

AIoT SerBot Prime X로 배우는

온디바이스 AI 프로그래밍

7. 인공지능

7.1 머신러닝

인공지능

- 인공지능은 지적 능력 수준에 따라 3가지로 분류 가능
 - ▣ 약인공지능, 강인공지능, 초인공지능
 - ▣ 약인공지능 : 특정 분야에서만 인간보다 우수한 능력을 보이는 인공지능
 - 바둑에 특화된 구글의 알파고, 머신러닝
 - ▣ 강인공지능 : 모든 분야에서 인간 수준의 지적 능력을 갖추고 있는 인공지능
 - 비대면 대화를 했을 때 인간과 구분이 힘든 수준의 인공지능
 - 영화 '아이언맨'의 자비스
 - ▣ 초인공지능 : 모든 분야에서 인간을 초월한 수준의 인공지능

머신러닝

□ 머신러닝 : 컴퓨터 시스템이 주어진 데이터를 학습하는 과정

▣ 지도 학습

- 훈련 데이터로부터 학습 모델을 유도할 때 학습 결과가 어떻게 출력되어야 하는지 알려줌
- 학습 모델이 출력한 결과와 비교하며 모델을 최적화하는 학습 방법

▣ 비지도 학습

- 훈련 데이터로부터 학습 모델을 유도할 때 목표값 없이 학습 모델 스스로 각 데이터의 특징을 추론
- 특징값의 편차에 따라 데이터를 군집화하는 학습 방법

▣ 강화 학습

- 보상이라는 개념을 이용
- 학습 모델이 이전 출력과 비교해 더 나은 결과를 출력할 때 보상을 주며 오차를 줄여나가는 학습 방법.

지도 학습

□ 지도 학습의 장점 :

- 짧은 학습 시간을 투자해 낮은 오차를 얻을 수 있음
- 데이터양 대비 감소하는 오차가 적은 편
 - 학습의 한계가 빨리 나타남
 - 복잡한 학습 모델일수록 필요한 데이터양은 급격히 증가
- 훈련에 필요한 데이터를 가공하는 과정이 필요
- 학습 데이터가 단순하고 목표값의 종류가 적은 문제에 적합
- 기본적인 지도 학습 기법 : 분류, 회귀

분류

□ 분류

- ▣ 입력과 처리 결과로 이루어진 훈련 데이터에서 분석할 수 있는 기준을 학습
- ▣ 학습된 기준에 따라 새로운 데이터를 어떤 종류로 구분할지 선택하는 기법
- ▣ 훈련 데이터
 - 값과 클래스 또는 라벨로 이루어져 있음
- ▣ 분류 모델
 - 값과 클래스를 학습해 새로운 값이 입력되면 어떤 클래스를 부여할지 선택

분류

□ 분류 기법

▣ 이진 분류

- 클래스가 True와 False로 이루어져 있는 모델로, 데이터를 두 가지로 분류
- 로지스틱 회귀를 이용해 구현하기도 함

▣ 다중 분류

- 클래스가 여러 개로 이루어져 있는 모델로, 데이터를 여러 클래스로 분류
- 소프트맥스 회귀를 이용해 구현하기도 함

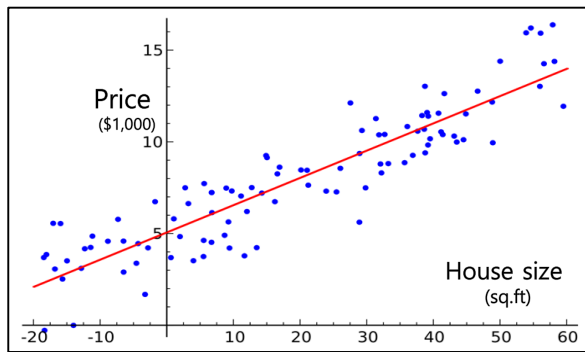
회귀

□ 회귀

- ▣ 입력과 처리 결과로 이루어진 훈련 데이터로부터 두 값의 관계를 학습
- ▣ 어떤 데이터가 새롭게 입력될지 예측하는 기법

선형 회귀

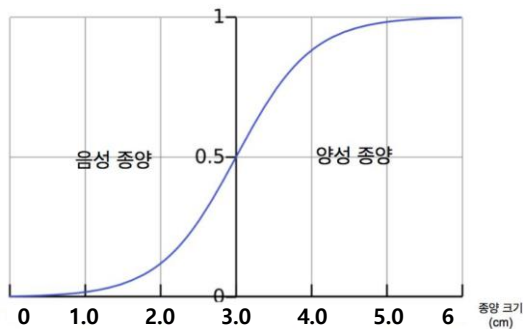
- ▣ 입력과 처리 결과를 선형적 관계로 모델링
- ▣ 관계식이 1차원 방정식
- ▣ 결괏값의 범위 : $-\infty \sim \infty$
- ▣ 단순히 입력이 어떤 결과를 가지는지에 대한 관계식을 갖음
- ▣ 대표적으로 함수 회귀 모델, 기후 예측 모델이 있음



집 면적 대비 가격에 대한 선형 회귀

로지스틱 회귀

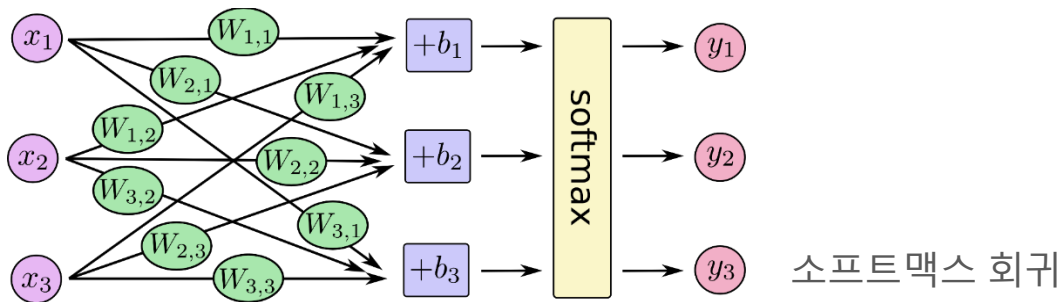
- ▣ 입력과 처리 결과를 이항 확률 관계로 모델링
- ▣ 관계식의 결과는 2개이며 값은 0과 1 사이
- ▣ 입력이 True(1)와 False(0) 중 어느 것일 확률이 높은지에 대한 관계식을 갖음
- ▣ 대표적으로 논리 회귀 모델, 시험 합격/불합격 예측 모델이 있음



종양의 크기에 따라 음성과 양성을 구분하는 로지스틱 회귀

소프트맥스 회귀

- 입력과 처리 결과를 다항 확률 관계로 모델링
- 관계식의 결과는 3개 이상이며 값은 0과 1 사이
- 입력이 3개 이상의 클래스 중 각 클래스일 확률에 대한 관계식을 가짐
- 대표적으로 품종 예측 모델이 있음



비지도 학습

- ▣ 학습 초기 오차가 비교적 높음
- ▣ 시간이 지날수록 오차가 급격히 줄어듦
- ▣ 학습 데이터를 가공할 필요 없이 많은 학습을 진행할 수 있는 장점이 있음
- ▣ 데이터 특징을 스스로 찾음
 - 설계자가 예측하지 못한 특징을 찾아낼 수도 있음
- ▣ 학습 시간이 긴 편이고 목표가 분명하지 않음
 - 원하지 않은 학습 모델이 나올 수 있음
- ▣ 주어진 학습 데이터가 복잡하거나 데이터 특징을 분석할 때 적합한 방법
- ▣ 기본적인 비지도 학습 기법 : 군집화

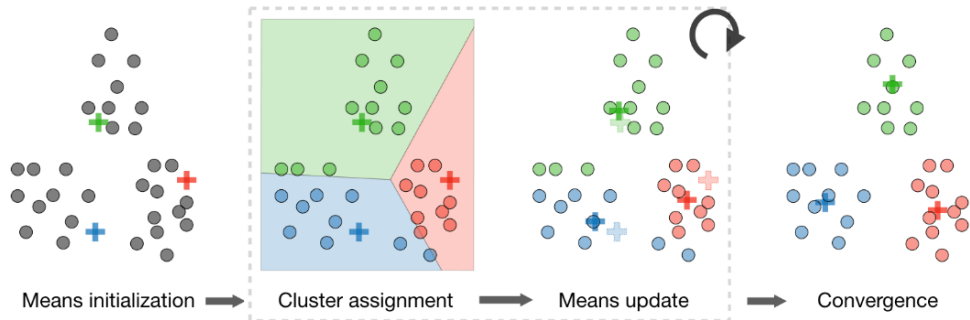
군집화

- ▣ 새로운 데이터에 대한 군집 기준을 만들어내는 기법
 - 입력으로만 이루어진 훈련데이터에서 비슷한 것끼리 군집시킴
- ▣ 군집 기준값과의 오차가 가난 낮은 군집에 소속시키고 군집 기준값을 조정
- ▣ 군집 기준값을 조정해나가며 최적의 값을 찾는 과정이 학습 과정
 - 군집 기준값을 이용해 새로운 데이터의 군집을 결정하는 과정이 응용 과정

군집화

▣ K 평균 군집화 (군집화 기법)

- 군집의 개수 K 개의 군집을 데이터 밀도가 높은 곳에 군집화
- 입력과 관계없이 초깃값을 랜덤으로 정함
- 원치 않는 결과가 나올 수 있으며 학습할 때마다 결과가 변함
- 군집마다 밀도 차이가 크거나 그 경계가 복잡하고 모호할수록 원하는 결과를 얻기 힘들



머신러닝 기법

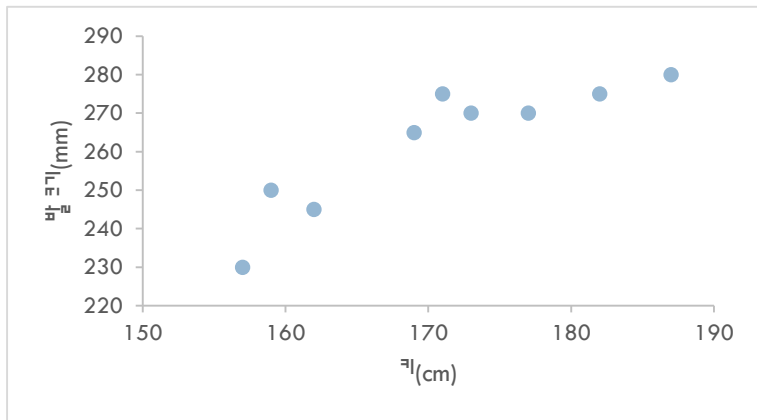
- 대표적인 머신러닝 기법

- ▣ 선형 회귀, 로지스틱 회귀, 소프트맥스 회귀, K-평균 군집화 등이 있음
- ▣ 이 기법들은 딥러닝의 기반이 되고 전처리 알고리즘으로 사용되기도 함

선형 회귀

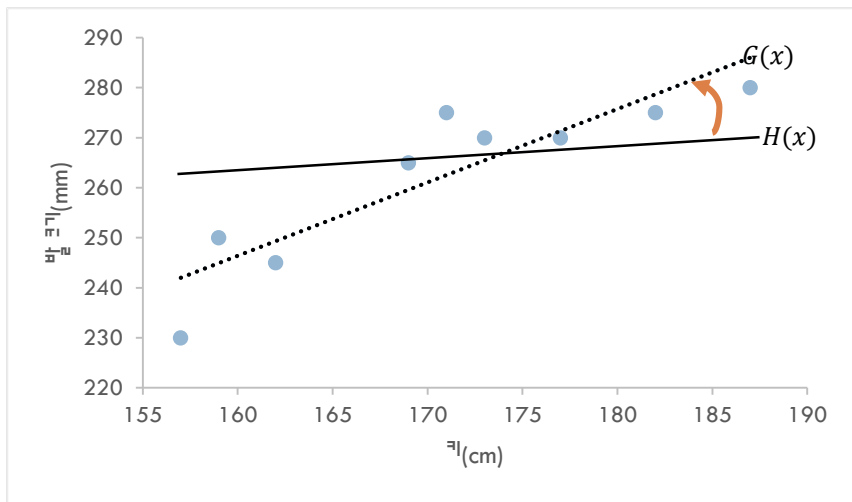
- 입력에 대한 결과가 선형 관계
 - ▣ 선형 관계 : 입력이 결과에 대해 직접적인 관계가 있을 때

키(cm)	발 크기(mm)
173	270
171	275
162	245
187	280
157	230
169	265
177	270
159	250
182	275



선형 회귀

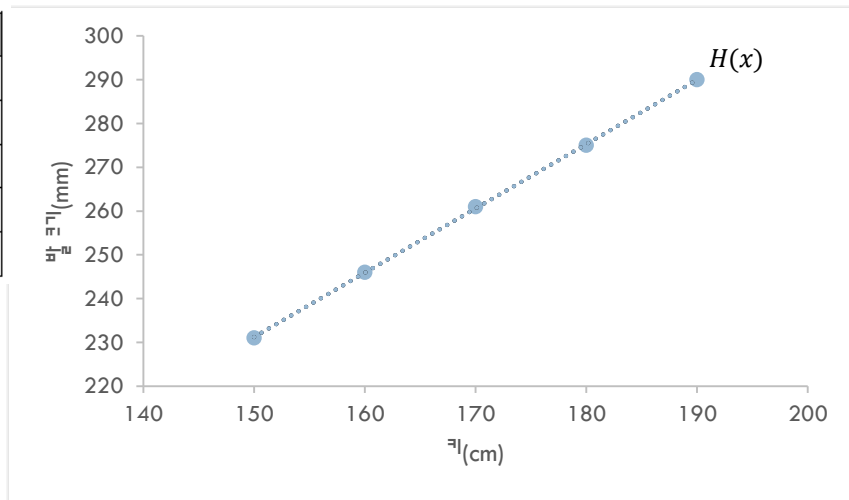
- ▣ 직선 $G(x)$: 모든 데이터의 관계를 하나의 직선을 표현
- ▣ 가설 $H(x)$: $G(x)$ 를 구하기 위해 선형 모델이 임의로 설정한 직선
- ▣ 최적화 : 선형 회귀 모델이 데이터와의 관계를 표현하는 가설 $H(x)$ 를 찾는 과정



선형 회귀

- 성공적으로 가설 최적화를 끝낸 선형 회귀 모델
 - 새로운 키 데이터가 주어졌을 때 발 크기가 몇일지 선형적으로 예측가능

키(cm)	발 크기(mm)
190	291
180	276
170	261
160	246
150	230



선형 회귀

□ Keras와 Pop.AI 라이브러리를 비교

Keras를 이용해 구현한 선형 회귀

```
01: from tensorflow import keras
02:
03: model = keras.models.Sequential()
04:
05: model.add(keras.layers.Input(shape=(1,)))
06: model.add(keras.layers.Dense(1))
07:
08: model.compile(loss='MSE', optimizer='SGD')
09:
10: model.fit(X, Y, epochs=100)
```

Pop.AI를 이용해 구현한 선형 회귀

```
01: from pop import AI
02:
03: LR = AI.Linear_Regression()
04:
05: LR.train()
06: LR.run()
```

선형 회귀

□ Keras와 Pop.AI 라이브러리를 비교

▣ Keras는 범용성을 위해 만들어진 라이브러리

- 선형 회귀를 구현하려면 학습 모델 설계, 손실 함수, 최적화 함수 등 고려해야 할 것이 많음

▣ Pop.AI 라이브러리는 특정 모델을 쉽게 사용할 수 있도록 사전 설계

- 입문자가 빠르게 실습 가능
- Pop.AI 라이브러리는 Keras를 기반으로 설계
- Keras를 이용한 학습 모델 설계, 손실 함수 등은 이후 챕터에서 진행

선형 회귀

□ pop.AI 라이브러리로 선형 회귀 구현

▣ Linear_Regression 객체

- restore: 최근 모델에 이어서 학습할지에 대한 여부를 Boolean으로 입력 (기본값: False)
- ckpt_name: 저장 및 불러올 모델 파일의 이름 (기본값: linear_regression)

▣ pop.AI라이브러리 import, Linear_Regression객체를 LR이라는 변수에 생성

```
01:         from pop import AI
02:
03:         LR = AI.Linear_Regression()
```

선형 회귀

▣ Linear_Regression 객체의 속성

- X_data : 입력 데이터
- Y_data : 입력에 대한 결괏값 데이터
- 입력 리스트와 결과 리스트는 1대1 대응

04: LR.X_data = [[173],[171],[162],[187],[157],[169],[177],[159],[182]]

05: LR.Y_data = [[270],[275],[245],[280],[230],[265],[270],[250],[275]]

선형 회귀

- ▣ Linear_Regression 객체의 train() 메소드 : 회귀 학습을 시작
 - 파라미터 : times와 print_every
 - times : 학습할 횟수 (기본값은 100)
 - print_every : 학습 상황을 몇 번째마다 출력할지를 의미 (기본값은 10)
 - train 메소드 : 실행하면 10회마다 회귀 모델의 오차 출력

06: LR.train()

선형 회귀

- ▣ Linear_Regression 객체의 run() 메소드 : 학습된 모델 사용
 - 파라미터 : inputs
 - 모델에 사용할 데이터
 - inputs에는 [[172], [162]]와 같이 2차원 리스트로 입력
 - 기본값은 X_data를 사용
 - run 메소드를 실행하면 입력에 대한 회귀 모델의 예측 발 크기를 출력

07: LR.run()

선형 회귀

□ 전체 코드

```
01:         from pop import AI
02:
03:         LR = AI.Linear_Regression()
04:
05:         LR.X_data = [[173],[171],[162],[187],[157],[169],[177],[159],[182]]
06:         LR.Y_data = [[270],[275],[245],[280],[230],[265],[270],[250],[275]]
07:
08:         LR.train()
09:         LR.run()
```

선형 회귀

- ▣ 추가 학습을 위해 train 메소드의 times 파라미터를 1000으로 설정하고 학습
- ▣ print_every 파라미터를 이용해 출력량을 조절가능

```
10: LR.train(times=1000, print_every=100)
```

선형 회귀

- ▣ 이전 학습 모델에 이어서 1,000회 학습해 총 1,100회를 학습
- ▣ 100회마다 학습 오차를 출력
- ▣ run 메소드를 이용해 학습 모델 예측값 출력

```
11:         LR.run()
```

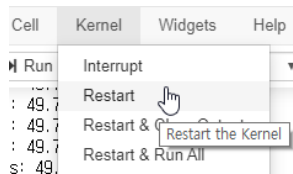
선형 회귀

- run 메소드의 파라미터로 새로운 데이터를 입력하여 출력 확인

```
12: LR.run([[150], [160], [170], [180], [190]])
```

선형 회귀

- 프로그램이 종료된 이후 학습 모델을 다시 불러와 사용하는 방법
 - ▣ Jupyter Notebook 상단 툴바에서 커널 재시작



Restart kernel?

Do you want to restart the current kernel? All variables will be lost.

Continue Running

Restart

선형 회귀

- AI 모듈을 import
- Linear_Regression 객체를 생성할 때 restore 파라미터를 True로 설정

```
01:         from pop import AI
02:
03:         LR = AI.Linear_Regression(restore=True)
```

- Linear_Regression 객체에 X_data와 Y_data를 입력
- run메소드 호출 -> 이전에 학습된 모델의 출력 결과 확인

```
04:         LR.X_data = [[173],[171],[162],[187],[157],[169],[177],[159],[182]]
05:         LR.Y_data = [[270],[275],[245],[280],[230],[265],[270],[250],[275]]
06:
07:         LR.run()
```

선형 회귀

- ▣ train 메소드를 호출하면 이전 학습 모델에 이어서 학습 가능

```
08:         LR.train()
```

- ▣ restore 파라미터를 설정하지 않거나 False로 설정한 경우

- 이전 학습 모델에 덮어 씌워지므로 주의
- ckpt_name 파라미터에 학습 모델을 구분할 수 있는 이름을 지정하여 개별 저장 가능

```
09:         LR = AI.Linear_Regression(restore=True, ckpt_name="model_1")
```

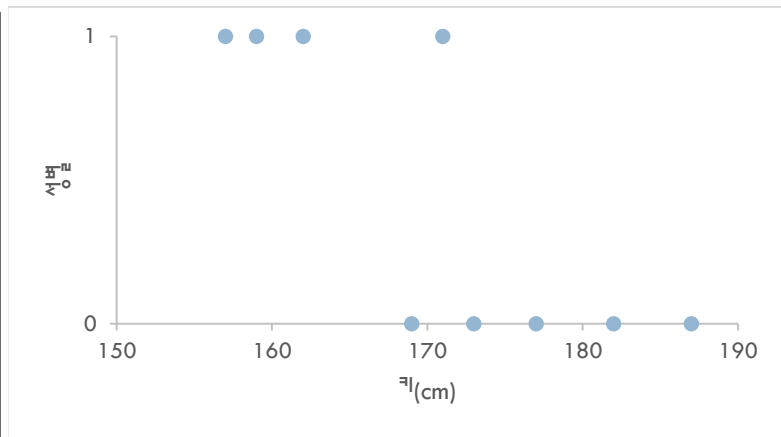
로지스틱 회귀

- ▣ 입력에 대해 이항 확률 관계
- ▣ 입력이 주어졌을 때 결과를 0~1 사잇값으로 표현
 - 입력이 True(1)일 확률을 의미

로지스틱 회귀

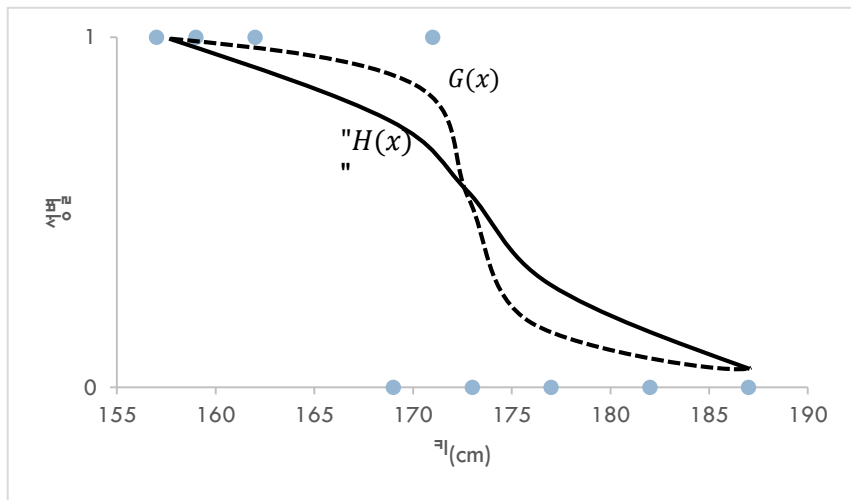
- 키에 따른 성별을 예측하기 위한 회귀 모델을 구하는 예
 - 키와 성별에 관한 데이터가 주어짐
 - 성별 값은 True(1)가 여성, False(0)가 남성일 경우로 설정

키(cm)	성별
173	0
171	1
162	1
187	0
157	1
169	0
177	0
159	1
182	0



로지스틱 회귀

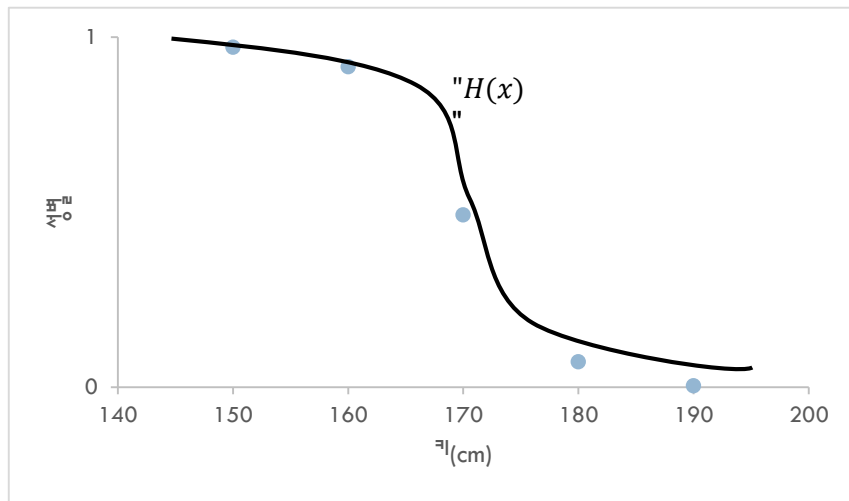
- $G(x)$: 모든 데이터의 관계를 하나의 선을 그어 표현
- $G(x)$ 를 구하기 위해 로지스틱 회귀 모델이 임의의 곡선을 가설 $H(x)$ 로 설정
- 로지스틱 회귀 모델은 이 가설 $H(x)$ 를 데이터의 관계를 가장 잘 표현하도록 최적화



로지스틱 회귀

- 최적화에 성공한 로지스틱 회귀 모델
 - 새로운 키 데이터가 주어졌을 때 여성일 확률을 예측 가능
 - 단, 여성보다 키가 작은 남성이 존재하므로 확률로써 예측

키(cm)	성별
190	0.005
180	0.073
170	0.493
160	0.916
150	0.972



로지스틱 회귀

□ 로지스틱 회귀 실습

- 키와 성별 데이터를 입력받아 로지스틱 회귀
- 새로운 키 데이터를 입력하면 성별을 예측하는 모델을 실습
- Pop.AI라이브러리 import, Logistic_Regression객체를 LR변수에 생성

```
01:         from pop import AI
02:
03:         LR = AI.Logistic_Regression()
```

로지스틱 회귀

- ▣ Logistic_Regression 객체 파라미터
 - input_size: 입력 데이터의 크기 (기본값: 1)
 - restore: 최근 모델에 이어서 학습할지에 대한 여부를 입력 (기본값: False)
 - ckpt_name: 저장 및 불러올 모델 파일의 이름 (기본값: logistic_regression)
- ▣ Logistic_Regression 객체 속성
 - X_data : 입력 데이터
 - Y_data : 입력에 대한 결괏값 데이터

04:	LR.X_data = [[173],[171],[162],[187],[157],[169],[177],[159],[182]]
05:	LR.Y_data = [[0],[1],[1],[0],[1],[0],[0],[1],[0]]

로지스틱 회귀

- ▣ Logistic_Regression 객체의 train() 메소드
 - 회귀 학습 시작
 - 파라미터 times : 기본값 100
 - 파라미터 print_every : 기본값 10
 - Train 메소드를 실행하면 10회마다 회귀 모델 오차 출력

06: LR.train()

로지스틱 회귀

- ▣ Logistic_Regression 객체의 run() 메소드

- 학습된 모델 사용 가능
- 파라미터 inputs : 기본값 X_data 사용
- run 메소드를 실행하면 입력에 대한 회귀 모델의 성별 확률을 출력

07: LR.run()

로지스틱 회귀

□ 전체 코드

```
01:         from pop import AI
02:
03:         LR = AI.Logistic_Regression()
04:
05:         LR.X_data = [[173],[171],[162],[187],[157],[169],[177],[159],[182]]
06:         LR.Y_data = [[0],[1],[1],[0],[1],[0],[0],[1],[0]]
07:
08:         LR.train()
09:         LR.run()
```

로지스틱 회귀

- 추가 학습 : train 메소드의 times 파라미터를 10000으로 설정하고 학습

```
10: LR.train(times=10000, print_every=1000)
```

- 이전 학습 모델에 이어서 10,000회 학습해 총 10,100회를 학습
- 1000회마다 학습 오차를 출력
- run 메소드를 이용해 학습 모델의 예측값을 출력

```
11: LogisticRegression.run()
```

로지스틱 회귀

- run 메소드의 파라미터로 새로운 데이터를 입력하여 출력 확인

```
12: LR.run([[150], [160], [170], [180], [190]])
```

- restore 파라미터를 True로 설정하면 최근 사용한 학습 모델을 불러와 다시 사용 가능

```
13: LR = AI.Logistic_Regression(restore=True, ckpt_name="model_1")
```

소프트맥스 회귀

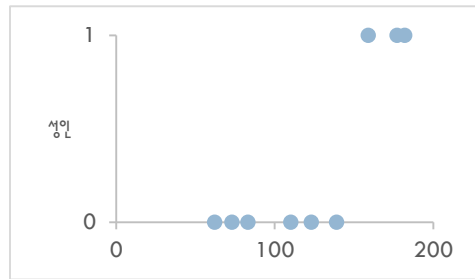
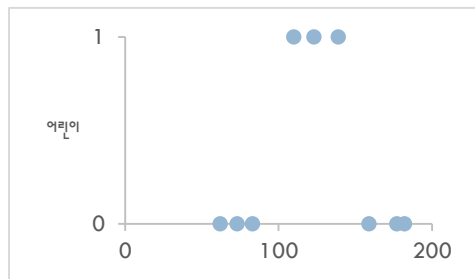
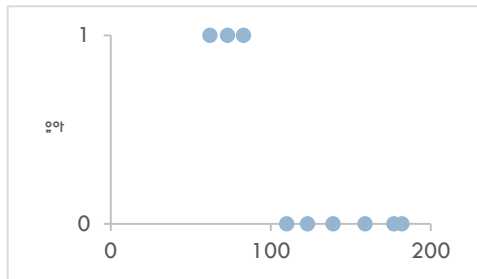
- 인공신경망ANN 개념 필요
 - 인공신경망은 딥러닝에서 설명
 - 현재 챕터에서는 다차원 방정식으로 대체하여 설명
- 소프트맥스 회귀는 입력에 대해 다항 확률 관계
- 입력이 주어졌을 때 각 클래스별로 0~1사잇값으로 표현
 - 모든 클래스 값의 합은 1

소프트맥스 회귀

- 키에 따른 연령층을 예측하기 위한 회귀 모델을 구하는 예제
 - 키와 연령층에 관한 데이터가 주어졌을 때
 - 각 연령층에 관한 클래스는 유아, 어린이, 성인으로 설정
 - 클래스의 개수 K 를 3으로 합니다.

키(cm)	유아	어린이	성인
73	1	0	0
62	1	0	0
83	1	0	0
110	0	1	0
139	0	1	0
123	0	1	0
177	0	0	1
159	0	0	1
182	0	0	1

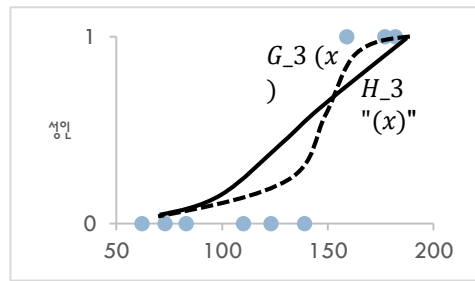
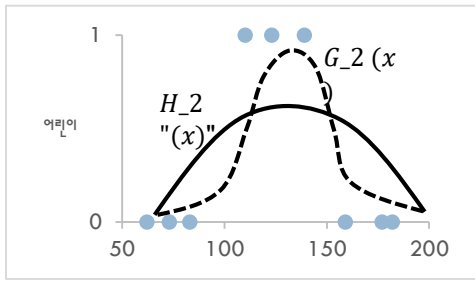
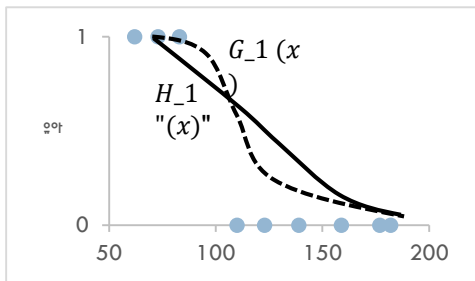
소프트맥스 회귀



소프트맥스 회귀

▣ $G_i(x)$: 모든 데이터와 클래스 간의 관계를 표현

- $G_i(x)$ 을 구하기 위해 소프트맥스 회귀 모델이 임의의 곡선들을 가설 $H_i(x)$ 로 설정
- 각 클래스에 대한 가설들 $H_i(x)$ 을 데이터의 관계를 가장 잘 표현하도록 최적화



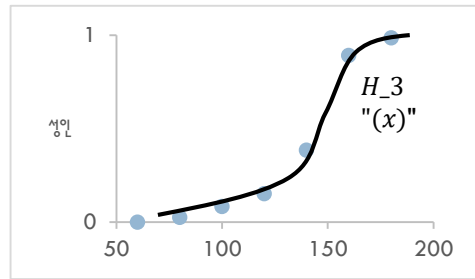
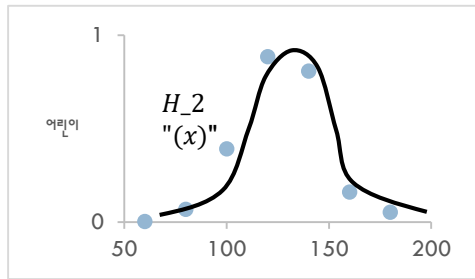
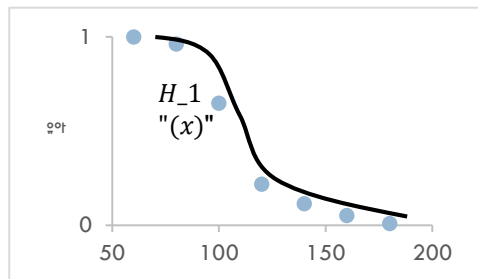
소프트맥스 회귀

□ 최적화에 성공한 소프트맥스 회귀 모델

- 새로운 키 데이터가 주어졌을 때 유아, 어린이, 성인일 확률 예측 가능
- 모든 클래스 값의 합이 1이어야 하므로 소프트맥스 함수를 이용해 각 클래스 값 조정

키(cm)	유아	어린이	성인
60	0.998	0.002	0
80	0.961	0.068	0.025
100	0.647	0.391	0.083
120	0.217	0.884	0.152
140	0.113	0.807	0.384
160	0.051	0.16	0.891
180	0.008	0.052	0.985

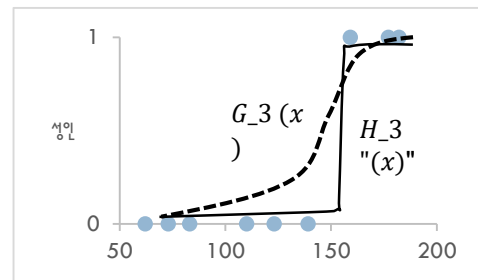
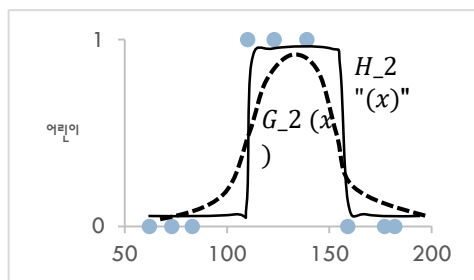
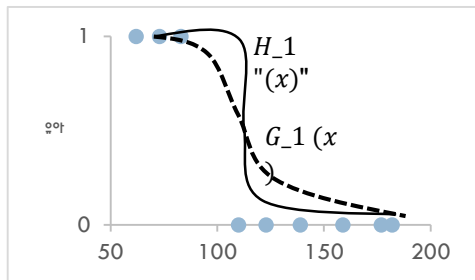
소프트맥스 회귀



소프트맥스 회귀

□ 과적합 (오버피팅)

- 과도하게 최적화 과정을 진행하면 극단적인 형태의 모델 $H_i(x)$ 이 생성
- 과도한 최적화로 인해 원하는 결과를 얻을 수 없는 상태



□ 과소적합 (언더피팅)

- 부족한 최적화로 인해 원하는 결과를 얻을 수 없는 상태

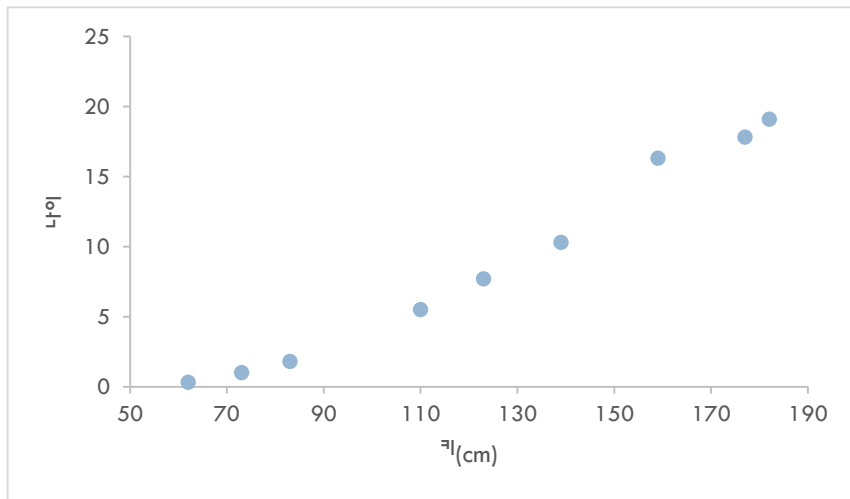
K-평균 군집화

- 예측과 최대화를 반복하며 최적해로 수렴하는 EM알고리즘을 기반
 - ▣ 1차 : 군집 기준점으로부터 가까운 데이터들을 묶음
 - ▣ 2차 : 묶인 데이터들의 중심점을 군집 기준점으로 재설정하는 과정 반복

K-평균 군집화

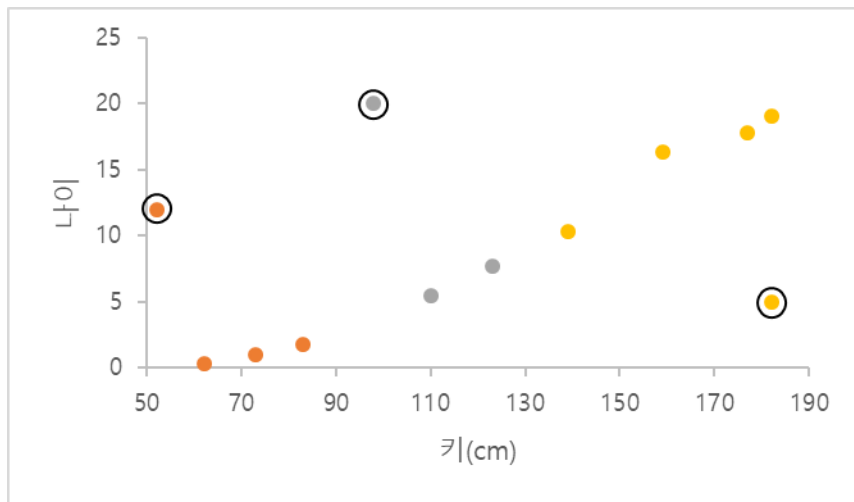
- 데이터들의 특징에 따라 K개의 군집을 구하는 예제
 - ▣ 키와 나이에 대한 데이터가 주어졌을 때

키(cm)	나이
73	1
62	0.3
83	1.8
110	5.5
139	10.3
123	7.7
177	17.8
159	16.3
182	19.1



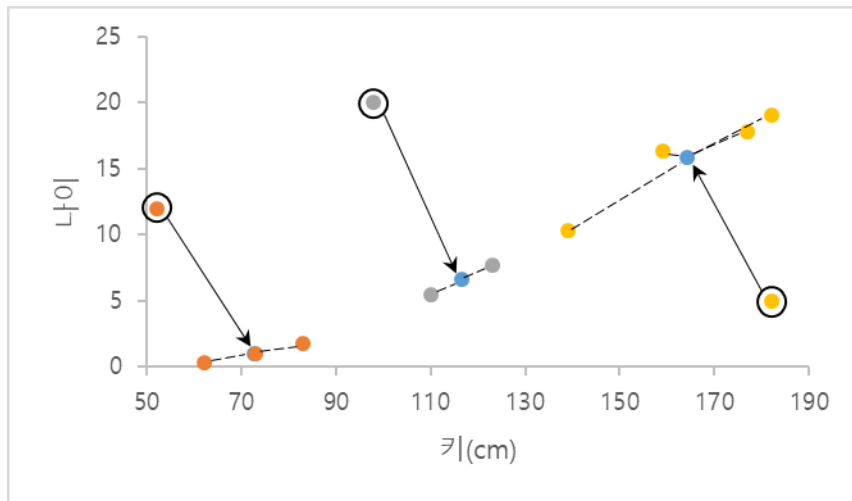
K-평균 군집화

- ▣ 랜덤으로 K개의 군집점 설정
- ▣ 각 군집점을 기준으로 가까운 데이터들을 묶음



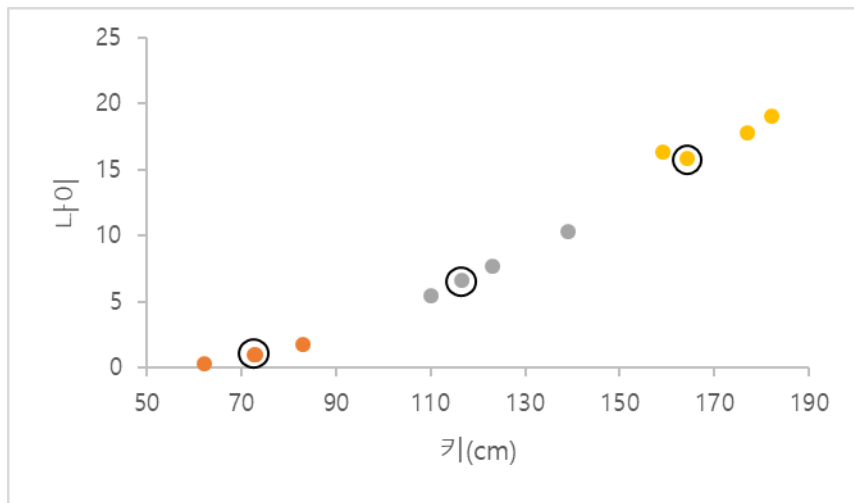
K-평균 군집화

- 묶인 데이터들의 중심점을 찾고 이 점을 군집점으로 재설정



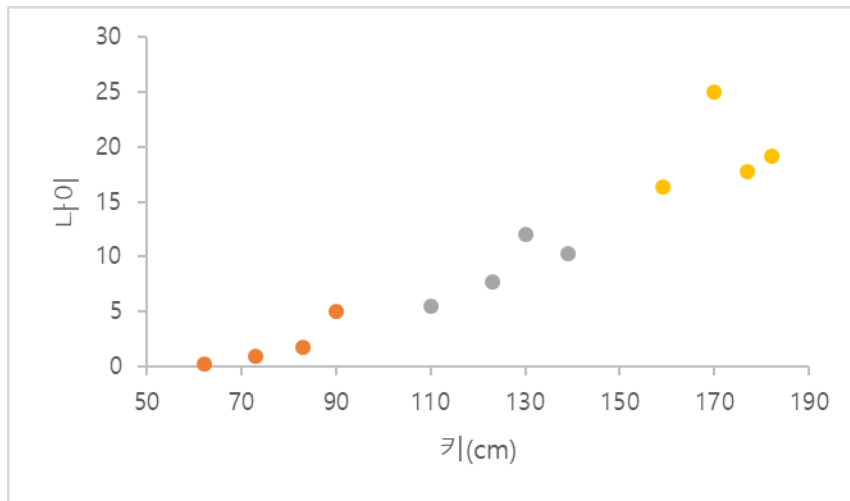
K-평균 군집화

- 이 과정을 군집점과 중심점의 오차가 최소화 될 때까지 반복 (최적화)



K-평균 군집화

- 최적화에 성공한 K-평균 군집화 모델
 - 새로운 키 데이터가 주어졌을 때 가장 가까운 군집에 포함
 - 필요에 따라 군집점을 재설정



내용 정리

□ 인공지능

- ▣ 약인공지능 : 특정 분야에서 우수한 능력을 가진 인공지능
- ▣ 강인공지능 : 모든 분야에서 인간과 비슷한 능력을 가진 인공지능
- ▣ 초인공지능 : 모든 분야에서 인간을 초월한 능력을 가진 인공지능

□ 머신러닝 : 컴퓨터 시스템이 데이터를 학습하는 과정

내용 정리

□ 지도 학습

- 컴퓨터가 출력한 결과와 비교하여 머신러닝 모델을 최적화하는 방법
 - 학습 결과가 어떻게 출력되어야 하는지 알려준 상태
- 분류 : 분류 기준을 학습하고 새로운 데이터를 어떤 결과로 분류할지 선택
 - 입력과 분류 결과가 주어진 상태
- 회귀 : 두 값의 수학적 관계를 학습하고 새로운 데이터의 결과의 출력 예측
 - 입력과 결과가 주어진 상태

내용 정리

□ 비지도 학습

- 컴퓨터가 입력 데이터의 특징을 추론하고 특징의 기준을 구체화하는 방법

- 입력 데이터만 주어진 상태

□ 군집화

- 입력 데이터 값이 비슷한 것끼리 군집시켜 군집 기준을 구체화
- 새로운 데이터에 대한 군집 그룹을 정하는 기법

□ 강화 학습

- 더 나은 결과를 출력할 때 보상을 주며 오차를 줄여나가는 학습 방법

내용 정리

□ 머신러닝 기법

- ▣ 선형 회귀 : 입력과 결과의 선형 관계를 분석하는 머신러닝 모델
- ▣ 로지스틱 회귀 : 입력과 결과에 대해 이항 관계를 분석하는 머신러닝 모델
- ▣ 소프트맥스 회귀 : 입력과 결과에 대해 다항 관계를 분석하는 머신러닝 모델
- ▣ K-평균 군집화 : 가까운 입력 데이터끼리 군집시키는 방법
 - 군집 기준을 추론하는 머신러닝 모델

연습문제

□ 문제 34. 다음 문장들을 읽고 빈 칸을 채워보세요.

- A. 특정 분야에서 우수한 능력을 가진 인공지능을 []라 한다.
- B. 컴퓨터 시스템이 주어진 데이터를 학습하는 과정을 []라 한다.
- C. 머신러닝은 학습 방식에 따라 []- []- []로 나눌 수 있다.
- D. 대표적 머신러닝 기법은 []- []- [] , []이 있다.

연습문제

- 문제 35. 다음 코드는 Pop.AI 라이브러리를 이용하여 선형 회귀를 구현한 코드입니다. 질문을 읽고 답해보세요.

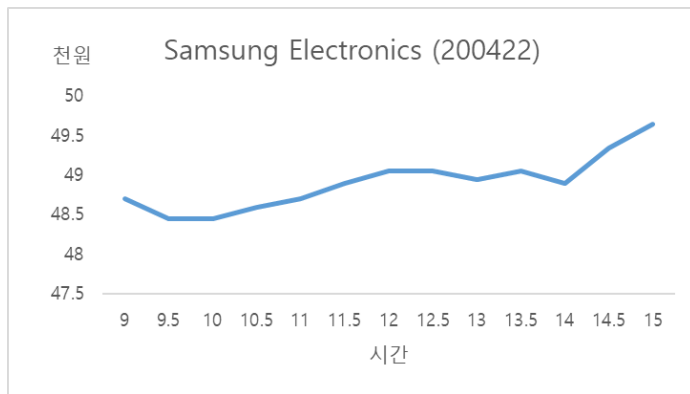
```
01:         from pop import AI
02:
03:         LR = AI.Linear_Regression()
04:
05:         LR.X_data = 
06:         LR.Y_data = 
07:
08:         LR.train(times=1000, print_every=100)
09:         LR.run()
```

연습문제

- ▣ A. 수식 $y=2x$ 를 회귀할 수 있도록 빈 칸 X, Y에 들어갈 학습 데이터셋을 작성해 보세요.
- ▣ B. A에서 작성한 데이터셋으로 손실율 0.001 이하까지 학습시키고, 100을 입력했을 때 출력을 작성하세요.

연습문제

- C. 선형회귀 모델을 새롭게 생성하고 다음과 같은 주가 그래프와 표가 주어졌을 때 선형 회귀하여 15시 30분 예측 값과 학습 횟수(Step)을 작성하세요.



9	9.5	10	10.5	11	11.5	12
48.7	48.45	48.45	48.6	48.7	48.9	49.05
12.5	13	13.5	14	14.5	15	
49.05	48.95	49.05	48.9	49.35	49.65	

연습문제

- 문제 36. 다음 코드는 Pop.AI 라이브러리를 이용하여 로지스틱 회귀를 구현한 코드입니다. 질문을 읽고 답해보세요.

```
01:         from pop import AI
02:
03:         LR = AI.Logistic_Regression()
04:
05:         LR.X_data = 
06:         LR.Y_data = 
07:
08:         LR.train()
09:         LR.run()
```

연습문제

- ▣ A. 양수면 1, 음수면 0을 출력하도록 회귀하고자 할 때 빈 칸 X, Y에 들어갈 학습 데이터셋을 작성하세요.
- ▣ B. 로지스틱 회귀 모델에 -1, 100, -0.01를 입력했을 때의 출력을 확인해보고, -0.01을 입력했을 때 오차가 0.1 이하인 모델을 만들어보고 모델의 손실율을 작성하세요.

연습문제

- 문제 37. 아래의 코드는 AloT SerBot의 Cds센서를 이용하여 주변 밝기를 저항값으로 출력하는 코드입니다. 조도계를 사용하거나, 스마트폰에서 '조도계' 애플리케이션을 다운 받아 다음 문제를 해결해보세요. (단, 조도계의 단위는 Lux로 합니다.)

```
01:         from pop import Al
02:         from pop import Cds, delay
03:
04:         cds = Cds(7)
05:
06:         value = cds.readAverage()
07:         print(value)
```

연습문제

- A. 조도계를 옆에 두고 코드를 실행시킨 후 조도계 값과 출력 값의 차이를 확인해보세요.
- B. 빈 배열을 생성하고, Cds 값을 0.5초 간격으로 10회 이상 추가하는 코드를 작성하세요.
- C. 다음 코드를 이용해 빈 배열 2개를 생성하고, Cds 값과 조도계 값을 동시에 측정하여 Cds 값 배열과 조도계 값 배열을 만들어보세요

```
01:         arr_cds = []
02:         arr_lux = []
03:
04:         for i in range(10):
05:             arr_cds.append(cds.readAverage())
06:             arr_lux.append(int(input("Lux: ")))
```

연습문제

- ▣ D. Pop.AI 라이브러리의 Linear_Regression 클래스를 사용해 C에서 만든 두 배열을 각각 X, Y 데이터로 하고 선형 회귀하는 코드를 작성하세요.
- ▣ E. 회귀 모델의 출력과 실제 조도계의 값을 비교해보고 데이터셋 추가 수집, 추가 학습 등 방법으로 ± 30 lux 미만의 오차 범위를 갖는 회귀 모델을 만들어보세요.

AIoT Serbot Prime X로 배우는

온디바이스 AI 프로그래밍

딥러닝

- 딥러닝 : 인공신경망을 기반으로 설계된 기법들을 일컫는 말
 - ▣ 인공신경망
 - 인간의 신경망을 모방하여 만들어진 알고리즘
 - 퍼셉트론에서 시작
 - ▣ 인공신경망을 기반으로 만들어진 딥러닝 기법
 - 심층신경망, 합성곱 신경망, 순환신경망 등

퍼셉트론

- 다수의 데이터로 하나의 결과를 출력하도록 하는 복합 논리 회로
 - ▣ 하나 이상의 데이터 값을 입력받고 각 입력에 가중치를 곱함
 - ▣ 이 값들을 모두 합해 기준값보다 크면 활성화(True)
 - ▣ 기준값보다 작으면 비활성화(False)
 - ▣ 복합적 요인 중 각 요인이 결과에 미치는 영향을 분석할 때 사용

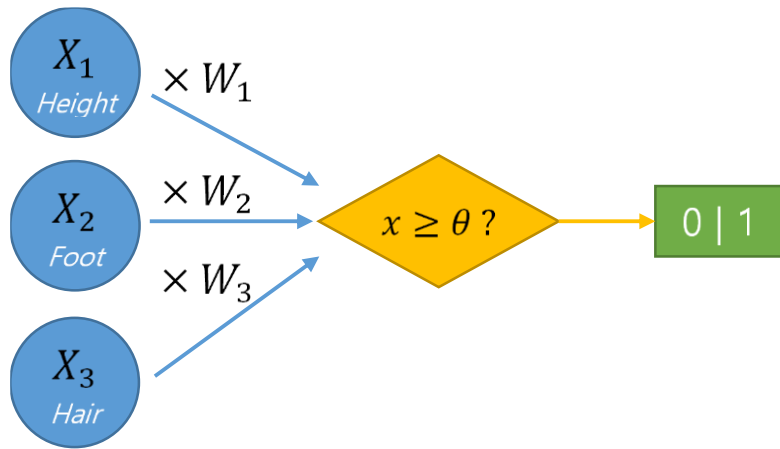
퍼셉트론

- 복합적 요인에 따른 성별 예측을 위한 퍼셉트론 모델 구하는 예제
 - ▣ 같이 키, 발 크기, 머리카락 길이와 성별에 관한 데이터
 - ▣ 성별 값은 True(1)가 여성, False(0)가 남성일 경우로 설정

키(cm)	발 크기(mm)	머리카락 길이(cm)	성별
173	270	17	0
171	275	51	1
162	245	62	1
187	280	12	0
157	230	47	1
169	265	30	0
177	270	5	0
159	250	32	1
182	275	0	0

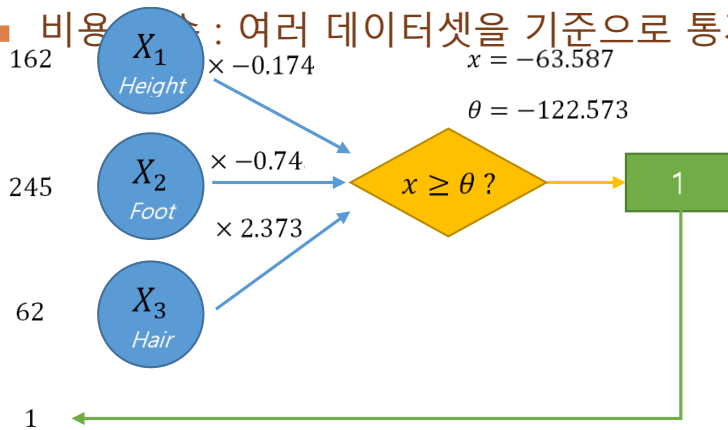
퍼셉트론

- ▣ 표에서의 튜플 : 데이터 형태일 때는 데이터셋
- ▣ 각 데이터에 가중치를 곱하고 모든 데이터를 합한 값 x 과 기준값 θ 을 비교
- ▣ 가중치는 각 데이터가 결과에 가지는 영향력을 의미



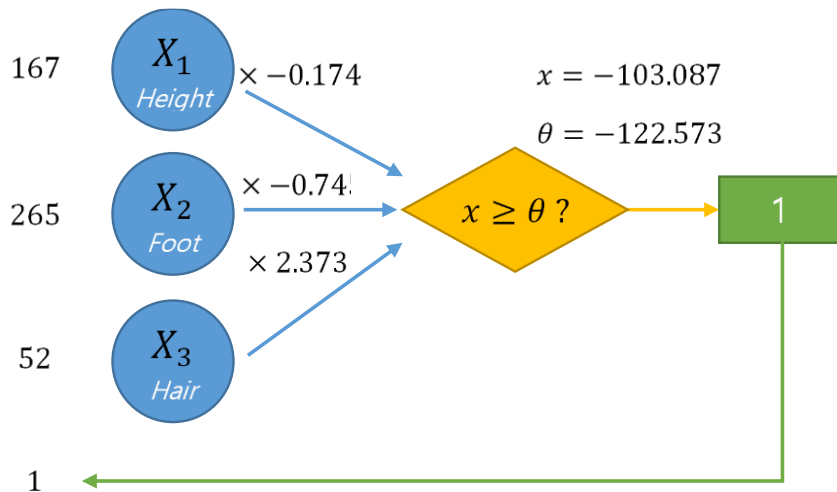
퍼셉트론

- 출력값과 실제값을 비교하고 가중치를 조절해 오차를 줄여나가도록 반복(최적화)
- 손실 함수 또는 비용 함수 : 모델의 출력값과 실제값을 비교하는 함수
 - 손실 함수 : 하나의 데이터셋에서 비교
 - 비용 함수 : 여러 데이터셋을 기준으로 통계를 내거나 성능을 평가



퍼셉트론

- 최적화에 성공한 퍼셉트론 모델
- 새로운 복합 데이터가 주어졌을 때 여성인지 아닌지를 판단
- 각 데이터에 대한 가중치를 조사해 성별과 관련 있는 데이터 분석 가능



퍼셉트론

□ 성별을 예측하는 모델 실습

- ▣ 키, 발 크기, 머리카락 길이, 성별 데이터를 입력받아 학습
- ▣ 새로운 데이터를 입력하면 성별을 예측

▣ Perceptron 객체의 파라미터

- input_size: 입력 데이터의 크기를 의미하며 최하위 차원의 크기 입력 (필수 입력)
- output_size: 결과 데이터의 크기를 의미 (기본값: 1)
- restore: 최근 모델에 이어서 학습할지에 대한 여부를 Boolean으로 입력 (기본값: False)
- ckpt_name: 저장 및 불러올 모델 파일의 이름 (기본값: perceptron)
- softmax: 모델의 예측 결과에 총합이 1이 되도록 할지에 대한 여부를 입력 (기본값: True)
 - 결과 데이터의 크기가 2 이상일 때

퍼셉트론

- ▣ input_size는 사용자 입력이 필수
- ▣ input_size 외 각 기본값은 1, False, perceptron, True로 설정
- ▣ Pop.AI라이브러리 import, Perceptron 객체의 input_size 파라미터 3으로 설정
- ▣ Perc 변수 생성

```
01:         from pop import AI
02:
03:         Perc = AI.Perceptron(input_size=3)
```

퍼셉트론

- ▣ Perceptron 객체의 속성
 - X_data : 입력 데이터
 - Y_data : 입력에 대한 결괏값 데이터

04:	Perc.X_data = [[173,270,17], [171,275,51], [162,245,62], [187,280,12], [157,230,47], [169,265,30], [177,270,5], [159,250,32], [182,275,0]]
05:	Perc.Y_data = [[0],[1],[1],[0],[1],[0],[0],[1],[0]]

퍼셉트론

▣ Perceptron 객체의 train() 메소드

- 퍼셉트론의 학습 시작
- 파라미터 times : 학습할 횟수 (기본값은 100)
- 파라미터 print_every : 학습 상황을 몇 번째마다 출력할지를 의미 (기본값은 10)
- train 메소드를 실행하면 10회마다 퍼셉트론 모델의 오차 출력

06: Perc.train()

퍼셉트론

- ▣ Perceptron 객체의 run() 메소드

- 학습된 모델을 사용
- 파라미터 inputs
 - 기본값은 X_data를 사용
- run 메소드를 실행하면 입력에 대한 퍼셉트론 모델의 성별 확률 출력

07: Perc.run()

퍼셉트론

□ 전체 코드

```
01:         from pop import AI
02:
03:         Perc = AI.Perceptron(input_size=3)
04:
05:         Perc.X_data = [[173,270,17], [171,275,51], [162,245,62], [187,280,12], [157,230,47],
                        [169,265,30], [177,270,5], [159,250,32], [182,275,0]]
06:         Perc.Y_data = [[0],[1],[1],[0],[1],[0],[0],[1],[0]]
07:
08:         Perc.train()
09:         Perc.run()
```

퍼셉트론

▣ 추가 학습

- train 메소드의 times 파라미터를 1000으로 설정하고 학습

```
10:         Perc.train(times=1000, print_every=100)
```

- 이전 학습 모델에 이어서 1,000회 학습해 총 1,100회를 학습했고
- 100회마다 학습 오차를 출력
- run 메소드를 이용해 학습 모델의 예측값 출력

```
11:         Perc.run()
```

퍼셉트론

- run 메소드의 파라미터로 새로운 데이터를 입력하여 출력 확인

```
12: Perc.run([[174,265,6], [152,230,30], [162,255,10]])
```

- 학습 횟수를 과도하게 늘려 100,000회를 학습시켰을 때 결과 확인

```
13: Perc.train(times=100000, print_every=10000)
```

- run 메소드의 파라미터로 새로운 데이터를 입력하여 출력 확인

```
14: Perc.run([[152,230,28], [152,230,30]])
```

퍼셉트론

- 두 표본 데이터의 차이는 2cm 정도의 머리카락 길이임에도 극단적인 결과임
- 이러한 현상이 소프트맥스 회귀에서 언급된 과적합 현상
- 최적의 학습 모델이 무조건 학습 오차를 낮추는 것이 아니라 적절한 수준을 유지하는 것

퍼셉트론

- ▣ Perceptron 객체의 restore 파라미터를 True로 설정
 - 최근 사용한 학습 모델을 불러와 다시 사용 가능

```
15: Perc = AI.Perceptron(input_size=3, restore=True, ckpt_name="model_1")
```

인공신경망

- 인공신경망은 퍼셉트론을 기반으로 고안
- 다수의 데이터로 하나 이상의 결과를 출력하도록 하는 알고리즘
 - ▣ 하나 이상의 데이터 값을 입력받고 각 입력에 가중치를 곱함
 - ▣ 이 값들을 모두 합해 활성화 함수에 입력
 - ▣ 0~1 사이 값으로 결과 출력

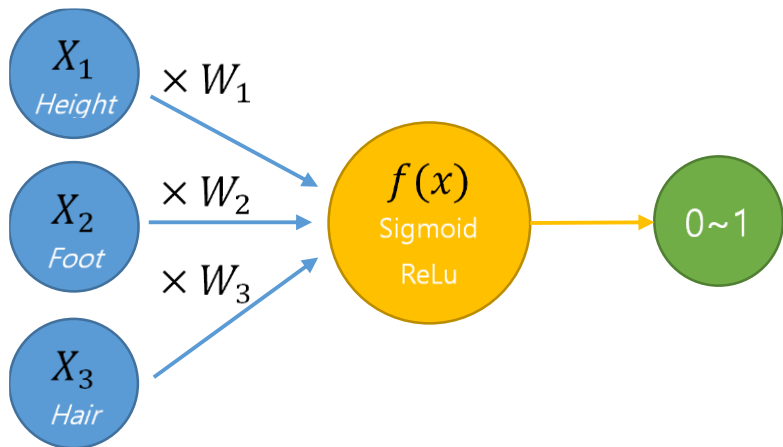
인공신경망

- 퍼셉트론 형태를 기반으로 기초적인 형태의 인공신경망 모델 설명
- 복합적 요인에 따른 성별을 예측하기 위한 인공신경망 모델 구하기
 - ▣ 키, 발 크기, 머리카락 길이와 성별에 관한 데이터가 주어졌을 때
 - ▣ 성별 값은 True(1)가 여성, False(0)가 남성일 경우로 설정

키(cm)	발 크기(mm)	머리카락 길이(cm)	성별
173	270	17	0
171	275	51	1
162	245	62	1
187	280	12	0
157	230	47	1
169	265	30	0
177	270	5	0
159	250	32	1
182	275	0	0

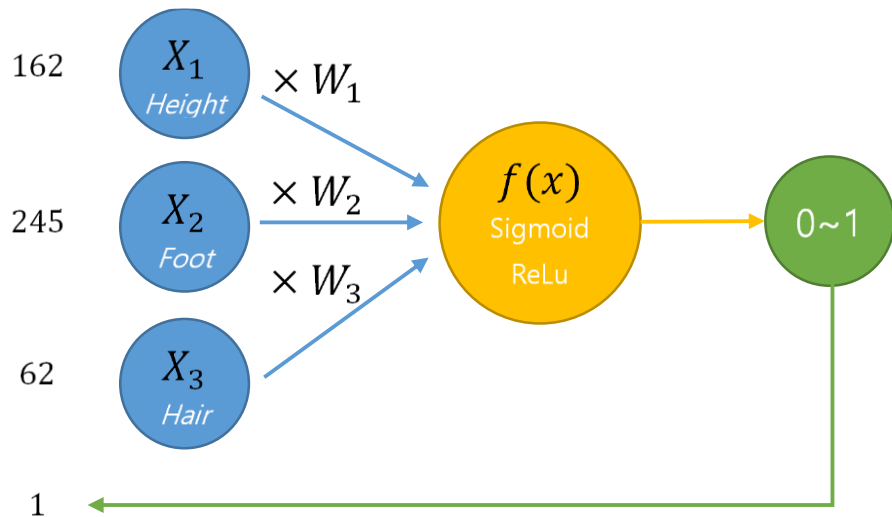
인공신경망

- ▣ 주어진 데이터셋을 나열하고 각 데이터에 임의의 가중치를 곱함
- ▣ 모든 데이터를 합한 값 x 을 활성화 함수에 입력하면 0~1사이값으로 출력
- ▣ 활성화 함수에는 주로 시그모이드 함수와 렐루 함수가 사용됨



인공신경망

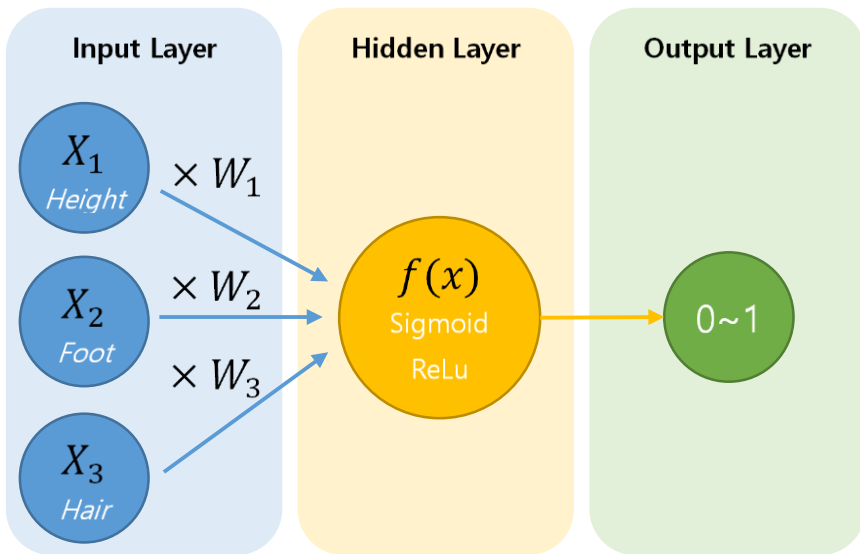
- 활성화 함수의 출력값과 실제값을 비교하고 가중치를 조절하는 과정을 반복
->모델의 최적화



인공신경망

■ 기초 인공신경망 모델

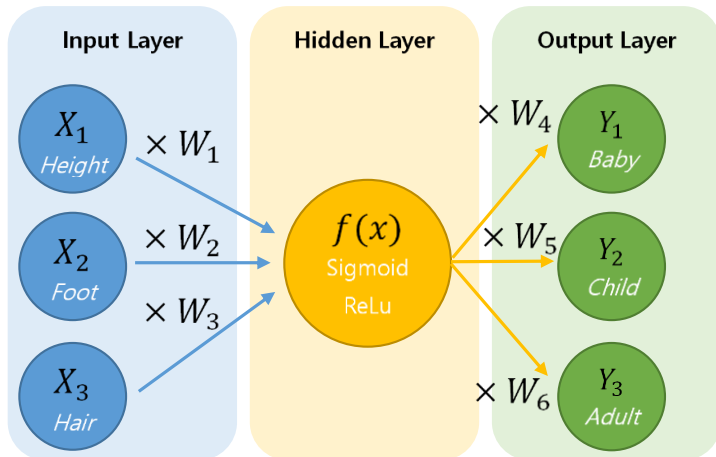
- 역할에 따라 입력층, 은닉층, 출력층으로 구분
- 노드 : 각 계층에 속해 있는 하나의 변수 또는 함수



인공신경망

□ 인공신경망

- 은닉층과 출력층 노드의 개수 조절 가능
- 다수의 입력을 받아 다수의 출력 가능
- 다수의 활성화 함수를 뒤서 복잡한 학습 가능

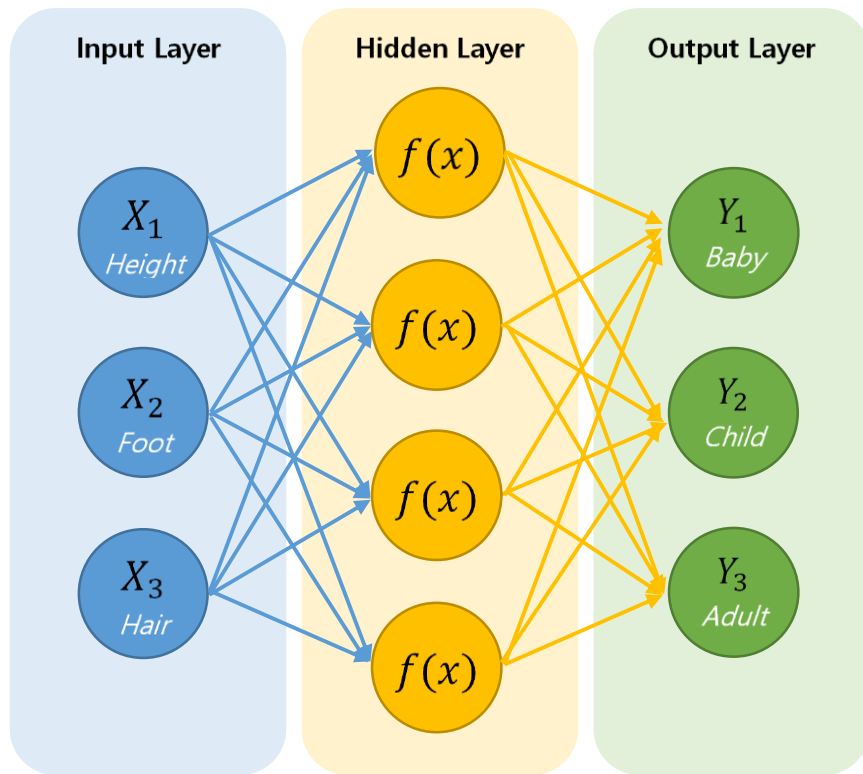


키, 발 크기, 머리카락 길이와 연령층 데이터가 주어졌을 때
복합적 요인에 따른 연령층을 예측하기 위한 인공신경망 모델

인공신경망

- ▣ 유아, 어린이, 성인에 대한 출력층 노드를 3개로 늘림
- ▣ 은닉 노드 : 출력 노드들에 대해 가중치를 갖음
 - 은닉 노드가 출력에 대한 가중치를 주지 않으면 모든 출력 노드가 같은 값을 갖음
 - 은닉층에서 1개 노드만으로 입력을 연산
 - 입력 데이터의 활용 한계가 낮고 잘못된 학습을 하게 될 가능성이 큼
 - 은닉층의 노드를 여러 개로 늘려야 함

인공신경망

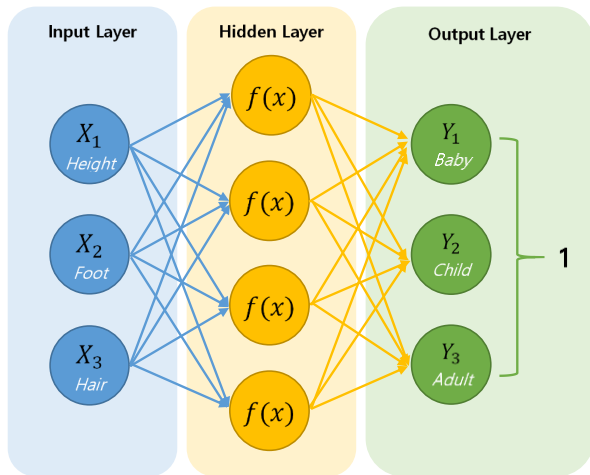


인공신경망

- ▣ 은닉층 노드가 여러 개일 경우
 - 노드들이 각각 받아들이는 입력에 대한 가중치가 다르게 적용
 - 출력에 대한 은닉 노드의 영향력을 분산
 - 한 노드가 잘못된 학습을 하더라도 나머지 3개의 노드에서 정상적인 값을 출력
 - 모델의 오차를 획기적으로 감소가능
- ▣ 예시 모델에서는 4개의 은닉 노드의 가중치가 각각 다름
 - 입력-은닉 구간에서 12개의 가중치와 은닉-출력 구간에서 12개의 가중치를 이용

인공신경망

- 은닉층에서 나온 3개의 출력은 실수 범위
 - 합산 범위를 알 수 없음
 - 결과를 활용하기 위해선 정규화 필요
 - 소프트맥스 함수를 이용해 출력 노드들의 합이 1이 되도록 조정



인공신경망

- 연령층을 예측하는 모델 실습
 - ▣ 키, 발 크기, 머리카락 길이, 연령층에 대한 데이터를 입력받아 학습
 - ▣ 새로운 데이터를 입력하면 연령층을 예측

인공신경망

■ ANN 객체의 파라미터

- input_size: 입력 데이터의 크기. 최하위 차원의 크기 입력 (필수 입력)
- hidden_size: 은닉층의 노드 수. (기본값: 10)
 - hidden_size를 조절하여 더 복잡한 학습이 가능하지만 크기가 커질수록 학습 속도는 느려짐
- output_size: 결과 데이터의 크기. 최하위 차원의 크기 입력 (기본값: 1)
- restore: 최근 모델에 이어서 학습할지에 대한 여부를 Boolean으로 입력 (기본값: False)
- ckpt_name: 저장 및 불러올 모델 파일의 이름 (기본값: ANN)
- softmax: 총합이 1이 되도록 할지에 대한 여부를 Boolean으로 입력 (기본값: True)
 - 결과 데이터의 크기가 2 이상일 때, 모델의 예측 결과에 소프트맥스 함수를 적용

인공신경망

- ▣ Pop.AI라이브러리 import
- ▣ ANN이라는 변수에 생성
 - ANN 객체의 input_size 파라미터를 3, output_size 파라미터를 3으로 설정

```
01:         from pop import AI
02:
03:         ANN = AI.ANN(input_size=3, output_size=3)
```

■ ANN 객체의 속성

- X_data : 입력 데이터
- Y_data : 입력에 대한 결괏값 데이터
- 입력할 데이터들은 유아, 어린이, 성인 각 분류별 3개의 데이터를 사용

04: ANN.X_data = [[73,90,1], [62,70,0], [83,100,2], [110,150,7], [139,220,15],
[123,190,10], [177,275,7], [159,240,35], [182,280,15]]

```
05: ANN.Y_data = [[1,0,0], [1,0,0], [1,0,0], [0,1,0], [0,1,0], [0,1,0], [0,0,1], [0,0,1], [0,0,1]]
```

인공신경망

▣ ANN 객체의 train() 메소드

- 인공신경망 학습 시작
- 파라미터 times : 학습할 횟수 (기본값은 100)
- 파라미터 print_every : 학습 상황을 몇 번째마다 출력할지를 의미 (기본값은 10)
- train 메소드를 실행하면 10회마다 인공신경망 모델의 오차 출력

06: ANN.train()

인공신경망

- ▣ run() 메소드

- 입력에 대한 인공신경망 모델의 세대 분류 결과 출력

07:	ANN.run()
-----	-----------

인공신경망

□ 전체 코드

```
01:         from pop import AI
02:
03:         ANN = AI.ANN(input_size=3, output_size=3)
04:
05:         ANN.X_data = [[73,90,1], [62,70,0], [83,100,2], [110,150,7], [139,220,15], [123,190,10], [177,275,7],
                        [159,240,35], [182,280,15]]
06:         ANN.Y_data = [[1,0,0], [1,0,0], [1,0,0], [0,1,0], [0,1,0], [0,1,0], [0,0,1], [0,0,1], [0,0,1]]
07:
08:         ANN.train()
09:         ANN.run()
```

인공신경망

□ 추가 학습

- train 메소드의 times 파라미터를 1000으로 설정하고 학습

```
01: ANN.train(times=1000, print_every=100)
```

- 이전 학습 모델에 이어서 1,000회 학습해 총 1,100회를 학습
- 100회마다 학습 오차를 출력
- run 메소드를 이용해 학습 모델의 예측값 출력

```
02: ANN.run()
```

인공신경망

- ▣ run() 메소드의 파라미터로 새로운 데이터를 입력하여 출력 확인

03: ANN.run([[174,270,10], [57,70,1], [140,220,10]])

- ▣ 학습 횟수를 과도하게 늘려 100,000회를 학습시켰을 때 결과 확인

04: ANN.train(times=100000, print_every=10000)

- ▣ run() 메소드의 파라미터로 [[140,220,10], [140,200,11]]을 입력 출력 확인

05: ANN.run([[140,220,10],[140,200,11]])

인공신경망

- ▣ ANN 객체의 restore 파라미터를 True로 설정
 - 최근 사용한 학습 모델을 불러와 다시 사용 가능

06: ANN = AI.ANN(input_size=3, output_size=3, restore=True, ckpt_name="model_1")

심층신경망

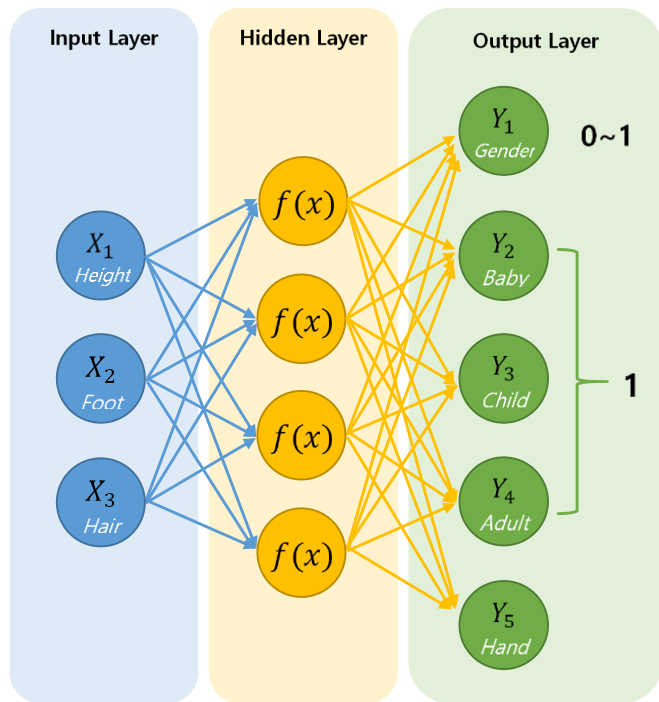
□ 심층신경망

- ▣ 복잡한 모델링을 위해 여러 개의 은닉층을 가지고 있는 인공신경망
- ▣ 많은 은닉 노드를 거치기 때문에 더 많은 클래스를 출력 가능
- ▣ 입력 데이터와 출력 데이터 사이에서 비선형적 관계 파악 가능

심층신경망

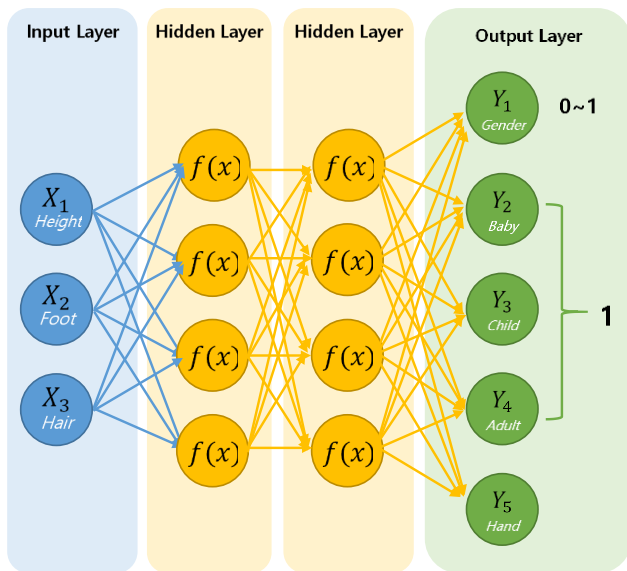
- 복합적 요인에 따른 성별, 연령층, 손 크기 예측 심층신경망 모델
 - ▣ 키, 발 크기, 머리카락 길이와 성별, 연령층, 손 크기에 관한 데이터 주어졌을 때
 - ▣ 성별 값은 True(1)가 여성, False(0)가 남성일 경우로 설정
 - ▣ 인공신경망과 비슷한 구조
 - ▣ 출력에 성별과 손 크기만 추가
 - ▣ 성별 값은 0~1 사이로 출력되도록 시그모이드 또는 ReLu 함수를 사용
 - ▣ 연령층 값들은 그 합이 1이 되도록 소프트맥스 함수를 사용

심층신경망



심층신경망

- 은닉층을 여러 개로 늘려 연결하면 은닉층 간 다양한 네트워크를 구성
- 복잡한 모델링에 적합



심층신경망

- ▣ 최적화에 성공한 심층신경망 모델
 - 새로운 복합 데이터가 주어졌을 때 여성일 확률, 연령층, 손 크기 예측

심층신경망 실습

- 퍼셉트론 실습의 데이터 활용

- ▣ 퍼셉트론과 비교해 얼마나 적은 시간으로 더 효율적인 학습을 하는지에 비교

심층신경망 실습

▣ DNN 객체의 파라미터

- input_size: 입력 데이터의 크기. 최하위 차원의 크기를 입력 (필수 입력)
- hidden_size: 은닉층의 노드 수 (기본값: 10)
 - hidden_size를 조절하여 더 복잡한 학습이 가능하지만 크기가 커질수록 학습 속도는 느려집니다.
- output_size: 결과 데이터의 크기. 최하위 차원의 크기를 입력 (기본값: 1)
- layer_level: 은닉층의 수 (기본값: 3)
 - layer_level을 조절하여 더 깊은 신경망을 만들어 복잡한 학습이 가능
 - 은닉층 차원이 커질수록 학습 속도는 느려지고 과적합 현상이 쉽게 발생할 가능성이 커짐
- restore: 최근 모델에 이어서 학습할지에 대한 여부를 Boolean으로 입력 (기본값: False)
- ckpt_name: 저장 및 불러올 모델 파일의 이름 (기본값: DNN)
- softmax: 총합이 1이 되도록 할지에 대한 여부를 Boolean으로 입력 (기본값: True)
 - 결과 데이터의 크기가 2 이상일 때, 모델의 예측 결과에 소프트맥스 함수를 적용

심층신경망 실습

- ▣ Pop.AI라이브러리 import
- ▣ DNN이라는 변수에 생성
 - input_size 파라미터를 3, output_size 파라미터를 2, layer_level 파라미터를 5로 설정

```
01:         from pop import AI
02:
03:         DNN=AI.DNN(input_size=3, output_size=2,layer_level=5)
```

심층신경망 실습

- DNN 객체의 속성
 - X_data : 입력 데이터
 - Y_data : 입력에 대한 결괏값 데이터
- 입력할 데이터들은 키, 발 크기, 머리카락 길이 데이터를 사용
- 결과 데이터로는 성별 데이터를 사용
- 성별 데이터는 [1, 0] 이면 여성, [0, 1] 이면 남성으로 설정

04: DNN.X_data = [[173,270,17], [171,275,27], [162,245,42], [187,280,12], [157,230,47],
[169,265,30], [177,270,5], [159,250,32], [182,275,0]]

```
05: DNN.Y_data = [[0,1],[1,0],[1,0],[0,1],[1,0],[0,1],[0,1],[1,0],[0,1]]
```

심층신경망 실습

▣ DNN 객체의 train() 메소드

- 심층신경망 학습 시작
- 파라미터 times : 학습할 횟수 (기본값은 100)
- 파라미터 print_every : 학습 상황을 몇 번째마다 출력할지를 의미 (기본값은 10)
- train() 메소드를 실행하면 10회마다 심층신경망 모델의 오차 출력

06: DNN.train()

심층신경망

▣ DNN 객체의 run() 메소드

- 학습된 모델 사용 가능
- 파라미터 inputs
- 기본값은 X_data 사용
- 입력에 대한 심층신경망 모델의 성별 분류 결과 출력

07: DNN.run()

심층신경망

□ 전체 코드

```
01:         from pop import AI
02:
03:         DNN=AI.DNN(input_size=3, output_size=2,layer_level=5)
04:         DNN.X_data = [[173,270,17], [171,275,27], [162,245,42], [187,280,12],
                        [157,230,47], [169,265,30], [177,270,5], [159,250,32], [182,275,0]]
05:         DNN.Y_data = [[0,1],[1,0],[1,0],[0,1],[1,0],[0,1],[0,1],[1,0],[0,1]]
06:         DNN.train()
07:         DNN.run()
```

심층신경망

- ▣ run() 메소드의 파라미터로 새로운 데이터를 입력하여 출력 확인

08: DNN.run([[174,265,6], [152,230,30], [162,255,10]])

합성곱 신경망

□ 합성곱 신경망

- ▣ 입력 데이터의 양이 많을 때 입력 데이터를 압축하여 모델링하는 인공신경망
- ▣ 필요한 수준으로 압축하여 성능과 속도를 모두 확보 가능
- ▣ 입력 데이터를 압축하는 과정에서 슬라이드 윈도우 방식 사용
 - 슬라이드 윈도우 방식 : 입력 데이터보다 작은 사이즈의 배열을 순서대로 옮겨가며 연산
 - 필터링 또는 마스킹 : 옮겨가며 연산하는 과정
 - 커널 또는 윈도우 : 필터링에 사용되는 배열

합성곱 신경망

- 4x4 텐서 데이터가 주어졌을 때 임의로 2x2 가중치 커널을 생성

Input Data

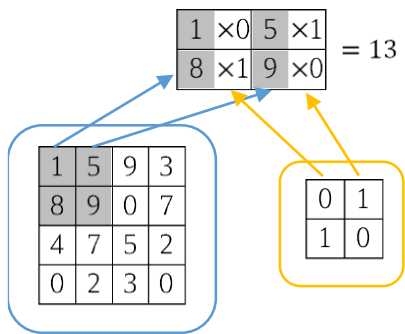
1	5	9	3
8	9	0	7
4	7	5	2
0	2	3	0

Weight

0	1
1	0

합성곱 신경망

- 데이터의 왼쪽 위부터 값을 곱한 후 모두 합해 1개 값으로 반환
- 옆으로 한 칸 옮겨 반복
- 이 과정을 필터링 또는 마스킹이라고 함



합성곱 신경망

□ 필터링 중 중간 과정

$$\begin{array}{|c|c|c|c|} \hline 1 & 5 & 9 & 3 \\ \hline 8 & 9 & 0 & 7 \\ \hline 4 & 7 & 5 & 2 \\ \hline 0 & 2 & 3 & 0 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline 0 & 1 \\ \hline 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 13 & 18 & 3 \\ \hline 13 & 7 & \\ \hline & & \\ \hline \end{array}$$

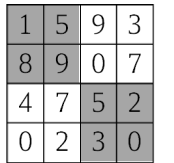
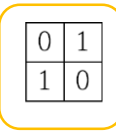
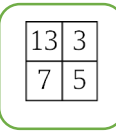
□ 필터링이 완료되면 3x3 사이즈의 압축 데이터가 반환

- 이 데이터를 심층신경망에 연결하면 보다 빠른 속도를 기대할 수 있음

$$\begin{array}{|c|c|c|c|} \hline 1 & 5 & 9 & 3 \\ \hline 8 & 9 & 0 & 7 \\ \hline 4 & 7 & 5 & 2 \\ \hline 0 & 2 & 3 & 0 \\ \hline \end{array} \times \begin{array}{|c|c|} \hline 0 & 1 \\ \hline 1 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 13 & 18 & 3 \\ \hline 13 & 7 & 12 \\ \hline 7 & 7 & 5 \\ \hline \end{array}$$

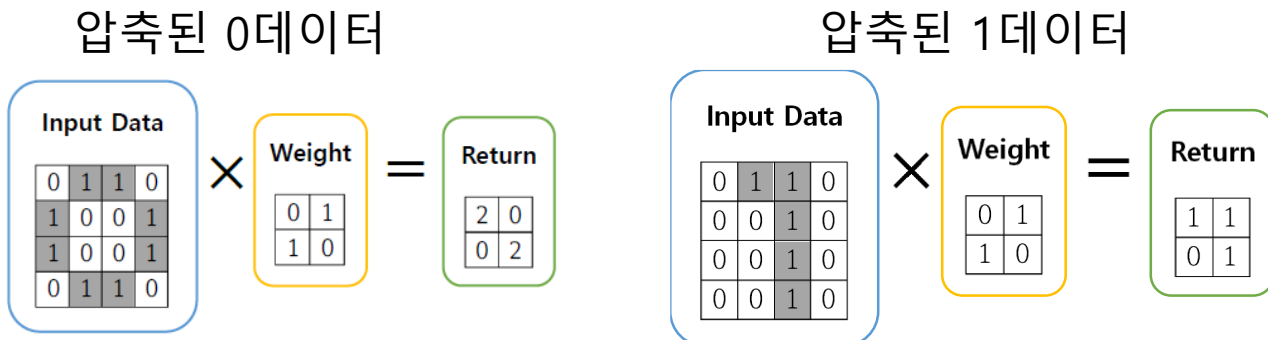
합성곱 신경망

- 스트라이드 : 커널이 슬라이드 할 때, 한 번에 넘어갈 칸수
 - ▣ 스트라이드를 2로 설정하면 2칸씩 건너뛰어 필터링 함

 \times  $=$ 

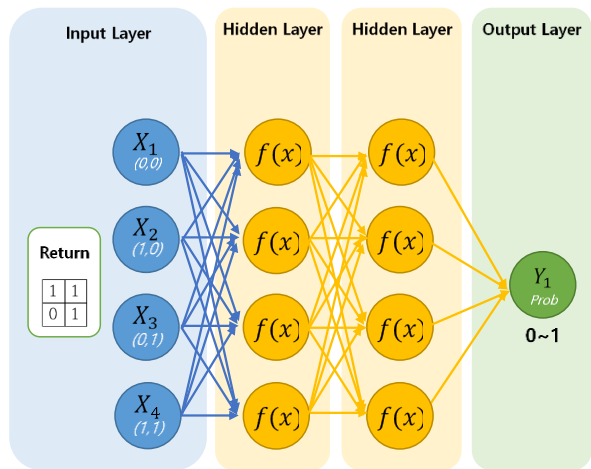
합성곱 신경망

- 0 또는 1을 그린 4x4 이미지를 분류하는 합성곱 신경망 모델
 - ▣ 데이터에서 흰색을 0, 검은색을 1로 설정
 - ▣ 스트라이드가 2인 2x2 커널을 이용해 압축 데이터를 구함



합성곱 신경망

- 반환된 값을 나열하여 심층신경망에 입력
 - ▣ 평탄화 : 압축 데이터를 나열하는 과정
 - ▣ 완전 연결 계층 : 압축 데이터와 심층신경망을 연결하는 은닉층



합성곱 신경망

- 숫자 식별 모델 실습
 - ▣ 손글씨로 쓴 숫자 이미지를 입력받아 학습
 - ▣ 새로운 이미지 데이터 입력시 숫자 식별

합성곱 신경망 실습

■ CNN 객체의 파라미터

- input_size: 2차원 데이터를 입력 데이터로 사용. 리스트로 입력 (기본값: [28, 28])
- input_level: RGB 이미지인 경우 3, 흑백 이미지인 경우 1을 입력 (기본값: 1)
- kernel_size: 합성곱 계층에서 사용할 커널의 크기. 리스트로 입력 (기본값: [3, 3])
- kernel_count: 하나의 입력 데이터에 사용할 가중치 커널의 개수 (기본값: 32)
 - 커널의 개수가 많아지면 더 다양한 특징을 찾아내지만, 속도는 급격히 느려짐
- strides: 커널의 스트라이드를 설정. 리스트로 입력 (기본값: [1, 1])
- hidden_size: 은닉층의 노드 수 (기본값: 128)
 - hidden_size를 조절하여 더 복잡한 학습이 가능하지만 크기가 커질수록 학습 속도는 느려짐

합성곱 신경망 실습

- `output_size`: 결과 데이터의 크기. 최하위 차원의 크기를 입력 (기본값: 1)
- `conv_level`: 합성곱 계층의 개수를 설정. 값이 커질수록 합성곱 계층은 깊어짐 (기본값: 2)
- `layer_level`: 은닉층의 수 (기본값: 1)
 - `layer_level`을 조절하여 더 깊은 신경망을 만들어 복잡한 학습이 가능
 - 은닉층 차원이 커질수록 학습 속도는 느려지고 과적합 현상이 쉽게 발생할 가능성 커짐
- `restore`: 최근 모델에 이어서 학습할지에 대한 여부를 Boolean으로 입력 (기본값: False)
- `ckpt_name`: 저장 및 불러올 모델 파일의 이름 (기본값: CNN)
- `softmax`: 총합이 1이 되도록 할지에 대한 여부를 Boolean으로 입력 (기본값: True)
 - 결과 데이터의 크기가 2 이상일 때, 모델의 예측 결과에 소프트맥스 함수를 적용

합성곱 신경망 실습

- ▣ Pop.AI라이브러리 import
- ▣ CNN이라는 변수에 생성
 - output_size 파라미터를 10으로 설정

```
01:         from pop import AI
02:
03:         CNN = AI.CNN(output_size=10)
```

합성곱 신경망 실습

- ▣ MNIST 데이터셋을 다운로드 받아 사용
 - CNN 객체 속성에 X_data와 Y_data가 있지만 이미지 데이터는 직접 입력하기에 부적합
- ▣ MNIST는 손으로 쓴 숫자 이미지 데이터 베이스
- ▣ CNN 객체의 load_MNIST() 메소드
 - 자동으로 데이터셋을 불러와 X_data와 Y_data에 입력
 - 만약 장치에 데이터셋이 없다면 자동으로 다운로드 (인터넷 연결이 필요)

04: CNN.load_MNIST()

합성곱 신경망 실습

- ▣ CNN 객체의 `show_img()` 메소드
 - 이미지를 출력
 - 파라미터 `input` : `x`, `y`, `color`를 담는 3차원 배열 입력
 - 이미지 사이즈는 CNN 객체 선언이 사용된 `input_size`로 지정

05: `CNN.show_img(CNN.X_data[501])`

합성곱 신경망 실습

▣ CNN 객체의 train() 메소드

- 합성곱 신경망 학습 시작
- 파라미터 times : 학습할 횟수 (기본값은 100)
- 파라미터 print_every : 학습 상황을 몇 번째마다 출력할지를 의미 (기본값은 10)
- train() 메소드를 실행하면 10회마다 합성곱 신경망 모델의 오차 출력

06: CNN.train()

합성곱 신경망 실습

▣ CNN 객체의 run() 메소드

- 학습된 모델 사용 가능
- 파라미터 inputs
- 기본값은 X_data 사용
 - X_data의 크기가 매우 크므로 X_data의 일부 입력
- run() 메소드를 실행하면 입력에 사용된 이미지를 표시
 - 입력에 대한 합성곱 신경망 모델의 숫자 분류 결과를 출력 배열의 순서대로 0~9일 확률을 출력

```
07:         X = [CNN.X_data[10]]
08:         CNN.run(X)
```

합성곱 신경망 실습

□ 전체 코드

```
01:         from pop import AI
02:
03:         CNN = AI.CNN(output_size=10)
04:         CNN.load_MNIST()
05:
06:         CNN.train()
07:
08:         X = [CNN.X_data[10]]
09:         CNN.run(X)
```

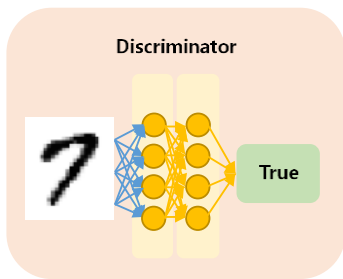
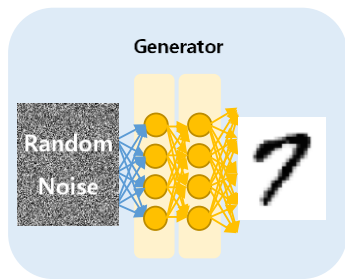
▣ restore 파라미터를 True로 설정

■ 최근 사용한 학습 모델을 불러와 다시 사용 가능

```
01:         CNN=AI.CNN(output_size=10, restore=True)
```

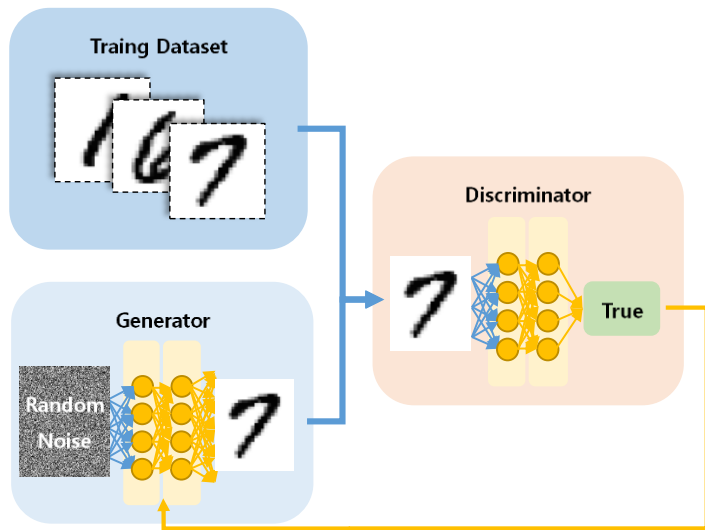
생산적 적대 신경망

- 판별 모델과 위조 모델을 학습시켜 고수준의 가짜 데이터를 모델링
 - ▣ 판별 모델 : 특정 데이터의 진위를 구별
 - ▣ 위조 모델 : 가짜 데이터를 생성
 - ▣ 생성자 : 데이터를 생성하는 인공신경망 모델
 - ▣ 판별자 : 데이터를 판별하는 인공신경망 모델



생산적 적대 신경망

- ▣ 생성자 모델 : 랜덤 노이즈를 생성하고 가중치를 조절해 위조 데이터 만들
- ▣ 판별자 모델 : 훈련 데이터셋과 위조 데이터셋을 입력받아 위조 여부 학습
- ▣ 위조와 판별을 반복하며 두 모델을 계속해서 발전시킴



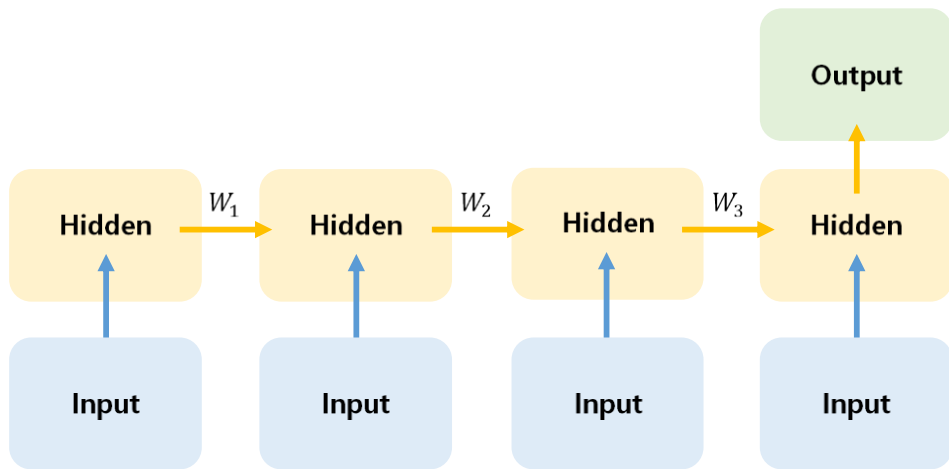
생산적 적대 신경망

- 성공적인 생산적 적대 신경망 모델 학습
 - ▣ 두 모델의 수준이 비슷하도록 학습률 조절 및 유지
 - ▣ 생성자 모델이 너무 뛰어날 경우
 - 판별자 모델은 진짜 데이터를 가짜라고 판별하게 학습할 가능성이 큼
 - ▣ 판별자 모델이 너무 뛰어날 경우
 - 생성자 모델은 False 피드백만 받음
 - 진짜 같은 데이터를 생성하기 위한 가중치를 찾을 수 없음

순환신경망

- 과거 입력 데이터를 이후 처리에도 반영하여 사용하는 인공신경망
 - ▣ 은닉층의 출력을 다음 스텝에서 입력 데이터와 함께 다시 입력
- 기존 딥러닝 알고리즘들의 최대 결점은 시간 개념이 없다는 것
 - ▣ 여러 데이터셋들 간의 관계나 순서를 반영하지 않음
- 순환신경망은 문장의 전후 관계, 이미지의 사물 관계를 파악 가능
 - ▣ 과거 데이터값을 요약하여 갖고 있음

순환신경망



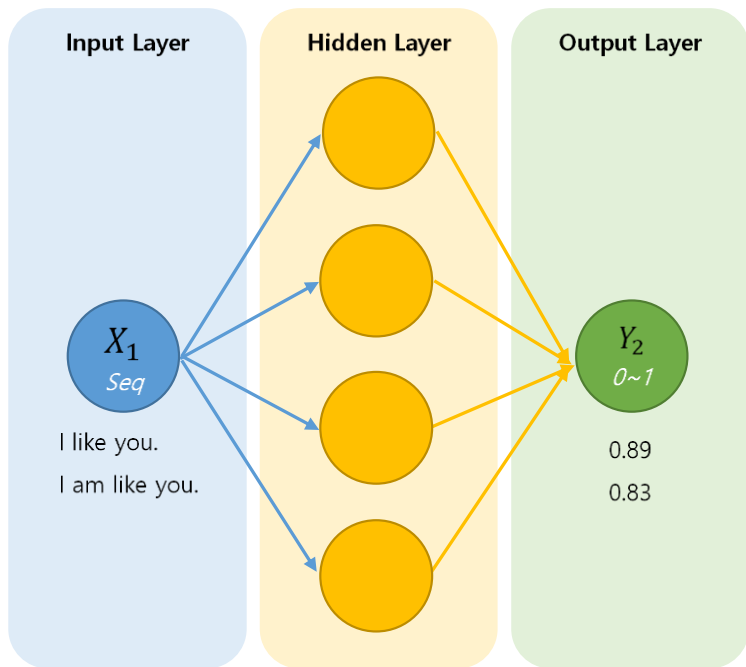
순환신경망

- ▣ 'Like'의 동사와 전치사를 구분하는 순환신경망 모델과 심층신경망 모델 비교
 - 다음과 같은 두 데이터셋이 입력으로 주어졌을 때
 - 전치사를 True(1), 동사를 False(0)이라고 설정

문장	구분
I like you.	0
I am like you.	1

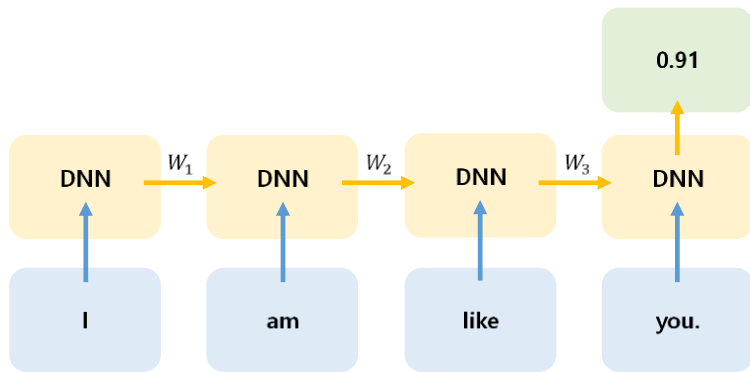
순환신경망

- ▣ 심층신경망 모델에서는 문장 전체를 입력으로 받아 처리

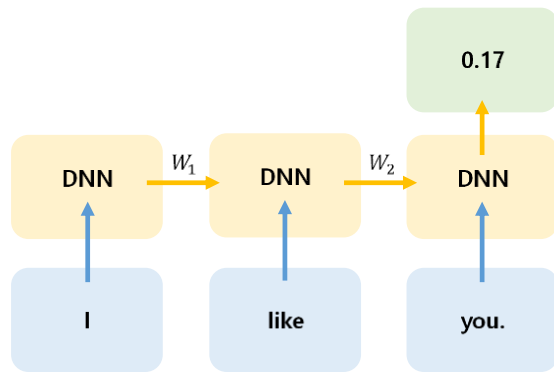


순환신경망

- 심층신경망은 전체적인 형태가 매우 비슷한 두 데이터셋을 똑같이 구분
 - 각 단어의 순서와 상호 관계를 알 수 없음
- 순환신경망은 'am'의 문장 결정력을 구분 가능
 - 문장에 포함된 단어를 순서대로 입력받고 다음 처리에 연결



순환신경망의 'I am like you' 구분



순환신경망의 'I like you' 구분

내용 정리

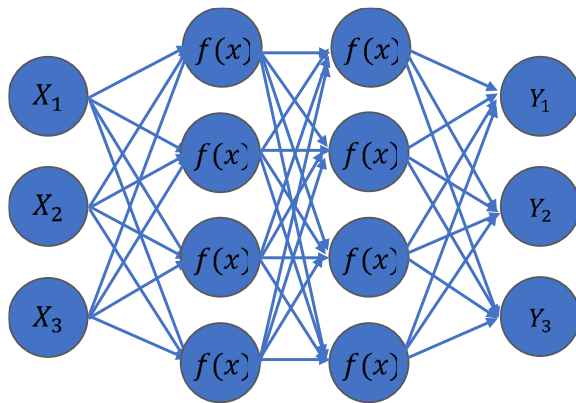
- 딥러닝 : 인공신경망 이론을 기반으로 설계된 머신러닝 기법
- 퍼셉트론 : 다수의 데이터로 하나의 결과를 출력하는 복합 논리 회로
 - ▣ 각 요인이 결과에 미치는 영향을 분석할 때 사용
- 인공신경망 : 퍼셉트론을 기반으로 고안
 - ▣ 다수의 데이터로 하나 이상의 결과를 출력하는 알고리즘
- 심층신경망 : 여러 은닉층을 가지고 있는 인공신경망
 - ▣ 입력과 결과의 복잡한 관계 분석

내용 정리

- 합성곱 신경망 : 대량의 입력 데이터를 압축하여 분석하는 인공신경망
- 생산적 적대 신경망 : 데이터의 진위를 구별하는 판별 모델과 가짜 데이터를 생성하는 위조 모델을 경쟁시켜 모방 데이터를 생성하도록 모델링하는 인공신경망
- 순환신경망 : 이전 입력을 이후 처리에도 반영하여 데이터의 연속 관계를 분석하는 인공신경망

연습문제

- 문제 38. 다음 신경망 그림을 보고 질문에 답해보세요.



- A. 신경망의 입력과 출력의 크기를 답해보세요.
- B. 신경망의 은닉층 수를 답해보세요.

연습문제

□ 문제 39. 다음 문장들을 읽고 빈 칸을 채워보세요.

- A.딥러닝은 [] 기반으로 설계된 머신러닝 기법들이다.
- B.퍼셉트론은 [] -[] 출력하는 복합 논리 회로이다.
- C.심층신경망은 여러개의 -은닉-층을 가지고 있다.
- D.합성곱 신경망은 -슬라이딩- [] -[] 방식으로 대용량 데이터를 압축한다.



연습문제

- ▣ E.합성곱 신경망의 데이터 압축 과정 중 필터링에 사용되는 배열을 이라 하고, 이것이 한 번에 이동할 칸 수를 고 한다.
- ▣ F.생산적 적대 신경망은 와 경쟁시키는 방법으로 학습하여 위조 데이터를 생성한다.

연습문제

- 문제 40. 다음 그림의 배열(좌)과 커널(우)을 이용해 질문에 답해보세요.

0	1	0	0	0
1	0	0	0	0
0	0	1	0	0
0	1	0	0	1
0	0	0	1	0

0	1
1	0

- A. 스트라이드가 3 (또는 $[3, 3]$) 일 때 출력을 작성하세요.
- B. A의 출력이 완전 연결 계층에 입력될 때 완전 연결 계층의 노드 수를 답하세요.

연습문제

- 문제 41. 다음 코드는 Pop.AI 라이브러리를 이용하여 2개의 입력을 받아 2개의 출력을 하는 심층신경망을 구현한 코드입니다. 질문을 읽고 답해보세요.

```
01:         from pop import AI
02:
03:         DNN = AI.DNN(input_size=2, output_size=2, softmax=True)
04:
05:         DNN.X_data = 
06:         DNN.Y_data = 
07:
08:         DNN.train(times=1000, print_every=100)
09:         DNN.run()
```

연습문제

- ▣ A.두 입력에 대해 크기를 비교하여 큰 쪽을 1, 작은 쪽을 0으로 출력할 수 있도록 빈 칸 X, Y에 들어갈 학습 데이터셋을 작성해보세요.
- ▣ B.A에서 작성한 데이터셋으로 학습시키고, [1,0]과 [-1,0]을 입력했을 때 출력을 작성하세요.

연습문제

- 문제 42. 다음 코드는 AIoT SerBot의 카메라 영상을 BGR 값으로 받아 이미지로 출력하는 코드입니다. 이 코드를 응용해 다음 문제들을 해결해보세요.

```
01:         from pop import Camera, Util
02:
03:         cam = Camera(width=50, height=50)
04:
05:         value = cam.value
06:         Util.imshow("Title", value)
```

연습문제

- ▣ A. 다음 코드를 실행해 카메라 BGR 데이터의 형태를 확인해보세요.

```
07:         print(value.shape)
```

- ▣ B. 다음 사진처럼 손바닥을 카메라에 비출 때 BGR 값을 받아 이미지로 확인해보세요.



연습문제

- ▣ C. 빈 배열을 생성하고, B와 같은 방법으로 다양한 손바닥 데이터를 20개 이상 추가하는 코드를 작성하고 실행해보세요.
- ▣ D. C의 배열에 손바닥이 없는 카메라 데이터를 20개 이상 추가해보세요.
- ▣ E. 빈 배열을 생성하고, 손바닥이 있는 사진과 없는 사진을 각각 1과 0으로 하여 C에서 추가한 개수만큼 1을 추가하고 D에서 추가한 개수만큼 0을 추가하세요.

연습문제

- ▣ F. Pop.AI 라이브러리의 CNN 클래스를 사용해 D에서 완성된 배열을 X 데이터, E에서 완성된 배열을 Y 데이터로 하는 합성곱 신경망 코드를 작성하세요.
- ▣ G. F에서 작성한 합성곱 신경망 모델을 학습시켜 손실율을 최소화해보세요.
- ▣ H. G에서 학습한 모델에 새로운 카메라 BGR 데이터를 입력하여 결과를 확인해보세요.

AIoT SerBot Prime X로 배우는

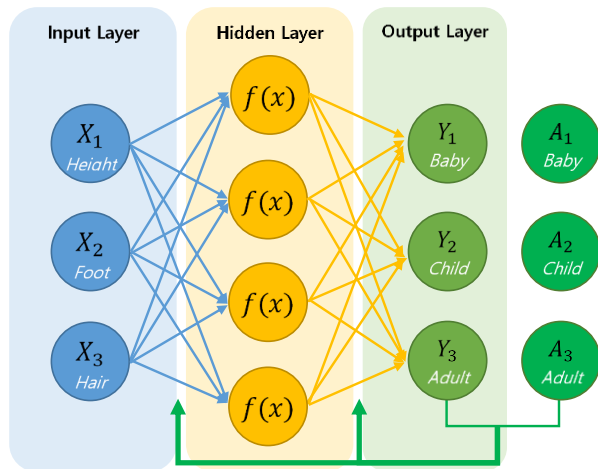
온디바이스 AI 프로그래밍

강화 학습

- 학습 초기에는 우연에 의존하여 학습. 점차 보상을 최대화하도록 학습
- 학습 초기 오차는 매우 높음. 시간이 지날수록 급격히 오차가 줄어듦
- 학습 데이터 가공이 필요 없고 필요한 선택만 하도록 학습
 - ▣ 고성능 환경에서 짧은 시간에 많은 학습 가능
- 초기 학습 모델에 의해 최종 학습 모델이 결정
- 학습 특징은 비지도 학습과 매우 유사하지만, 그 적용 대상이 다름
- 데이터 식별보다는 데이터 시뮬레이션 용도에 적합

Deep Q-Network

- 일반적인 딥러닝 알고리즘은 강화학습 불가능
 - ▣ 역전파 : 신경망 모델이 출력한 결과와 정답 데이터를 비교해 어떤 가중치 변수를 얼마나 조절할지 결정하고, 해당 가중치 변수에 전달하는 과정



Deep Q-Network

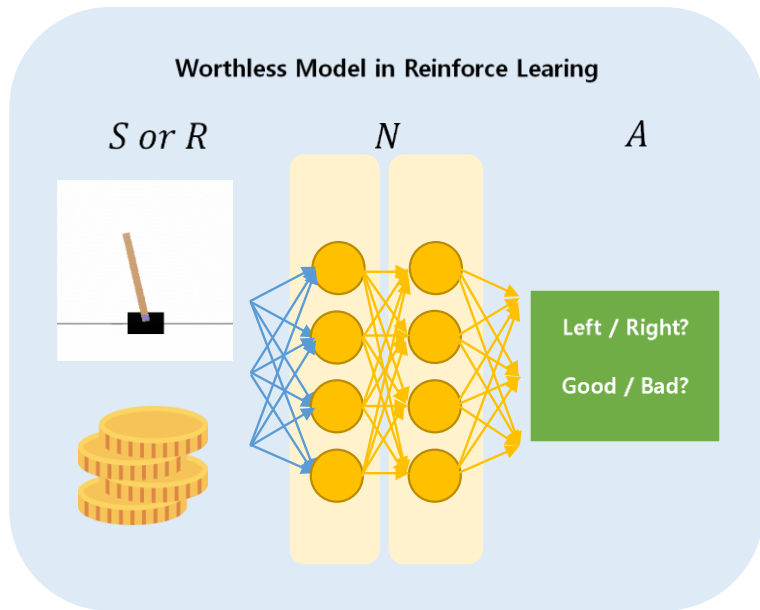
- 역전파를 하려면 ?
 - ▣ 모델의 출력 데이터와 비교할 정답 데이터가 같은 의미를 가진 데이터여야 함
 - ▣ 의미 있는 학습을 위해서는 가중치가 정답 데이터와 연관이 있어야 함

Deep Q-Network

- 강화 학습 이론 (벽돌 깨기를 가정)
 - ▣ 학습(train)하는 시점에는 '보상'입력
 - ▣ 행동(run)하는 시점에는 '상태'입력
 - ▣ 행동 입력과 학습 입력이 다르므로 입력에 대한 가중치는 하나로 수렴 불가
 - '상태'에 집중한 신경망 모델은 '보상'을 전혀 학습하지 못함
 - '보상'에 집중한 신경망 모델은 '상태'에 대한 행동을 전혀 하지 못함
 - ▣ 신경망의 핵심인 역전파 과정이 불가능한 학습
 - 강화 학습 실현의 걸림돌이 됨

Deep Q-Network

- 강화 학습 이론 (벽돌 깨기를 가정)



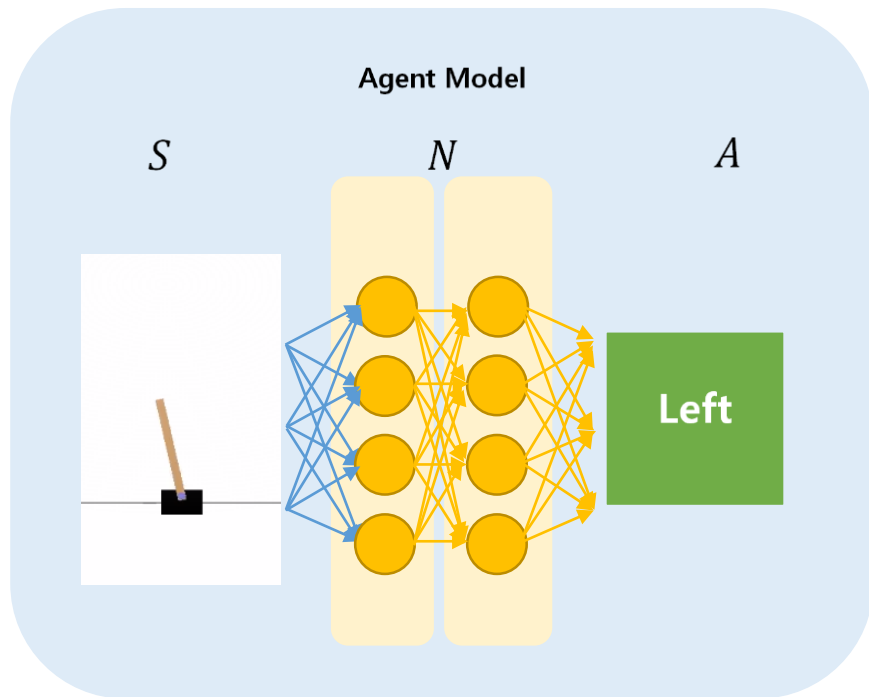
Deep Q-Network

□ DQN 알고리즘

- ▣ 게임을 진행하는 모델과 보상에 대해 학습하는 모델을 분리
- ▣ 리플레이라는 개념을 추가해 문제 해결
 - 에이전트 : 게임을 플레이하는 행동 모델
 - 리플레이 : 이전에 진행했던 게임 상태 기록, 에이전트 행동 기록, 보상 기록
- ▣ 에이전트의 행동에 대하여 게임은 어떤 보상을 줬는지 기록

Deep Q-Network

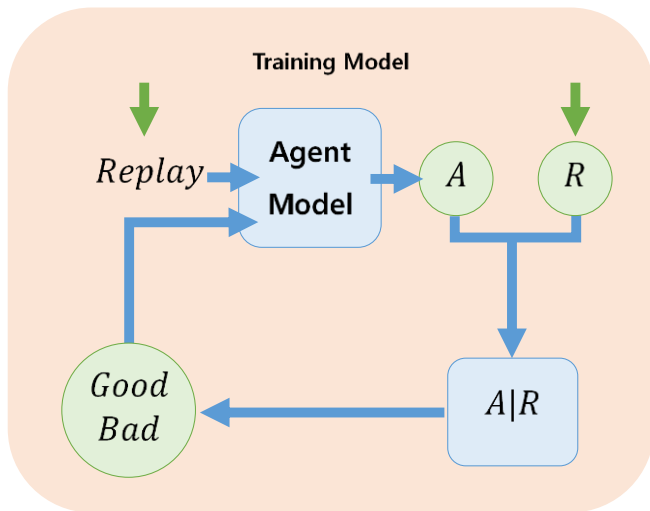
- 에이전트 : 게임 상태 S 에서 신경망 N 에 입력되어 결과로 행동 A 출력



Deep Q-Network

▣ 학습 모델

- 보상 R 이 주어졌을 때 리플레이 데이터를 불러와 에이전트에 입력
- 출력된 행동 데이터와 보상 R 을 비교해 에이전트의 행동 결과를 판단하여 가중치 조절



Deep Q-Network

- 강화 학습에 유용한 라이브러리인 OpenAI Gym을 사용하여 DQN을 실습
 - OpenAI Gym : 강화 학습을 위해 그래픽 요소, 행동과 보상 등을 제공
 - 알고리즘 구현에 집중할 수 있는 환경 제공
 - Pop.AI 라이브러리를 이용해 DQN을 구현
 - OpenAI Gym의 그래픽 창을 Jupyter 노트북에 띄워 실습
 - DISPLAY 환경변수 값 변경
 - 소스 코드 실행전에 실행 필요

```
01: %set_env DISPLAY=:0.0
```

- Pop.Util 라이브러리의 imshow() 메소드 사용

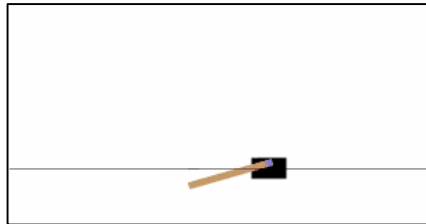
```
01: from pop import Util
02: Util.imshow("Title", Image, width=300, height=300, mode='RGB')
```

OpenAI Gym

□ CartPole 예제

- Gym에서 간단한 물리 엔진을 포함한 게임을 표시
- 카트가 빠른 속도로 오른쪽으로 움직여 사라짐
- 아무런 게임 규칙이 적용되지 않아 리셋 안됨

```
01: import gym
02: from pop import Util
03:
04: env = gym.make('CartPole-v1')
05: env.reset()
06:
07: for _ in range(1000):
08:     img = env.render(mode = 'rgb_array')
09:     Util.imshow("CartPole", img, mode='RGB')
10:     env.step(env.action_space.sample())
11: env.close()
```



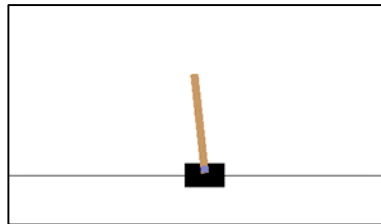
OpenAI Gym

■ 게임의 규칙 적용

- 게임 종료 : 막대가 15도 이상 기울거나 500 스텝 이상 진행될 경우 게임 종료
- 보상 : 막대가 15도 이하로 직립한 스텝에서 1의 보상이 주어짐
- 행동 : env.step 메소드에 True 또는 False를 입력해 좌우로 움직일 수 있음
- 결과 : env.step 메소드를 실행하면 결과값들이 반환
- state : 행동으로 인해 변화된 상태.
 - 리스트로 반환되며 차례대로 카트의 위치, 카트의 속도, 막대의 각도, 막대의 속도
- reward : 해당 스텝의 보상. 0 또는 1이 반환
- done : 게임이 종료되었는지 여부

OpenAI Gym

```
01: import gym
02: from pop import Util
03:
04: env = gym.make('CartPole-v1')
05: for i_episode in range(20):
06:     state = env.reset()
07:     for t in range(100):
08:         img = env.render(mode='rgb_array')
09:         Util.imshow("CartPole", img, mode='RGB')
10:
11:         action = env.action_space.sample()
12:         state, reward, done, _ = env.step(action)
13:         if done:
14:             print("Episode finished after {} timesteps".format(t+1))
15:             break
16: env.close()
```



OpenAI Gym – pop.AI

▣ DQN 객체의 파라미터

- state_size: 상태 데이터의 크기. 최하위 차원의 크기 입력 (필수 입력)
- hidden_size: 은닉층의 노드 수 (기본값: 5)
 - hidden_size를 조절하여 더 복잡한 학습이 가능하지만 크기가 커질수록 학습 속도는 느려짐
- output_size: 결과 데이터의 크기. 최하위 차원의 크기 입력 (기본값: 1)
- layer_level: 은닉층의 수 (기본값: 1)
 - layer_level을 조절하여 더 깊은 신경망을 만들어 복잡한 학습이 가능
 - 은닉층 차원이 커질수록 학습 속도는 느려지고 과적합 현상이 쉽게 발생할 가능성이 커짐
- restore: 최근 모델에 이어서 학습할지에 대한 여부를 Boolean으로 입력 (기본값: False)
- ckpt_name: 저장 및 불러올 모델 파일의 이름 (기본값: DQN)

OpenAI Gym – pop.AI

- ▣ Gym, Pop.AI 라이브러리 import
- ▣ DQN이라는 변수에 생성
 - AI모듈에서 DQN 객체의 state_size 파라미터를 4로 설정
- ▣ gym의 CartPole 환경을 생성하여 env변수에 저장

```
01:         import gym
02:         from pop import Ai, Util
03:
04:         DQN=AI.DQN(state_size=4)
05:
06:         env = gym.make('CartPole-v1')
```

OpenAI Gym – pop.AI

- ▣ 게임을 1000번 플레이하도록 for 루프 생성
- ▣ env객체의 reset() 메소드로 환경을 초기화
- ▣ step과 total_reward를 0으로 초기화
- ▣ 리플레이 구현을 위해 상태, 보상, 행동을 기록할 리스트 생성

```
07:         for i_episode in range(1000):
08:             state = env.reset()
09:             step = 0
10:             total_reward = 0
11:
12:             states, rewards, actions=[], [], []
```

OpenAI Gym – pop.AI

- ▣ 한 게임을 진행할 때 게임이 종료될 때까지 행동하도록 무한 루프 생성
- ▣ env객체의 render() 메소드로 현재 상황을 그래픽으로 출력
- ▣ DQN객체의 run() 메소드에 현재 상태state를 입력하면 에이전트의 행동 값 출력
- ▣ 이 행동 값을 env.step에 입력하면
 - CartPole환경에서 행동으로 인해 변화된 상태와 보상 출력
- ▣ 상태, 보상, 행동 기록을 리플레이 리스트에 추가
- ▣ 총 보상과 스텝 갱신

OpenAI Gym – pop.AI

```
13:         while True:
14:             img = env.render(mode = 'rgb_array')
15:             Util.imshow("CartPole", img, mode='RGB')
16:
17:             action=DQN.run([state])
18:
19:             state, reward, done, _ = env.step(action)
20:
21:             states.append(state)
22:             rewards.append(reward)
23:             actions.append(action)
24:
25:             total_reward+=reward
26:             step+=1
```

OpenAI Gym – pop.AI

- ▣ done 변수가 True인 경우
 - 한 스텝의 게임 종료
- ▣ 게임 종료 시
 - 이번 게임에서 몇 스텝까지 진행했는지 출력
 - DQN객체의 train메소드에 리플레이 리스트들을 입력하여 에이전트 가중치 조절
 - train메소드 파라미터 : states, rewards, actions

OpenAI Gym – pop.AI

```
27:                 if done:
28:                     print("Done after {} steps".format(step+1))
29:
30:                 loss=DQN.train(states, rewards, actions)
31:                 print('episode ' + str(i_episode + 1) + " reward : ", total_reward, ", loss : ",loss)
32:                 break
```

▣ 게임이 모두 끝나면 CartPole 환경 종료

```
33:         env.close()
```

OpenAI Gym – pop.AI

□ 전체 코드

```
01: import gym
02: from pop import AI, Util
03:
04: DQN=AI.DQN(state_size=4)
05:
06: env = gym.make('CartPole-v1')
07: for i_episode in range(1000):
08:     state = env.reset()
09:     step = 0
10:     total_reward = 0
11:
12:     states, rewards, actions=[], [], []
13:     while True:
14:         img = env.render(mode = 'rgb_array')
15:         Util.imshow("CartPole", img, mode='RGB')
16:
17:         action=DQN.run([state])
```

OpenAI Gym – pop.AI

□ 전체 코드

```
18:         state, reward, done, _ = env.step(action)
19:
20:         states.append(state)
21:         rewards.append(reward)
22:         actions.append(action)
23:
24:         total_reward+=reward
25:         step+=1
26:
27:         if done:
28:             print("Done after {} steps".format(step+1))
29:
30:             loss=DQN.train(states, rewards, actions)
31:             print('episode '+ str(i_episode + 1)+" reward : ", total_reward, ", loss : ",loss)
32:             break
33:
34: env.close()
```

내용 정리

□ 강화 학습

- ▣ 상황에 대한 데이터가 주어진 경우
- ▣ 어떤 행동을 하면 그에 따른 보상 및 벌이 발생
- ▣ 상황, 행동, 보상의 관계를 분석해 머신러닝 모델을 최적화 하는 방법

□ DQN 알고리즘

- ▣ 게임을 진행하는 모델과 보상을 학습하는 모델을 분리
- ▣ 진행했던 게임 기록으로 보상이 발생한 행동을 학습하는 기법

내용 정리

- 에이전트
 - ▣ 게임을 플레이하는 행동 모델
- 리플레이
 - ▣ 이전에 진행했던 게임의 상태 기록, 에이전트 행동 기록, 보상 기록
- OpenAI Gym
 - ▣ 강화학습에 대해 쉽게 배우고 개발할 수 있도록 그래픽 환경을 제공하는 패키지

연습문제

□ 문제 43. 다음 문장들을 읽고 빈 칸을 채워보세요.

- A. 신경망 모델이 출력한 결과를 비교해 가중치를 조절하는 과정을 한다.
- B. 강화 학습은 최대화하도록 학습한다.
- C. DQN에서 게임을 플레이하는 모델을 한다.
- D. DQN은 상태, 행동, 보상 기록을 통해 학습하고, 이 기록들을 한다.

연습문제

- 문제 44. 다음 코드는 Gym 라이브러리를 이용하여 강화 학습 환경을 구현한 코드입니다. 질문을 읽고 답해보세요.

```
01: import gym
02: from pop import Util
03:
04: env = gym.make('CartPole-v1')
05: for i_episode in range(20):
06:     state = env.reset()
07:     for t in range(100):
08:         img = env.render(mode = 'rgb_array')
```

```
09:         Util.imshow("CartPole", img, mode='RGB')
10:
11:         action = env.action_space.sample()
12:         state, reward, done, _ = env.step(action)
13:         if done:
14:             print("Episode finished after {} timesteps".format(t+1))
15:             break
16: env.close()
```

연습문제

- ▣ A. 코드를 실행해보고 게임 20회의 최대 보상값과 평균 보상값을 답해보세요.
- ▣ B. Pop.AI 라이브러리의 DQN 클래스를 사용해 보상을 최대화하는 코드를 작성해보세요.
- ▣ C. B에서 작성한 코드를 실행해 최대 보상값과 평균 보상값을 답해보세요.

AIoT SerBot Prime X로 배우는

온디바이스 AI 프로그래밍

Tensorflow

□ 텐서플로우

- ▣ 구글에서 제공하는 머신러닝 프레임워크
- ▣ 가벼운 활성화 함수부터 어려운 이론까지 사용하기 쉬운 API로 제공
- ▣ Tensorflow 1: Tensorflow Core를 기반으로 동작
- ▣ Tensorflow 2 :주로 Keras를 기반으로 동작. 다음 장에서 설명
- ▣ Tensorflow 실습을 위해 Tensorflow 2를 비활성화, Tensorflow 1 활성화

```
01:         import tensorflow.compat.v1 as tf
02:         tf.disable_v2_behavior()
```

상수, 변수와 플레이스 홀더

□ Tensorflow의 특별한 연산 방법 실습

- ▣ Tensorflow 라이브러리 import

- ▣ 상수와 변수를 생성한 뒤 출력

- 파이썬의 상수, 변수와 다르게 값이 출력되지 않고 객체 출력

```
01:         import tensorflow.compat.v1 as tf
02:         tf.disable_v2_behavior()
03:
04:         C = tf.constant(10)
05:         V = tf.Variable(5)
06:
07:         print(C)
08:         print(V)
```

상수, 변수와 플레이스 홀더

- ▣ 다음 소스 코드를 추가한 뒤 실행 결과 확인
 - 상수, 변수의 값이 출력되는 것을 확인할 수 있음

```
09:         sess = tf.Session()
10:         sess.run(tf.global_variables_initializer())
11:
12:         print(sess.run(C))
13:         print(sess.run(V))
```

상수, 변수와 플레이스 홀더

□ 세션

- ▣ 프로그램의 흐름과는 다른 연산 흐름
- ▣ 세션에서 연산을 시작해야 결과가 출력
- ▣ 세션의 상수, 변수는 파이썬의 상수, 변수와 같은 역할
 - 상수: 처음 선언에서 절대 변하지 않음
 - 변수: 사용자나 세션에 의해 변할 수 있음
 - 변수는 `global_variables_initializer()` 메소드로 초기화 필요

상수, 변수와 플레이스 홀더

□ 플레이스 홀더

- 플레이스 홀더 : 세션 연산에 사용될 값의 자리를 미리 잡아놓는 역할
- 세션의 feed_dict 파라미터 : 프로그램의 입력을 세션의 플레이스 홀더로 전달

```
01: import tensorflow.compat.v1 as tf
02: tf.disable_v2_behavior()
03:
04: C = tf.constant(10)
05: V = tf.Variable(5)
06: X = tf.placeholder(tf.int32)
07:
08: sess = tf.Session()
09: sess.run(tf.global_variables_initializer())
```

```
10:
11: F1 = C * X
12: F2 = V * X
13:
14: R1 = sess.run(F1, feed_dict = {X : 2})
15: R2 = sess.run(F2, feed_dict = {X : 5})
16:
17:
18: print(R1)
19: print(R2)
```

그래프와 세션

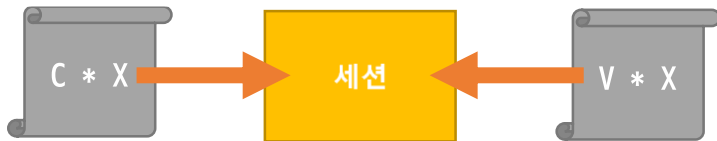
□ 그래프와 세션

▣ 그래프

- 하나의 텐서 연산 묶음
- 상수, 변수, 플레이스 홀더 등을 연산하는 하나의 과정

▣ 세션

- 하나의 연산 흐름
- 그래프를 연산



최적화 함수

□ 최적화 과정

- ▣ 결과를 비교해 가중치를 조절
- ▣ 텐서플로우에서는 이 과정을 간단한 객체와 메소드로 제공
 - 경사하강법, Adam, Nadam 등 다양한 최적화 제공
- ▣ 경사하강법 예제
 - optimizer 변수에 GradientDescentOptimizer 객체를 생성
 - minimize() 메소드를 이용해 최적화를 연산 그래프 생성

```
01:         optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
02:         train_op = optimizer.minimize(loss)
```

선형 회귀

□ 선형 회귀

- 변수 W 와 B 를 조절하여 입력 데이터들에 대해 최적의 가설 H 를 추측

- 선형 회귀의 기반은 1차 함수

$$Y = W \cdot X + B$$

- 입력되는 값은 X , 입력에 대한 결과값은 Y

- 회귀 모델은 W 와 B 만 조절 가능

- 모델은 W 와 B 를 조절해 입력 X 에 대한 출력과 실제 결과값 Y 를 비교

- 최적의 W 와 B 를 찾음

선형 회귀

```
01: import tensorflow.compat.v1 as tf
02:     tf.disable_v2_behavior()
03:
04:     W = tf.Variable(1.)
05:     B = tf.Variable(0.)
06:     X = tf.placeholder(tf.float32)
07:     Y = tf.placeholder(tf.float32)
```

선형 회귀

- ▣ 가설 모델 H 를 생성
- ▣ 모델 H 가 출력하는 결과와 실제 결과와의 오차를 계산하는 그래프 Loss 생성
 - `reduce_mean()` 메소드는 최하위 차원의 값들의 평균을 구해 차원을 줄여나가는 메소드

```
08:         H = W * X + B
09:         Loss = tf.reduce_mean(tf.reduce_mean(tf.abs(Y - H)))
```

- ▣ Loss를 최소화하는 방법으로 W 와 B 를 조절
- ▣ 경사하강법 객체를 생성하고 최소화할 대상을 Loss로 지정

```
10:         optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
11:         train_op = optimizer.minimize(Loss)
```

선형 회귀

- ▣ 세션을 생성하고 텐서플로우 변수 초기화

```
12:         sess = tf.Session()
13:         sess.run(tf.global_variables_initializer())
```

- ▣ 간단한 데이터셋을 만들고 train_op에 생성된 최적화 함수를 100회 반복 실행
- ▣ 최적화 함수가 끝날 때마다 변화된 오차 Loss 출력

```
14:         X_data = [[0],[1],[2],[3],[4]]
15:         Y_data = [[0],[2],[4],[6],[8]]
16:
17:         for i in range(100):
18:             sess.run(train_op, feed_dict={X : X_data, Y : Y_data})
19:             loss = sess.run(Loss, feed_dict={X : X_data, Y : Y_data})
20:             print(loss)
```

선형 회귀

- 최적화가 끝난 가설 모델 H 에 임의의 데이터 n 을 입력하여 결과를 확인

```
21:         n = [[7], [-10], [358]]
22:         result = sess.run(H, feed_dict={X : n})
23:         print(result)
```

- 입력 데이터와 출력 데이터를 학습해 $2 \cdot X + 0$ 에 근접하게 예측한 것 확인

선형 회귀

□ 전체 코드

```
01: import tensorflow.compat.v1 as tf
02: tf.disable_v2_behavior()
03:
04: W = tf.Variable(1.)
05: B = tf.Variable(0.)
06: X = tf.placeholder(tf.float32)
07: Y = tf.placeholder(tf.float32)
08:
09: H = W * X + B
10: Loss = tf.reduce_mean(tf.reduce_mean(tf.abs(Y - H)))
11:
12: optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
13: train_op = optimizer.minimize(Loss)
14:
```

```
15: sess = tf.Session()
16: sess.run(tf.global_variables_initializer())
17:
18: X_data = [[0],[1],[2],[3],[4]]
19: Y_data = [[0],[2],[4],[6],[8]]
20:
21: for i in range(1000):
22:     sess.run(train_op, feed_dict={X : X_data, Y : Y_data})
23:     loss = sess.run(Loss, feed_dict={X : X_data, Y : Y_data})
24:     print(loss)
25:
26: n = [[7], [-10], [358]]
27: result = sess.run(H, feed_dict={X : n})
28: print(result)
```

인공신경망

- 인공신경망

- ▣ 은닉층과 출력층으로 구성

- 은닉층 : 입력 데이터에 가중치를 곱함

- 출력층 : 은닉층 데이터에 다시 가중치를 곱하여 최종 결과값 출력

- ▣ 행렬곱 메소드 `tf.matmul`을 사용하여 구현

인공신경망

□ 인공신경망 예제

- ▣ 플레이스 홀더를 생성하고 2차원 텐서로 가중치를 생성
- ▣ 입력 가중치 텐서의 형태는 [input size, hidden size]로 지정
- ▣ 은닉층 가중치 텐서의 형태는 [hidden size, output size]로 지정
 - random_uniform 메소드는 특정한 형태로 랜덤 값 생성

```
01:         import tensorflow.compat.v1 as tf
02:         tf.disable_v2_behavior()
03:
04:         X = tf.placeholder(tf.float32)
05:         Y = tf.placeholder(tf.float32)
06:         W1 = tf.Variable(tf.random_uniform([1, 10], -1., 1.))
07:         W2 = tf.Variable(tf.random_uniform([10, 1], -1., 1.))
```

인공신경망

- ▣ 입력층 → 은닉층, 은닉층 → 출력층 구간
 - 행렬곱 메소드 `tf.matmul()`을 이용해 수식 만들
 - `tf.nn.relu()` 메소드를 이용해 ReLu 활성화 함수 적용
 - 단, 은닉층 → 출력층 구간은 최종값이 나오는 구간이므로 이 신경망의 모델이 됨
- ▣ 각각 `L`, `model`이라는 이름으로 생성

```
08:         L = tf.matmul(X, W1)
09:         L = tf.nn.relu(L)
10:         model = tf.matmul(L, W2)
11:         Loss = tf.reduce_mean(tf.reduce_mean(tf.square(Y - model)))
```

인공신경망

- 최적화 함수로 W1과 W2를 조절
- 경사하강법 객체 생성하고 최소화할 대상을 Loss로 지정

```
12:         optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
13:         train_op = optimizer.minimize(Loss)
```

- 세션을 생성하고 텐서플로우 변수를 초기화

```
14:         sess = tf.Session()
15:         sess.run(tf.global_variables_initializer())
```

인공신경망

- ▣ 간단한 데이터셋을 만들고 train_op에 생성된 최적화 함수를 500회 반복 실행
- ▣ 최적화 함수가 끝날 때마다 변화된 오차 Loss 출력

```
16:         X_data = [[-4],[-3],[-2],[-1],[0],[1],[2],[3],[4]]
17:         Y_data = [[-8],[-6],[-4],[-2],[0],[2],[4],[6],[8]]
18:
19:         for i in range(500):
20:             sess.run(train_op, feed_dict={X : X_data, Y : Y_data})
21:             loss = sess.run(Loss, feed_dict={X : X_data, Y : Y_data})
22:             print(loss)
```

- ▣ 최적화가 끝난 신경망 모델에 임의의 데이터 n을 입력하여 결과 확인

```
23:         n = [[7], [-10], [358]]
24:         result = sess.run(model, feed_dict={X : n})
25:         print(result)
```

인공신경망

□ 전체 코드

```
01: import tensorflow.compat.v1 as tf
02: tf.disable_v2_behavior()
03:
04: X = tf.placeholder(tf.float32)
05: Y = tf.placeholder(tf.float32)
06: W1 = tf.Variable(tf.random_uniform([1, 10], -1., 1.))
07: W2 = tf.Variable(tf.random_uniform([10, 1], -1., 1.))
08:
09: L = tf.matmul(X, W1)
10: L = tf.nn.relu(L)
11: model = tf.matmul(L, W2)
12: Loss = tf.reduce_mean(tf.reduce_mean(tf.square(Y - model),axis=1))
13:
14: optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
15: train_op = optimizer.minimize(Loss)
```

```
16:
17: sess = tf.Session()
18: sess.run(tf.global_variables_initializer())
19:
20: X_data = [[-4],[-3],[-2],[-1],[0],[1],[2],[3],[4]]
21: Y_data = [[-8],[-6],[-4],[-2],[0],[2],[4],[6],[8]]
22:
23: for i in range(500):
24:     sess.run(train_op, feed_dict={X : X_data, Y : Y_data})
25:     loss = sess.run(Loss, feed_dict={X : X_data, Y : Y_data})
26:     print(loss)
27:
28: n = [[7], [-10], [358]]
29: result = sess.run(model, feed_dict={X : n})
30: print(result)
```


심층신경망

- 심층신경망

- ▣ 인공신경망에서 은닉층의 수를 늘려 구현 가능

- 은닉층 → 은닉층 구간을 중간에 추가

심층신경망

□ 심층신경망 예제

- 플레이스 홀더를 생성하고 2차원 텐서로 가중치 생성
- 입력층 → 은닉층 구간의 가중치는 $W1$
- 은닉층 → 은닉층 구간의 가중치는 $W2$
- 은닉층 → 출력층 구간의 가중치는 $W3$

심층신경망

```
01:         import tensorflow.compat.v1 as tf
02:         tf.disable_v2_behavior()
03:
04:         X = tf.placeholder(tf.float32)
05:         Y = tf.placeholder(tf.float32)
06:         W1 = tf.Variable(tf.random_uniform([1, 10], -1., 1.))
07:         W2 = tf.Variable(tf.random_uniform([10, 10], -1., 1.))
08:         W3 = tf.Variable(tf.random_uniform([10, 1], -1., 1.))
```

심층신경망

- ▣ 행렬곱을 이용해 각 구간의 수식을 만듦
 - 은닉층 → 출력층 구간은 최종값이 나오는 구간이므로 이 신경망의 모델이 됨
- ▣ 각각 L1, L2, model이라는 이름으로 생성

```
09:         L1 = tf.matmul(X, W1)
10:         L1 = tf.nn.relu(L1)
11:         L2 = tf.matmul(L1, W2)
12:         L2 = tf.nn.relu(L2)
13:         model = tf.matmul(L2, W3)
14:         Loss = tf.reduce_mean(tf.reduce_mean(tf.square(Y - model)))
```

심층신경망

- 최적화 함수로 W1, W2, W3를 조절
- 경사하강법 객체 생성하고 최소화할 대상을 Loss로 지정

```
15:         optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
16:         train_op = optimizer.minimize(Loss)
```

- 세션을 생성하고 텐서플로우 변수 초기화

```
17:         sess = tf.Session()
18:         sess.run(tf.global_variables_initializer())
```

심층신경망

- 간단한 데이터셋을 만들고 train_op에 생성된 최적화 함수를 500회 반복 실행
- 최적화 함수가 끝날 때마다 변화된 오차 Loss 출력

```
19:         X_data = [[0],[1],[2],[3],[4]]
20:         Y_data = [[0],[2],[4],[6],[8]]
21:
22:         for i in range(500):
23:             sess.run(train_op, feed_dict={X : X_data, Y : Y_data})
24:             loss = sess.run(Loss, feed_dict={X : X_data, Y : Y_data})
25:             print(loss)
```

- 최적화가 끝난 신경망 모델에 임의의 데이터 n을 입력하여 결과 확인

```
26:         n = [[7], [-10], [358]]
27:         result = sess.run(model, feed_dict={X : n})
28:         print(result)
```

심층신경망

□ 전체 코드

```
01: import tensorflow.compat.v1 as tf
02: tf.disable_v2_behavior()
03:
04: X = tf.placeholder(tf.float32)
05: Y = tf.placeholder(tf.float32)
06: W1 = tf.Variable(tf.random_uniform([1, 10], -1., 1.))
07: W2 = tf.Variable(tf.random_uniform([10, 10], -1., 1.))
08: W3 = tf.Variable(tf.random_uniform([10, 1], -1., 1.))
09:
10: L1 = tf.matmul(X, W1)
11: L1 = tf.nn.relu(L1)
12: L2 = tf.matmul(L1, W2)
13: L2 = tf.nn.relu(L2)
14: model = tf.matmul(L2, W3)
15: Loss = tf.reduce_mean(tf.reduce_mean(tf.square(Y - model)))
16: optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
```

```
17: train_op = optimizer.minimize(Loss)
18:
19: sess = tf.Session()
20: sess.run(tf.global_variables_initializer())
21:
22: X_data = [[-4],[-3],[-2],[-1],[0],[1],[2],[3],[4]]
23: Y_data = [[-8],[-6],[-4],[-2],[0],[2],[4],[6],[8]]
24:
25: for i in range(500):
26:     sess.run(train_op, feed_dict={X : X_data, Y : Y_data})
27:     loss = sess.run(Loss, feed_dict={X : X_data, Y : Y_data})
28:     print(loss)
29:
30: n = [[7], [-10], [358]]
31: result = sess.run(model, feed_dict={X : n})
32: print(result)
```

내용 정리

- 텐서플로우(Tensorflow)
 - ▣ 구글에서 제공하는 머신러닝 프레임워크
 - ▣ 프로그램의 흐름과는 다른 흐름인 '세션'에서 연산
- 플레이스 홀더
 - ▣ 세션 연산에 사용될 미지수의 공간을 예약
- 그래프
 - ▣ 하나의 텐서 연산 묶음. 연산자를 이용해 연산 과정을 표현

내용 정리

- 최적화

- ▣ 인공신경망이 올바른 출력을 위해 표본(정답) 데이터와 출력 데이터를 비교해 가중치를 조절하는 과정

- 최적화 함수

- ▣ 인공신경망의 복잡한 가중치 최적화 과정을 간단하게 구현한 함수

연습문제



- 문제 45. 다음은 코드 조각들입니다. Tensorflow 상수와 변수 값을 출력하고자 할 때 코드를 순서에 맞게 나열해보세요.

A	<pre>01: print(C) 02: print(V)</pre>
B	<pre>01: sess = tf.Session() 02: sess.run(tf.global_variables_initializer())</pre>
C	<pre>01: import tensorflow.compat.v1 as tf 02: tf.disable_v2_behavior()</pre>
D	<pre>01: C = tf.constant(13) 02: V = tf.Variable(1)</pre>

연습문제

- 문제 46. 다음은 Tensorflow 플레이스 홀더를 사용해 1차 함수 값을 출력하는 코드입니다. 정상적으로 출력되도록 빈 칸에 들어갈 코드를 작성해보세요.

```
01: import tensorflow.compat.v1 as tf
02: tf.disable_v2_behavior()
03:
04: C = tf.constant(13)
05: V = tf.Variable(1)
06: X = tf.placeholder(tf.int32)
07:
08: sess = tf.Session()
09: sess.run(tf.global_variables_initializer())
```



```
10:
11: F1 = C * X
12: F2 = V * X
13:
14: 
15: R2 = sess.run(F2, )
16:
17: print(R1)
18: print(R2)
```

연습문제

- 문제 47. 다음 수식과 코드는 선형 회귀 목표 가설과 Tensorflow로 선형 회귀 모델을 구현한 코드입니다. 질문을 읽고 답해보세요.

$$y = 2x + 1$$

```
01: import tensorflow.compat.v1 as tf
02: tf.disable_v2_behavior()
03:
04: W = tf.Variable(1.)
05: B = tf.Variable(1.)
06: X = tf.placeholder(tf.float32)
07: Y = tf.placeholder(tf.float32)
08:
09: H = W * X + B
10: Loss = tf.reduce_mean(tf.reduce_mean(tf.abs(Y - H)))
11:
12: optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
```

```
13: train_op = optimizer.minimize(Loss)
14:
15: sess = tf.Session()
16: sess.run(tf.global_variables_initializer())
17:
18: X_data = 
19: Y_data = 
20:
21: for i in range(1000):
22:     sess.run(train_op, feed_dict={X : X_data, Y : Y_data})
23:     loss = sess.run(Loss, feed_dict={X : X_data, Y : Y_data})
24:     print(loss)
```

연습문제

- ▣ A. 수식을 회귀할 수 있도록 빈 칸 X, Y에 들어갈 학습 데이터셋을 작성해보세요.
- ▣ B. 선형 회귀 모델에 100, 1000, 10000를 입력했을 때의 출력을 작성해보세요.
- ▣ C. 10000를 입력했을 때 오차가 ± 0.01 이하인 모델을 만들고 손실율을 작성해보세요.

연습문제

- 문제 48. 다음 코드들은 각각 Tensorflow를 이용한 선형 회귀 모델을 구현한 코드와 Cds센서를 이용하여 밝기 값을 출력하는 코드입니다. 조도계를 사용하거나, 스마트폰에서 '조도계' 애플리케이션을 다운 받아 다음 문제를 해결해보세요. (단, 조도계의 단위는 Lux로 합니다.)

연습문제

```
01: import tensorflow.compat.v1 as tf
02: tf.disable_v2_behavior()
03:
04: W = tf.Variable(1.)
05: B = tf.Variable(0.)
06: X = tf.placeholder(tf.float32)
07: Y = tf.placeholder(tf.float32)
08:
09: H = W * X + B
10: Loss = tf.reduce_mean(tf.reduce_mean(tf.abs(Y - H)))
11:
12: optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.001)
13: train_op = optimizer.minimize(Loss)
14:
15: sess = tf.Session()
16: sess.run(tf.global_variables_initializer())
```

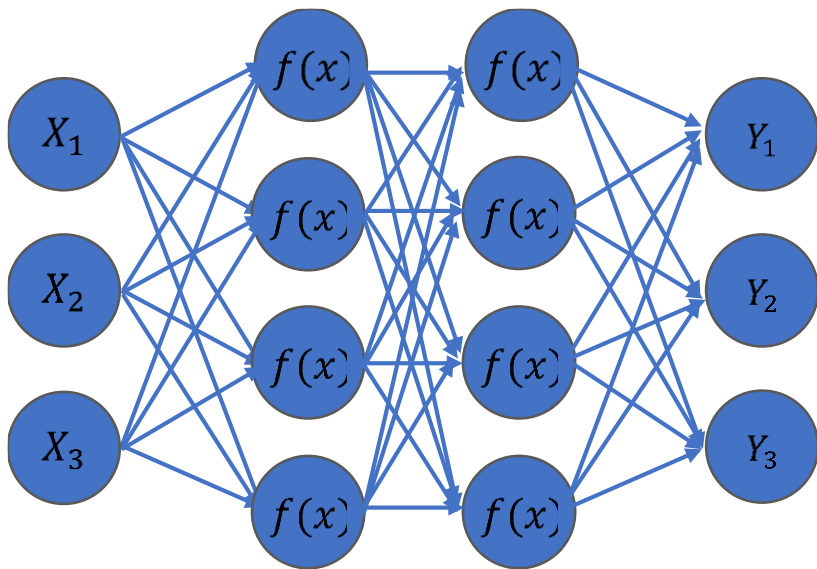
```
01: from pop import Cds
02:
03: cds = Cds(7)
04:
05: value = cds.readAverage()
06: print(value)
```

연습문제

- ▣ A. 빈 배열 2개를 생성하고, Cds 값과 조도계 값을 동시에 측정하여 Cds 값 배열과 조도계 값 배열을 만들어보세요.
- ▣ B. Tensorflow를 이용하여 A에서 만든 두 배열을 선형 회귀하는 코드를 작성하세요.
- ▣ C. 회귀 모델의 출력과 실제 조도계의 값을 비교해보고 데이터셋 추가 수집, 추가 학습 등 방법으로 ± 30 lux 미만의 오차 범위를 갖는 회귀 모델을 만들어보세요.

연습문제

- 문제 49. 다음 신경망 그림을 Tensorflow로 구현해보세요.



AIoT SerBot Prime X로 배우는

온디바이스 AI 프로그래밍

Keras

□ Keras

- ▣ 딥러닝 구현에 특화된 고수준 신경망 API
- ▣ 구글의 Tensorflow2 프레임워크와 함께 사용 가능
- ▣ CNN, RNN 등 복잡한 신경망을 쉬운 API로 제공
- ▣ 직관적인 구조로 되어 있어, 다양한 신경망을 쉽게 설계 가능

모델과 레이어

- 모델

- ▣ 입력층, 은닉층, 출력층 등 레이어가 연결되어 이루는 하나의 레이어 연결체

- 레이어

- ▣ 모델을 구성하는 하나의 연산 계층
 - ▣ 가중치를 조절하여 모델 학습

모델과 레이어

- Keras를 이용하면 직관적으로 레이어를 연결해 학습 모델 생성 가능
- Keras의 Sequential() 메소드로 순차 모델 생성 가능
- 순차 모델에 add() 메소드로 레이어를 추가
 - ▣ 자동으로 가중치를 생성해 레이어 연결

모델과 레이어

□ Keras 실습 예제

- Tensorflow에서 라이브러리 import

- 간단한 모델을 만들어 임의 값을 입력하고 출력 확인

```
01:         from tensorflow import keras
02:
03:         model = keras.models.Sequential()
04:
05:         model.add(keras.layers.Input(shape=(1,)))
06:         model.add(keras.layers.Dense(50))
07:         model.add(keras.layers.Dense(10))
08:
09:         value = model.predict([1])
10:         print(value)
```

모델과 레이어

▣ summary 메소드를 통해 모델의 구조 확인

```
11:         model.summary()
```

[출력]

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 50)	100
dense_1 (Dense)	(None, 10)	510

Total params: 610

Trainable params: 610

Non-trainable params: 0

- 출력에는 입력층은 표시되지 않고 은닉층과 출력층만 표시
- 각 구간별 가중치의 수를 Parameter에 표시

컴파일

□ Keras의 컴파일

▣ 학습 과정을 명시하는 것

- 어떤 방식과 어떤 손실 함수를 이용할 것인지 명시

▣ model 객체의 compile() 메소드로 컴파일

▣ 손실 함수, 최적화 함수 등을 선택해 모델의 학습 방향을 지정하는 역할

▣ 손실 함수로 MSE, 최적화 함수로 SGD를 사용해 컴파일

- MSE : 평균 제곱 오차, SGD : 경사하강법

```
model.compile(loss='MSE', optimizer='SGD')
```

컴파일

- 대표적으로 많이 사용하는 손실 함수, 최적화 함수
 - ▣ 손실함수
 - `binary_crossentropy`: 이진 교차 엔트로피
 - `categorical_crossentropy`: 다항 교차 엔트로피
 - ▣ 최적화 함수
 - SGD: 경사 하강법
 - Adam, Nadam, RMSProp 등

선형 회귀

□ Keras의 선형 회귀

- 입력층과 출력층의 노드 : 1개
- 평균 제곱 오차와 경사하강법으로 컴파일
- model의 fit 메소드를 사용해 컴파일한대로 모델 학습
- 기본적인 선형 회귀 모델을 목표로 X와 Y데이터를 입력
- epochs에 학습 횟수를 입력하여 fit 메소드 호출

선형 회귀

```
01:         from tensorflow import keras
02:
03:         model = keras.models.Sequential()
04:
05:         model.add(keras.layers.Input(shape=(1,)))
06:         model.add(keras.layers.Dense(1))
07:
08:         model.compile(loss='MSE', optimizer='SGD')
09:
10:         X=[[0],[1],[2],[3],[4],[5]]
11:         Y=[[0],[2],[4],[6],[8],[10]]
12:
13:         model.fit(X, Y, epochs=100)
```

선형 회귀

- ▣ 학습이 끝나면 model의 predict() 메소드를 이용해 모델 출력 확인

```
14:         value = model.predict([[7], [-10], [358]])  
15:         print(value)
```

선형 회귀

□ 전체 코드

```
01:         from tensorflow import keras
02:
03:         model = keras.models.Sequential()
04:
05:         model.add(keras.layers.Input(shape=(1,)))
06:         model.add(keras.layers.Dense(1))
07:
08:         model.compile(loss='MSE', optimizer='SGD')
09:
10:         X=[[0],[1],[2],[3],[4],[5]]
11:         Y=[[0],[2],[4],[6],[8],[10]]
12:
13:         model.fit(X, Y, epochs=100)
14:
15:         value = model.predict([[7], [-10], [358]])
16:         print(value)
```

로지스틱 회귀

- 로지스틱 회귀
 - ▣ 입력에 대해 0 ~ 1을 출력하는 회귀 모델
 - ▣ 기본적인 구성은 선형 회귀 모델과 같음
 - ▣ 활성화 함수, 손실 함수, 최적화 함수를 조절해 구현 가능

로지스틱 회귀

- 로지스틱 회귀 예제
 - ▣ 출력층의 활성화 함수를 Sigmoid로 지정
 - ▣ 컴파일 및 학습
 - 손실 함수 : `binary_crossentropy`, 최적화 함수 : `Adam`
 - ▣ 모델은 음수와 양수를 구분하는 것을 목표로 X와 Y데이터를 입력
 - ▣ `epochs`에 학습 횟수를 입력하여 `fit()` 메소드 호출

로지스틱 회귀

```
01:         from tensorflow import keras
02:
03:         model = keras.models.Sequential()
04:
05:         model.add(keras.layers.Input(shape=(1,)))
06:         model.add(keras.layers.Dense(1, activation='sigmoid'))
07:
08:         model.compile(loss='binary_crossentropy', optimizer='Adam')
09:
10:         X=[[-3],[-2],[-1],[1],[2],[3]]
11:         Y=[[0],[0],[0],[1],[1],[1]]
12:
13:         model.fit(X, Y, epochs=100)
```

로지스틱 회귀

- ▣ 학습이 끝나면 model의 predict() 메소드를 이용해 모델 출력 확인

```
14:         value = model.predict([[7], [-10], [358]])  
15:         print(value)
```

로지스틱 회귀

□ 전체 코드

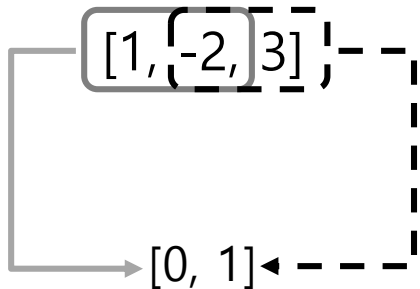
```
01:         from tensorflow import keras
02:
03:         model = keras.models.Sequential()
04:
05:         model.add(keras.layers.Input(shape=(1,)))
06:         model.add(keras.layers.Dense(1, activation='sigmoid'))
07:
08:         model.compile(loss='binary_crossentropy', optimizer='Adam')
09:
10:         X=[[-3],[-2],[-1],[1],[2],[3]]
11:         Y=[[0],[0],[0],[1],[1],[1]]
12:
13:         model.fit(X, Y, epochs=100)
14:
15:         value = model.predict([[7], [-10], [358]])
16:         print(value)
```

심층신경망

□ 심층신경망 실습

■ 심층신경망의 학습 능력을 실험해보기 위해 비교적 복잡한 데이터셋을 준비

- 3개의 입력 데이터
- 1번째 원소와 2번째 원소의 크기를 비교한 결과를 1번째 출력
- 2번째 원소와 3번째 원소의 크기를 비교한 결과를 2번째 출력
- 각 출력은 좌항이 크면 0, 우항이 크면 1을 출력



심층신경망

- 입력층

- 노드 3개

- 은닉층

- 노드 100개, 레이어 4개
- Relu 활성화 함수 사용

- 출력층

- 노드 2개
- sigmoid 활성화 함수 사용

심층신경망

- ▣ 컴파일
 - 손실함수: `binary_crossentropy`
 - 최적화 함수: `sigmoid`
- ▣ 심층신경망이 조금만 학습해도 충분한 학습이 이루어지는지 실험
 - 20회 학습

심층신경망

```
01:         from tensorflow import keras
02:
03:         model = keras.models.Sequential()
04:
05:         model.add(keras.layers.Input(shape=(3,)))
06:         model.add(keras.layers.Dense(100, activation='relu'))
07:         model.add(keras.layers.Dense(100, activation='relu'))
08:         model.add(keras.layers.Dense(100, activation='relu'))
09:         model.add(keras.layers.Dense(100, activation='relu'))
10:         model.add(keras.layers.Dense(2, activation='sigmoid'))
11:
12:         model.compile(loss='binary_crossentropy', optimizer='Adam')
13:
14:         X=[[1,-2,3],[2,-3,4],[3,5,4],[4,-5,6],[7,-5,3], [1,6,8], [3,8,1]]
15:         Y=[[0, 1],[0, 1],[1, 0],[0, 1],[0, 1],[1, 1],[1, 0]]
16:
17:         model.fit(X, Y, epochs=20)
```

심층신경망

- ▣ 학습이 끝나면 model의 predict 메소드를 이용해 모델 출력 확인

```
18:         value = model.predict([[10,-5,3], [-1,6,-8], [3,5,8]])  
19:         print(value)
```

심층신경망

□ 전체 코드

```
01: from tensorflow import keras
02:
03: model = keras.models.Sequential()
04:
05: model.add(keras.layers.Input(shape=(3,)))
06: model.add(keras.layers.Dense(100, activation='relu'))
07: model.add(keras.layers.Dense(100, activation='relu'))
08: model.add(keras.layers.Dense(100, activation='relu'))
09: model.add(keras.layers.Dense(100, activation='relu'))
10: model.add(keras.layers.Dense(2, activation='sigmoid'))
```

```
11:
12: model.compile(loss='binary_crossentropy', optimizer='Adam')
13:
14: X=[[1,-2,3],[2,-3,4],[3,5,4],[4,-5,6],[7,-5,3], [1,6,8], [3,8,1]]
15: Y=[[0, 1],[0, 1],[1, 0],[0, 1],[0, 1],[1, 1],[1, 0]]
16:
17: model.fit(X, Y, epochs=20)
18:
19: value = model.predict([[10,-5,3], [-1,6,-8], [3,5,8]])
20: print(value)
```


내용 정리

- 케라스(Keras)
 - ▣ 딥러닝에 특화된 고수준 신경망 API 라이브러리
- 모델
 - ▣ 입력층, 은닉층, 출력층이 연결된 하나의 레이어 연결체
- 레이어
 - ▣ 모델을 구성하는 하나의 연산 계층
- 컴파일
 - ▣ 모델이 어떻게 학습할지 명시하는 과정.
 - ▣ 손실 함수, 최적화 함수 등을 설정.

연습문제

- 문제 50. 다음은 코드 조각들입니다. Keras로 3개의 입력에 대해 4개의 출력을 하는 모델을 작성하고자 할 때 코드를 순서에 맞게 나열하세요.

A	<pre>01: model = keras.models.Sequential()</pre>
B	<pre>01: model.add(keras.layers.Input(shape=(3,)))</pre>
C	<pre>01: from tensorflow import keras</pre>
D	<pre>01: model.add(keras.layers.Dense(4))</pre>

연습문제

- 문제 51. 어떤 모델 A를 평균 제곱 오차로 손실율을 계산하고, 경사하강법으로 최적화하는 모델로 컴파일하려고 할 때 빈 칸에 들어갈 내용을 작성하세요.

```
01:         A.compile( )
```

연습문제

- 문제 52. 다음 코드는 Keras로 로지스틱 회귀 모델을 구현한 코드입니다. 질문을 읽고 답하세요.

```
01:         from tensorflow import keras
02:
03:         model = keras.models.Sequential()
04:
05:         model.add(keras.layers.Input(shape=(1,)))
06:         model.add(keras.layers.Dense(1, activation='sigmoid'))
07:
08:         model.compile(loss='binary_crossentropy',
09:                       optimizer='Adam')
10:
11:         X=
12:         Y=
13:
14:         model.fit(X, Y, epochs=100)
```

연습문제

- ▣ A. 양수면 1, 음수면 0을 출력하도록 회귀하고자 할 때 빈 칸 X, Y에 들어갈 학습 데이터셋을 작성하세요.
- ▣ B. 로지스틱 회귀 모델에 -1, 1001, -10000를 입력했을 때의 출력을 작성해보세요.
- ▣ C. 0.1을 입력했을 때 오차가 ± 0.01 이하인 모델을 만들어보고 모델의 손실율을 작성해보세요.

연습문제

- 문제 53. 다음 코드들은 각각 Keras를 이용한 선형 회귀 모델을 구현한 코드와 Cds센서를 이용하여 밝기 값을 출력하는 코드입니다. 조도계를 사용하거나, 스마트폰에서 '조도계' 애플리케이션을 다운 받아 다음 문제를 해결하세요. (단, 조도계의 단위는 Lux로 합니다.)

```
01:         from tensorflow import keras
02:
03:         model = keras.models.Sequential()
04:
05:         model.add(keras.layers.Input(shape=(1,)))
06:         model.add(keras.layers.Dense(1))
07:
08:         model.compile(loss='MAE', optimizer='Adam')
```

```
01:         from pop import Cds
02:
03:         cds = Cds(7)
04:
05:         value = cds.readAverage()
```

연습문제

- ▣ A. 빈 배열 2개를 생성하고, Cds 값과 조도계 값을 동시에 측정하여 Cds 값 배열과 조도계 값 배열을 만들어보세요.
- ▣ B. Keras를 이용하여 A에서 만든 두 배열을 선형 회귀하는 코드를 작성하세요.
- ▣ C. 회귀 모델의 출력과 실제 조도계의 값을 비교해보고 데이터셋 추가 수집, 추가 학습 등 방법으로 ± 30 lux 미만의 오차 범위를 갖는 회귀 모델을 만들어 보세요.

연습문제

- 문제 54. 다음 신경망 그림을 Keras로 구현해보세요.

