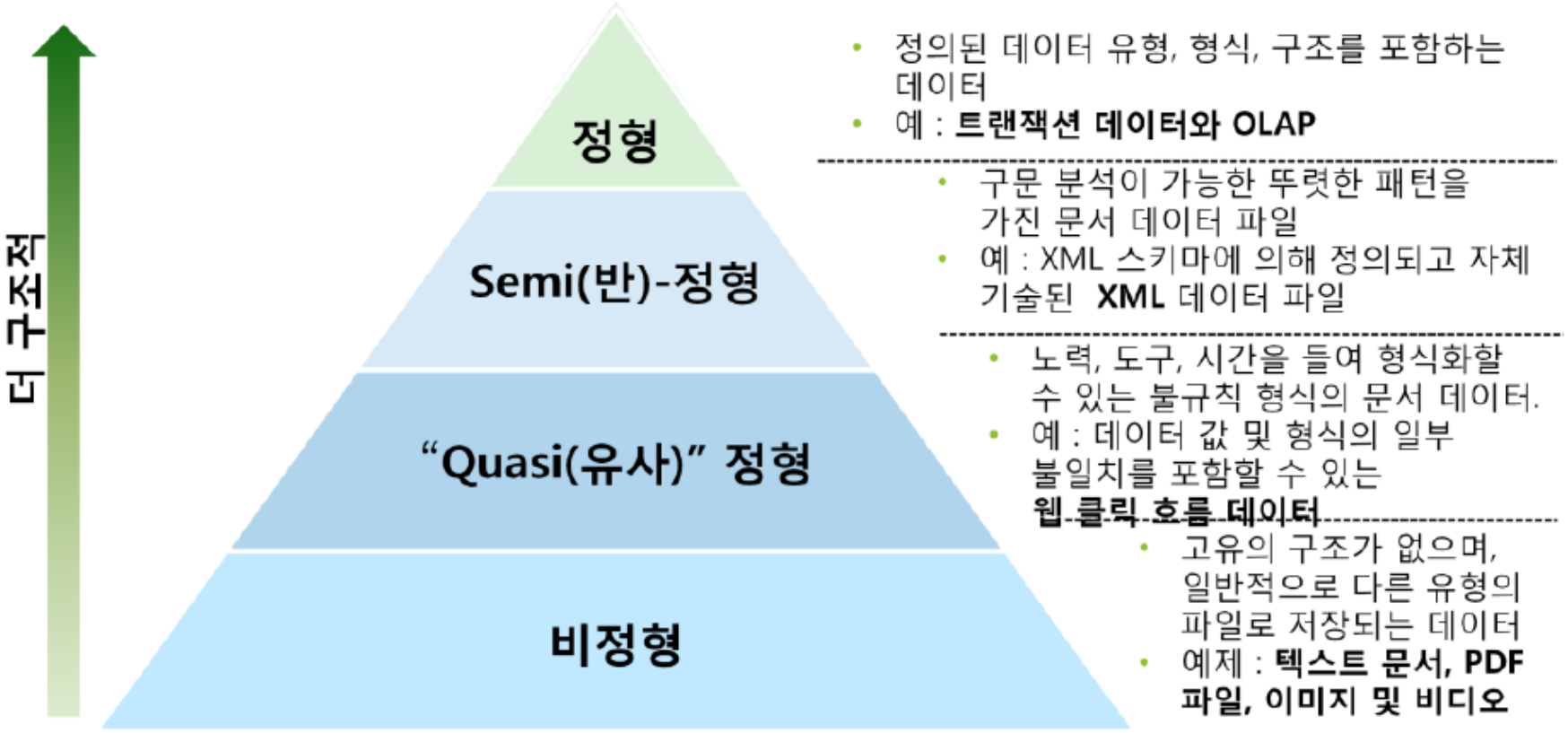
1. 딥러닝 이해
2. tensorflow

데이터



정형 – 고정된 필드에 저장된 데이터 예) 관계형 데이터베이스, 스프레드 시트
반정형 – 고정된 필드에 저장되어 있지는 않지만, 메타데이터나 스키마등을 포함하는 데이터
비정형 - 고정된 필드에 저장되어 있지 않은 데이터 예) 텍스트 분석이 가능한 텍스트 문서, 이미지, 동영상, 음성 데이터

Data 구조의 네가지 주요 유형

정형 데이터

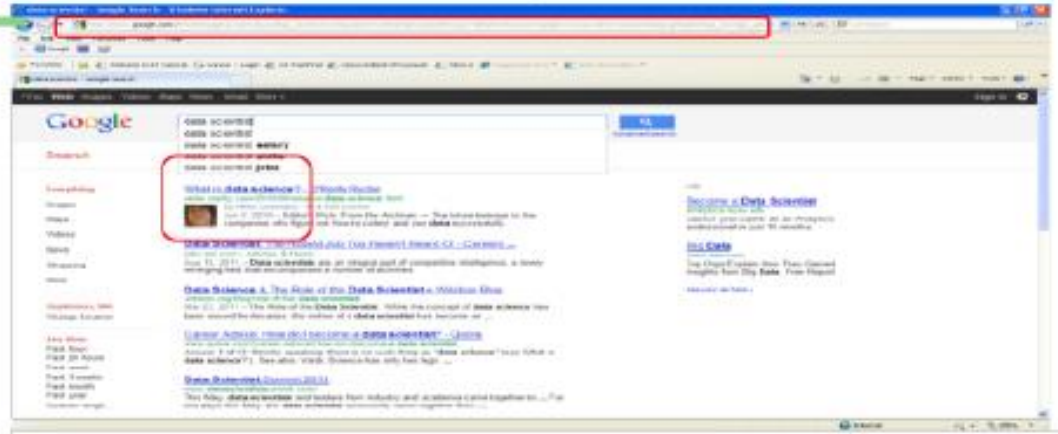
SUMMER FOOD SERVICE PROGRAM 1]
(Data as of August 01, 2011)

Fiscal Year	Number of Sites	Peak (July) Participation	Meals Served	Total Federal Expenditures 2]
	----------Thousands----------		--Mil.--	--Million $--
1969	1.2	99	2.2	0.3
1970	1.9	227	8.2	1.8
1971	3.2	559	29.0	8.2
1972	5.5	1,030	73.5	21.9
1973	11.2	1,437	65.4	26.6
1974	10.6	1,403	63.6	33.6
1975	12.0	1,785	84.3	50.3
1976	16.0	2,453	104.8	73.4
TQ 3]	22.4	3,455	198.0	88.9
1977	23.7	2,791	170.4	114.4
1978	22.4	2,333	120.3	100.3
1979	23.0	2,126	121.8	108.6
1980	21.6	1,922	108.2	110.1

반 정형 데이터

Xml파일

유사 정형 데이터



http://www.google.com/#hl=en&sugexp=kjrmc&cp=8&gs_id=2m&xhr=t&q=data+scientist&pq=big+data&pf=p&sclient=psyb&source=hp&pbx=1&oq=data+sci&aq=0&aqi=g4&aql=f&gs_sm=&gs_upl=&bav=on.2,or.r_gc.r_pw.,cf.osb&fp=d566e0fbd09c8604&biw=1382&bih=651

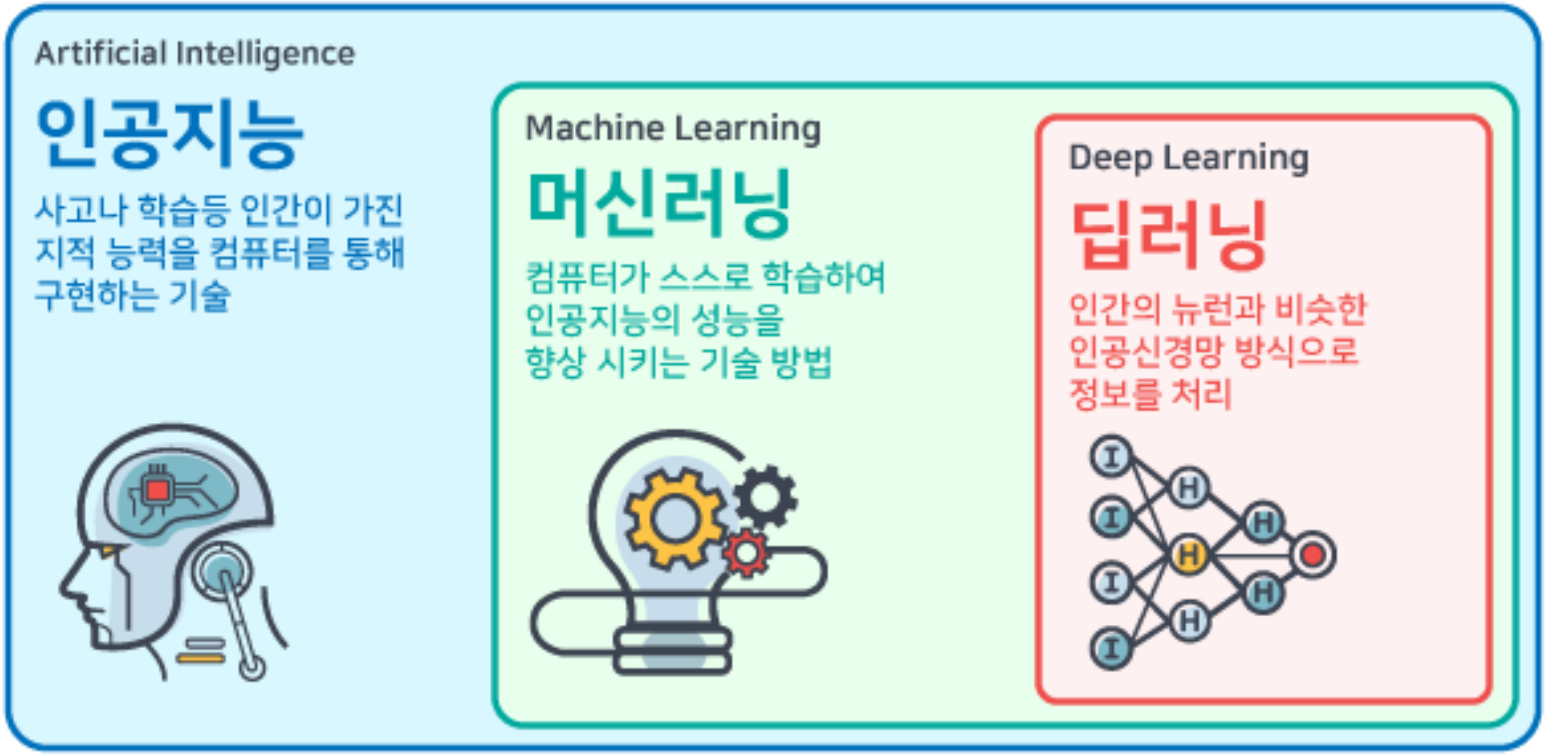
비 정형 데이터

The Red Wheelbarrow, by William Carlos Williams

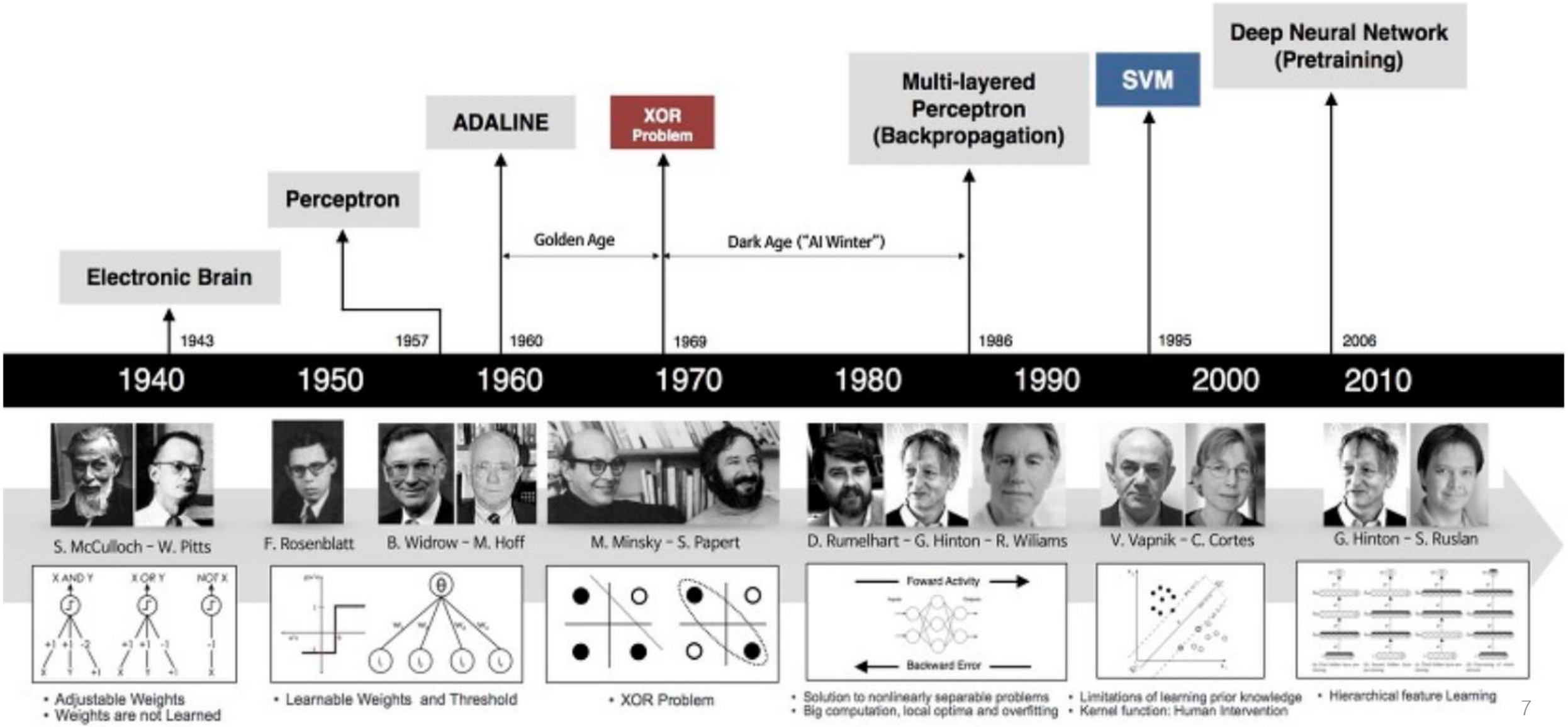
so much depends
upon
a red wheel
barrow
glazed with rain
water
beside the white
chickens.

인공 지능 Artificial Intelligence

- 인간의 학습능력, 추론능력, 지각능력, 자연언어 등의 이해능력을 컴퓨터 프로그래밍으로 구현한 기술
- 기계(컴퓨터)가 인간처럼 생각·판단하는 기술
- 인공지능 ⊃ 머신러닝 ⊃ 인공신경망 ⊃ 딥러닝



인공 지능 역사


Deep Neural Network (Pretraining)
SVM
Multi-layered Perceptron (Backpropagation)
XOR Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Williams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

인공 지능 역사

인공지능의 탄생 **1943년 ~ 1956년**	• 전기 스위치처럼 켜고 끄는 기초 기능의 인공 신경을 그물망 형태로 연결하면 사람 뇌에서 동작하는 아주 간단한 기능을 흉내 낼 수 있음을 증명 (워렌 맥클록 & 월터 피츠) • 기계가 인간과 얼마나 비슷하게 대화할 수 있는지를 기준으로 기계 지능을 판별하는 튜링 테스트(Turing Test)를 제안 (엘론 튜링) • 존 매카시 교수(다트머스 대학)가 개최한 다트머스 회의에서 10명의 과학자가 모여 인공지능 개념을 정의하고, 인공지능 용어를 처음으로 사용
태동기 **1956년 ~ 1974년**	•인간의 뇌신경을 묘사한 인공신경 뉴런 '퍼셉트론(Perceptron)'을 제시 - 프랑크 로젠블럿 • 최초의 전문가 시스템(Expert System)인 '댄드랄(DENDRAL) ' 개발 - 전문가가 하던 일들을 시스템화한 것 •민스키와 페퍼트가 '퍼셉트론즈'라는 책을 출판 - 퍼셉트론의 결정적 문제점이 노출되어 신경망 연구 침체기가 시작되었고, 연결주의가 추락

퍼셉트론의 결정적 문제점 : A라는 개념과 B라는 개념을 정확히 분리해낼 수 있는 선형 분리 방식을 퍼셉트론이 적용하지 못하는 분야를 찾아냄 (XOR)
전문가 시스템 : 전문가의 지식을 논리적인 규칙으로 생성하여 특정 영역에 대해서 사람의 질문에 답할 수 있는 인공지능
기호주의(Symbolism) : 마빈 민스키(Marvin Lee Minsky) ,인간의 지능과 지식을 기호화해 매뉴얼화하는 접근법 , 디지털 컴퓨터의 작동방식을 기반으로 AI를 연구하는 방식

인공 지능 역사

첫 번째 암흑기 **1974년 ~ 1980년**	• 전문가 시스템으로 연구 방향을 전환 (기호 주의의 부상)
발전기 **1980년 ~ 1987년** 신경망의 부활	• 단층 퍼셉트론 모델이 다층 퍼셉트론(신경망이 레이어드된 형태)으로 컴백 • 다층 퍼셉트론에 쓰이는 역전파 알고리즘(오차를 줄일 수 있음)을 제안 • 신경망은 패턴인식을 통해 문자, 인식, 영상 등의 인식에 크게 기여했다. • 기울기 소실 문제, 신경망을 적용할 수 있는 범위가 한정됨 (데이터의 집합이 크고 복잡한 패턴을 처리하기 위해서는 히든 레이어를 여러 개 연결해야 하는데 오류역전파 방법으로는 제대로 학습이 되지 않음)
두 번째 암흑기 **1987년 ~ 1993년**	• 다층 신경망의 제한적 성능과 느린 컴퓨터의 속도로 복잡한 계산이 필요한 신경망 연구가 정체되었다
안정기 **1993년 ~ 2011년** 인공지능 중흥기	• 방대한 데이터를 수집,저장, 빅데이터를 분석하여 인공지능 시스템 스스로 학습하는 머신러닝 형태로 진화 •많은 층을 적층하고 결과물을 향상 시키는 딥러닝 기반의 학습 알고리즘 제안(RBM) - 제프리 •비지도 학습방법이 가능 •딥러닝 알고리즘은 주로 음성 인식, 영상 이해, 기계 번역 등에 쓰이고 있다.

인공 지능 역사

부흥기 **2011년 ~ 현재**	• 구글이 심층신경망(DNN)을 구현하여 고양이 영상인식을 성공시켰다. (1만 6천 개의 컴퓨터로 10억 개 이상의 신경망을 구성) • 페이스북에서 '딥페이스'라는 얼굴인식 알고리즘을 개발 (97%의 성능으로 사람 얼굴을 구분) • 구글 딥마인드의 알파고(AlphaGo)가 이세돌 9단과의 바둑대결에서 승리 (알파고는 지도학습과 비지도 학습을 동시에 사용) • 알파고는 비지도 학습의 한 종류인 강화 학습 기술로 혼자 대국을 두며 학습할 수 있었다

인공지능 부흥 이유 :
데이터 : 2025년까지 163 제타바이트(1 테라의 10억 배)의 데이터가 생성될 것이라고 IDC에서 발표
알고리즘 : 딥러닝 등 최신 알고리즘이 개발되고 계속해서 성능이 발전하고 있다.
컴퓨팅 파워 : 2018년 출시된 GPU(병렬처리가 가능)는 5년 전 출시된 가장 빠른 버전보다 40~80배 빠르다.

머신러닝

- 기계에게 어떻게 동작할지 일일이 코드로 명시하지 않고 데이터를 이용해 학습할 수 있도록 하는 알고리즘과 기술을 개발하는 인공지능의 하위 분야
- 알고리즘을 이용해 데이터를 분석하고, 분석을 통해 학습하며, 학습한 내용을 기반으로 판단이나 예측을 수행
- 대규모 데이터 세트에서 패턴과 상관관계를 찾고 분석을 토대로 최적의 의사결정과 예측을 수행하도록 훈련

  예] 문자인식, 음성인식, 바둑 또는 장기 등의 게임 전략, 의료 진단, 로봇개발

머신러닝 학습모델

- 머신러닝 학습 모델 : 지도, 비지도, 준지도, 강화
- 머신러닝 알고리즘은 사물 분류, 패턴 발견, 결과 예측, 정보 기반 의사결정 등을 수행하도록 설계됨.
- 알고리즘은 하나씩 사용할 수도 있고 복잡하고 보다 예측 불가능한 데이터가 포함된 경우에는 정확도를 극대화하기 위해 다양한 알고리즘 기법을 적용 결합할 수도 있다.

머신러닝 학습모델

➢ 지도 학습 (Supervised Learning)

- 정답이 있는 데이터를 활용해 데이터를 학습시키는 것
- 입력 값(X data)이 주어지면 입력값에 대한 Label(Y data)를 주어 학습시킴.
- 머신에 정답 키를 제공해 모든 올바른 결과 중에서 상관관계를 찾아 학습하도록 합니다

예측과 분류

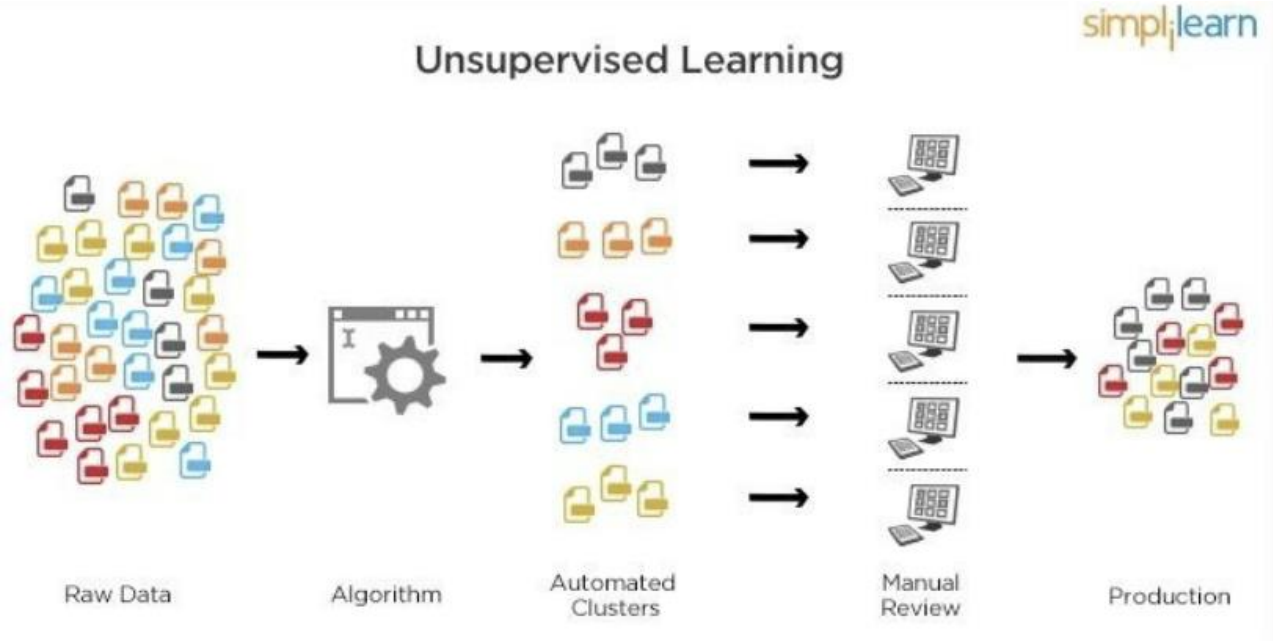
개, 고양이 학습 데이터 이미지 인식 영상 처리

머신러닝 학습모델

➢ 비지도 학습(Unsupervised Learning)

- 입력데이터만 이용해서 컴퓨터를 학습시키는 방법
- 레이블이 없는 비정형 데이터를 학습한 다음 관련성이 있고 액세스 가능한 데이터를 모두 사용해 패턴과 상관관계를 인식 합니다 .
- 입력된 데이터를 비슷한 그룹으로 묶어 예측하는 모델을 학습합니다.
- 비지도 학습은 예측이 목적이 아니라, 데이터의 구성 또는 특징을 밝히는 목적으로 사용되는 그룹핑 알고리즘입니다.

안면 인식,
유전자 서열 분석,
시장 조사,
사이버 보안



머신러닝 학습모델

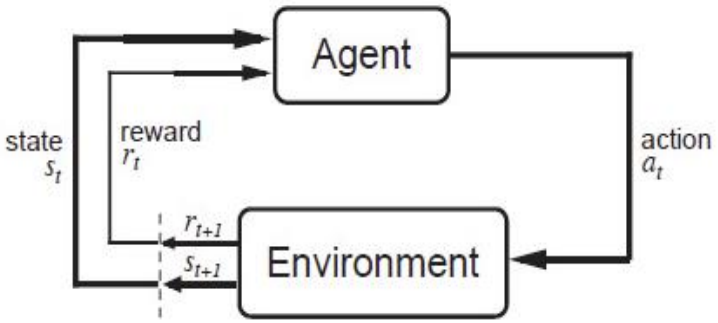
➢ 지도학습, 비지도학습의 대표적인 알고리즘

지도학습 (Supervised Learning)	Classification	kNN Naive Bayes Support Vector Logistic Regression Decision Tree Random Forest Neural Network
	Regression	Linear Regression Locally Weighted Linear Ridge Lasso
비지도학습 (Unsupervised Learning)	Clustering K Means Density Estimation Exception Maximization Pazen Window DBSCAN	

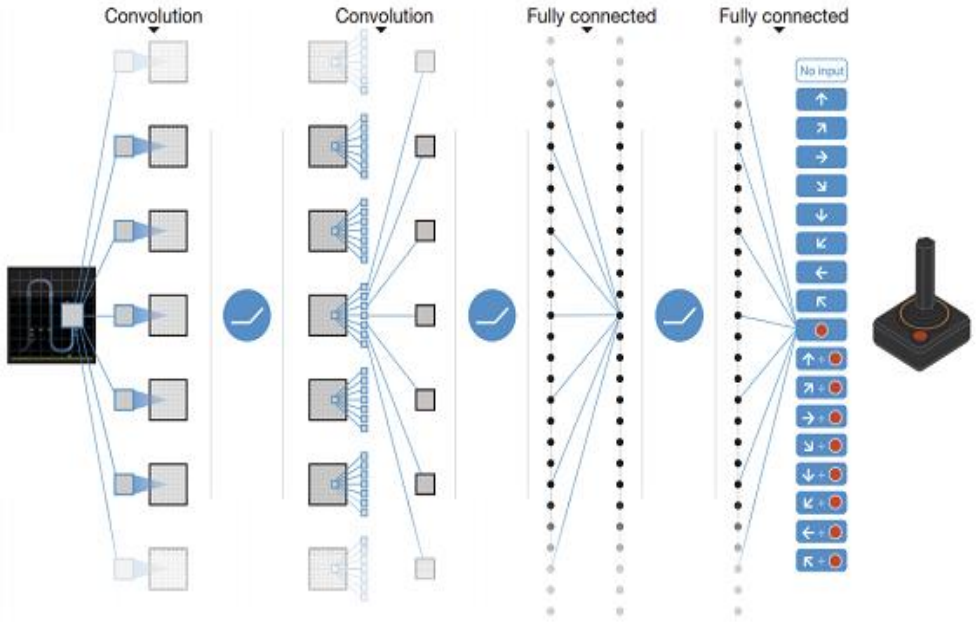
머신러닝 학습모델

➢ 강화 학습 (Reinforcement Learning)

- 컴퓨터가 주어진 상태(state)에 대해 최적의 행동(action)을 선택하는 기계 학습(ML) 방법
- 행동에 대한 보상을 받으며 학습하여 어떤 환경 안에서 선택 가능한 행동들 중 보상을 최대화하는 행동 또는 행동 순서를 선택하는 방법
- 행동의 좋고 나쁜 정도를 학습 알고리즘에게 알려 주는 것 - 보상 또는 강화(reinforcement)
- 강화학습에 딥러닝을 성공적으로 적용한 대표적 알고리즘 : DQN과 A3C (딥마인드에서 발표, 다른 강화학습 알고리즘의 베이스라인이 되었습니다.)



AlphaGo

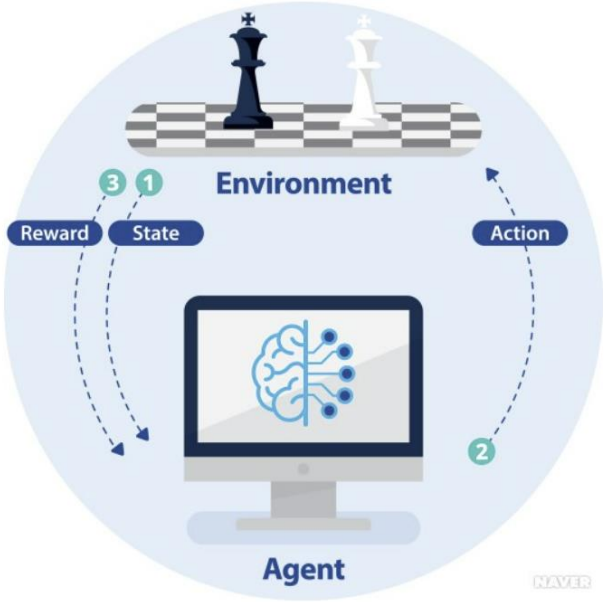


온라인 광고 구매자의 자동 가격 입찰
컴퓨터 게임 개발
고위험 주식 시장 거래

머신러닝 학습모델

➢ 강화 학습 (Reinforcement Learning)

- 에이전트(agent) : 강화형 기계 학습의 대상이 되는 컴퓨터 프로그램
- 에이전트는 주어진 상태에서 자신이 취할 행동을 표현하는 최대의 보상을 받을 수 있는 정책 (policy) 을 수립하도록 학습시키는 것
- 강화학습 모델에서는 정답 키는 제공되지 않지만 일련의 허용 가능한 행동, 규칙, 잠재적 최종 상태가 입력됩니다.
- 강화학습 모델에서 '보상'은 숫자이며, 시스템에서 수집하려는 항목으로 알고리즘에 프로그래밍 됩니다



강화 학습의 게임 예 :

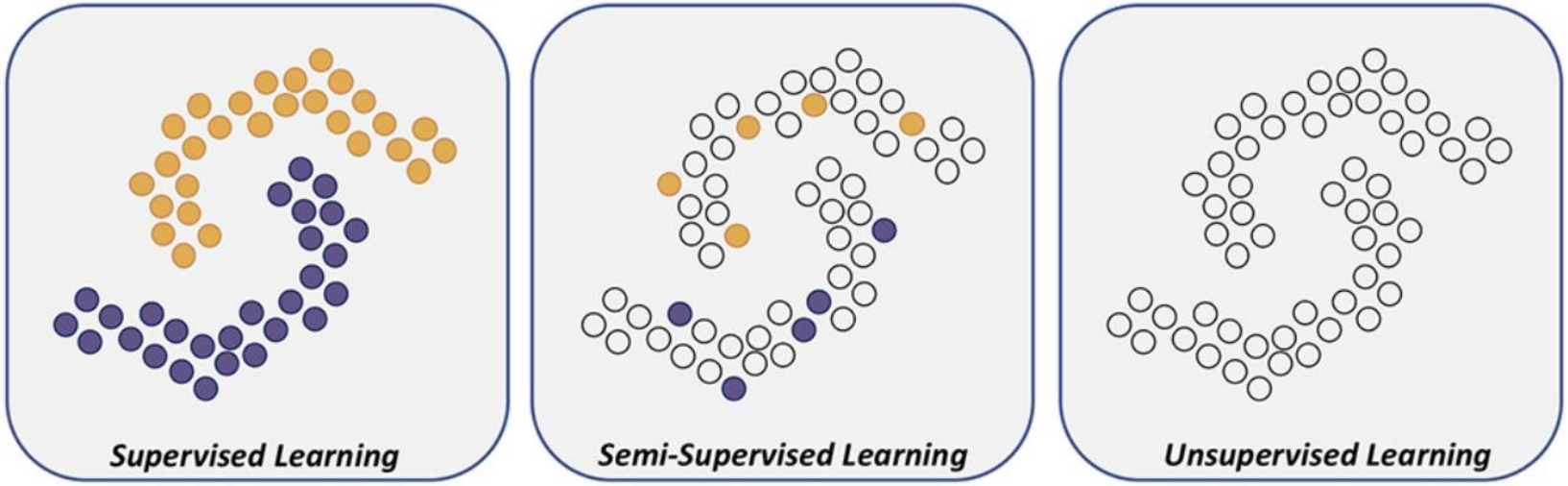
게임의 규칙을 따로 입력하지 않고 자신(Agent)이 게임 환경(environment)에서 현재 상태(state)에서 높은 점수(reward)를 얻는 방법을 찾아가며 행동(action)하는 학습 방법으로 특정 학습 횟수를 초과하면 높은 점수(reward)를 획득할 수 있는 전략이 형성되게 됩니다.

단, 행동(action)을 위한 행동 목록(방향키, 버튼)등은 사전에 정의가 되어야 합니다.

머신러닝 학습모델

➢ 준지도 학습(Semi-supervised learning)

- 소량의 labeled data에는 supervised learning을 적용하고 대용량 unlabeled data에는 unsupervised learning을 적용해 추가적인 성능향상을 목표로 하는 방법론
- 준지도학습에서는 한 쪽의 데이터에 있는 추가 정보를 활용해 다른 데이터 학습에서의 성능을 높이는 것을 목표로 합니다
- 정답 데이터를 수집하는 '데이터 레이블링' 작업에 소요되는 많은 자원과 비용 때문에 등장
- 분류 분야에서는 기존 지도학습 데이터에 레이블링되지 않은 데이터 정보를 추가로 사용해 성능을 향상시키고,
- 클러스터링 분야에서는 새로운 데이터를 어느 클러스터에 넣을지 결정함에 있어 도움을 줄 수 있다



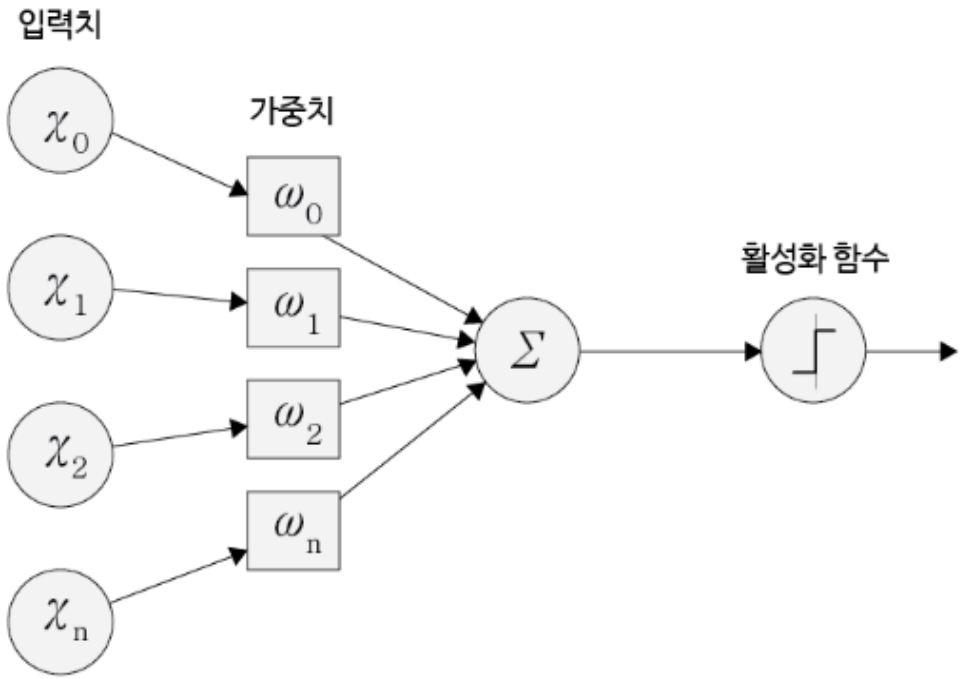
머신러닝 적용사례

➢ 기업의 머신러닝 적용 영역

- **추천 엔진** : 소매 및 쇼핑 플랫폼, 음악 및 동영상 스트리밍 서비스에서 선호하는 스트리밍 서비스의 추천 시스템 (Netflix 또는 Spotify)
- **디지털 마케팅** : 소비자 상호 작용을 기반으로 한 데이터를 통해 잠재 고객에 대한 접근 방식을 개인화
- **ERP 및 프로세스 자동화** : ERP 데이터베이스의 데이터에서 상관관계와 패턴을 찾고, 확보한 인사이트로 거의 모든 비즈니스 영역에 정보를 제공할 수 있습니다.
  네트워크 내 사물인터넷(IoT) 기기의 워크플로 최적화, 반복적이거나 오류가 잦은 업무 자동화를 위한 최선책 찾기
- **예측 정비** : 고장과 비효율로 막대한 비용과 운영 중단이 발생할 수 있습니다.
- **교육** : 스마트 교육 개인화 실현
- **검색 엔진** : 검색에 더 관련성 있는 결과를 표시하도록 지속적으로 지원
- **의료 분야** : 다른 사례에서 수집 한 정보를 기반으로 치료 프로토콜에 대한 제안을 제공
- **가상 비서 강화**
- **고객 행동 예측**

딥러닝

- 인간의 사고를 모방하기 위해 만들어진 인공 신경망(Artificial Neural Networks)을 만들고, 다중으로 쌓아 만든 심층 신경망(Deep Neural Networks)을 이용하여
- 머신 러닝 하위 분야
- 무인 자동차에서 활용되는 핵심 기술 (자동차가 정지 신호를 인식하고 보행자와 가로등을 구별)



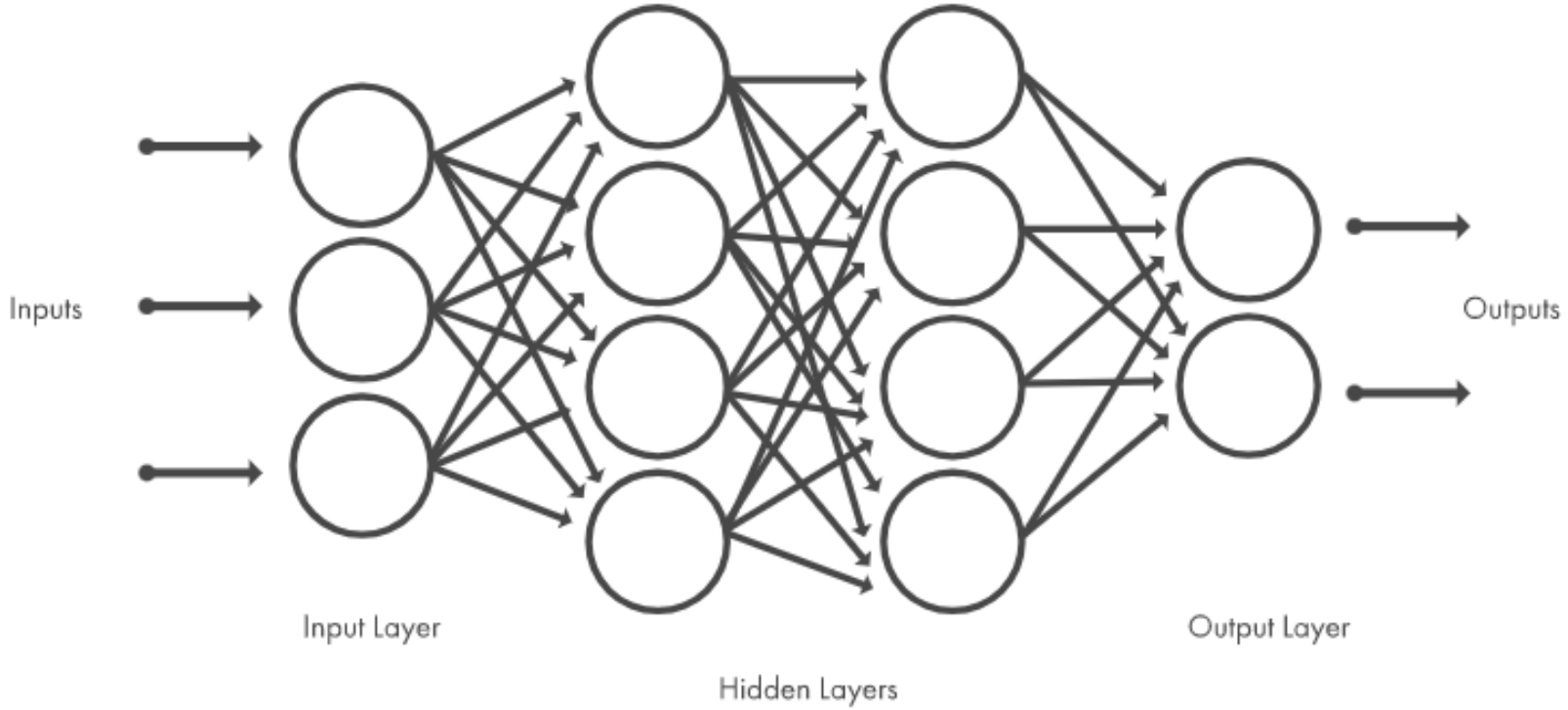
단층 퍼셉트론(single layer perceptron)이 동작

1. 각 노드의 입력치와 가중치를 서로 곱하여 모두 합한다.
2. 가중치곱의 합한 값을 활성화 함수의 임계치(선택의 기준이 되는 값)와 비교
3. 임계치보다 크면 뉴런은 활성화되고, 임계치보다 작으면 뉴런은 비활성화 된다.

퍼셉트론에서 가중치와 임계치를 적절히 변경하여, 상황에 맞는 적절한 의사결정을 수행

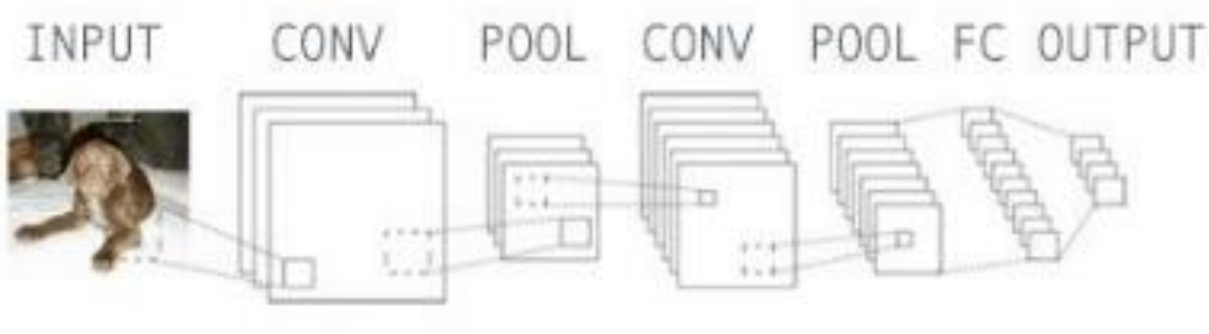
➢ 다층 퍼셉트론(MultiLayer Perceptron, MLP)

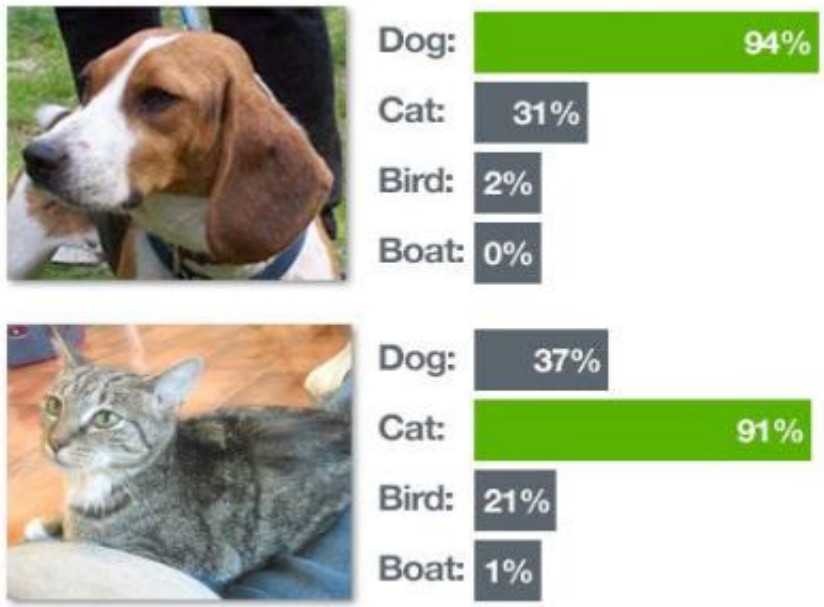
- 단층 퍼셉트론을 여러 개 조합하면 더욱 복잡한 문제도 판단할 수 있고, 비선형 문제 해결
- 심층 신경망 (Deep Neural Networks)
- "Deep"이라는 용어는 뉴럴 네트워크를 구성하는 숨겨진 레이어(Hidden Layer)의 수를 의미



➢ 딥 신경망 유형

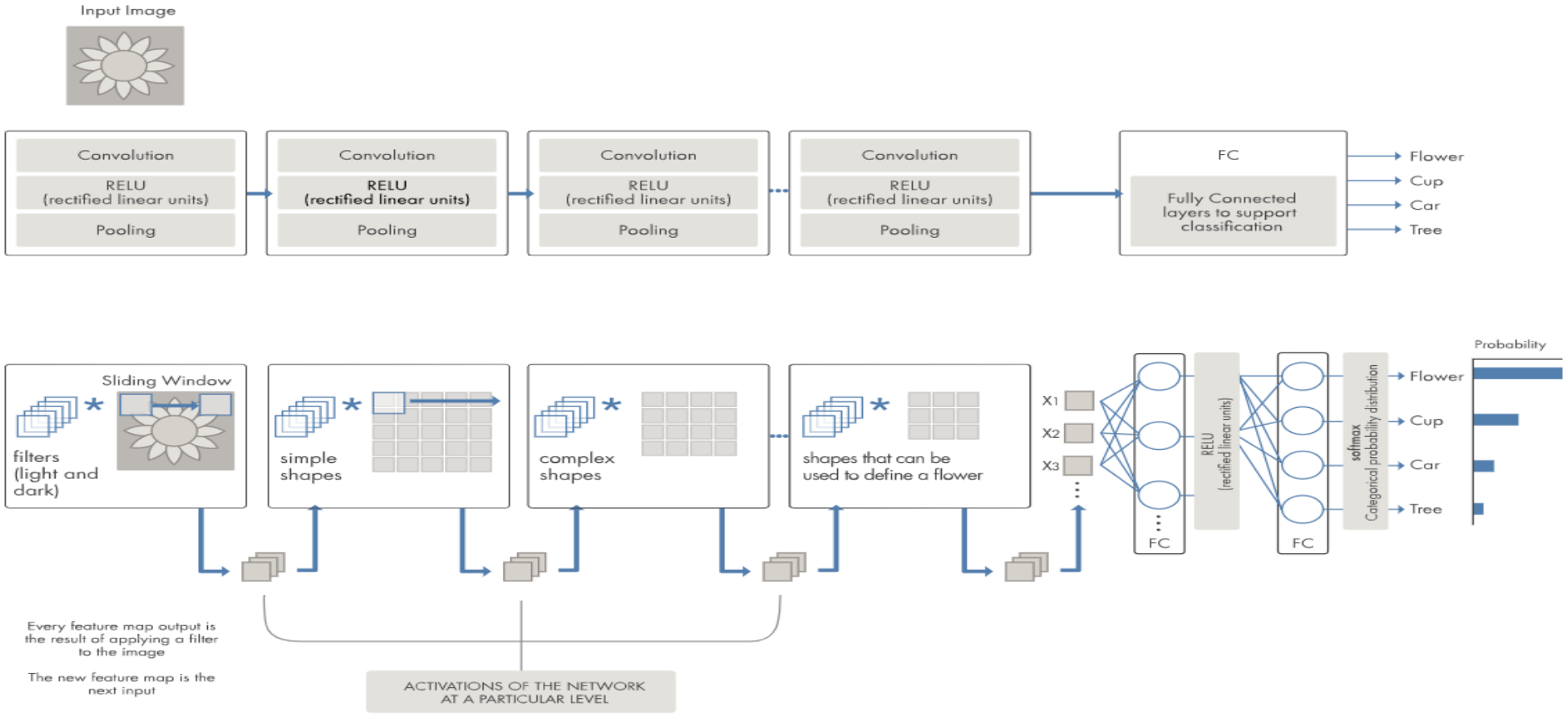
- CNN(Convolutional Neural Networks) ConvNet
- 입력 데이터에 대해 2D 컨벌루션 레이어를 사용으로써 특징을 추출, 이미지와 같은 2차원 데이터 처리에 적합한 아키텍처입니다.
- CNN은 이미지에서 직접 특징을 추출하여 작동됩니다.
- 자동화된 특징 추출은 객체 분류와 같은 컴퓨터 비전 작업에서 딥러닝 모델을 매우 정확하게 구현합니다.





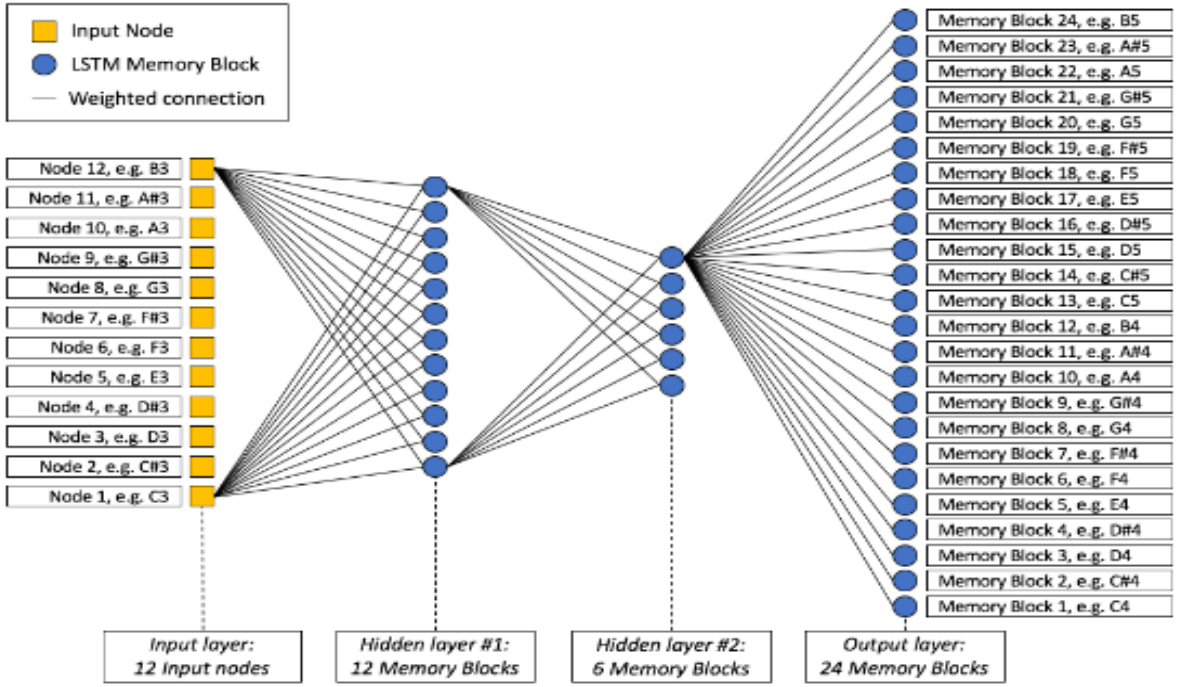
➢ 딥 신경망 유형

- CNN(Convolutional Neural Networks) : 수십 개 또는 수백 개의 숨겨진 레이어를 사용하여 이미지의 다른 특징을 감지하는 방법을 학습
  숨겨진 모든 레이어는 학습된 이미지 특징의 복잡도를 증대합니다.



➢ 딥 신경망 유형

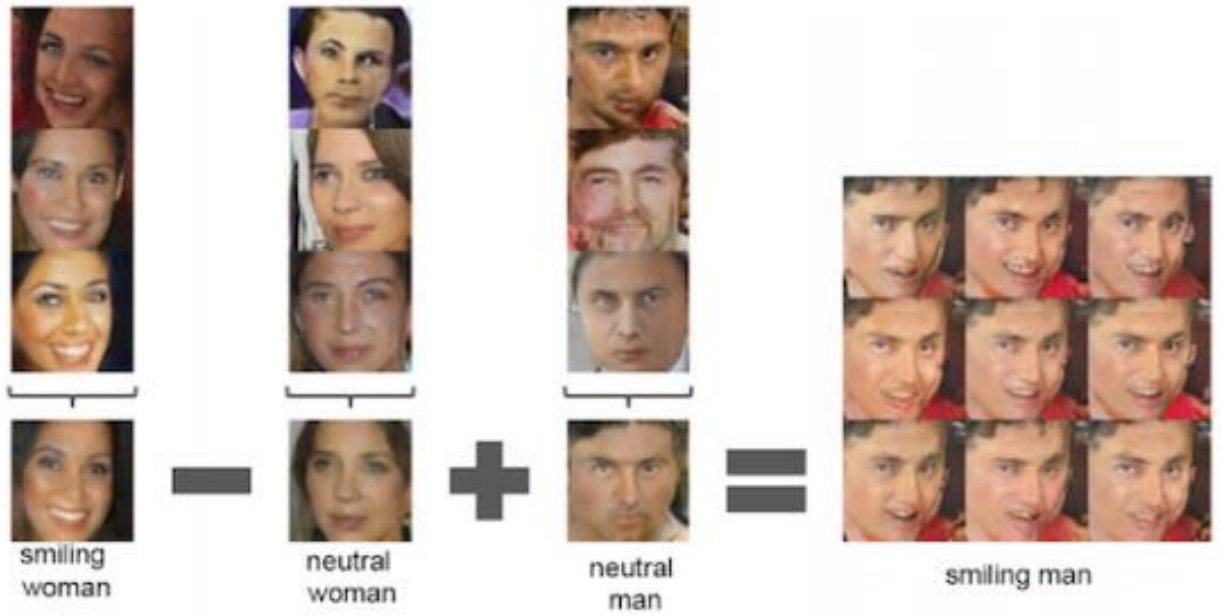
- 순환신경망(RNN. Recurrent Neural Network) : 순차적 정보가 담긴 데이터에서 규칙적인 패턴을 인식하고, 추상화된 정보를 추출
- 텍스트, 음성, 음악, 영상 등 순차적 데이터를 다루는 데 적합
- 그레디언트 소실 문제(Gradient Vanishing Problem)가 있어 패턴 학습을 못하는 경우를 개선하기 위해, LSTM(Long Short Term Memory)가 개발되었다.
- 자동 작곡, 작사, 저술, 주가 예측 등 다양한 분야에 활용 되고 있다.



LSTM 기반 음악 작곡(LSTM RNN Music Composition)

➢ 딥 신경망 유형

- GAN(Generative Adversarial Network. 생성 대립 신경망) : 비지도 학습 방법, 훈련으로 학습된 패턴을 이용해 이미지나 음성을 생성할 수 있다.
- GAN은 이미지 및 음성 복원 등에 적용되었다.
- DCGAN(Deep Convolution GAN)은 불안정한 GAN 구조를 개선해 새로운 의미를 가진 이미지를 생성할 수 있다.

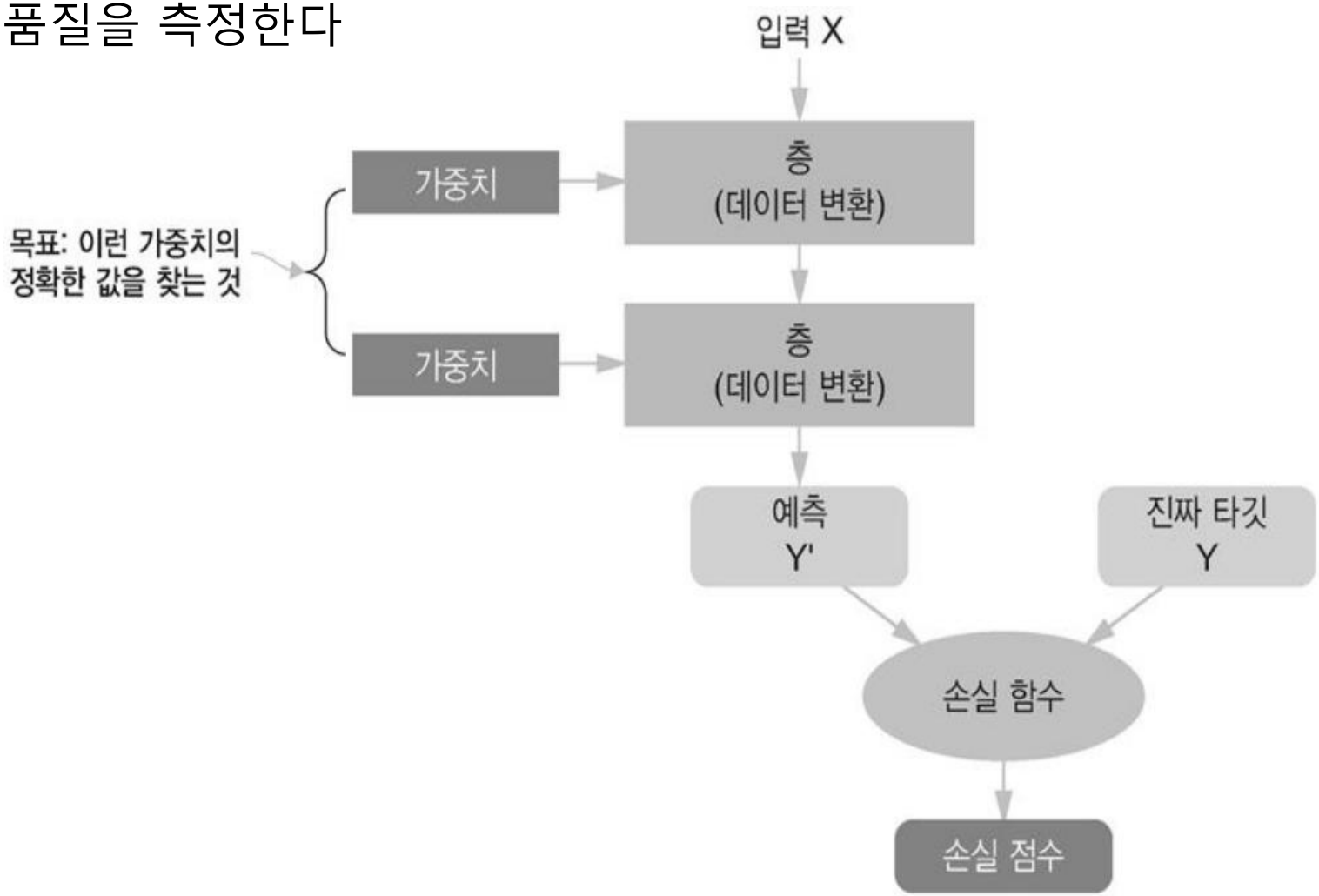


DCGAN(Unsupervied Representation Learning with Deep Convolutional Generative Adversarial Network, 2016)

딥러닝

➢ 딥러닝 동작 원리

- 신경망은 가중치를 파라미터로 가진다
- 손실 함수가 신경망의 출력 품질을 측정한다



➢ 딥러닝 모델 생성 방법

1. **데이터셋 생성하기**
   - 데이터들로부터 train, validation, test 데이터 셋을 생성
   - 딥러닝 모델이 데이터를 읽고, 학습하기 위한 포맷 변환
2. **모델 구성하기**
   - Sequence 모델을 생성, 필요한 레이어를 추가, 구성
3. **모델 학습과정 설정하기**
   - 학습에 손실함수, 최적화 방법 대한 설정
   - 케라스에서는 compile( ) 함수를 이용하여 학습과정을 설정
4. **모델 학습**
   - 구성한 모델을 fit( ) 함수를 이용하여 train 데이터 셋을 학습
5. **학습과정 살펴보기**
   - 모델 학습 시, train, validation 데이터셋의 손실 및 정확도 등을 측정
   - 반복횟수에 따른 손실 및 정확도의 추이를 보며 학습 상황을 판단합니다.
6. **모델 평가하기** - evaluate( ) 함수로 test 데이터셋으로 모델을 평가
7. **모델 사용하기** - predict( ) 함수로 학습에 사용되지 않는 입력값에 대한 모델의 예측 출력값을 얻습니다.

➢ 딥러닝 모델 생성 방법

```
# 0. 필요한 패키지 불러오기
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np

# 1. 데이터셋 생성하기
(X_train, y_train), (X_test, y_test) = mnist.load_data()
# 차원 변환 후, 테스트셋과 학습셋으로 나눕니다.
X_train = X_train.reshape(X_train.shape[0], 784).astype('float32') / 255
X_test = X_test.reshape(X_test.shape[0], 784).astype('float32') / 255
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# 2. 모델 구성하기
model = Sequential()
model.add(Dense(units=512, input_shape=(28*28,), activation='relu'))
```

➢ 딥러닝 모델 생성 방법

```
model.add(Dense(units=10, activation='softmax'))
model.summary()

# 3. 모델 학습과정(실행 환경) 설정하기
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# 모델 최적화를 위한 설정 구간입니다.
modelpath = "./MNIST_MLP.hdf5"
checkpointer = ModelCheckpoint(filepath=modelpath, monitor='val_loss', verbose=1,
save_best_only=True)
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=10)

# 4. 모델 학습시키기
history = model.fit(X_train, y_train, validation_split=0.25, epochs=30, batch_size=200, verbose=0,
callbacks=[early_stopping_callback, checkpointer])
# 테스트 정확도를 출력합니다.
print("₩n Test Accuracy: %.4f" % (model.evaluate(X_test, y_test)[1]))
# 검증셋과 학습셋의 오차를 저장합니다.
y_vloss = history.history['val_loss']
y_loss = history.history['loss']
```

➢ 딥러닝 모델 생성 방법

```
# 5. 학습과정 살펴보기( 그래프로 표현 )
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')
# 그래프에 그리드를 주고 레이블을 표시해 보겠습니다.
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
# 6. 모델 평가하기
loss_and_metrics = model.evaluate(X_test, y_test, batch_size=32)
print('## evaluation loss and_metrics ##')
print(loss_and_metrics)
# 7. 모델 사용하기
xhat = X_test[0:1]
yhat = model.predict(xhat)
print('## yhat ##')
print(yhat)
print(y_test)
```

딥러닝

➢ 딥러닝 적용 분야

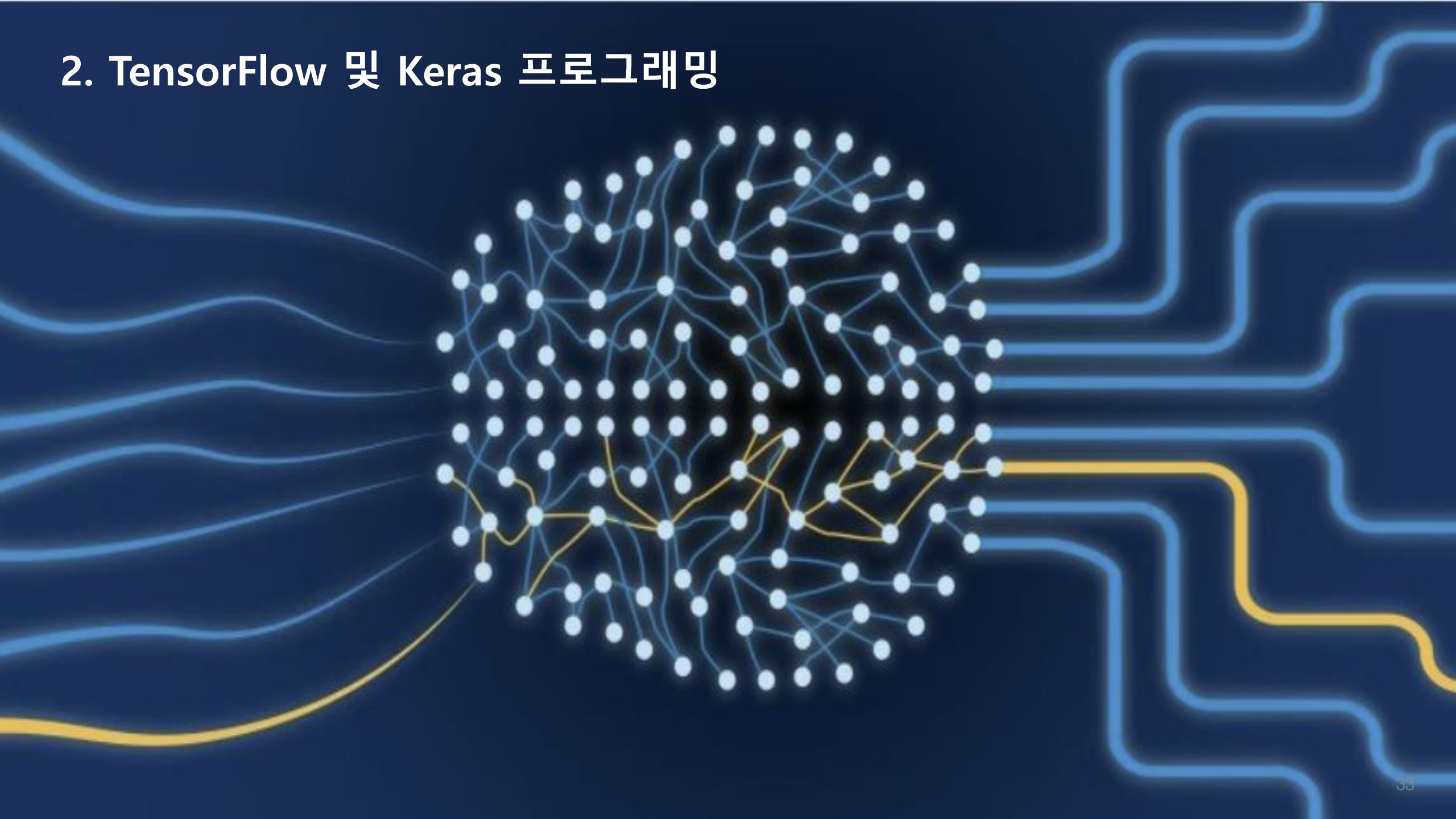
- **자율 주행** : 정지 신호, 신호등과 같은 물체를 자동으로 탐지
  딥러닝은 보행자를 탐지하는 데도 사용되어 사고를 줄이는 데 기여
- **항공 우주 및 방위** : 위성에서 객체를 식별하여 관심 영역을 찾고 병력을 파견하기에 안전하거나 안전하지 않은 지역을 확인하는 데 사용.
- **의학 연구** : 암 세포를 정확히 식별하는 법을 학습하는 데 사용될 고차원 데이터 세트를 생성하는 첨단 현미경을 개발
- **산업 자동화** : 사람이나 물체가 기계와 안전 거리를 유지하지 않을 때 딥러닝이 이를 자동으로 탐지하여 중장비를 다루는 작업자 안전을 개선하는 데 도움이 됩니다.
- **전자** : 자동 청취 및 음성 번역에 딥러닝이 사용되고 있습니다.
  예] 사용자의 음성에 응답하고 사용자의 기호를 파악하는 가전 기기
- **이미지 채색**
- **안면 인식**
- **챗봇 및 서비스 봇** : 음성 및 텍스트 기반 쿼리에 대한 지능적인 답변을 제공 예) Chat GPT

딥러닝

➢ 딥러닝 성과

- 사람과 비슷한 수준의 이미지 분류
- 사람과 비슷한 수준의 음성 인식
- 사람과 비슷한 수준의 필기 인식
- 향상된 기계 번역
- 향상된 TTSText-To-Speech 변환
- 구글 나우Now와 아마존 알렉사Alexa 같은 디지털 비서
- 사람과 비슷한 수준의 자율 주행 능력
- 구글, 바이두Baidu, 빙Bing에서 사용하는 향상된 광고 타기팅targeting
- 향상된 웹 검색 엔진의 결과
- 자연어 질문에 대답하는 능력
- 사람을 능가하는 바둑 실력

2. TensorFlow 및 Keras 프로그래밍



TensorFlow 프로그래밍

➢ TensorFlow

- 구글이 공개, 데이터 플로우 그래프를 이용한 수치 계산을 위한 오픈 소스 라이브러리
- 텐서를 포함한 계산을 정의하고 실행하는 프레임워크
- 상업적인 용도로 사용할 수 있는 오픈소스(Apache 2.0)
- https://www.tensorflow.org
- GPU를 이용한 병렬처리 등이 가능
- TensorFlow, TensorBoard, TensorFlow Serving이라는 소프트웨어들의 묶음

```
pip install tensorflow

import tensorflow as tf
print(tf.__version__)
```

* **TensorFlow :** 모델 정의, 데이터 학습
* **TensorBoard :** 네트워크 시각화 프로그램
* **TensorFlow Serving :** 이미 학습된 텐서플로우 모델을 쉽게 활용할 수 있게 하는 소프트웨어. 모델을 만들고 돌려보는 작업을 훨씬 빠르게 반복적으로 할 수 있도록 해줍니다.

TensorFlow 프로그래밍

➢ 텐서 연산과 미니배치

- 텐서(Tensor) : 텐서플로우가 연산하기 위해 사용하는 자료형, 데이터의 형태
  벡터와 행렬을 일반화한 다차원 숫자 배열
  고차원으로 확장 가능
- 0차원 스칼라, 1차원 벡터, 2차원 행렬, 3차원 이상의 숫자 배열 모두 텐서 .
- 벡터 연산

```
import tensorflow as tf
#tf.constant()함수로 정수인 상수 텐서 생성
a=tf.constant(2)
b=tf.constant(3)
c=tf.constant(5)
#다양한 텐서 연산 함수('+,-'와 같은 수식 기호도 사용 가능)
add = tf.add(a, b)
sub = tf.substract(a, b)
mul = tf.multiply(a, b)
div = tf.divide(a, b)
#텐서 값 출력(텐서를 넘파이 배열로 변환)
print("add=", add.numpy())
```

TensorFlow 프로그래밍

➢ Tensor

- 텐서 (Tensor) 객체의 **랭크 (Rank)** : 차원의 수 (n-dimension)
  order또는 degree, n-dimension
- tf.rank() - 텐서의 랭크를 반환
- 텐서의 형태(shape) : 각 차원에 있는 원소 개수로 표현
- 텐서플로는 그래프 계산 과정에서 자동으로 텐서 형태를 추론합니다

```
#텐서의 차원-랭크 확인
import tensorflow as tf
scalar = tf.constant(1)
vector = tf.constant([1, 2, 3])
matrix = tf.constant([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
tensor = tf.constant([[[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                      [[1, 2, 3], [4, 5, 6], [7, 8, 9]],
                      [[1, 2, 3], [4, 5, 6], [7, 8, 9]]])
print(tf.rank(scalar), tf.rank(vector), tf.rank(matrix), tf.rank(tensor))
#tf.constant()는 상수 텐서를 만듭니다.
a = tf.constant(1)
b = tf.constant([2])
c = tf.constant([[1, 2], [3, 4]])
print(a, b, c)
```

TensorFlow 프로그래밍

➢ Tensor

- tf.zeros(shape) - 모든 요소가 0인 텐서를 만듭니다.
- tf.ones(shape) - 모든 요소가 1인 텐서를 만듭니다.
- tf.range() : 파이썬 range()와 비슷하게, 주어진 범위와 간격을 갖는 숫자들의 시퀀스를 만듭니다
- tf.linspace() : numpy.linspace()와 비슷하게, 주어진 범위를 균일한 간격으로 나누는 숫자의 시퀀스를 반환

```
a = tf.zeros(1)          #인수는  생성될 Tensor의 구조 (shape)
b = tf.zeros([2])
c = tf.zeros([2, 3])
print(a, b, c)

a = tf.ones(3)
b = tf.ones([4])
c = tf.ones([2, 2, 2])
print(a, b, c)

a = tf.linspace(0, 1, 3)
b = tf.linspace(0, 3, 10)
print(a, b)
```

TensorFlow 프로그래밍

➢ Tensor 자료형과 형태, 수학연산

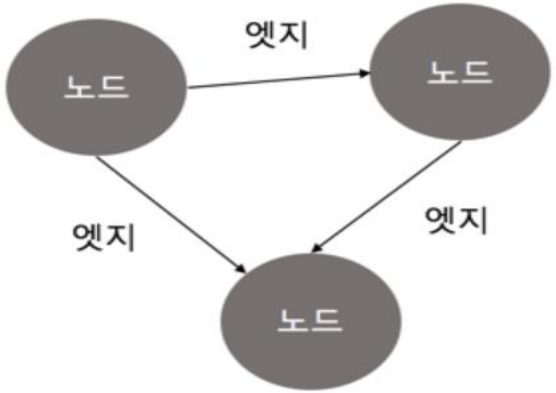
- dtype, shape 속성 : 텐서 (Tensor)의 자료형과 형태

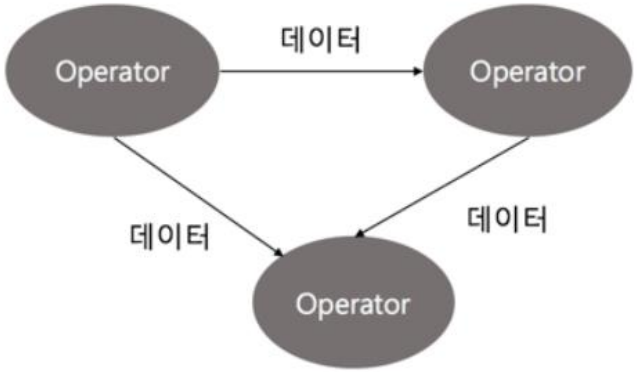
```
a = tf.constant(1)
b = tf.constant([1., 2.])
c = tf.constant([[1, 2., 3]])
d = tf.constant([[1, 2], [3, 4]])
print(a.dtype, a.shape)
print(b.dtype, b.shape)
print(c.dtype, c.shape)
print(d.dtype, d.shape)

# 수학연산 함수
a = tf.add(1, 2)
b = tf.subtract(10, 5)
c = tf.square(3)
d = tf.reduce_sum([1, 2, 3])
e = tf.reduce_mean([1, 2, 3])
print(a, b, c, d, e)
```

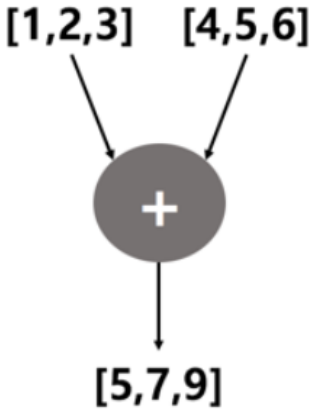
TensorFlow 프로그래밍

➢ 텐서플로우의 Data Flow Graph Computation

- 노드를 연결하는 엣지(데이터) : 텐서(Tensors)
- 노드는 데이터를 통해 수 곱하고, 나누는 등 텐서를 처리하는 연산 역할
- 엣지의 방향은 텐서의 흐름을 의미
- Data Flow Graph Computation : 데이터가 edge역할을 하여 node로 흘러가는 그래프 구조를 갖으며 node에 지정된 연산을 하는 연산방법







a = [1,2,3], b = [4,5,6] 일 때
a + b를 구하는 것을
Data Flow Graph로 표현

TensorFlow 프로그래밍

➢ TensorFlow

- 텐서 (Tensor)는 NumPy 어레이와 비슷하지만, 텐서는 GPU, TPU와 같은 가속기에서 사용할 수 있고, 텐서는 값을 변경할 수 없습니다
- 텐서의 numpy() : 텐서를 NumPy 어레이로 변환
- TensorFlow 연산은 자동으로 NumPy 어레이를 텐서로 변환

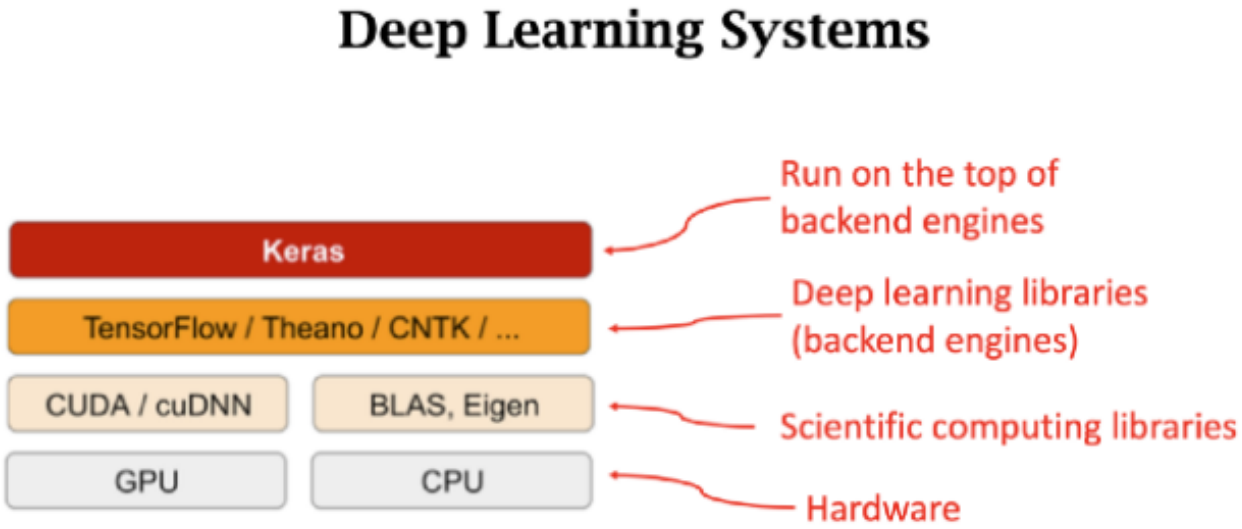
```
import numpy as np

a = tf.constant([1, 2, 3])
print(a)
print(a.numpy())

b = np.ones(3)
print(b)
print(tf.multiply(b, 3))
```

```
import tensorflow as tf
#"hello world" 문자열을 가지는 상수 오퍼레이션을 생성

tensor_a = tf.constant('hello world')
print(tensor_a)
tensor_a.numpy()
print(type(tensor_a))
print(type(tensor_a.numpy()))
print(tensor_a.numpy().decode('utf-8'))
print(type(tensor_a.numpy().decode('utf-8')))
tensor_a = tf.constant("안녕")
print(tensor_a.numpy())
print(tensor_a.numpy().decode('utf-8'))
```

Keras API

➢ 딥러닝 라이브러리 Keras

- 모든 종류의 딥러닝 모델을 간편하게 만들고 훈련시킬 수 있는 고수준의 구성 요소를 제공하는 파이썬 딥러닝 프레임워크
- 동일한 코드로 CPU와 GPU에서 실행 가능
- 컴퓨터 비전을 위한 합성곱 신경망, 시퀀스 처리를 위한 순환 신경망을 지원
- https://keras.io/ or https://www.tensorflow.org/guide/keras?hl=ko
- 다중 입력이나 다중 출력 모델, 층의 공유, 모델 공유 등 네트워크 구조도 만들 수 있습니다. (Generative Adversarial Network, 뉴럴 튜링 머신Neural Turing Machine 딥러닝 모델에 적합).
- MIT 라이선스를 따르므로 상업적인 프로젝트에도 자유롭게 사용 가능

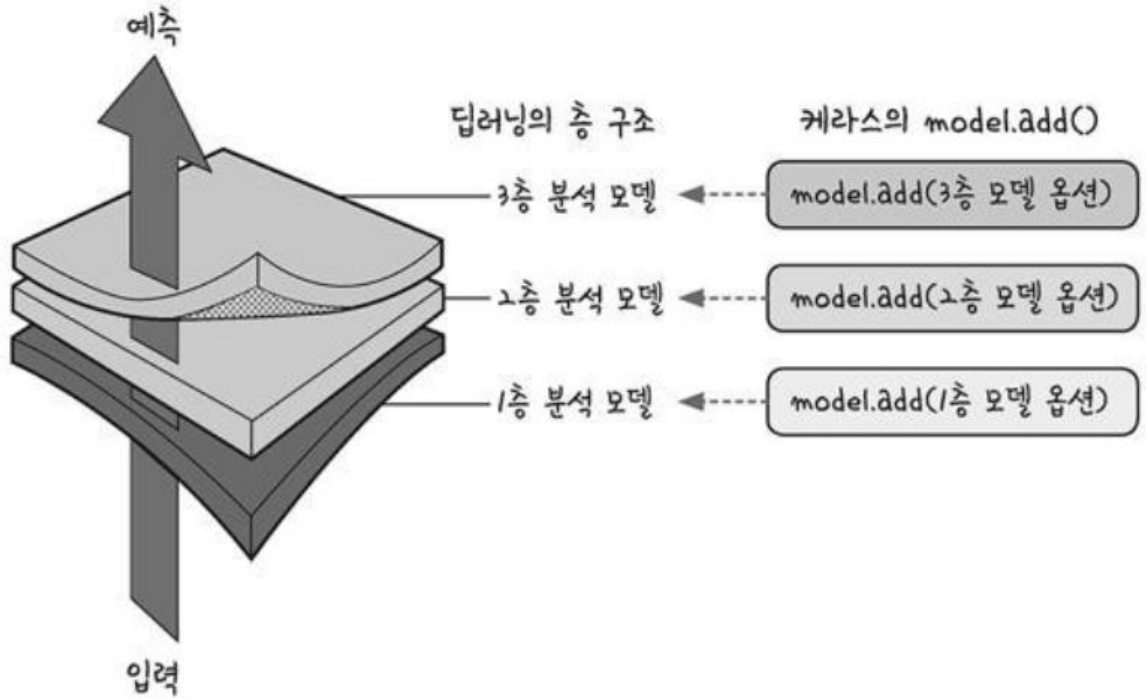


딥러닝 소프트웨어와 하드웨어 스택

Keras API

➢ Keras 작업 흐름

1. 입력 텐서와 타깃 텐서로 이루어진 훈련 데이터를 정의
2. 입력과 타깃을 매핑하는 층으로 이루어진 네트워크(또는 모델)를 정의
3. 손실 함수, 옵티마이저, 모니터링하기 위한 측정 지표를 선택하여 학습 과정을 설정
4. 훈련 데이터에 대해 모델의 fit() 메서드를 반복적으로 호출



Keras API

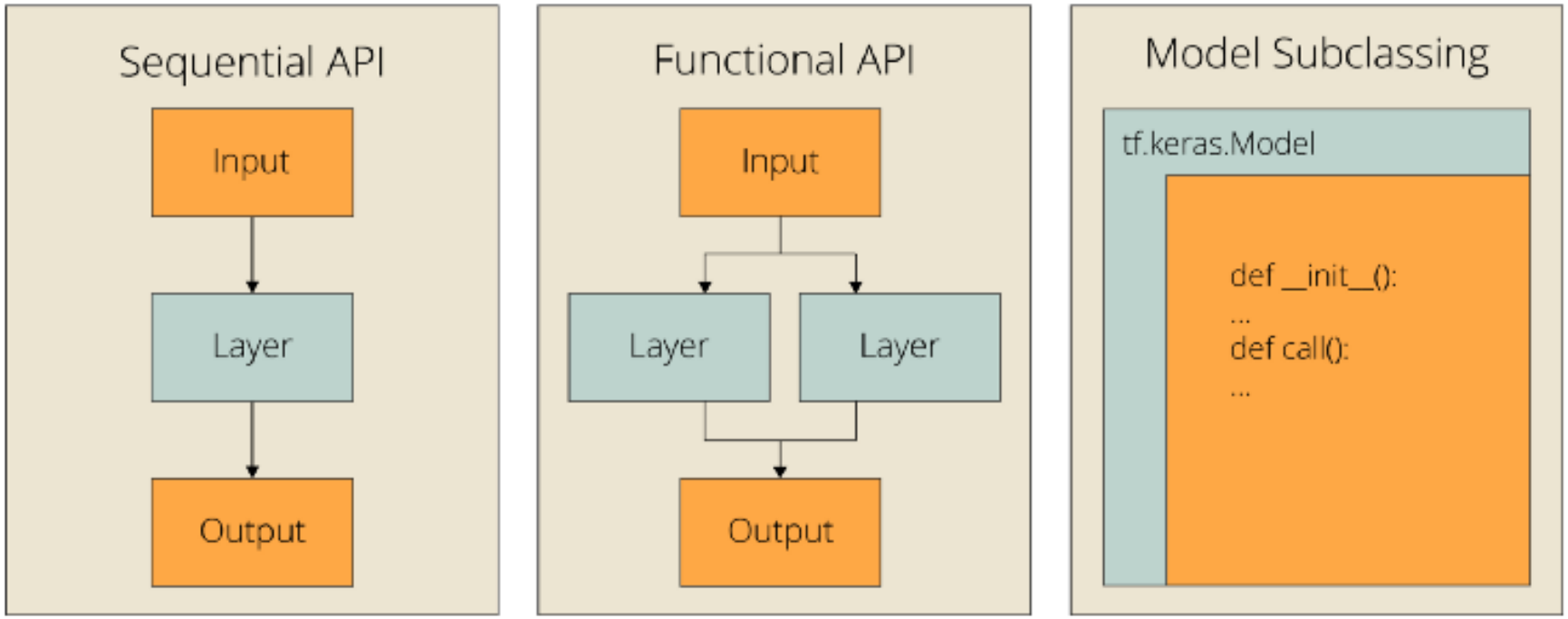
➢ Keras Sequential 클래스

- tf.keras.Sequential : '순차적으로 층을 쌓은 인공신경망 모델'
- tf.keras.layers.Dense(activation=, loss=, optimizer=) : 완전 연결층을 만들기 위한 클래스 , 은닉층과 출력층을 Dense 객체로 구성
  - 딥러닝의 각 층(Dense), 활성화 함수, Dropout 등 을 Sequential의 add()로 추가
  - activation: 다음 층으로 어떻게 값을 넘길지 결정하는 부분(가장 많이 사용되는 relu와 sigmoid 함수)
  - loss: 한 번 신경망이 실행될 때마다 오차 값을 추적하는 함수
  - optimizer: 오차를 어떻게 줄여 나갈지 정하는 함수
- compile(loss=' ', optimizer=' ') : 모델 학습을 위한 설정
- 손실함수 : Neural Network의 예측이 얼마나 잘 맞는지 측정하는 역할
- 옵티마이저 : 더 개선된 예측값을 출력하도록 최적화하는 알고리즘
- fit(x, y, epoch) : 모델에 데이터를 학습
- 에포크 (epoch) : 주어진 데이터를 한 번 훈련하는 단위
- Keras로 머신러닝을 수행할 때 Numpy 배열 데이터를 전달해야 한다
- tf.keras.Model.evaluate() - 테스트 데이터를 통해 학습한 모델에 대한 정확도를 평가
- tf.keras.Model.predict() – 학습된적 없는 입력 데이터에 대한 모델의 출력값을 예측

Keras API

➢ 모델을 정의하는 방법

1. Sequential 클래스 -가장 자주 사용하는 구조인 층을 순서대로 쌓아 올린 네트워크
2. 함수형 API - 완전히 임의의 구조를 만들 수 있는 비순환 유향 그래프를 만듭니다
3. Subclassing API



Keras API

➢ Keras Sequential 클래스

- add() : 층을 단계적으로 추가
- summary() : 모델의 정보를 요약

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

#Embedding()을 통해 생성하는 임베딩 층(embedding layer)을 추가
model = Sequential()
model.add(Embedding(vocab_size, output_dim, input_length))

# 전결합층(fully-connected layer)을 추가
model = Sequential()
model.add(Dense(1, input_dim=3, activation='relu'))
```

- save() : 인공 신경망 모델을 hdf5 파일에 저장
  학습이 끝난 신경망의 구조를 보존하고 계속해서 사용할 수 있다는 의미입니다.
- load_model() : 저장해둔 모델을 불러옴

```
model.evaluate(X_test, y_test, batch_size=32)
model.predict(X_input, batch_size=32)
model.save("model_name.h5")
from tensorflow.keras.models import load_model
model = load_model("model_name.h5")
```

Keras API

➢ Keras Sequential 클래스

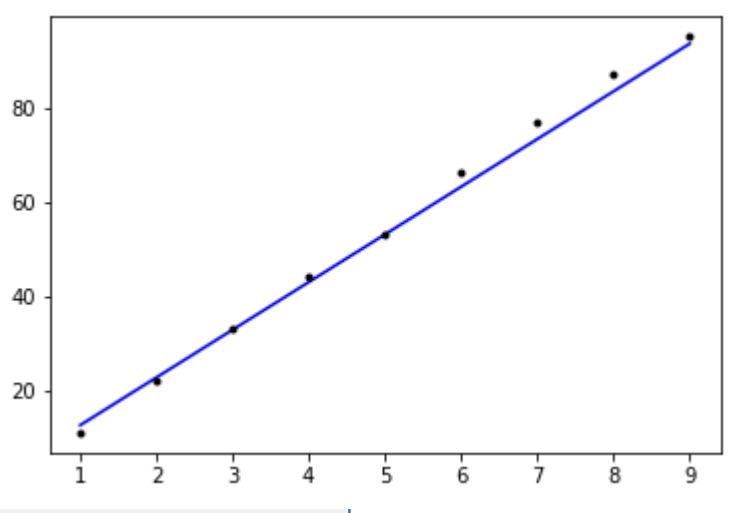
- 검증 데이터를 사용하면 각 에포크마다 검증 데이터의 정확도나 오차를 함께 출력
- 검증 데이터의 오차(loss)가 낮아지다가 높아지기 시작하면 이는 과적합(overfitting)의 신호입니다.
- verbose : 학습 중 출력되는 문구를 설정
  - 0 : 아무 것도 출력하지 않습니다.
  - 1 : 훈련의 진행도를 보여주는 진행 막대를 보여줍니다.
  - 2 : 미니 배치마다 손실 정보를 출력합니다.

```
#validation_data(x_val, y_val)  검증 데이터(validation data)를 사용
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0, validation_data(X_val, y_val))

# 훈련 데이터의 20%를 검증 데이터로 사용
model.fit(X_train, y_train, epochs=10, batch_size=32, verbose=0, validation_split=0.2))
```

Keras API

➢ **케라스 Sequential API로 구현하는 선형 회귀**

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers

x = [1, 2, 3, 4, 5, 6, 7, 8, 9] # 공부하는 시간
y = [11, 22, 33, 44, 53, 66, 77, 87, 95] # 공부하는 시간에 맵핑되는 성적
model = Sequential()
# 출력 y의 차원 units은 1. 입력 x의 차원(input_dim)은 1
# 선형 회귀이므로 활성화함수는 'linear'
model.add(Dense(1, input_dim=1, activation='linear'))
# sgd는 경사 하강법을 의미. 학습률(learning rate, lr)은 0.01.
sgd = optimizers.SGD(lr=0.01)
# 손실 함수(Loss function)는 평균제곱오차 mse를 사용
model.compile(optimizer=sgd, loss='mse', metrics=['mse'])
# 주어진 x와 y데이터에 대해서 오차를 최소화하는 작업을 300번 시도합니다.
model.fit(x, y, epochs=300)
plt.plot(x, model.predict(x), 'b', x, y, 'k.')
```



Keras API

➢ Keras Functional API

- Sequential API는 여러층을 공유하거나 다양한 종류의 입력과 출력을 사용하는 등의 복잡한 모델을 만드는 일에는 한계가 있습니다
- Functional API는 모델을 구성하기 위해 층(layer)을 함수처럼 호출하여 연결합니다.
- 층 간의 입력과 출력을 명시적으로 정의하여 복잡한 모델 구조를 구성할 수 있습니다.
- 다중 입력 및 다중 출력 모델, 공유 층(Shared Layer), 분기 모델(Branching Model) 등의 복잡한 모델을 쉽게 구현할 수 있습니다.
- Model 클래스를 사용하여 컴파일 및 학습
- Functional API는 정적인 모델 구조에 적합하며, 모델의 각 부분을 재사용하고 공유하는 데 유용합니다.

Keras API

➢ **Keras Functional API로 구현하는 선형 회귀 .**

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras import optimizers
from tensorflow.keras.models import Model

X = [1, 2, 3, 4, 5, 6, 7, 8, 9] # 공부하는 시간
y = [11, 22, 33, 44, 53, 66, 77, 87, 95] # 각 공부하는 시간에 맵핑되는 성적

inputs = Input(shape=(1,))
output = Dense(1, activation='linear')(inputs)
linear_model = Model(inputs, output)

sgd = optimizers.SGD(lr=0.01)

linear_model.compile(optimizer=sgd, loss='mse', metrics=['mse'])
linear_model.fit(X, y, epochs=300)
```

Keras API

➢ Keras Functional API로 구현하는 로지스틱 회귀

```
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

inputs = Input(shape=(3,))
output = Dense(1, activation='sigmoid')(inputs)
logistic_model = Model(inputs, output)
```

Keras API

➢ Keras Functional API로 다중 입력 (multiple inputs) 을 받는 모델

```
from tensorflow.keras.layers import Input, Dense, concatenate
from tensorflow.keras.models import Model
# 두 개의 입력층을 정의
inputA = Input(shape=(64,))
inputB = Input(shape=(128,))
# 첫번째 입력층으로부터 분기되어 진행되는 인공 신경망을 정의
x = Dense(16, activation="relu")(inputA)
x = Dense(8, activation="relu")(x)
x = Model(inputs=inputA, outputs=x)
# 두번째 입력층으로부터 분기되어 진행되는 인공 신경망을 정의
y = Dense(64, activation="relu")(inputB)
y = Dense(32, activation="relu")(y)
y = Dense(8, activation="relu")(y)
y = Model(inputs=inputB, outputs=y)
# 두개의 인공 신경망의 출력을 연결(concatenate)
result = concatenate([x.output, y.output])
z = Dense(2, activation="relu")(result)
z = Dense(1, activation="linear")(z)
model = Model(inputs=[x.input, y.input], outputs=z)
```

Keras API

➢ Keras Functional API로 RNN(Recurrence Neural Network) 은닉층 구현

```
from tensorflow.keras.layers import Input, Dense, LSTM
from tensorflow.keras.models import Model

inputs = Input(shape=(50,1))
lstm_layer = LSTM(10)(inputs)
x = Dense(10, activation='relu')(lstm_layer)
output = Dense(1, activation='sigmoid')(x)

model = Model(inputs=inputs, outputs=output)
```

Keras API

➢ Keras Subclassing API

- Subclassing API는 밑바닥부터 새로운 수준의 아키텍처를 구현해야 하는 실험적 연구를 하는 연구자들에게 적합합니다.
- 새로운 모델 클래스를 작성하고 call 메서드 내에서 원하는 계산을 수행하여 모델을 구성합니다.
- 모델 구조를 동적으로 조작할 수 있으므로 조건문과 반복문과 같은 제어 흐름을 사용할 수 있습니다.
- 기존 층 클래스를 상속받아 새로운 층을 만들 수 있으며, 새로운 계층 구조와 동작을 정의할 수 있습니다.
- Subclass API는 유연성이 높으며 동적인 모델 구조에 적합합니다.
- Functional API가 구현할 수 없는 모델의 특정한 커스터마이징이 필요한 경우 Subclass API를 사용할 수 있습니다.

Keras API

➢ Keras Subclassing API로 선형회귀 구현

- 클래스(class) 형태의 모델은 tf.keras.Model을 상속받습니다.
- init()에서 모델의 구조와 동적을 정의하는 생성자를 정의합니다.
- call() 메서드는 모델이 데이터를 입력받아 예측값을 리턴하는 포워드(forward) 연산을 진행시키는 함수

```
import tensorflow as tf
class LinearRegression(tf.keras.Model):
    def __init__(self):
        super(LinearRegression, self).__init__()
        self.linear_layer = tf.keras.layers.Dense(1, input_dim=1, activation='linear')

    def call(self, x):
        y_pred = self.linear_layer(x)
        return y_pred

model = LinearRegression()
X = [1, 2, 3, 4, 5, 6, 7, 8, 9] # 공부하는 시간
y = [11, 22, 33, 44, 53, 66, 77, 87, 95] # 각 공부하는 시간에 맵핑되는 성적
sgd = tf.keras.optimizers.SGD(lr=0.01)
model.compile(optimizer=sgd, loss='mse', metrics=['mse'])
model.fit(X, y, epochs=300)
```

Keras API

➢ 케라스(Keras) 전처리

- Tokenizer() : 토큰화와 정수 인코딩을 위해 사용

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer()
train_text = "The earth is an awesome place live"

# 단어 집합 생성, Tokenizer
tokenizer.fit_on_texts([train_text])

# 정수 인코딩 , Embedding
sub_text = "The earth is an great place live"
sequences = tokenizer.texts_to_sequences([sub_text])[0]

print("정수 인코딩 : ",sequences)
print("단어 집합 : ",tokenizer.word_index)
```

Keras API

➢ 케라스(Keras) 전처리

- 전체 훈련 데이터에서 각 문서 또는 각 문장은 단어의 수가 서로 다를 수 있습니다.
- 모델의 입력으로 사용하려면 모든 샘플의 길이를 동일하게 맞추어야 할 때 패딩(padding) 작업으로 보통 숫자 0을 넣어서 길이가 다른 샘플들의 길이를 맞춰줍니다.
- pad_sequence()는 정해준 길이보다 길이가 긴 샘플은 값을 일부 자르고, 정해준 길이보다 길이가 짧은 샘플은 값을 0으로 채웁니다.

```
pad_sequences([[1, 2, 3], [3, 4, 5, 6], [7, 8]], maxlen=3, padding='pre')
#maxlen = 모든 데이터에 대해서 정규화 할 길이
#padding = 'pre'를 선택하면 앞에 0을 채우고 'post'를 선택하면 뒤에 0을 채움.
```

Keras API

➢ 케라스(Keras) 전처리

- 워드 임베딩(Word Embedding)
- 텍스트 내의 단어들을 밀집 벡터(dense vector)로 만드는 것
- 원-핫 벡터는 대부분이 0의 값을 가지고, 단 하나의 1의 값을 가지는 벡터이며 벡터의 차원이 대체적으로 크다는 성질을 가졌습니다.
- 원-핫 벡터는 단어 집합의 크기만큼 벡터의 차원을 가지며 단어 벡터 간의 유의미한 유사도를 구할 수 없다는 단점이 있습니다.
- 워드 임베딩으로부터 얻은 임베딩 벡터(embedding vector)는 상대적으로 저차원을 가지며 모든 원소의 값이 실수입니다.
- 원-핫 벡터의 차원이 주로 20,000 이상을 넘어가는 것과는 달리 임베딩 벡터는 주로 256, 512, 1024 등의 차원을 가집니다.
- 베딩 벡터는 초기에는 랜덤값을 가지지만, 인공 신경망의 가중치가 학습되는 방법과 같은 방식으로 값이 학습되며 변경됩니다.
- Embedding() :임베딩 층(embedding layer)을 만드는 역할을 합니다. 정수 인코딩이 된 단어들을 입력을 받아서 임베딩을 수행합니다.

```
# 1. 토큰화
tokenized_text = [['Hope', 'to', 'see', 'you', 'soon'], ['Nice', 'to', 'see', 'you', 'again']]
# 2. 각 단어에 대한 정수 인코딩
encoded_text = [[0, 1, 2, 3, 4],[5, 1, 2, 3, 6]]
# 3. 정수 인코딩 데이터가  임베딩 층의 입력이 된다.
vocab_size = 7
embedding_dim = 2
Embedding(vocab_size, embedding_dim, input_length=5).
```

Keras API

➢ 케라스(Keras)

- compile() : 모델을 기계가 이해할 수 있도록 컴파일 합니다. 손실 함수와 옵티마이저, 메트릭 함수를 선택합니다.

```
from tensorflow.keras.layers import SimpleRNN, Embedding, Dense
from tensorflow.keras.models import Sequential

vocab_size = 10000
embedding_dim = 32
hidden_units = 32

model = Sequential()
model.add(Embedding(vocab_size, embedding_dim))
model.add(SimpleRNN(hidden_units))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
#optimizer = 훈련 과정을 설정하는 옵티마이저
#loss = 훈련 과정에서 사용할 손실 함수(loss function)
#metrics = 훈련을 모니터링하기 위한 지표
model.fit(X_train, y_train, epochs=10, batch_size=32)
```