Introduction to Artificial Neural Networks with Keras [1]

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hankyong.ac.kr

국립한경대학교
HANKYONG NATIONAL UNIV.

"새들은 우리에게 날아다니는 법에 대해 영감을 주었다."

⇒ 자연은 우리에게 발명에 대한 무한한 영감을 준다.

지능적인 기계를 만드는 방법에 대해 **뇌의 구조**를 보고 영감을 얻어

⇒ 인공 신경망(ANN) 모델 창안

"비행기는 새에서 영감을 받았지만 날개를 펄럭일 필요는 없다. 유사하게, ANN은 그들의 생물학적 사촌과 점차적으로 상당히 달라졌다."

ANN : **딥러닝의 핵심**

- Google Images - 수십억 개의 이미지를 분류
- Apple Siri - 음성 인식 서비스를 강화
- YouTube - 매일 수억 명의 사용자에게 베스트 비디오를 추천
- DeepMind's AlphaGo - 게임에서 세계 챔피언을 이기는 법을 배움

From Biological to Artificial Neurons

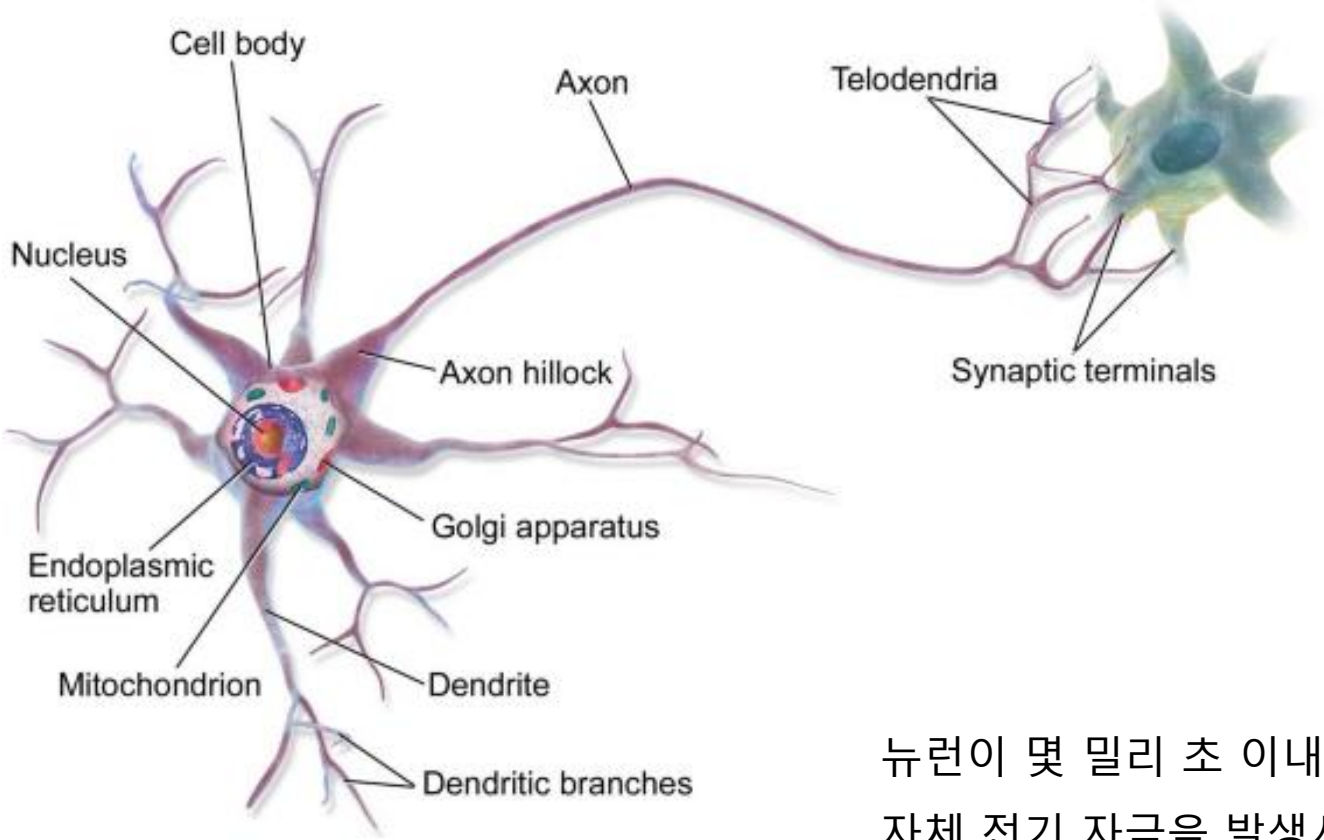
놀랍게도, ANN은 꽤 오랫동안 우리 주변에 있었다:

- ANN은 1943년 신경생리학자 Warren McCulloch와 수학자인 Walter Pitts에 의해 처음 소개
- McCulloch와 Pitts는 획기적인 논문인 "A Logical Calculus of Ideas Immanent in Nervous Activity"에서 생물학적 뉴런들이 명제 논리를 사용하여 복잡한 계산을 수행하기 위해 함께 동작하는 방법에 대한 단순화된 계산 모델 제시
- 최초의 인공 신경망 아키텍처 => 그 이후로 우리가 앞으로 보게 될 많은 다른 아키텍처들이 발명됨

ANN의 초창기 성공 => 곧 진정한 지능형 기계와 대화도 가능할 것이라는 믿음이 널리 퍼짐

- 1960년대(긴 암흑기) : 이 약속이 이루어질 수 없다는 사실이 분명 => 연구지원 펀드 사라짐
- 1980년대 초 : 새로운 아키텍처 개발 & 더 나은 학습 기법 개발 => 신경망 연구에 대한 관심이 다시 부각
- 1990년대 : 발전 느림 => SVM(Support Vector Machines)처럼 다른 강력한 머신러닝 기술 발명
- 신경망 연구 다시 한 번 보류 : 이러한 머신러닝 기술이 ANN보다 더 나은 결과와 더 강력한 이론적 토대를 제공하는 것처럼 보였으므로
- 현재 : 우리는 이제 ANN에 대한 또 다른 관심의 물결을 목도하고 있다!!

Biological Neurons



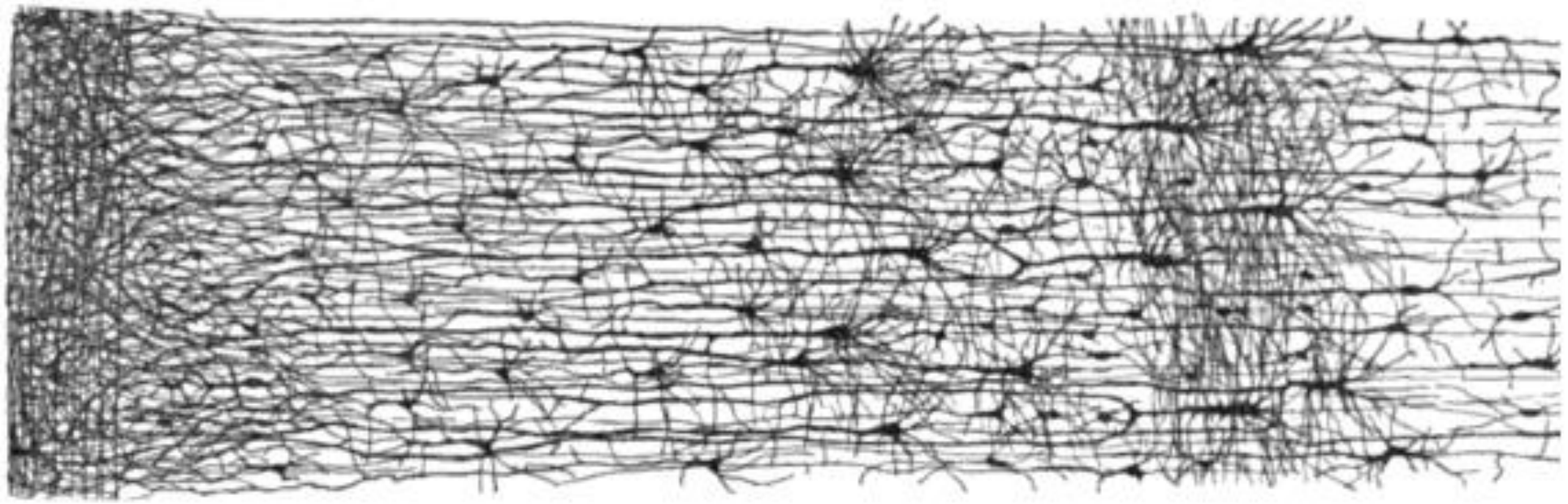
Biological neuron

뉴런이 몇 밀리 초 이내에 충분한 양의 신경 전달 물질을 받으면 자체 전기 자극을 발생시킨다.

Biological Neurons

개별 생물학적 뉴런은 간단한 방식으로 작동하는 것처럼 보이지만 수십억의 광대한 네트워크로 구성

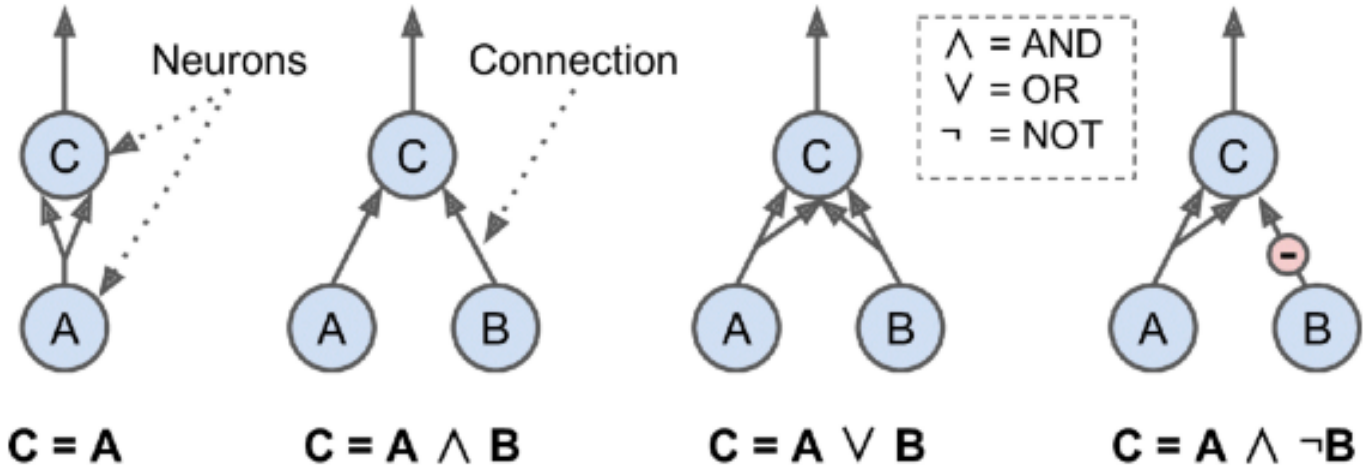
- 각 뉴런은 일반적으로 수천 개의 다른 뉴런에 연결된다.
- 뉴런은 보통 연속적인 층으로 조직되는 것으로 보인다.



Logical Computations with Neurons

McCulloch와 Pitts 제안

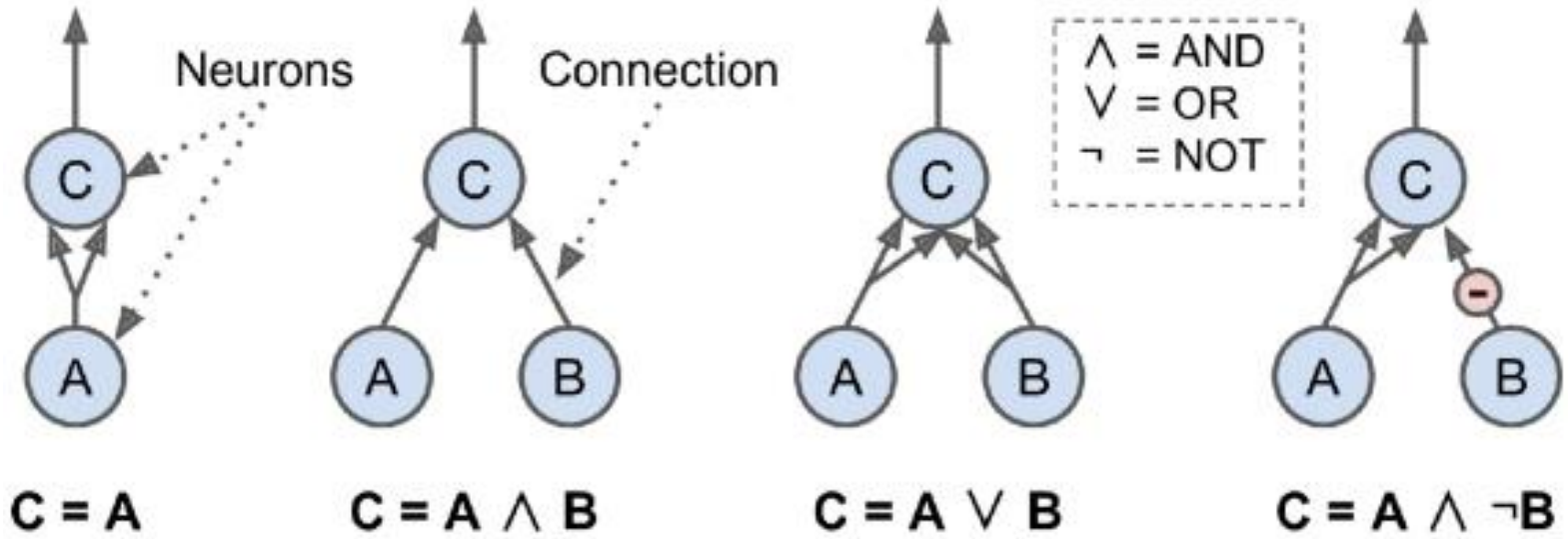
- 매우 간단한 생물학적 뉴런 모델 제안 => 이 모델은 나중에 인공 뉴런으로 알려지게 된다:
  - ✓ 1개 이상의 바이너리(on/off) 입력, 1개의 바이너리 출력
- 인공 뉴런 => 어떤 개수 이상의 입력이 활성화될 때 출력이 활성화됨



ANNs performing simple logical computations

Logical Computations with Neurons

가정: 적어도 2개의 입력이 활성화될 때 뉴런 활성화!



뉴런 A 활성화 => 뉴런 C 활성화
(뉴런 A로부터 2개의 입력 시그널)

Logical AND
(A와 B가 둘 다)

Logical OR
(A 또는 B)

A: 활성화 & B: off일 때만
⇒ C: 활성화

The Perceptron

Perceptron => 가장 간단한 ANN 구조 (Frank Rosenblatt, 1957)

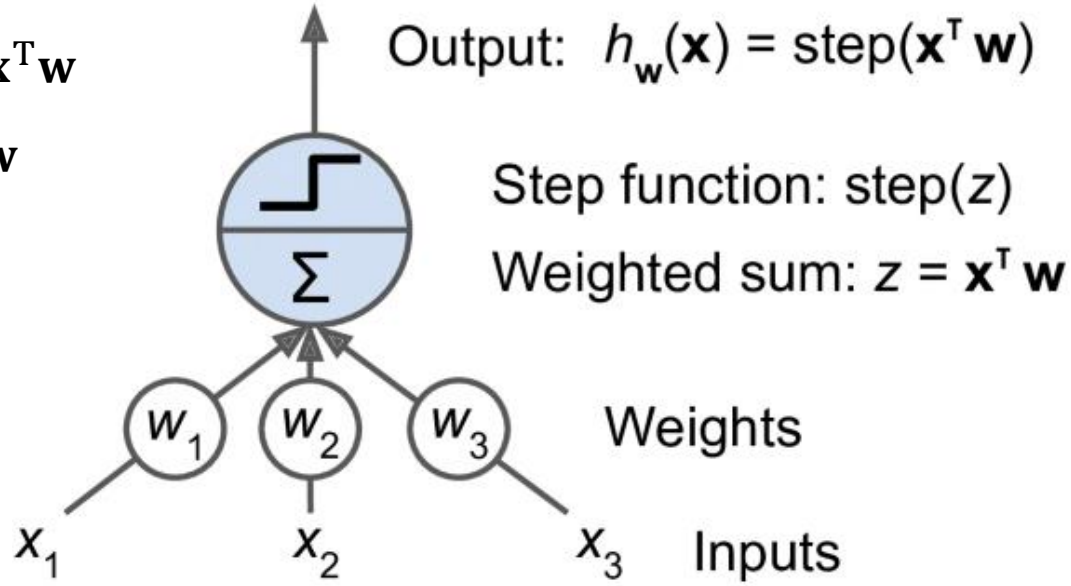
- Threshold Logic Unit (TLU)라는 약간 다른 형태의 인공 뉴런에 기반
- 입력과 출력이 바이너리 on/off 값 대신 숫자 / 각 입력은 가중치로 연결
- TLU 계산:

$$z = w_1 x_1 + w_1 x_1 + \cdots + w_1 x_1 = \mathbf{x}^{\mathrm{T}}\mathbf{w}$$

$$h_{\mathbf{w}}(\mathbf{x}) = \mathrm{step}(z), \quad where\ z = \mathbf{x}^{\mathrm{T}}\mathbf{w}$$

입력의 가중치 합 계산

Step function 적용



Perceptron에 흔하게 사용되는 step 함수: *Heaviside step* 함수, sometimes sign 함수 사용

Common step functions used in Perceptrons (assuming threshold = 0)

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

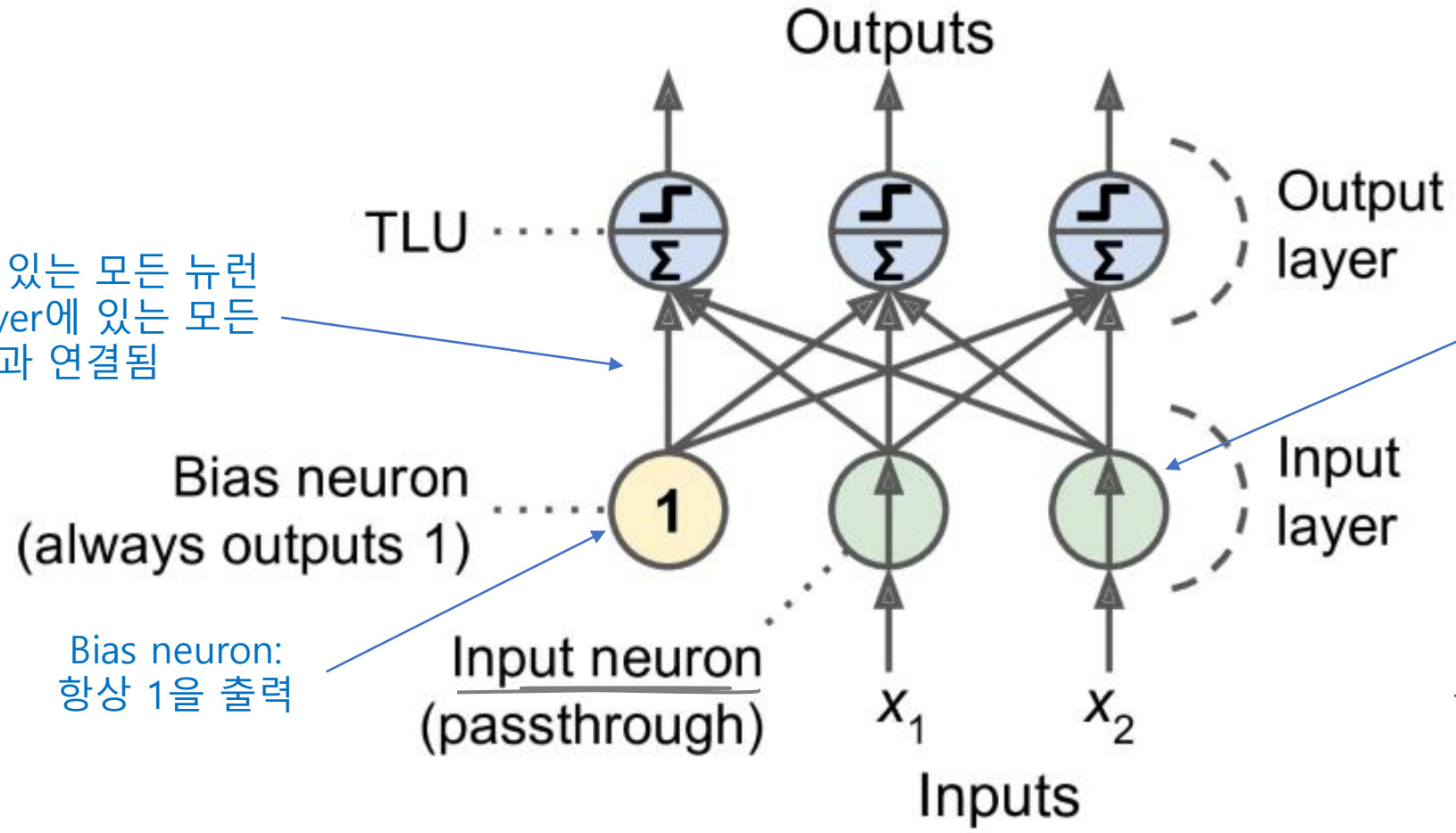
- Single TLU => 간단한 Linear Binary Classification에 사용될 수 있음
- 입력에 대해 선형 조합 계산

  ⇒결과가 임계 값을 초과하면 positive class 출력

  ⇒그렇지 않으면 negative class 출력
- Logistic Regression 또는 Linear SVM Classifier와 비슷함
- TLU 학습 => w_0, w_1, and w_2에 대한 적절한 값을 찾는 일

The Perceptron

2개의 입력과 3개의 출력을 갖는 Perceptron:



한 layer에 있는 모든 뉴런은 다른 layer에 있는 모든 뉴런과 연결됨

그대로 통과!

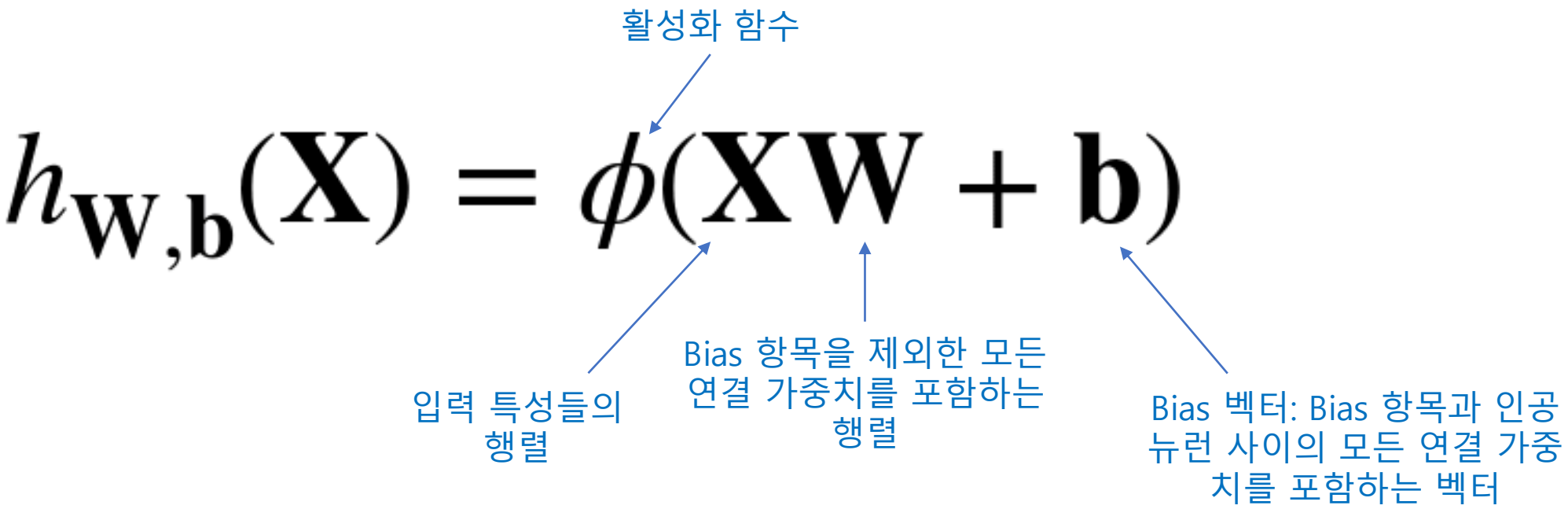
Bias neuron:
항상 1을 출력

동시에 3개의 클래스를 분류할 수 있다!
(multioutput classifier)

Architecture of a Perceptron with two input neurons, one bias neuron, and three output neurons

선형대수 덕분에 여러 개의 인스턴스에 대해 인공 뉴런들로 구성된 레이어의 출력을 동시에 효율적으로 계산 가능

- Computing the outputs of a fully connected layer



The Perceptron

Hebbian learning : 예측 수행 시 네트워크에 의해 발생된 에러를 고려하는 법칙

Perceptron learning : 출력 에러를 줄이도록 가중치 연결 값을 조절

- ✓ 한 번에 1개의 인스턴스 입력시킴, 각 인스턴스에 대해 예측 수행

Perceptron 학습 알고리즘

- Perceptron learning rule (weight update)

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

- $w_{i,j}$ is the connection weight between the i^{th} input neuron and the j^{th} output neuron.
- x_i is the i^{th} input value of the current training instance.
- $\hat{y}_j$ is the output of the j^{th} output neuron for the current training instance.
- y_j is the target output of the j^{th} output neuron for the current training instance.
- η is the learning rate.

각 출력 뉴런의 decision boundary => linear

- Perceptron은 복잡한 패턴을 학습할 수 없다!
- Training set의 인스턴스들이 선형적으로 분리 가능하면(linearly separable) 솔루션에 수렴 가능 (Rosenblatt이 증명)
- Scikit-Learn의 `Perceptron` 클래스 이용 => single-TLU 네트워크 구현

```
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()
per_clf.fit(X, y)

y_pred = per_clf.predict([[2, 0.5]])
```

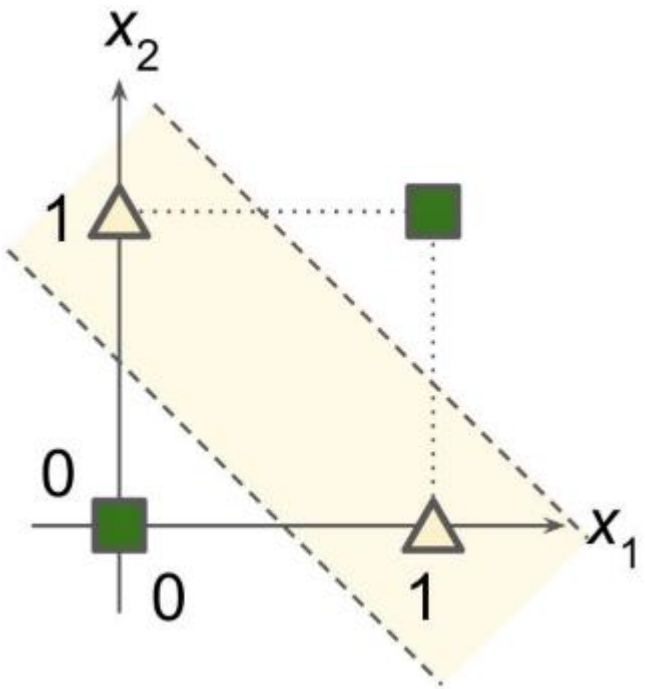
Iris dataset

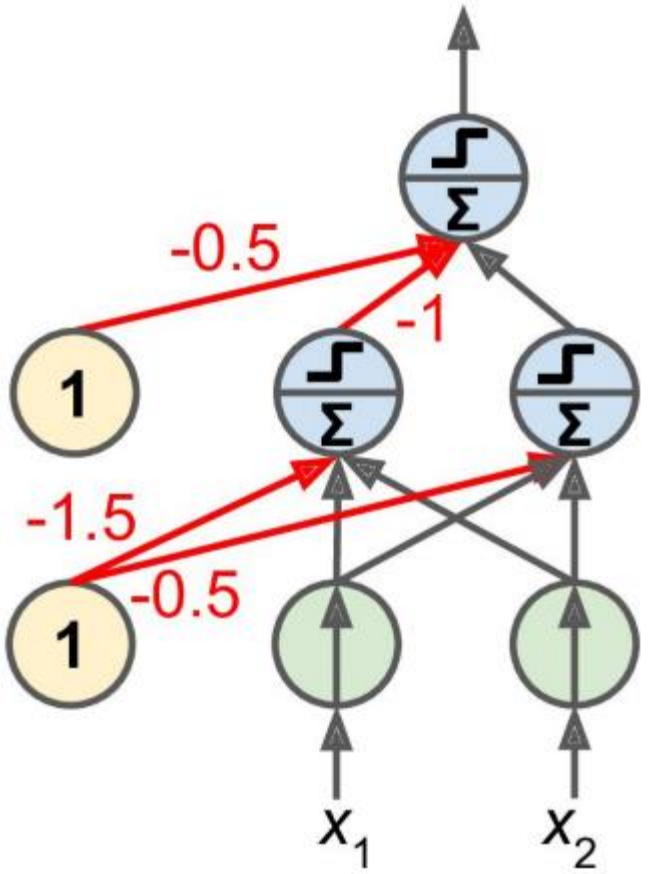
Perceptron 학습 알고리즘은 Stochastic Gradient Descent와 흡사하다!

Monograph **Perceptrons** (Marvin Minsky and Seymour Papert, 1969)

- Perceptron의 심각한 문제점을 지적
- 특히, Perceptron이 Exclusive OR (XOR) classification 문제를 풀 수 없음을 증명함

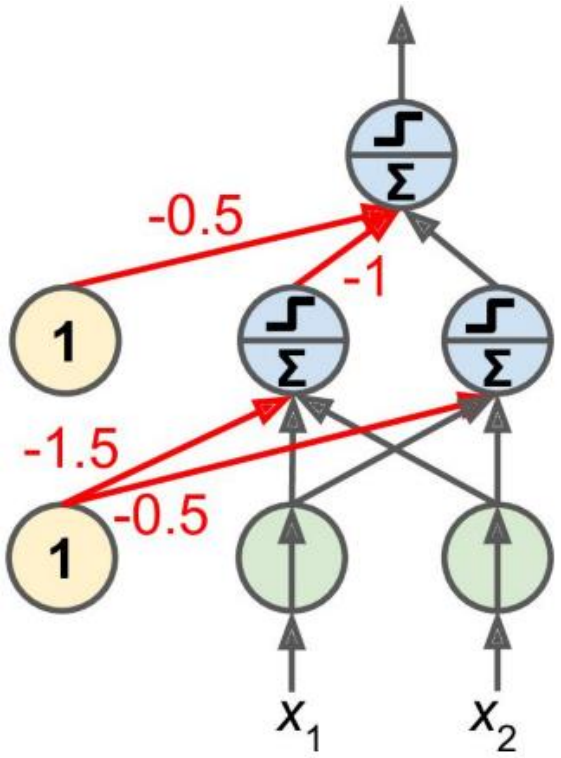
직선을 하나만 그어서는 문제를 풀 수 없음!
(모든 선형 분류 모델에 해당)





여러 개의 Perceptron을 이용하면 문제 해결
⇒ Multilayer Perceptron (MLP)

The Perceptron



x_1	x_2	TLU(left)	TLU(right)	Output
0	0	(0+0-1.5) => 0	(0+0-0.5) => 0	(0+0-0.5) => 0
0	1	(0+1-1.5) => 0	(0+1-0.5) => 1	(0+1-0.5) => 1
1	0	(1+0-1.5) => 0	(1+0-0.5) => 1	(0+1-0.5) => 1
1	1	(1+1-1.5) => 1	(1+1-0.5) => 1	(-1+1-0.5) => 0

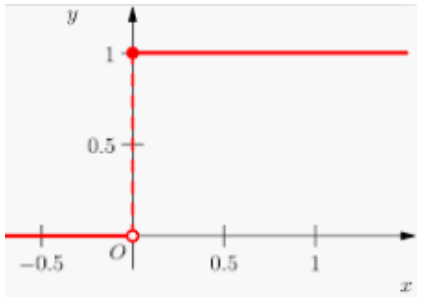
$Predict$ y=1 if $-1.5 + x_1 + x_2 \geq 0$

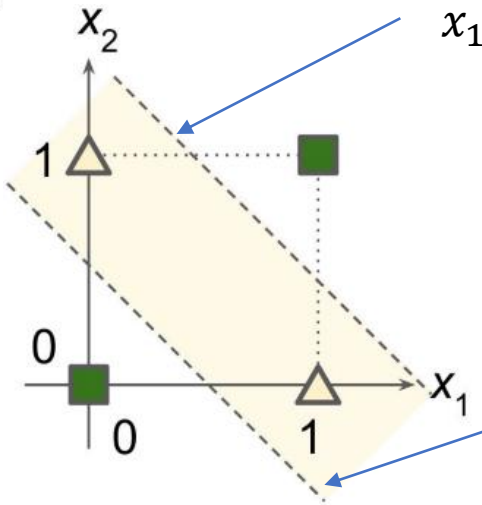
$x_1 + x_2 \geq 1.5$

$-(x_1 + x_2) \geq 1.5$

$x_1 + x_2 < 1.5$

$$\begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix}$$



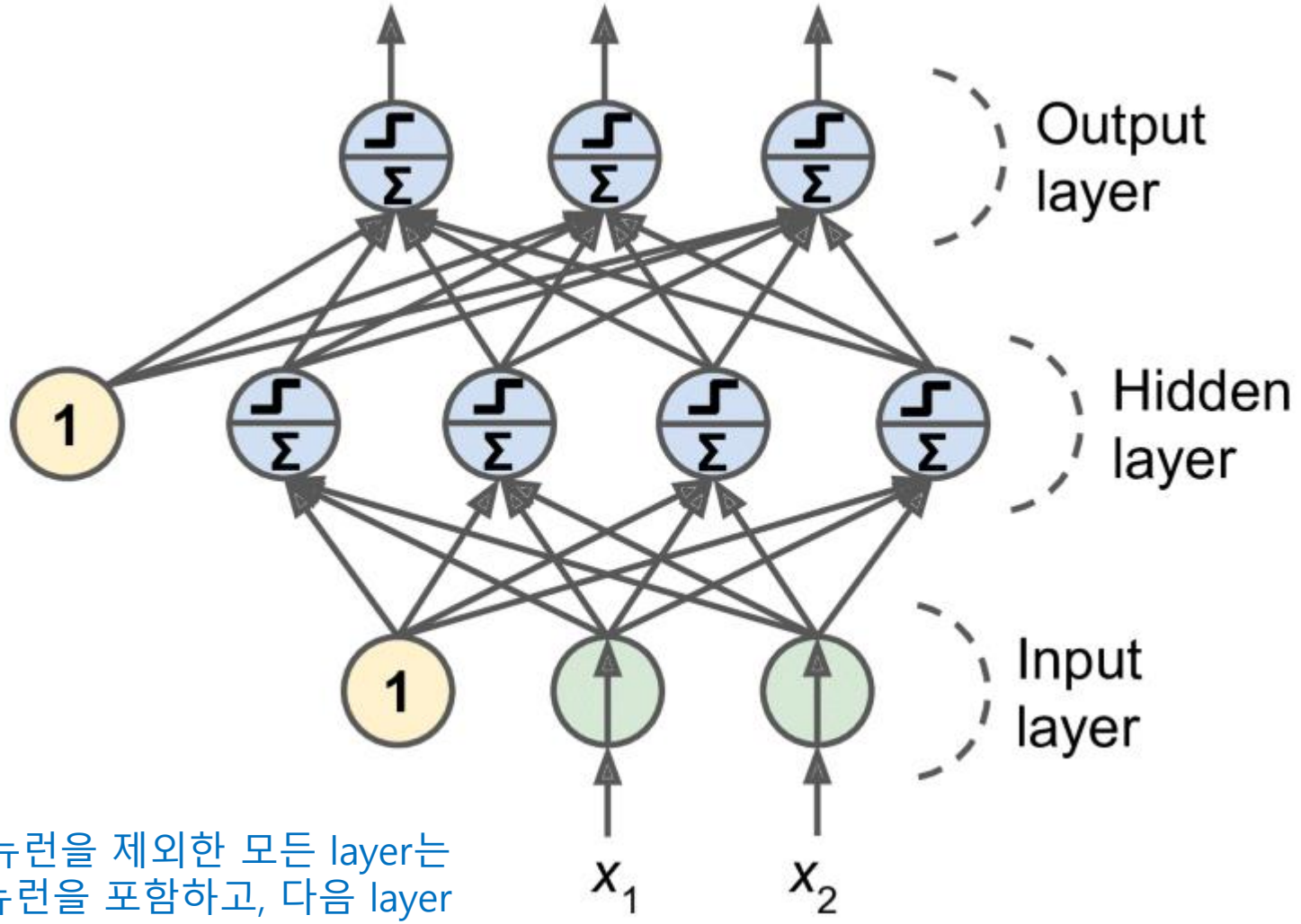


$Predict$ y=1 if $-0.5 + x_1 + x_2 \geq 0$

$x_1 + x_2 \geq 0.5$

$$\begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}$$

The Multilayer Perceptron and Backpropagation



2개의 입력, 4개의 뉴런으로 구성된 1개의 hidden layer, 3개의 출력 뉴런을 갖는 MLP 구조

시그널이 한쪽 방향으로만 흐른다 하여 이런 구조를 "feedforward neural networks(FNN)이라 함.

출력 뉴런을 제외한 모든 layer는 bias 뉴런을 포함하고, 다음 layer와 fully connected!

ANN이 심층(deep) 은닉층들을 가질 때 deep neural network (DNN)이라 함

- Deep Learning => DNN을 연구, 보통 심층 스택으로 된 모델에서 계산을 연구
- 사람들은 심층이 아니더라도 그냥 Deep Learning이라 한다.

MLP를 학습시키기 위한 방법 찾기 위해 수년 동안 노력함

- 드디어 backpropagation 알고리즘 발견 (Rumelhart, Hinton, and Williams, 1986)
- 오늘날에도 여전히 활발히 사용되고 있음

Backpropagation 알고리즘

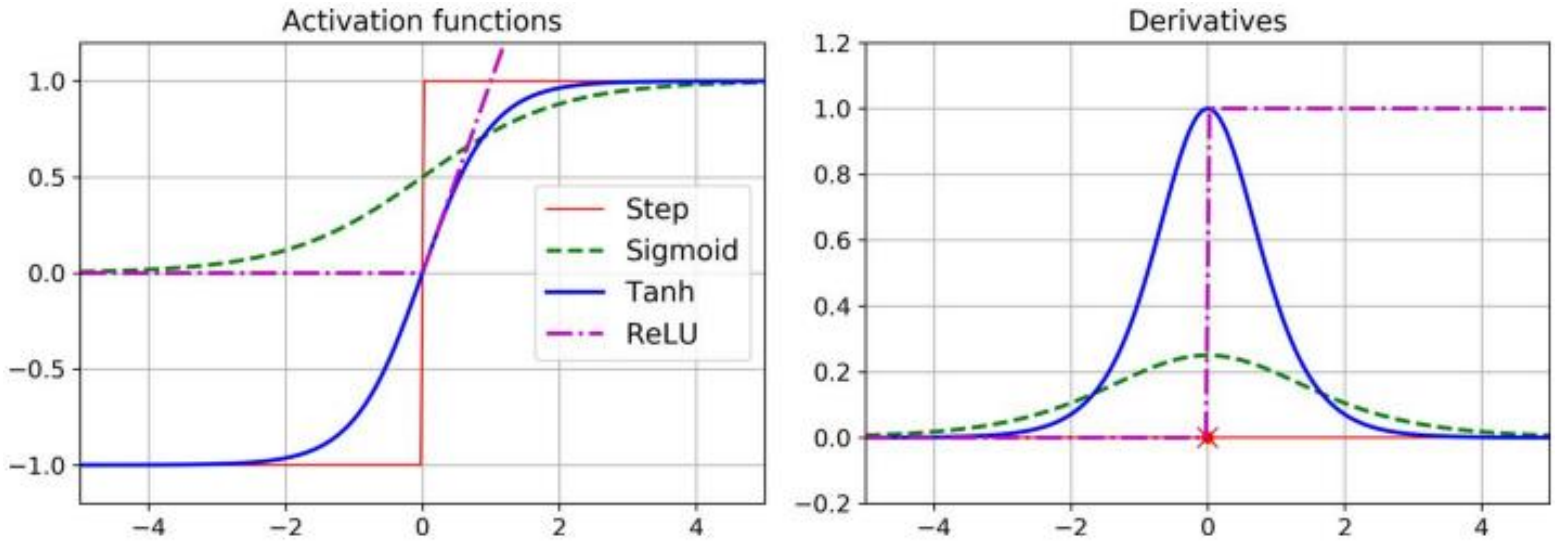
- 네트워크 에러의 기울기를 자동으로 계산하기 위해 Gradient Descent 방법 사용
- One forward, one backward 두 가지 단계로 기울기를 계산
- 에러를 줄이는 방향으로 연결 가중치와 bias term을 조정
- 네트워크가 수렴할 때까지 전체 과정 반복

Backpropagation 알고리즘이 동작하기 위해서는 MLP 구조를 변경해야

- Step 함수 => 활성 함수를 Logistic(Sigmoid) 함수 $\sigma(z) = \frac{1}{1+e^{-z}}$로 변경
- Step 함수는 평편해서 기울기를 구할 수 없다 => 기울기를 구할 수 없다면 Gradient Descent 방법을 쓸 수 없다.
- 기타 활성화 함수들

  $Hyperbolic\ tangent\ function$: $\tanh(z) = 2\sigma(2z) - 1$

  $Rectified\ Linear\ Unit\ function$: $ReLU(z) = \max(0, z)$



왜 활성화 함수가 필요한가?

- 여러 개의 선형 변환을 체인으로 연결해도 얻는 것은 역시 선형 변환이다.
- 예를 들어, $f(x) = 2x + 3 \; and \; g(x) = 5x - 1$ 두 함수를 체인으로 엮는다 해도 또 다른 선형 함수일 뿐이다:

$$f\big(g(x)\big) = 2(5x - 1) + 3 = 10x + 1$$

- 레이어 간에 비선형성이 없다면 아무리 많은 레이어 스택을 쌓아도 단일 레이어와 동일하다.
- 선형 함수만 가지고서는 매우 복잡한 문제를 해결할 수 없다!

Regression MLPs

MLP => regression 작업에 사용될 수 있다.

- 1개의 값을 예측한다면 1개의 출력 뉴런이 필요하다.
- 다변량 회귀의 경우 출력 차원당 1개의 출력 뉴런을 가져야 한다:
  - ✓ 예를 들어, 이미지에서 물체의 중심을 찾기 위해서는 2D 좌표 값이 필요하고 따라서 2개의 출력 뉴런이 필요하다. 또 물체의 주변에 사각 박스를 표시하려면 2개의 숫자(폭과 높이)가 더 필요하다. 따라서 총 4개의 출력 뉴런이 필요하게 된다.
- 보통 Regression을 위한 MLP를 만들 경우 출력 뉴런에 활성화 함수를 사용하지 않는다.
- 그러나 출력 값이 항상 positive 값을 갖기를 원하면 ReLU 활성화 함수를 사용할 수 있다.
- 학습에 사용되는 loss 함수는 보통 MSE이다.
- 그러나 training set에 outlier가 많다면 mean absolute error (MAE)를 쓰는 것이 좋다.

Classification MLPs

MLP => 분류 작업에 사용될 수 있다:

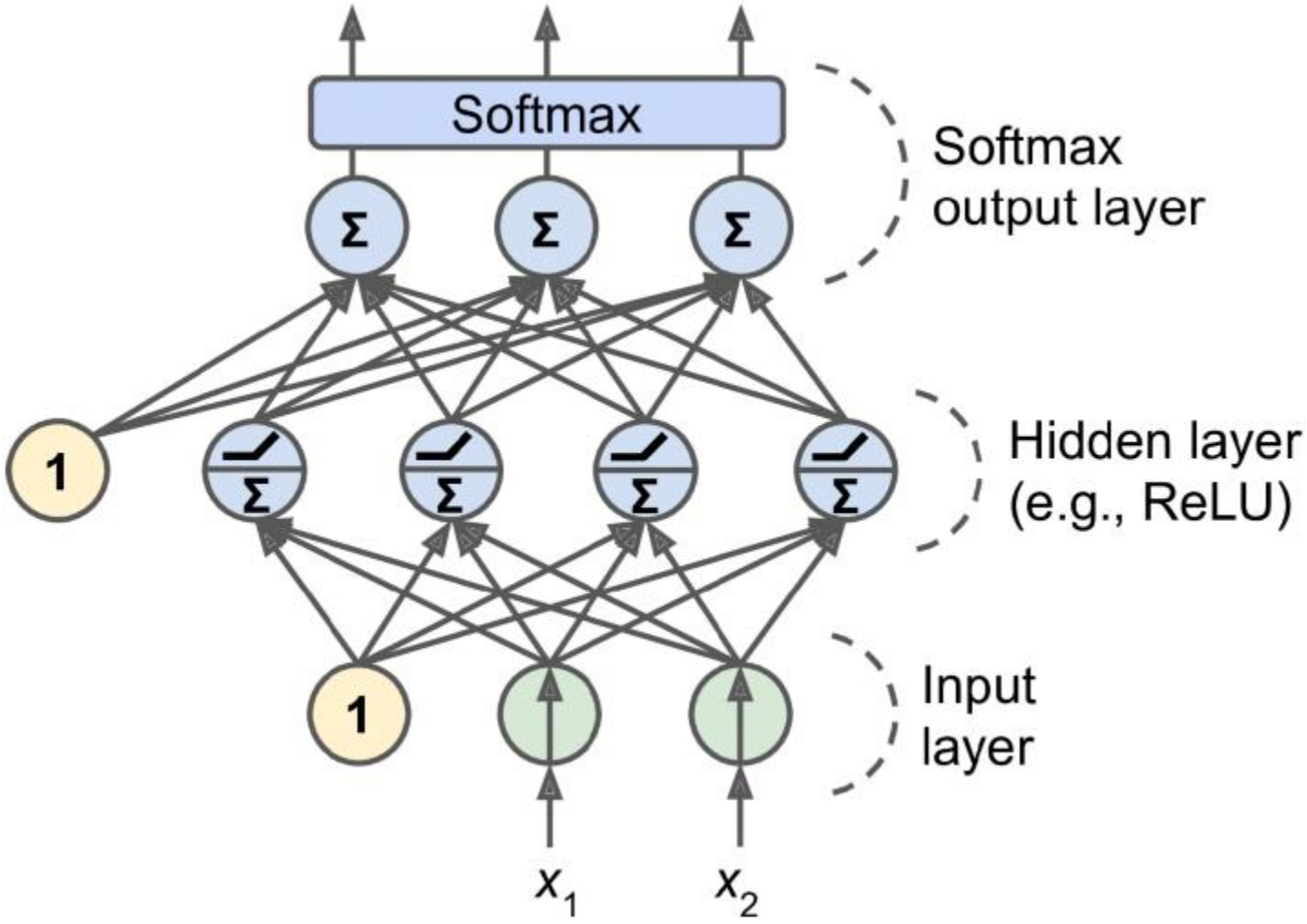
- Binary 분류인 경우 => logistic 활성화 함수를 이용하는 1개의 출력 뉴런이 필요
- 출력 값은 0과 1 사이의 숫자 => 이 숫자를 positive 클래스의 **'추정된 확률'**로 해석 가능

MLP => multilabel binary classification 작업을 쉽게 해결할 수 있다.

- 각 인스턴스가 여러 개의 클래스 중 1개의 클래스에만 속한다면 클래스당 1개의 출력 뉴런이 있어야
- 전체 출력 레이어에 ***softmax*** 활성화 함수를 사용
- Softmax 함수 => 모든 추정된 확률이 0과 1 사이의 값이고, 합해서 1이 됨을 보장
- **Multiclass** classification이라고 함

Classification MLPs

A modern MLP (including ReLU and softmax) for classification:



실습과제 11-1

- 본문에 나오는 전체 내용 PyCharm에서 실행하기

★ **반드시 결과 값 및 결과로 나온 모든 그래프에 대해 자세한 분석을 하시오.**

참고: 제 11강 실습과제 11-1 Introduction to Artificial Neural Networks [1] - Perceptrons & MLP.pdf

실습과제 11-2

The TensorFlow Playground is a handy neural network simulator built by the TensorFlow team. In this exercise, you will train several binary classifiers in just a few clicks, and tweak the model's architecture and its hyperparameters to gain some intuition on how neural networks work and what their hyperparameters do. Take some time to explore the following:

a. The patterns learned by a neural net. Try training the default neural network by clicking the Run button (top left). Notice how it quickly finds a good solution for the classification task. The neurons in the first hidden layer have learned simple patterns, while the neurons in the second hidden layer have learned to combine the simple patterns of the first hidden layer into more complex patterns. In general, the more layers there are, the more complex the patterns can be.

b. Activation functions. Try replacing the tanh activation function with a ReLU activation function, and train the network again. Notice that it finds a solution even faster, but this time the boundaries are linear. This is due to the shape of the ReLU function.

c. The risk of local minima. Modify the network architecture to have just one hidden layer with three neurons. Train it multiple times (to reset the network weights, click the Reset button next to the Play button). Notice that the training time varies a lot, and sometimes it even gets stuck in a local minimum.

d. What happens when neural nets are too small. Remove one neuron to keep just two. Notice that the neural network is now incapable of finding a good solution, even if you try multiple times. The model has too few parameters and systematically underfits the training set.

e. What happens when neural nets are large enough. Set the number of neurons to eight, and train the network several times. Notice that it is now consistently fast and never gets stuck. This highlights an important finding in neural network theory: large neural networks almost never get stuck in local minima, and even when they do these local optima are almost as good as the global optimum. However, they can still get stuck on long plateaus for a long time.

f. The risk of vanishing gradients in deep networks. Select the spiral dataset (the bottom-right dataset under "DATA"), and change the network architecture to have four hidden layers with eight neurons each. Notice that training takes much longer and often gets stuck on plateaus for long periods of time. Also notice that the neurons in the highest layers (on the right) tend to evolve faster than the neurons in the lowest layers (on the left). This problem, called the "vanishing gradients" problem, can be alleviated with better weight initialization and other techniques, better optimizers (such as AdaGrad or Adam), or Batch Normalization.

g. Go further. Take an hour or so to play around with other parameters and get a feel for what they do, to build an intuitive understanding about neural networks.