

# 인공지능

## 06. Deep Learning

Dept. of Digital Contents



# 1. 딥러닝(Deep Learning)

---

## ○ 다층 퍼셉트론의 한계

- 그레이디언트 소멸(gradient vanishing)
  - 오류 역전파 알고리즘에서 여러 층을 거치면서 그레이디언트값이 작아져 거의 없어지는 문제
  - 입력층에 가까워지면 거의 변화가 없음
- 과대 적합(Overfitting)
  - 훈련 집합 크기는 작은 상태로 고정
  - 추정할 매개변수는 크게 늘어나는 현상
- 과다한 계산 시간

# 1. 딥러닝(Deep Learning)

---

## ○ 딥러닝(Deep Learning)

- MLP(다층 퍼셉트론)에서 은닉층의 개수를 증가시킨 것
- DNN(Deep Neural Network) 기술 혁신 요인
  - 저렴한 GPU 등장 : 손쉬운 병렬 처리 → 학습시간 10 ~ 100배 단축
  - 인터넷 : 학습 데이터가 크게 증가
  - 다양한 학습 알고리즘 개발
    - 활성함수 ReLU 발견(de facto standard)
    - 다양한 규제 기법(Regulations) 개발 : 가중치 축소, 드롭아웃, 조기 멈춤, 데이터 확대, 앙상블 등
    - 여러가지 손실 함수와 최적화 알고리즘 개발

# 1. 딥러닝(Deep Learning)

---

## ○ 딥러닝의 기술 혁신 사례

- 컨볼루션 신경망(CNN)

- 다층 퍼셉트론보다 매개변수가 훨씬 적지만 우수한 특징을 추출
- 1990년대 뉴욕 대학교의 르쿤 → 필기 숫자 인식에서 획기적인 성능 향상

- AlexNet

- 2012년 ILSVRC 대회(ImageNet 경진대회)에서 필기 숫자 인식의 오류율 15.3%
- 컴퓨터 비전 연구가 딥러닝으로 패러다임 전환

- 토론토 대학 한튼 교수 연구팀

- 2009년 경 음성 인식에 딥러닝 적용
- 안드로이드 음성 인식 소프트웨어의 인식 오류율을 25% 줄임

# 1. 딥러닝(Deep Learning)

## ○ 딥러닝 소프트웨어

이름	개발 그룹	최초 공개일	작성언어	인터페이스 언어	전이학습 지원	철저한 관리
씨아노(Theano)	몬트리올 대학교	2007년	python	python	O	X
카페(Caffe)	UC 버클리	2013년	C++	python, Matlab, C++	O	X
텐서플로 (TensorFlow)	구글 브레인	2015년	C++, python, CUDA	python, C++, Java, Javascript, R, Julia, Swift, Go	O	O
케라스(Keras) (François Chollet)	프랑소와 솔레	2015년	python	python, R	O	O
파이토치(PyTorch)	페이스북	2016년	C++, python, CUDA	python, C++	O	O

# 2. 텐서

## ○ 텐서(Tensor)의 구조

- 딥러닝에서 사용하기 위한 다차원 데이터 배열

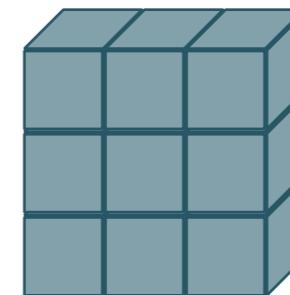


0차원 텐서(스칼라)



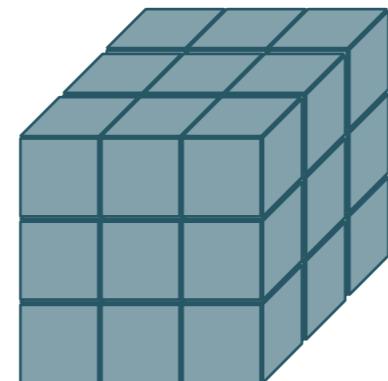
1차원 텐서(벡터)

$$\begin{pmatrix} 1.2 & -0.1 & 0.0 \\ 2.1 & 0.0 & 2.3 \\ 4.5 & -9.0 & 2.34 \end{pmatrix}$$



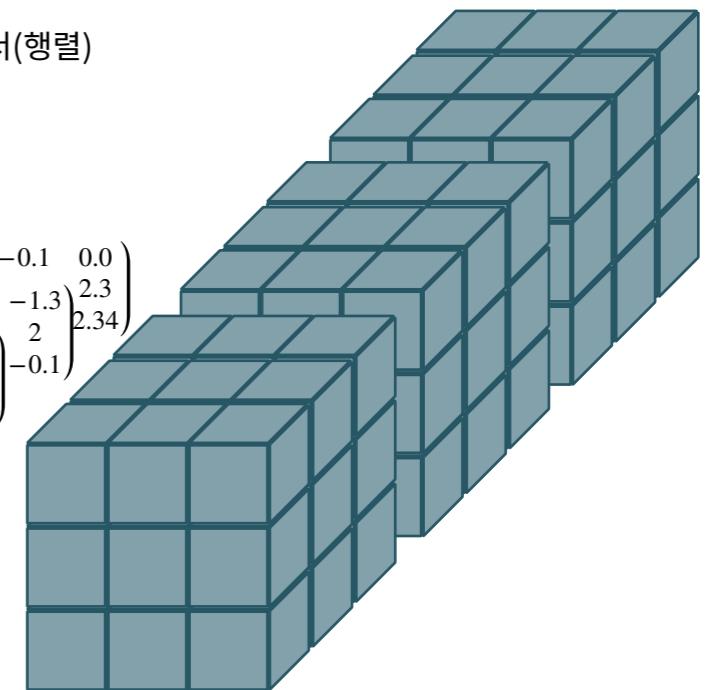
2차원 텐서(행렬)

$$\begin{pmatrix} (1.2 & -0.1 & 0.0) \\ (3 & 0 & -1.3) \\ (-4.1 & -1.2 & 0.01) \\ (0.3 & 1.8 & 2.3) \\ (1.5 & 2.1 & 2.9) \end{pmatrix}^2.34$$



3차원 텐서

$$\begin{pmatrix} (1.2 & -0.1 & 0.0) \\ (3 & 0 & -1.3) \\ (-4.1 & -1.2 & 0.01) \\ (0.3 & 1.8 & 2.3) \\ (1.5 & 2.1 & 2.9) \end{pmatrix}^2.34$$
  
$$\begin{pmatrix} (1.2 & -0.1 & 0.0) \\ (3 & 0 & -1.3) \\ (-4.1 & -1.2 & 0.01) \\ (0.3 & 1.8 & 2.3) \\ (1.5 & 2.1 & 2.9) \end{pmatrix}^2.34$$
  
$$\begin{pmatrix} (1.2 & -0.1 & 0.0) \\ (3 & 0 & -1.3) \\ (-4.1 & -1.2 & 0.01) \\ (0.3 & 1.8 & 2.3) \\ (1.5 & 2.1 & 2.9) \end{pmatrix}^2.34$$



4차원 텐서

## 2. 텐서

### ○ 텐서(Tensor)의 구조

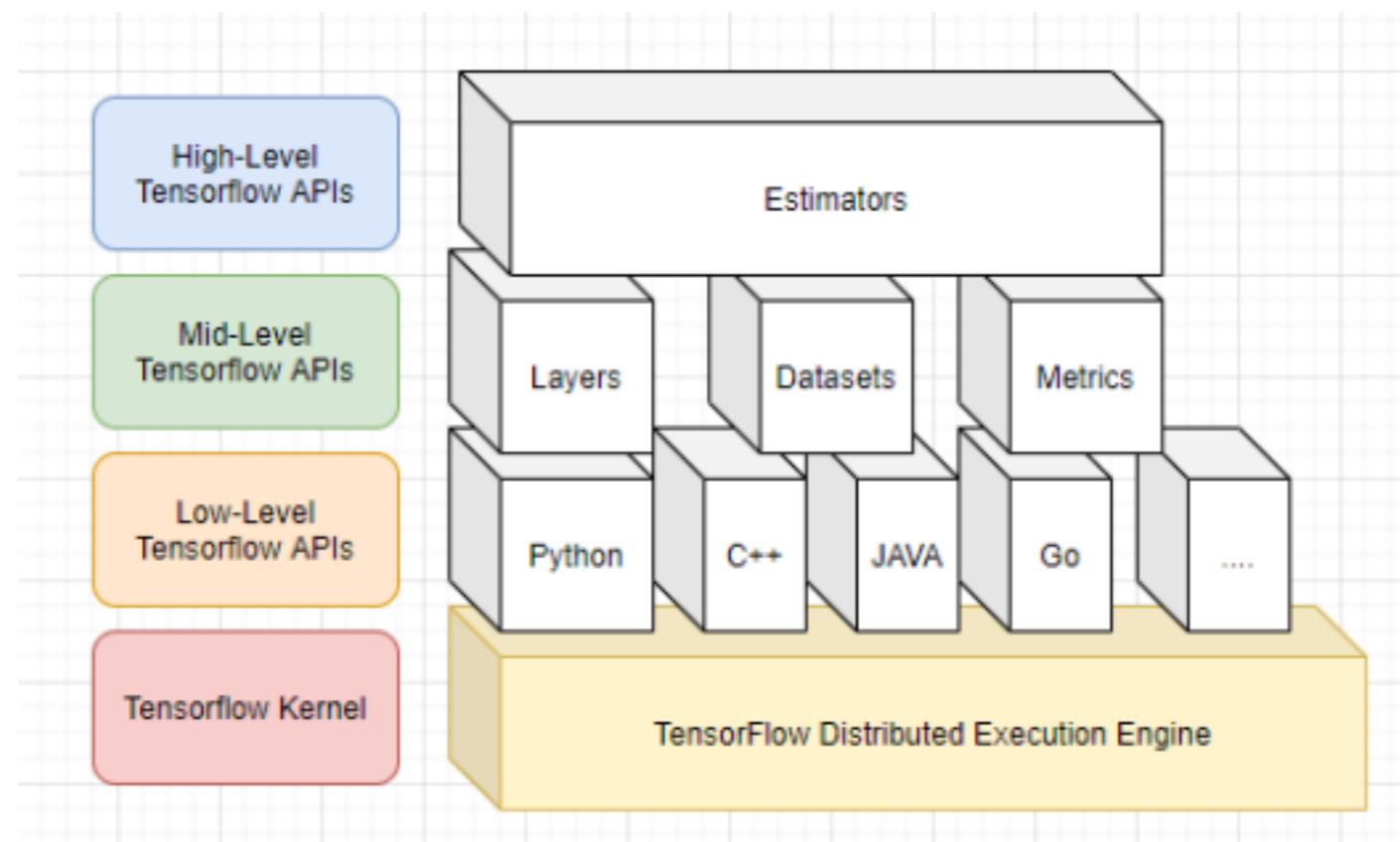
#### ■ 데이터 세트의 텐서 구조

데이터셋	내용	x_train	y_train
MNIST	60,000개 손글씨 이미지 28x28 픽셀, 흑백	(60000, 28, 28) 3D 텐서	(60000, ) 1D 텐서
CIFAR-10	비행기, 자동차, 새 등 10개 클래스 60,000개 이미지 32x32픽셀, RGB 컬러	(50000, 32, 32, 3) 4D 텐서	(50000, 1) 2D 텐서
Boston Housing	보스턴 근교 404개 주택의 방 수, 동네 범죄 율, 고속도로까지 거리 등 13개 속성을 기록	(404, 13) 2D 텐서	(404, ) 1D 텐서
Reuters	총 11, 258개 뉴스 기사를 46개 카테고리로 분류	(8982, ) 1D 텐서	(8982, ) 1D 텐서

# 3. 텐서 플로우

## ○ 텐서 플로우(Tensorflow)

- 구글이 만든 오픈소스 머신러닝 플랫폼(프레임워크)
  - 머신러닝과 딥러닝을 쉽게 사용할 수 있도록 다양한 기능을 제공
  - 데이터 플로우 그래프(Data Flow Graph) 구조를 사용
  - 기본적으로 C++ 구현, Python, Java, Go 등 다양한 언어를 지원



# 3. 텐서 플로우

## ○ 텐서 플로우 동작 확인



The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Tensorflow\_Test.ipynb
- File Menu:** 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말, 모든 변경사항이 저장됨
- Code Cell:** [ ]

```
[ ] import tensorflow as tf

{x}
print(tf.__version__)  # tensorflow 버전 출력
a = tf.random.uniform([2,3], 0, 1)  # [0,1] 사이 실수 난수 값을 갖는 2x3 행렬
print(a)
print(type(a))
```

# 3. 텐서 플로우

## ○ 텐서 플로우 동작 확인

```
▶ # 데이터셋의 텐서 구조  
import tensorflow.keras.datasets as ds  
import numpy as np  
  
# MNIST 읽고 텐서 모양 출력  
(x_train, y_train), (x_test, y_test) = ds.mnist.load_data()  
print('MNIST: ', x_train.shape, y_train.shape)  
  
# CIFAR-10 읽고 텐서 모양 출력  
(x_train, y_train), (x_test, y_test) = ds.cifar10.load_data()  
print('CIFAR-10: ', x_train.shape, y_train.shape)  
  
# Boston Housing 읽고 텐서 모양 출력  
(x_train, y_train), (x_test, y_test) = ds.boston_housing.load_data()  
print('Boston Housing: ', x_train.shape, y_train.shape)  
  
# Reuters 읽고 텐서 모양 출력  
(x_train, y_train), (x_test, y_test) = ds.reuters.load_data()  
print('Reuters: ', x_train.shape, y_train.shape)
```

### 3. 텐서 플로우

---

#### ○ 케라스(Keras)

- python 딥러닝 라이브러리
- 사용자가 쉽게 딥러닝을 구현할 수 있도록 해주는 인터페이스(API)
- Tensorflow, CNTK, Theano 등 다양한 딥러닝 라이브러리를 선택하여 사용할 수 있음
- Tensorflow2.0부터 Tensorflow와 keras 라이브러리가 통합됨
- 특징
  - 모델(Model) 중심 : 기본 모델(Sequential 모델 등)을 생성하고 레이어를 쌓아 모델을 생성
  - 모델 생성부터 모델 사용의 모든 과정에서 고유 API 제공
  - 모델 생성, 모델 평가 및 관리에 최적화

# 3. 텐서 플로우

---

## ○ 케라스(Keras)

### ■ 케라스(Keras) 작동 순서

0. 데이터셋 생성 : sklearn이나 Numpy등을 활용하여 생성
1. 모델 만들기 : 선형모델인 Sequential model을 기본으로 사용, layer를 추가하여 다양한 모델을 생성
2. 모델 학습 환경 설정 : compile() 함수 사용, optimizer(최적화함수), loss(손실함수), metric(분류시 기준) 설정
3. 모델 학습 : fit() 함수 사용, 학습시 학습 단위(epoch, batch\_size)나 검증 셋(validation)등을 설정
4. 학습과정 확인 : fit() 함수 사용시 히스토리 객체가 반환됨
  - loss : 매 epoch마다 훈련 손실값
  - acc : 매 epoch마다 훈련 정확도
  - val\_loss : 매 epoch마다 검증 손실값
  - val\_acc : 매 epoch마다 검증 정확도

# 3. 텐서 플로우

---

## ○ 케라스(Keras)

### ■ 케라스(Keras) 작동 순서

5. 모델 평가 : evaluate() 함수 사용, 테스트 셋으로 학습이 끝난 모델 평가

6. 모델 사용

- predict() : 모델 사용(예측)
- save() : 모델 저장
- load\_model() : 모델 불러오기

# 3. 텐서 플로우

## ○ 텐서 플로우로 MNIST 인식하기

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** Tensorflow\_MNIST.ipynb
- Toolbar:** 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말, 모든 변경사항이 저장됨
- Code Cell:** The cell contains the following Python code:

```
[ ] import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
```

### 3. 텐서 플로우

#### ○ 텐서 플로우로 MNIST 인식하기

```
▶ # MNIST 데이터를 읽어 신경망에 입력한 형태로 변환하기  
# 텐서 모양 변화 : 2D(28x28) -> 1D(784)  
(x_train, y_train), (x_test, y_test) = mnist.load_data()  
  
# [0, 1]사이 실수값으로 변환  
x_train = x_train.reshape(60000, 784)  
x_test = x_test.reshape(10000, 784)  
  
# 원핫 코드(onehot code)로 변환 : 클래스의 개수에 따라 하나의 출력노드만 1이 되게 함  
# 예) 0의 레이블 -> 1 0 0 0 0 0 0 0 0 0  
y_train = tf.keras.utils.to_categorical(y_train, 10)  
y_test = tf.keras.utils.to_categorical(y_test, 10)
```

### 3. 텐서 플로우

#### ○ 텐서 플로우로 MNIST 인식하기

```
▶ # 신경망 구조 설계
n_input = 784      # 입력 노드 수
n_hidden = 1024    # 은닉층 노드 수
n_output = 10       # 출력 노드 수

# 신경망 모델 구축 : Sequential 모델
mlp = Sequential()

# 은닉층(hidden layer) 추가
mlp.add(Dense(units=n_hidden, activation='sigmoid', input_shape=(n_input,)))

# 출력층(output layer) 추가
mlp.add(Dense(units=n_output, activation='sigmoid'))

mlp.summary()

# 신경망 학습 환경 설정 : 손실함수 = MSE, 최적화 함수 = Adam
mlp.compile(loss='mean_squared_error', optimizer=Adam(learning_rate=0.001),
            metrics=['accuracy'])
```

### 3. 텐서 플로우

---

#### ○ 텐서 플로우로 MNIST 인식하기

```
# 신경망 학습
hist = mlp.fit(x_train, y_train, batch_size=128, epochs=30,
                 validation_data=(x_test, y_test), verbose=2)

# 학습된 신경망 평가
res = mlp.evaluate(x_test, y_test, verbose=0)
print('정확률은 ', res[1]*100, '%입니다.')
```

### 3. 텐서 플로우

#### ○ 텐서 플로우로 MNIST 인식하기

```
▶ import matplotlib.pyplot as plt

# 정확률 곡선
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='best')
plt.grid()
plt.show()

# 손실함수 곡선
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='best')
plt.grid()
plt.show()
```

### 3. 텐서 플로우

# O fashion MNIST

- 28x28 픽셀 맵으로 이루어진 패션 아이템 이미지 데이터셋
  - 훈련집합 60,000개, 테스트 집합 10,000개 샘플로 구성



# 3. 텐서 플로우

## ○ fashion MNIST

The screenshot shows a Jupyter Notebook interface with a sidebar containing icons for file operations, search, and cell types. The main area displays two code cells. The first cell contains imports for numpy, tensorflow, tensorflow.keras.datasets, tensorflow.keras.models, tensorflow.keras.layers, and tensorflow.keras.optimizers. The second cell contains code to load the Fashion-MNIST dataset and reshape the input data from 2D (28x28) to 1D (784). A red dashed box highlights the 'fashion\_mnist' import in the first cell and the 'load\_data()' call in the second cell.

```
[5] import numpy as np
     import tensorflow as tf
     from tensorflow.keras.datasets import fashion_mnist
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Dense
     from tensorflow.keras.optimizers import Adam

[6] # MNIST 데이터를 읽어 신경망에 입력한 형태로 변환하기
     (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

# 텐서 모양 변화 : 2D(28x28) -> 1D(784)
     x_train = x_train.reshape(60000, 784)
     x_test = x_test.reshape(10000, 784)
```

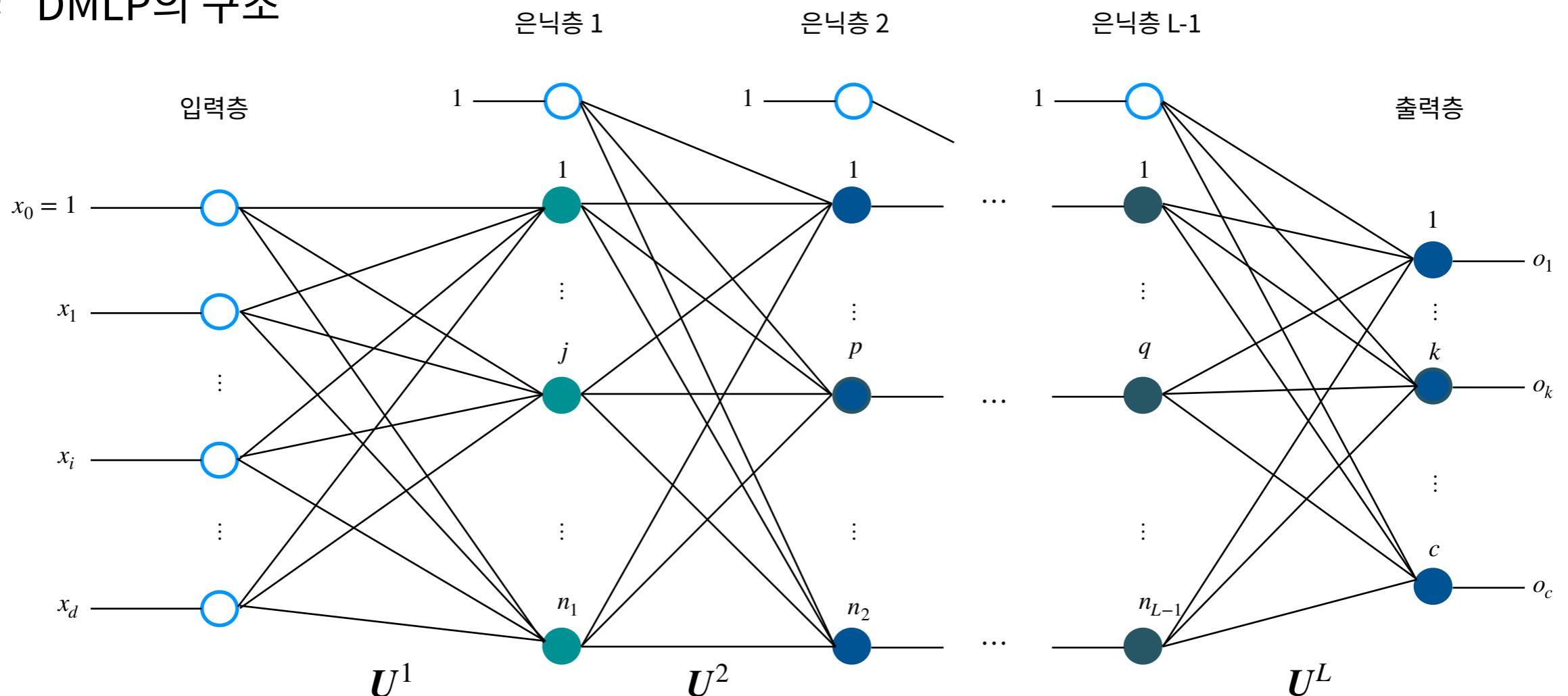
# 4. Deep MLP

## ○ 깊은 다층 퍼셉트론(DMLP)

- 많은 은닉층이 추가된 다층 퍼셉트론

- 가장 쉬운 딥러닝 모델

- DMLP의 구조



# 4. Deep MLP

## ○ 깊은 다층 퍼셉트론(DMLP)

The screenshot shows a Jupyter Notebook interface with the following details:

- Title:** DMLP\_MNIST.ipynb
- Toolbar:** 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말, 모든 변경사항이 저장됨
- Code Cells:**
  - [ ] import numpy as np  
import tensorflow as tf  
from tensorflow.keras.datasets import mnist  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
from tensorflow.keras.optimizers import Adam
  - [ ] # MNIST 데이터 읽기  
(x\_train, y\_train), (x\_test, y\_test) = mnist.load\_data()  
x\_train = x\_train.reshape(60000, 784)  
x\_test = x\_test.reshape(10000, 784)  
x\_train = x\_train.astype(np.float32)/255.0  
x\_test = x\_test.astype(np.float32)/255.0  
y\_train = tf.keras.utils.to\_categorical(y\_train, 10)  
y\_test = tf.keras.utils.to\_categorical(y\_test, 10)

# 4. Deep MLP

---

## ○ 깊은 다층 퍼셉트론(DMLP)

```
[ ] # 신경망 구조 설정 : 4개의 은닉층
n_input = 784
n_hidden1 = 1024
n_hidden2 = 512
n_hidden3 = 512
n_hidden4 = 512
n_output = 10

mlp = Sequential()
mlp.add(Dense(units=n_hidden1, activation='sigmoid', input_shape=(n_input,)))
mlp.add(Dense(units=n_hidden2, activation='sigmoid'))
mlp.add(Dense(units=n_hidden3, activation='sigmoid'))
mlp.add(Dense(units=n_hidden4, activation='sigmoid'))
mlp.add(Dense(units=n_output, activation='sigmoid'))
```

# 4. Deep MLP

## ○ 깊은 다층 퍼셉트론(DMLP)

```
[ ] # 신경망 학습
mlp.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
hist = mlp.fit(x_train, y_train, batch_size=128, epochs=30,
                validation_data=(x_test, y_test), verbose=2)

res = mlp.evaluate(x_test, y_test, verbose=0)
print('정확률은 ', res[1]*100, '%입니다.')
```

```
▶ import matplotlib.pyplot as plt

# 정확률 곡선
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='best')
plt.grid()
plt.show()
```

# 4. Deep MLP

---

## ○ 깊은 다층 퍼셉트론(DMLP)

```
# 손실함수 곡선
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='best')
plt.grid()
plt.show()
```

# 4. Deep MLP

---

## ○ models 클래스

- 모델 객체를 생성하고 레이어를 조직하는 방식을 지정
  - Sequential : 선형 모델
    - 입력층 → 은닉층 → 출력층 순서로 순차적으로 생성
    - add() : 레이어를 추가
    - compile() : 학습과정 조정
    - fit() : 학습 수행
    - evaluate() : 모델 성능 평가
  - functional API : 복잡한 구조의 그래프를 구축
    - 다중 입력과 다중 출력 모델을 설계

# 4. Deep MLP

---

## ○ layers 클래스

- 텐서 입력과 텐서 출력 계산 기능을 갖는 기본 빌딩 블록
  - 기본 레이어
    - Dense, Activation, Embedding, Masking, Lambda
  - 컨볼루션 레이어
    - Conv1D, Conv2D, Conv3D, SeparableConv1D, SeparableConv2D, DepthwiseConv2D, Conv2DTranspose, Conv3DTranspose
  - 풀링 레이어
    - MaxPooling1D/2D/3D, AveragePooling1D/2D/3D, GlobalMaxPooling1D/2D/3D, GlobalAveragePooling1D/2D/3D
  - Reshape 레이어
    - Reshape, Flatten, RepeatVector, Permute, Cropping1D/2D/3D, UpSampling1D/2D/3D, ZeroPadding1D/2D/3D

# 4. Deep MLP

---

## ○ 가중치 초기화 방법

### ■ Dense API

```
keras.layers.Dense(units, activation=None, use_bias=True,  
kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None,  
bias_regularizer=None, activity_regularizer=None, kernel_constraint=None,  
bias_constraint=None)
```

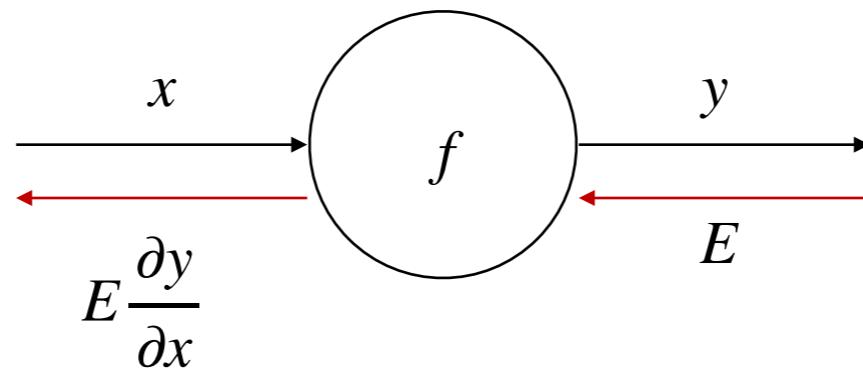
- units : 출력 노드의 수
- activation : 사용할 활성화 함수, 정하지 않으면 활성화가 적용되지 않음
- use\_bias : boolean, 레이어가 편향 벡터(바이어스)를 사용하는지 여부
- kernel\_initializer : 가중치 행렬의 초기값 설정 방법, default = glorot\_uniform(random\_uniform 보다 성능이 좋음)
- bias\_initializer : 편향 벡터의 초기값 설정기, default = zeros
- kernel\_regularizer : 가중치 행렬에 적용되는 정규화 함수
- bias\_regularizer : 편향 벡터에 적용되는 정규화 함수
- activity\_regularizer : 레이어의 output에 적용되는 정규화 함수
- kernel\_constraint : 가중치 행렬에 적용되는 제약 함수
- bias\_constraint : 편향 벡터에 적용하는 제약 함수

# 5. Deep MLP 전략

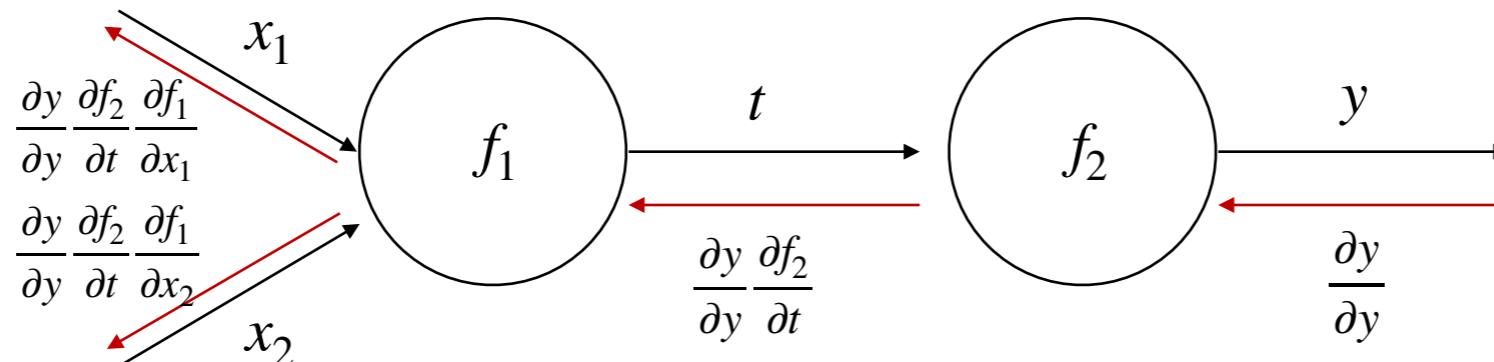
## ○ 그래디언트 소멸 문제

- 미분 이론의 연쇄 법칙(chain rule)

- 역전파는 순방향과 반대 방향으로 국소적 미분을 곱함



- 여러 함수로 구성된 합성함수의 미분은 각 함수의 미분의 곱으로 나타낼 수 있음



# 5. Deep MLP 전략

---

## ○ 그래디언트 소멸 문제

### ■ 그래디언트 소멸(Vanishing Gradient )

- 작은 미분값이 왼쪽으로 진행하면서 기하급수적으로 작아지는 현상
- 왼쪽으로 갈 수록 갱신이 매우 더디게 되어 신경망 학습이 매우 느려짐
- 해결 방법

### ○ 병렬 처리

- ◆ GPU를 이용하여 병렬 처리 → 하드웨어 비용, 병렬처리 프로그래밍
- ◆ 클라우드 GPU 또는 TPU 이용 → 구글 Colab의 TPU 사용

TPU(Tensor Processing Unit) : 구글이 신경망 학습을 빠르게 할 목적으로 개발한 기계 학습 전용 병렬처리 기계

### ○ ReLU 함수 사용

- ◆ 시그모이드 함수의 단점 : 활성 함수 매개변수 값이 클수록 그레이디언트 값이 매우 작음
- ◆ ReLU 함수 :  $s$  가 양수이면 그레이디언트 = 1, 음수이면 0 → 소멸이 발생할 가능성이 낮음

# 5. Deep MLP 전략

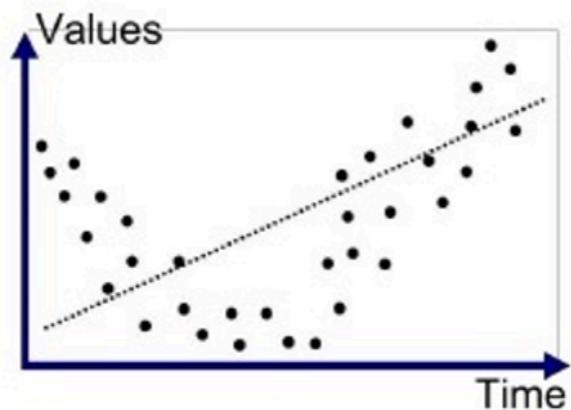
## ○ 과잉 적합과 과잉 적합 회피 전략

### ■ 과소 적합(underfitting)

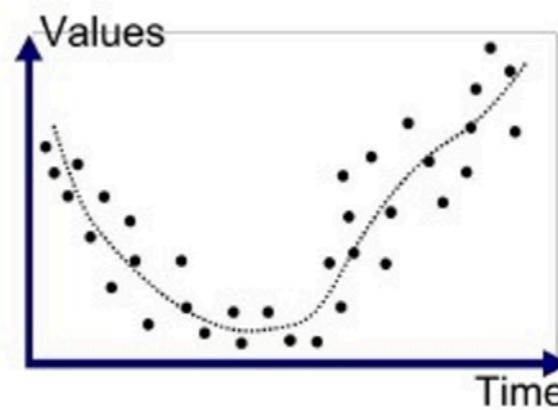
- 데이터에 비해 작은 용량(레이어, 적합함수 차수 등)의 모델을 사용해 오차가 많아지는 현상
- 해결 방법 : 모델 용량을 크게 하면 됨

### ■ 과잉 적합(overfitting)

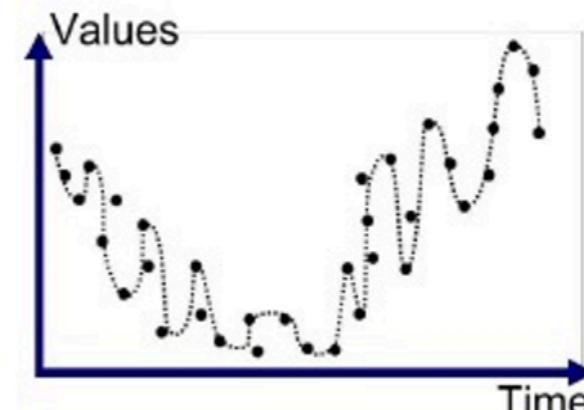
- 모델용량이 데이터 복잡도에 비해 너무 커서 일반화 현상을 잃는 현상
- 해결 방법 : 데이터 양을 늘리면 됨



Underfitted



Good Fit/Robust



Overfitted

# 6. Deep MLP를 위한 손실함수

---

## ○ 손실함수

- 손실 함수 : 예측값이 레이블과 다른 정도(오차)로 학습 정도를 측정하는 함수
- 평균제곱오차(MSE : mean squared error)
  - 미니 배치 단위로 제곱 오차를 구하여 평균을 취한 값
    - 제곱 오차
    - 통계학에서 오래전부터 사용된 대표적인 손실함수
    - 오차가 커도 그레이디언트가 더 작은 상황이 발생할 수 있음
- 교차 엔트로피(Cross Entropy)
  - 엔트로피 : 확률 분포의 무작위성(불확실성)을 측정하는 함수

$$H(x) = - \sum_{i=1,k} P(e_i) \log P(e_i)$$

x : {e<sub>1</sub>, e<sub>2</sub>, .., e<sub>k</sub>}의 요소값을 갖는 확률 변수, k : 서로 다른 사건의 개수, e<sub>i</sub> : i 번째 사건

# 6. Deep MLP를 위한 손실함수

---

## ○ 손실함수

### ■ 교차 엔트로피(Cross Entropy)

- 교차 엔트로피 : 두 확률 분포가 다른 정도를 측정

○ 두 확률 분포 P와 Q가 같은 확률 분포를 갖는 경우 교차 엔트로피가 작음

$$H(P, Q) = - \sum_{i=1,k} P(e_i) \log Q(e_i)$$

- 딥 러닝에서 사용하는 교차 엔트로피 손실함수

$$e = - \sum_{i=1,c} y_i \log o_i \quad c : \text{부류(class) 개수}$$

# 6. Deep MLP를 위한 손실함수

---

## ○ 손실함수

- 텐서플로에서 제공하는 손실함수

- Probabilistic losses : 분류에 사용되는 손실함수

- Binary Crossentropy , Categorical Crossentropy, Sparse Categorical Corssentropy 등 10가지

- Regression losses : 회귀타입에 사용되는 손실함수

- Mean Squared Error(MSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error(MAPE), Mean Squared Logarithmic Error(MSLE) 등 14가지

- Hinge losses : 최대 마진 분류를 위한 힌지 손실함수

- Hinge, Squared Hinge 등 6가지

- <https://keras.io/losses> 참조

# 7. Deep MLP를 위한 최적화 함수

---

## ○ 최적화 함수(Optimizer)

- Optimizer : 손실함수의 최저점을 찾는 최적화 알고리즘
  - 신경망 학습 데이터는 잡음과 변화가 심해 SGD와 같은 옵티마이저에 한계 발생
  - 한계 극복 방법
    - 모멘텀(momentum)
      - ◆ 이전 운동량을 현재에 반영 → 이전 미니배치에서 얻은 방향정보를 고려해 잡음을 줄임
      - ◆ 고전적 SGD :  $w = w - \rho \frac{\partial J}{\partial w}$
      - ◆ 모멘텀을 적용한 SGD :  $\left. \begin{array}{l} v = \alpha v - \rho \frac{\partial J}{\partial w} \\ w = w + v \end{array} \right\} \alpha \in [0,1] \text{ } \alpha \text{가 } 1 \text{에 가까울 수록 gradient 정보에 큰 가중치를 줌}$

○ momentum에 따른 최적값 변화 : <https://distill.pub/2017/momentum>

○ 텐서플로에서 모멘텀 적용 : <https://keras.io/optimizers>

# 7. Deep MLP를 위한 최적화 함수

---

## ○ 최적화 함수(Optimizer)

### ■ Optimizer

- 한계 극복 방법

- 적응적 학습률(adaptive learning rate)

- ◆ 고전적 SGD : 작은 학습률을 사용해 조금씩 이동하는 보수적 정책을 사용

- 적응적 학습률을 적용한 옵티마이저

- ◆ Adagrad : 이전 그레이디언트를 누적한 정보를 이용

- ◆ RMSProp : 이전 그레이디언트 누적시 오래된 것의 영향을 줄이는 정책을 사용

- ◆ Adam : RMSProp에 모멘텀을 적용하여 개선

# 7. Deep MLP를 위한 최적화 함수

## ○ 옵티마이저 성능 비교 실험

CO Optimizers.ipynb ☆

파일 수정 보기 삽입 런타임 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

▶ import numpy as np  
import tensorflow as tf  
from tensorflow.keras.datasets import fashion\_mnist  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
from tensorflow.keras.optimizers import SGD, Adam, Adagrad, RMSprop

[ ] # fashion MNIST 데이터를 읽고 신경망 입력으로 변환  
(x\_train, y\_train), (x\_test, y\_test) = fashion\_mnist.load\_data()  
x\_train = x\_train.reshape(60000, 784)  
x\_test = x\_test.reshape(10000, 784)  
x\_train = x\_train.astype(np.float32)/255.0  
x\_test = x\_test.astype(np.float32)/255.0  
y\_train = tf.keras.utils.to\_categorical(y\_train, 10)  
y\_test = tf.keras.utils.to\_categorical(y\_test, 10)

# 7. Deep MLP를 위한 최적화 함수

## ○ 옵티마이저 성능 비교 실험

```
▶ # 신경망 구조 설정 : 4개의 은닉층  
n_input = 784  
n_hidden1 = 1024  
n_hidden2 = 512  
n_hidden3 = 512  
n_hidden4 = 512  
n_output = 10  
n_batch = 256  
n_epoch = 50  
  
# 모델 설계 함수  
def build_model():  
    model = Sequential()  
    model.add(Dense(units=n_hidden1, activation='relu', input_shape=(n_input,)))  
    model.add(Dense(units=n_hidden2, activation='relu'))  
    model.add(Dense(units=n_hidden3, activation='relu'))  
    model.add(Dense(units=n_hidden4, activation='relu'))  
    model.add(Dense(units=n_output, activation='softmax'))  
    return model
```

# 7. Deep MLP를 위한 최적화 함수

## ○ 옵티마이저 성능 비교 실험

```
▶ # SGD 옵티마이저를 사용하는 모델  
dmlp_sgd = build_model()  
dmlp_sgd.compile(loss='categorical_crossentropy', optimizer=SGD(),  
                  metrics=['accuracy'])  
hist_sgd = dmlp_sgd.fit(x_train, y_train, batch_size=n_batch, epochs=n_epoch,  
                        validation_data = (x_test, y_test), verbose=2)  
  
# Adam 옵티마이저를 사용하는 모델  
dmlp_adam = build_model()  
dmlp_adam.compile(loss='categorical_crossentropy', optimizer=Adam(),  
                   metrics=['accuracy'])  
hist_adam = dmlp_adam.fit(x_train, y_train, batch_size=n_batch, epochs=n_epoch,  
                           validation_data = (x_test, y_test), verbose=2)  
  
# Adagrad 옵티마이저를 사용하는 모델  
dmlp_adagrad = build_model()  
dmlp_adagrad.compile(loss='categorical_crossentropy',  
                      optimizer=Adagrad(), metrics=['accuracy'])  
hist_adagrad = dmlp_adagrad.fit(x_train, y_train, batch_size=n_batch,  
                                 epochs=n_epoch,  
                                 validation_data = (x_test, y_test), verbose=2)
```

# 7. Deep MLP를 위한 최적화 함수

## ○ 옵티마이저 성능 비교 실험

```
# RMSprop 옵티마이저를 사용하는 모델
dmlp_rmsprop = build_model()
dmlp_rmsprop.compile(loss='categorical_crossentropy',
                      optimizer=RMSprop(), metrics=['accuracy'])
hist_rmsprop = dmlp_rmsprop.fit(x_train, y_train, batch_size=n_batch,
                                 epochs=n_epoch,
                                 validation_data = (x_test, y_test), verbose=2)
```



```
# 모델의 정확도 측정
print('SGD의 정확도는 ',
      dmlp_sgd.evaluate(x_test, y_test, verbose=0)[1]*100, '%')
print('Adam의 정확도는 ',
      dmlp_adam.evaluate(x_test, y_test, verbose=0)[1]*100, '%')
print('Adagrad의 정확도는 ',
      dmlp_adagrad.evaluate(x_test, y_test, verbose=0)[1]*100, '%')
print('RMSprop의 정확도는 ',
      dmlp_rmsprop.evaluate(x_test, y_test, verbose=0)[1]*100, '%')
```

# 7. Deep MLP를 위한 최적화 함수

## ○ 옵티마이저 성능 비교 실험

```
▶ # 정확도 곡선
import matplotlib.pyplot as plt
plt.plot(hist_sgd.history['accuracy'], 'r')
plt.plot(hist_sgd.history['val_accuracy'], 'r--')
plt.plot(hist_adam.history['accuracy'], 'g')
plt.plot(hist_adam.history['val_accuracy'], 'g--')
plt.plot(hist_adagrad.history['accuracy'], 'b')
plt.plot(hist_adagrad.history['val_accuracy'], 'b--')
plt.plot(hist_rmsprop.history['accuracy'], 'm')
plt.plot(hist_rmsprop.history['val_accuracy'], 'm--')
plt.title('Model Accuracy Comparison between Optimizers')
plt.ylim(0.6, 1.0)
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train_sgd', 'Val_sgd', 'Train_adam', 'Val_adam', 'Train_adagrad',
           'Val_adagrad', 'Train_rmsprop', 'Val_rmsprop'])
plt.grid()
plt.show()
```

# 7. Deep MLP를 위한 최적화 함수

## ○ 런타임 유형 변경

Optimizers.ipynb ☆

파일 수정 보기 삽입 **런타임** 도구 도움말 모든 변경사항이 저장됨

+ 코드 + 텍스트

모두 실행	⌘/Ctrl+F9
이전 셀 실행	⌘/Ctrl+F8
초점이 맞춰진 셀 실행	⌘/Ctrl+Enter
선택항목 실행	⌘/Ctrl+Shift+Enter
이후 셀 실행	⌘/Ctrl+F10
실행 중단	⌘/Ctrl+M I
세션 다시 시작	⌘/Ctrl+M .
세션 다시 시작 및 모두 실행	
런타임 연결 해제 및 삭제	

**런타임 유형 변경**

런타임 유형

Python 3

하드웨어 가속기 [?](#)

CPU  T4 GPU  A100 GPU  L4 GPU  
 V100 GPU (deprecated)  TPU (deprecated)  
 TPU v2

프리미엄 GPU를 이용하시겠어요? [추가 컴퓨팅 단위 구매](#)

취소 저장

런타임 유형 변경

세션 관리

리소스 보기

런타임 로그 보기