PORTPOLIO

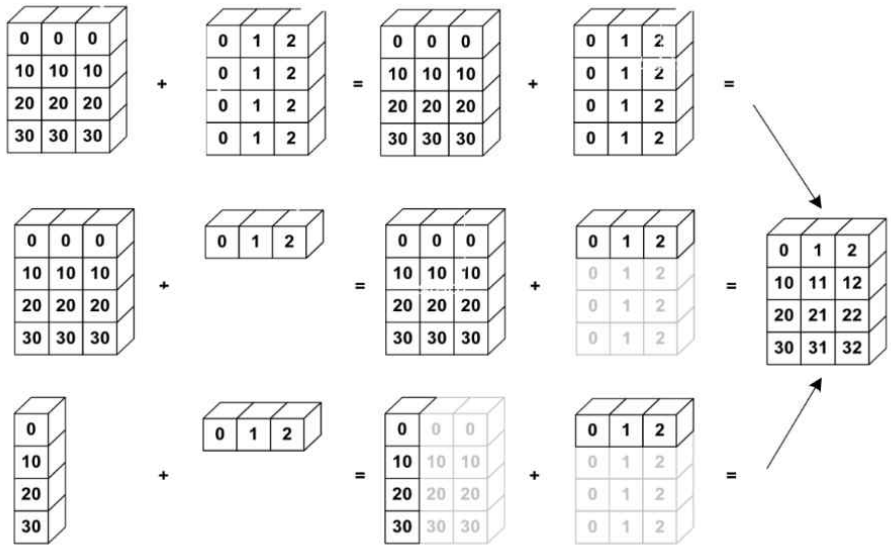
과목명	인공지능응용프로그래밍
학과	컴퓨터정보공학과
학번	20170666
이름	박승찬

목차

1. 텐서플로

● 브로드캐스팅

shape이 다르더라도 연산이 가능하도록 가지고 있는 값을 이용하여 shape을 맞추는 것



In [22]:
```
import numpy as np

print(np.arange(3))
print(np.ones((3, 3)))
print()

x = tf.constant((np.arange(3)))
y = tf.constant([5], dtype=tf.int64)
print(x)
print(y)
print(x+y)
```

```
[0 1 2]
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]

tf.Tensor([0 1 2], shape=(3,), dtype=int64)
tf.Tensor([5], shape=(1,), dtype=int64)
tf.Tensor([5 6 7], shape=(3,), dtype=int64)
```

In [26]:
```
x = tf.constant((np.arange(3)))
y = tf.constant([5], dtype=tf.int64)
print((x+y).numpy())

x = tf.constant((np.ones((3, 3))))
y = tf.constant(np.arange(3), dtype=tf.double)
print((x+y).numpy())

x = tf.constant(np.arange(3).reshape(3, 1))
y = tf.constant(np.arange(3))
print((x+y).numpy())
```

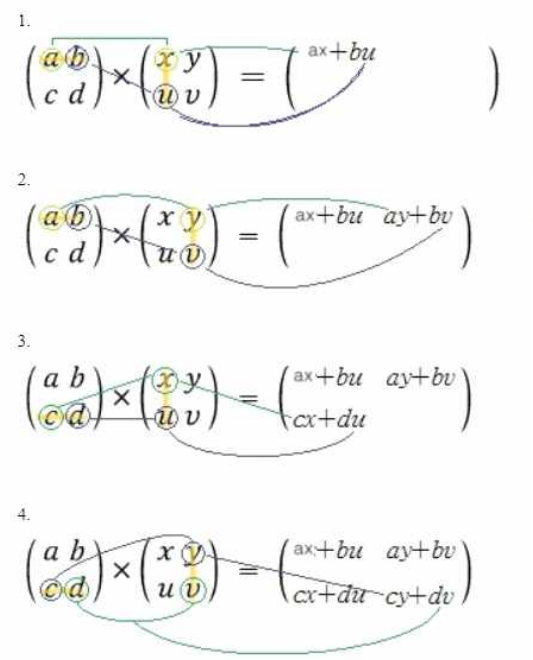
```
[5 6 7]
[[1. 2. 3.]
 [1. 2. 3.]
 [1. 2. 3.]]
[[0 1 2]
 [1 2 3]
 [2 3 4]]
```

- **텐서플로의 연산**

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \times \begin{pmatrix} x & y \\ u & v \end{pmatrix} = \begin{pmatrix} ? & ? \\ ? & ? \end{pmatrix}$$

첫 번째로 a와 x를 곱한 ax 와, 두 번째로 b와 u를 곱한 bu를 더한 값이 1행 1열의 값이다.

마찬가지로, a와 y를 곱한 ay와, b와 v를 곱한 bv를 더한 값은 1행 2열의 값이 된다. 이와 같은 방식으로 2행 1열과 2행 2열 까지 계산해주면 된다.




```
In [30]: # Matrix multiplications 1
matrix1 = tf.constant(([1., 2.], [3., 4.]))
matrix2 = tf.constant(([2., 0.], [1., 2.]))

gop = tf.matmul(matrix1, matrix2)
print(gop.numpy())

# Matrix multiplications 2
gop = tf.matmul(matrix2, matrix1)
print(gop.numpy())
```

```
[[ 4.  4.]
 [10.  8.]]
[[ 2.  4.]
 [ 7. 10.]]
```

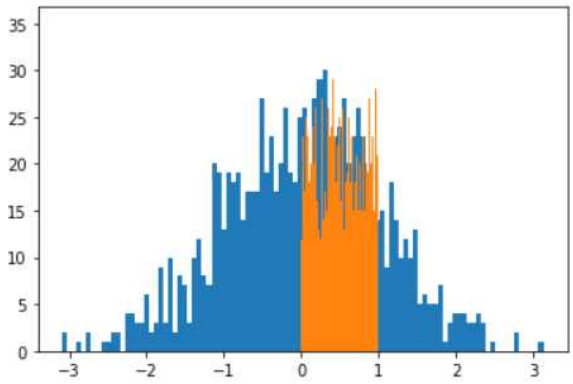
- **정규분포와 균등분포**

정규분포는 과거의 축적된 경험적 데이터를 이미 보유하고 있어 이를 이용하여 미래에 발생할 결과 값 x의 각 예상되는 범위별로 발생될 확률을 어느정도 추정할 수 있을 때 사용하고,

균등분포는 일반적으로 각 이벤트의 결과값을 알 수 없는 경우에 미래에 발생할 이벤트의 결과값이 각 예상되는 범위별로 균등한 확률로 일어날 것이라고 예상될 때 사용한다.

```
import matplotlib.pyplot as plt
rand1 = tf.random.normal([1000], 0, 1)
rand2 = tf.random.uniform([2000], 0, 1)
plt.hist(rand1, bins=100)
plt.hist(rand2, bins=100)
```

rand1 = 정규분포 (파란색)
rand2 = 균등분포 (주황색)



2. MNIST 프로그래밍

- **MNIST 데이터셋**

딥러닝 손글씨 인식에 사용되는 데이터셋

"필기 숫자 이미지" 와 정답인 "레이블" 의 쌍으로 구성

- **손글씨 하나의 구조**

28 * 28 즉, 784 픽셀로 이루어져 있으며, 내부 값은 0 ~ 255

- **딥러닝 구현 순서**

0. 필요 모듈 임포트
1. 훈련과 정답 데이터 지정
2. 모델 구성
3. 학습에 필요한 최적화 방법과 손실 함수 등 설정
4. 생성된 모델로 훈련 데이터 학습
5. 테스트 데이터로 성능 평가

- **데이터셋**

훈련용(train data set)과, 테스트용(test data set)이 있으며, x는 입력이나 문제를 나타내고 y는 정답과 레이블을 나타낸다.

- **모델**

딥러닝의 핵심 신경망으로, 여러 층으로 구성

완전연결층 : Dense()
1차원 배열로 평탄화 : Flatten()

- **학습 방법의 여러 요소들**

옵티 마이저 : 입력된 데이터와 손실 함수를 기반으로 모델을 업데이트하는 메커니즘 (경사하강법 : 내리막 경사 따라 가기)

손실 함수 : 훈련 데이터에서 신경망의 성능을 측청하는 방법, 모델이 옳은 방향으로 학습될 수 있도록 도와주는 기준 값 (Cross entropy, MSE)

- **딥러닝 훈련**

Epochs : 총 훈련 횟수, 훈련 데이터를 한번 모두 훈련시키는 것이 1에폭

- **드롭아웃**

훈련 단계보다 더 많은 유닛이 활성화되기 때문에 균형을 맞추기 위해 층의 출력 값을 드롭아웃 비율만큼 줄이는 방법
테스트 단계에서는 어떤 유닛도 드롭아웃하지 않고, 일반적으로 훈련단계에서 적용

[0.2, 0.5, 1.3, 0.8, 1.1] 이라는 벡터를 출력하는 층이 있다고 가정하고, 만약 드롭아웃하게 되면 [0, 0.5, 1.3, 0, 1.1] 이 됨

MNIST 딥러닝 전 소스

```
#########################################################
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 샘플 값을 정수(0~255)에서 부동소수(0~1)로 변환
x_train, x_test = x_train / 255.0, x_test / 255.0

# 층을 차례대로 쌓아 tf.keras.models.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

# 모델 요약 표시
model.summary()

# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 모델에 설정
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
              # metrics=['accuracy', 'mse'])

# 모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=5)

# 모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               100480
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.2946 - accuracy: 0.9143
Epoch 2/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.1413 - accuracy: 0.9575
Epoch 3/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.1072 - accuracy: 0.9670
Epoch 4/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0864 - accuracy: 0.9736
Epoch 5/5
1875/1875 [==============================] - 3s 1ms/step - loss: 0.0741 - accuracy: 0.9765
313/313 [==============================] - 0s 843us/step - loss: 0.0724 - accuracy: 0.9780
[0.07243845611810684, 0.9779999852180481]
```

3. 인공신경망

인공신경망은 두뇌의 신경세포인 뉴런이 연결된 형태를 모방한 모델이다. 생물학적인 뉴런과 같이 인공신경망 뉴런은 여러 입력 값을 받아서 일정 수준이 넘어서면 활성화되어 출력 값을 내보낸다.

- **활성화 함수**

뉴런의 출력 값을 정하는 함수
relu : 선형함수 y = x를 의미, 양수만을 사용하며 선형 함수를 정류하여 0 이하는 모두 0으로 한 함수이다.
sigmoid : s자 형태의 곡선을 의미

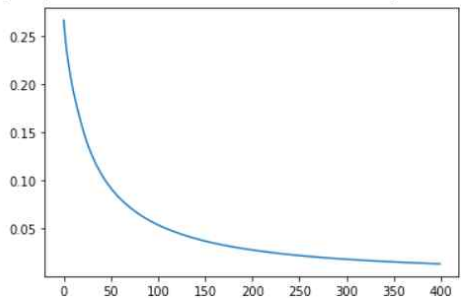
- **논리 게이트 신경망**

선형결합인 퍼셉트론은 배타적 논리합인 XOR 분류 문제를 해결할 수 없지만, 단일 퍼셉트론을 여러 개 쌓은 다중 퍼셉트론을 통해 XOR 분류 문제를 해결할 수 있다.

```
#tf,keras를 이용한 AND 네트워크 계산
import numpy as np
x = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
y = np.array([[1], [0], [0], [0]])

model = tf.keras.Sequential([
  tf.keras.layers.Dense(units=1, activation='sigmoid', input_shape=(2, )),
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr = 0.3), loss='mse')
model.summary()
```

```
# 3.34 2-레이어 AND 네트워크의 loss 변화를 선 그래프로 표시
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
```

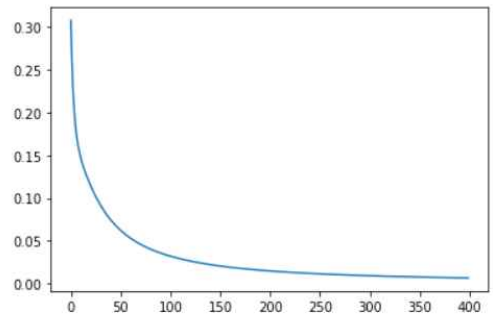
```
[<matplotlib.lines.Line2D at 0x7fb1003bdcf8>]
```

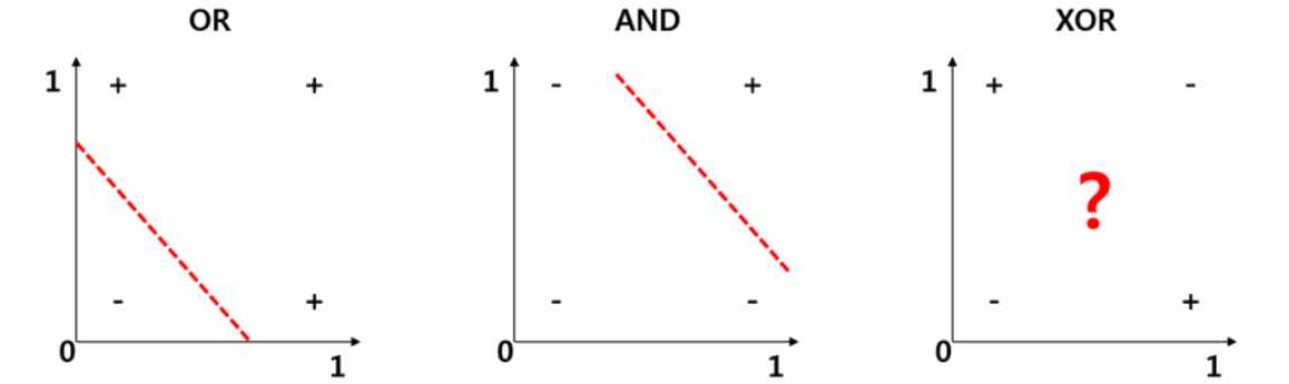



```
# tf.keras 를 이용한 OR 네트워크 계산
import numpy as np
x = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
y = np.array([[1], [1], [1], [0]])

model = tf.keras.Sequential([
  tf.keras.layers.Dense(units=1, activation='sigmoid', input_shape=(2,)),
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='mse')
model.summary()

history = model.fit(x, y, epochs=400, batch_size=1)
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7f09a20475f8>]
```





x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

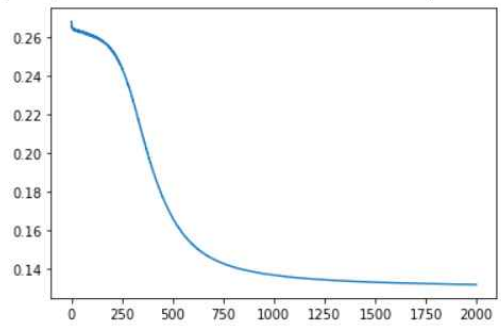
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

```python
# 3.27 tf.keras 를 이용한 XOR 네트워크 계산
import numpy as np
x = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
y = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([
  tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
  tf.keras.layers.Dense(units=1, activation='sigmoid')
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='mse')
model.summary()

# 3.28 tf.keras 를 이용한 XOR 네트워크 학습
history = model.fit(x, y, epochs=2000, batch_size=1)

# 3.28 tf.keras 를 이용한 XOR 네트워크 평가
print(model.predict(x))

# 3.30 XOR 네트워크의 가중치와 편향 확인
for weight in model.weights:
  print(weight)
```

```python
# 3.34 2-레이어 XOR 네트워크의 loss 변화를 선 그래프로 표시
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7fb10271c128>]
```



4. 회귀와 분류

분류는 미리 정의된, 가능성이 있는 여러 클래스 레이블 중 하나를 예측하는 것이다. 앞 장에서 붓꽃의 품종을 예측하는 것은 분류에 속한다.

회귀는 연속적인 숫자(실수)를 예측하는 것이다. 어떤 사람의 교육 수준, 나이 등을 이용해 연봉을 예측하는 것도 회귀 문제의 예이고, 몸무게를 이용해 키를 예측하는 것도 회귀 문제라고 볼 수 있다.

- **단순 선형 회귀 분석**

입력된 특징이 하나, 하나의 값을 출력 (ex: 키로 몸무게 추정)

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

- **다중 선형 회귀 분석**

입력된 특징이 여러 개, 하나의 값을 출력 (ex: 역세권, 아파트 평수, 주소로 아파트값을 추정)

$$y = W_1x_1 + W_2x_2 + \ldots W_nx_n + b$$

score(x)	result(y)
45	불합격
50	불합격
55	불합격
60	합격
65	합격
70	합격

- **로지스틱 회귀**

입력되는 값이 하나 또는 여러 개, 0 아니면 1 출력 (ex: 타이타닉의 승객 정보로 죽음을 추정)

- **선형회귀**

데이터의 경향성을 가장 잘 설명하는 하나의 직선을 예측하는 방법
y = wx + b (가중치 w와 편향인 b를 구하는 것)

```
import matplotlib.pyplot as plt
x_test = [1.2, 2.3, 3.4, 4.5, 6.0]
y_test = [2.4, 4.6, 6.8, 9.0, 12.0]

# 그래프 그리기
fig = plt.figure(figsize=(8,6))
plt.scatter(x_test, y_test, label='label')
plt.plot(x_test, y_test, 'y--')

x = [2.9, 3.5, 4.2, 5, 5.5, 6]
pred = model.predict(x)
plt.scatter(x, pred.flatten(), label='prediction')

plt.legend(loc='best')
plt.xlabel('x')
plt.ylabel('y')
```