

AIoT AutoCar Prime 으로 배우는 온디바이스 AI 프로그래밍

7. 인공지능

7.4 Tensorflow

Tensorflow

- 텐서플로우

- ▣ 구글에서 제공하는 머신러닝 프레임워크
- ▣ 가벼운 활성화 함수부터 어려운 이론까지 사용하기 쉬운 API로 제공

- Tensorflow 1

- ▣ Tensorflow Core를 기반으로 동작
- ▣ Tensorflow 실습을 위해 Tensorflow 2를 비활성화, Tensorflow 1 활성화
- ▣ Tesnorflow 2 :주로 Keras를 기반으로 동작. 다음 장에서 설명

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

상수, 변수와 플레이스 홀더

□ Tensorflow의 특별한 연산 방법 실습

- 소스 코드를 입력해 Tensorflow 라이브러리 import
- 상수와 변수를 생성한 뒤 출력
- 파이썬의 상수, 변수와 다르게 값이 출력되지 않고 상태 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

- ▣ 세션이라는 개념이 있음
- ▣ 세션에서 연산을 시작해야 결과가 출력

```
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

- ▣ 텐서플로우는 프로그램의 흐름과는 다른 흐름인 ‘세션’ 에서 연산
- ▣ 변수 : `tf.global_variables_initializer()` 메소드로 변수를 초기화 후 연산 가능
- ▣ 텐서플로우의 상수, 변수는 파이썬의 상수, 변수와 같은 역할
 - 세션이라는 개념 제외
- ▣ 상수 : 처음 선언이후 불변
- ▣ 변수 : 사용자나 세션에 의해 가변

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: print(R1)
18: 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
- ▣ 경사하강법의 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 함수로 표현 가능

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

- 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 찾는

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 H에 임의의 데이터 n을 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 형태(Shape)는 [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, model이라는 이름으로 생성

```
08:         L = tf.matmul(X, W1) #input - hidden
09:         L = tf.nn.relu(L)
10:         model = tf.matmul(L, W2) #hidden - output
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) #input - hidden
10: L = tf.nn.relu(L)
11: model = tf.matmul(L, W2) #hidden - output
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) #input - hidden
10:         L1 = tf.nn.relu(L1)
11:         L2 = tf.matmul(L1, W2) #hidden - hidden
12:         L2 = tf.nn.relu(L2)
13:         model = tf.matmul(L2, W3) #hidden - output
14:         Loss = tf.reduce_mean(tf.reduce_mean(tf.sqaure(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) #input - hidden
11: L1 = tf.nn.relu(L1)
12: L2 = tf.matmul(L1, W2) #hidden - hidden
13: L2 = tf.nn.relu(L2)
14: model = tf.matmul(L2, W3) #hidden - output
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

- ▣ 인공지능망이 올바른 출력을 위해 표본데이터와 출력 데이터를 비교해 가중치를 조절하는 과정

- 최적화 함수

- ▣ 인공지능망의 복잡한 가중치 최적화 과정을 간단하게 구현한 함수

연습문제



- 문제 46. 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 47. 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:  A
15: R2 = sess.run(F2, ) B
16:
17: print(R1)
18: print(R2)
```

연습문제

- 문제 48. 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(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 = 
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 외귀할 수 있도록 빈 칸 X, Y에 들어갈 학습 데이터셋을 작성해보세요.
- ▣ B. 선형 외귀 모델에 100, 1000, 10000를 입력했을 때의 출력을 작성해보세요.
- ▣ C. 10000를 입력했을 때 오차가 ± 0.01 이하인 모델을 만들고 손실율을 작성해보세요.

연습문제

- 문제 49. 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 외귀하는 코드를 작성하세요.
- ▣ C. 외귀 모델의 출력과 실제 조도계의 값을 비교해보고 데이터셋 추가 수집, 추가 학습 등 방법으로 ± 30 lux 미만의 오차 범위를 갖는 외귀 모델을 만들어보세요.

연습문제

- 문제 50. 다음 신경망 그림을 Tensorflow로 구현해보세요.

