2주차 발표 자료

이지환

Multinomial Classification (다항 분류)

@Multinomial classfication (다항 분류)

- 해당 데이터가 True/False인지 분류하는 이진 분류와는 달리, 해당 데이터가 A,B,C 등 여러 class 중 어디에 속하는지 분류하는 것이 다항 분류이다.

$$\begin{pmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} w_{A1}x_1 & w_{A2}x_2 & w_{A3}x_3 \\ w_{B1}x_1 & w_{B2}x_2 & w_{B3}x_3 \\ w_{C1}x_1 & w_{C2}x_2 & w_{C3}x_3 \end{pmatrix} = \begin{pmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{pmatrix}$$

와 같은 꼴로 다항 분류 식을 작성할 수 있으며, 이때 y값(score)을 소프트맥스 함수를 사용하여 변환한다 (이진 분류에선 시그모이드 함수를 사용).
소프트맥스 함수는 다음과 같으며, 모든 y값을 확률로 반환하며 이때 나오는 확률의 합이 1이 되도록 만들어준다.

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

이 때, $i = 1, 2$이고 $y_2 = 0$이면 위 소프트맥스 함수는

$$S(y_i) = \frac{1}{1 + e^{y_1}} \quad (i = 1, 2)$$

와 같은 형태로 변환할 수 있으며, 이는 이진 분류에서 사용한 시그모이드 함수와 동일하다.
즉, 소프트맥스 함수는 시그모이드 함수를 포함하는 개념으로 볼 수 있다.
소프트맥스 함수로 변환된 값들은 다음과 같이 One hot encoding(OHE)을 통해 0과 1의 값으로 변환하며, 그 결과로 어디 class에 속하는지를 예측한다.

$$OHE\left(\begin{pmatrix} S(y_1) \\ S(y_2) \\ S(y_3) \end{pmatrix}\right) = OHE\left(\begin{pmatrix} 0.7 \\ 0.2 \\ 0.1 \end{pmatrix}\right) = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

Multinomial Classification (다항 분류)

@다항 분류의 Cross-entropy Cost function

다항 분류의 Cost function은 Cross-entropy Cost function이라고도 하며, 식은 다음과 같다.

$$D(S, L) = -\sum_i L_i \log(S_i)$$

$$L_i = OHE(S(y_i))$$

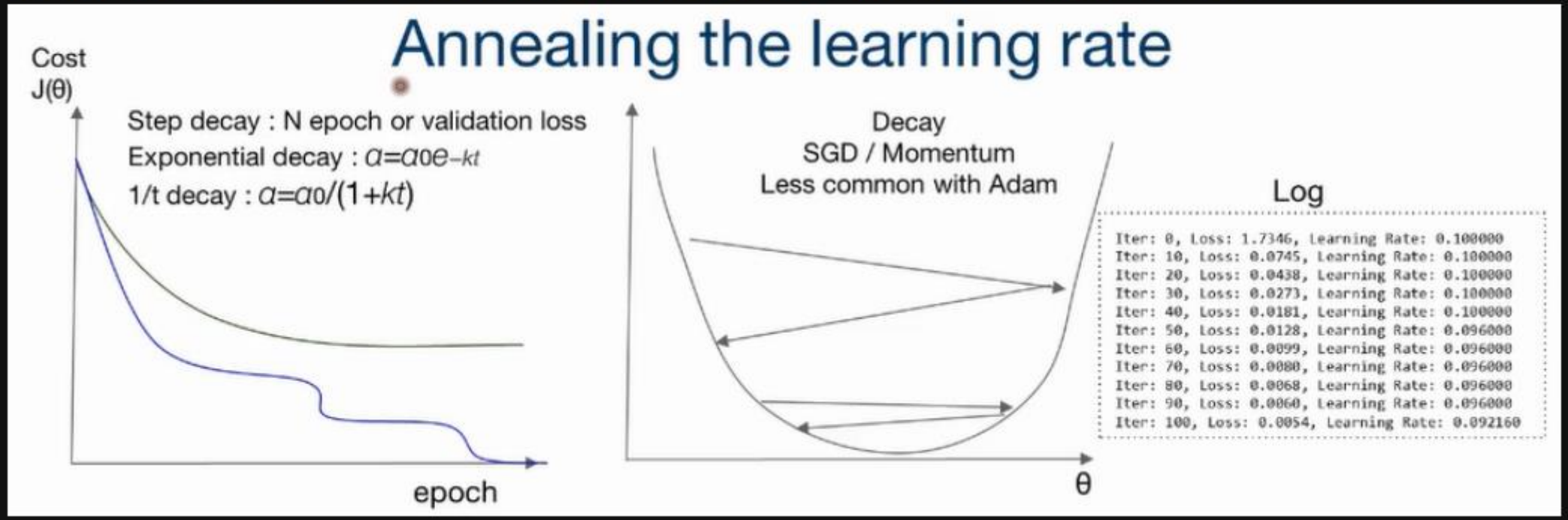
특히, training set이 여러 개일 경우 다음과 같은 식을 사용한다.

$$TotalLoss = \frac{1}{N} \sum_i D(S(WX_i + b), L) \qquad i = 1, \ldots, N$$

Learning Rate Decay

@Learning Rate Decay

- 기존의 Learning Rate가 높은 경우 loss 값을 빠르게 내릴 수는 있지만, 최적의 Cost를 찾지 못할 수 있고
- 낮은 경우 최적의 Cost를 찾을 확률이 높지만 최적화 시간이 오래 걸린다.
- 또한, 고정된 learning rate에선 Gradient descent를 적용하여 Cost를 줄이는 정도에는 한계가 있다.
- 따라서, 최초 learning rate를 높게 설정한 뒤, 특정 시점(epoch)마다 learning rate를 조정함으로써 최초 learning rate로 얻은 cost보다 더 낮은 cost를 얻을 수 있도록 한다.



Data Preprocessing: Feature Scaling & Noisy Data

@Data Processing - Feature Scaling

- Feature scaling: feature들의 크기와 범위를 줄임으로써 학습 시 각 feature에 불필요한 가중치가 적용되지 않도록 만드는 작업이다.
- Feature scaling에는 크게 Normalization, Standardization 두 가지가 있다.
- Normalization(정규화): feature의 범위를 조정한다.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- Standardization(표준화): z-score normalization이라고도 부르며, feature의 평균을 0, 표준편차를 1로 변환한다.

$$x_{new} = \frac{x - \mu}{\sigma}$$

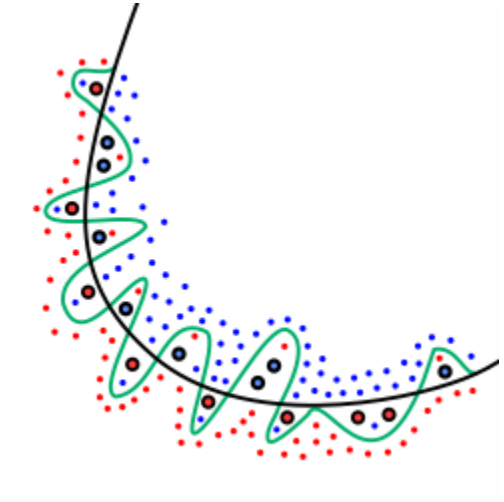
@Noisy Data & Data Preprocessing

- Noisy Data: 유의미한 분석 결과를 이끌어 내는 데 도움이 되지 않는 데이터를 말한다.
- Noisy Data에는 이상치와 결측치가 있으며, NLP 관점에서는 불필요한 단어, 이미지 처리 관점에서는 배경과 같은 불필요한 부분도 포함한다.
- 이러한 Noisy Data가 학습에 사용되지 않도록 미리 제거하거나 수정하는 과정을 데이터 전처리 (Data Preprocessing)이라고 부른다.

Overfitting & Underfitting

@Overfitting & Underfitting

- Overfitting이란 말그대로 AI가 훈련 데이터에 과도하게 학습된 결과를 말하며, Model Capacity가 높을수록 발생할 확률이 증가한다. (Model Capacity: 모델의 복잡성 정도)
  - 문제점: train에 대한 validation의 accuracy에 비해 test에선 accuracy가 낮아진다.
  - 해결방법: Feature Normalization, Regularization, Dropout, Batch Normalization, 더 많은 데이터 확보, Model Capacity 축소, 교차 검증(Cross Validation)
- Underfitting이란 Overfitting과 반대로 가지고 있는 train data에 대한 학습이 부족하여 accuracy가 낮은 상황을 말한다.



Source: https://ko.wikipedia.org/wiki/%EA%B3%BC%EC%A0%81%ED%95%A9

L1 Loss & L2 Loss

@L1 loss, L2 loss

- **L1 loss** (MAE): Least Absolute Deviations (LAD)라고도 부르며, 다음과 같이 오차에 절댓값을 취하여 Loss를 구한다.

$$L = \sum_{i=1}^{n} |y_i - f(x_i)|$$

- **L2 loss** (MSE): Least Square Error (LSE)라고도 부르며, 오차를 제곱한 값들을 모두 합하여 loss를 구한다.

$$L = \sum_{i=1}^{n} (y_i - f(x_i))^2$$

- 둘은 다음과 같은 차이점을 가진다.
  1. Robustness (강건함)
     - outlier로 인해 loss가 **변하지 않는**(강건한) 정도를 의미한다.
     - L1가 L2에 비해 outlier로 인한 변화 정도가 적으며 Robustness가 크다.
  2. Stability
     - 모델이 비슷한 데이터에 대해 일관적으로 예측하는 정도를 의미한다.
     - 추가로 들어온 데이터가 Outlier가 아닌 현재 데이터와 비슷한 경향성을 가진 데이터라고 볼 수 있을 때, L1가 L2에 비해 변화가 크다.

Regularization

@Regularization

- Overfitting되지 않도록 모델을 구성하는 기울기 또는 계수(weight)에 정규화 요소(Regularization Term)을 더해주는 작업이다. 아래 식에서 λ는 정규화 요소의 계수이며 0에 가까울 수록 정규화 효과는 감소한다.
- L1 Regularization

$$cost(W,b) = \frac{1}{m}\sum_{i}^{m} L(\hat{y}_i, y_i) + \lambda|w|$$

- L2 Regularization

$$cost(W,b) = \frac{1}{m}\sum_{i}^{m} L(\hat{y}_i, y_i) + \lambda w^2$$

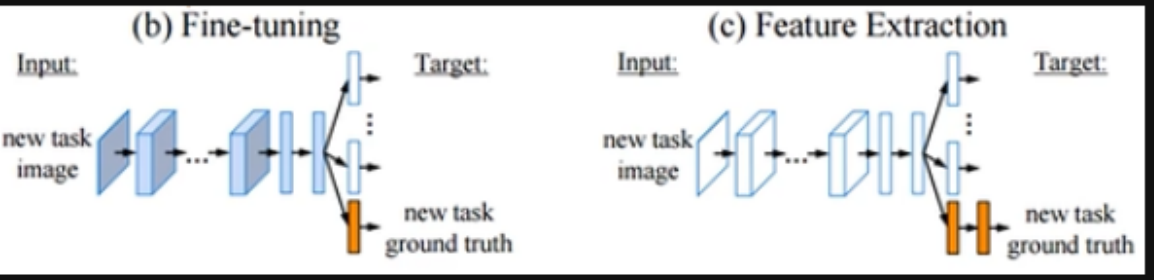
- 더 나아가, L1 Regularization을 사용하는 선형 회귀 모델을 **Lasso Model**이라고 하며, L2 Regularization을 사용하는 선형 회귀 모델을 Ridge 모델이라고 한다.

- Weight의 수가 많아지면 수만큼 더한다 tf.nn.l2_loss(W2) + tf.nn.l2_loss(W3)

Fine tuning & Feature Extraction

@Fine tuning & Feature Extraction

- Fine tunning: 미세한 조정을 통해 더 나은 결과를 도출하는 과정을 말하며, '제안서를 파인 튜닝해서 다시 제출하라'처럼 범용적으로 불린다. 특히, AI를 Fine tunning한다는 것은 추가 데이터를 학습시킴으로써 사전 학습된 파라미터 등을 업데이트 시키는 작업을 말한다.
- Feature Extraction: 데이터들이 어떤 특징을 가지고 있는지 찾아내고 그 결과를 벡터로 변환하는 작업을 말한다. Feature Extraction은 분류 또는 군집 분석 시 불필요한 정보를 제거하고 핵심적인 정보를 추출하거나, 차원 축소를 통해 계산량을 줄임으로써 분석 효율과 성능을 향상시킬 수 있다.
- 강의에선 다음과 같이 Feature Extraction을 Fine tuning과 비슷한 의미로 해석하였는데, 정확하지 않은 내용으로 보여진다.



Decorator

1주차 발표자료 中 with 구문

decorator -> 추후에 살펴볼 것

tensorflow 공식 문서 내 with가 사용된 예제

```
@tf.function
def train_step(images, labels):
  with tf.GradientTape() as tape:
    # training=True is only needed if there are layers with different
    # behavior during training versus inference (e.g. Dropout).
    predictions = model(images, training=True)
    loss = loss_object(labels, predictions)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))

  train_loss(loss)
  train_accuracy(labels, predictions)
```

First class function & Closure

Decorator를 살펴보기 전에...

First class citizen이란? -> OOP에서 함수 자체를 값으로 사용할 수 있다는 개념
First class citizen의 특징

1. 변수 혹은 데이터 자료 구조 안에 담을 수 있다
2. 매개변수로 전달할 수 있다
3. return으로 사용될 수 있다

Python에서 function은 First class citizen에 속하며, 이를 First class function라고 부른다.
따라서, Python에서 function은 위와 같은 특징을 동일하게 가진다.

```
def calculate(x):
    def plus(y):
        return x+y
    return plus
```

```
cal_ = calculate
cal_
```

```
<function __main__.calculate(x)>
```

함수를 값처럼 변수에 저장할 수 있다

First class function & Closure

Decorator를 살펴보기 전에...

이러한 Python의 First class function 개념을 이용하여 클로저라는 함수를 만들 수 있다.

클로저란?

➔ 어떤 함수의 내부 함수가 외부 함수의 변수를 참조할 때, 외부 함수가 return, 즉 종료된 후에도 외부 함수의 변수를 계속 참조 가능하도록 저장하는 함수를 일컫는다.

클로저 함수의 특징은 다음과 같다.

➔ 어떤 함수의 내부 함수이며, 그 내부 함수가 어떤 함수의 변수를 참조한다

➔ 외부 함수가 내부 함수를 return한다

```
def calculate(x):
    def plus(y):
        return x+y
    return plus

cal_ = calculate(3)
cal_

<function __main__.calculate.<locals>.plus(y)>

cal_(4)

7
```

여기서 plus함수가 클로저의 조건을 모두 만족한다.

또한, calculate라는 외부 함수를 호출하면서 3이라는 인자를 입력하면서 cal_이라는 변수에 plus를 저장하였다.
(First class function 개념 활용)

이후, calculate함수는 return되었음에도 불구하고 (메모리에서 삭제) cal_변수에 저장된 plus함수는 calculate의 x라는 변수를 참조하여 7이라는 값을 return시켰다.

First class function & Closure

Decorator를 살펴보기 전에...

이러한 Python의 First class function 개념을 이용하여 클로저라는 함수를 만들 수 있다.

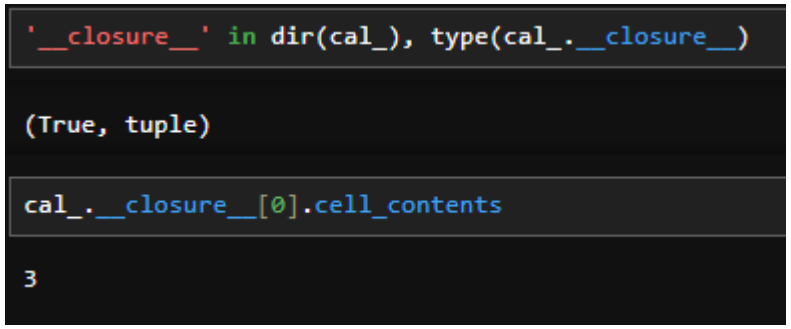
클로저란?

➔ 어떤 함수의 내부 함수가 외부 함수의 변수를 참조할 때, 외부 함수가 return, 즉 종료된 후에도 외부 함수의 변수를 계속 참조 가능하도록 저장하는 함수를 일컫는다.

클로저 함수의 특징은 다음과 같다.

➔ 어떤 함수의 내부 함수이며, 그 내부 함수가 어떤 함수의 변수를 참조한다

➔ 외부 함수가 내부 함수를 return한다



cal_변수, 즉 plus함수는 __closure__라는 tuple을 가지고 있으며 인덱싱 후 cell_contents를 확인하면 이전에 calculate를 호출하면서 인자로 입력하였던 3을 return한다.

만약 클로저 함수 조건을 만족하지 않았다면 __closure__는 None이다.

Decorator

decorator (데코레이터)

다른 함수를 돌려주는 함수인데, 보통 `@wrapper` 문법을 사용한 함수 변환으로 적용됩니다. 데코레이터의 흔한 예는 `classmethod()` 과 `staticmethod()` 입니다.

데코레이터 문법은 단지 편의 문법일 뿐입니다. 다음 두 함수 정의는 의미상으로 동등합니다:

```
def f(arg):
    ...
f = staticmethod(f)

@staticmethod
def f(arg):
    ...
```

Decorator = 함수를 꾸며주는 역할 = 함수를 인자로 전달
closure의 개념을 활용하여 외부 함수 deco의 인자인 fn함수를 내부 함수인 deco_에 전달함

```
def deco(fn):
    def deco_(n):
        print("it's decorator.")
        return fn()*n
    return deco_
```

```
@deco
def hi():
    return 'hi'

hi(3)
```

```
it's decorator.
'hihihi'
```

```
def hi():
    return 'hi'

deco(hi)(3)
```

```
it's decorator.
'hihihi'
```

```
@deco
def hello():
    return 'hello'

@deco
def no():
    return 'no'

hello(2)
```

```
it's decorator.
'hellohello'
```

```
no(4)
```

```
it's decorator.
'nononono'
```

다양한 함수를 동일한 로직으로 꾸며주며 정의할 수 있음!

Source: https://docs.python.org/ko/3/index.html

더 알아보기

Sequential & 하위 layers들의 기능과 사용법

Tensorflow Keras

- Tensorflow keras API를 통해 모델에 대한 정의

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```