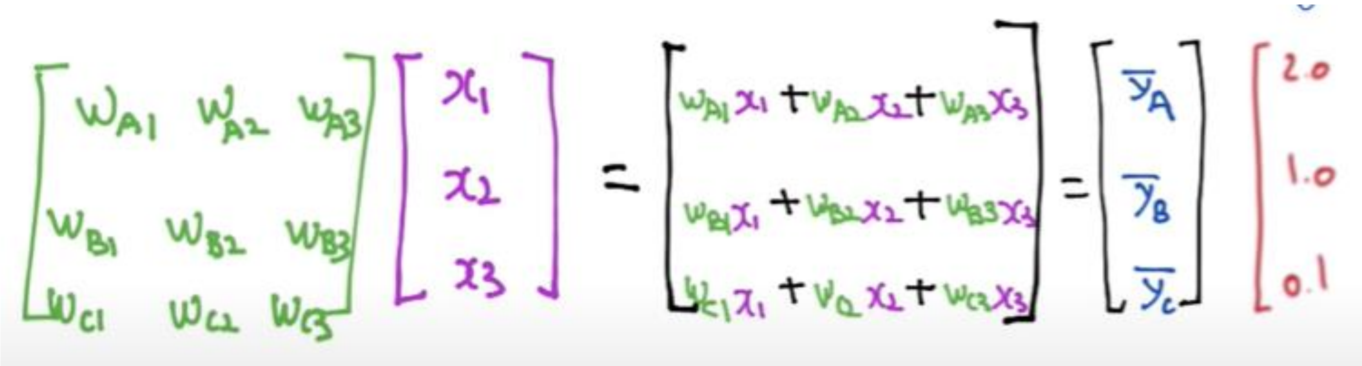


$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 \end{bmatrix}$$

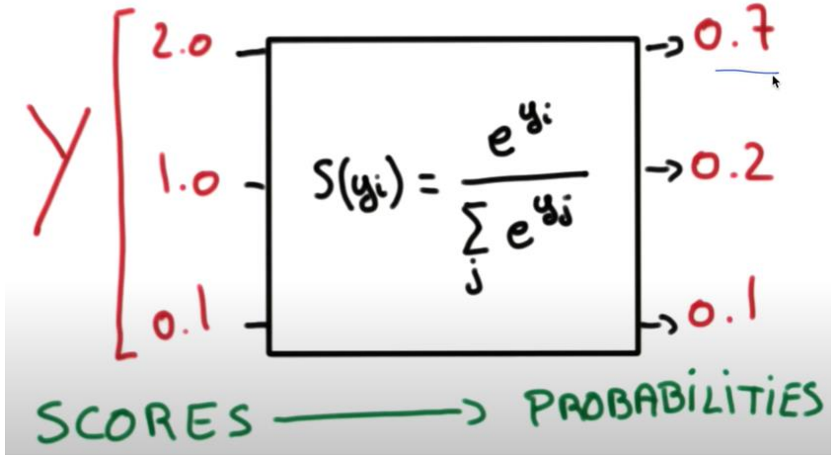
$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_1x_1 + w_2x_2 + w_3x_3 \end{bmatrix}$$

Multinomial (Multi-class) Classification

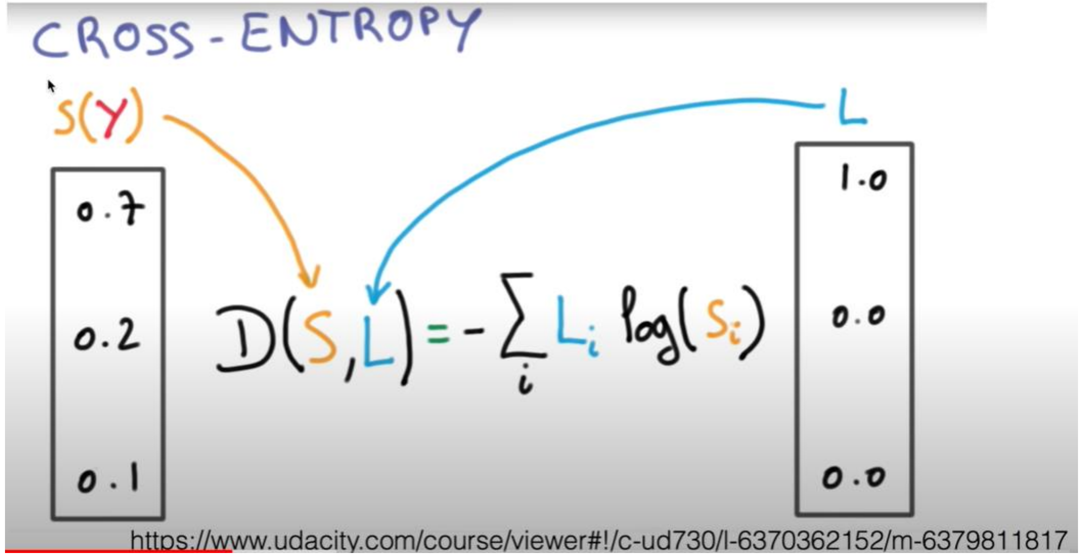


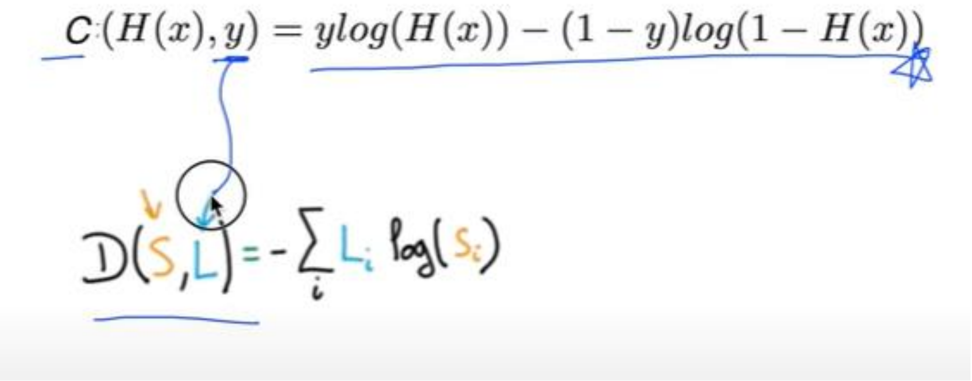
- **Softmax function**



Multinomial (Multi-class) Classification

- **Loss function for Multinomial classification**





Logistic cost (binary cross-entropy)

Cross-entropy cost

Multinomial (Multi-class) Classification

- Cross Entropy (교차 엔트로피)

$$E = -\sum_k t_k \log y_k$$

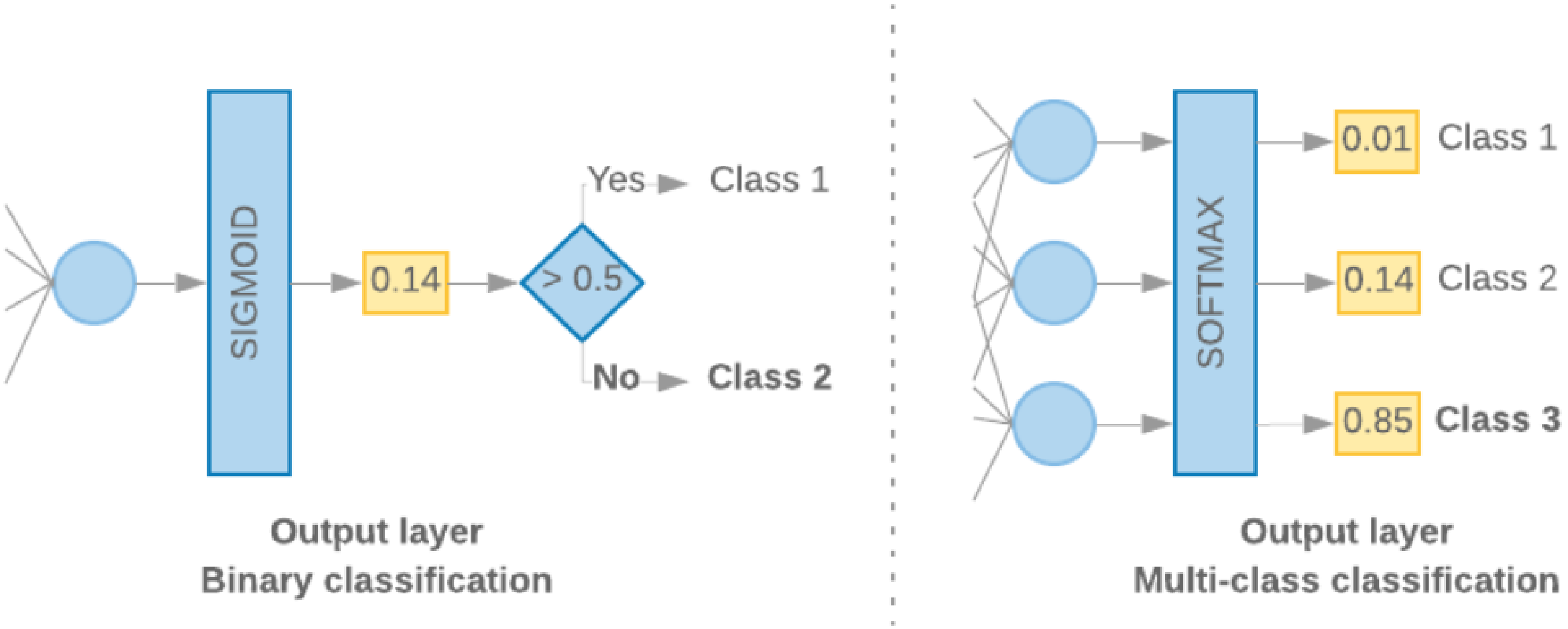
(y: predict, t: target, k: dimension, log: natural log)

- Softmax (소프트맥스)

$$\sigma(j) = \frac{\exp(\mathbf{w}_j^\top \mathbf{x})}{\sum_{k=1}^{K} \exp(\mathbf{w}_k^\top \mathbf{x})} = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}$$

Cross Entropy - Softmax

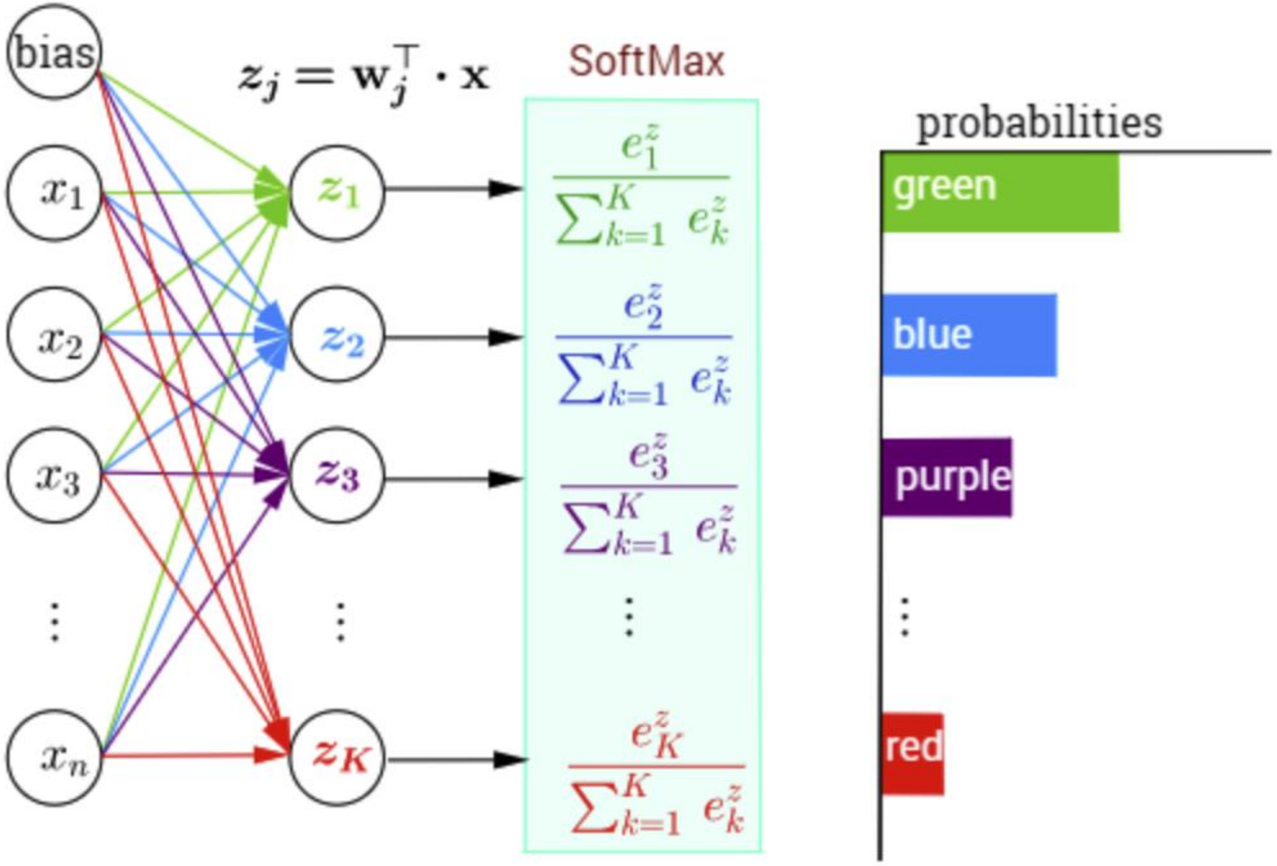
- **Output layer of the Classifier in Deep Learning**



출처: https://developers.google.com/machine-learning/guides/text-classification/step-4

Cross Entropy - Softmax

- 상대적인 점수 비교 : 확률처럼 0~1 사이 값으로 매핑



Matrix Notation for Deep Learning

From https://www.youtube.com/watch?v=9l1YHo-pbf8&list=PLQ28Nx3M4Jrguyuwg4xe9d9t2XE639e5C&index=7

Hypothesis using Matrices

$$w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n$$

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1w_1 + x_2w_2 + x_3w_3 \end{pmatrix}$$

$$H(X) = XW$$

x_1	x_2	x_3	y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test Scores for General Psychology

$$H(x_1, x_2, x_3) = w_1x_1 + w_2x_2 + w_3x_3$$

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1w_1 + x_2w_2 + x_3w_3 \end{pmatrix}$$

$$H(X) = XW$$

Hypothesis using Matrices

x_1	x_2	x_3	y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

$$w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n$$

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

$$H(X) = XW$$

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

[5, 3] [3, 1] [5, 1]

$$H(X) = XW$$

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

[n, 3] [3, 1] [n, 1]

$$H(X) = XW$$

Hypothesis using Matrices

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \; ? \; = \begin{pmatrix} x_{11}w_{11} + x_{12}w_{21} + x_{13}w_{31} & x_{11}w_{12} + x_{12}w_{22} + x_{13}w_{32} \\ x_{21}w_{11} + x_{22}w_{21} + x_{23}w_{31} & x_{21}w_{12} + x_{22}w_{22} + x_{23}w_{32} \\ x_{31}w_{11} + x_{32}w_{21} + x_{33}w_{31} & x_{31}w_{12} + x_{32}w_{22} + x_{33}w_{32} \\ x_{41}w_{11} + x_{42}w_{21} + x_{43}w_{31} & x_{41}w_{12} + x_{42}w_{22} + x_{43}w_{32} \\ x_{51}w_{11} + x_{52}w_{21} + x_{53}w_{31} & x_{51}w_{12} + x_{52}w_{22} + x_{53}w_{32} \end{pmatrix}$$

[n, 3] [?, ?] [n, 2]

$$H(X) = XW$$

Hypothesis using Matrices

WX vs **XW**

- Lecture (theory)

$$H(x) = Wx + b$$

$$h_\theta(x) = \theta_1 x + \theta_0$$

$$f(x) = ax + b$$

- Implementation (TensorFlow)

$$H(X) = XW$$

Hypothesis using Matrices

```python
data = np.array([
    # X1,   X2,    X3,   y
    [ 73.,  80.,  75., 152. ],
    [ 93.,  88.,  93., 185. ],
    [ 89.,  91.,  90., 180. ],
    [ 96.,  98., 100., 196. ],
    [ 73.,  66.,  70., 142. ]
], dtype=np.float32)

# slice data
X = data[:, :-1]
y = data[:, [-1]]
```

$$H(X) = XW$$

```python
W = tf.Variable(tf.random_normal([3, 1]))
b = tf.Variable(tf.random_normal([1]))

# hypothesis, prediction function
def predict(X):
  return tf.matmul(X, W) + b
```

```python
# slice data
X = data[:, :-1]
y = data[:, [-1]]

W = tf.Variable(tf.random_normal([3, 1]))
b = tf.Variable(tf.random_normal([1]))

learning_rate = 0.000001

# hypothesis, prediction function
def predict(X):
  return tf.matmul(X, W) + b

n_epochs = 2000
for i in range(n_epochs+1):
    # record the gradient of the cost function
    with tf.GradientTape() as tape:
        cost = tf.reduce_mean((tf.square(predict(X) - y)))

    # calculates the gradients of the loss
    W_grad, b_grad = tape.gradient(cost, [W, b])

    # updates parameters (W and b)
    W.assign_sub(learning_rate * W_grad)
    b.assign_sub(learning_rate * b_grad)

    if i % 100 == 0:
      print("{:5} | {:10.4f}".format(i, cost.numpy()))
```

Hypothesis using Matrices (example)

$$(x_1 \quad x_2 \quad x_3) \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = (x_1 w_1 + x_2 w_2 + x_3 w_3)$$

$$H(X) = XW$$

```
# initialize W
w1 = tf.Variable(tf.random_normal([1]))
w2 = tf.Variable(tf.random_normal([1]))
w3 = tf.Variable(tf.random_normal([1]))

# hypothesis, prediction function
w1 * x1 +  w2 * x2 + w3 * x3 + b

# update w1,w2,w3
w1.assign_sub(learning_rate * w1_grad)
w2.assign_sub(learning_rate * w2_grad)
w3.assign_sub(learning_rate * w3_grad)
```

```
# initialize W
W = tf.Variable(tf.random_normal([3, 1]))

# hypothesis, prediction function
tf.matmul(X, W) + b

# updates parameters (W and b)
W.assign_sub(learning_rate * W_grad)
```