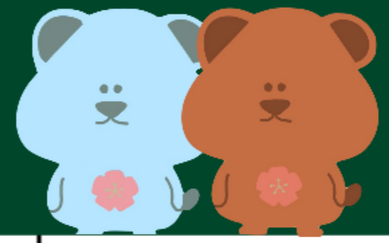
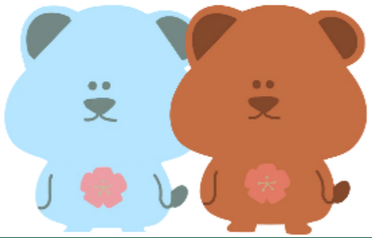


(7) 다중 클래스 분류
(8) 프로그래밍 프레임워크 소개

중급 6기 주민서

EWHA EURON

Softmax Regression

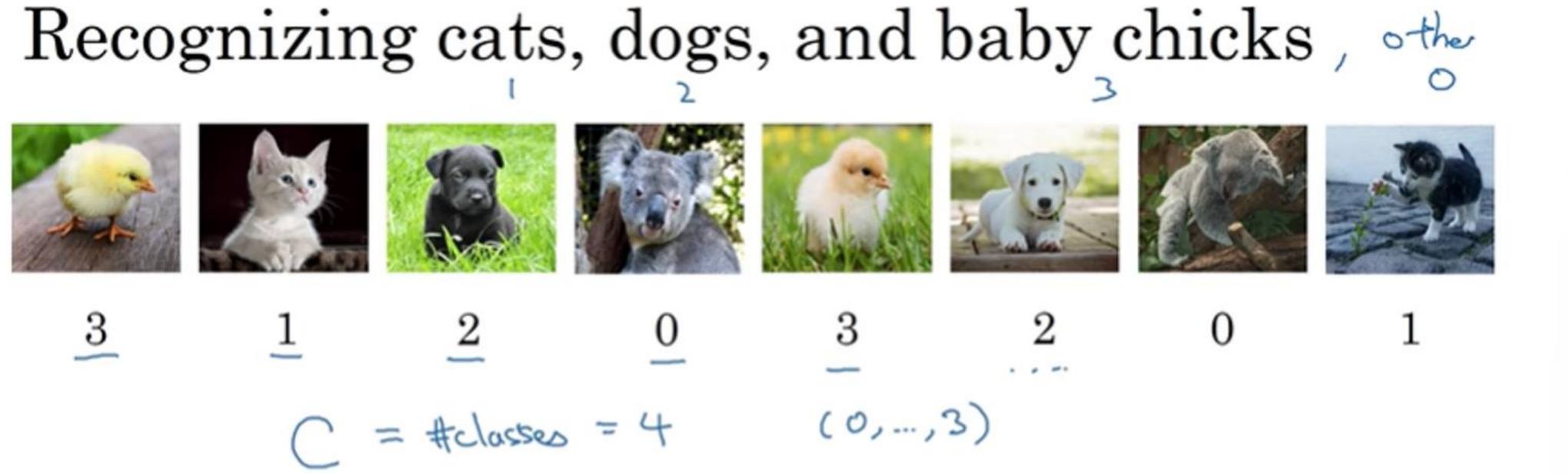


EWHA EURON

#01 Softmax Regression

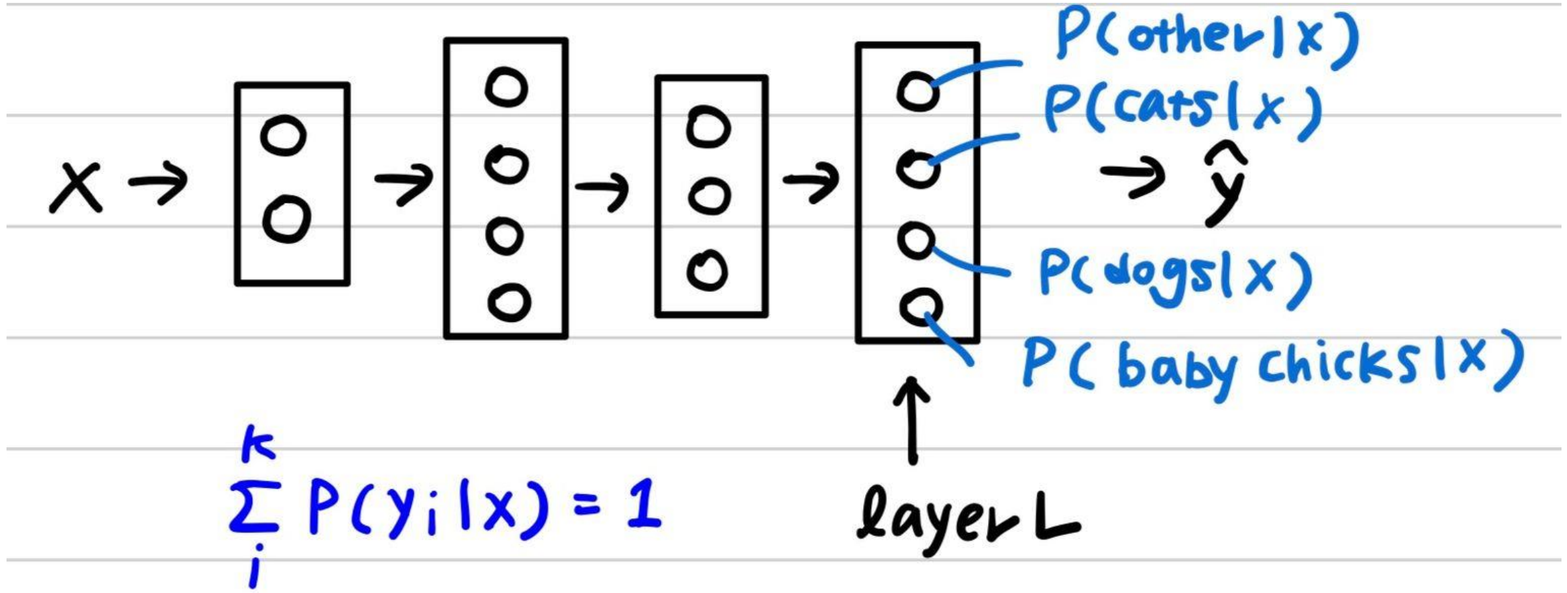
- Softmax Regression은 여러 개의 클래스를 분류할 시 사용된다.

- 예를 들어 cats, dogs, baby chicks의 3개의 클래스가 있고, 각각을 클래스 1, 2, 3이라고 하자. 셋 중 어떤 클래스에도 해당되지 않은 사진은 클래스 0으로 분류하자.



#01 Softmax Regression

- 아래와 같은 Neural Network가 있다면, 마지막 층 L의 각 노드의 값은 input X가 i번째 클래스에 속할 확률(probability)이다.



#01 Softmax Activation Function

- Softmax 층의 활성화 함수는 다음과 같이 정의된다.

$$z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]} \in R^{4\times 1}$$

Activation function

$$a^{[l]} = \frac{e^{z^{[l]}}}{\sum_{i=1}^{4} e^{z_i^{[l]}}} \in R^{4\times 1}$$

→ $a^{[l]}$ 의 크기를 1로 정규화시키기 위함

$$a_i^{[l]} = \frac{e^{z_i^{[l]}}}{\sum_{i=1}^{4} e^{z_i^{[l]}}}$$

ex) $z^{[l]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix}$

for 정규화

→ $e^{z^{[l]}} = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$ & $\sum_{i=1}^{4} e^{z_i^{[l]}} = 176.3$

→ $a^{[l]} = \frac{e^{z^{[l]}}}{176.3} = 1$

input X가 클래스 0에 속할 확률

$P(\text{other}|x) = \frac{e^5}{176.3} = 0.842$ → 84.2%

$P(\text{cats}|x) = \frac{e^2}{176.3} = 0.042$ → 4.2%

$P(\text{dogs}|x) = \frac{e^{-1}}{176.3} = 0.02$ → 2%

$P(\text{baby chicks}|x) = \frac{e^3}{176.3} = 0.114$ → 11.4%

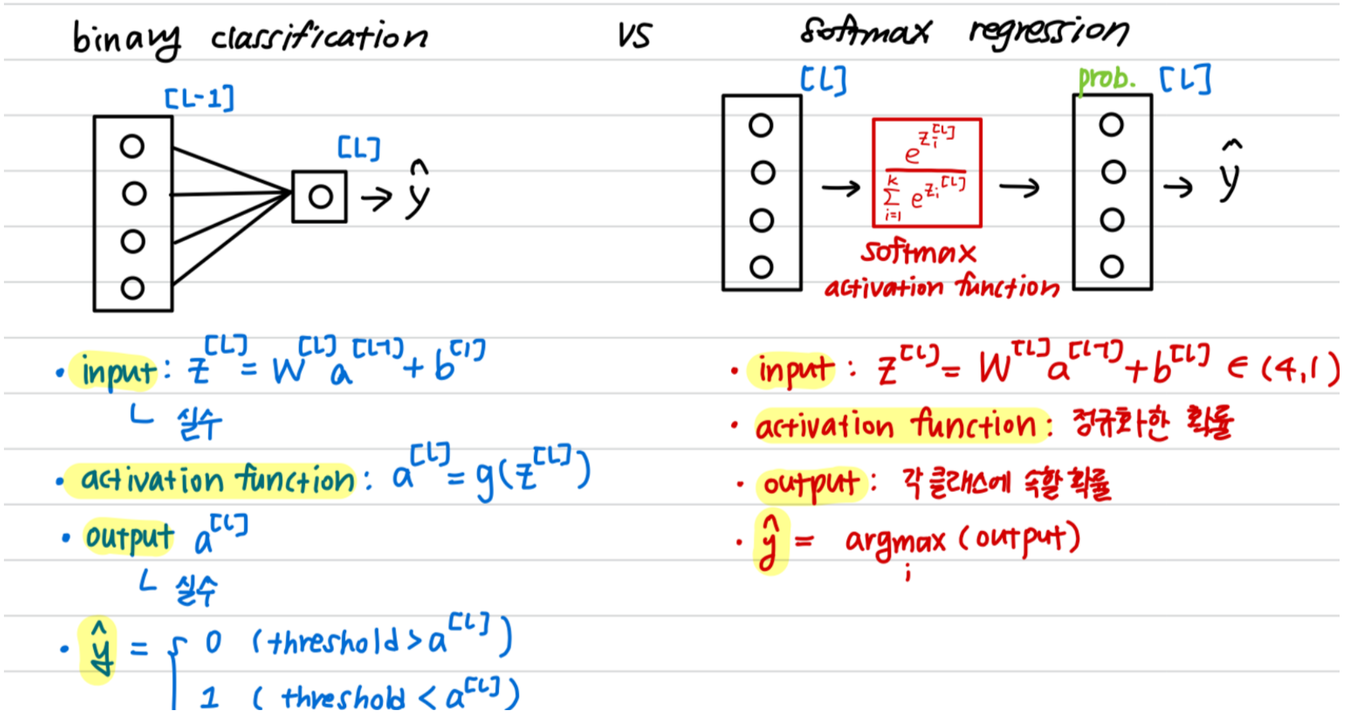
#01 Softmax Activation Function

- Softmax 활성화 함수의 특징은 실수가 아닌 벡터를 입력으로 받고 출력으로 내놓는다는 점이다. 수업에서는 이를 '정규화 '하기 위함이라고 했는데, 정확히 어떤 의미일까?

#01 Softmax Activation Function

- Binary classification: L층의 활성화 값 a가 threshold보다 낮으면 0, 높으면 1로 예측한다.

- Softmax regression: L층은 실수가 아니라 벡터를 받는데, 이는 클래스가 3개 이상이기 때문이다. 하나의 실수 값으로 0 / 1을 결정하는 것이 아니라 각 클래스별 확률을 계산하고, 그 중에서 가장 큰 확률을 가진 클래스를 예측 클래스로 사용한다.

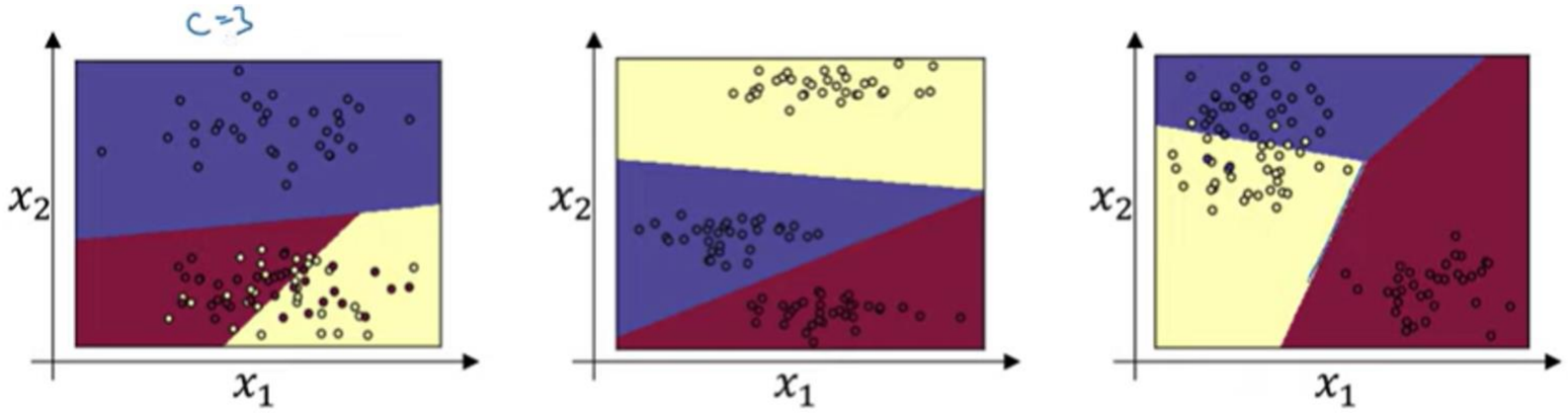


→ 정규화를 하는 것은 모든 확률의 합을 1로 만들기 위한 것

→ '정규화하기 위해 벡터를 입출력으로 받는다' 는 표현은 각각의 클래스의 확률을 나타내기 위함이라는 말과 같다.

#01 Softmax Examples

- 은닉층이 없고, C = 3일 때 입력 특성 x1, x2에 따라 데이터의 클래스를 softmax regression으로 분류한 예시



- 2번째가 데이터를 제일 잘 분류했고 1, 3번째는 분류에 약간의 오차가 보인다.

- 얻을 수 있는 직관: 두 클래스 사이의 경계는 선형이다.

→ 만약 여러 개의 은닉층을 추가한다면 더 복잡한 비선형의 경계도 형성할 수 있다.

Softmax 분류기 훈련시키기

EWHA EURON

#02 Softmax Regression의 특징

- hardmax: softmax의 반대 개념

  - hardmax: z 벡터의 원소 중 가장 큰 값을 1, 나머지를 0으로 대응시킴 -> 아주 단호한 느낌

  - softmax: z 벡터의 값을 각각의 클래스일 확률로 대응 -> 부드러운 느낌

- softmax regression generalizes logistic regression to C classes

  - Softmax regression에서 C = 2이면 logistic regression과 같다. C = 2인 softmax regression은 2개의 클래스에 대한 확률을 내놓는데, 어차피 확률의 합이 1이므로 하나만 계산해도 된다 -> 결국 y = 1의 확률을 계산하는 logistic regression과 똑같은 흐름이다.

#02 Loss Function

- 일반적으로 loss function의 목표는 예측 확률에 따른 예측 클래스가 뭐든 간에 정답 클래스에 대응하는 확률을 최대화하는 것이다.

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \qquad a^{[L]} = \hat{y} = \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix}$$

$$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{4} y_i \log \hat{y}_i$$

→ $y_i = 0$ 인 항은 고려X

$$\Rightarrow -y_2 \log \hat{y}_2 = \boxed{-\log \hat{y}_2}$$

minimize
= maximize $\log \hat{y}_2$
= maxmize $\hat{y}_2$

#02 Cost Function

- Cost function J는 전체 데이터의 loss의 합을 전체 데이터의 수 m으로 나눠줌으로써 구할 수 있다.

$$J(w^{[1]}, b^{[1]}, \cdots w^{[L]}, b^{[L]}) = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

$$\boxed{Y} = [\, y^{(1)} \;\; y^{(2)} \;\; \cdots \;\; y^{(m)} \,]$$

vectorized

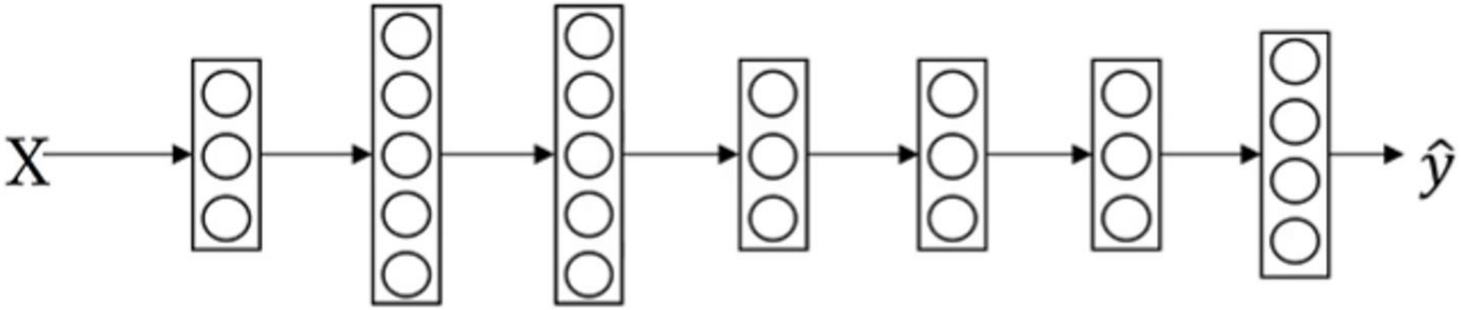
$$= \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \cdots \Bigg] \in R^{4 \times m}$$

$$\boxed{\hat{Y}} = [\, \hat{y}^{(1)} \;\; \hat{y}^{(2)} \cdots \;\; \hat{y}^{(m)} \,]$$

vectorized

$$= \begin{bmatrix} 0.3 \\ 0.2 \\ 0.1 \\ 0.4 \end{bmatrix} \cdots \Bigg] \in R^{4 \times m}$$

#02 Gradient Descent with Softmax



마지막층 L

- forward propagation

$z^{[L]} \rightarrow a^{[L]} = g(z^{[L]}) = \hat{y} \longleftrightarrow y : d(\hat{y}, y)$

(4,1)

- Backward propagation

$dz^{[L]} = \hat{y} - y$

(4,1)

$$\frac{\partial L}{\partial o_i} = -\sum_k y_k \frac{\partial \log p_k}{\partial o_i} = -\sum_k y_k \frac{1}{p_k} \frac{\partial p_k}{\partial o_i}$$

$$= -y_i(1 - p_i) - \sum_{k \neq i} y_k \frac{1}{p_k}(-p_k p_i)$$

$$= -y_i(1 - p_i) + \sum_{k \neq i} y_k (p_i)$$

$$= -y_i + y_i p_i + \sum_{k \neq i} y_k (p_i)$$

$$= p_i \left(\sum_k y_k \right) - y_i = p_i - y_i$$

#02 Gradient Descent with Softmax

- 딥러닝 프레임워크: forward propagation 하는 법을 정하면 프레임워크가 알아서 미분 계산을 통해 backpropagation을 해준다.

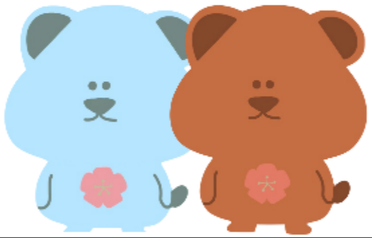
- 예시) pytorch의 forward propagation & backpropagation

```python
class MLP(nn.Module):

    def __init__(self):
        super(MLP, self).__init__()

        self.layers = nn.Sequential(
            nn.Flatten(),
            nn.Linear(28 * 28 * 1, 64),
            nn.Dropout(p = 0.5),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.Dropout(p = 0.5),
            nn.ReLU(),
            nn.Linear(32, 10)
        )

    def forward(self, x):
        x = self.layers(x)
        return x
```

```python
# define criterion & optimzier
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(mlp.parameters(), lr = 1e-4)
```

```python
optimizer.zero_grad()
outputs = mlp(inputs)
loss = criterion(outputs, targets)
loss.backward()
optimizer.step()
```

Deep Learning Frameworks



EWHA EURON

#03 Deep Learning Frameworks

- Caffe/Caffe2
- CNTK
- DL4J
- Keras
- Lasagne
- mxnet
- PaddlePaddle
- TensorFlow
- Theano
- Torch

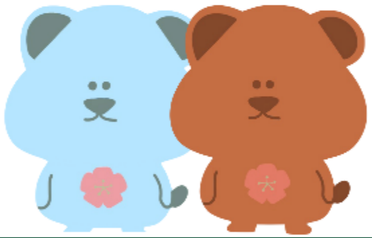
How to choose deep learning frameworks?

1. Ease of Programming: development & deployment

2. Running speed

3. Truly open: open source with good governance -> 오픈 소스이며 잘 관리되고 있는지 & 오픈 소스였던 것을 폐쇄할 가능성이 있는지
   -> 신뢰도의 문제

Tensorflow



EWHA EURON

#04 Tensorflow

```
import numpy as np
import tensorflow as tf
```

```
# weights(variable)
w = tf.Variable(0.0, dtype = tf.float32)

def cost__():
    return w**2 + 10*w + 25

# define optimizer
optimizer = tf.optimizers.SGD(learning_rate=0.01)

for step in range(1000):
    optimizer.minimize(cost__, var_list = [w])
    if step == 999:
        print(w.numpy())
```

```
-4.9999886
```

-버전 2으로 코드를 바꾼 것

-torch가 아닌 tensorflow를 import

-최소화해야 할 cost function 정의

- optimizer 정의에서, pytorch에서는 torch.optim.SGD로 쓰지만 tensorflow에서는 tf.optimizers.SGD로 쓰는 것을 확인할 수 있다.

-강의에서처럼 1000번째로 w를 업데이트했을 때의 값이 -4.9999886임을 확인할 수 있다.

#04 Tensorflow

```
coefficient = np.array([[1.], [10.], [25.]])

# weights(variable)
w = tf.Variable(0.0, dtype = tf.float32)
# tf.placeholder는 사라짐
x = tf.Variable(coefficient, dtype = tf.float32)

# x becomes data which controls coefficients of cost function
def cost__():
    return x[0][0]*w**2 + x[1][0]*w + x[2][0]

# define optimizer
optimizer = tf.optimizers.SGD(learning_rate=0.01)
# optimizer.minimize(cost__, var_list = [w])

for step in range(1000):
    optimizer.minimize(cost__, var_list = [w])
    if step == 999:
        print(w.numpy())
```

```
-4.9999886
```

- 강의에서 교수님께서 사용하신 tf.placeholder는 버전이 2.0으로 바뀌면서 사라졌다.
- X에 어떤 값(coefficient)가 주어지냐에 따라 최소화하는 cost function이 달라지고, 이에 따라 최적의 w도 바뀔 수 있다.
- 이 예시에서는 앞선 예제와 같이 x = coefficient = [1, 10, 25]로 주었기 때문에 같은 결과(같은 최적의 weigths)가 나온다.
- 왜 굳이 이렇게? -> 값을 계속 바꿀 거라면 함수 내에서보다 함수 밖에서 하는 것이 가독성도 있고 코드 수정에도 편할 것이라고 생각

#04 Tensorflow

딥러닝 프레임워크의 정방향 전파 & 역방향 전파

- 앞서 언급했듯, tensorflow에서도 마찬가지로 비용 함수를 명시해주면 자동으로 미분을 계산하고, 이를 이용하여 역방향 전파가 이루어진다.

#04 Pytorch vs Tensorflow

Tensorflow

- 초기에 딥러닝 프레임워크 시장에서 매우 큰 점유율을 가짐
- 업계 중심 프레임워크: 상업적 환경/ 대규모 배포

Pytorch

- 코드 자체가 파이썬과 유사하기 때문에 배우기 쉬움
- 연구 중심 프레임워크

퀴즈 리뷰

EWHA EURON

05 퀴즈 리뷰

2. 딥러닝 프레임워크에 관한 설명으로 옳은 것을 골라주세요 * 1점

- (●) Keras, TensorFlow와 같은 딥러닝 프레임워크는 일반적으로 C/C++ 등의 언어보다 적은 코드로 딥러닝 알고리즘을 작성할 수 있게 해준다.
- ( ) 딥러닝 프로그래밍 프레임워크는 실행을 위해 클라우드 기반의 컴퓨터가 필요하다.
- ( ) 데이터 수집을 간편하게 할 수 있도록 도와주는 고수준 API이다

1. 딥러닝 프레임워크를 이용하면 앞서 계속 언급했듯 자동 미분 등의 수학적 연산을 해주기 때문에 C / C++ 등의 언어보다 간결한 코드로 알고리즘을 작성할 수 있다. -> 옳은 선지
2. 딥러닝 프레임워크를 실행하기 위해 꼭 클라우드 기반의 컴퓨터가 필요한 것은 아니다. 개인 컴퓨터, 슈퍼 컴퓨터 등 꼭 클라우드 기반의 컴퓨터가 필요한 것은 아니다. -> 틀린 선지
3. MNIST, CIFAR10 등의 데이터를 간단히 load해서 사용할 수 있는 것은 맞지만, 딥러닝 프레임워크 자체가 데이터를 수집을 하는 것은 아니다. -> 틀린 선지

#05 퀴즈 리뷰

아래 코드는 강의에서 활용된 코드로써, TensorFlow1로 작성되었습니다.

```
# a)
coefficients = np.array([[1.], [-20], [100]])

# b)
w = tf.Variable(0, dtype=tf.float32)
# c)
x = tf.placeholder(tf.float32, [3, 1])

# d)
cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]
# e)
train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

# f)
init = tf.global_variables_initializer()
# g)
session = tf.Session()

# h)
session.run(train, feed_dict={x: coefficients})
print(session.run(w))
```

TensorFlow 2.0은 TensorFlow 2.0으로 코드를 업데이트하면서 eager execution이 기본적으로 활성화되었습니다. 따라서 세션을 열거나 변수 초기화를 수동으로 처리할 필요가 없습니다. 또한, TensorFlow 2.0은 tf.placeholder() 대신 tf.Variable()을 사용하고, 그래프 모드로 cost를 활용해야 합니다. 그리고, tf.train.GradientDescentOptimizer() 대신 tf.optimizers.SGD()를 사용합니다.

(a): x에 값을 전달할 coefficients 배열을 정의한 부분

(b), (f): 변수 w 생성-> 버전 1.0에서는 변수 w를 생성하고 값을 지정해도 tf.global_variables_initializer()가 없으면 변수에 그 값이 할당되지 않았음. 그러나 버전 2.0에서는 변수가 생성되면서 즉시 초기화되기 때문에 변수 초기화를 수동으로 해줄 필요가 없음.

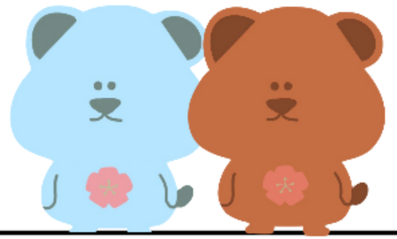
(c): 버전이 2.0으로 바뀌면서 tf.placeholder는 사라지고 대신 x = tf.Variable(coefficient, dtype = tf.float32)라는 코드가 추가되어야 함

(d): 그래프 모드로 cost를 활용해야 함. -> 그래프 모드를 사용하면 유연성과 속도가 크게 향상됨.

(e): tf.train.GradientDescentOptimizer대신 tf.optimizers.SGD()를 사용함.

(g), (h): session을 열거나 돌릴 필요 없음.

THANK YOU



EWHA EURON