지능형 컴퓨팅과정
포트폴리오

학과 : 컴퓨터정보공학과
학번 : 20164107
이름 : 박민성

● 목차

1. 인공지능과 딥러닝

1.1 앨런 튜링

- 1950년, 논문 <Computing machinery and intelligence>을 발표

- 생각하는 기계의 구현 가능성에 대한 내용, 인공지능 실험, '튜링 테스트'

- 텍스트로 주고받는 대화에서 기계가 사람인지 기계인지 구별할 수 없을 정도로 대화를 잘 이끌어 간다면, 이것은 기계가 "생각"하고 있다고 말할 충분한 근거가 된다 (지금의 챗봇)

- 인공지능의 처음 사용
  1956년 다트머스대 학술대회에서 세계최초의 AI프로그램인 논린 연산기를 발표

1.2 머신러닝과 딥러닝 비교

	기계학습	딥 러닝
데이터 의존성	중소형 데이터 세트에서 탁월한 성능	큰데이터 세트에서 뛰어난 성능
하드웨어 의존성	저가형 머신에서 작업	GPU가있는 강력한 기계가 필요, DL은 상당한양의 행렬 곱셈을 수행
기능 공학	데이터를 나타내는 기능을 이해해야함	데이터를 나타내는 최고의 기능을 이해할 필요가 없다
실행 시간	몇 분에서 몇시간	최대 몇 주. 신경망은 상당한 수의 가중치를 계산해야함

1.3 퍼셉트론

- 세계 최초의 인공신경망을 제안
  : 1957년 코넬대 교수, 심리학자인 프랭크 로젠블랫(Frank Rosenblatt)

- 신경망에서는 방대한 양의 데이터를 신경망으로 유입
  : 데이터를 정확하게 구분하도록 시스템을 학습시켜 원하는 결과를 얻어냄

1.4 다양한 분야에서 활용

- 인간과 대화하는 지능형 에이전트와 실시간 채팅이 가능한 챗봇(chatbot)
  : 음성인식과 자연어처리, 자동번역 등의 분야 애플의 시리, 삼성의 빅스비, IBM의

왓슨, 구글 나우, 마이크로소프트의 코타나, 아마존의 알렉사와 대시 등

- 얼굴을 비롯한 생체인식, 사물 인식, 자동자 번호판 인식 등 다양한 인식 분야
- X-ray 사진 판독과 각종 진단 등의 의료분야
- 드론의 자율비행이나 자동차의 자율주행 분야
- 주식이나 펀드, 환율, 일기예보 등의 예측 분야
- 음악의 작곡과 그림을 그리는 회화, 소설을 쓰는 분야 등에도 활용

2. 텐서플로

2.1 텐서플로

- 연구 및 프로덕션용 오픈소스 딥러닝 라이브러리
  : 딥러닝 프로그램을 쉽게 구현할 수 있도록 다양한 기능을 제공
  : 데스크톱, 모바일, 웹, 클라우드 개발용 API를 제공

- 구현 및 사용
  : Python, Java, Go 등 다양한 언어를 지원
  : 텐서플로 자체는 기본적으로 C++로 구현
  : 파이썬을 최우선으로 지원
  : 대부분의 편한 기능들이 파이썬 라이브러리로만 구현
  : Python에서 개발하는 것이 편함

2.2 코랩

- 코랩에서 쉽게 버전 사용방법
  : %tensorflow_version 1.x
  : %tensorflow_version 2.x

- Import 하기 전
  : 위 매직 명령어 사용
  : 사용 중에 바꾸려면 '런타임 다시시작' 후 바로
  %tensorflow_version 1.x
  %tensorflow_version 2.x

- 2.2 사용 중에 1.x으로 변경
  : 1. 메뉴, 런타임 | 런타임 다시 시작 단축키: ctrl+M .
  : 2. 바로 실행 %tensorflow_version 1.x

2.3 텐서플로 실행 및 사용

```
[1]  import tensorflow as tf
     tf.__version__
```

```
'2.3.0'
```

```
[2]  a = tf.constant(5)
     b = tf.constant(3)
     print(a+b)
```

```
tf.Tensor(8, shape=(), dtype=int32)
```

```
[3]  c = tf.constant('Hello, world!')
     print(c)
     print(c.numpy())
```

```
tf.Tensor(b'Hello, world!', shape=(), dtype=string)
b'Hello, world!'
```

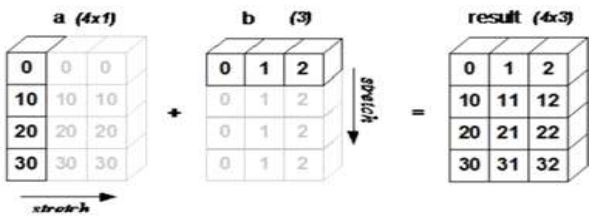
- **tf.cond()**
  : pred를 검사해 참이면 true_fc 반환 pred를 검사해 거짓이면 false_fc 반환

```
[6]  x = tf.constant(1.)
     bool = tf.constant(True)
     res = tf.cond(bool, lambda: tf.add(x, 1.), lambda: tf.add(x, 10.))

     print(res)
     print(res.numpy())
```

```
tf.Tensor(2.0, shape=(), dtype=float32)
2.0
```

- **브로드캐스팅**

  : Shape이 다르더라도 연산이 가능하도록 가지고 있는 값을 이용하여 Shape을 맞춤



```
[17] x = tf.constant([[0], [10], [20], [30]])
     y = tf.constant([0, 1, 2])

     print((x+y).numpy())
```

```
[[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```

```
[44] 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)
```

- **tf.rank 행렬의 차수 반환**

```
[28] my_image = tf.zeros([2, 5, 5, 3])
     my_image.shape
```

```
TensorShape([2, 5, 5, 3])
```

```
[19] tf.rank(my_image)
```

```
<tf.Tensor: shape=(), dtype=int32, numpy=4>
```

- **tf.cast tf.Tensor의 자료형의 다른 것으로 변경**

```
[28]  my_image = tf.zeros([2, 5, 5, 3])
      my_image.shape

      TensorShape([2, 5, 5, 3])
```

```
[19]  tf.rank(my_image)

      <tf.Tensor: shape=(), dtype=int32, numpy=4>
```

2.4 변수 Variable

- 변수로 사용
  : 텐서플로 그래프에서 tf.Variable의 값을 사용하려면 이를 단순히 tf.Tensor로 취급

- 메소드 assign, assign_add
  : 값을 변수에 할당

- 메소드 read_value
  : 현재 변수값 읽기

2.5 텐서플로 난수

- tf.random.uniform([1], 0, 1)
  : 배열, [시작, 끝]

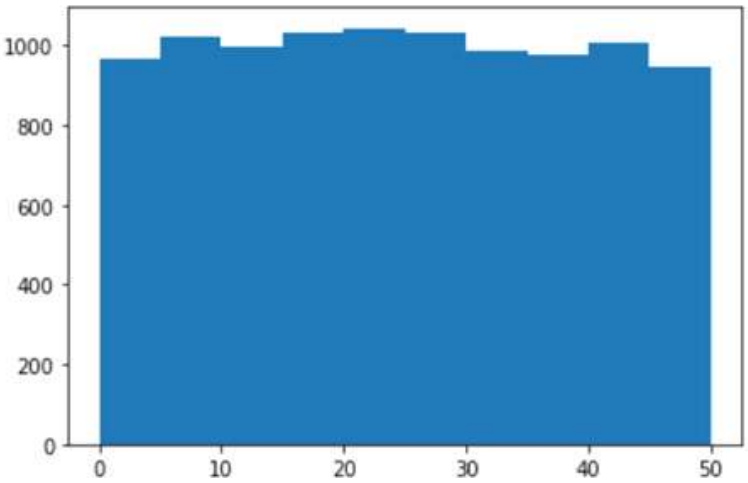
```
[6]   1 # 3.7 랜덤한 수 얻기 (균일 분포)
      2 rand = tf.random.uniform([1],0,1)
      3 print(rand)

      tf.Tensor([0.5543064], shape=(1,), dtype=float32)
```

```
[8]   1 rand = tf.random.uniform([5, 4],0,1)
      2 print(rand)

      tf.Tensor(
      [[0.43681145 0.84187937 0.9562702  0.7846168 ]
       [0.6079582  0.95665395 0.9038415  0.19482386]
       [0.51012075 0.8609252  0.9433547  0.9636986 ]
       [0.2134043  0.9559026  0.5170028  0.4017253 ]
       [0.0141474  0.15949261 0.23697984 0.7221806 ]], shape=(5, 4), dtype=float32)
```

```
[11]  1 rand = tf.random.uniform([1000],0,10)
      2 print(rand[:10])

      tf.Tensor(
      [5.1413307 1.548909  8.911686  9.880335  5.5388713 5.6710424 6.80269
       1.9444573 7.549943  6.573516 ], shape=(10,), dtype=float32)
```

```
[14]  1 import matplotlib.pyplot as plt
      2 rand = tf.random.uniform([10000],0,50)
      3 plt.hist(rand, bins=10)
```

```
(array([ 965., 1020.,  994., 1032., 1043., 1030.,  987.,  976., 1008.,
         945.]),
 array([6.0796738e-04, 4.9998469e+00, 9.9990854e+00, 1.4998324e+01,
        1.9997562e+01, 2.4996801e+01, 2.9996040e+01, 3.4995281e+01,
        3.9994518e+01, 4.4993759e+01, 4.9992996e+01], dtype=float32),
 <a list of 10 Patch objects>)
```



- tf.random.normal([4], 0, 1)
  : 크기, 평균, 표준편차

```
[53]  1 # 3.9 랜덤한 수 여러 개 얻기 (정규 분포)
      2 rand = tf.random.normal([4],0,1)
      3 print(rand)
```

```
tf.Tensor([-0.5962639   0.47093895  1.9455601  -0.42773333], shape=(4,), dtype=float32)
```

```
[54]  1 # 3.9 랜덤한 수 여러 개 얻기 (정규 분포)
      2 rand = tf.random.normal([2, 4],0,2)
      3 print(rand)
```

```
tf.Tensor(
[[-2.145662    0.64699423  2.0760484  -1.4640687 ]
 [ 1.3588632  -0.9740333   1.4347676  -1.3747462 ]], shape=(2, 4), dtype=float32)
```

2.6 MNIST 데이터셋

- MNIST 데이터셋

  : 딥러닝 손글씨 인식에 사용되는 데이터셋

  : 미국 국립 표준 기술원(NIST)

: 손으로 쓴 자릿수에 대한 데이터 집합
  * Yann Lecun의 The MNIST DATABASE of handwritten numerics 웹 사이트에서 배포
    http://yann.lecun.com/exdb/mnist/

: "필기 숫자 이미지"와 정답인 " 레이블 " 의 쌍으로 구성
  * 숫자의 범위는 0에서 9까지, 총 10 개의 패턴을 의미

: 필기 숫자 이미지
  * 크기가 28 x 28 픽셀인 회색조 이미지

: Label
  * 이미지의 정답: 필기 숫자 이미지가 나타 내는 실제 숫자, 0에서 9

: 대표적인 두 가지의 데이터 가져오는 방법
  * tensorflow.keras.datasets.mnist
  * tensorflow.examples.tutorials.mnist

- 케라스 딥러닝 구현
: 5개 과정
  * 딥러닝을 모델을 만들기 define
  * 주요 훈련방법을 설정 compile
  * 훈련 fit
  * 테스트 데이터를 평가 evaluate
  * 정답을 예측 predict

- **딥러닝 구현 전 소스**

```
# mnist 모듈 준비
mnist = tf.keras.datasets.mnist

# MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
(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.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(.2),
tf.keras.layers.Dense(10, activation='softmax')
])

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

# 모델 요약 표시
model.summary()

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

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

- **예측이 잘못된 20개 그리기 소스**

```
from random import sample  import numpy as np

################################################
# 예측 틀린 것 첨자를 저장할 리스트
mispred = []
# 예측한 softmax의 확률이 있는 리스트 pred_result  pred_result =
model.predict(x_test)

# 실제 예측한 정답이 있는 리스트 pred_labels
pred_labels = np.argmax(pred_result, axis=1)

for n in range(0, len(y_test)):
if pred_labels[n] != y_test[n]:  mispred.append(n)
print('정답이 틀린 수', len(mispred))

# 랜덤하게 틀린 것 20개의 첨자 리스트 생성  samples = sample(mispred, 20)
print(samples)

# 틀린 것 20개 그리기
count = 0
nrows, ncols = 5, 4  plt.figure(figsize=(12,10))  for n in samples:
count += 1
plt.subplot(nrows, ncols, count)
plt.imshow(x_test[n].reshape(28, 28), cmap='Greys', interpolation='nearest')  tmp =
"Label:" + str(y_test[n]) + ", Prediction:" + str(pred_labels[n])  plt.title(tmp)

plt.tight_layout()  plt.show()
```

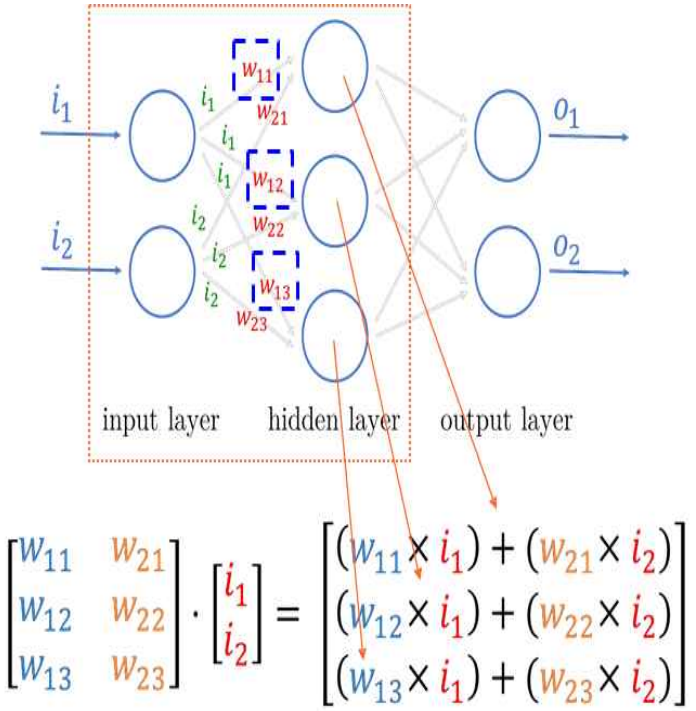
2.7 인공 신경망

- 뉴런
  : 입력, 편향

- 신경망
  : 뉴런의 연결

- 편향
  : 편향을 조정해 출력을 맞춤

- 입력 2개, 출력3개인 신경망 예시



$$\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} (w_{11} \times i_1) + (w_{21} \times i_2) \\ (w_{12} \times i_1) + (w_{22} \times i_2) \\ (w_{13} \times i_1) + (w_{23} \times i_2) \end{bmatrix}$$

- 논리 게이트 AND OR XOR 신경망
  : AND

```
# tf.keras 를 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()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```

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

```
4/4 [==============================] - 0s 1ms/step - loss: 0.0145
Epoch 372/400
4/4 [==============================] - 0s 2ms/step - loss: 0.0144
Epoch 373/400
4/4 [==============================] - 0s 2ms/step - loss: 0.0144
Epoch 374/400
```

: XOR 소스

```
import tensorflow as tf
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()

# XOR 네트워크 학습
history = model.fit(x, y, epochs=2000, batch_size=1)

# XOR 네트워크 평가
print(model.predict(x))

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

2.8 회귀와 분류

- 회귀 모델
  : 연속적인 값을 예측

- 분류모델
  : 불연속적인 값을 예측

- 회귀 분석(regression analysis)
  : 관찰된 연속형 변수들에 대해 두 변수 사이의 모형을 구한 뒤 적합도를 측정해 내는 분석 방법
  : 회귀분석은 시간에 따라 변화하는 데이터나 어떤 영향, 가설적 실험, 인과 관계의 모델링 등의 통계적 예측에 이용

- 회귀
  : 옛날 상태로 돌아가는 것을 의미
  : 영국의 유전학자 프랜시스 골턴은 "평균으로의 회귀(regression to the mean)"

- **선형 회귀 y = 2x 예측**

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

# 문제와 정답 데이터 지정
x_train = [1, 2, 3, 4]
y_train = [2, 4, 6, 8]
```

```
# 모델 구성(생성)
model = Sequential([
Dense(1, input_shape=(1, ), activation='linear')
#Dense(1, input_dim=1)
])

# 학습에 필요한 최적화 방법과 손실 함수 등 지정
# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수,
# Mean Absolute Error, Mean Squared Error
model.compile(optimizer='SGD', loss='mse',
metrics=['mae', 'mse'])

# 모델을 표시(시각화)
model.summary()

# 생성된 모델로 훈련 데이터 학습
model.fit(x_train, y_train, epochs=1000)

# 테스트 데이터로 성능 평가
x_test = [1.2, 2.3, 3.4, 4.5]
y_test = [2.4, 4.6, 6.8, 9.0]
print('정확도:', model.evaluate(x_test, y_test))

print(model.predict([3.5, 5, 5.5, 6]))
```

- **선형 회귀 y = 2x + 1 예측**

```
import tensorflow as tf
import numpy as np
# 훈련과  테스트 데이터
x = np.array([0, 1, 2, 3, 4])
y = np.array([1, 3, 5, 7, 9]) #y = x * 2 + 1

# 인공신경망 모델 사용
model = tf.keras.models.Sequential()

# 은닉계층 하나 추가
model.add(tf.keras.layers.Dense(1, input_shape=(1,)))

# 모델의 패라미터를 지정하고 모델 구조를 생성
model.compile('SGD', 'mse')

model.fit(x[:3], y[:3], epochs=1000, verbose=0)

# 테스트 자료의 결과를 출력
print('Targets(정답):', y[3:])
# 학습된 모델로 테스트 자료로 결과를 예측(model.predict)하여 출력
print('Predictions(예측):', model.predict(x[3:]).flatten())
```