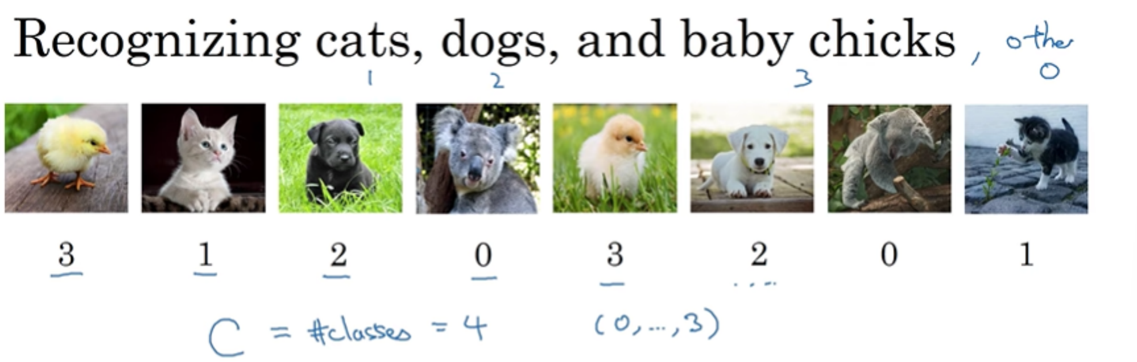
12주차

태그	완료

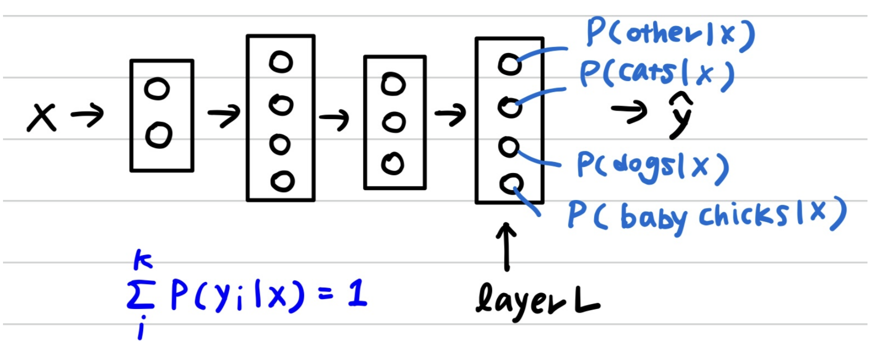
Softmax Regression

Softmax Regression

- softmax는 여러 개의 클래스를 분류할 시 사용된다.
- 예를 들어 cats, dogs, baby chicks의 3개의 클래스가 있고, 각각을 1, 2, 3 클래스라고 하자. (셋 중 아무것도 해당되지 않은 사진 other은 0 클래스로 분류한다)



- 아래와 같은 NN이 있다면, 마지막 층의 각 노드는 input X가 i번째 클래스에 속할 확률(probability) 이다.



- NN의 마지막 층은 소프트맥스 층이다.

Softmax Activation Function

- 소프트맥스의 층의 활성화 함수는 다음과 같이 정의된다.

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

Activation function

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

→ $a^{[L]}$ 의 크기를 1로 정규화시키기 위함

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

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

$$\rightarrow e^{z^{[L]}} = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.4 \\ 0.4 \\ 20.1 \end{bmatrix}$$

for 정규화

$$\& \sum_{i=1}^{4} e^{z_i^{[L]}} = 176.3$$

$$\rightarrow a^{[L]} = \frac{e^{z^{[L]}}}{176.3} = 1$$

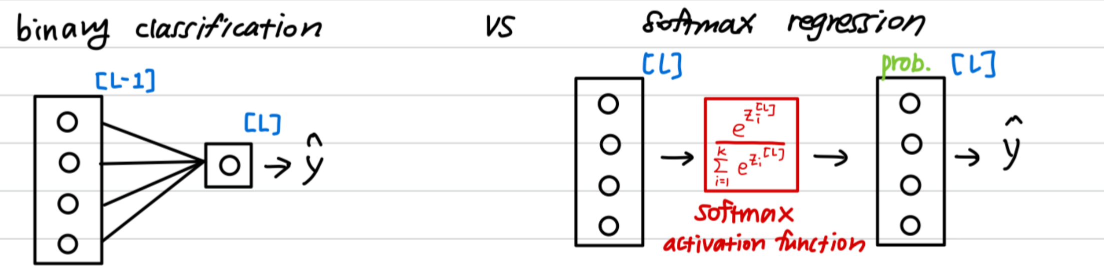
$$P(\text{other}|x) = \frac{e^5}{176.3} = 0.842 \rightarrow$$ input X가 클래스 0에 속할 확률 : 84.2%

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

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

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

- 특징은 실수를 입력으로 받아 실수를 출력하는 다른 활성화 함수(sigmoid, relu)와는 다르게 벡터 (이 문제에서는 (4, 1) 벡터)를 입력 받아 벡터를 출력한다는 점이다. 이는 정규화를 하기 위함이다.
    - binary classification은 L층에서 실수를 받고, 그 실수가 threshold보다 작다면 0, 크다면 1로 분류되는 메커니즘이다.
    - softmax regression은 L층에서 실수가 아니라 벡터를 받는데, 이는 클래스가 3개 이상이기 때문이다. 하나의 실수로 0 / 1을 결정하는 것이 아니라 각 클래스별 확률을 계산하고, 그 중에서 가장 큰 확률을 가진 클래스를 예측 클래스로 사용한다. 이때 정규화를 하는 것은 모든 확률의 합을 1로 만들기 위함이다. 즉, '정규화하기 위해 벡터를 입출력으로 받는다'는 표현은 각각의 클래스의 확률을 나타내기 위함이라는 말과 같다.

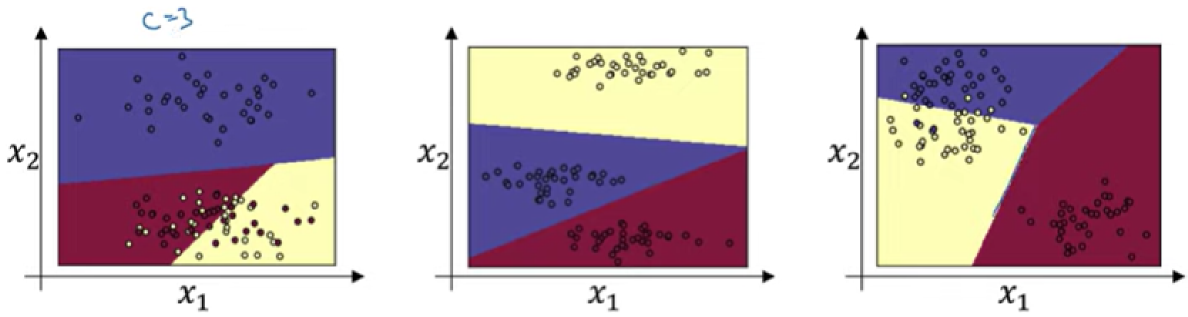


- input : $z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]}$
  └ 실수
- activation function : $a^{[L]} = g(z^{[L]})$
- output $a^{[L]}$
  └ 실수
- $\hat{y} = \begin{cases} 0 & (\text{threshold} > a^{[L]}) \\ 1 & (\text{threshold} < a^{[L]}) \end{cases}$

- input : $z^{[L]} = W^{[L]} a^{[L-1]} + b^{[L]} \in (4,1)$
- activation function : 정규화한 확률
- output : 각 클래스에 속할 확률
- $\hat{y} = \underset{i}{\text{argmax}}(\text{output})$

Softmax Examples

- 은닉층이 없고, C = 3일 때 입력 특성 x1, x2에 따라 data의 클래스를 softmax regression으로 분류한 예시
- 2번째 그래프가 제일 잘 분류했고 1, 3번째는 약간의 오차가 보인다.



- 얻을 수 있는 직관: 두 클래스 사이의 경계가 선형이다.
- 만약 여러 개의 은닉층을 추가한다면 더 복잡한 비선형의 경계도 볼 수 있다.

Softmax 분류기 훈련시키기

Softmax Regression의 특징

- hardmax: softmax의 반대 개념
    - softmax: z 벡터 → 활성화 함수 g → a 벡터 → z 벡터의 값을 부드러운 느낌으로 각각의 클래스일 확률로 대응
    - hardmax: z 벡터 중 가장 큰 값이 있는 곳에 1, 나머지는 0을 할당 → 아주 단호한 느낌
- softmax regression generalizes logistic regression to C classes
    - C = 2이면 logistic regression과 같다. C = 2인 softmax regression은 두 개의 확률을 내놓는데, 어차피 확률의 합이 1이므로 하나만 계산해도 된다. 이는 결국 y = 1일 확률을 계산하는 logistic regression과 똑같은 흐름이.

$$a^{[L]} = \begin{bmatrix} 0.842 \\ \cancel{0.158} \end{bmatrix}$$

Loss function

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

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

→ $y_i = 0$ 인 항은 고려X

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

minimize

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

= maxmize $\hat{y}_2$

- 일반적으로 loss function은 예측 확률에 따른 예측 클래스가 뭐든 간에 정답 클래스에 대응하는 확률을 가능한 한 크게 만드는 것이 목표다.

Cost function

- cost function J는 전체 데이터의 loss를 구한 뒤 전체 데이터의 개수 m으로 나눠줘서 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

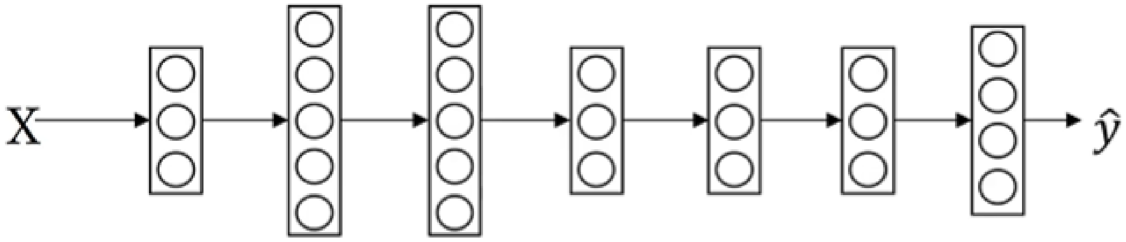
$$= \begin{bmatrix} 0 & 0 & \cdots \\ 1 & 0 & \\ 0 & 1 & \\ 0 & 0 & \end{bmatrix} \in \mathbb{R}^{4 \times m}$$

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

vectorized

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

Gradient Descent with softmax



마지막층 L

- forward propagation

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

(4,1)

- Backward propagation

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

(4,1)

$$\frac{\partial L}{\partial o_i} = -\sum_k y_k \frac{\partial \log p_k}{\partial o_i} = -\sum_k y_k \frac{1}{p_k}\frac{\partial p_k}{\partial o_i}$$
$$= -y_i(1-p_i) - \sum_{k \neq i} y_k \frac{1}{p_k}(-p_k p_i)$$
$$= -y_i(1-p_i) + \sum_{k \neq i} y_k (p_i)$$
$$= -y_i + y_i p_i + \sum_{k \neq i} y_k (p_i)$$
$$= p_i \left(\sum_k y_k\right) - y_i = p_i - y_i$$

- 딥러닝 프레임워크: forward propagation하는 법을 정하면 프레임워크가 스스로 미분 계산을 통해 backpropagation을 해준다.

Deep Learning Frameworks

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

choosing deep learning frameworks

1. Ease of Programming: development and deployment(전개, 상용화)
2. running speed
3. truly open: open source with good governance → 오픈 소스이며 잘 관리되고 있는지, 혹은 오픈 소스였던 것을 폐쇄할 가능성이 있는지. 즉, 신뢰도의 문제이다.

Tensorflow

** tensorflow 2.x를 기준으로 코드를 다시 작성함**

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

- optimizer.apply_gradients(zip(gradients, [w])): gradients와 [w]를 반복 가능한 객체(iterator)로 만들어 gradients와 [w]의 요소를 각각 하나씩 가져와 튜플 형태로 반환하는 것을 반복한다.

```
# 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
```

- 강의에서 사용하신 tf.placeholder는 버전이 2.x로 바뀌면서 사라졌다. x에 어떤 coefficient가 주어지냐에 따라 최소화해야 하는 cost function이 달라지기 때문에 이에 따른 최적의 w가 달라질 수 있다. 이 예시에서는 위와 같이 1, 10, 25를 계수로 썼기 때문에 같은 결과가 나온다.

```
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
```

- tensorflow에서 비용 함수를 명시해주면, 자동으로 미분을 계산하고 비용 함수를 최소화해준다. 즉, 정방향 전파를 정의하는 것은 역방향 전파 함수도 구현한 것과 같다.