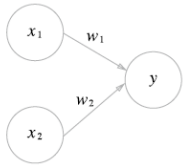
[1주차] KUBIG 22-1_NLP

퍼셉트론 (perceptron)
  퍼셉트론이 작동하는 방식 ?
  단층 퍼셉트론
    단층 퍼셉트론의 한계
  다층 퍼셉트론 (MLP)
인공 신경망 (NN)
  활성화 함수 (activation function)
    활성화 함수의 처리과정 ?
  활성화 함수의 종류
    1. 계단 함수 (step function)
    2. 시그모이드 함수 (sigmoid function)
    시그모이드 함수의 기울기 소실 문제
    3. 하이퍼볼릭탄젠트 함수 (tanh)
    4. ReLU 함수
    5. Leaky ReLU 함수
    6. 소프트맥스 함수 (softmax function)
    7. 항등 함수 (identity function)
딥러닝 학습 방법
  손실 함수 (loss function)
    1. 평균 제곱 오차, MSE (Mean Squared Error)
    2. 이항 교차 엔트로피, BCE (Binary Cross-Entropy)
    3. 범주형 교차 엔트로피, CCE (Categorical Cross-Entropy)
  배치 크기에 따른 경사 하강법
    1. 배치 경사 하강법 (Batch Gradient Descent)
    2. 확률적 경사 하강법 (Stochastic Gradient Descent)
    3. 미니 배치 경사 하강법 (Mini-batch Gradient Descent)
  옵티마이저 (optimizer)
    1. 모멘텀 (Momentum)
    2. 아다그라드 (Adagrad)
    3. RMSprop
    4. 아담 (Adam)
자연어처리(natural language processing)이란?
  자연어처리 과정
  실전 프로젝트
자연어처리의 응용 분야
  1. 텍스트 분류 (text classification)
  2. 감성 분석 (sentiment analysis)
  3. 내용 요약 (text summarization)
  4. 기계 번역 (machine translation)
  5. 질의 응답 (question answering)

퍼셉트론 (perceptron)

: 다수의 신호를 입력받아 하나의 신호를 출력하는 알고리즘



- 입력 신호 : x1, x2
- 가중치 : w1, w2
- 출력 신호 : y
- 원 : 뉴런/노드를 나타냄.
- 선 : 간선을 나타냄.

퍼셉트론이 작동하는 방식 ?

1. 입력 신호가 뉴런에 보내질 때 각각의 고유한 가중치가 곱해짐. (x1w1, x2w2)
2. 뉴런에서 보내온 신호의 총합이 정해진 한계치(임계치)를 넘어설 때만 1을 출력하고, 나머지의 경우에는 0을 출력함. (이는 '뉴런이 활성화'된다와도 같은 개념임.)

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

   * 여기서 가중치는 각 신호가 결과에 주는 영향력을 조절하는 요소로, 가중치가 클수록 해당 신호가 더 중요하다는 의미 !

퍼셉트론에는 2가지 종류가 존재한다: 단층 퍼셉트론(single-layer perceptron) & 다층 퍼셉트론(multi-layer perceptron)

단층 퍼셉트론

: 값을 보내는 단계(input layer) + 값을 받아서 출력하는 단계(output layer)의 2개의 단계로 이루어진 퍼셉트론

- 단층 퍼셉트론 이용시 AND, NAND, OR 게이트 구현 가능
  (게이트? : 컴퓨터가 2개의 값(0, 1)을 입력해 하나의 값을 출력하는 회로가 모여 만들어지는데, 여기서 회로가 게이트와 동일함.)
  - **AND** 게이트 → 입력 신호가 모두 1일때는 1 출력, 하나라도 0이면 0 출력

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

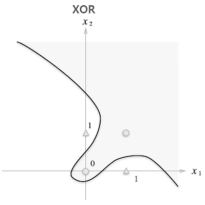
  - **NAND** 게이트 → AND 게이트를 뒤집은 형태, 입력 신호가 모두 1일때는 0 출력, 하나라도 0이면 1 출력

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

- **OR** 게이트 → 입력 신호가 모두 0일때는 0 출력, 하나라도 1이면 1 출력

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

단층 퍼셉트론의 한계



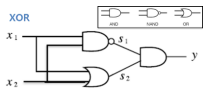
단층 퍼셉트론으로는 XOR 게이트 (모두 1이거나 모두 0일때만 0 출력, 나머지에는 1 출력)를 구현할 수 없음.
단층 퍼셉트론은 **직선으로 나뉜 2개의 영역**을 만들지만, XOR 게이트는 직선으로 2개의 영역을 나눌 수 없다.
즉, 단층 퍼셉트론은 직선 하나로 나눈 영역만 표현할 수 있으며, 곡선(비선형) 영역의 표현이 어렵다는 단점이 존재한다. → **다층 퍼셉트론의 등장 !**

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

다층 퍼셉트론 (MLP)

: 단층이 입력층과 출력층만 존재하는 반면, 다층 퍼셉트론은 중간에 층(=은닉층, hidden layer)을 더 추가한 구조 (둘의 차이는 층의 개수)

- XOR 게이트 → AND, OR, NAND 게이트를 조합해서 구현한 게이트



x_1	x_2	s_1	s_2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

- 다층 퍼셉트론은 심층 신경망 (Deep Neural Network, DNN)의 한 종류 (DNN: 은닉층이 2개 이상인 신경망)

다층 퍼셉트론만으로도 복잡한 함수를 표현할 수는 있지만, 원하는 결과를 출력하도록 가중치를 설정하는 작업이 상당히 수동적인 작업임. → 인공신경망의 등장 !

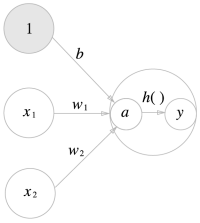
인공 신경망 (NN)

: 가중치 매개변수의 적절한 값을 데이터로부터 자동하는 학습하는 능력을 가진 신경망

활성화 함수 (activation function)

: 입력 신호의 총 합을 출력 신호로 변환하는 함수

활성화 함수의 처리과정 ?

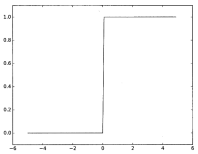


1. 가중치가 있는 입력 신호와 편향의 총합 (x1w1 + x2w2 + b) 계산 (=a)
2. 활성화 함수 h()에 넣어서 출력 신호인 y 출력

활성화 함수의 종류

- 단층 퍼셉트론에서의 활성화 함수 → 계단 함수
- 다층 퍼셉트론에서의 활성화 함수 → 시그모이드 함수, 렐루 함수 등의 비선형 형태

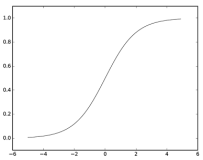
1. 계단 함수 (step function)



```
import numpy as np

def step_function(x):
  y = x > 0
  return y.astype(np.int)
```

2. 시그모이드 함수 (sigmoid function)



```
def sigmoid(x):
  return 1 / (1 + np.exp(-x))
```

$$h(x) = \frac{1}{1 + \exp(-x)}$$

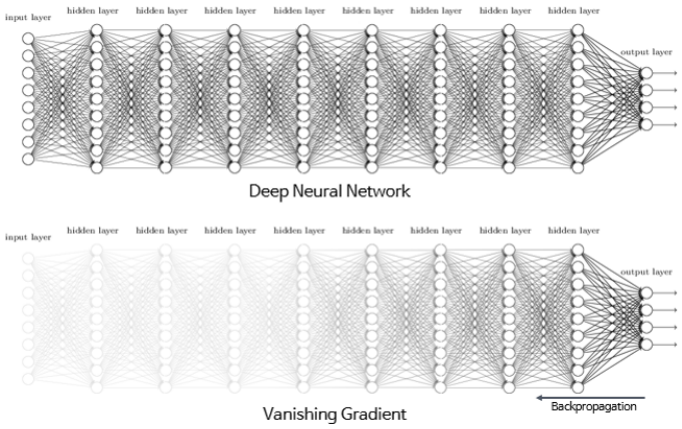
시그모이드 함수의 기울기 소실 문제

인공신경망의 학습과정은 다음과 같다:

1. 입력에 대해 순전파(forward propagation) 연산 진행

2. 순전파 연산을 통해서 나온 예측값 ~ 실제값 사이의 오차를 손실 함수(loss function)을 통해서 계산 → 손실 구함
3. 구한 손실을 미분해 기울기(gradient)를 구함 → **여기서 시그모이드 함수의 문제점 발생 !**
4. 출력층 → 입력층의 방향으로 가중치와 편향을 업데이트하는 과정인 역전파(backward propagation) 연산 진행 (경사하강법 사용)

위 그래프를 보면, 시그모이드 함수의 출력값의 경우, 0 또는 1에 가까워지면 그래프의 기울기가 완만해지는 모습을 볼 수 있다.
이 부분에서는 미분값이 0에 가까운 아주 작은 값으로, 역전파 과정에서 0에 가까운 값이 누적해서 곱해지게 되면, 앞단에는 기울기(미분값)이 제대로 전달되지 않는 상황이 발생한다. (= **기울기 소실(vanishing graident)**의 문제)
이렇게 되면, 가중치가 제대로 업데이트 되지 않아 학습이 제대로 되지 않는 문제가 발생한다.



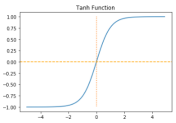
즉, 시그모이드 함수의 은닉층에서의 사용은 지양 ! + 주로 이진 분류를 위해 출력층에서 사용됨 !

계단 함수 VS. 시그모이드 함수

Aa 특징	계단 함수	시그모이드 함수
모양	0을 경계로 출력이 갑자기 바뀜	매끄러움 → 입력에 따라 출력이 **연속적으로** 변화
출력값	0과 1 중 하나의 값만 돌려줌	실수를 돌려줌
공통점	- 입력이 작을 때 → 출력 0에 가까움 - 입력이 클 때 → 출력 1에 가까움 - 출력은 항상 0과 1 사이 - 비선형 함수	- 입력이 작을 때 → 출력 0에 가까움 - 입력이 클 때 → 출력 1에 가까움 - 출력은 항상 0과 1 사이 - 비선형 함수

3. 하이퍼볼릭탄젠트 함수 (tanh)

: 모든 입력값을 -1과 1 사이의 값으로 변환하는 함수



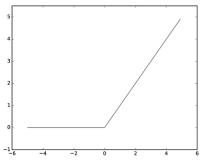
$$sinhx = sinhx = \frac{e^x - e^{-x}}{2}$$
$$coshx = \frac{e^x + e^{-x}}{2}$$
$$tanhx = \frac{sinhx}{coshx} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

```
def tanh(x):
  p_exp_x = np.exp(x)
  m_exp_x = np.exp(-x)
  y = (p_exp_x - m_exp_x) / (p_exp_x + m_exp_x)
  return y
```

하이퍼볼릭탄젠트 함수에서도 기울기 소실 문제는 발생하지만, 시그모이드 함수와 달리 미분시 최대값이 1 (시그모이드 함수의 최대값:0.25)로, 전반적으로 더 큰 값을 가짐. 따라서, 기울기 증상이 적은 편이며 은닉층에서의 사용이 더 선호된다.

4. ReLU 함수

: 입력이 0을 넘으면(양수) 입력을 그대로 출력, 0 이하(음수)면 0을 출력하는 함수



```
def relu(x):
  return np.maximum(0,x)
# np.maximum : 2개의 입력값 중 더 큰 값을 선택해 반환하는 함수
```

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

- 입력값이 음수면 기울기(미분값)도 0이 됨. → 이렇게 기울기가 0이 된 뉴런을 다시 회생하는 것은 매우 어렵기 때문에 이런 문제를 '**죽은 렐루(dying ReLU)**'라고 부름.

5. Leaky ReLU 함수

: 죽은 렐루를 보완하기 위한 ReLU의 변형 함수로, 입력값이 음수인 경우, 0이 아닌 0.0001과 같은 매우 작은 수로 반환하는 함수

```
def leaky_relu(x):
    return np.maximum(a*x, x)
```

- 식에서 a = leaky(새는) 정도를 결정하는 하이퍼파라미터 (대부분 0.01)

$$f(x) = max(ax, x)$$

6. 소프트맥스 함수 (softmax function)

: 출력층에서 사용되는 함수로, 다중 클래스 분류 문제 풀 시에 사용되는 함수

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

```
def softmax(a):
  exp_a = np.exp(a)
  sum_exp_a = np.sum(exp_a)
  y = exp_a / sum_exp_a
  return y
```

하지만, 위와 같이 일반적인 소프트맥스 함수 사용시 '**오버플로(overflow)**' 문제가 발생한다. 오버플로란, 지수 함수(np.exp)를 식에 사용함으로써 아주 큰 값을 출력하기 때문에 개선이 필요하다. → log 취해주기

$$\begin{aligned} y_k &= \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} = \frac{C \exp(a_k)}{C \sum_{i=1}^{n} \exp(a_i)} \\ &= \frac{\exp(a_k + \log C)}{\sum_{i=1}^{n} \exp(a_i + \log C)} \\ &= \frac{\exp(a_k + C')}{\sum_{i=1}^{n} \exp(a_i + C')} \end{aligned}$$

```
def softmax(a):
  c = np.max(a)
  exp_a = np.exp(a - c)  # 오버플로 대책
  sum_exp_a = np.sum(exp_a)
  y = exp_a / sum_exp_a
  return y
```

1. C라는 임의의 정수를 분자 분모에 각각 곱해줌.
2. C를 지수 함수 exp() 안으로 옮겨 log(C) 생성함.
3. log(C)를 C'로 대체함.

7. 항등 함수 (identity function)

: 입력값과 출력값이 항상 같은 함수

정리하자면,

1. **은닉층**에서 사용되는 활성화 함수 → 계단 함수, 하이퍼볼릭탄젠트 함수, 렐루 함수, 리키 렐루 함수
2. **출력층**에서 사용되는 활성화 함수 → 시그모이드 함수, 소프트맥스 함수, 항등 함수

   * 어떤 문제냐에 따라서 출력층에서 사용되는 활성화 함수가 달라짐 !

   - 회귀 (regression) → 항등 함수 (입력 신호를 그대로 출력)
   - 이진 분류 (binary classification) → 시그모이드 함수
   - 다중 분류 (multiclass classification) → 소프트맥스 함수

딥러닝 학습 방법

✔️ **학습이란?**
: 훈련 데이터(train data)로부터 가중치 매개변수의 최적값(손실함수의 결과값을 가장 작게 만드는 값)을 자동으로 획득하는 것

- 훈련 데이터 : 학습에 사용되어 최적의 매개변수 탐색하는 데이터
- 시험 데이터 : 훈련한 모델의 능력을 평가하는 데이터 (범용능력을 보기위함)

손실 함수 (loss function)

: 실제값과 예측값의 차이를 수치화해주는 함수

1. 평균 제곱 오차, MSE (Mean Squared Error)

: 선형 회귀 학습시 사용되는 손실 함수, 연속형 변수 예측에 사용됨

$$E = \frac{1}{2}\sum_{k}(y_k - t_k)^2$$

- y_k : 신경망이 추정한 값 (출력)
- t_k : 실제 정답
- k : 정답의 차원

```
def mean_squared_error(y,t):
  return 0.5 * np.sum((y-t)**2)
```

수

MSE의 사용법

```
model.compile(optimizer='adam', loss='mse', metrics=['mse'])
model.compile(optimizer='adam', loss=tf.keras.losses.MeanSquaredError(), metrics=['mse'])
```

2. 이항 교차 엔트로피, BCE (Binary Cross-Entropy)

: 이진 분류 학습시 사용되는 손실 함수, 출력층의 활성화 함수는 시그모이드 함수

BCE 사용법

```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
model.compile(loss=tf.keras.losses.BinaryCrossentropy(), optimizer='adam', metrics=['acc'])
```

3. 범주형 교차 엔트로피, CCE (Categorical Cross-Entropy)

: 다중 분류 학습시 사용되는 손실 함수, 출력층의 활성화 함수는 소프트맥스 함수

CCE 사용법

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
model.compile(loss=tf.keras.losses.CategoricalCrossentropy(), optimizer='adam', metrics=['acc'])
```

One-hot encoding(0과 1로 변환하는 작업)을 생략하고, 정수값을 가진 레이블에 대해 다중 클래스 분류하는 경우, `sparse_categorical_crossentropy` 사용함.

```
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['acc'])
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(), optimizer='adam', metrics=['acc'])
```

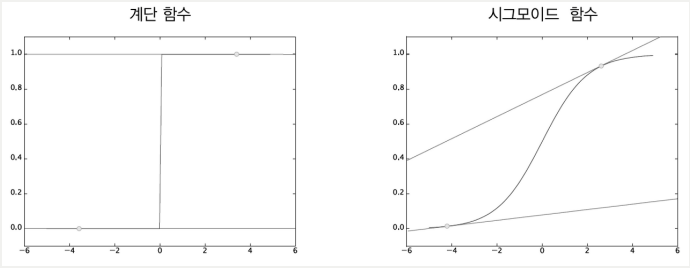
왜 '정확도'라는 지표 대신에 '손실 함수'를 사용하는 것일까?

: 신경망 학습에서 '미분'에 주목 !

즉, 최적의 매개변수(가중치나 편향)을 찾을 때 우리는 손실 함수를 가능한 작게하는 값을 찾는다.

매개변수의 미분을 계산해서 값을 갱신하는 과정을 반복하는데, 정확도를 지표로 삼으면 매개변수의 미분이 대부분의 장소에서 0이 되기 때문에 매개변수의 값을 갱신할 수 없다.

또한, 정확도는 매개변수의 미미한 변화에는 거의 반응을 보이지 않고, 반응을 보이더라도 값이 불연속적으로 갑자기 변한다. (이는 활성화 함수로 계단 함수를 잘 사용하지 않는 것과 같은 이유)



배치 크기에 따른 경사 하강법

✔️ **경사 하강법?**
: 기울기 값을 이용해서 나아갈 방향을 찾고, 최종적으로 함수의 최솟값을 찾는 방법

- **배치(batch)** : 매개변수 (가중치/편향)의 값을 조정하기 위해 사용하는 데이터의 양
  - 전체 데이터를 갖고 매개변수의 값 조정?
  - 정해준 일정 양의 데이터만 갖고 매개변수의 값 조정?

→ 배치 크기에 따라서 인공 신경망의 학습 방법이 달라짐 !

1. 배치 경사 하강법 (Batch Gradient Descent)

: 전체 데이터에 대한 1번의 훈련 횟수 = 1 epoch일 때, 1 epoch에 모든 매개변수 업데이트를 단 한 번 수행하는 경사 하강법

- 한 번의 매개변수 업데이트에 시간이 오래 소요됨
- 메모리 샤용량이 큼

```
model.fit(X_train, y_train, batch_size=len(X_train))
```

2. 확률적 경사 하강법 (Stochastic Gradient Descent)

: 배치 크기가 1인 경사 하강법 → 전체 데이터가 아닌, 랜덤으로 선택한 1개의 데이터에 대해서만 계산해서 매개변수를 업데이트하는 방법

- 더 적은 데이터 사용 → 더 빠른 수행 시간
- 매개변수의 변경폭이 불안정함


```
model.fit(X_train, y_train, batch_size=1)
```

3. 미니 배치 경사 하강법 (Mini-batch Gradient Descent)

: 배치 크기를 사전에 정의해 해당 데이터 개수만큼에 대해 계산해서 매개변수를 업데이트하는 방법

- 전체 데이터를 계산하는 배치 경사 하강법보다 빠름
- 배치 크기가 1인 확률적 경사 하강법보다 안정적임
- 배치 크기 정하는 방법
  : 2의 n제곱에 해당하는 숫자로 선택하는 것이 보편적

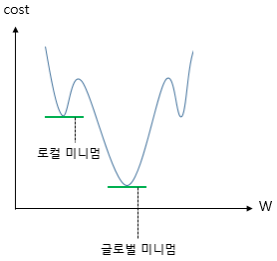
```
model.fit(X_train, y_train, batch_size=128)
```

옵티마이저 (optimizer)

경사 하강법 알고리즘에 변화를 주는 요소

1. 모멘텀 (Momentum)

: 경사 하강법에 관성을 더해주는 요소 → 계산된 접선의 기울기에 한 시점 전의(t-1) 접선의 기울기값을 일정한 비율만큼 반영함



- 글로벌 미니멈 → 전체 함수에 걸친 최솟값
- 로컬 미니멈 → 특정 구역에서의 최솟값

로컬 미니멈에 도달했을 때에 글로벌 미니멈으로 잘못 인식해 탈출하지 못하는 과정에서 모멘텀의 힘을 빌릴 수 있다.
→ 관성의 힘으로 로컬 미니멈을 빠져나와서 글로벌 미니멈으로 갈 수 있는 효과를 얻을 수 있음.

```
tf.keras.optimizers.SGD(lr=0.01, momentum=0.9)
```

2. 아다그라드 (Adagrad)

: 학습에 사용되는 모든 매개변수들은 각자 의미하는 바가 다른데, 모든 매개변수에 동일한 학습률을 적용하는 것은 비효율적임 → Adagrad로 각 매개변수에 서로 다른 학습률 적용함.

- 변화가 많은 매개변수 → 작은 학습률
- 변화가 적은 매개변수 → 높은 학습률

```
tf.keras.optimizers.Adagrad(lr=0.01, epsilon=1e-6)
```

3. RMSprop

: Adagrad로 학습 진행시 학습을 진행할수록 학습률이 지나치게 떨어질 수 있음 → 단점을 개선함.

- `rho` : 학습률 감소에 사용되는 파라미터, 각 시점에 유지되는 기울기의 비율 (이전 시점의 학습률을 90% 유지함)

```
tf.keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-06)
```

4. 아담 (Adam)

: Momentum + RMSprop을 합친 옵티마이저 → 방향과 학습률을 모두 잡기 위한 방법

```
tf.keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

이렇게 해서 대략적으로 딥러닝과 신경망 학습에서 중요한 개념들에 대한 복습을 끝마치도록 하겠습니다!
(더 궁금한 부분은 따로 Q&A 노션 페이지에 남겨주시거나 따로 공부해보시길 바랍니다!)

자연어처리(natural language processing)이란?

- **자연어** (natural language) : 우리가 일상생활에서 사용하는 언어들
- **자연어처리** : 자연어의 의미를 분석해 컴퓨터로 다양한 문제를 해결하는 것

자연어처리 과정

1. 2주차 **Preprocessing (텍스트 데이터 전처리)**
    - 불용어 제거 (stopwords removal)
    - 토큰화 (tokenization)
    - 형태소 분석 (stemming)
    - 표제어 추출 (lemmatization)
    - 정규 표현식 (regular expression)
    - 원핫 인코딩 (one-hot encoding)
    - 패딩 (padding)
2. 2주차 3주차 **Vectorization**
    - 언어 모델
    - Count vectorization
    - TF-IDF
    - 벡터의 유사도
3. 4주차 **Embedding (워드 임베딩)**
    - word2vec
    - doc2vec
    - 글로브 (GloVe)
    - 패스트텍스트 (FastText)
    - 엘모 (ELMo)
4. 5주차 6주차 **Modeling (자연어처리 모델링)**
    - RNN / LSTM (순환신경망)
    - Attention, Transformer
    - 사전학습된 언어모델 (BERT, GPT 등)

실전 프로젝트

7주차 8주차

6주차까지 배운 이론과 코드 구현 과제를 토대로 팀별 (아마 한 팀으로 진행될듯 합니다 !) KUBIG CONTEST 프로젝트를 진행해주시면 됩니다. ▫ KUBIG CONTEST

✔️ **예) 텍스트 분류와 관련된 프로젝트 (기사의 감성 분류)**
1. 기사 데이터 크롤링/수집
2. 기사의 각 문장별로 형태소 분석 + 불용어 제거 등의 데이터 전처리 과정
3. 인코딩으로 변환해주는 vectorization 진행
4. 각 문장별로 서로 다른 길이를 갖기 때문에 일정한 길이를 갖도록 padding 진행
5. 처리된 문장 벡터들을 특정 차원으로 임베딩 진행 (pre-trained embedding인 GloVe 사용)
6. BERT를 이용해서 학습, 성능 평가

- **주제** : 자유 주제 (팀원끼리 상의 하에 주제와 사용 모델 선정)
- `2/24` → 중간 발표 (NLP 세션 시간에 중간 발표)
- `3/3` → 최종 발표 (KUBIG 전체 세션 시간에 최종 발표)

모두에게 스펙이 될 수 있는 실전 프로젝트 기회라고 생각합니다 !

자연어처리의 응용 분야

자연어처리로 해결 가능한 문제들은 무엇이 있을까 ?

1. 텍스트 분류 (text classification)

> 특정 문장/문서를 어떠한 카테고리로 분류하는 문제

예) 스팸 메일 분류

2. 감성 분석 (sentiment analysis)

> 텍스트에서 어떤 주제에 대한 주관적인 인상, 감정, 개인의 의견을 추출하는 문제

예) 사업보고서의 긍/부정을 파악한 기업 평가, 사람과 대화하는 인공지능이 대화 상대방의 감정을 파악해 대화 주제를 바꾸는 것

3. 내용 요약 (text summarization)

- 추출 요약 (extractive) : 문서에서 중요하다고 생각되는 문장들을 뽑아내 요약문으로 이용
- 생성 요약 (abstractive) : 요약문을 새롭게 생성하는 방법

4. 기계 번역 (machine translation)

> 한 언어를 다른 언어로 번역하는 문제

5. 질의 응답 (question answering)

> 특정 종류의 맥락(context)와 질문을 이용해 이에 대한 정답을 내는 문제

KUBIG 22-1 NLP 분반의 역할 : 자연어처리에 대한 전반적인 이론과 기술을 익히고, 자신의 관심 응용 분야를 찾아 가는 과정 !