ML/DL for Everyone Season2

with TensorFlow

Lab06-2
Softmax Classifier (fancy version)
: Animal classification

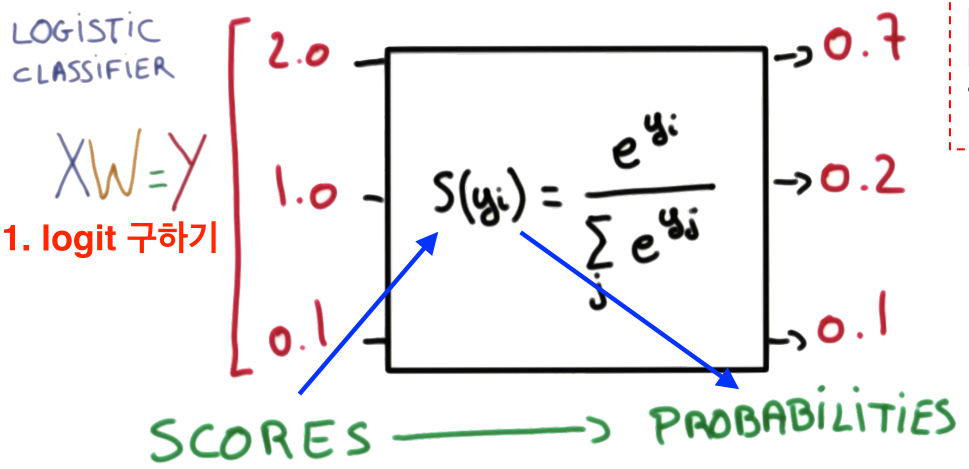
Code: https://github.com/deeplearningzerotoall/TensorFlow
Slides: https://drive.google.com/drive/folders/1twBsdLkI2P15J0DgYs77_E_EVKt7Ghav
Lecturer: sungjin7127@gmail.com

Lab6-2:
Softmax Classifier (Animal Classification)

- Softmax function
- Softmax_cross_entropy_with_logits
- Sample Dataset
- tf.one_hot_and_reshape
- Implementation
- What’s Next

Softmax function

softmax 의 총합값은 1



1. logit 구하기

hypothesis = tf.nn.**softmax**(tf.matmul(X,W)+b)

tf.matmul(X,W)+b

2. logit 에서 나온 값을 softmax 에 적용하여 확률 값으로 치환

https://www.udacity.com/course/viewer#!/c-ud730/l-6370362152/m-6379811817

Softmax_cross_entropy_with_logits

```
logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)
```

1 기존에 했던 방법

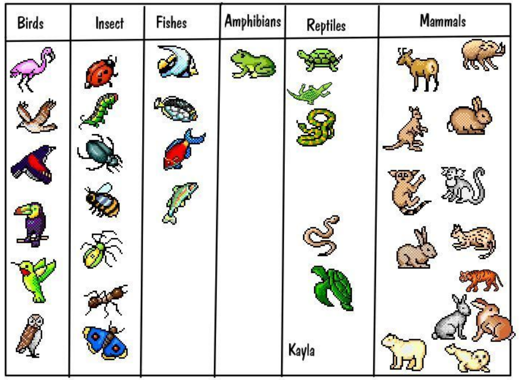
```
# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

2

```
# Cross entropy cost/loss
cost_i = tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits,
                                                    labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)
```

Sample Dataset

Animal classification
with *softmax_cross_entropy_with_logits*



0~6까지 7종으로 분류

1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	0
0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	3
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	1
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	6
0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	1
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0

```python
# Predicting animal type based on various features
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
```

https://kr.pinterest.com/explore/animal-classification-activity/

이 데이터의 shape 은 (?, 1) 인데, 여기는 데이터가 2개 이므로 (2, 1) ← [[0], [3]]

one_hot 을 적용하면 이런 형태가 된다

tf.one_hot and reshape

one_hot 적용하면 한 차원이 추가돼서

[
[[1, 0, 0, 0, 0, 0, 0]],
[[0, 0, 0, 1, 0, 0, 0]]
]

총 7개 클래스
3차원이 된다

.reshape 해서

[
[1, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 1, 0, 0, 0]
]

1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	1	1	1	1	0	0	4	0	0	1	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0
1	0	0	1	0	0	0	1	1	1	0	0	4	1	1	1	0
0	0	1	0	0	1	0	1	1	0	0	1	0	1	1	0	3
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
1	0	0	1	0	0	0	1	1	1	0	0	4	0	1	0	0
1	0	0	1	0	0	1	1	1	1	0	0	4	1	0	1	0
0	1	1	0	1	0	0	0	1	1	0	0	2	1	1	0	1
0	0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	3
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	4	0	0	0	6
0	0	1	0	0	1	1	0	0	0	0	0	6	0	0	0	6
0	1	1	0	1	0	1	0	1	1	0	0	2	1	0	0	1
1	0	0	1	0	0	0	1	1	1	0	0	4	1	0	1	0

이렇게만 하면 오류 발생 / 우리가 원하는 모양이 아니다

전부를 의미

```
nb_classes = 7  # 0 ~ 6
Y_one_hot = tf.one_hot(list(y_data), nb_classes) # one hot shape=(?, 1, 7)
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes]) # shape=(?, 7)
```

tf.reshape(이런 모양을, 이런 모양으로 바꿔주세요)

최종적인 shape

If the input indices is rank N, the output will have rank N+1. The new axis is created at dimension axis (default: the new axis is appended at the end).
https://www.tensorflow.org/api_docs/python/tf/one_hot

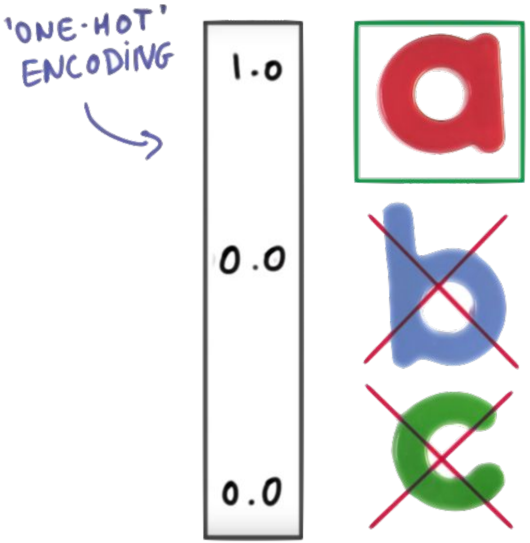
Implementation - Load Dataset

```python
# Predicting animal type based on various features
xy = np.loadtxt('data-04-zoo.csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]

print(x_data.shape, y_data.shape)

nb_classes = 7  # 0 ~ 6

# Make Y data as onehot shape
Y_one_hot = tf.one_hot(list(y_data), nb_classes)
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])
```



Implementation - Dataset

```
dataset =
tf.data.Dataset.from_tensor_slices((x_data,Y_one_hot)).shuffle(len(x_data)).batch(16).repeat(100)
```

```
<RepeatDataset shapes: ((?, 16), (?, 7)), types: (tf.float32, tf.float32)>
```

Implementation - Softmax Classifier

```
#Weight and bias setting
W = tfe.Variable(tf.random_normal([16, nb_classes]), name='weight')
b = tfe.Variable(tf.random_normal([nb_classes]), name='bias')
variables = [W, b]
```

weight와 bias를 업데이트 할 변수를 따로 저장

```
# tf.nn.softmax computes softmax activations
def logit_fn(X):
    return tf.matmul(X, W) + b

def hypothesis(X):
    return tf.nn.softmax(logit_fn(X))
```

이후에 우리가 정확도를 맞추기 위한 일환으로 뒤에서 설명

```
def cost_fn(X, Y):
    logits = logit_fn(X)
    cost_i = tf.nn.softmax_cross_entropy_with_logits_v2(logits=logits,
                                                          labels=Y)
    cost = tf.reduce_mean(cost_i)
    return cost
```

Implementation - Softmax Classifier

```python
def grad_fn(X, Y):
    with tf.GradientTape() as tape:
        loss = cost_fn(X, Y)
        grads = tape.gradient(loss, variables)
        return grads
```

W, b

최솟값을 구하기 위해 접근하는 방법

logit보다 hypothesis를 통해 가장 높은 값을 간편하게 구할 수 있어서 위에서 함수 만든 것

```python
def prediction(X, Y):
    pred = tf.argmax(hypothesis(X), 1)
    correct_prediction = tf.equal(pred, tf.argmax(Y, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    return accuracy
```

Implementation - Training

```
def fit(X, Y, epochs=100, verbose=50):
    optimizer =  tf.train.GradientDescentOptimizer(learning_rate=0.1)

    for i in range(epochs):
        grads = grad_fn(X, Y)
        optimizer.apply_gradients(zip(grads, variables))
        if (i==0) | ((i+1)%verbose==0):
            acc = prediction(X, Y).numpy()
            loss = tf.reduce_sum(cost_fn(X, Y)).numpy()

            print('Loss & Acc at {} epoch {}, {}'.format(i+1, loss, acc))

 fit(x_data, Y_one_hot)
```

Implementation - Result

Steps: 1 Loss: 7.090885639190674, Acc: 0.0891089141368866
Steps: 100 Loss: 0.7543396353721619, Acc: 0.8118811845779419
Steps: 200 Loss: 0.42519062757492065, Acc: 0.8910890817642212
Steps: 300 Loss: 0.3010515570640564, Acc: 0.9108911156654358
Steps: 400 Loss: 0.23578841984272003, Acc: 0.9405940771102905
Steps: 500 Loss: 0.19521062076091766, Acc: 0.9603960514068604
Steps: 600 Loss: 0.16714605689048767, Acc: 0.9603960514068604
Steps: 700 Loss: 0.1463650017976761, Acc: 0.9702970385551453
Steps: 800 Loss: 0.13026459515094757, Acc: 0.9900990128517151
Steps: 900 Loss: 0.11738719791173935, Acc: 0.9900990128517151
Steps: 1000 Loss: 0.10684021562337875, Acc: 1.0

What’s Next?

- learning_rate_and_evaluation - Eager execution