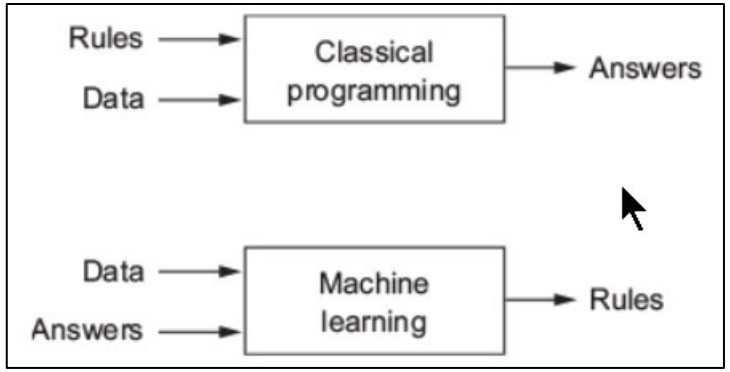
머신러닝 개요 - 역사

- ‘컴퓨터가 사람과 같이 생각할 수 있는가’라는 질문에서 시작.
- Turing test
- 보통 사람이 수행하는 지능적인 작업을 자동화하기 위한 연구활동(F. Chollet)



- Symbolic AI
  - 프로그래머들이 지식을 명시적인 규칙으로 충분히 많이 만들어 구현할 수 있을 것이다.
  - 1980년대의 전문가 시스템(체스 기계, 의료분야 시스템)
- Symbolic AI의 문제점
  - 유지보수에 큰 비용이 소요됨.
  - 이미지, 음성인식, 언어처리 등 복잡하고 불문명한 문제의 규칙을 찾는 것은 어려움.
- 머신러닝
  - 컴퓨터가 데이터를 보고 처리규칙을 학습할 수 있는가.

머신러닝 개요 - 머신러닝

- "컴퓨터에 명시적인 프로그램 없이 배울 수 있는 능력을 부여하는 연구 분야" (아서 사무엘, 1959).
- 많은 학습데이타를 제공하면 이 데이터에서 통계적인 구조를 찾고 그 과정을 자동화하기 위한 규칙을 만들어냄.
  - 데이터와 이 데이터로부터 기대되는 해답을 입력하면 규칙을 출력
  - 프로그램 되는 것이 아니라 훈련됨.

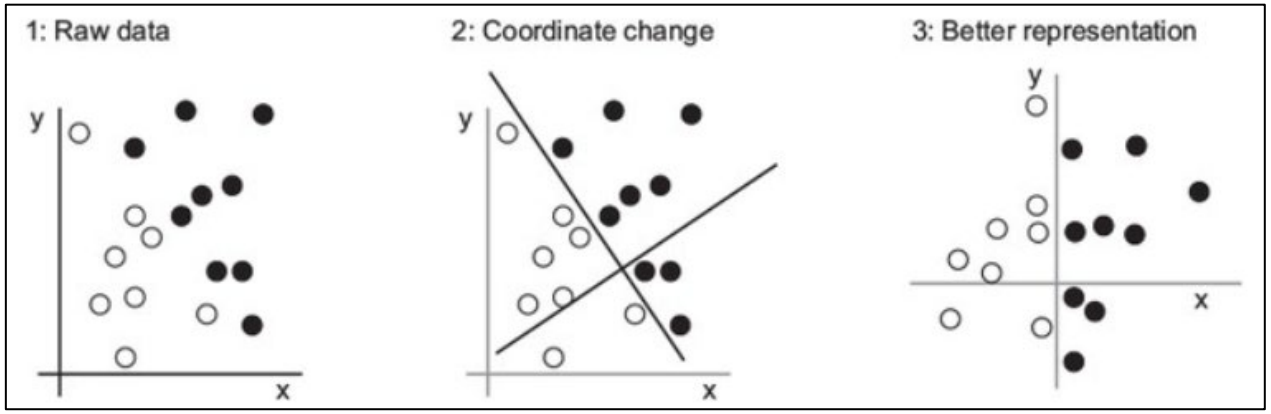
- 머신러닝의 필요조건
  - 입력 데이타포인트(샘플) : 대상데이타 중 규칙을 찾기 위한 문제에 해당하는 데이터
  - 기대출력 : 입력데이타포인트와 짝이 되는 기대값
  - 알고리즘 성능측정방법 : 알고리즘의 현재출력과 기대출력(정답)간의 차이를 결정하는 방법

- 머신러닝이란(딥러닝)
  - 가능성 있는공간을 사전 정의하고 피드백 신호의 도움을 받아 입력 데이터에 대한 유용한 변환을 (자동으로) 찾는 작업.

머신러닝 개요 – 간단한 예

- 머신러닝이란 : 가능성 있는공간을 사전 정의하고 피드백 신호의 도움을 받아 입력 데이터에 대한 유용한 변환을 (자동으로) 찾는 작업.

- 머신러닝의 간단한 예
  - 문제 : 흰점과 검은 점을 구분하는 알고리즘.
  - 입력 : 점의 좌표
  - 출력 : 흰점/검은 점 판별
  - 성능측정 : 정확히 분류한 비율



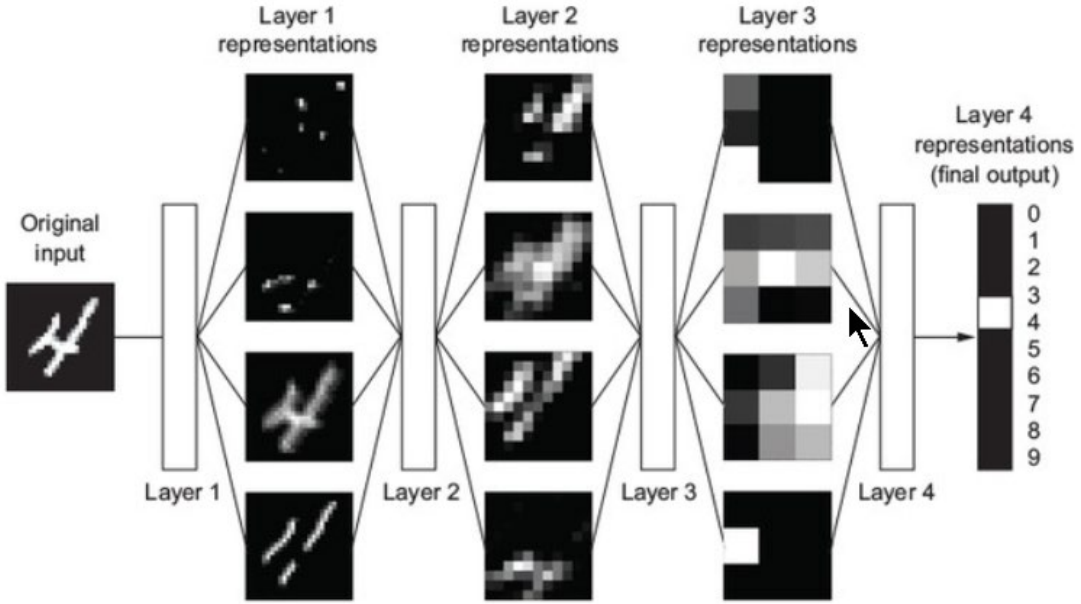
- 일반적인 해결방법(데이터 특징을 알고 있음)
  - y 축의 위-아래로 점들이 나뉘도록 좌표계를 변환

- 머신러닝 방법(데이터 특징을 모름)
  - 임의의 x-y축 설정
  - 정확히 분류한 비율 계산
  - 정확도가 최대가 되는 축을 찾을때까지 반복

머신러닝 개요 - 딥러닝

- 딥러닝
  - 머신러닝의 한 분야로서 연속된 층이 점진적으로 의미있는 표현을 배우는 방식.
  - 각 층은 신경망으로 이루어짐.
  - 딥 : 여러 층을 연결함. cf. 전통머신러닝은 얕은 학습구조.
- 숫자인식 과정 예
  - 각층은 진행됨에 따라 최종출력에 대해 점점 더 많은 정보를 가짐.
  - 원본이미지와는 다른 표현으로 숫자이미지가 변환됨.

Figure 1.6. Deep representations learned by a digit-classification model

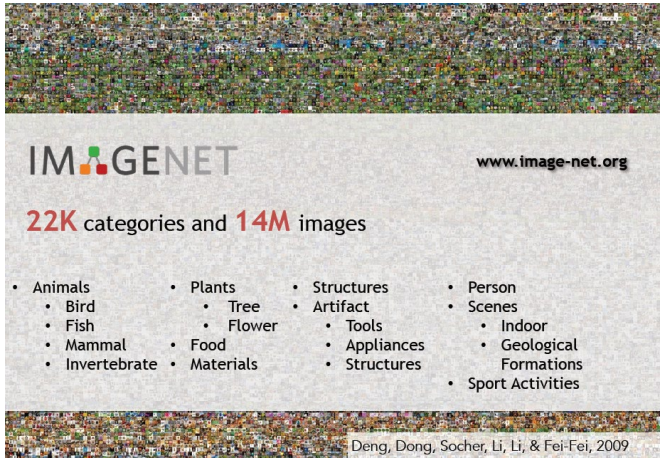


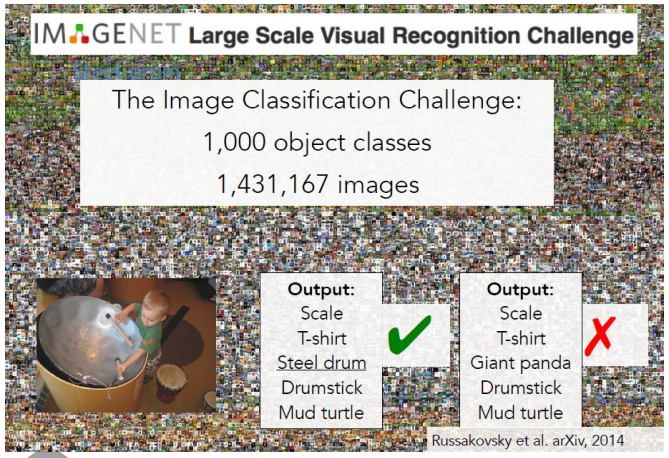
머신러닝 기술의 발전과정

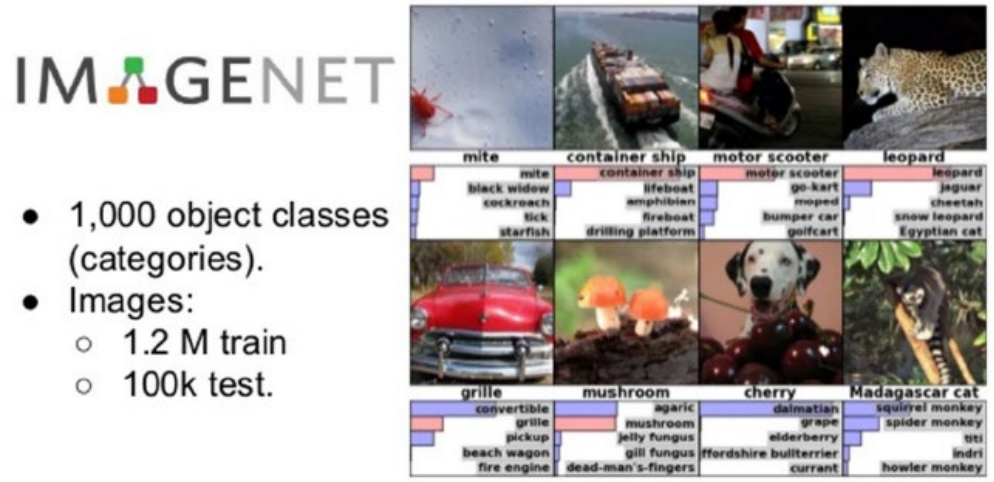
- Logistic Regression
  - 분류 알고리즘. 컴퓨터 이전부터 존재했던 알고리즘이며 지금도 기본 성능을 확인하기 위해 사용됨.
- 초창기 신경망
  - 1950년대 제안되었으나 큰 신경망을 학습시킬 수 있는 방법을 찾지 못하였음.
  - 1980년대 중반 경사하강법을 사용한 역전파 알고리즘이 개발되면서 큰 신경망 학습가능하게 됨.
  - CNN(Convolutional Neural Network)을 이용하여 손글씨 숫자를 분류하는 LeNet 발표(Yann LeCun , 1989)
- 커널 방법(1990년대 중반)
  - 커널 기법 : 분류문제를 간단하게 만들기 위해 데이터를 고차원으로 표현하기 위해 새로운 공간에서 (두 포인트의 좌표를 구하지 않고) 두 포인트 사이의 거리만 계산하는 기법.
  - 간단한 분류문제에 최고 수준의 성능 보임.
  - 대량 데이타셋으로 확장이 어렵고 이미지 등 지각 관련 데이터에 대한 성능 떨어짐.
- 결정트리, 랜덤포레스트, 그래디언트부스팅(2010년대)
  - 랜덤포레스트 : 서로 다른 결정트리를 많이 만들고 출력을 종합. 2010 Kaggle 경연이 시작될 때 가장 선호하는 알고리즘임.
  - 그래디언트 부스팅 : 이전 모델에서 잘못 예측한 데이터를 잘 예측하도록 보완하는 모델. 2014년 이후 가장 우수한 성능을 보이는 모델.
- 신경망(2012 ~)

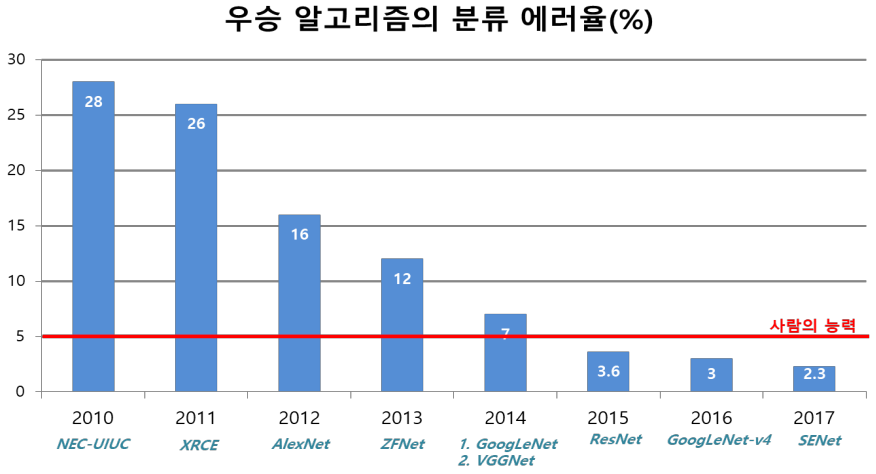
머신러닝 기술의 발전과정 - 신경망

- Stanford 대학에서 주최한 이미지 맞추기 경진대회(2010 ~ 2017)
- 2012년, Hinton 팀이 CNN 기법을 이용하여 획기적으로 성능을 향상시킴.





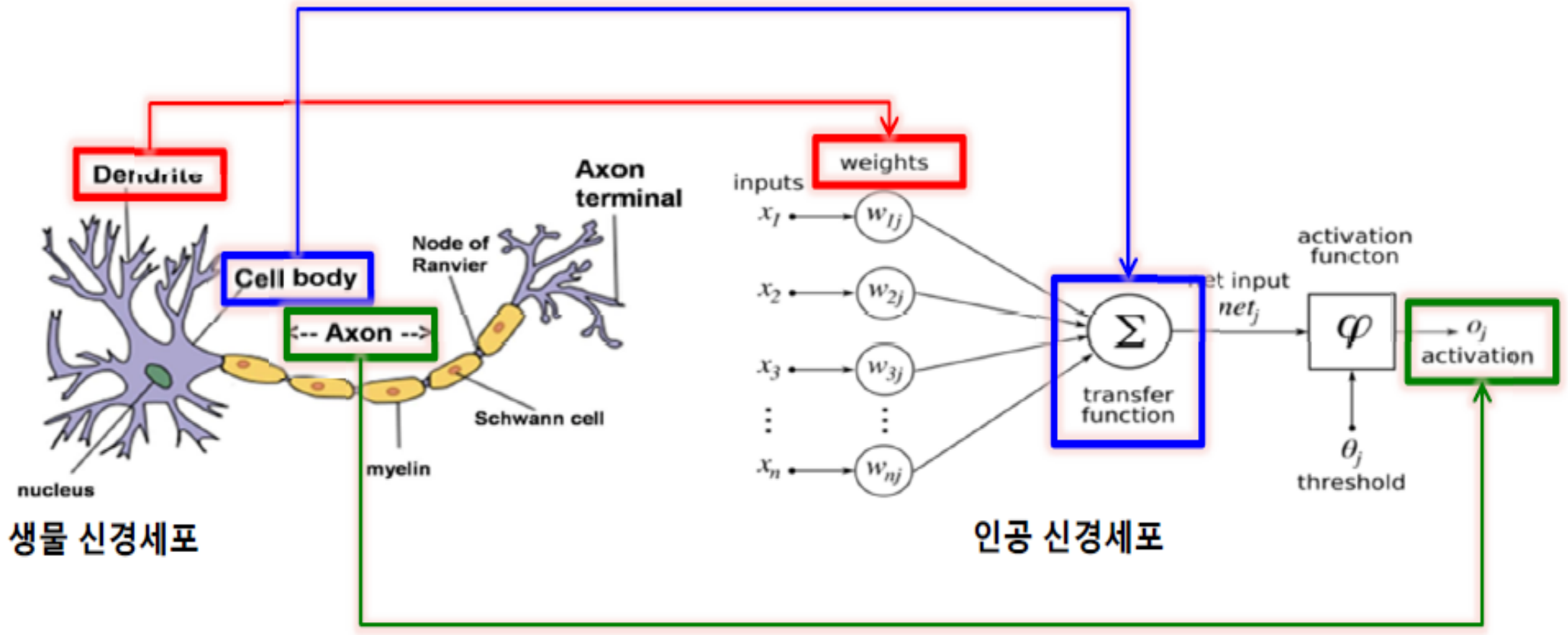




머신러닝의 성공요인

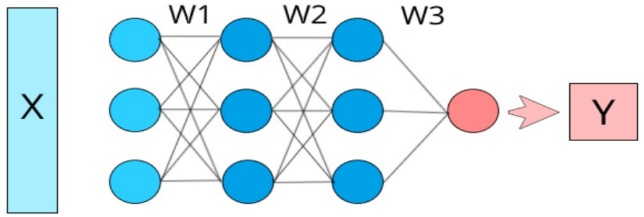
- 중요한 알고리즘들은 1990대 이전에 이미 개발됨.
- 2012년 이후 큰 주목을 받게된 이유
- 하드웨어의 발전
  - 인터넷의 발전으로 인한 컴퓨터 하드웨어 고성능화
  - 게임산업의 발전으로 인한 GPU 고성능화(딥러닝에 중요)
- 풍부한 데이타셋
  - 인터넷의 발전으로 풍부한 이미지, 텍스트, 비디오 데이터 확보 가능.
- 딥러닝 알고리즘 진전
  - 2000년대 후반까지는 2개층 정도의 얕은 신경망 수준 학습가능
  - 2009~2010 년경 그래디언트 역전파를 위한 중요한 알고리즘 진전
  - 활성화함수 개선(relu)
  - 효율적인 가중치 초기화 방법(Xavier Initialization)
  - Optimizer개선(RMSProp, Adam)
  - Batch Normalization, Residual Connection 등.
- 개발도구의 대중화
  - 기본 파이선 스크립트 기술만 있으면 머신러닝 공부 및 구현이 가능함.
  - Theano, Tensorflow : 자동미분지원
  - scikit-learn, Keras : 모든 머신러닝 모델의 구현 라이브러리 제공

신경망의 구조

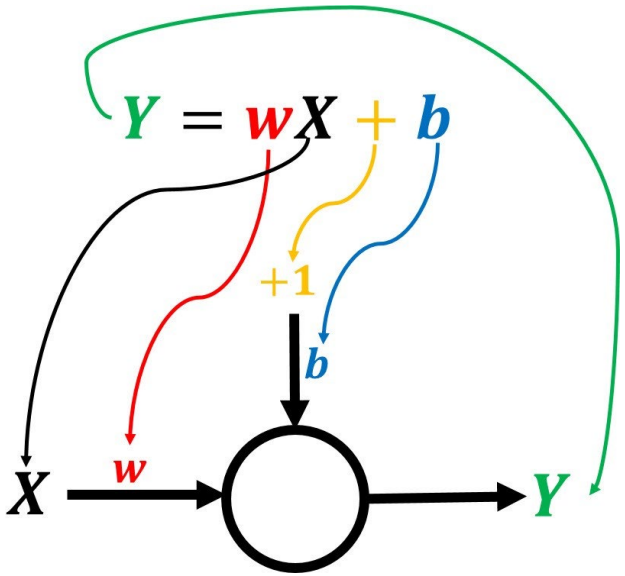


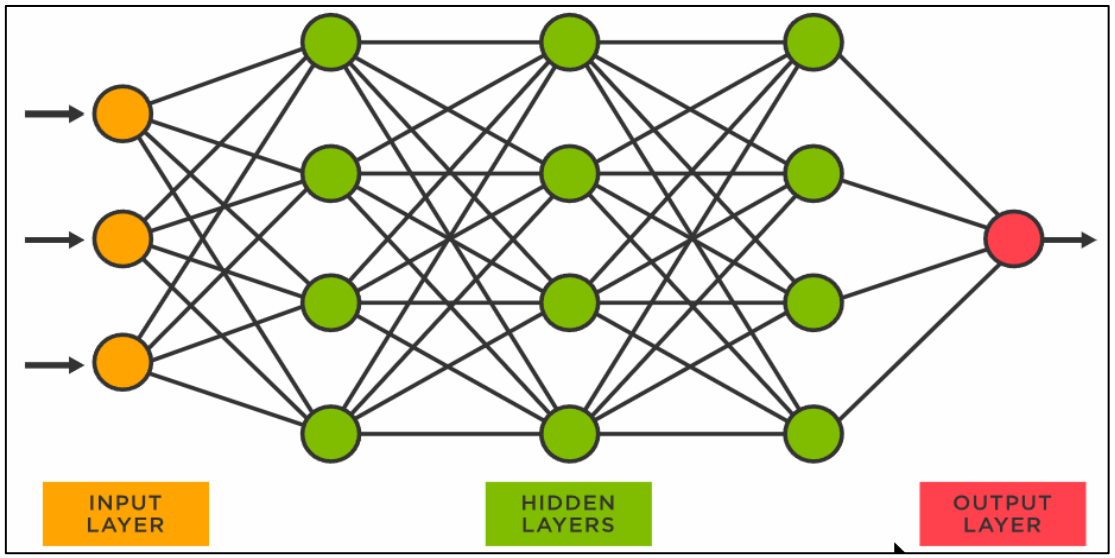
인공 신경세포 (Artificial Neuron)

신경망의 구조



$$Y = f(W_3 * f(W_2 * f(W_1 * X)))$$





$$f(x) = g \circ f_K \circ \dots f_2 \circ f_1(x)$$
$$= g(a(\dots a(w_2 a(w_1 x + b_1) + b_2) \dots + b_K))$$

실습 환경 구성

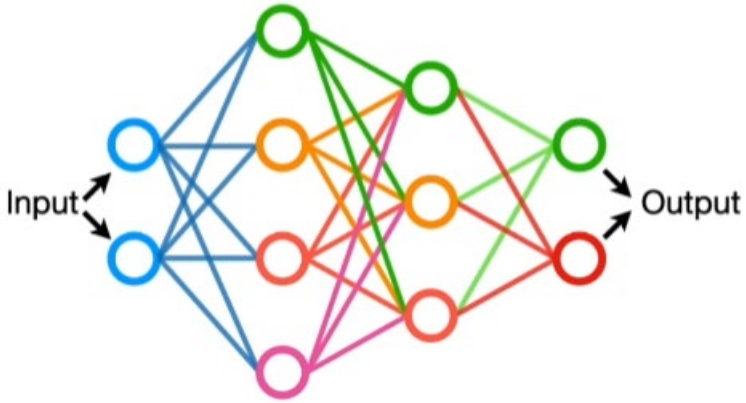
- $ conda deactivate
- $ conda create --name dl python=3.8 tensorflow
- $ conda activate dl
- $ python -c 'import tensorflow as tf;print(tf.reduce_sum(tf.random.normal([1000, 1000])))'

- $ conda install pip ipykernel
- $ python -m ipykernel install --user --name dl
- $ pip install keras numpy pandas matplotlib sklearn
- # 모델의 그래프 표현 도구 설치
- pip install graphviz 또는 sudo apt install graphviz
- pip install pydot

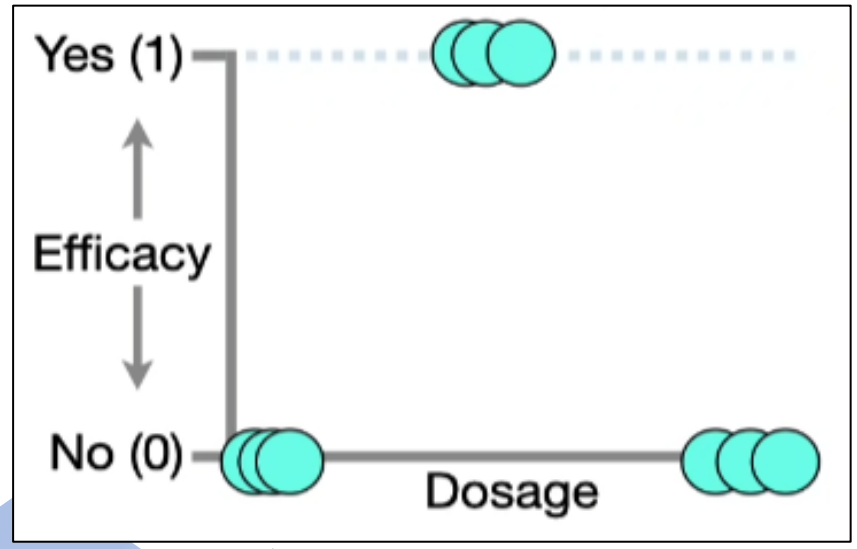
신경망 예제

- 실습 – MNIST 데이타셋
  - 1980년대 미국 표준연구소(NIST)에서 수집한 숫자 필기 데이타셋
  - 6만개의 훈련이미지. 1만개의 테스트 이미지로 구성됨.
  - 데이타는 0 ~ 9 까지의 28 x 28 픽셀로 구성됨.

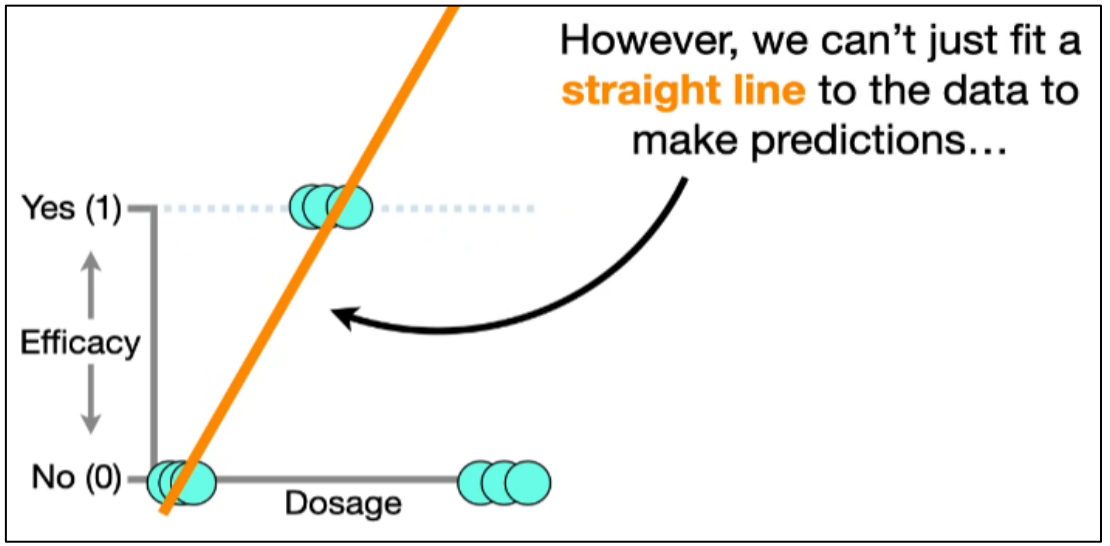
신경망 작동예제



문제 : 투약량을 학습하여 치료효과 예측



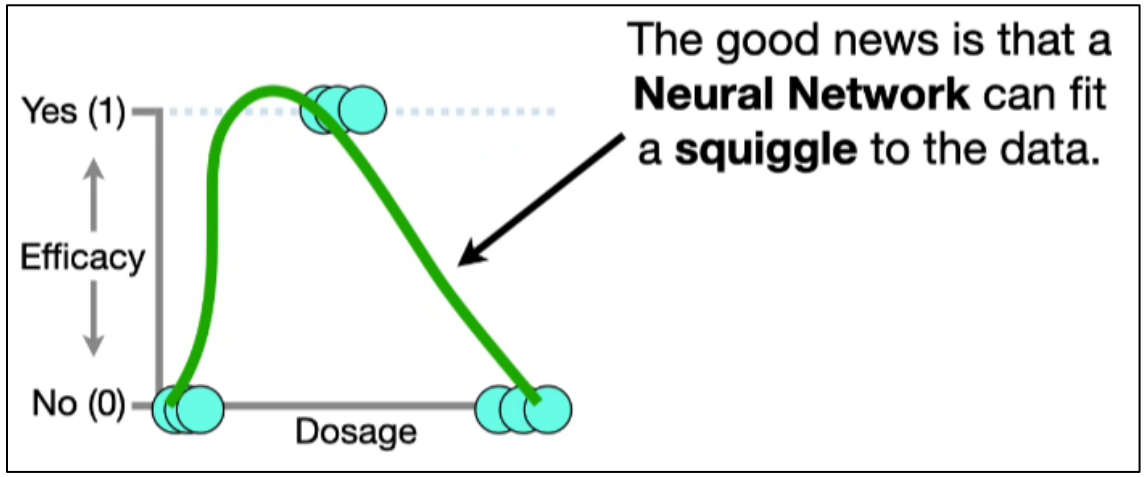
Not a solution



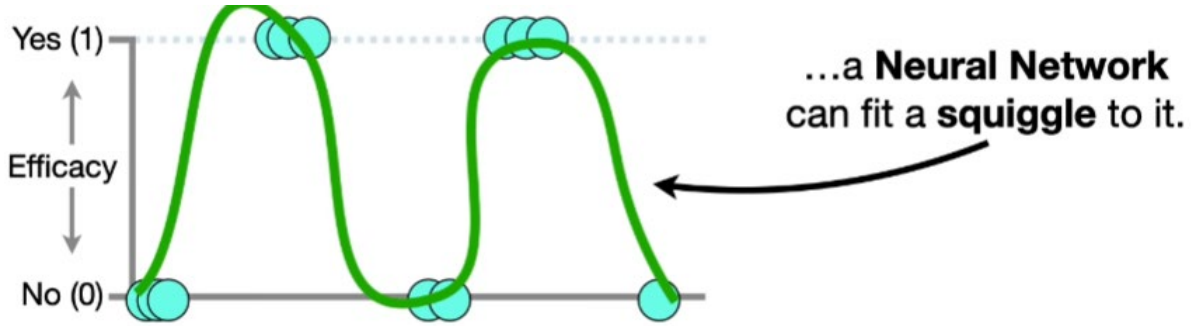
https://www.youtube.com/watch?v=CqOfi41LfDw&list=PLblh5JKOoLUICTaGLRoHQDuF_7q2GfuJF&index=69

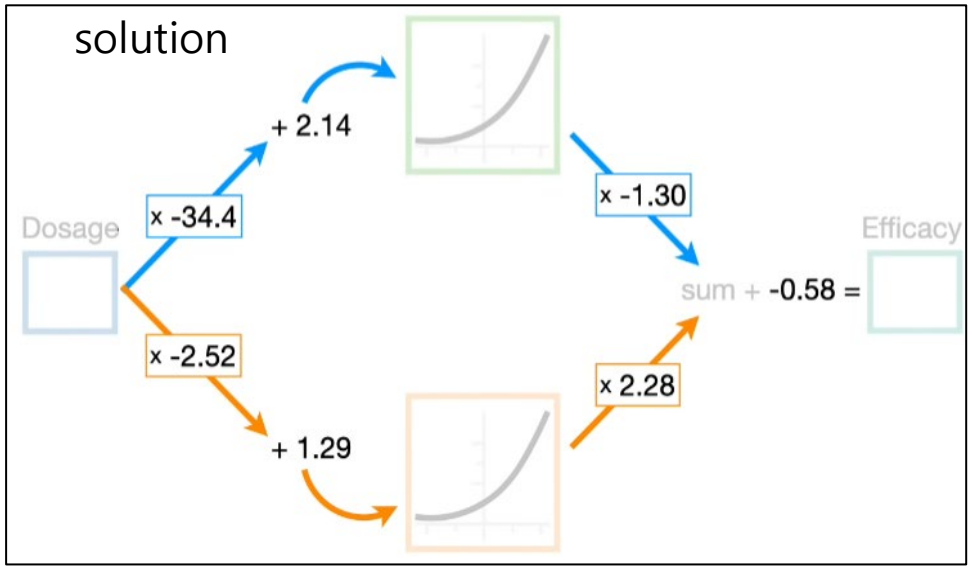
신경망 작동예제

가능한 답



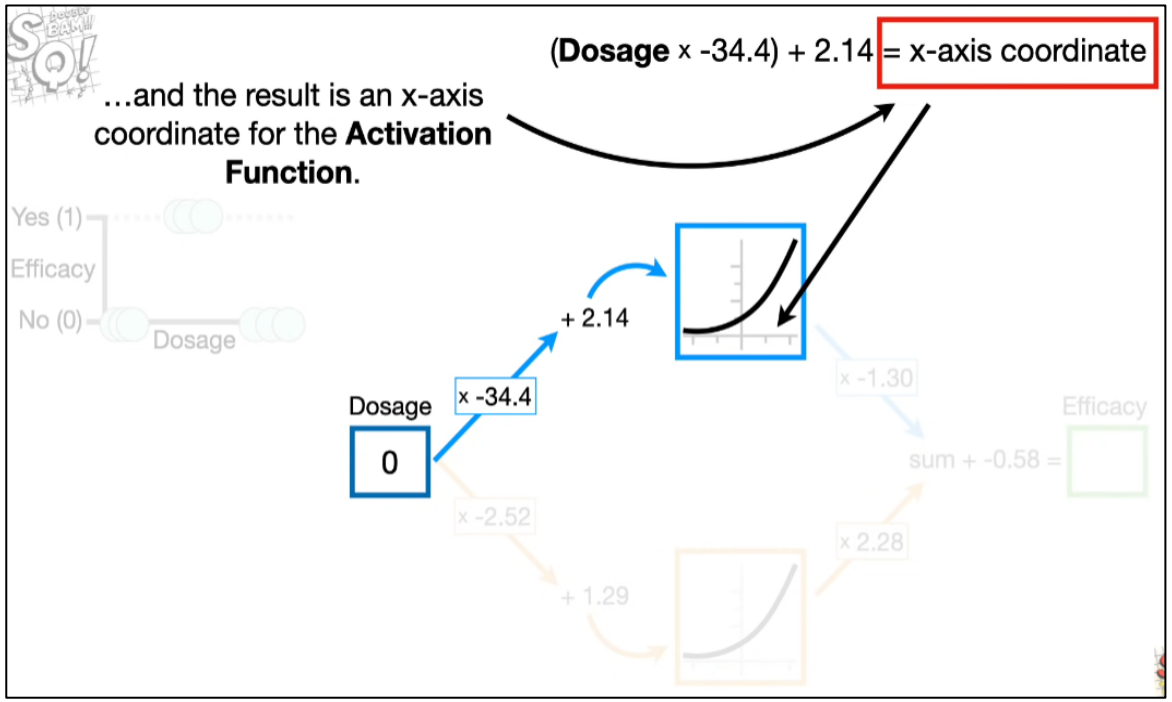
더 복잡한 패턴도 해결가능



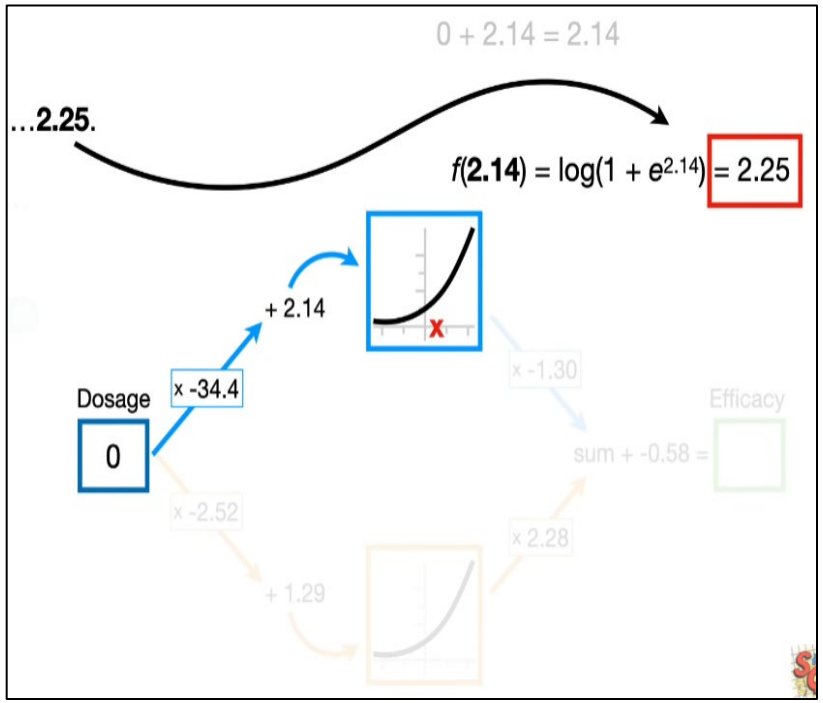


신경망 작동예제

(윗쪽 노드) Dosage 0 값에 대하여 Wx + b 계산

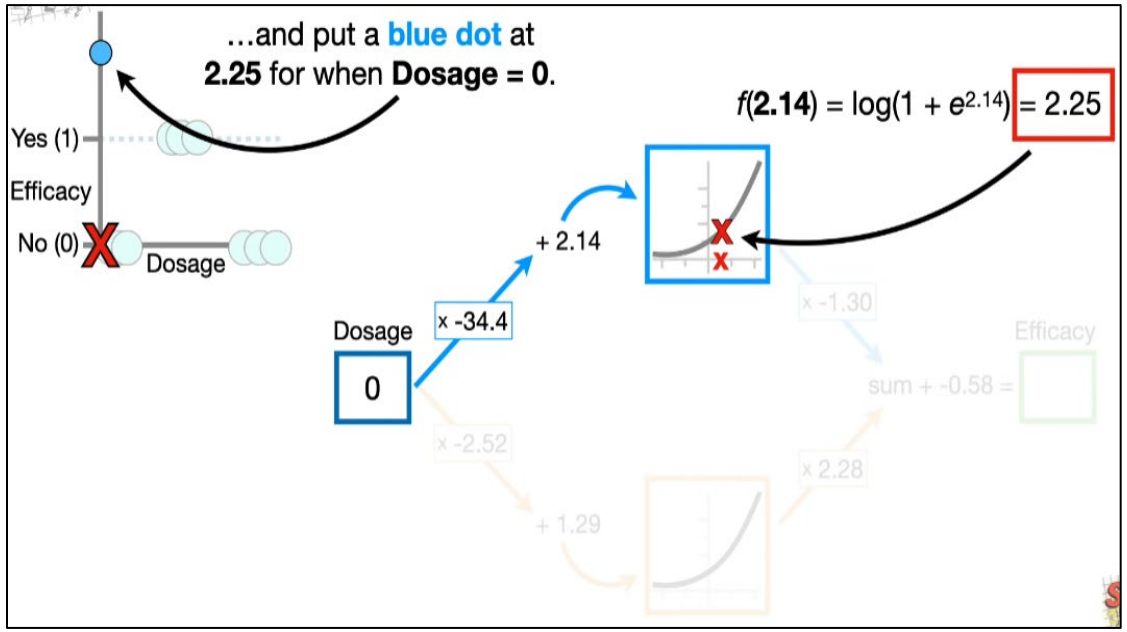


(윗쪽 노드) 결과에 활성함수(softplus) 적용

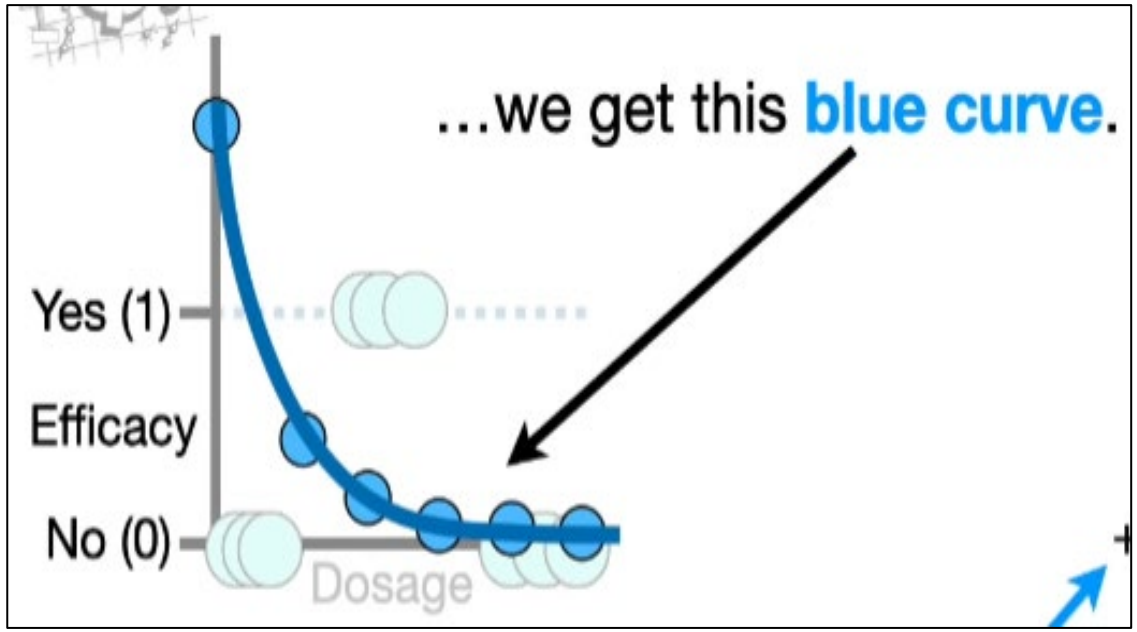


신경망 작동예제

(윗쪽 노드) Dosage = 0일 때 뉴런 계산값은 2.25.

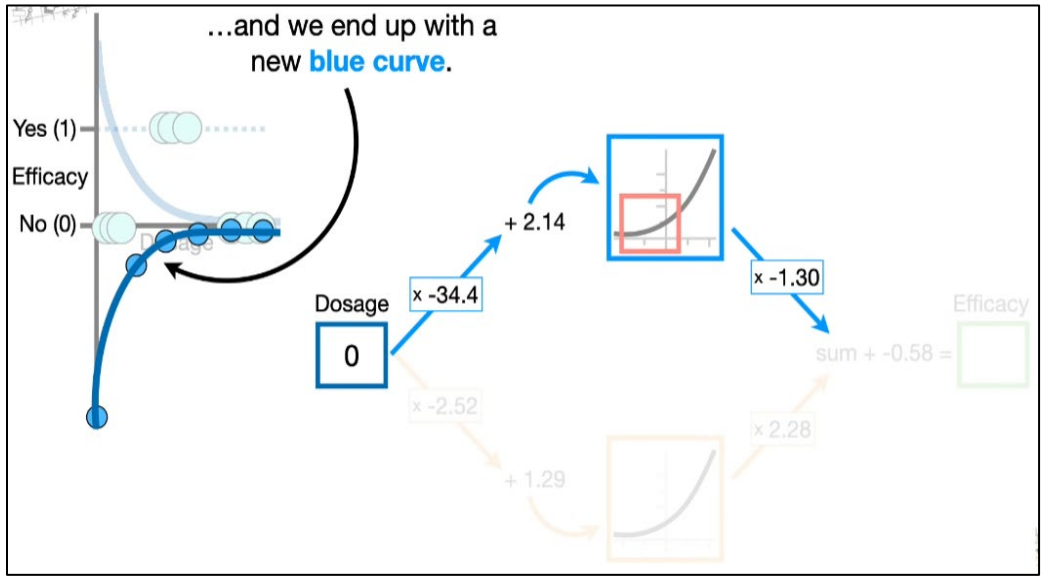


(윗쪽 노드) 모든 샘플에 대해 활성함수까지 적용한 결과



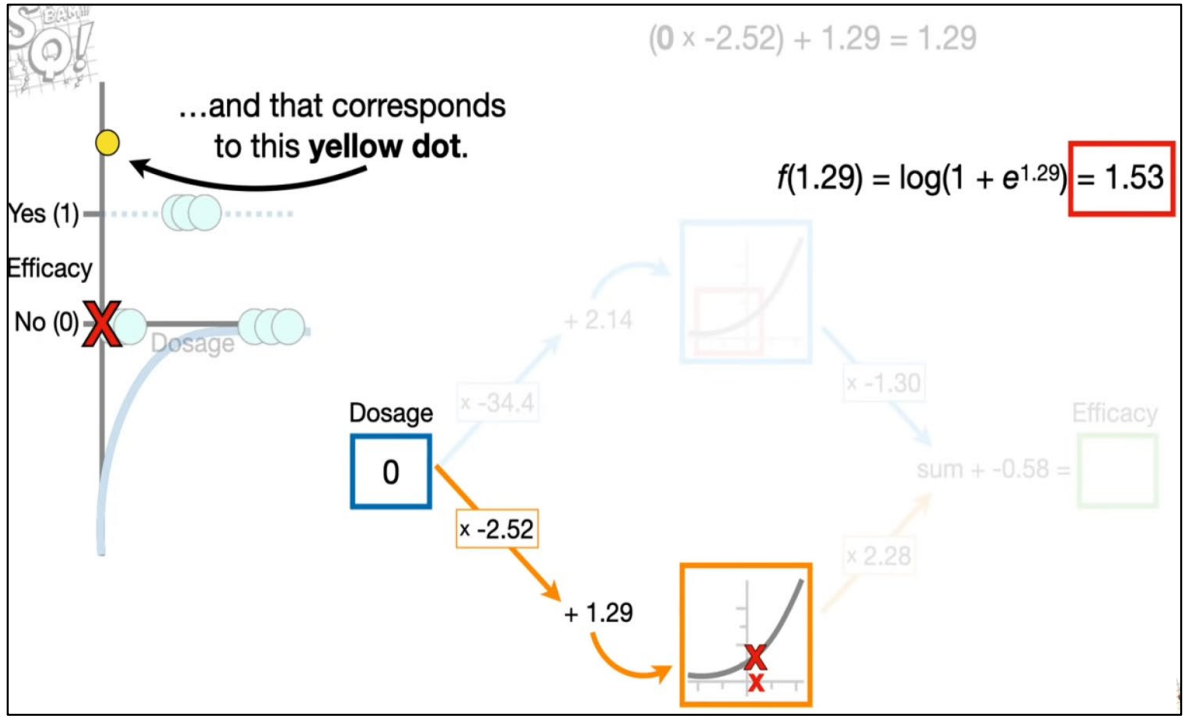
신경망 작동예제

(윗쪽 노드) 마지막 노드 가중치 적용(* -1.30)



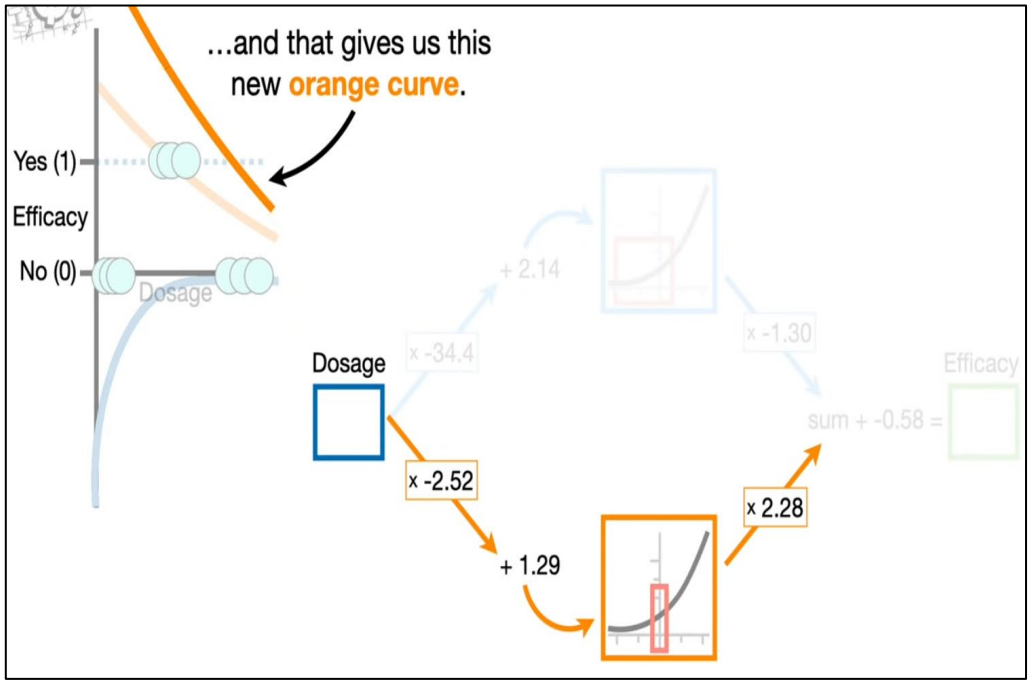
신경망 작동예제

(아래쪽 노드) wx + b → 활성함수 적용

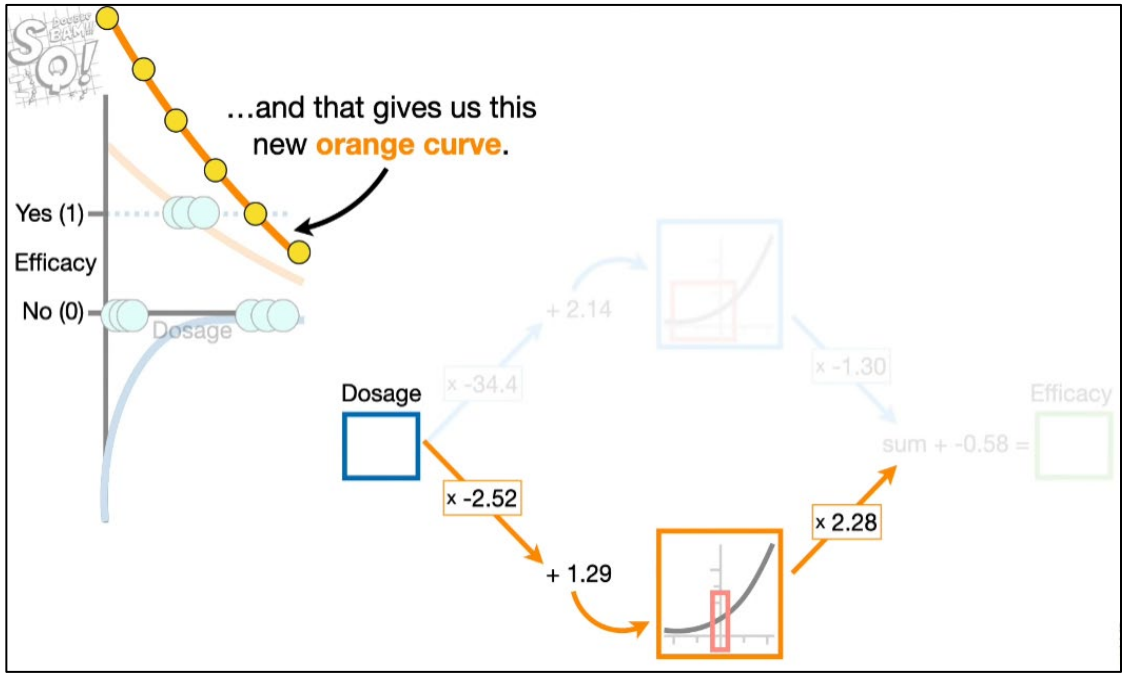


신경망 작동예제

(아래쪽 노드) 모든 샘플에 대해 마지막 노드가중치까지 적용

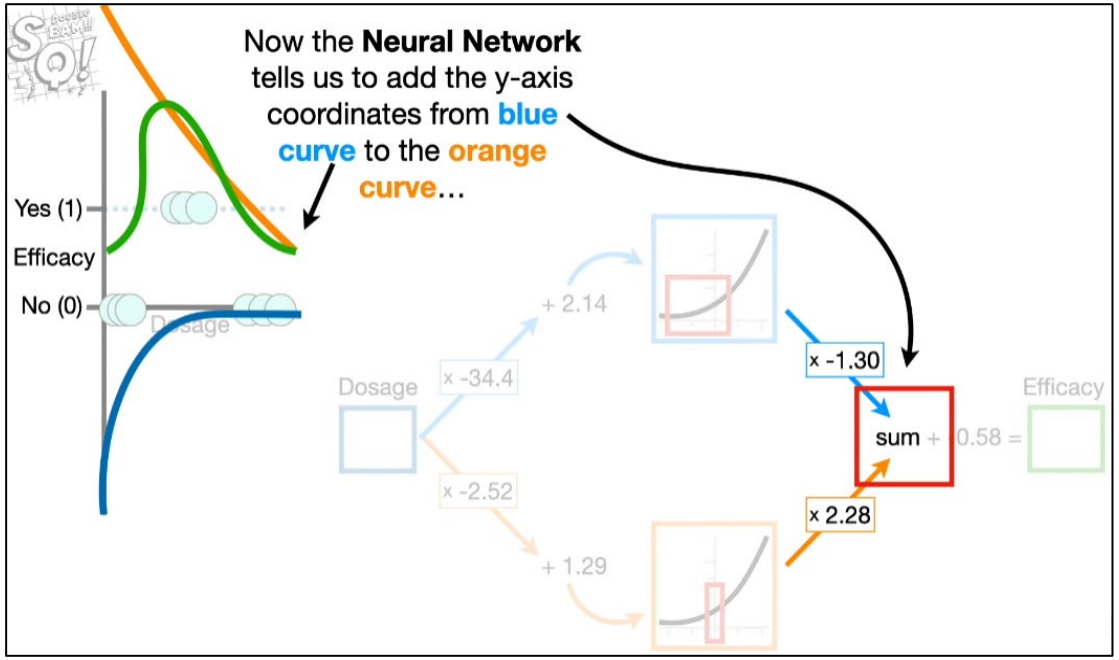


(아래쪽 노드) 모든 샘플 계산

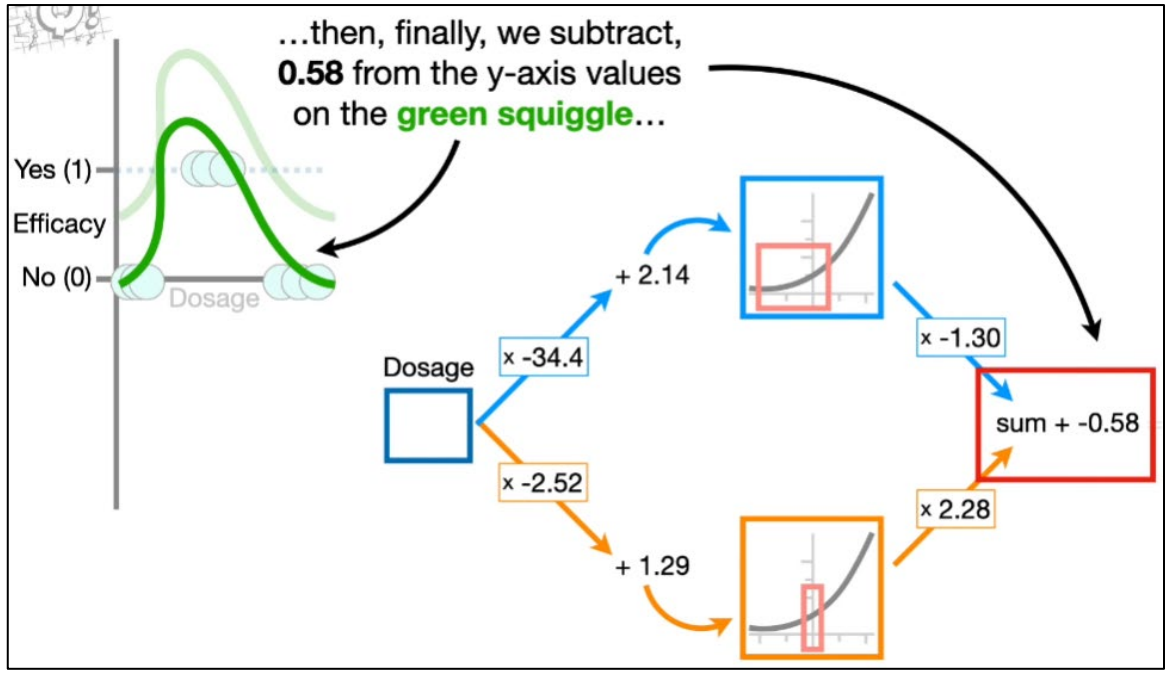


신경망 작동예제

위.아래 그래프 합함(녹색 그래프)

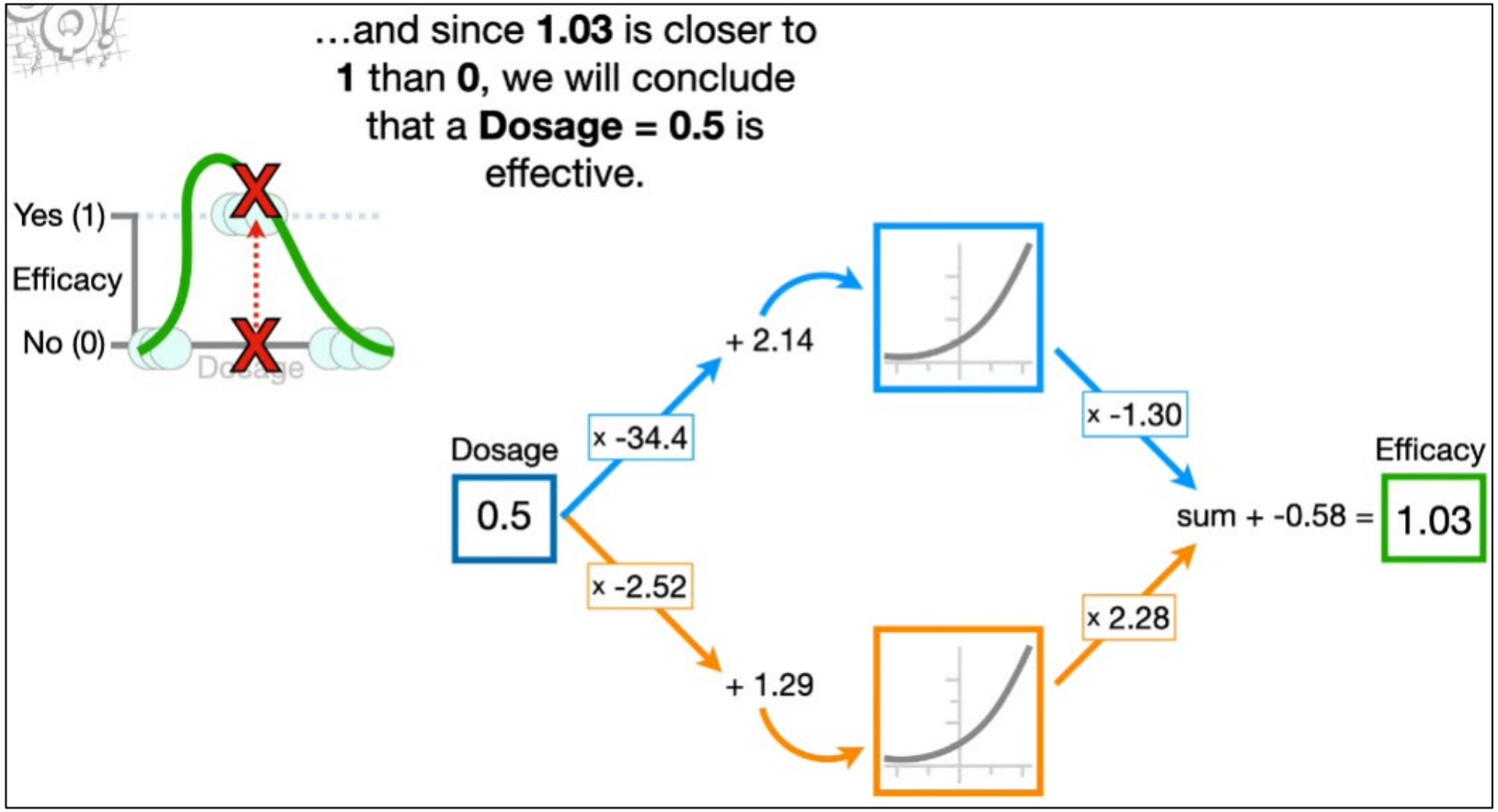


마지막 bias 반영 → 최종 모델 완성

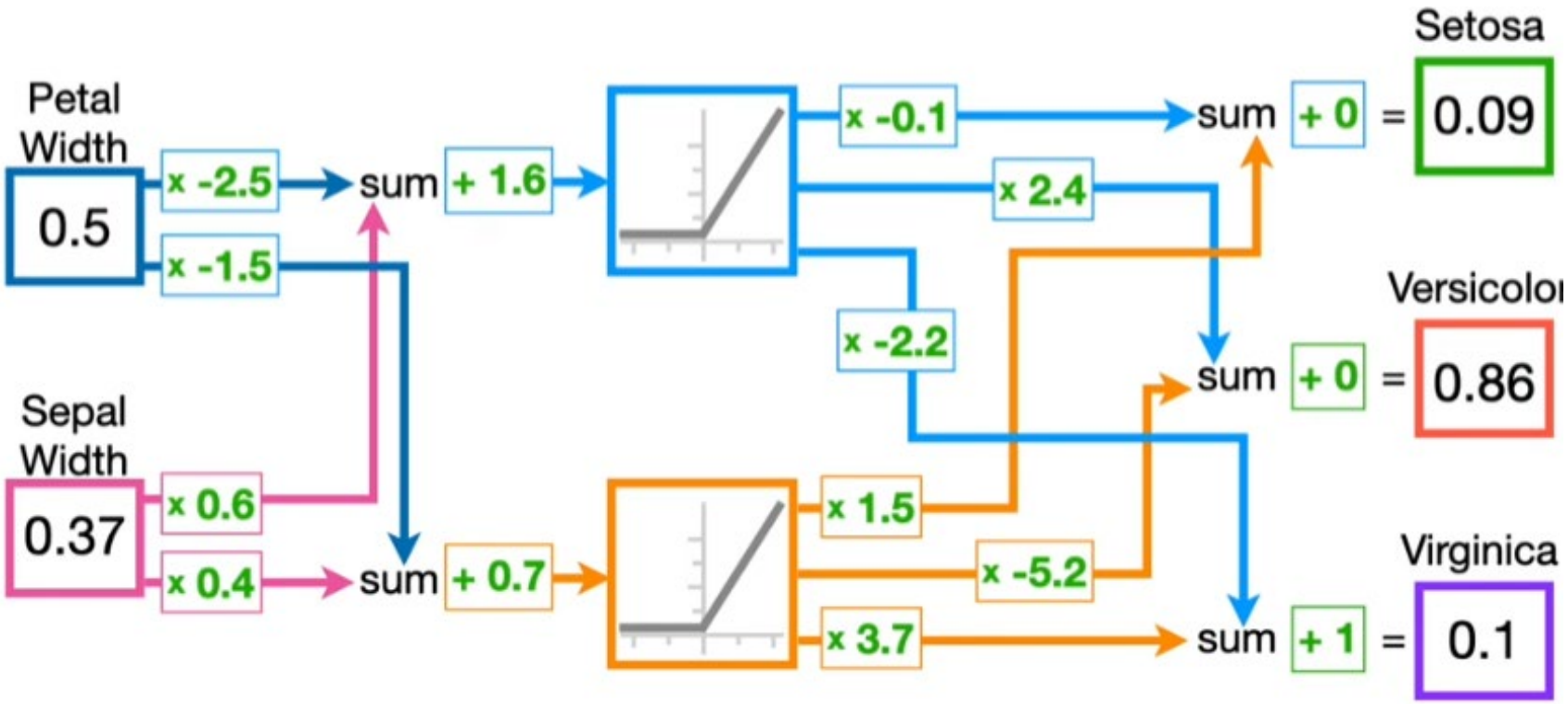


신경망 작동예제

새 샘플 예측
Dosage = 0.5 일때



신경망 작동예제 – 붓꽃 분류



신경망의 구조

- 층(layer)
  - 딥러닝의 핵심적인 구성단위. 기본 연산 unit 으로 구성.
  - 하나이상의 텐서를 입력을 받아 하나 이상의 텐서를 출력하는 데이터 처리모듈.
  - 가중치를 가지며 여기에 네트워크가 학습한 지식이 담겨있음.
  - 밀집연결층(densely connected layer), 순환층(recurrent layer), 합성곱층(convoluitonal layer)등이 있음.

- 손실함수와 옵티마이저
  - 손실함수 : 훈련과정에서 최소화되어야 하는 값. 주어진 문제의 성능지표.
  - 옵티마이저 : 손실함수를 기반으로 네트워크가 어떻게 업데이트될 지 결정

- 모델
  - 입력층과 출력층이 순서대로 쌓인 구조.
  - 텐서연산에 포함된 가중치 텐서의 최적값을 찾아야 함.
  - 최적의 네트워크 구조를 찾는 것은 과학보다는 예술에 가깝다.

- 활성함수
  - 신경망 노드의 계산값의 출력방법을 결정하는 비선형 함수

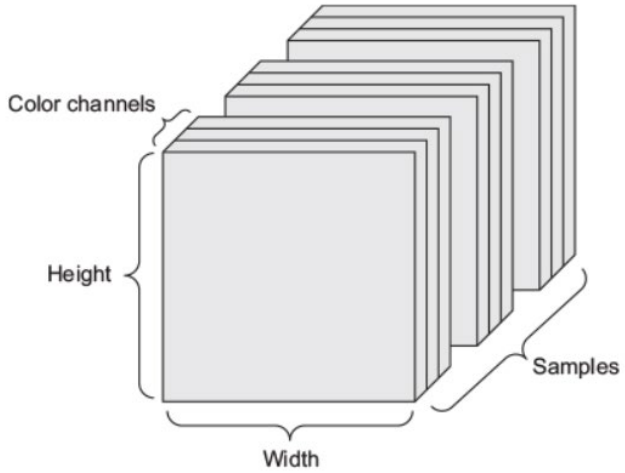
텐서

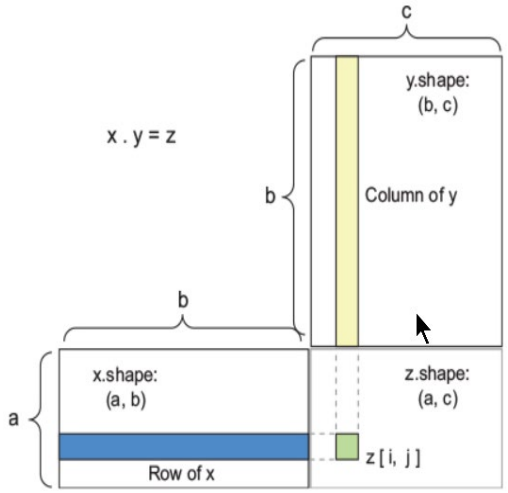
- 텐서 - 신경망을 위한 데이타 표현
  - (숫자)데이터 컨테이너
  - 임의의 차원을 가지는 행렬의 일반화된 모습

- 텐서의 핵심속성
  - 축 개수 : 2D, ~ 5D
  - 크기(shape) : 텐서의 각 축을 따라 몇 개의 차원이 있는지.
  - 데이터 타입(dtype) : float32, float64, uint8

- 텐서의 예
  - 벡터 데이터 : (samples, features) 2D 텐서
  - 시계열 데이터 : (samples, timesteps, features) 3D 텐서
  - 이미지 데이터 : (samples, height, width, channels) 4D 텐서
  - 동영상 데이터 : (samples, frames, height, width, channels) 5D 텐서

- 텐서연산
  - keras.layers.Dense(64, activation='relu', input_shape=(28 * 28,))
  - 2D 텐서를 입력을 받아 64개의 텐서를 출력하는 함수
  - output = relu(dot(W, input) + b)
  - relu(Retified Linear Unit) = max(x, 0)





행렬 점곱(dot) 다이어그램

훈련원리

- 각 층에서 입력데이타가 처리되는 데 필요한 정보는 가중치(weights)에 저장됨.
- 학습 : 주어진 입력을 정확한 타깃에 매핑하기 위해 신경망의 모든 층에 있는 가중치 값을 찾는 것.
- W(weight, 가중치), b(bias, 편향)
  - 훈련되는 파라미터임. – 훈련데이타를 신경망에 노출시켜 학습된 정보 유지.
  - 초기값은 임의의 난수.
  - 훈련이 계속되면서 피드백 신호에 기반하여 W, b 값들이 변화되면서 목적을 달성.

- 가중치 갱신하는 방법
  - 전체 가중치 중 나머지는 모두 고정값으로 하고 한 개만 계산
  - 가중치 값을 높였을 때, 낮추었을 때 손실 계산
  - 손실값이 낮아지는 가중치 값으로 갱신.
  - 모든 가중치에 대해 반복 수행.

- 효율적인 가중치 갱신방법 - 그래디언트 연산
  - 신경망에 사용된 모든 연산은 미분가능
  - 손실의 그래디언트를 계산하는 것이 효율적
  - 손실(loss) = f(W, b)

훈련방법

- 훈련과정
- 1. 훈련샘플 x, 대응 목표값(정답) y 설정
- 1. 초기 가중치 설정
- 2. 설정가중치를 기반으로 예측(Y')
- 3. y_pred를 y(정답)과 비교하여 네트워크 손실 계산
- 3. Optimizer : 손실이 감소되는 방향으로 네트워크의 모든 가중치를 조정
  - ➢ 4. 네트워크의 파라미터에 대한 손실함수의 그래디언트 계산
  - ➢ 5. 그래디언트 반대방향으로 파라미터값 이동 (W -= step * gradient)
    → 손실 감소

- 4. 목표 손실값(정확도)에 도달할 때까지 2-3 반복

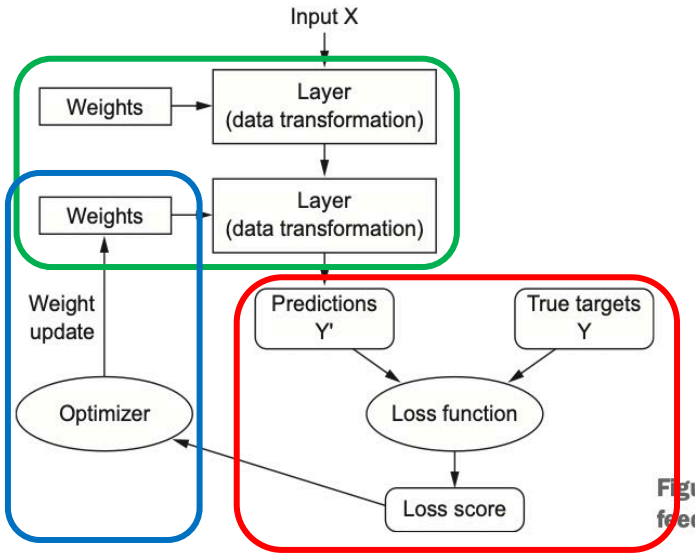


Figure 1.9 The loss score is used as a feedback signal to adjust the weights.

https://velog.io/@gabie0208/1.1-Artificial-intelligence-machine-learning-and-deep-learning

그래디언트 연산 – 손실함수

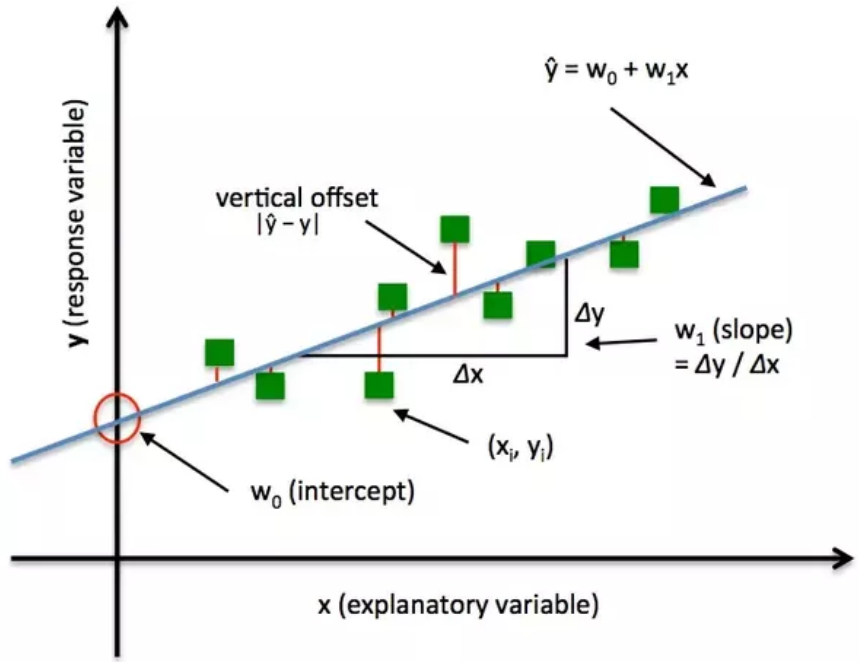
$$H(x) = Wx + b$$

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

Simplified Expr.

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$



그래디언트 연산 – 손실함수 계산

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

x	Y
1	1
2	2
3	3

- W=1, cost(W)=0

$$\frac{1}{3}((1*1-1)^2 + (1*2-2)^2 + (1*3-3)^2)$$

- W=0, cost(W)=4.67

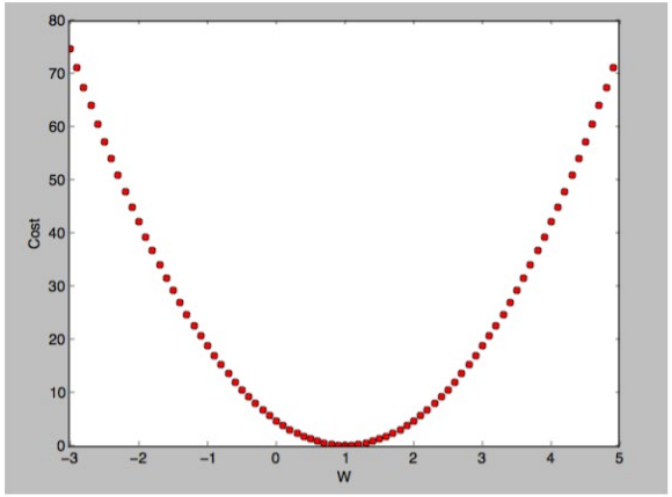
$$\frac{1}{3}((0*1-1)^2 + (0*2-2)^2 + (0*3-3)^2)$$

- W=2, cost(W)=?

http://hunkim.github.io/ml/lec3.pdf

그래디언트 연산 – 손실함수 계산

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$



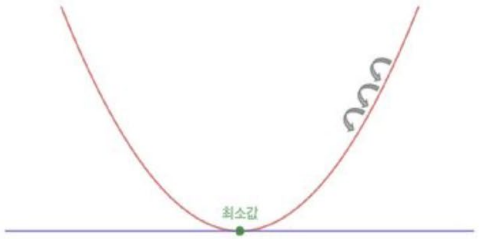
$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$

$$W := W - \alpha\frac{\partial}{\partial W}\frac{1}{2m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

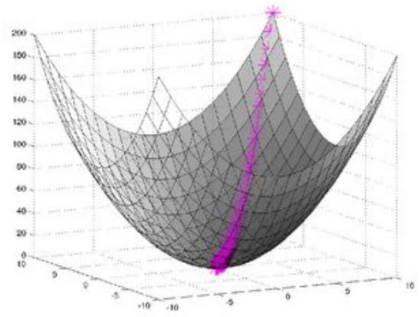
$$W := W - \alpha\frac{1}{2m}\sum_{i=1}^{m}2(Wx^{(i)} - y^{(i)})x^{(i)}$$

$$W := W - \alpha\frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})x^{(i)}$$

Simple Linear Regression



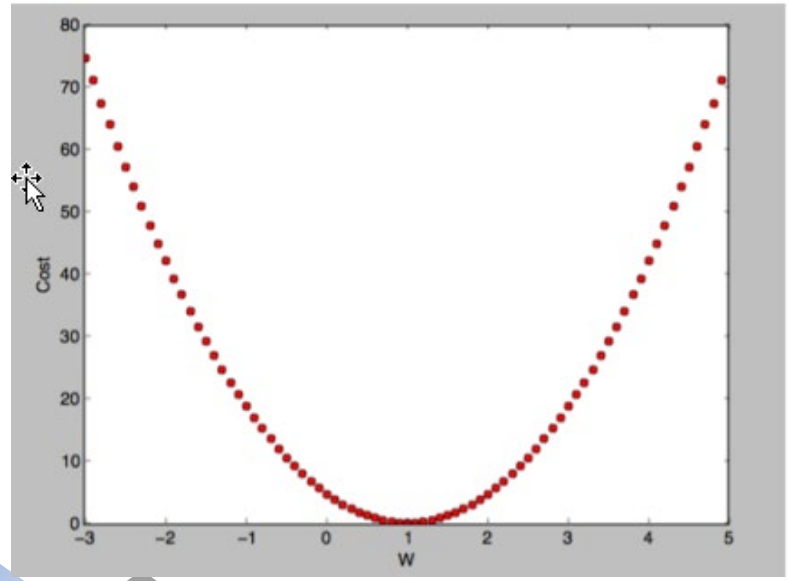
Multiple Linear Regression



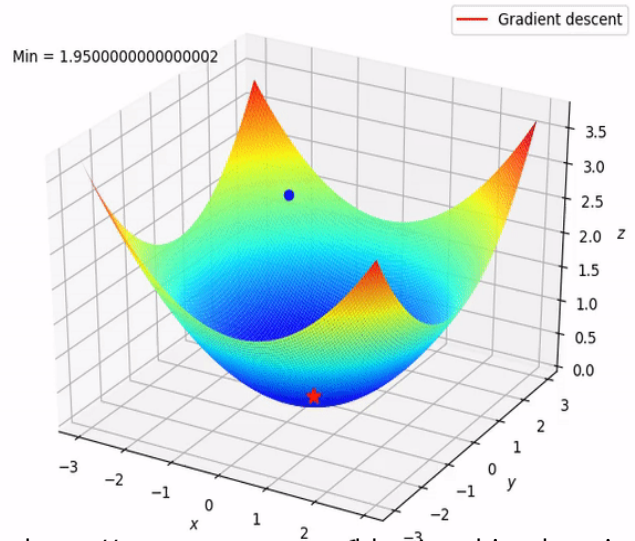
손실함수(cost function, loss function) – 회귀 손실함수

- 옵티마이저
  - 손실함수를 기반으로 네트워크가 어떻게 업데이트 될 지 결정하는 알고리즘.
  - 현재는 RMSProp, Adam 이 최상의 성능 발휘함.
- 손실함수
  - 훈련과정에서 최소화되어야할 계산값. 미분가능.
  - 회귀 손실함수 : MSE(Mean Squared Error)
  - MSE의 미분값으로 학습 가중치에 대하여 경사하강 적용

회귀 손실함수



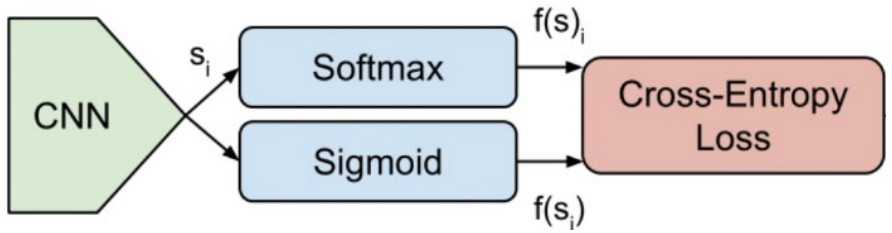
옵티마이저의 작동



https://www.xpertup.com/blog/machine-learning/loss-functions-and-optimization-algorithms/

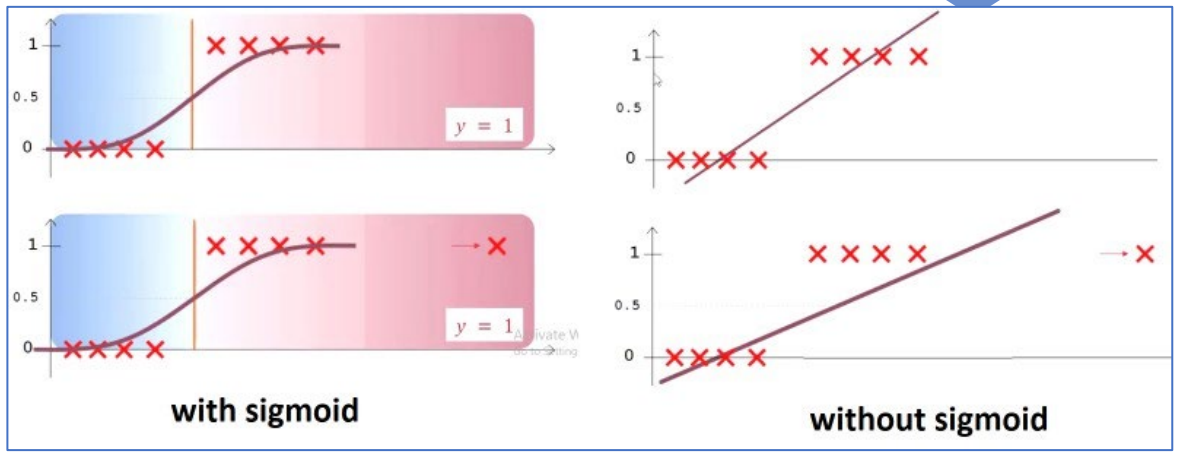
손실함수 – 이진분류

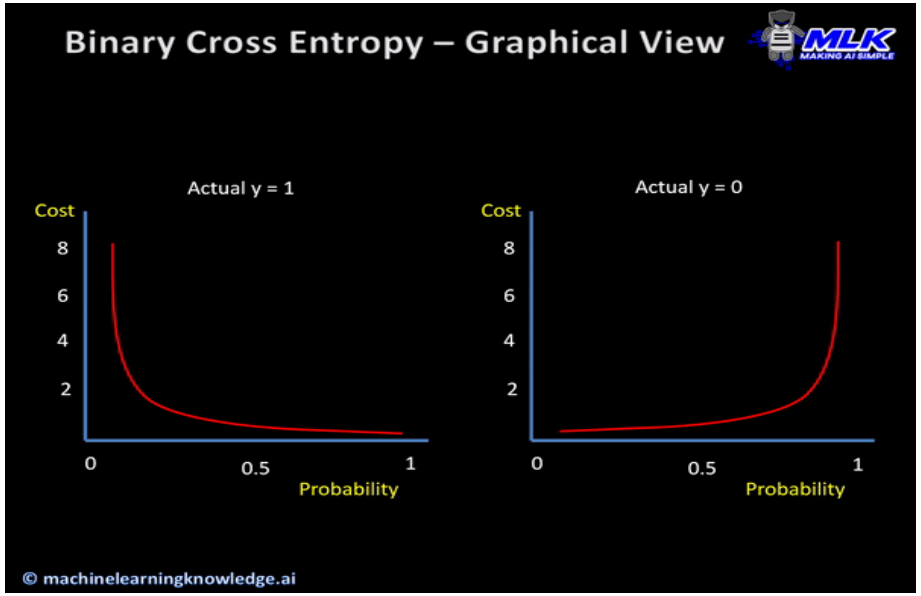
- 이진분류, 다중분류의 계산흐름.
- 이진분류 : 계산값을 Sigmoid에 주입하고 그 결과를 binary cross entropy로 계산
- 다중분류 : 계산값을 Softmax에 주입하고 그 결과를 categorical cross entropy로 계산



- 이진분류 손실함수
  - ➢ binary crossentropy

$$\text{Loss} = -\frac{1}{\text{output size}} \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log (1 - \hat{y}_i)$$

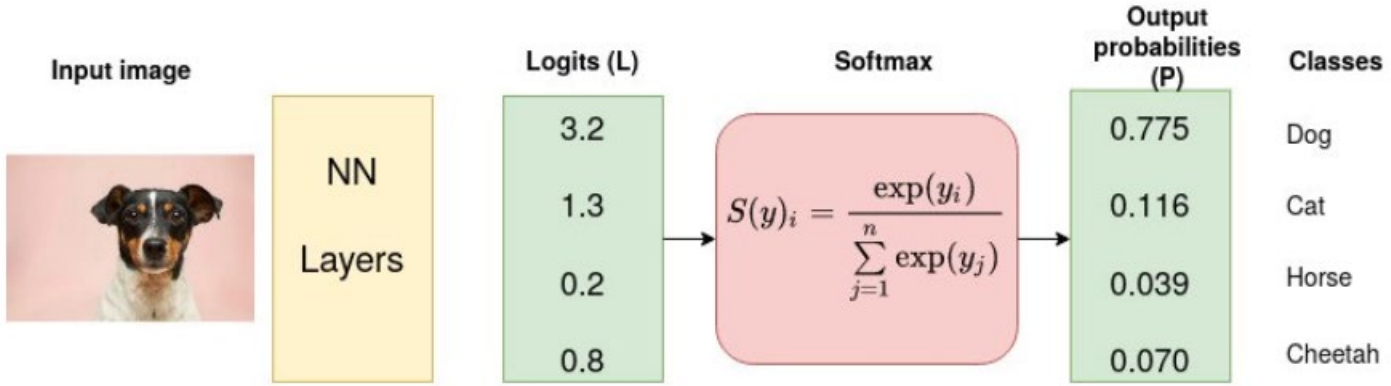




https://machinelearningknowledge.ai/types-of-keras-loss-functions-explained-for-beginners/

손실함수 –다중분류

- 다중 분류 계산 흐름.



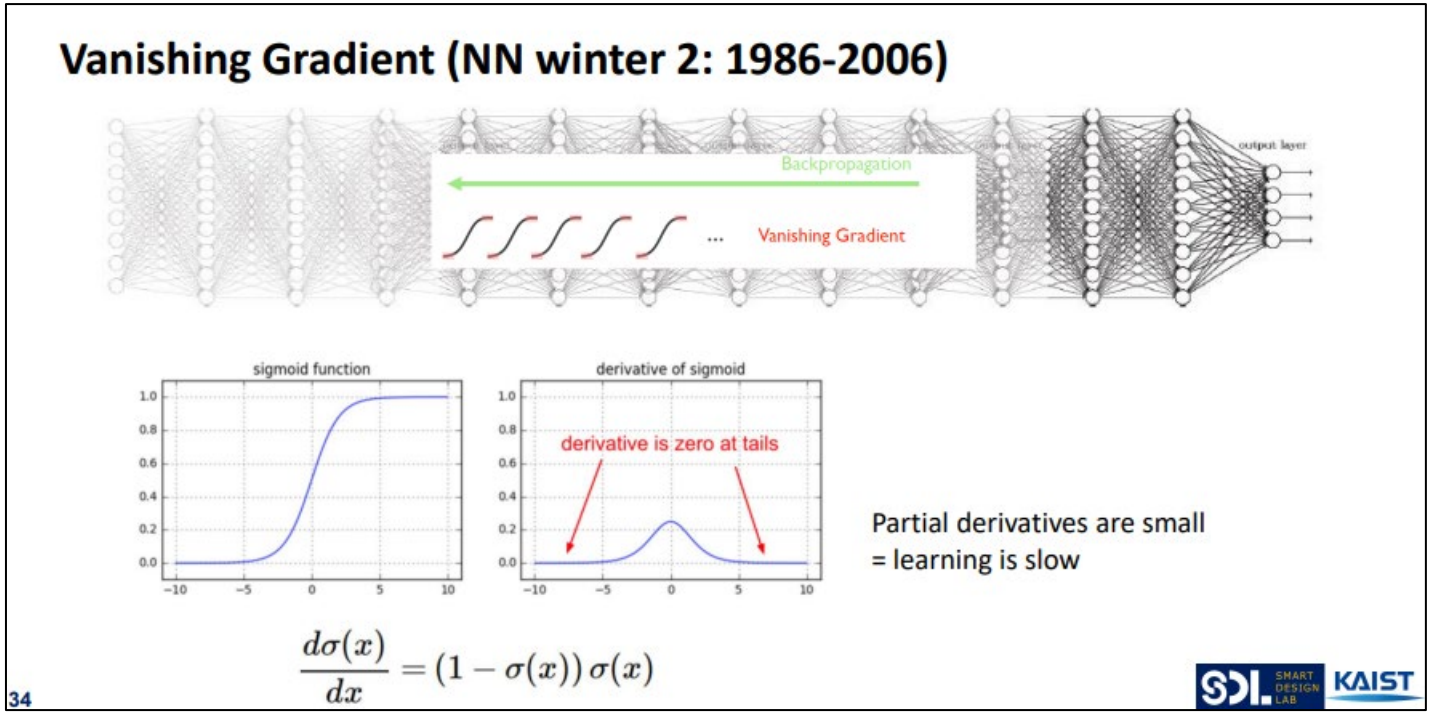
https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e

- 다중 분류 손실함수 :
  - categorical crossentropy
  - categorical crossentropy 의 미분값으로 학습 가중치에 대하여 경사하강 적용

$$\text{Loss} = -\sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

활성함수

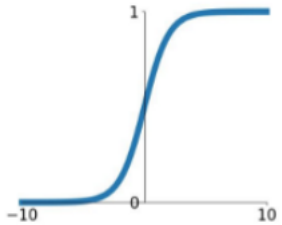
- 활성함수
  - 신경망 노드의 계산값(WX + b)의 출력방법을 결정하는 비선형 함수
  - 만약 생략되면 연결된 두층(매트릭스)은 또다른 선형층이 되며 학습을 하지 못함.
  - 더 복잡한 패턴을 학습하기 위한 도구
- relu
  - 계산이 쉽다. 역전파시에도 값이 소멸되지 않는다.
- Vanishing Gradient
  - - 초기에는 sigmoid를 활성함수로 사용.
  - 0 ~ 1 사이의 값.
  - 역전파시 층을 거치면서 미분값이 계속 곱해지므로 값이 소멸되는 문제.



Activation Functions

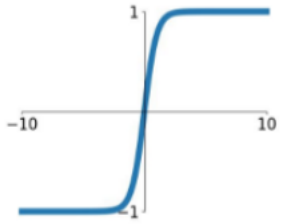
Sigmoid

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



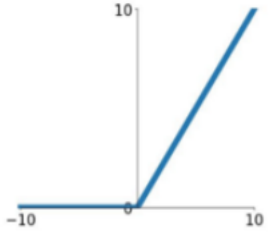
tanh

$$\tanh(x)$$



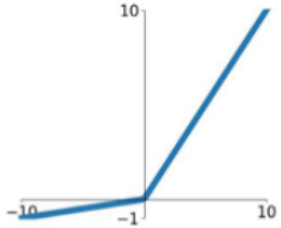
ReLU

$$\max(0, x)$$



Leaky ReLU

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

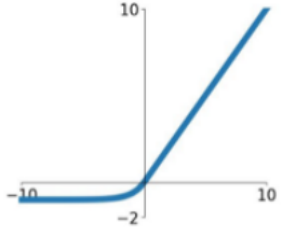


Maxout

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

ELU

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



http://cs231n.stanford.edu/slides/2021/lecture_4.pdf

learning rate

- 학습률 : 가중치를 조절하는 비율
- 너무 작으면 학습에 오랜 시간 소요.
- 너무 크면 최적값 지나치거나 발산 위험

[Train Log]
Iter: 0, Loss: 6.0257, Learning Rate: 0.1000
Iter: 1000, Loss: 0.3723, Learning Rate: 0.0960
Iter: 2000, Loss: 0.2779, Learning Rate: 0.0922
Iter: 3000, Loss: 0.2293, Learning Rate: 0.0885
Iter: 4000, Loss: 0.1977, Learning Rate: 0.0849
Iter: 5000, Loss: 0.1750, Learning Rate: 0.0815

High Learning Rate is Overshooting
Normal Learning Rate is 0.01
3e-4 is the best learning rate for Adam,
hands down (andrej karpathy)

[Tensorflow Code]

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.01)
```

데모

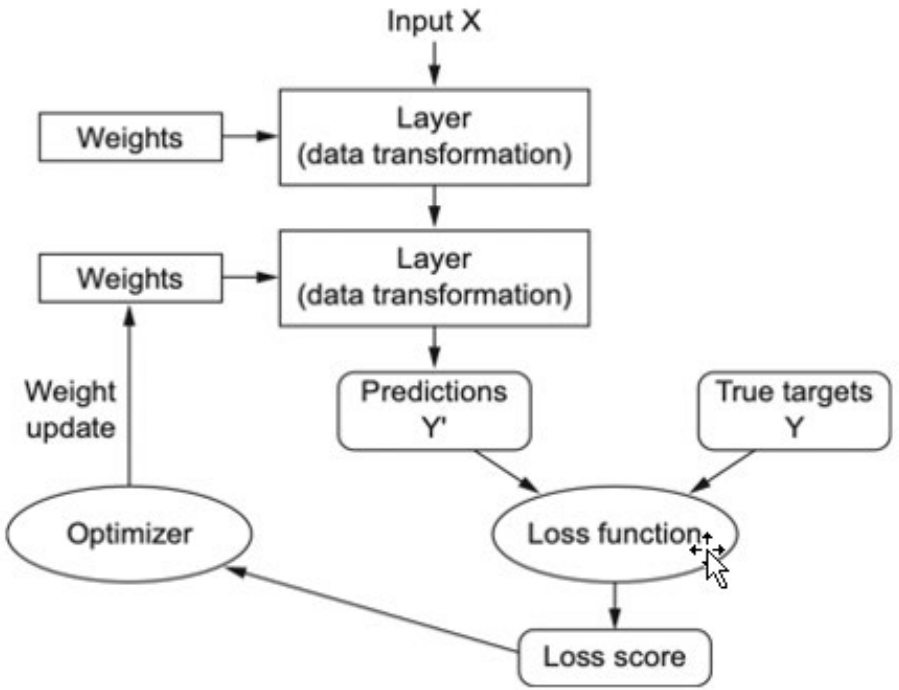
- < 문제 >

  - x = [[1], [2], [3]]
  - y = [1, 2, 3]

  - 기울기를 찾아가는 과정 추적
  - 학습률에 따라 수렴 또는 발산

신경망의 구조

- 층(layer)
  - - 딥러닝의 핵심적인 구성단위. 기본 연산 unit 으로 구성.
  - - 하나이상의 텐서를 입력을 받아 하나 이상의 텐서를 출력하는 데이터 처리모듈.
  - - 가중치를 가지며 여기에 네트워크가 학습한 지식이 담겨있음.
  - - 밀집연결층(densely connected layer) : 일반 2D 데이터를 처리하는 층유형.
  - 순환층(recurrent layer) : 시퀀스 데이터를 처리하는 층유형
  - 합성곱층(convoluitonal layer) : 이미지 데이터를 처리하는 층유형.
- 모델
  - - 입력층과 출력층이 순서대로 쌓인 구조.
  - 가능성 있는 공간(가설공간)의 정의 – 허용연산이 제한되는 범위.
  - - 텐서연산에 포함된 가중치 텐서의 최적값을 찾아야 함.
  - - 최적의 네트워크 구조를 찾는 것은 과학보다는 예술에 가깝다.
- 옵티마이저 : 손실함수를 기반으로 네트워크가 어떻게 업데이트될 지 결정
- 손실함수 : 훈련과정에서 최소화되어야 하는 값. 주어진 문제의 성능지표.
  - 회귀 : MSE
  - 이진분류 : binary cross entropy
  - 다중분류 : caategorical cross entropy
  - 시퀀스 학습 : CTC(connection temporal Classification)

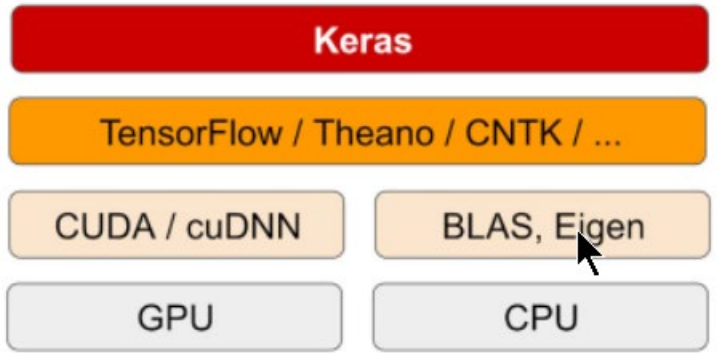


keras 소개

- keras
  - 대부분의 딥러닝 모델을 만들고 훈련시킬 수 있는 프레임워크
  - 단일코드로 CPU, GPU 실행가능
  - 비전용 CNN, 시퀀스용 RNN 지원.
  - 다중입력, 다중출력모델, 층의 공유, 모델공유 등 어떤 네크웤 구조도 구현 가능.
  - 상업적인 프로젝트에 자유롭게 사용 가능(MIT License)

- keras, tensorflow, Theano, CNTK
  - keras는 딥러닝 모델을 만들기 위한 고수준의 구성요소 제공.
  - 텐서조작, 미분 등 저수준 연산은 backend engine 사용(platform)
  - backend : tensorflow, Theano, CNTK

Figure 3.3. The deep-learning software and hardware stack



keras 를 사용한 개발

- keras를 사용한 작업 흐름
  - 입력텐서와 목표값 텐서 훈련데이타 정의
  - 입력과 목표값을 매핑하는 모델(네트워크) 정의
  - 손실함수, 옵티마이저, 측정지표 설정
  - 학습 조건 설정
  - 학습(fit)

- 중요 학습조건
  - batch_size : 가중치 갱신을 위한 단위 샘플 수
  - epoch : 전체 샘플 반복 횟수
  - validation_data : 학습 후 검증까지 수행시 사용 데이타

- 모델을 정의하는 방법
  - Sequential 클래스 방법 : 설계하고자 하는 층을 순서대로 쌓는 구조
  - 함수형 API 방법 : 임의의 구조를 만들 수 있는 비순환유향그래프 구현가능

이진분류

- Dense 층
  - model.add(layers.Dense(64, activation='relu'))
  - 표현 공간의 차원. 신경망이 내재된 표현을 학습할 때 가질 수 있는 자유도
  - 가중치 행렬(W)의 크기 : 784(input_shape) * 64(은닉 유닛)
  - 은닉 유닛을 늘리면 (표현 공간을 더 고차원으로) 더욱 복잡한 표현을 학습할 수 있지만 계산 비용이 커지고
  - 학습데이타에만 최적화될 수 있음 (테스트 데이터에서는 성능저하).

- 활성함수
  - 중간에 있는 은닉층은 학습표현을 풍부하게 하는 비선형 활성함수 가짐
  - 중간층은 `relu`를 사용 – 음수를 0으로 만드는 함수
  - 주어진 W, input에 대하여 relu(dot(W, input) + b) 값을 출력함.
  - 마지막 층은 시그모이드 활성함수를 사용 – 임의의 값을 [0, 1] 사이로 압축하므로 출력 값을 확률처럼 해석가능.

- 손실함수
  - 이진 분류 문제에서 스칼라 시그모이드 출력에 대해 사용하는 손실 함수는 `binary_crossentropy`

- 옵티마이저
  - `rmsprop`, Adam 옵티마이저는 문제에 상관없이 일반적으로 충분히 좋은 선택

이진분류

- 실습예제 :
  - 패션-MNIST 샘플 (Zalando, MIT License)
  - 10가지 종류 중 'T-shirt/top', 'Trouser' 만 추려서 학습

```
# 데이타 모양을 2차원 이미지에서 1차원 으로 변경. 0과 1 사이로 스케일을 조정
train_images = images.reshape((12000, 28 * 28))
train_images = train_images.astype('float32') / 255

# X_train 데이타를 학습, 검증 세트로 나눔.
x_val = x_train[:5000], partial_x_train = x_train[5000:]

# 모델 정의
• model.add(layers.Dense(64, activation='relu', input_shape=(28 * 28, )))
• model.add(layers.Dense(1, activation='sigmoid'))

• # 손실함수, 옵티마이저 정의
• model.compile(optimizer='rmsprop',  loss='binary_crossentropy', metrics=['accuracy'])
• # 학습 실행
• history = model.fit(partial_x_train, partial_y_train, epochs=20, batch_size=32, validation_data=(x_val, y_val))
• # 테스트셋으로 검증
• score1 = model.evaluate(x_test, y_test)
```

다중분류

- 다중분류 신경망 모델
  - - 패션-MNIST 샘플 (Zalando, MIT License)
  - - 10가지 종류 모두 적용
  - 입력 데이터가 벡터이고 레이블은 여러 개 중 하나
  - 기대출력 : 여러 개 중 하나이므로 정수형 또는 one-hot encoding으로 변환해야 함.

- 유닛
  - `Dense` 층에 전달한 매개변수는 은닉 유닛의 개수. – 더 많은 데이터와 더 많은 기대출력이 있으므로 네트웍 용량도 확대 필요.

- 활성함수
  - 중간에 있는 은닉층은 활성함수로 `relu`를 사용
  - 마지막 층은 softmax 활성함수를 사용 – 레이블마다 기대값을 가지고 전체합이 1이 됨. 출력 값을 확률처럼 해석가능.

- 손실함수
  - softmax 출력에 대응하는 손실함수 : `categorical_crossentropy

- 옵티마이저
  - `rmsprop`, Adam 옵티마이저

회귀(Regression)

- 연속적인 값을 예측
- 예제 : 보스턴 교외 주택가격 예측

- 데이터 특징
  - - 전체 506개로 적은 수의 데이타셋
  - - 각 특성들의 값범위가 다름. → 역전파 알고리즘에서 큰 값의 영향을 많이 받기 때문에 학습에 어려움 초래
  - - 모든 특성값을 일정 범위로 한정하는 정규화 필요

- 유닛
  - `Dense` 층에 전달한 매개변수는 은닉 유닛의 개수.
  - 적은 양의 데이터이므로 적은 용량의 모델 구성

- 활성함수
  - 중간에 있는 은닉층은 활성함수로 `relu`를 사용
  - 마지막 층은 출력값이 예상값이므로 활성함수 없음
  - # 활성화 함수를 적용하면 출력 값의 범위를 제한하게 됨.

- 손실함수
  - 회귀문제에서는 MSE(Mean Squared Error) 사용 : 예측과 타깃 사이의 거리의 제곱
  - 데이터 양이 적으므로 k-겹 검증 사용