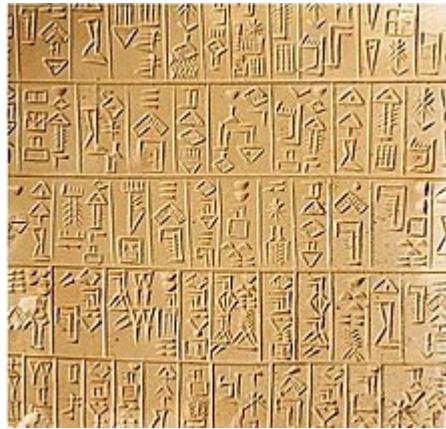


4.1 언어를 인식하는 인공지능

옛날의 인류는 정보를 교환하기 위해 사람과 사람의 의사소통에 의존할 수밖에 없었습니다.

문자가 생긴 뒤로는 정보를 기록할 수 있게 되었고, 사람들은 궁금한 점을 책 등의 기록물로부터 해결할 수 있었습니다.

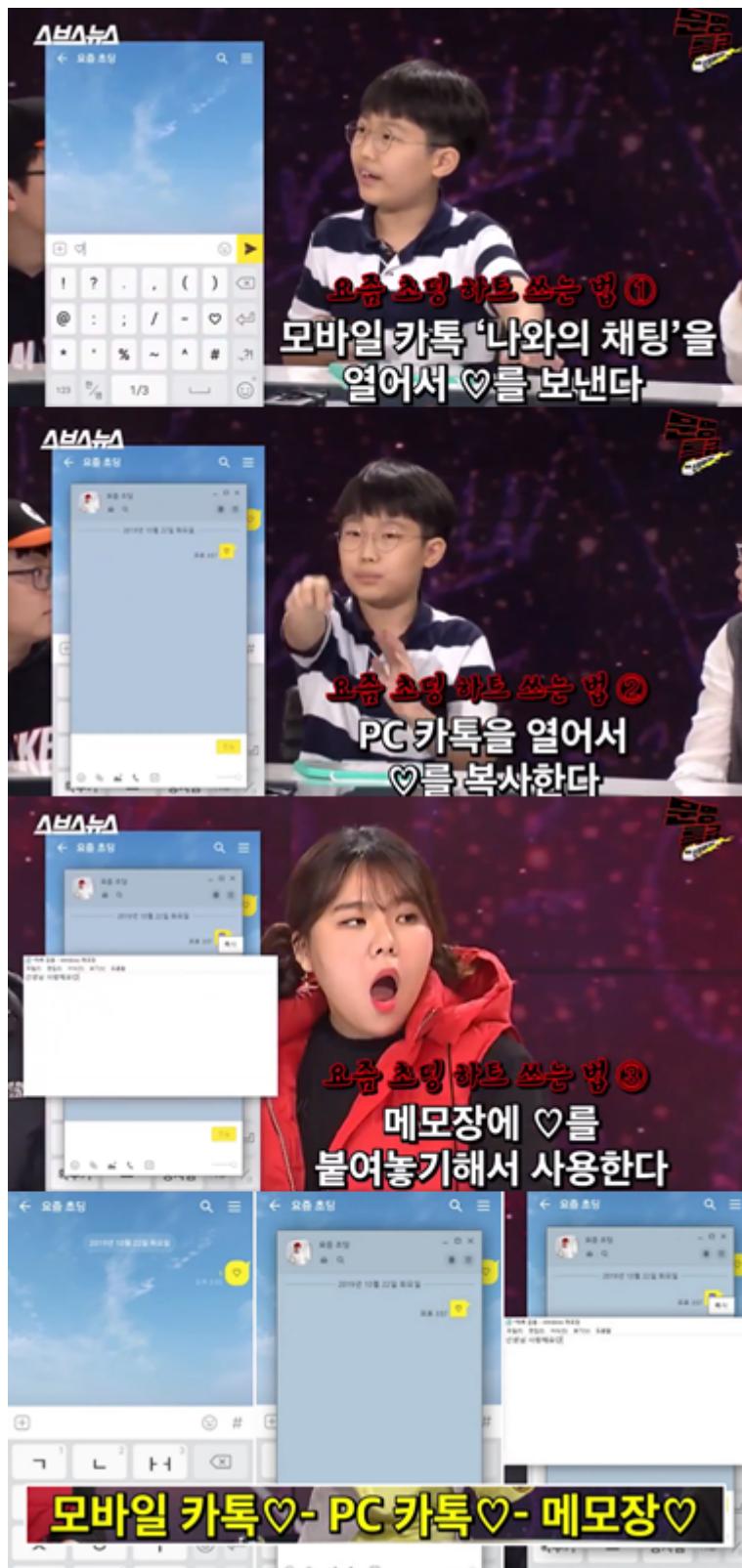


[그림 1. 인류 최초의 기록문자 (기원전 26세기 수메르인들의 쌓기 문자)]

요즈음의 우리는 궁금한 것이 생기면 책이나 주변 지인에게 묻기보다 검색엔진을 활용합니다.

검색엔진은 한 권의 책이나 한 명의 사람이 담을 수 있는 이상의 지식을 훨씬 빠르고 폭넓게 제공합니다.

미래엔 어떻게 될까요? 밀레니얼 세대 다음은 Z세대라고 합니다.



[그림 2. 요즘 세대가 PC에서 ♥를 쓰는 방법]

지금의 초등학교 저학년정도인 이들은 컴퓨터보다도 모바일 기기에 더 익숙합니다.
이들은 질문이 생기면 AI스피커에게 음성으로 물어봅니다(짱구야~ OO에 대해 알려줘~).



[그림 3. 요즘 아이들이 숙제를 하는 방법]

답변을 구하는 것에서 조차 점점 사람의 수고를 줄이는 방향으로 기술이 발전하고 있습니다.

AI 스피커에 물어본다면 컴퓨터나 모바일 기기를 켜서 자판을 치고 검색할 필요도 없이 바로 대화로 답변을 얻을 수 있죠.
하지만 집에 AI 스피커가 있어서 이것저것 테스트해보신 분들은 단순 질문답변, 그 이상의 것은 기대하기 어렵다는 것을 공감하실겁니다.

AI 스피커와 한 문장씩 주고받는 대화는 '대화'라고 하기엔 너무나 기초적인 수준입니다.

근미래에 기계와 정말로 자연스럽게 대화를 하며 우리의 질문에 대한 답변을 얻을 수 있을까요?

아니 그 이전에, 기계가 사람의 언어를 올바로 인지하고 맥락을 이해할 수 있을까요?

AI 스피커는 우리가 무엇을 궁금해하는지 어떻게 인식하고 답변을 해준 걸까요?

4.1.1 자연어이해(NLU; Natural Language Understanding)

자연어이해, 또는 자연어처리(NLP; Natural Language Processing)라는 말을 많이 들어보셨을 겁니다.

말 그대로 자연어를 이해하거나 처리하는 기능에 대한 말인데요, 주체는 '기계'입니다.

기계가 자연어를 이해한다는 것은 어떤 걸까요? 명칭에 대해 잠깐 짚고 넘어가겠습니다.

4.1.1.1 자연(Natural)

자연어(Natural Language; 自然語)란, 사람들이 일상적으로 쓰는 언어로, 기원을 찾기 힘들며 자연 발생한다는 특징이 있습니다.

예를 들어 우리가 쓰는 한국어나 영어, 중국어, 일본어 등이 그 예입니다.

반대되는 개념으로는 **인공어**(Constructed Language; 人工語)가 있습니다.

이는 의도와 목적에 따라 인공적으로 만든 언어로, 프로그래밍 언어와 같은 기계어라든가 전자구인의 공용 언어 사용을 위해 만들어진 에스페란토 등이 이에 해당합니다.

“팀장님 이번 회식 소고기집 가도 돼요?”
“이번 달 법카 얼마 남았어?”

```

1 def is_ok(menu, card) :
2     if card.limit > menu.price :
3         return "OK"
4     else :
5         return "SORRY"
6
7 print is_ok(menu("소고기"), get_team_card(2018,5))
8

```

[그림 4. 자연어와 인공어]

4.1.1.2 언어(Language)

'언어'라는 데이터 타입이 갖는 특성은 무엇일까요?

언어는 말하고 듣는 음성과, 쓰고 읽는 문자로 이루어져 있습니다.

예를 들어 여러분이 영화 '아이언맨'을 친구에게 설명해준다고 가정해봅시다.

친구는 영화 내용을 전혀 모르기 때문에 여러분의 설명으로만 내용을 상상할 겁니다.

어떻게 영화의 줄거리를 요약할 수 있을까요?

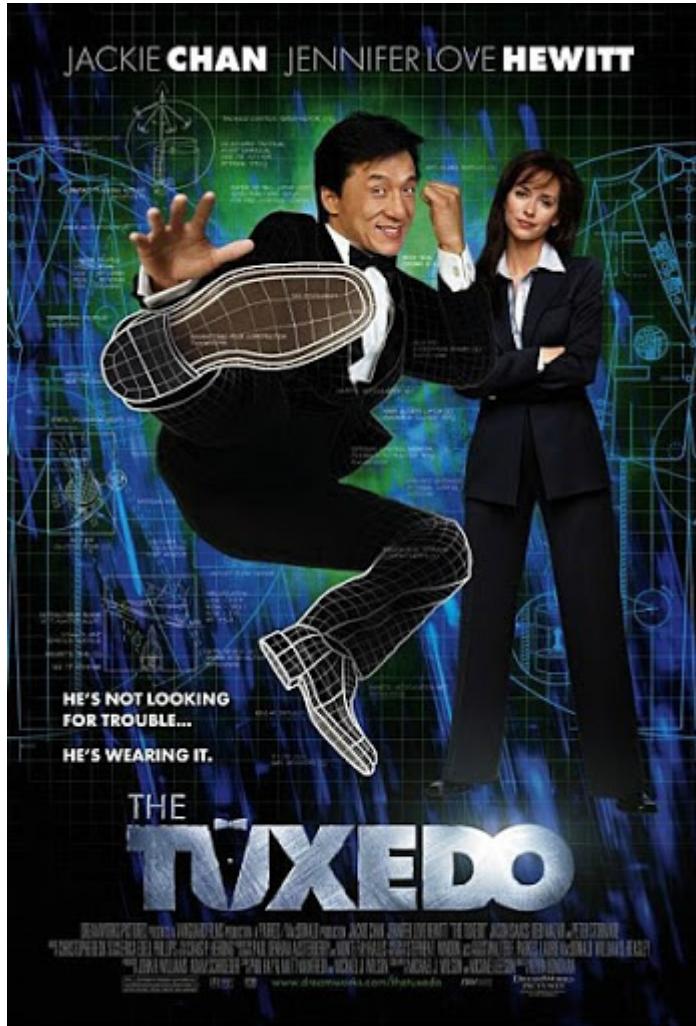
역만장자 천재 발명가인 토니 스타크가 심장에 치명적인 상처를 입은 자신의 목숨을 지키며
아이언맨 강화 수트를 제작한다. 세계를 지킬 과학의 결정체인 아이언 맨은 범죄와 싸워나간다.

친구는 한 줄 줄거리만으로 내용을 상상할 수 있을까요?

아마 과학의 결정체인 강화수트라는게 뭔지부터 혼란을 겪을거라고 생각합니다.

어쩌면 '수트'라는 말로부터 과거 성룡이 출연한 영화 '럭시도'같은 의상을 생각할 수도 있구요..

참고: 텍시도



하지만 영화 포스터나 스틸컷을 보여줄 수 있다면 어떨까요?

아 강화수트라는게 저런 로봇같은 수트를 말하는거였구나, 주인공이 저렇게 생겼나보네, 딱보니 가운데 있는 놈이 악당인가보다..

더 많은 정보를 얻을 수 있을겁니다.



[그림 5. 아이언맨 영화포스터]

아니면 아예 영화를 보여줄 수 있다면요? 설명할 수고도 줄어들고, 훨씬 더 깊게 이해할 수 있습니다.



[그림 6. 뭐야... 수트를... 본인이 입는다는거였어...?]

백 번 듣는 것보다 한 번 보는 것이 낫다는 말도 있습니다.

영화라는 다차원 데이터(동영상)는 많은 정보를 포함할 수 있지요.

반면 텍스트 형식으로 표현된 언어 데이터는 그 많은 정보를 굉장히 함축하여 인코딩한 자료입니다.

사람은 언어가 표현하는 대상의 내용을 머리속에서 3D로 자유자재 구현할 수 있지만 기계는 그러한 사전정보가 없습니다.

기계가 언어를 인식하고 맥락에 포함된 내용을 이해하기란 쉽지 않겠죠?

4.1.1.3 이해(Understanding)

이해한다는 것은 처리한다는 것보다 한 단계 더 높은 수준을 요구합니다.

처리는 기계적으로 규칙을 따라 얼마든지 수행 가능하겠지만 이해는 맥락 내용에 대한 파악을 전제로 합니다.

요즘은 자연어 처리(NLP)나 자연어 이해(NLU)라는 용어를 거의 구분없이 사용하고는 있습니다.

하지만 오늘날 인공지능에게 기대하는 것은 단순 규칙에 따른 처리가 아닌, 내용과 맥락을 이해하여 업무를 수행하는 것 이므로 '자연어 이해'라는 표현을 이용하도록 하겠습니다.

엄밀히 나누자면 NLP 안에 NLU가 포함됩니다.

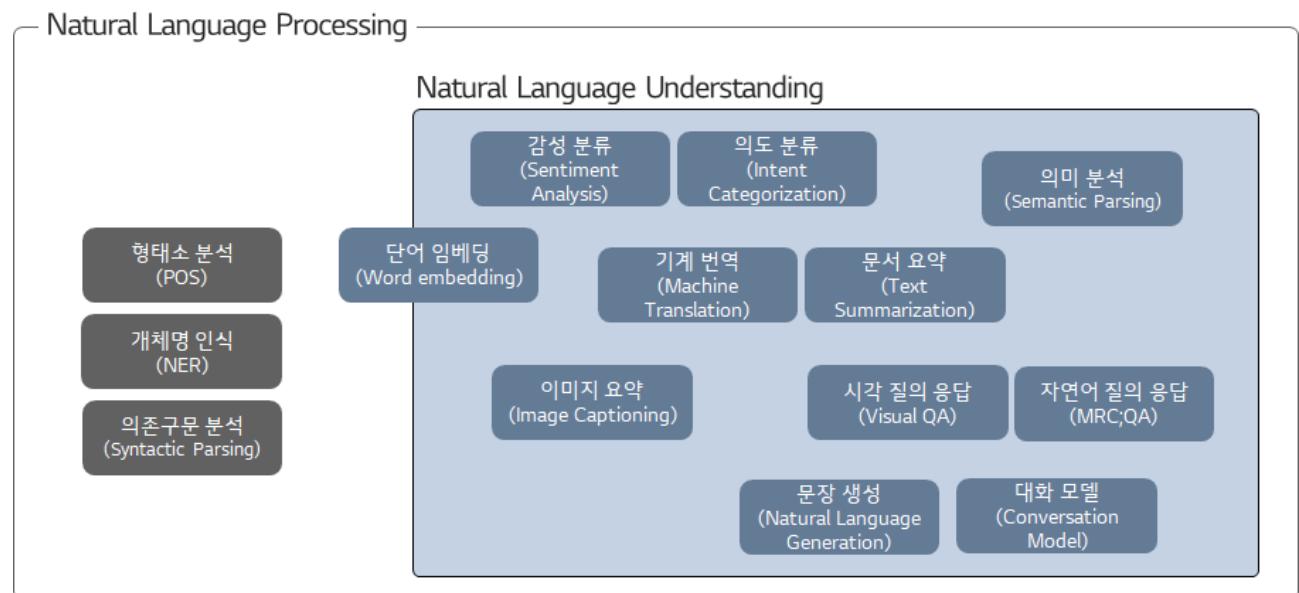
NLP에만 해당하는 부분은 언어의 형식, 구문론과 관련된 기능들입니다.

문장을 형태소 단위로 쪼갠다든지(POS), 구문 분석을 수행한다든지 하는 기능입니다.

반면 NLU는 문장 내 표현된 감성을 인식하는 감성분류라든지, 문서 내 중요한 부분을 캐치하는 요약 등의 기능을 포함합니다.

그런데 이마저도 요즘은 딥러닝 기술의 발달로 NLP/NLU의 경계가 애매해지고 있습니다.

전통적인 NLP 영역쪽도 딥러닝 기반의 기술을 적용했을 때 더 좋은 성능이 나오고 있거든요.



[그림 7. NLP와 NLU, 하지만 최근 두 경계를 나누는 것은 상당히 모호하다]

어쨌든 기계가 언어를 이해하기 위해서는 일단 언어를 인식해야 합니다.

우리가 배웠던 인공신경망에 데이터를 넣어 딥러닝 모델을 학습시키는 방식으로요.

하지만 인공신경망에 데이터를 태우려면 데이터의 생김새가 인공 뉴런이 계산(가중합+비선형함수)할 수 있는 형태여야 합니다.

텍스트로 된 언어 데이터를 계산할 수 있을까요?

(참고: 음성의 경우, STT(speech-to-text) 기술을 활용하여 텍스트로 변환 가능합니다)

4.2 기계에게 사람의 언어를 인식시키려면?

언어를 계산 가능한 형태로 바꾸려면 어떻게 해야 할까요?

인공신경망에 태울 데이터는 실수로 이루어진 벡터나 매트릭스 등의 타입이어야 합니다. [지난 게시물](#)²⁸의 이미지 데이터처럼요!

우리는 이러한 다차원의 실수 데이터를 텐서(Tensor)라고 하겠습니다.

이미지 데이터는 픽셀값을 2차원, 또는 3차원의 텐서로 만들어 인공신경망에 태울 수 있었습니다.

언어의 경우도 텐서로 바꾸어 인식시킬 수 있습니다. 예를 들어 간단한 문장 하나를 예시로 들겠습니다.

여름휴가는 시작하기 전이 가장 즐겁다

4.2.1 Tokenizing(Parsing)

한 덩이로 되어있는 문장을 인공신경망에 인식시키기 위해서 세부 단위로 쪼개는 작업이 필요합니다.

이를 토크나이징 또는 파싱이라고 하며, 이 쪼개진 단위를 토큰(token)이라 부릅니다.

어떻게 쪼개면 좋을지는 언어의 특징과 수행하고자 하는 태스크, 데이터의 특징에 따라 다릅니다.

어절

여름휴가는 + 시작하기 + 전이 + 가장 + 즐겁다

형태소

여를휴가/NNG + 는/JX + 시작/NNG + 하/XSV + 기/ETN + 전/NNG + 이/JKS + 가장/MAG + 즐겁/VNA + 다/E

음절

여 + 름 + 휴 + 가 + 는 + + 시 + 작 + 하 + 기 + + 저 + 이 + + 가 + 장 + 즐 + 겁 + 다

자소

○ + ± + + ± ± + - + □ + ± ± + ± ± + | + ± + - + ± + + ± + | + + ± ± + | + ± + ± + ...

[그림 8. 여러 방식에 따른 문장 토크나이징 예]

영어의 경우에는 어절, 즉 띄어쓰기 단위로 잘라도 좋고 중국어의 경우에는 한 문자 한 문자에 뜻이 담겨있기 때문에 글자 단위로 쪼갤 수 있습니다.

일반적으로 한국어의 경우엔 교착어라는 언어 특성상 문장을 형태소 단위로 자르거나 음절단위로 자릅니다. (참고: 교착어²⁹⁾)

28 <https://tec.lgcns.com/tec/pages/viewpage.action?pageId=90359347>

28 <https://tec.lgcns.com/tec/pages/viewpage.action?pageId=90339347>
29 <https://ko.wikipedia.org/wiki/%EA%B5%90%EC%B0%A9%EC%96%B4>

뉴스기사나 논문처럼 정제된 글인 경우 형태소 단위로 잘 쪼개질 수 있지만 채팅이나 SNS 글의 경우 맞춤법을 제대로 지키지 않고 작성하는 경우가 많아 음절 단위로 쪼개는 것이 좋습니다.

요새는 이 둘의 장점을 취한 토크나이징 기법(등장빈도가 높은 N-gram 단위의 토크나이징 등)도 많이 활용합니다.

어절

- 한국어에 적합 X

형태소

- 정제된 글에 강하고 맞춤법 파괴, 신조어 등에 취약
- 형태소 수만큼 vocabulary list를 가짐
- 형태소 분석기에 따라 결과 달라질 수 있음

음절

- vocabulary list 13,000개 수준
- 오탏에 덜 민감

자소

- vocabulary list 몇 백 개 수준



[그림 9. 데이터의 특징에 따라 잘 맞는 토크나이징 단위]

4.2.2 워드임베딩(word embedding)

이제 한 덩이의 문장이 토큰들로 쪼개졌습니다.

쪼개진 토큰들은 인공신경망이 계산할 수 있도록 벡터로 바꾸어줘야 하는데요, 토큰을 벡터화하는 것을 워드임베딩이라고 부릅니다.

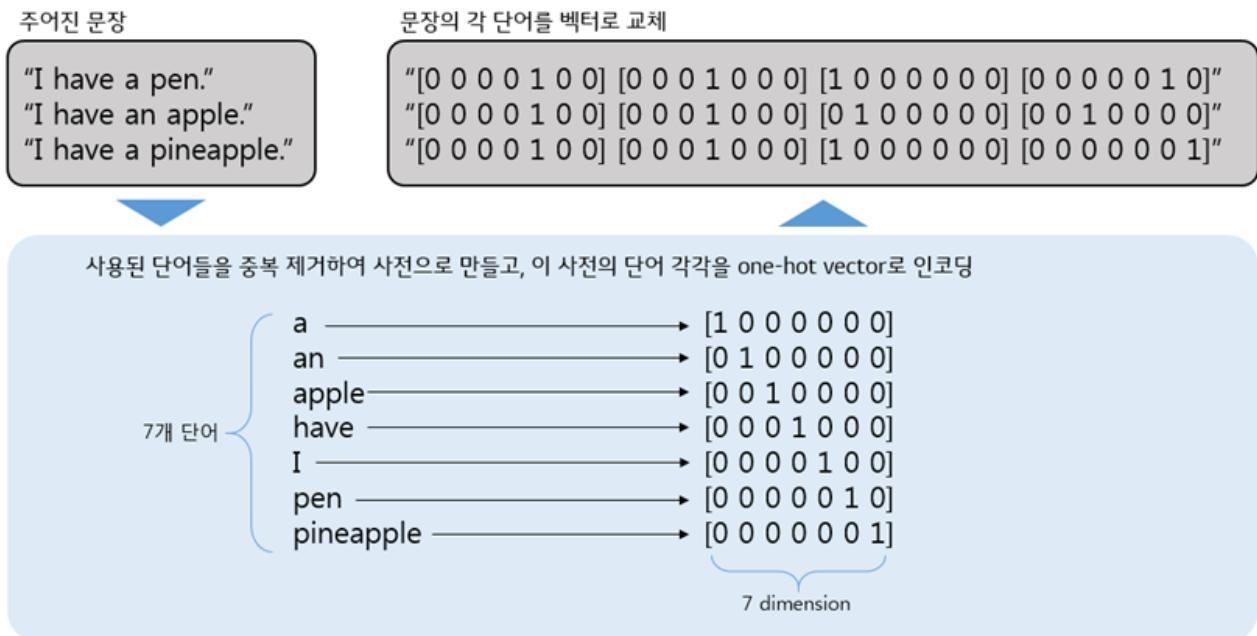
우리가 읽을 수 있는 토큰(보통 단어 단위)을 다차원의 벡터 공간의 한 점으로 매핑한다, 꽂아 넣는다는 의미에서 워드 임베딩이라는 표현을 씁니다.

토큰을 벡터화하는 워드임베딩 기법에는 여러가지가 있습니다.

4.2.2.1 원-핫 인코딩(One-hot Encoding)

토큰을 벡터화하는 가장 쉽고 직관적인 방법은 우리가 알고있는 모든 토큰들을 쭈루룩 줄세워 사전을 만들고, 순서대로 번호를 붙이는 작업입니다.

1. 내가 가진 모든 문장, 문서들을 토크나이징한 뒤 토큰들의 종복을 제거한 사전을 만듭니다.
2. 사전의 길이(중복 제외 토큰 수)만큼 벡터를 정의하고, 벡터에서 토큰의 순서에 해당하는 위치만 1, 나머지는 0의 값으로 채워 벡터를 구성합니다.
3. 내가 가진 문장, 문서의 토큰들을 해당 벡터로 교체합니다.



[그림 10. 세 영어 문장을 기준으로 원-핫 인코딩 변환을 하는 작업 순서도, 이 경우 토크나이징 단위는 어절(띄어쓰기 단위)]

하지만 원-핫 인코딩 방식은 아주 간단하긴 합니다만 큰 문제가 있습니다.

토큰이 다양하고 수가 많을수록 토큰 하나를 표현하기 위해서 굉장히 길이가 긴 벡터를 필요로 합니다.

한국어 어휘 표준국어대사전 기준 51만개의 단어가 있다고 합니다.

한 단어를 표현하기 위해서는 무려 51만 길이를 갖는 벡터를 활용해야합니다.

그리고 그 벡터의 대부분은 0의 값으로, 길이에 비해 가진 정보는 턱없이 부족한 편이죠(very sparse).

참고: 왜 임베딩을 1,2,3,... 으로 하지 않고 [1,0,0,...], [0,1,0,...]과 같이 하나요?

그렇게 했을 때 스칼라 값 하나로 모든 토큰을 표현할 수는 있겠지요.

하지만 각 토큰들은 서로 대등하며, 동일한 양의 정보를 가져야 합니다.

예를 들어 [그림 10]의 'apple' 토큰을 3, 'pineapple' 토큰을 7이라고 한다면 어떻게 될까요?

신경망에서 파인애플이 사과보다 +4만큼 더 크게 계산되어야 할 어떤 수치적인 우월함도 없습니다.

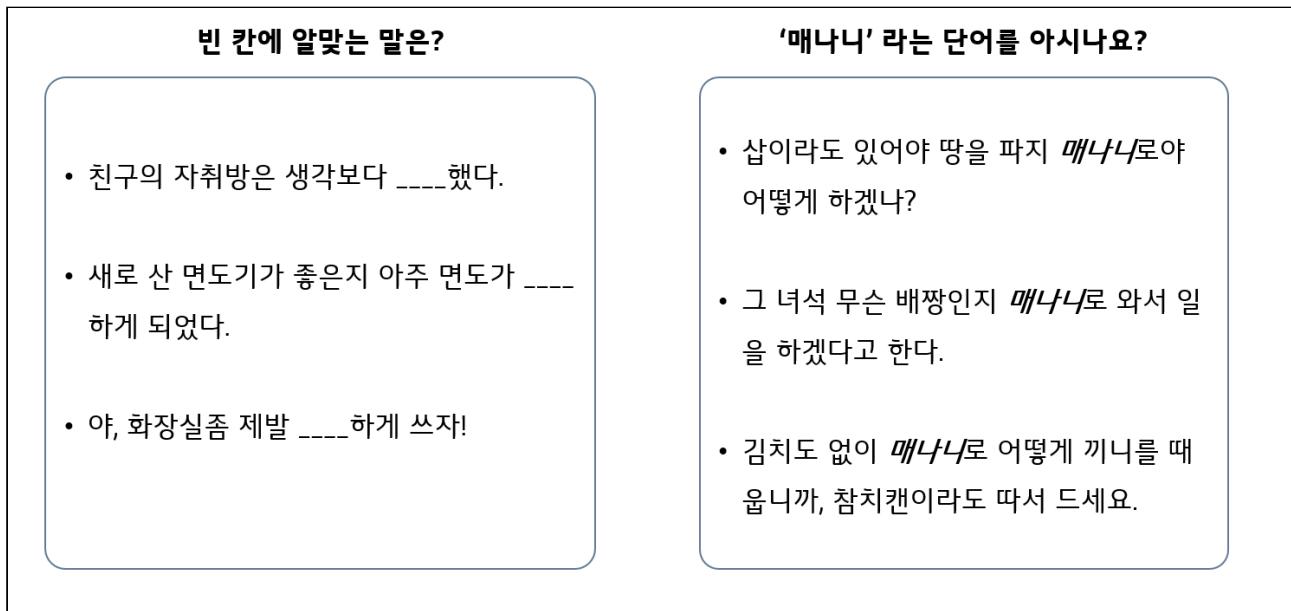
원-핫 인코딩 방식으로 토큰을 임베딩하게되면 모든 토큰간 $\sqrt{2}$ 만큼의 서로 동일한 거리를 갖습니다.

이로서 원-핫 인코딩 방식은 여러 토큰을 동등하게 취급합니다.

그래서 이를 개선하기 위해, 벡터의 길이도 작으면서 정보도 많이 담긴(dense) 두 가지 방식이 나왔습니다.

4.2.2.2 CBOW와 SKIPGRAM

다음 두 예를 보겠습니다.



[그림 11. CBOW와 SKIPGRAM 알고리즘 학습 방식을 비유한 사례, [자료³⁰](#)]

첫 번째 예제의 빈 칸에 들어갈 말이 뭐가 있을까요?

정답이 있는 건 아니지만, 아마 '깨끗', '깔끔', '청결' 등등이 빈칸에 들어갈 수 있는 단어가 되겠습니다.

이들은 뉘앙스는 조금 다를지라도 의미는 비슷합니다.

두 번째 예제도 살펴보겠습니다.

'매나니'라는 단어를 아시나요?

자주 쓰는 단어는 아니기 때문에 아마 모르는 분이 대다수일겁니다.

하지만 매나니가 정확히 뭔지는 몰라도, 사용 예시를 보면 대충 맨땅에서, 밑바닥부터, 맨손으로... 뭔가 그런 의미를 담은 말이겠죠?

사람은 언어를 이렇게 배웁니다.

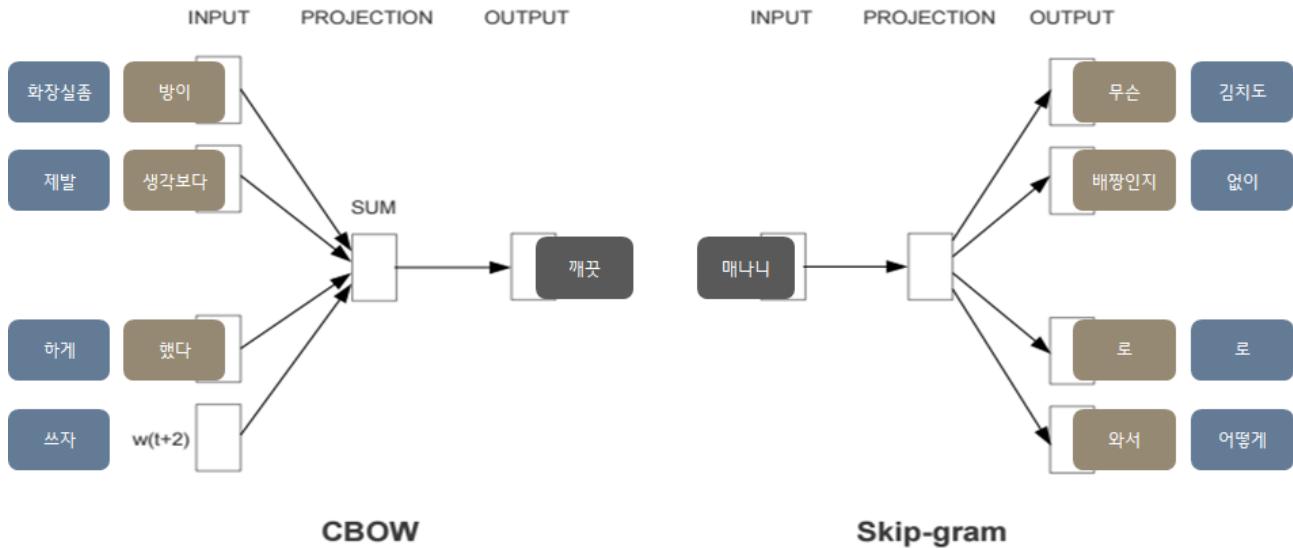
우리는 모든 단어를 사전을 뒤져가며 정확한 의미를 습득하는 게 아닙니다.

비슷한 문맥에서 비슷한 위치에 등장하는 단어끼리는 유사한 의미를 가진다는 걸 알 수 있으니 '청결'이라는 단어를 만일 몰라도, '깨끗'이 나올만한 위치에 쓰인다면 그 뜻을 대강 알 수 있습니다.

또 '매나니'라는 단어의 사전적 정의를 몰라도 등장하는 문맥을 여럿 보면 대강의 의미를 뽑아낼 수 있죠.

이런 과정을 알고리즘으로 구현한 것이 각각 CBOW와 SKIPGRAM입니다.

³⁰ <https://www.lucypark.kr/docs/2015-pyconkr/#20>



[그림 12. CBOW와 SKIPGRAM]

둘 중 어떤 방식을 이용하든, 먼저 단어(토큰)를 특정 길이를 가진 임의 벡터로 만들어줍니다.

이 때의 벡터 길이는 사람이 지정하며, 원-핫 인코딩 방식보다는 훨씬 작은 길이입니다. (일반적으로 몇십~몇백 단위 정도)

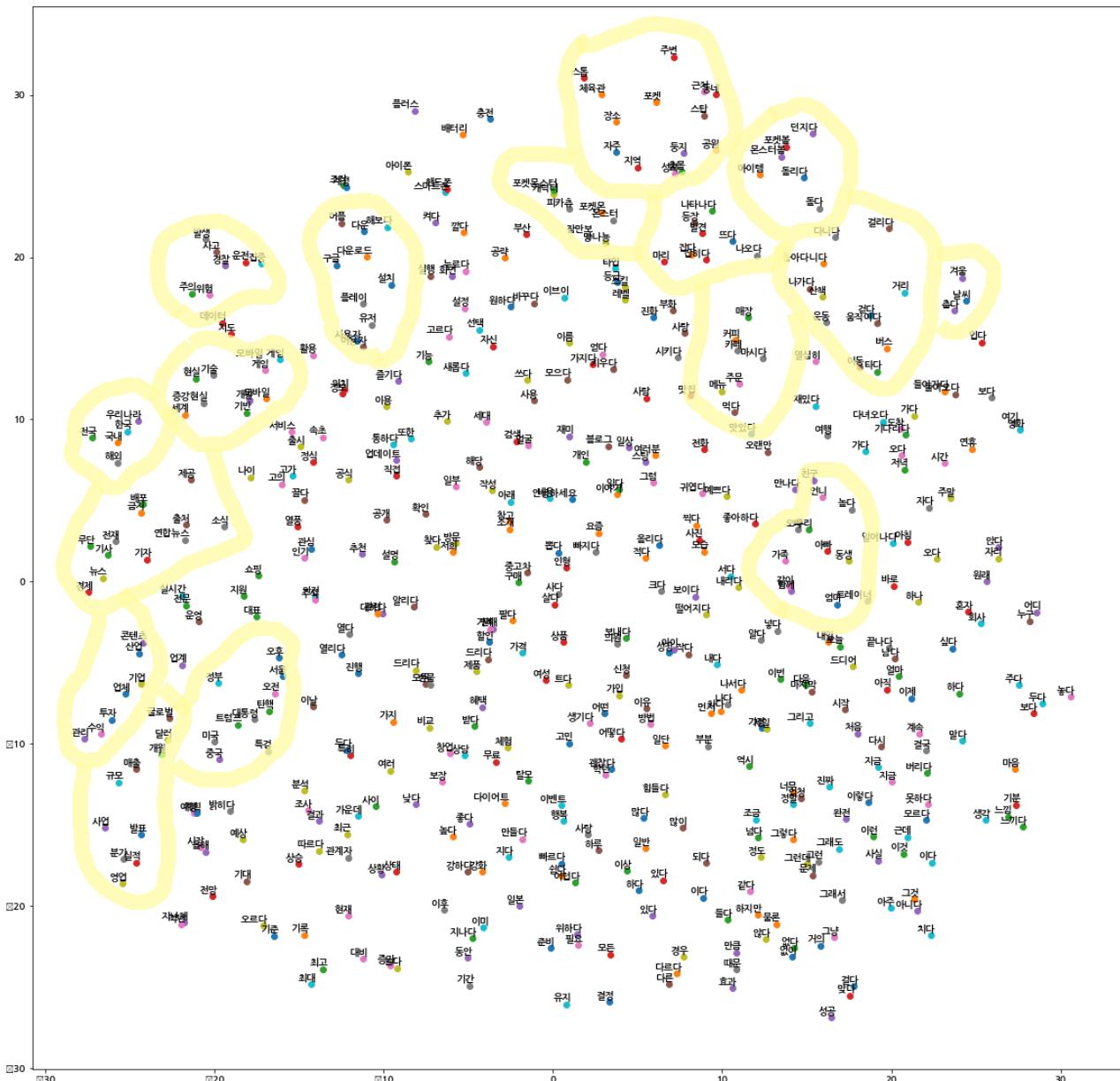
그 뒤, CBOW 방식은 인공지능에게 문장을 알려주되 중간중간 빈칸을 만들어 들어갈 단어(토큰)를 유추시킵니다.

SkipGram 방식은 인공지능에게 단어(토큰) 하나를 알려주고, 주변에 등장할만한 그럴싸한 문맥을 만들도록 시킵니다.

많은 문장을 학습시키면 시킬수록 더 좋은 품질의 벡터가 나옵니다.

아래 예시는 '포켓몬고' 게임이 출시될 당시의 소셜 데이터로 학습한 워드임베딩 결과를 시각화한 것입니다.

학습 방식은 CBOW 활용, 길이 300의 임베딩 벡터를 PCA를 통해 2차원으로 변환하였습니다.



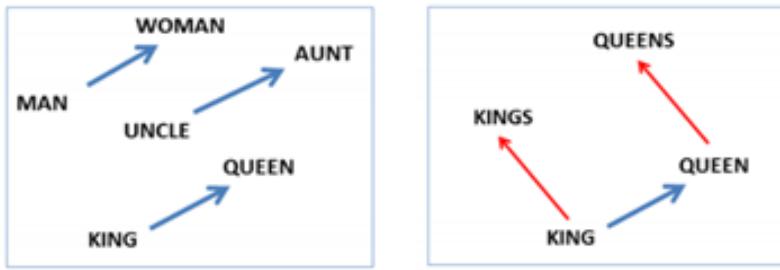
[그림 13. 포켓몬고 워드임베딩 시각화(AI기술팀), 클릭시 크게 보입니다]

신기하게도 지역과 관련된 단어, 비슷한 동작과 관련된 단어들이 유사한 벡터 공간으로 임베딩되었습니다.

즉, 유사한 의미를 갖는 토큰은 유사한 벡터값을 가지도록 학습이 된 것이지요.

원-핫 인코딩에 비해 벡터의 길이가 훨씬 작아(51만 → 300) 저장공간을 효율적으로 사용할 뿐만 아니라, 비슷한 의미의 토큰이 인공신경망에서 비슷한 연산결과값을 가지도록 하는 효과도 얻을 수 있습니다.

놀라운 점은 토큰끼리의 의미 연산이 가능하다는 것인데요, 이를테면 'LG전자'-'LG'+'삼성'='삼성전자' 같은 벡터 연산이 가능합니다.



[그림 14. 잘 학습된 워드벡터는 의미 연산이 가능하다]

이해를 돋기 위해, 한국어 벡터에 대해 연산을 해볼 수 있는 사이트가 있어 소개해드립니다. ([링크³¹](#))

이 세가지 전통적인 워드임베딩 기법 외에도 맥락을 고려하여 조금 더 스마트하게 벡터화하는 다양한 기법들이 최근에 많이 등장했습니다.

기회가 되면 다른 포스트에서 추가로 더 다루도록 하겠습니다 😊.

4.3 다양한 자연어이해 과제들

텍스트를 벡터로 바꾼 뒤라면 인공신경망을 어떻게 구성하느냐에 따라 다양한 태스크를 수행할 수 있습니다.

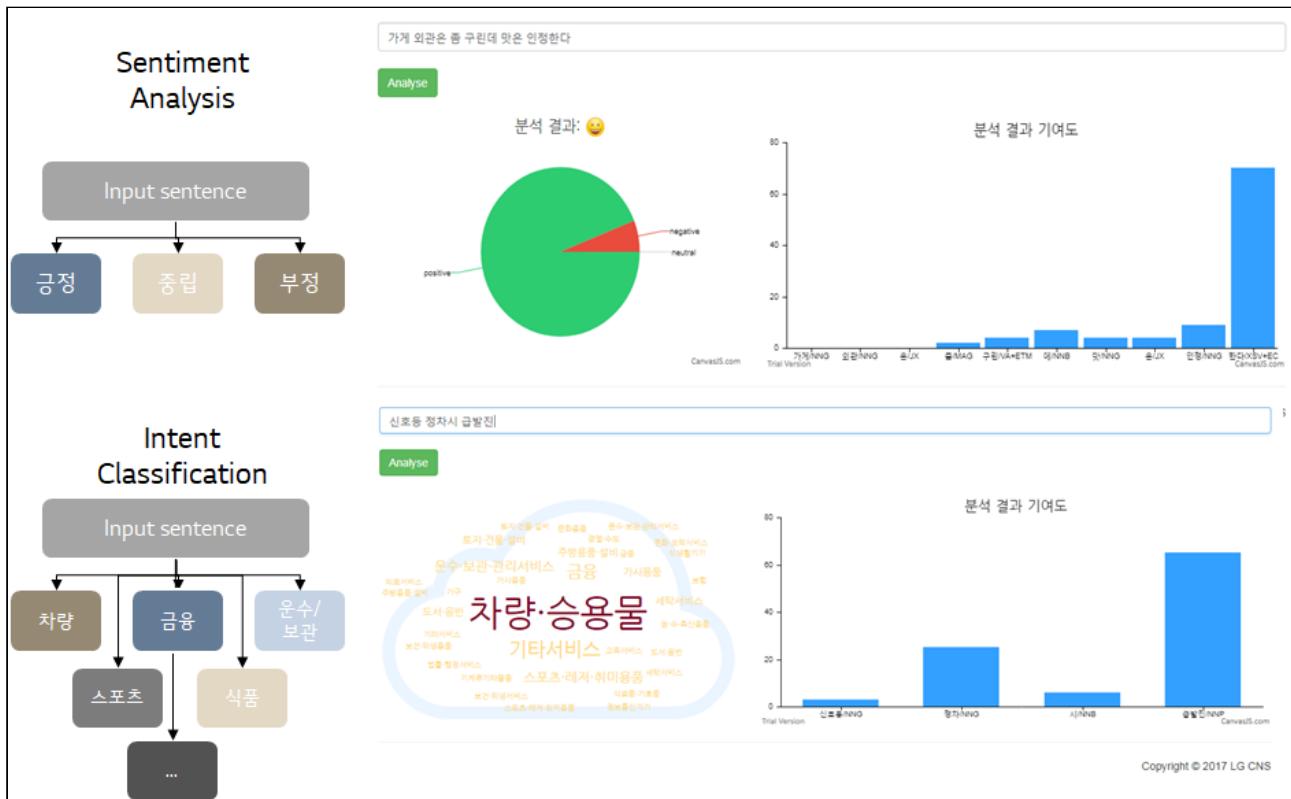
대표적인 자연어이해 태스크 몇 가지를 소개해드리겠습니다.

4.3.1 문장/문서 분류(Sentence/Document Classification)

입력받은 텍스트를 지정된 K개의 클래스(또는 카테고리) 중 하나로 분류하는 과제입니다.

사용자 리뷰를 감성분석(긍/부정)하거나, 유저의 발화문을 챗봇이 처리할 수 있는 기능 중 하나로 매핑하는 의도분류 등에 쓰일 수 있습니다.

³¹ <https://word2vec.kr/search/?query=%EA%B8%B0%EA%B3%84-%EC%B2%A0%2B%ED%94%BC%EB%B6%80>



[그림 15. 감성분석과 의도분류]

4.3.2 Sequence-to-Sequence

문장/또는 문서를 입력으로 받아 문장을 출력하는 과제입니다.

한 나라의 언어를 다른 나라로 옮기는 번역과제라든지, 긴 문서를 핵심 문장으로 추리는 요약과제가 이에 해당합니다.

이외에도 자유 대화 등 다양한 과제에 활용 가능합니다.



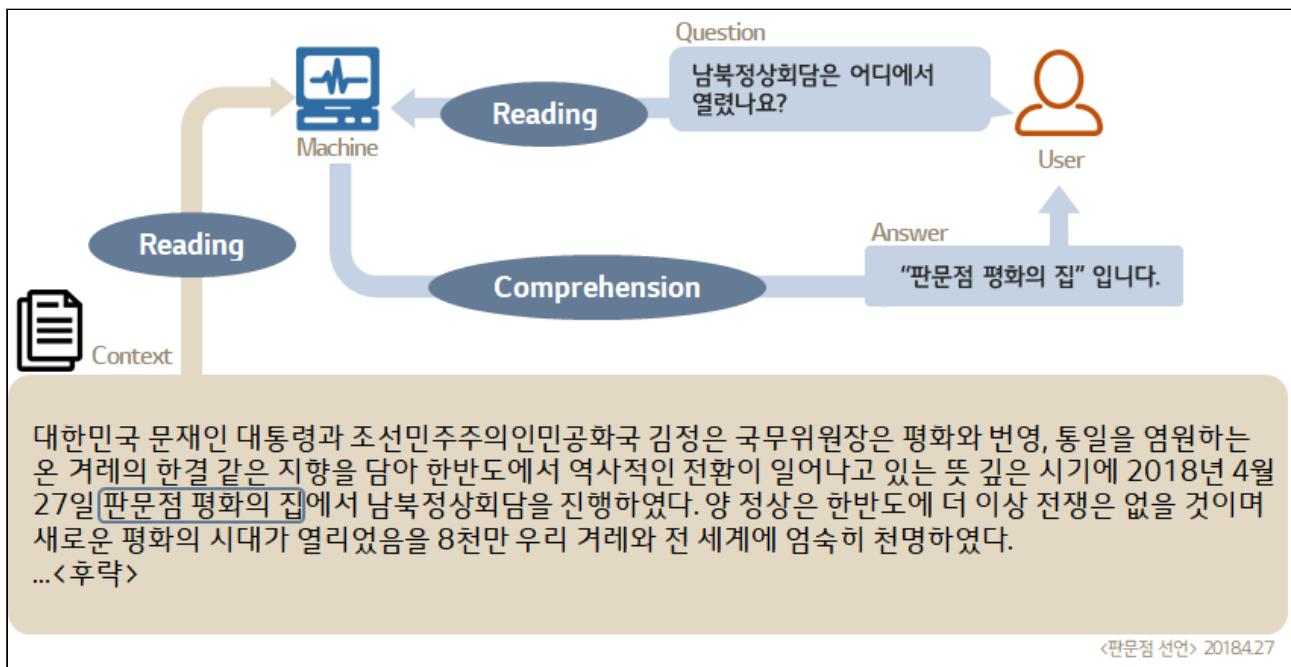
[그림 16. 기계번역과 뉴스요약]

4.3.3 질의 응답(Question Answering)

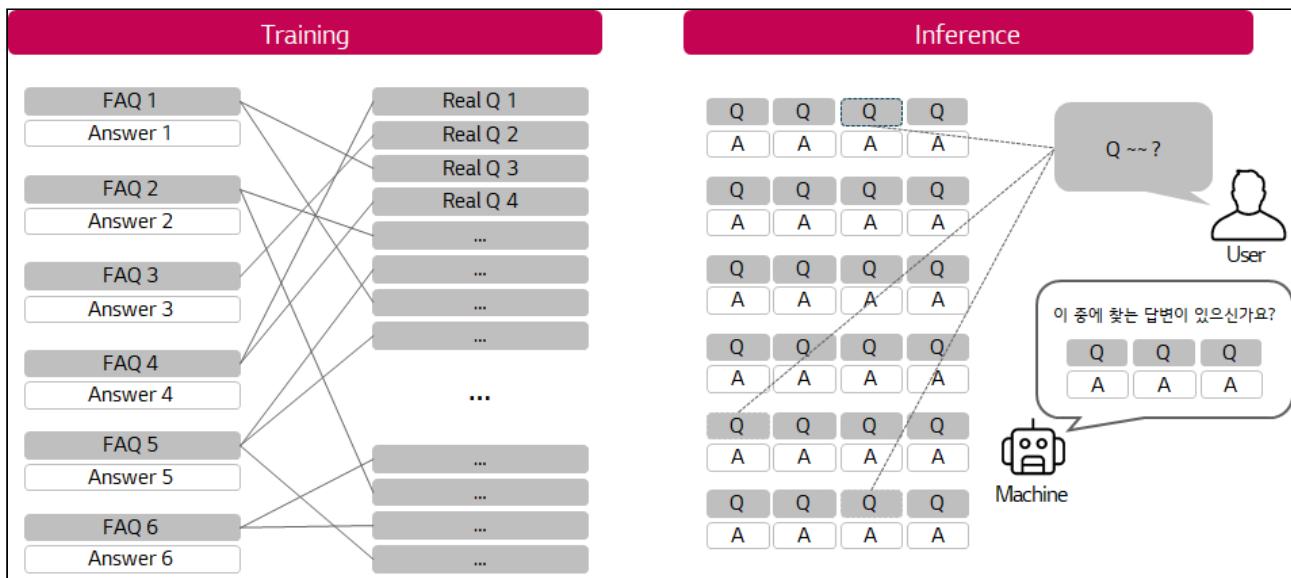
사용자 질문이 들어오면 내가 가진 매뉴얼 내에서 가장 답변이 될 가능성이 높은 영역을 리턴하는 MRC(Machine Reading Comprehension)와,

가장 유사한 과거 질문/답변(FAQ)를 꺼내주는 IR(Information Retrieval) 형태가 있습니다.

주로 상담챗봇 및 콜센터에 자주 활용되는 기술입니다.



[그림 17. MRC 질의응답 예]



[그림 18. IR 질의응답 예]

4.4 마무리

어떻게 기계가 우리가 의사소통하는 언어를 수치화하고 학습하는지 다뤄봤습니다.

의미를 어느정도 이해하는 수준의 인공지능이 앞으로 어떻게 우리 삶을 편리하게 해줄지에 대한 방식은 무궁무진합니다.

물론 사람이 언어를 이용하며 만드는 미묘한 행간의 의미와 뉘앙스, 문체에서 느껴지는 아름다움이나 긴장감, 이런 것들은 아직 기계가 이해하기에 먼 길입니다.

하지만 언제나 길은 있습니다. 인간도 기계도 언제나 할 수 있는 것을 할 뿐입니다.

기계가 마법처럼 사람 말을 알아듣는 게 아닌, 도식화를 통해서 학습하는 것처럼요

언젠가는 찰떡같이 알아듣는 인공지능이 나오길 바랍니다.

이번 시간은 자연어의 개념과 인공지능이 이를 처리하는 방법에 대해 설명드렸습니다.

그리고 대표적인 NLP/NLU 과제 종류를 살펴보았습니다.

다음 시간은 '**인공지능이 순차적인 데이터(시계열 데이터)를 학습하는 방법**'에 대해 알아보겠습니다.

감사합니다 😊

참고자료

- ‘잔머리의 대가’ 요즘 아이들, <http://mn.kbs.co.kr/news/view.do?ncd=4107322>
- [문명특급 EP.81] 펭수 모른다던 그 초딩들 만났습니다, SBS 뉴스, https://news.sbs.co.kr/news/endPage.do?news_id=N1005493660&plink=SEARCH&cooper=SBSNEWSSEARCH&plink=COPYPASTE&cooper=SBSNEWSEND
- 쌔기 문자, 위키피디아, https://ko.wikipedia.org/wiki/%EC%90%90%EA%B8%B0_%EB%AC%B8%EC%9E%90
- 밀레니얼 세대 다음은?, <https://brunch.co.kr/@moonkils/6>
- NLU, 인간의 언어를 이해하는 기계, TEC, <https://wire.lgcns.com/confluence/pages/viewpage.action?pageId=114306888>
- 아이언맨, 위키피디아, <https://ko.wikipedia.org/wiki/%EC%95%84%EC%9D%B4%EC%96%B8%EB%A7%A8>
- 교착어, 위키피디아, <https://ko.wikipedia.org/wiki/%EA%B5%90%EC%B0%A9%EC%96%B4>
- 자연어 감성분류를 위한 딥러닝, TEC, <https://wire.lgcns.com/confluence/pages/viewpage.action?pageId=112689711>
- 한국어와 NLTK, Gensim의 만남, <https://www.lucypark.kr/docs/2015-pyconkr/#20>
- 포켓몬고 Word2Vec CBOW-300 실험일지, AI기술팀 김명지, 2017.
- Efficient Estimation of Word Representations in Vector Space, Tomas Mikolov, et al., 2013.
- Korean Word2Vec, <https://word2vec.kr/search/?query=%EA%B8%B0%EA%B3%84-%EC%B2%A0%2B%ED%94%BC%EB%B6%80>

5 [5편] 과거의 경험을 통해 현재를 배우는 인공지능

안녕하세요, CTO AI빅데이터연구소입니다.

한 달에 두 번씩 **AI 테크레터**를 통해 인공지능 지식을 임직원 여러분들께 공유드리고 있습니다.

모든 CNSer가 이해하실 수 있도록 쉽게 작성하려고 하니, 상세 기술에 대한 궁금증이 생기시면 댓글이나 이메일을 통해 언제든 연락 바랍니다 😊

본 업로드는 [TECH wiki AI게시판](#)(see page 7)에서 연재됩니다.

작성 : CTO AI빅데이터연구소 AI기술팀 [김명지 팀장/총괄 CONSULTANT](#)/[언어AI LAB](#)³²

- [시간 흐름에 따른 데이터\(Sequential data\) 처리하기](#)(see page 80)
- [Recurrent Neural Network\(RNN; 순환 신경망\)](#)(see page 82)
 - [장점](#)(see page 83)
 - RNN은 시간 흐름에 따른 과거 정보를 누적할 수 있다(see page 83)
 - RNN은 가변 길이의 데이터를 처리할 수 있다(see page 83)
 - RNN은 다양한 구성의 모델을 만들 수 있다(see page 84)
 - [단점](#)(see page 85)
 - 연산 속도가 느리다(see page 85)
 - 학습이 불안정하다(see page 85)
 - 실질적으로 과거 정보를 잘 활용할 수 있는 모델이 아니다(see page 86)
 - [성능 보완](#)(see page 86)
 - LSTM(Long-short term memory)(see page 86)
- [활용 사례](#)(see page 88)
- [마무리](#)(see page 89)

지난 시간에는 자연어의 개념과 인공지능이 이를 처리하는 방법에 대해 다루었습니다.

NLU 관련 기술로 인해 이제는 기계가 사람의 말을 알아듣는 범위가 조금씩 넓어지고 있습니다.

오늘은 시간의 흐름에 따른 데이터 처리 방식, 인공지능이 '시계열 데이터'를 처리하는 방식에 대해 알아보겠습니다.

지난 시간까지의 내용이 궁금하신 분은 [★AI Tech Letter](#)(see page 7)[★](#)를 확인하시기 바랍니다.

³²<https://wire.lgcns.com/confluence/display/~78628>

5.1 시간 흐름에 따른 데이터(Sequential data) 처리하기

딥러닝 알고리즘의 강점은 비정형 데이터를 규칙 없이도 잘 처리할 수 있다는 점입니다.

지난 시간까지 이미지라든가 텍스트와 같은 비정형 데이터를 인공신경망이 처리하는 방식에 대해 소개시켜드렸습니다.

하지만 아마 그동안 대다수가 접할 기회가 많았던 데이터는 주로 정형 데이터였을거라고 생각합니다.

즉 관계형 데이터베이스 자료나 엑셀, CSV파일 등으로 나타낼 수 있는 정리된 데이터 타입이죠.

정형 데이터를 분석하는 대표적인 과제로 시계열 예측분석이 있습니다.

순차적인 시계열 데이터를 활용하여 근미래에 어떤 데이터 값이 나타날지를 예측하는 과제입니다.

여기에는 시간 흐름에 따라 변화하는 로그성 데이터를 활용합니다.

예를 들어, 직장인들의 꿈(?)인 주가 예측이라든지, 내일의 기상정보 예측과 같은 일입니다.

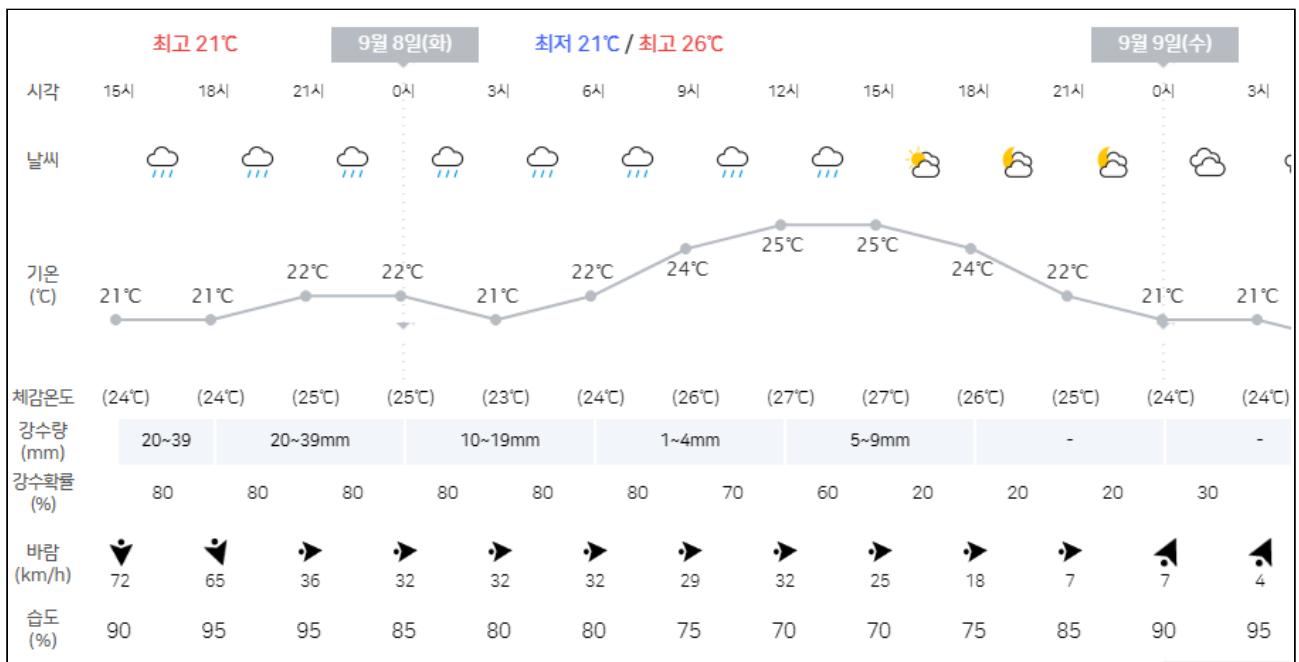


[그림 1. 시간 흐름에 따라 변화하는 시계열 데이터(Sequential data), 자료:[네이버금융](#)^{33]}]

우리가 내일의 코스피 지수나 기온을 예측하려 할 땐 한 시점의 입력 데이터만 고려하지는 않을 겁니다.

예를 들어 시간대별로 기온을 예측해야 하는 기상청을 예로 들어보겠습니다.

³³ https://finance.naver.com/sise/sise_index.nhn?code=KOSPI

[그림 2. 시간대별 일기예보, 자료:기상청³⁴]

기상청에서 현 시각의 기온을 예측해야 한다고 했을 때, 기온 예측에 중요한 변수로 이전 시각의 기온, 바람의 풍속, 습도 데이터가 중요하다고 가정하겠습니다.
일기예보관은 세 시간마다 이전 시각까지 확보한 데이터를 가지고 현 시각의 기온이 어떻게 될지 예측을 합니다.
(※ 위 상황은 설명을 돋기 위한 예시로, 실제 기상청의 예측 프로세스와 다를 수 있습니다)

시각	9시	12시	15시	18시	t
이전시각 기온	23	25	26	24	x_t
이전시각 풍속	29	32	25	18	
이전시각 습도	80	90	70	70	
현시각 기온	25	26	24	?	y_t

[그림 3. 세 시간마다의 기온 예측]

예를 들어 9시엔 이전 시각의 기온 23도, 풍속 29 km/h, 습도 80%라는 값을 활용하여 9시의 기온이 25가 될 것이라고 예측합니다.

12시에는 9시의 기온(25도), 풍속, 습도를 가지고 26도가 될 것이라고 예측하고, 15시에도 마찬가지 작업을 합니다.

³⁴ <https://www.weather.go.kr/w/weather/today.do>

매 시각마다 '기온'이라는, 내가 예측하고자 하는 Y 값을 맞추기 위해 이전의 기온/풍속/습도라는 입력값 X 를 활용하는 것 이죠.

지금까지는 우리가 배운 인공지능 학습 방식과 동일합니다.

입력값 X 를 가지고 Y 를 예측한다, 즉 이미지를 넣으면 강아지/고양이를 구별한다든가, 텍스트 문장을 넣어서 긍정/부정을 분류한다든가 하는 문제와 같습니다.

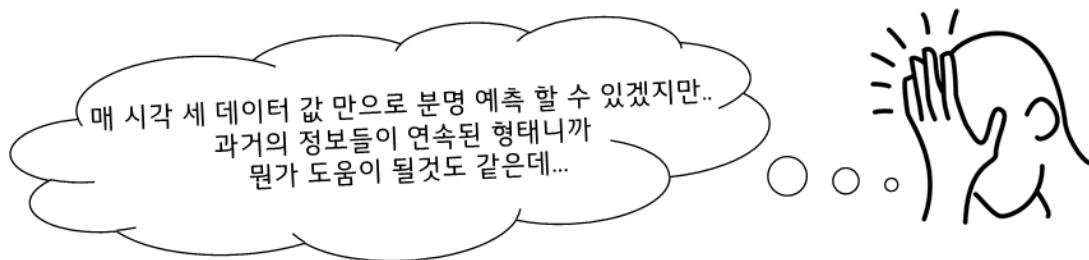
하지만 여기서 한 생각이 머릿속에 떠오릅니다.

일기예보관이 매 시간 같은 일을 반복하는데, 매시간 일을 반복하면서 쌓인 예측 노하우가 있을 겁니다.

그리고 밤낮이라는 시간의 흐름, 계절이라는 흐름에 따른 패턴이 분명히 존재할텐데

이렇게 그 때 그 때의 입력값 3개만으로 기온을 예측하는 것은 활용할 수 있는 정보를 다 쓰지 못하는 느낌 아닌가요?

한 시각의 데이터 세 개 만으로 분명 예측을 할수도 있겠지만 과거의 정보들이 연속된 형태니까 뭔가 도움이 될것도 같은 생각이 듭니다.

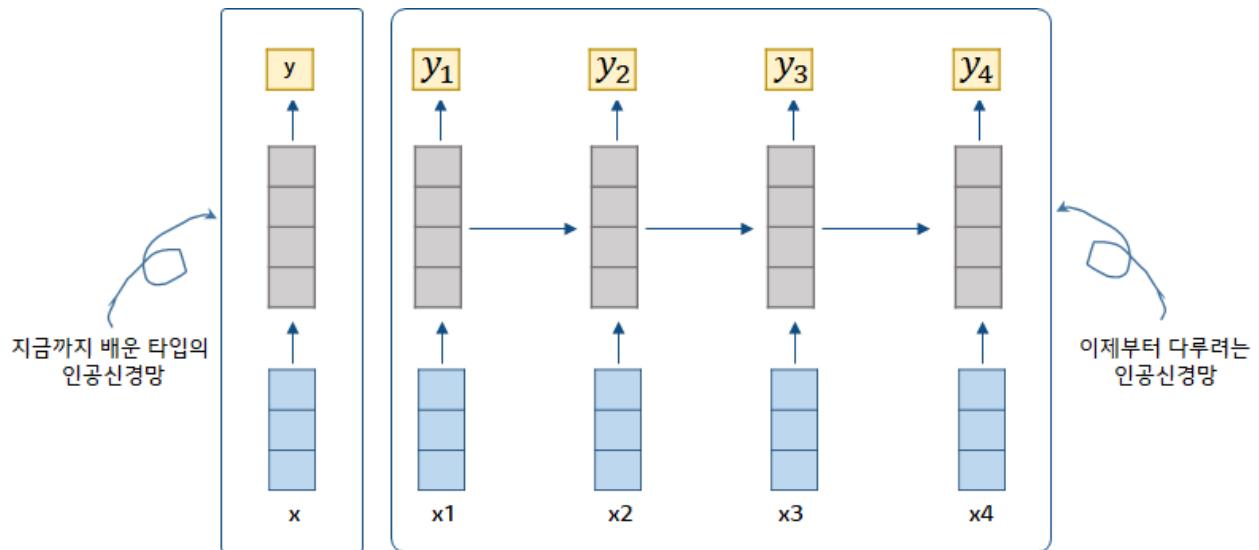


[그림 4. 과거의 노하우를 활용할 방법을 생각해봅시다]

5.2 Recurrent Neural Network(RNN; 순환 신경망)

이런 형태의 데이터를 잘 다루기 위한 인공신경망 종류가 있습니다.

'순환신경망'이라고 하는 RNN(Recurrent Neural Network)인데요, 아래 그림과 같이 생겼습니다.



[그림 5. 원·입력값으로 출력값을 예측하는 기존의 인공신경망, 오:과거의 처리 이력을 압축하여 반영하는 RNN]

기존의 신경망이 벡터, 또는 매트릭스로 변환된 입력 데이터를 가지고 출력을 한다면
RNN 역시 마찬가지 작업을 하지만 하나 다른 점이 있습니다.

[그림 5]에서 볼 수 있듯이, 과거에 데이터를 처리하여 결과를 출력했던 과정의 일부를 가져와 현 시점에서 데이터를 처리하고 결과를 출력하는 데 도움을 주는데요,

그림상에선 오른쪽으로 향하는 가로 방향의 화살표에 해당합니다. RNN에서는 벡터 형태로 정보(feature)가 넘어가죠.
그래서 2번째 시점에는 해당 시각의 입력 데이터 뿐 아니라 1번째 시점의 누적된 정보를 가지고 예측을 출력합니다.
3번째 시점에는 1,2번째 시점의 누적된 정보가 반영될 것이고, 4번째 시점이 되면 1,2,3번째의 압축된 정보가 도움을 줍니다.

입력 데이터만으로 예측하는 것보다 과거의 누적된 정보(feature)가 있다면 더 개선된 예측을 할 수 있겠죠?

5.2.1 장점

RNN의 특징에 대해 살펴보았는데요, RNN이 가진 장단점에 대해 정리해보겠습니다.

5.2.1.1 RNN은 시간 흐름에 따른 과거 정보를 누적할 수 있다

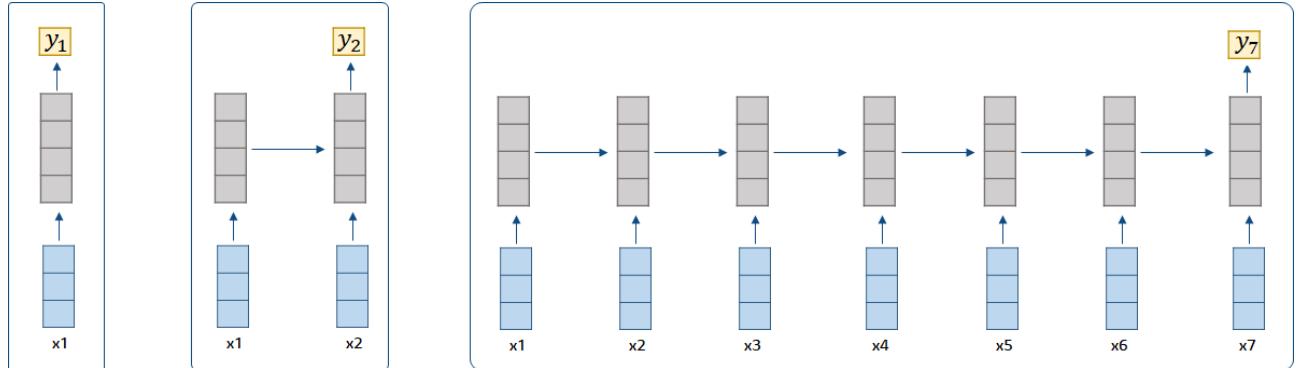
위에서 언급한대로, 다른 인공신경망과 달리 RNN이라는 특수한 형태의 신경망은
입력 데이터뿐 아니라 과거의 처리 내역을 반영하여 더 나은 결정을 할 수 있다는 점이 가장 큰 장점입니다.

5.2.1.2 RNN은 가변 길이의 데이터를 처리할 수 있다

위 일기예보 예시의 경우 3시각 단위로 예측을 하는데, 이러한 단위 시간마다의 매 시점을 timestep이라고 합니다.
timestep은 시간 단위가 될 수도 있고, 일 단위, 월 단위, 초 단위 등... 순서만 존재한다면 각자 구성하기 나름입니다.
RNN은 과거의 정보를 매 timestep마다 압축하여 다음 timestep으로 넘기므로 데이터의 길이에 무관하게 자유롭게 구성
할 수 있습니다.

하루치 데이터로 예측을 할 수도 있고, 일주일치 데이터를 모아서 예측할 수도 있고, 한 달치 데이터를 모아서 예측할 수도 있습니다.

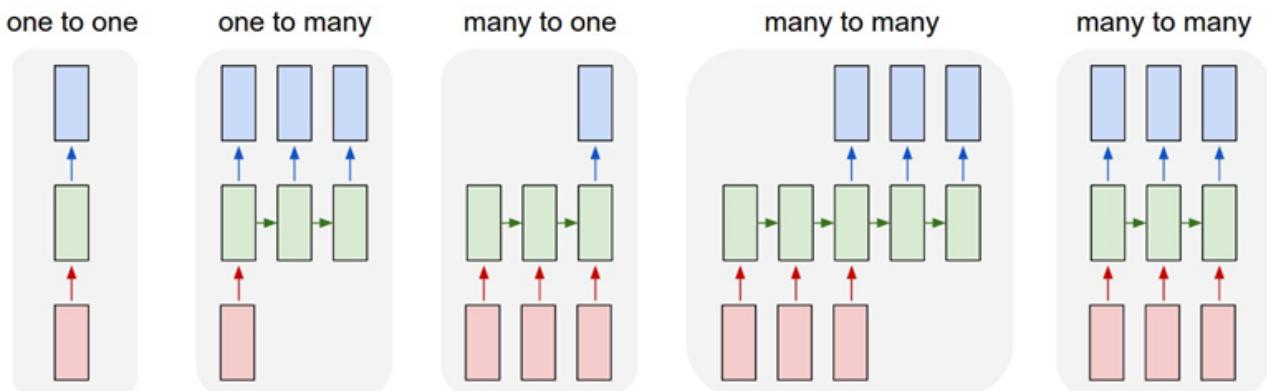
데이터가 아무리 긴 시간이나 짧은 시간에 대해 구성되어있다고 해도 같은 내용을 예측한다면 모델을 별도로 구성할 필요가 없습니다.



[그림 6. 위 세 가지 경우의 데이터 길이 모두 동일한 하나의 RNN 모델로 처리할 수 있다]

5.2.1.3 RNN은 다양한 구성의 모델을 만들 수 있다

유연한 구조를 가진 RNN은 다양한 구조를 활용하여 신경망을 구성할 수 있습니다.



[그림 7. 다양한 구조의 RNN]

- [그림 7]의 첫 번째 경우처럼 입력 데이터만으로 출력을 만들 수 있습니다. 이 경우는 이제까지의 신경망과 동일한 형태입니다.
- [그림 7]의 두 번째 경우처럼 한 번의 입력으로 여러 timestep의 데이터를 예측할 수 있습니다. 예를 들어 9시의 데이터로 12시, 15시, 18시의 기온을 예측하는 과제가 되겠네요
- 반대로 입력을 여러 timestep 받아 정보를 누적하다가 한번에 예측할 수도 있습니다. 일주일치의 데이터를 모아 8일차의 데이터를 예측한다던가 하는 등입니다.
- 입력 정보를 여러 timestep 누적하고 출력도 여러 timestep 진행할 수 있습니다. 월, 화, 수의 데이터로 목, 금, 토의 결과를 예측하는 경우입니다.
- 또는 마지막 경우처럼 그 때 그 때 입력과 출력을 처리할 수도 있습니다.

상황에 따라 RNN은 다양하게 입력 데이터를 처리, 누적하고 결과를 예측할 수 있습니다.

이 때 입력 데이터의 정보를 누적하는 부분을 **인코딩(Encoding)**, 결과를 출력하는 부분을 **디코딩(Decoding)**이라고 표현합니다.

5.2.2 단점

5.2.2.1 연산 속도가 느리다

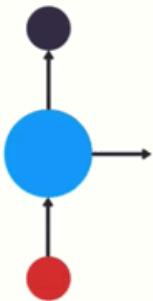
RNN은 과거의 처리 내역을 현재에 반영해야 하기 때문에, 현 시점의 데이터를 처리하려면 반드시 이전 시점의 데이터가 처리 완료되어야 합니다.

따라서 병렬학습이 어렵고, 순차적으로 데이터를 처리해야 하는 성질로 인해 연산 속도가 다소 느린 편입니다.

많은 경우 딥러닝 모델을 이용하여 데이터를 학습시킬 때 GPU 서버를 활용하곤 하는데요,

그래픽(주로 2차원 매트릭스나 3차원 이상의 텐서)을 다루는 데 특화된 GPU 칩은 병렬 연산에 광장한 이점을 가진 장비입니다.

하지만 RNN을 처리하는 경우 이러한 병렬처리의 이점을 잘 활용할 수 없는 한계가 있습니다.



[그림 8. RNN은 앞 연산이 완료된 뒤 뒷 연산이 순차적으로 진행되어야 한다.]

하지만 정형 데이터(수치, 범주형 등)를 활용하는 경우 속도 저하를 크게 체감하지 못하는 경우가 대부분입니다.

연산 속도 저하는 텍스트 데이터를 다루는 경우에 주로 문제가 됩니다.

5.2.2.2 학습이 불안정하다

또한 단순 RNN은 학습시키기 매우 어려운 딥러닝 알고리즘 중 하나입니다.

미분 수식 전개가 필요하여 자세히 설명드리지는 않겠습니다만, 다루는 데이터의 timestep이 길면 길수록 문제가 발생 할 확률이 높습니다.

timestep이 길어지면 RNN 인공신경망이 반영해야 할 과거의 이력이 많아지게됩니다.

이 과정에서 인공신경망이 학습해야 할 값이 폭발적으로 증가하는 현상이 발생할 수 있습니다. 이를 **Gradient Exploding**이라 합니다.

또 이와는 반대로, timestep이 길어지면 저 멀리 있는 과거의 이력은 현재의 추론에 거의 영향을 미치지 못하는 문제도 생깁니다.

이를 **Gradient Vanishing**이라고 합니다.

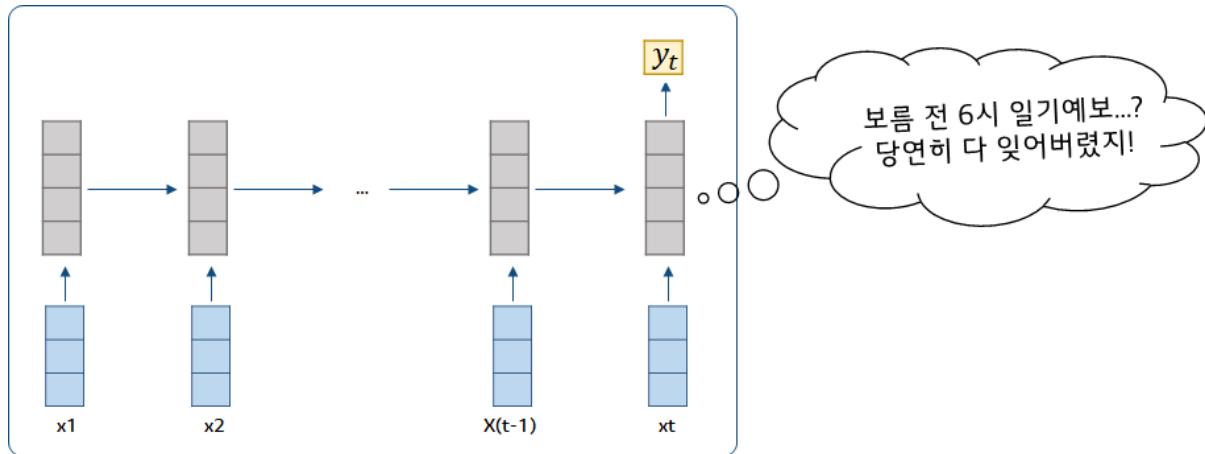
RNN은 상당히 학습이 불안정하여, 이 두 가지 문제가 자주 발생하곤 합니다.

5.2.2.3 실질적으로 과거 정보를 잘 활용할 수 있는 모델이 아니다

이론적으로 RNN은 과거의 정보가 누적되며 현재 추론에 도움을 주곤 합니다만, 실질적으로 먼 과거의 정보를 반영하기 힘듭니다.

RNN은 한 timestep씩 정보를 누적하여 인코딩하는데, 먼 과거의 정보는 여러 번 압축되고 누적되다보니 거의 영향을 미치지 못합니다.

이를 RNN의 **장기 종속성/의존성 문제(Long-term dependency)**라고 합니다.



[그림 9. RNN의 장기 종속성 문제. 먼 과거는 오늘날 영향을 거의 미치지 않는다.]

이는 사람도 마찬가지입니다. 오늘 점심 메뉴는 뭐였는지 기억하지만, 어제 점심이 뭐였는지 바로 기억하시는 분 있나요? 일주일 전은요?

최근의 정보일수록 잘 기억하고 반영하며, 먼 과거일수록 잊어버리는 경향이 인공신경망에서도 동일하게 나타납니다.

5.2.3 성능 보완

RNN에 이러한 단점이 있다 보니, 당연히 이를 해결하는 방안도 등장하였습니다.

5.2.3.1 LSTM(Long-short term memory)

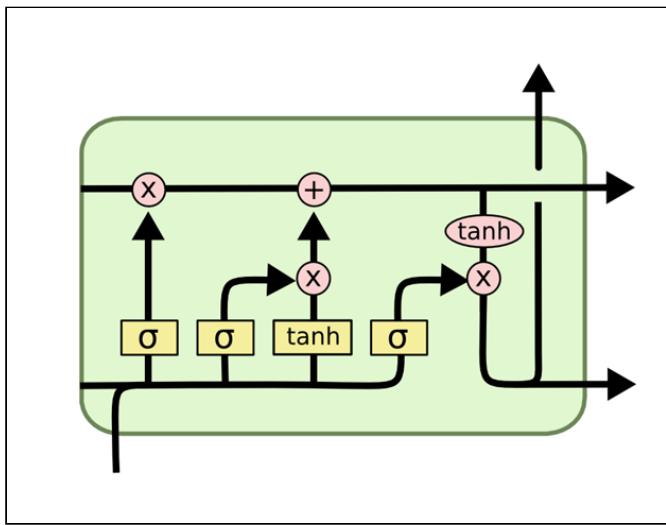
저 먼 과거의 정보 중 중요한 것은 기억하고, 불필요한 것은 잊어버리도록 스스로 조절 가능한 RNN 유닛이 있습니다.

매번 동일하게 과거의 정보를 누적하고 압축하는 기본 RNN(naive RNN) 유닛과는 달리,

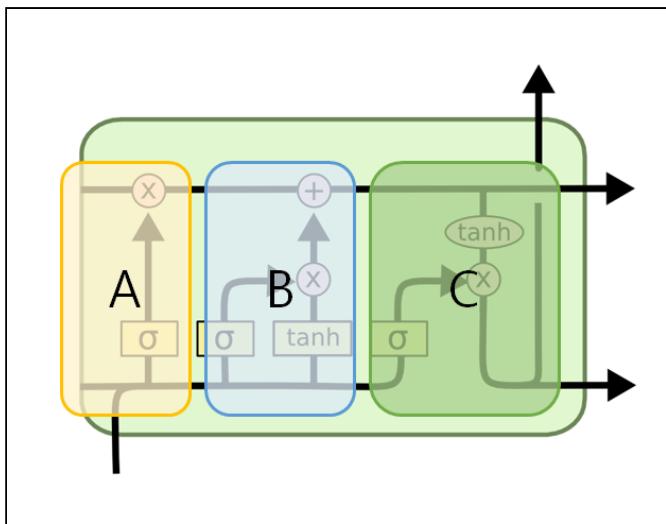
정보 흐름을 잘 조절하기 위해 성능을 개선한 특별한 형태의 뉴런이라고 생각하시면 됩니다.

대표적으로 **LSTM**(Long-short term memory)이라는 유닛이 있습니다.

장/단기 메모리 유닛이라는 뜻인데요, 이 유닛은 아래와 같이 생겼습니다.

[그림 10. LSTM unit의 구조, 자료³⁵]

무섭게 생겼지만, 이 복잡해 보이는 연산은 어차피 컴퓨터가 수행하니 전혀 걱정하실 것 없습니다.
우리는 구조를 조금 단순하게 둑어서 3 부분의 Gate가 있다는 것만 알아둡시다.



[그림 11. 단순화한 LSTM 구조, A/B/C는 각각 forget/input/output gate에 해당]

Gate라 하는 부분은 정보의 흐름을 조절하는 관문 역할을 수행합니다.

[그림 11]의 A 부분은 **forget gate**라 불리며, 말 그대로 잊어버림에 대한 조절을 합니다.

과거의 정보중 불필요하다고 생각하는 부분은 통과시키지 않고 "이건 안중요하니까 잊어버려도돼"라고 결정합니다.

B 부분은 **input gate**로, 현재의 정보(*input data*)를 얼마나 반영할지를 결정합니다.

³⁵ <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

"오늘 이런 정보가 있어? 이건 중요하니까 반영하자!" 또는, "오늘엔 별로 안중요해보이는데? 과거 정보로 충분해! 거르자!" 이런 결정을 수행합니다.

C 부분은 **output gate**로, 현재 시점에 연산된 최종 정보를 다음 시점에 얼마나 넘길지를 결정합니다.

"그래 이건 내일도 쓸 수 있겠다" 라든가, "오늘은 써먹었지만 내일은 불필요할듯" 같은 역할을 합니다.

LSTM에는 이 세 가지의 gate가 있어서, 정보의 흐름을 인공지능이 자체적으로 더 원활하게 조절하는 기능을 합니다.

따라서 데이터가 길어진다고 해도 일반 RNN에 비해 더 좋은 예측을 할 수 있게 됩니다.

다만 계산 과정이 복잡한 만큼 연산속도는 조금 더 느려지는 단점이 있습니다.

그래서 이를 조금 개선한 GRU(Gated Recurrent Unit)도 있습니다만, LSTM과 거의 유사한 기능을 갖고있어 소개하지는 않겠습니다.

오늘날 인공신경망을 활용하는 대부분의 시계열 예측은 아주 간단한 과제를 제외하고는 기본 RNN을 사용하는 경우는 거의 없습니다.

대부분은 LSTM, GRU과 같은 개선된 유닛을 활용하며, 대부분의 딥러닝 프레임워크에서 쉽게 구현할 수 있도록 기능을 제공하고 있습니다.

5.3 활용 사례

만일 시계열 데이터가 익숙하지 않은 분이시라면 기존의 과제와 어떻게 다른지 혼란스러울 수 있습니다.

Sequential Data라는 말 그대로 알 수 있듯이 뭔가 순차적인 흐름을 가지고 진행되는 데이터가 있다면 전부 시계열 데이터라고 부를 수 있습니다.

우리가 오늘 다룬 예제의 경우를 살펴보면, 물론 풍속이나 습도 만으로도 현 기온을 예측할 수 있겠지요.

하지만 기온이라는 것은 매 시점 독립적으로 정해지는 것이 아닌 시간에 따른 패턴을 갖습니다.

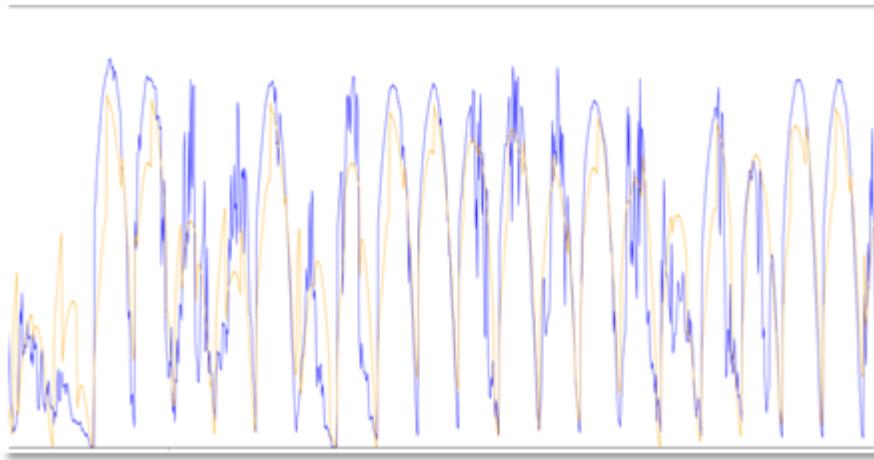
아침에 추웠다면 점심에도 약간 쌀쌀한 경향이 반영될 것이고, 대체로 아침/저녁에 기온이 내려가는 반면 낮에는 약간 올라가곤 하죠.

인공신경망을 활용한다면 이러한 패턴을 규칙화하지 않아도 자동으로 학습할 수 있습니다.

자사에서도 2018년, LSTM 모델을 활용해서 태양광 에너지 발전량을 예측한 적이 있습니다.

이 경우 10분 간격 timestep에 따라 경북 오태지역의 일시, 일조량, 풍속, 온도를 가지고 에너지 발전량을 예측하였습니다.

기존 수리모델 대비 LSTM이라는 딥러닝 모델이 성능을 개선하였다고 하네요.



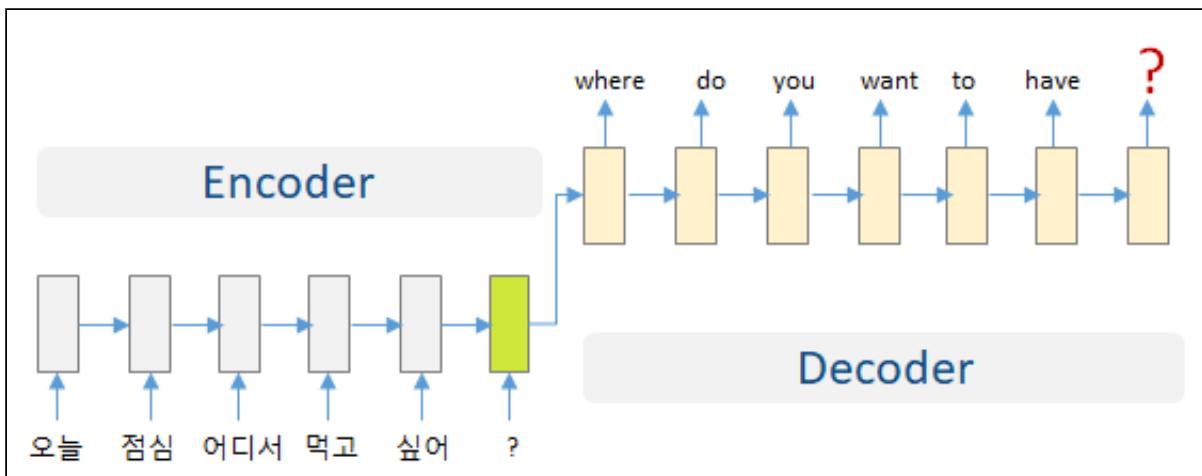
[그림 12. 태양광 에너지 발전량 예측 (2018), LG CNS]

텍스트 문장도 일종의 시계열 데이터로 볼 수 있습니다.

문장이란 단어가 일정한 순서대로 등장하는 데이터입니다.

한 단어 한 단어를 벡터로 바꿀 수 있으니([\[4편\] 참고\(see page 60\)](#)), RNN의 한 timestep에 단어 벡터를 하나씩 입력시키면 됩니다.

RNN을 텍스트 데이터에 적용한다면 번역과 같은 과제에 활용할 수 있습니다.



[그림 13. 한국어 문장을 인코딩하여 영어 문장을 디코딩하는 RNN (sequence-to-sequence)]

5.4 마무리

우리는 오래전부터 과거의 경험을 통하여 미래를 예견하고자 하였습니다.

사람은 한 번 수행했던 업무는 노하우를 얻어 다음에 더 빠르고 잘 수행할 수 있습니다.

과거에 실패를 경험했다면 같은 실패를 또 경험할 가능성도 적지요.

최근의 기억은 더 뚜렷하게 남고, 먼 과거는 잊어버리기 마련입니다.

인간의 신경계를 본딴 인공신경망에도 비슷한 기능을 수행하는 알고리즘이 있습니다.

전통적인 머신러닝 기법에서는 ARIMA 기법 등을 통하여 데이터의 정보 흐름을 파악하고, 주기적으로 반복되는 패턴을 반영하여 분석합니다.

인공신경망에서는 RNN이라고 불리우는 특별한 형태의 신경망이 그 역할을 수행합니다.



[그림 14. AI 미래 예측은 마법이 아닙니다. '가장 그럴싸한 것'에 대한 수리모델적 추론입니다.]

미래를 예측한다는 것은 인공지능이 미래를 좌지우지한다거나, 예언한다는 개념이 아닙니다.

과거의 패턴을 통해 근미래에 벌어질 가능성성이 가장 높은 데이터를 유추하는 것이지요.

이번 시간은 인공지능이 '시계열 데이터'를 처리하는 RNN 방식에 대해 설명드렸습니다.

그리고 이 방식이 지닌 장단점과, 이를 보완하기 위한 개선방법도 살펴보았습니다.

다음 시간은 '**오버피팅(Overfitting)**과 **정규화(Regularization)**'에 대해 알아보겠습니다.

감사합니다 😊

참고자료

- 네이버 금융, 코스피, https://finance.naver.com/sise/sise_index.nhn?code=KOSPI
- 기상청, 오늘의 날씨, <https://www.weather.go.kr/w/weather/today.do>
- CS231n-lecture10, Fei-fei Li&Andrej Karpathy&Justin Johnson, 2016, http://cs231n.stanford.edu/slides/2016/winter1516_lecture10.pdf
- 자사 딥러닝 실무과정 교재보기, [교재보기] 딥러닝 실무³⁶
- Illustrated Guide to Recurrent Neural Networks, MC.AI, <https://mc.ai/illustrated-guide-to-recurrent-neural-networks/>
- Understanding LSTM Networks, colah's blog, <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- 전사기술세미나 20차, 창작하는 인공지능, 김명지, 2020, 20차 "인공지능, 스스로의 학습을 넘어 크리에이터가 되다!!" - 2020.07.30³⁷

³⁶ <https://wire.lgcns.com/confluence/pages/viewpage.action?pageId=73005264>

³⁷ <https://wire.lgcns.com/confluence/pages/viewpage.action?pageId=114632185>

6 [6편] 헛똑똑이 인공지능 제대로 가르치기

안녕하세요, CTO AI빅데이터연구소입니다.

한 달에 두 번씩 **AI 테크레터**를 통해 인공지능 지식을 임직원 여러분들께 공유드리고 있습니다.

모든 CNSer가 이해하실 수 있도록 쉽게 작성하려고 하니, 상세 기술에 대한 궁금증이 생기시면 댓글이나 이메일을 통해 언제든 연락 바랍니다 😊

본 업로드는 [TECH wiki AI게시판](#)(see page 7)에서 연재됩니다.

작성 : CTO AI빅데이터연구소 AI기술팀 [김명지 팀장/총괄 CONSULTANT](#)/언어AI LAB³⁸

- [AI Process](#)(see page 92)
 - [Offline Process](#)(see page 92)
 - [Online Process](#)(see page 94)
- [오버피팅\(Overfitting\)과 일반화 성능\(Generalization\)](#)(see page 95)
 - [Training, Validation, Test](#)(see page 97)
 - [Training set](#)(see page 97)
 - [Validation set](#)(see page 98)
 - [Test set](#)(see page 98)
 - [학습 곡선\(Learning curve\) 확인하기](#)(see page 98)
- [Regularization](#)(see page 100)
 - [데이터 증강\(Data Augmentation\)](#)(see page 101)
 - [Capacity 줄이기](#)(see page 101)
 - [조기 종료\(Early stopping\)](#)(see page 102)
 - [드롭아웃\(Dropout\)](#)(see page 102)
- [마무리](#)(see page 103)

지난 시간에는 인공지능이 '시계열 데이터'를 처리하는 RNN 방식에 대해 다루었습니다.

전통적인 분석 기법이 수행하던 영역에서도 딥러닝 알고리즘이 좋은 성능을 낼 수 있습니다.

오늘은 딥러닝 모델을 학습할 때 마주치기 쉬운 문제인 '오버피팅'에 대해 알아보고 이를 해결하는 방식에 대해 알아보겠습니다.

지난 시간까지의 내용이 궁금하신 분은 ★[AI Tech Letter](#)(see page 7)★를 확인하시기 바랍니다.

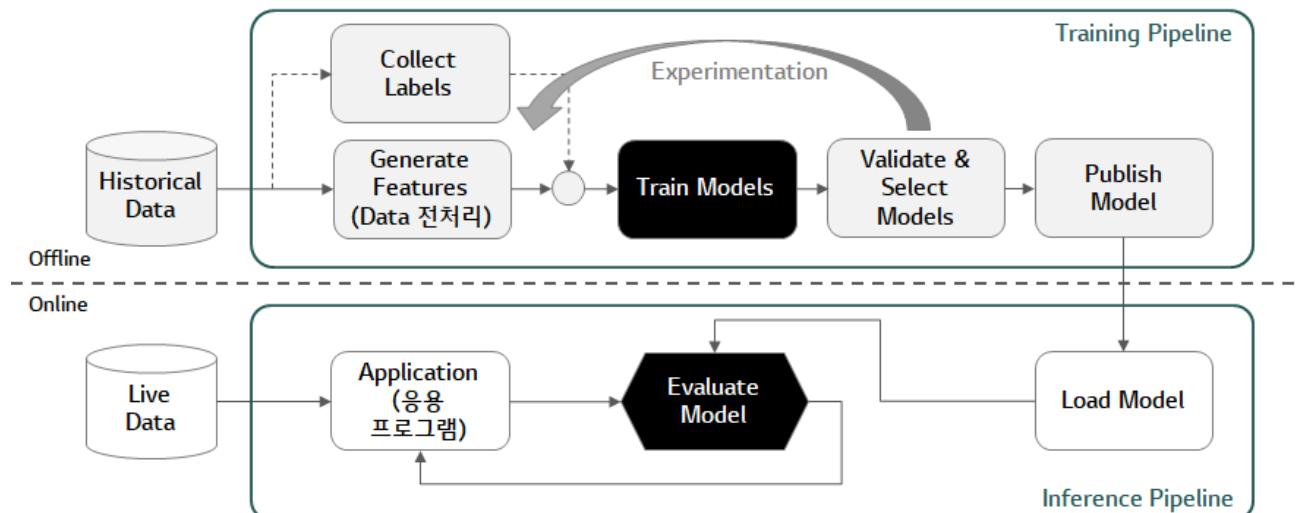
³⁸<https://wire.lgcns.com/confluence/display/~78628>

6.1 AI Process

AI를 활용해서 우리 주변을 더 나아지도록 할 방법을 찾고 계신가요? 그렇다면 아래와 같은 프로세스를 따라야 합니다.

여기서 말하는 AI는 딥러닝 모델 뿐 아니라, 빅데이터 분석과 같은 전통적인 머신러닝 기반의 모델 및 규칙 기반의 기계 자동화 모델이 포함된, 넓은 의미의 AI를 말합니다.

고객사의 요청으로 기존 시스템을 개선하기 위해 AI를 적용한다고 가정해보겠습니다.



[그림 1. AI pipeline, 참고:Netflix³⁹]

AI를 도입하는 과정은 크게 오프라인 프로세스와 온라인 프로세스로 나뉩니다.

[그림 1]의 윗부분과 아랫부분에 해당하는 과정이지요.

쉽게 말한다면 각 과정은 고객사 프로젝트에서 오픈을 위해 열심히 준비하는 개발 과정과, 고객사 오픈 이후의 운영 환경에 해당하는 과정이라고 볼 수 있습니다.

6.1.1 Offline Process

오프라인 프로세스는 과거에 만들어진 데이터(historical data)를 가공하는 것에서부터 시작합니다.

이는 그 때 그 때 발생하는 실시간 데이터가 아닌, DB 등에 이미 수집되어 있는 데이터입니다.

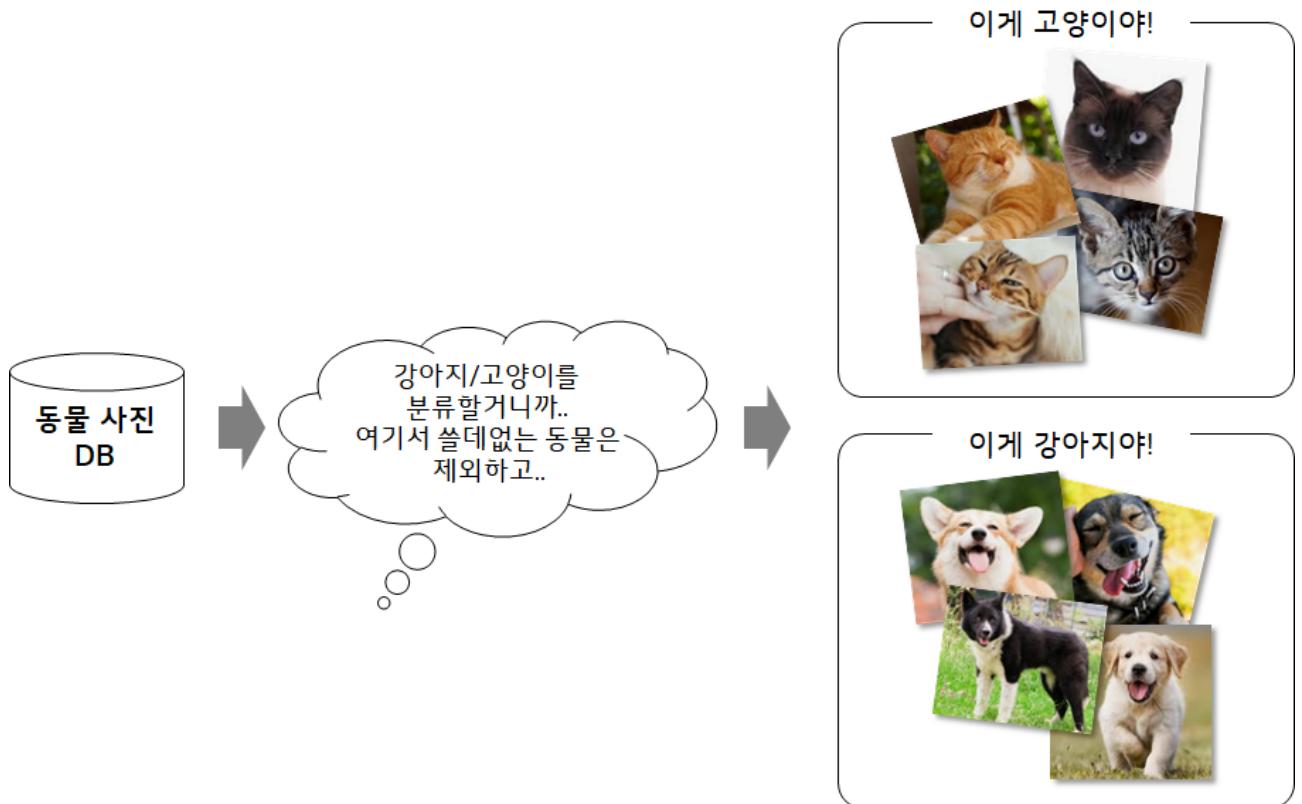
AI 모델러는 기 확보된 데이터를 확인하고 정제하여 필요한 부분을 취하거나(Generate features) 필요한 경우 라벨을 붙입니다(Collect labels).

이 때의 라벨이란 AI 모델 학습을 위해 필요한 정보로, 인공지능이 맞춰야 하는 '정답'이라고 생각하시면 됩니다.

예를 들어 "강아지와 고양이를 자동 분류하는 AI모델을 만들겠다!"라는 목표가 있다면 historical data는 강아지와 고양이의 이미지 파일이 되고,

라벨을 붙이는 작업은 각 사진이 강아지인지 고양이인지 태깅하는 작업이 되겠습니다.

³⁹ <https://www.slideshare.net/justinbasilico/making-netflix-machine-learning-algorithms-reliable>



[그림 2. 강아지/고양이 자동분류를 위해 historical Data를 전처리하고 라벨링하는 과정]

데이터를 마련했으면 어떤 머신러닝/딥러닝 알고리즘을 활용하여 모델을 학습할 것인지 정하고,

좋은 성능을 달성할 때까지(Validate & Select models) 반복실험을 진행합니다(Train models).

대부분의 머신러닝/딥러닝 모델 개발 경우 한 번의 시도만에 최적 모델이 완성되는 경우는 거의 없습니다.

보통은 실험의 결과를 진단하고 이 때 발견한 문제점을 해결하거나 더 개선된 성능을 낼 수 있도록 **실험(Experimentation)**을 반복하는데, 이 과정을 **튜닝**이라고 합니다.

튜닝은 모델 학습에 필요한 여러 수치 설정값(하이퍼파라미터)을 조절하는 등의 역할을 포함합니다.

만일 누군가 '실력 좋은 AI 모델 개발자'란 어떤 사람인가? 하고 물으신다면 저는 모델 튜닝을 더 폭넓고 빠르게 하는 사람이라고 대답하겠습니다.

이전 실험의 문제점을 찾아 다음 실험에 어떤 전략을 적용해야 모델의 성능을 개선할 수 있을지,

다양한 대안을 파악하고 노하우가 많은 사람일수록 적은 시행착오로 최적 모델을 찾을 수 있겠죠?

여러 실험을 반복하며 개선된 성능의 모델은 **배포할 수 있도록 최종 선택(Publish model)**됩니다.

여기까지의 과정은 더 나은 AI모델을 만들기 위해 모델을 학습시키는 과정으로, **Training Pipeline**이라고 합니다.

6.1.2 Online Process

최적의 성능을 내기 위해 모델을 개발하는 부분은 끝났습니다.

이제는 완성된 모델을 고객사 운영 환경에 올려야 합니다.

AI 모델은 오프라인 프로세스에서 배울 것을 모두 배웠으니, 실전 환경에 뛰어들어 추론(*Inference*)을 해야 합니다.



[그림 3. 모델 Training(위)과 Inference(아래)의 차이. Training이 훈련과정이라면 Inference는 실전 투입이다.]

모델 추론을 포함한 온라인 프로세스의 대부분은 머신러닝/딥러닝이라기보다는 **개발 영역(Application)**에 가깝습니다.

AI 모델을 운영 환경에 띄운다면(Load Model) 나머지는 GUI를 불인다든지, 고객사의 데이터베이스와 연결하고 다른 시스템과 연동한다든지 하는 일이 되겠지요.

이제부터 모델이 처리할 데이터는 기준에 정리하여 모아놓은 데이터가 아닙니다.

실시간으로 들어오는, 운영 환경의 **스트리밍 데이터(Live Data)**입니다.

AI 모델은 과거의 학습 지식을 바탕으로 현장의 데이터를 추론하게 됩니다.

머신러닝/딥러닝 관련 기법과 팁에 대한 대부분의 내용은 오프라인 프로세스의 모델 학습 과정에 적용할 수 있는 것들입니다.

AI 모델의 기본적인 성능을 끌어올리고, 주어진 데이터를 더 잘 맞출 수 있도록 최적화하는 데 필요한 지식들이죠. 하지만 우리는 주어진 데이터를 잘 맞추는 것이 목적이 아닙니다.

우리는 주어진 데이터로 잘 훈련된 AI 모델이 개발 환경 뿐 아니라 고객사 운영 환경과 같은 실제 현장에 나가서도 잘 작동되기를 바랍니다.

6.2 오버피팅(Overfitting)과 일반화 성능(Generalization)

회사에서 우스개 소리로 말하는 '시연(DEMO)의 법칙'을 아시나요?

분명 내가 어젯밤에 테스트해 볼 때는 잘 작동하던 것이, 중요한 보고나 고객 앞에서 시연할 때만 갑자기 기능이 동작하지 않는다거나, 느려지거나, 예외가 발생하거나 하는 일이 있지요.



[그림 4. 시연의 법칙. 이게 다 Generalization이 부족해서입니다.]

억울한 일이지만, 이와 같은 문제는 기계학습에서도 굉장히 자주 발생합니다.

분명히 학습시킬 때는 주어진 데이터를 잘 맞췄는데, 이상하게 고객사 운영환경에만 올라가면 추론 성능이 뚝뚝 떨어집니다.

지금부터 기계학습에서 가장 중요한 개념을 소개시켜드리려고 합니다.

Generalization이라고 부르는, 일반화 성능과 관련된 개념입니다.

Generalization의 정의는 '이전에 본 적 없는 데이터에 대해서도 잘 수행하는 능력입니다.

즉, 우리가 만든 AI 모델은 훈련시에는 본 적이 없는 새로운 입력 데이터(Live data)에 대해서도 잘 수행되어야 한다는 것입니다.

훈련시에만 잘 작동하고 일반화 성능이 떨어지는 모델을 **오버피팅(Overfitting)**되었다고 합니다.

예시를 통해 알아보겠습니다.

똑딱이와 펭수는 고3 수험생으로 올해 수능을 치른다. 남은 시간은 한달 남짓..

똑딱이는 이해력은 조금 떨리지만, 굉장한 노력파로 한달동안 최근 5년간의 수능 기출을 전부 암기해버렸다.

그래서 똑딱이는 문제만 들으면 정답이 몇 번 보기에 어떤 문장이었는지 정확하게 복원해냈다.

다만 문제의 의도가 뭔지, 왜 그 보기가 정답인지는 이해할 수 없었다. 단순히 문제와 답을 외웠기 때문이다.

펭수는 한달간 과거 기출의 출제 의도를 파악하고 그 보기의 왜 정답인지를 이해하며 공부했다.

최근 5개년의 출제 동향이나 문항 패턴을 파악하였지만 어려운 문제도 있었기에, 기출문제를 전부 맞출 수는 없었다.

기출을 암기한 똑딱이와 내용을 이해한 펭수는 올해 수능 수험장에 들어갔다. 수능 결과가 어땠을까?



	똑딱이	펭수
5개년 과거기출	상위 1% (전부 외워서 만점)	상위 10% (어려운 문제 몇개 틀림)
올해 수능	상위 80% (기출문제 그대로 나온 것 + 짹은것만 맞춤)	상위 10% (어려운 문제 몇개 틀림)

여러분이 학부모라면 자녀를 어떻게 키우고 싶으신가요?

저는 자녀는 없지만... 대신에 인공지능을 연구하는 연구원으로서 제 모델은 펭수처럼 학습할 수 있도록 노력하고 있습니다.

여기서 똑딱이는 인공지능으로 따지자면 오버피팅된 모델이라고 볼 수 있습니다.

이미 습득한 데이터에 대해서는 모두 외웠기 때문에 기가 막히게 잘 맞출 수 있지만, 일반화 능력이 떨어지기 때문에 외운 것과 다른 데이터에 대해서는 제대로 역할을 수행하지 못합니다.

우리의 목적은 무엇일까요?

만일 여러분이 인공지능을 도입하려는 적용처가 늘 한정된 데이터 몇 가지만 다루는 곳이라면, 데이터를 외워서 추론하는 모델을 만들어도 무관합니다.

이 경우엔 기계학습이나 딥러닝 모델을 활용하지 않는다고 해도 간단하게 처리할 수 있는 경우가 많겠지요.

하지만 현장은 다양한 패턴의 데이터가 셀 수 없이 쏟아져 나오는 경우가 대부분일겁니다.

특히 사진, 동영상, 글, 음성 등과 같은 비정형 데이터를 다루는 곳이라면요.

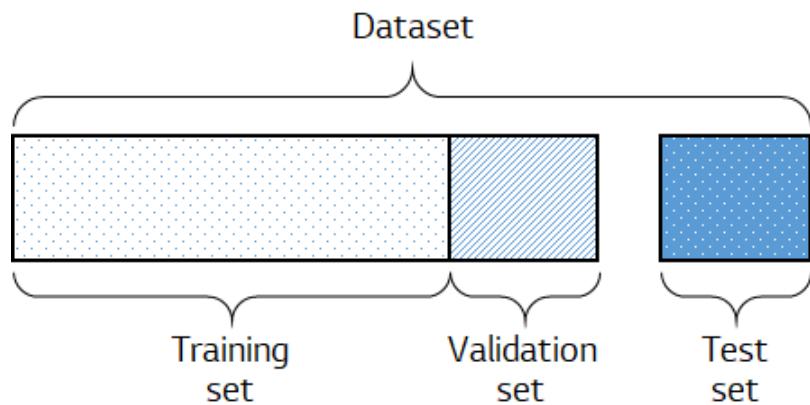
미리 쌓아놓은 데이터를 잘 맞추도록 학습하는 것은 물론 중요하지만 앞으로 실시간으로 발생할 데이터도 잘 추론하도록 하는 것이 우리의 목적이 됩니다.

즉, 운영 환경에서의 일반화 성능 또한 학습 과정에서의 최적화만큼이나 중요하며 우리는 늘 이를 염두에 두고 모델을 만들어야 합니다.

6.2.1 Training, Validation, Test

우리의 모델이 현장에서도 잘 작동할지는 어떻게 확인할 수 있을까요?

먼저, 우리가 확보한 데이터(Historical data)를 Training, Validation, Test의 세 set으로 나누고 각 set이 수행할 역할을 구분해주도록 하겠습니다.



[그림 5. Training/Validation/Test set의 구분]

엄격히 비율이 정해져 있는 것은 아니지만, 보통은 Training, Validation, Test set을 8:1:1, 6:2:2 정도로 구분합니다.

나누기 전에는 데이터를 골고루 섞어주어 set별로 데이터의 성향이 다르거나 치우침이 없도록 합니다.

각 set의 역할을 설명드리겠습니다.

6.2.1.1 Training set

Training set은 머신러닝/딥러닝 모델을 학습하는 데 이용하는 데이터입니다.

모델은 Training set의 입력 데이터와 정답을 보고, 정답을 더 잘 맞추기 위해 노력합니다.
이는 단순히 최적화(Optimization)에 해당합니다.

6.2.1.2 Validation set

Validation set은 머신러닝/딥러닝 모델에게 정답을 알려 줄 데이터는 아니지만, 우리가 모델을 튜닝하는 데 도움을 주는 데이터입니다.

즉 모델의 일반화 성능을 판단하여 이어질 실험을 계획하는 데 이용합니다.

모델은 Validation set의 정답은 본 적이 없지만 이 set의 입력 데이터만으로 일단 정답을 추론하게 됩니다.

모델이 배우지 않았던 데이터, 즉 처음 보는 데이터에 대해 얼마나 잘 맞추는지를 계산할 수 있으니 이 결과를 보고 우리는 실험을 개선하게 됩니다.

예를 들어 우리가 만든 모델이 Training set에 대해서는 잘 맞추는데 Validation set에 대해서는 너무 못맞춘다고 가정해 봅시다.

위에서 말씀드린 예제의 똑딱이처럼, 알려준 것만을 달달 외우고 일반화 성능이 부족한 오버피팅 현상이 발생했다고 볼 수 있습니다.

우리는 이를 통해 다음 실험시 오버피팅을 방지할 전략을 세울 수 있습니다.

오버피팅을 확인하는 방법과 해결 전략은 뒤이어 다루겠습니다.

6.2.1.3 Test set

Test set은 모델의 학습에 어떤 식으로도 전혀 관여하지 않는 데이터로, 오로지 모델의 최종 성능을 평가하기 위해 따로 떼어놓은 데이터입니다.

여러 모델간 성능을 비교할 땐 Test set에 대한 스코어를 활용합니다.

Training set으로 모델을 학습하고, Validation set에 대한 성능을 확인하며 모델을 개선해왔으니 그 결과를 공정히 평가하기 위한 기준이죠.

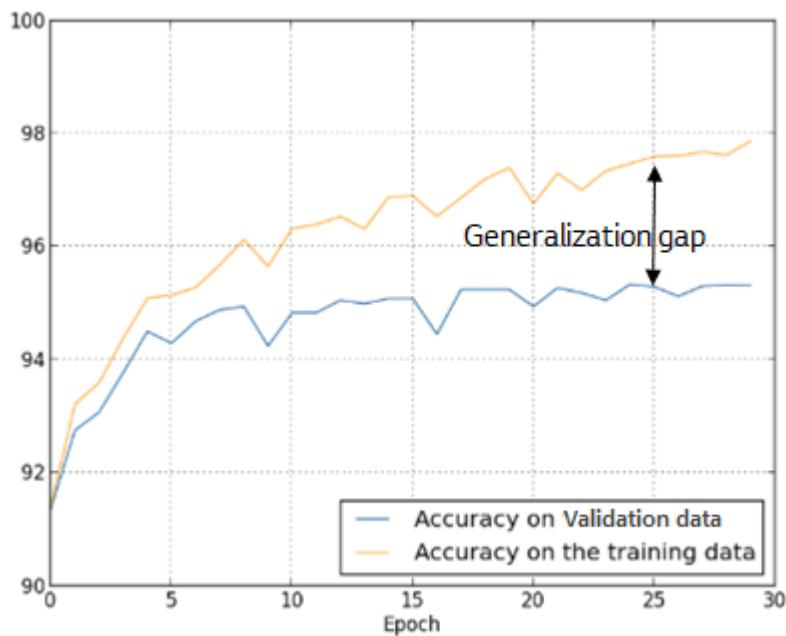
똑딱이와 펭수가 치른 금년도 수능 시험같은 데이터라고 생각하시면 됩니다.

모든 수험생이 제각기 다른 방식으로 공부했겠지만 모두가 처음 보는 문제로 동시에 공정하게 평가하는 기준이 수능입니다.

6.2.2 학습 곡선(Learning curve) 확인하기

역할에 따라 데이터를 나누었다면 학습이 제대로 이루어지는지, 일반화 성능이 떨어지거나 하여 오버피팅이 발생하지는 않았는지 확인해야 합니다.

확인하는 방법은 학습 곡선(Learning curve)을 그려보는 것인데, 학습 곡선이란 학습이 진행됨에 따라 모델의 성능을 기록하는 그래프입니다.



[그림 6. 학습 곡선의 예. 대체로 가로 축은 학습의 진행 시점을 나타낸다]

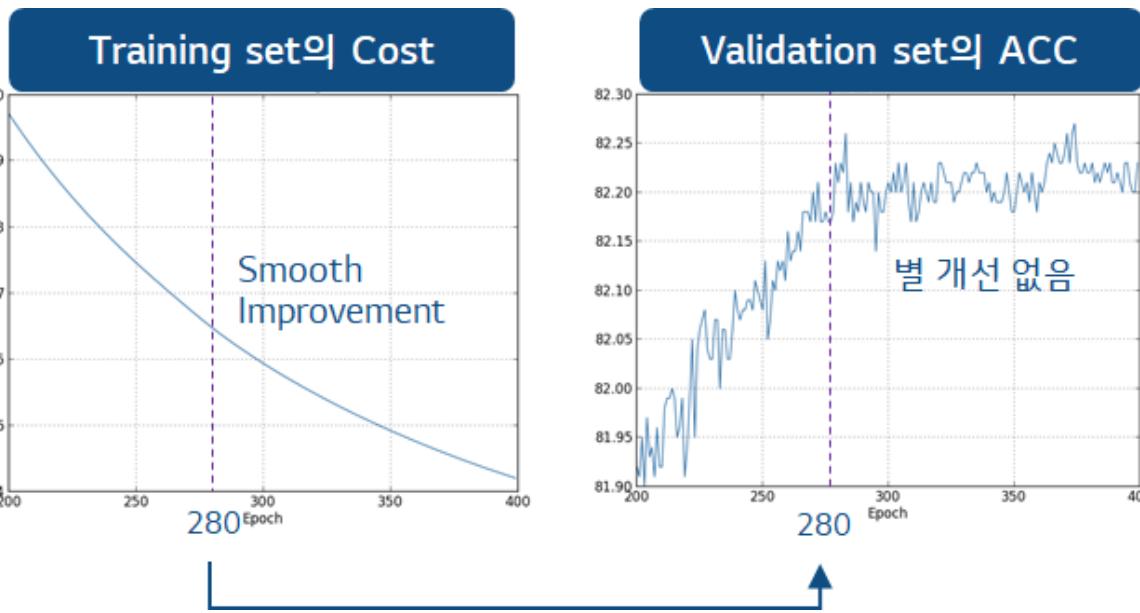
오버피팅이 발생했는지 확인할 수 있는 방법은 학습 곡선상에서 Training set와 Validation set에 대한 모델의 성능이 어떻게 변화하는지를 확인해보는 것입니다.

정답을 알면서 배우는 데이터에 대해서는 당연히 잘 맞춰야겠지만, 정답을 모르고 추론만 진행하는 데이터에 대해서는 어떨까요?

어느 순간 Validation set에 대해서 성능 개선이 없다면 모델은 '학습'이 아닌, '암기'를 하는 것이라 볼 수 있겠습니다.

전반적인 데이터의 패턴을 배운다기보다 주어진 데이터의 정답을 잘 맞추도록 문제와 정답을 달달 외우는 것이지요.

아래 학습 곡선은 전형적인 오버피팅의 예입니다.



[그림 7. 오버피팅이 발생한 경우의 학습 곡선]

[그림 7]의 경우 400 epoch(Training set 전부를 한 번씩 학습에 이용하는 단위)동안 모델 학습을 시켰습니다.

여기서 왼쪽의 그래프를 보면 Training set, 즉 모델에게 정답을 알려주면서 잘 맞추도록 유도하는 경우에 대해서는 문제 없이 최적화가 진행되는 것처럼 보입니다.

참고로 왼쪽 그래프의 세로 축을 이루는 Cost라는 것은 실제 정답과 모델이 예측한 예측값의 차이를 정량화해준 수치입니다. 작을 수록 모델이 잘 맞춘 것입니다.

하지만 오른쪽의 그래프를 보면, 모델이 정답을 모른 채 예측만 해야 하는 Validation set에 대해서는 280 epoch째부터 큰 개선이 없다는 것을 알 수 있는데요,

이 때의 세로축은 분류 정확도(Accuracy)로, 그 값이 높을 수록 분류를 잘 한다는 뜻입니다.

Training set에 대해서만 성능을 개선하고 있고, Validation set에 대해서는 별다른 향상이 없는 시점이 오버피팅 발생 시점입니다.

6.3 Regularization

만일 여러분이 머신러닝/딥러닝 모델을 학습할 때 [그림 7]과 같은 경향을 마주했다면 다음 실험에서는 이 현상을 개선하기 위한 전략을 취해야겠죠?

오버피팅을 피하기 위한 모든 전략들을 Regularization이라고 합니다.

굳이 번역하자면 '정규화'이지만, Normalization의 정규화와는 조금 다른 개념이므로 영어 원문으로 계속 표기하도록 하겠습니다.

Regularization의 목적은 Generalization, 즉 일반화 성능을 향상하는 데 있습니다.

Training set에 대해 더 잘 맞추고자 하는 것이 아닌, 현장에 나가 처음 보는 데이터에 대해서도 잘 맞출 수 있도록 하는 목적이지요.

이를 위해 취할 수 있는 여러가지 방법을 간단히 소개하겠습니다.

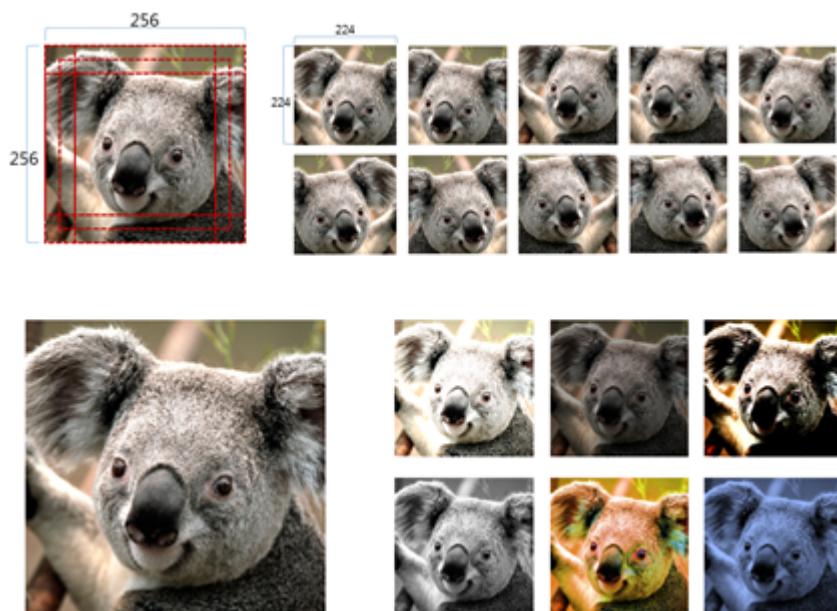
6.3.1 데이터 증강(Data Augmentation)

오버피팅을 피하는 가장 좋은 방법은 데이터를 더 많이 확보하는 것입니다.

다양한 데이터가 있다는 것을 알려준다면 인공지능 모델도 현장에 나갔을 때 새로운 데이터를 더 잘 맞출 수 있을 겁니다.

하지만 현실적으로 데이터가 절대적으로 부족한 경우가 대부분인데요,

이를 위해 얼마 없는 데이터 건수를 마치 많은 것처럼, 데이터를 증강시키는 다양한 기법이 있습니다.



[그림 8. 코알라 이미지 데이터 증강 예]

이미지 데이터 학습을 예로 든다면, 한 장의 이미지를 좌우 반전 시키거나, 일부 영역을 크롭하거나, 노이즈를 추가하거나,

색상, 명암, 채도 등에 변화를 주어 모델 학습에 추가로 데이터를 이용할 수 있습니다.

이렇게 하면 인공지능에게 더 다양한 환경에서 찍은 듯한 이미지를 학습시키는 효과를 주어 오버피팅을 방지할 수 있습니다.

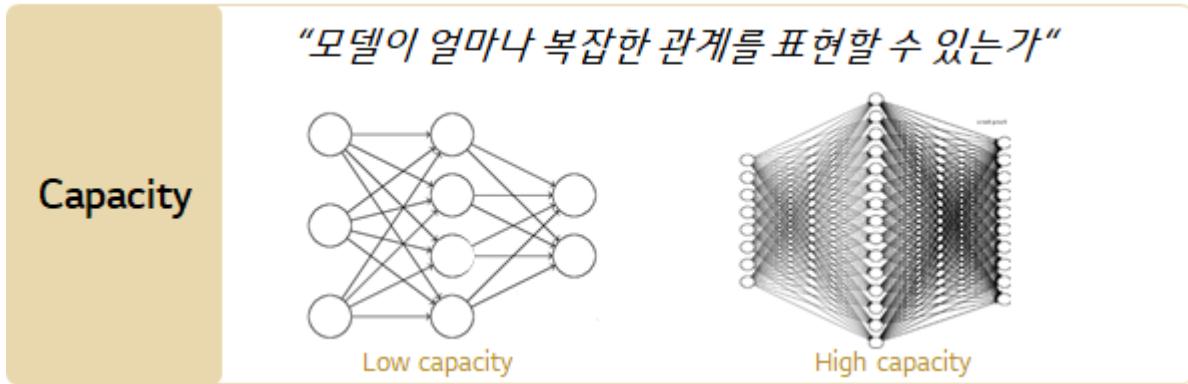
하지만 너무 과도한 변형은 오히려 해가 될 수 있으니, 주의해서 이용해야 합니다.

6.3.2 Capacity 줄이기

Capacity는 모델의 복잡한 정도를 나타내는 개념입니다.

일반적으로 딥러닝 모델이 머신러닝 모델보다 Capacity가 높으며, 그 중에서도 신경망을 여러층 쌓거나 뉴런의 수를 많이 둘 수록 Capacity가 높아진다고 말할 수 있습니다.

Capacity가 높은 모델은 처리할 데이터의 복잡 다양한 패턴을 더 잘 담아낼 수 있습니다.



[그림 9. 인공신경망의 Capacity]

하지만 Capacity가 필요 이상으로 너무 높은 모델은 주어진 데이터를 외우게 될 가능성이 높습니다.

따라서 내가 수행하려는 태스크에 알맞은 Capacity를 가진 모델을 선택하는 것이 좋은데요,

태스크의 복잡도와 Capacity의 관계는 딱 정해진 규칙이 없어서 어느 수준이 적정선이라고 말하기 어렵습니다.

만일 오버피팅의 경향이 발견된다 싶으면 내 모델의 층 수를 줄여본다든지, 한 층의 뉴런 수를 줄인다든지 등의 조치를 취해볼 필요가 있습니다.

6.3.3 조기 종료(Early stopping)

Early stopping이라는 기법은 너무나도 간단합니다.

말 그대로 오버피팅이 감지될 경우 목표하는 학습 시간이 다 되지 않았다고 하더라도 '조기 종료'해버리는 것인데요,

[그림 7]의 경우 400 epoch을 학습하기로 계획했지만, 오버피팅이 감지되니 280 epoch쯤에서 학습을 중단할 수 있습니다.

조기 종료 기능은 물론 코드 상으로 직접 구현해도 쉽지만, 요즈음의 기계학습 프레임워크에서는 이를 적용하기 쉽도록 관련 기능을 잘 패키징해둔 경우가 대부분입니다.

적용하기도 쉽고, 불필요하게 학습되는 경우를 방지하기 때문에 가급적 Early stopping을 적용해보는 것이 좋겠죠?

6.3.4 드롭아웃(Dropout)

드롭아웃은 학습 과정(Training pipeline)에서 일정 비율 p 만큼의 노드(인공 뉴런)를 무작위로 끄고 진행하는 Regularization 기법입니다.

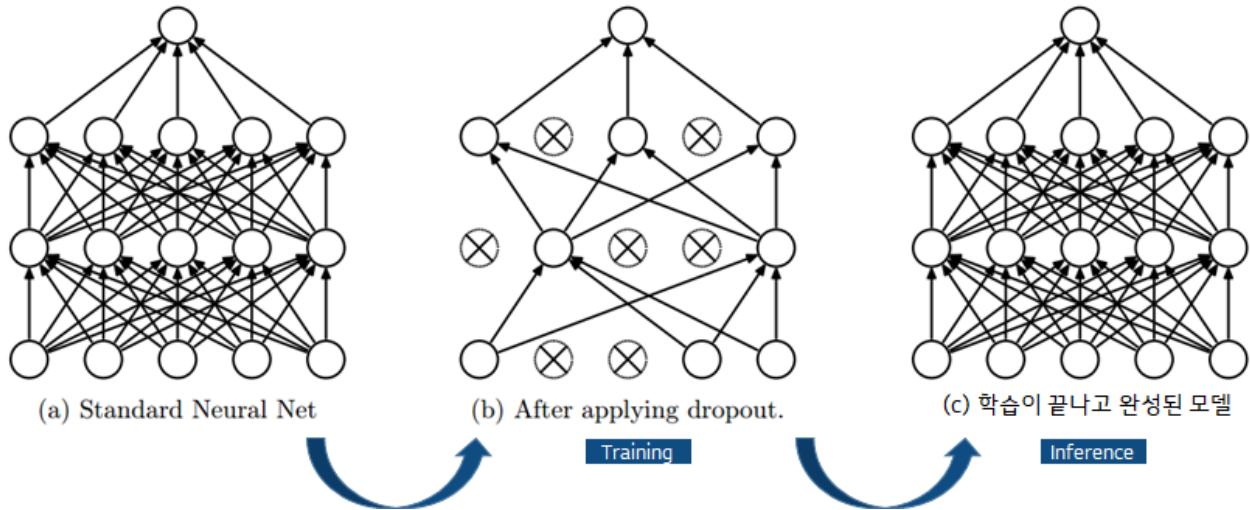
딥러닝 모델 학습 시 굉장히 많이 적용하는 Regularization 기법 중 하나인데요,

일부 노드가 사라진 상태에서 남아있는 노드만으로 어떻게든 정답을 맞춰야 하는 인공지능 모델은 훨씬 더 강력해진다고 하네요.

없어진 동료 노드의 둑까지 수행해야 하니 훨씬 기능이 강화된 정예부대가 되는 것이죠.

휴가철 회사의 팀원과 마찬가지입니다.

휴가를 떠난 동료의 뒷까지 수행해야 할 땐 힘들지만, 그만큼 한 명 한 명의 업무 스킬이 더 강화(?)된다고 볼 수 있겠지요.



[그림 10. 드롭아웃의 학습 과정 모습과 추론 과정 모습]

이외에도 L1/L2 Regularization, Batch Normalization, 앙상블 등.. 다양한 Regularization 기법이 있습니다.

여러가지 기법을 소개해드리는 것 보다는 Regularization의 취지가 무엇인지, 어떨 때 적용할 수 있는지를 기억하고 넘어가면 좋겠습니다. 😊

6.4 마무리

인공지능을 현장에 적용할 때도 '나무를 보지 말고 숲을 보라'는 말이 유효하게 적용됩니다.

이미 확보해놓은 과거의 데이터에 대해 좋은 성능을 낼 수 있도록 잘 학습하는 것,

이는 물론 굉장히 중요하지만 우리의 목적은 거기에서 끝나지 않습니다.

잘 만들어진 AI 모델이 향후 실제 고객사 현장에서도 좋은 성능을 내며 데이터를 처리할 수 있게 되는 것이 우리의 최종 목표일 겁니다.

위에서는 이를 Generalization이라는 용어로 소개드렸습니다.

알고 있는 것에만 몰두하여 새로운 환경에 적용하지 못하는 안타까운 일은 AI에서도 발생합니다.

오버피팅이 발생한다면 이를 해결하기 위한 다양한 Regularization 기법을 적용해보아야 합니다.

기계학습에서 한 번에 완벽한 모델이 만들어지는 경우는 없습니다.

첫 번째 모델의 성능이 엉망진창이라고 해도 실패한 것은 아닙니다.

다만 어떤 현상이 문제인지를 확인하고, 앞으로의 개선 전략을 잘 세워서 두 번째, 세 번째 모델을 만들 수 있으면 됩니다.

여기에서 의욕을 잃고 멈춰버린다거나, 고쳐나갈 방법을 모른다면 그땐 문제가 되겠지요.

이번 시간은 딥러닝 모델을 학습할 때 마주치기 쉬운 문제인 '오버피팅'에 대해 설명드렸습니다.

그리고 이 문제를 해결하는 다양한 'Regularization'의 대표적인 기법 몇 가지를 살펴보았습니다.

다음 시간은 '**인공지능 재활용하기, 전이학습(Transfer Learning)**'에 대해 알아보겠습니다.

감사합니다 😊

참고자료

- Justin Basilico, Making Netflix Machine Learning Algorithms Reliable, <https://www.slideshare.net/justinbasilico/making-netflix-machine-learning-algorithms-reliable>
- 자사 딥러닝 실무과정 교재보기, 김명지, [교재보기] 딥러닝 실무⁴⁰
- Michael Nielsen, Neural Networks and Deep Learning, <http://neuralnetworksanddeeplearning.com/>

⁴⁰ <http://wire.lgcns.com/confluence/pages/viewpage.action?pageId=73005264>

7 [7편] 다시쓰고 바꿔쓰자! 인공지능 재활용하기

안녕하세요, CTO AI빅데이터연구소입니다.

한 달에 두 번씩 **AI 테크레터**를 통해 인공지능 지식을 임직원 여러분들께 공유드리고 있습니다.

모든 CNSer가 이해하실 수 있도록 쉽게 작성하려고 하니, 상세 기술에 대한 궁금증이 생기시면 댓글이나 이메일을 통해 언제든 연락 바랍니다 😊

본 업로드는 [TECH wiki AI게시판](#)(see page 7)에서 연재됩니다.

작성 : CTO AI빅데이터연구소 AI기술팀 [김명지 팀장/총괄 CONSULTANT](#)/[언어AI LAB](#)⁴¹

- 쉽지 않은 인공지능 적용하기(see page 105)
 - 구체적이지 않으며 불명확한 태스크(see page 106)
 - 적은 데이터, 낮은 품질의 데이터(see page 108)
 - 다른 도메인 환경(see page 109)
 - Transfer Learning : 한번 만든 인공지능 모델 우려먹기(see page 111)
 - Catastrophic forgetting : 치명적인 기억상실!(see page 113)
 - 더 나은 Transfer Learning을 위한 방법(see page 113)
 - Transfer Learning 모델 이용(see page 115)
 - 컴퓨터 비전에서의 Transfer Learning(see page 115)
 - 자연어 이해(NLU)에서의 Transfer Learning(see page 117)
 - 마무리(see page 117)
-

지난 시간에는 딥러닝 모델을 학습할 때 마주치기 쉬운 문제인 '오버피팅'에 대해 설명드렸습니다.

그리고 이 문제를 해결하는 다양한 'Regularization'의 대표적인 기법 몇 가지를 살펴보았습니다.

오늘은 한 번 만들어놓은 딥러닝 모델을 알뜰살뜰 다음 태스크에도 활용하는 전이학습(Transfer Learning)에 대해 알아보겠습니다.

지난 시간까지의 내용이 궁금하신 분은 ★[AI Tech Letter](#)(see page 7)★를 확인하시기 바랍니다.

7.1 쉽지 않은 인공지능 적용하기

AI, 그 중에서도 특히 딥러닝 모델을 현장에 적용하려는 시도를 해 보신 분이 있다면 모두 공감할 이야기가 있습니다.

"아니 AI는 왜 배울땐 모든 다 될 것 같았는데 내가 해보려고 하면 엉망진창인거야??"

⁴¹<https://wire.lgcns.com/confluence/display/~78628>

물론 어느 기술분야고 그렇지 않겠냐마는 특히나 딥러닝 기반의 인공지능은 기술을 습득할 때와 실제 현장에 적용할 때의 괴리가 큰 분야중 하나입니다.



AI를 글로만 배운 사람

실제 현장에 적용된 AI 모델

[그림 1. 늘 이상과 현실은 괴리가 있기 마련이죠. 하지만 AI는 그게 너무 크지...]

최신 기술을 소개하는 논문부터 모델 학습에 필요한 상세 코드까지, 오픈 사이언스인 딥러닝은 누구나 접근 가능한 곳에 무료로 모든 자료가 공개되어있습니다.

글로벌 탑 티어 학회에 소개된 논문은 순식간에 전세계의 연구자들에 의해 블로그에 쉽게 재해석되며 유튜브 영상으로 만들어지고,

깃털에 올라오는 공식 코드와 모델은 누구나 이용하기 편리하도록 2차 3차 패키징이 되곤 합니다.

온라인 커뮤니티는 최신 트렌드를 소개하고 다양한 기사와 의견, 자신만의 참신한 AI 프로젝트를 소개하는 사람들로 매일 붐빕니다.

하지만 이 수많은 자료들은 쉽고 편하게 잘 다져진 튜토리얼에 불과합니다.

대부분의 내용은 정제된 다량의 데이터와 잘 만들어진 환경, 제한된 태스크에 대한 것이죠.

이것들이 각자가 접한 실생활의 문제 해결을 위해 도입될 때에도 잘 작동하리라는 보장은 없습니다.

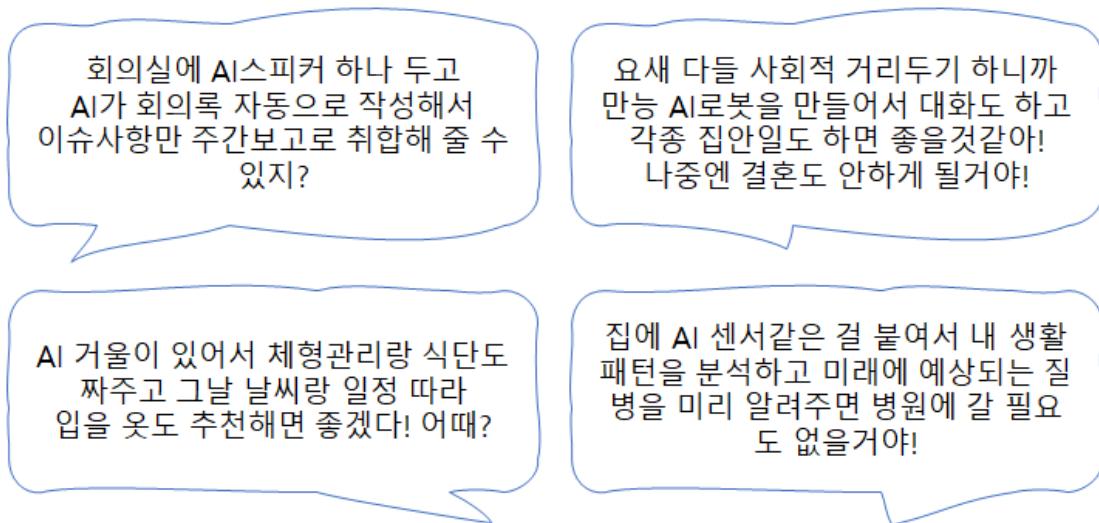
현실에는 다음과 같은 걸림돌이 있기 때문입니다.

7.1.1 구체적이지 않으며 불명확한 태스크

AI를 막 접하기 시작한 사람들 및 대다수의 고객은 AI에 대해 큰 기대를 하고 있습니다.

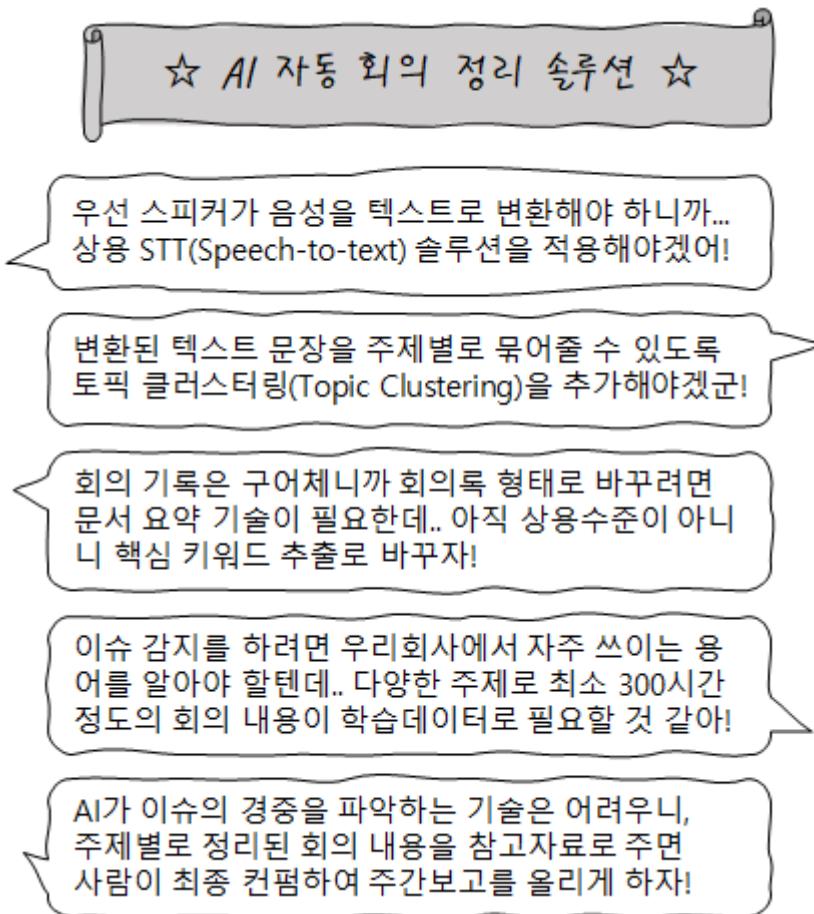
AI가 우리 삶의 문제의 모든 것을 해결해 줄 수 있을 거라고 막연히 생각하지요.

AI의 특장점이나 개념적인 부분, 포장된 사례만을 접했다면 꿈같은 요구사항을 내걸 수 있습니다.



[그림 2. 아뇨 그런거 못합니다... (※위 내용은 실제 고객의 요구사항과 무관합니다)]

AI는 삶의 많은 부분을 더 편리해 지도록 만들어 줄 수는 있겠지만, 모든 일에 대한 근본적인 해결책은 아닙니다. 두루뭉실하고 막연한 요구사항보다는 구체적이고 명확한 목표를 세우고 필요한 하위 기능을 쪼개어 생각해야 합니다. 예를 들어 첫 번째 요구사항의 경우,



[그림 3. 요구사항 구체적으로 break-down 하기]

등등의 구체적인 하위 기능으로 쪼개볼 수 있겠습니다.

하지만 대부분의 고객은 AI 전문가가 아니기 때문에 요구사항을 기능 단위로 쪼개기 어려운 경우가 많을 겁니다.

어떤 기능들이 AI로 가능한지, 이 중 어떤 것은 현실적으로 아직 상용화하기에 무리가 있을지, 어떤 부분은 굳이 AI가 아니어도 처리할 수 있는지 등은

기술을 깊이 이해하고 적용해 본 경험이 없는 사람이라면 쉽게 도출하기 어려운 내용입니다.

이런 경우 기술을 더 잘 알고 있는 우리가 상세 기능을 나누고 각 목표 수준을 정의한 다음 전체를 아우르는 구성을 어플리케이션 레벨로 설계할 수 있어야겠죠?

7.1.2 적은 데이터, 낮은 품질의 데이터

태스크가 구체적으로 정해졌다면 이제 AI 모델을 학습시킬 차례입니다.

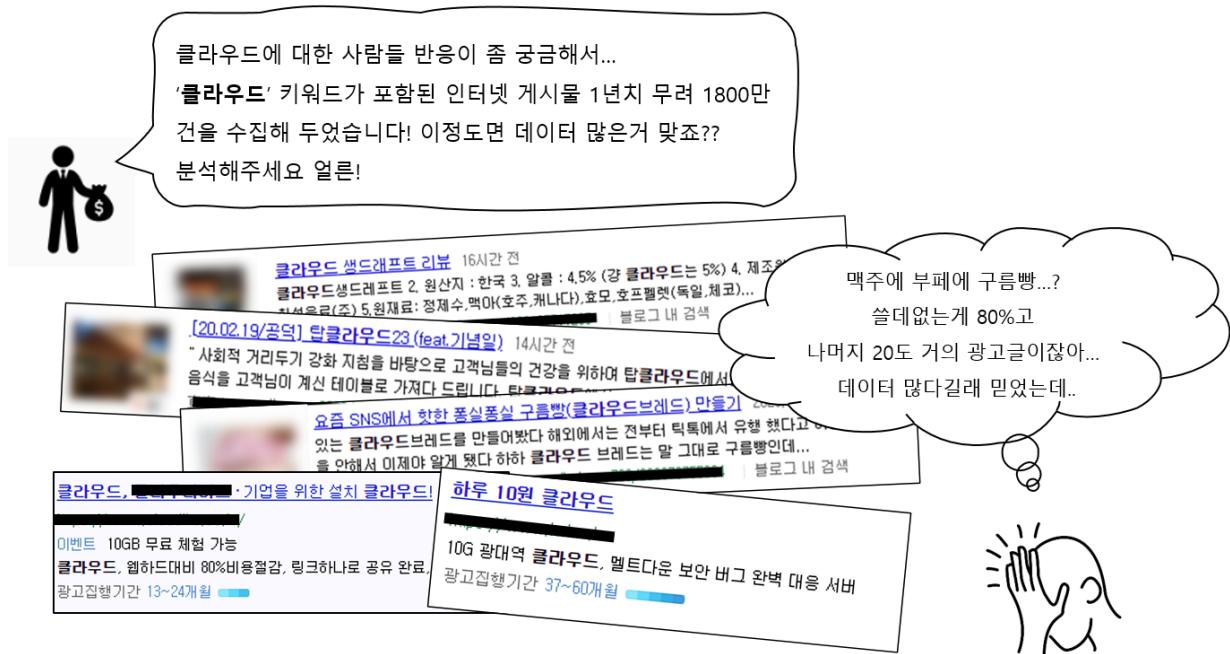
단순하고 정해진 패턴을 다루는 태스크라면 상관이 없지만, 이미지나 텍스트같은 비정형 데이터를 다루거나

코드로 판단 및 예측 규칙을 명문화하기 어려운 태스크라면 딥러닝 기반의 AI 모델을 이용하는 것이 좋습니다.

딥러닝 모델 학습은 사람이 규칙을 지정하지 않는 대신 데이터를 통해 패턴을 학습해야 하는데요,

그 학습 방법의 특성상 양질의 다양한 데이터를 필요로 합니다.

하지만 대부분의 경우 고객이 데이터를 차곡차곡 모아놓은 뒤 사업을 발주하는 경우는 드물고, 데이터를 보유하고 있다고 해도 학습데이터로 활용하기에 부적격인 경우가 다반사입니다.



[그림 4. 데이터 정제는 숙명입니다. (※위 내용은 실제 고객의 요구사항과 무관합니다)]

그렇다고 해서 학습 데이터를 마련하는 것부터 시작하자니, 이 또한 쉽지 않습니다.

요즈음은 인공지능 학습용 데이터를 라벨링하고 전처리해주는 크라우드 소싱 업체도 많이 생겼지만

그래도 여전히 데이터의 소스를 제공해야 합니다. (예: 공정품의 양품/불량 체크를 위해 공정품 사진을 찍어 제공)

이마저도 다량의 데이터를 구축할 경우 비용이 많이 들어갈 수도 있고, 대부분 고객데이터이다 보니 데이터의 반출이 불가능하여 외부업체를 활용하기 어렵지요.

AI 모델 개발자는 늘 모자란 데이터와 싸워야 합니다.

7.1.3 다른 도메인 환경

도메인 환경이 달라지는 것도 문제가 됩니다.

아무리 오버피팅을 피하고 General하게 만든 모델이라고 해도 추론 환경이 달라지면 성능이 감소하기 마련입니다. (오버 피팅 참고(see page 91))

예를 하나 들어보겠습니다.

동네에서 편의점을 운영중인 만득이는 아르바이트를 구하는 것이 어려워 야간시간에는 무인편의점으로 바꾸려 한다.

이에 CCTV를 여러 대 설치해두고, 계산하지 않고 나가는 손님이 발생하면 알림을 주도록 <AI 절도 감지 모델>을 만들기로 했다.



[그림 5. 무인 결제는 LG CNS가 담당할테니 만득이는 절도감지에 힘쓰라구~! (자료: LG CNS 본사 3층 무인편의점)]

만득이는 매장에서 다양한 절도 상황을 연출하여 데이터를 만들고, 모델을 학습시켰다.

감지 성능은 놀라웠다. 만득이의 모델은 한 달 동안 발생한 모든 절도 사건을 100% 검거했다.

"이런 매장이 우리 가게뿐이 아닐텐데, 이 모델을 다른 매장에 팔아 이용료를 받으면 나는 부자가 되겠구만!"

만득이는 해당 모델을 주변 편의점 점주들에게 팔았다. 하지만 결과는 꽝이었다.

만득이의 모델은 다른 편의점에서 전혀 절도를 감지하지 못했다.

만득이는 손해배상을 하기 위해 자신의 매장을 내놓아야 했다.

만득이가 간과한 것은 도메인 적응 문제(Domain adaptation problem)입니다.

대부분의 딥러닝 모델은 동일한 기능을 수행하는 모델이라고 해도 추론 환경이 달라지면 제 기능을 수행하지 못합니다.

다른 매장의 CCTV 화질, 밝기와 채도, 채광, 조명, 카메라 위치, 매장 내 크기, 선반 위치 등은 만득이의 매장과 상이할 겁니다.

이런 환경에서의 절도행위는 모델이 학습한 적이 없기 때문에 제대로 검거하지 못하게 되는 것이죠.

만득이네 편의점에서는 높은 검거율을 보였을지라도, 이는 만득이의 매장 환경에 한해서만 최적화되어 발휘된 성능입니다.

다양한 편의점에서도 잘 작동하려면 다양한 편의점의 절도 데이터를 만들어 학습시켜야 합니다. 또 다시 데이터 부족과 다양성 문제로 넘어가게 되는 것이죠.

이처럼 하나의 AI 모델이 좋은 성능을 보인 전적이 있다고 해도 다른 고객사 환경에서 여전히 잘 작동한다는 보장은 할 수 없습니다.

이렇게 실제 현장에 AI를 도입할 때에는 결과를 낙관할 수만은 없습니다.

열에 아홉은 데이터 부족으로 모델 학습에 고생하고, 잘 만들어진 모델이라 해도 또다른 곳에서 잘될 것이라 장담하지도 못합니다.

공부할 때 배웠던 잘나간다는 AI 모델들은 다 이론적인 허상에 불과할까요?

확장성 없이 매 번 수 만 건의 학습데이터를 만드는 셋바퀴를 반복해야 할까요?

다 방법이 있습니다!

7.2 Transfer Learning : 한번 만든 인공지능 모델 우려먹기

'전이 학습'으로도 불리는 **Transfer Learning**은 한 번 만들어진 딥러닝 모델을 재활용하여 쓸 수 있는 기법입니다.

비슷한 태스크를 다른 도메인에 적용할 때, 그리고 그 태스크를 위한 학습 데이터가 부족한 경우 유용하게 쓰일 수 있습니다.

유참고로, 유사 용어 Fine-tuning(미세조정)이 있긴 합니다만, 더 넓은 의미에서 Transfer Learning을 알아둡시다.

[참고] Fine-tuning: 해독은 인공지능 모델 다시쓰기

Fine-Tuning은 더 정교하게 파라미터를 튜닝하는 것으로,

온라인 프로세스([참고\(see page 0\)](#))에 적용된 AI 모델이 데이터 패턴 변경 등으로 인해 추론(inference) 성능이 떨어지게 될 때

이를 보강하고자 기존 모델에 새로운 데이터 등을 넣어서 파라미터를 재학습하는 작업을 말합니다.

다시말해서 동일한 태스크에 대해 한번 만들어진 모델의 성능을 보강하기 위한 목적입니다.

예를 들어 쇼핑몰 리뷰를 공정/부정 분류하는 모델이 있다고 가정합시다.

시대 흐름에 따라 신조어가 등장하고, 새로운 상품에 대한 리뷰가 등장하면 모델의 분류 성능이 저하될 수 있습니다.

이 때 새로운 데이터를 추가하여 인공지능을 이어 학습시키면 새로이 등장한 패턴에 대해서도 잘 분류하게 됩니다.

즉 기존 모델의 v2, v3 .. 업데이트 버전을 위한 기법이라고 볼 수 있겠네요.

반면 Transfer Learning은 대체로 만들어진 모델을 다른 태스크에 적용하려 할 경우 활용하는 기법입니다.

예를 들어 아래와 같은 다양한 경우 Transfer Learning을 적용해볼 수 있습니다.

- 만득이네 편의점 절도 감지 모델을 짱구네 편의점에 적용하려 할 경우
- 영화 리뷰 평가에 대한 긍/부정 분류 모델을 뉴스기사 댓글 긍/부정 분류에 적용하려 할 경우
- A 회사의 이메일 보안 위반 탐지 모델을 B 회사에 적용하려 할 경우
- 일상생활의 다양한 이미지를 분류하는 모델을 농산품 품종 이미지 구분에 적용하려 할 경우
- 일반 상식에 대한 질의응답 모델을 금융권 콜센터 무인자동 질의응답 과제에 적용하려 할 경우
- ... 등등

비슷한 경우에 대해 만들어 놓은 AI 모델이 있다면 다른 태스크 진행시 맨땅에서부터 모델을 만들 필요가 없습니다.

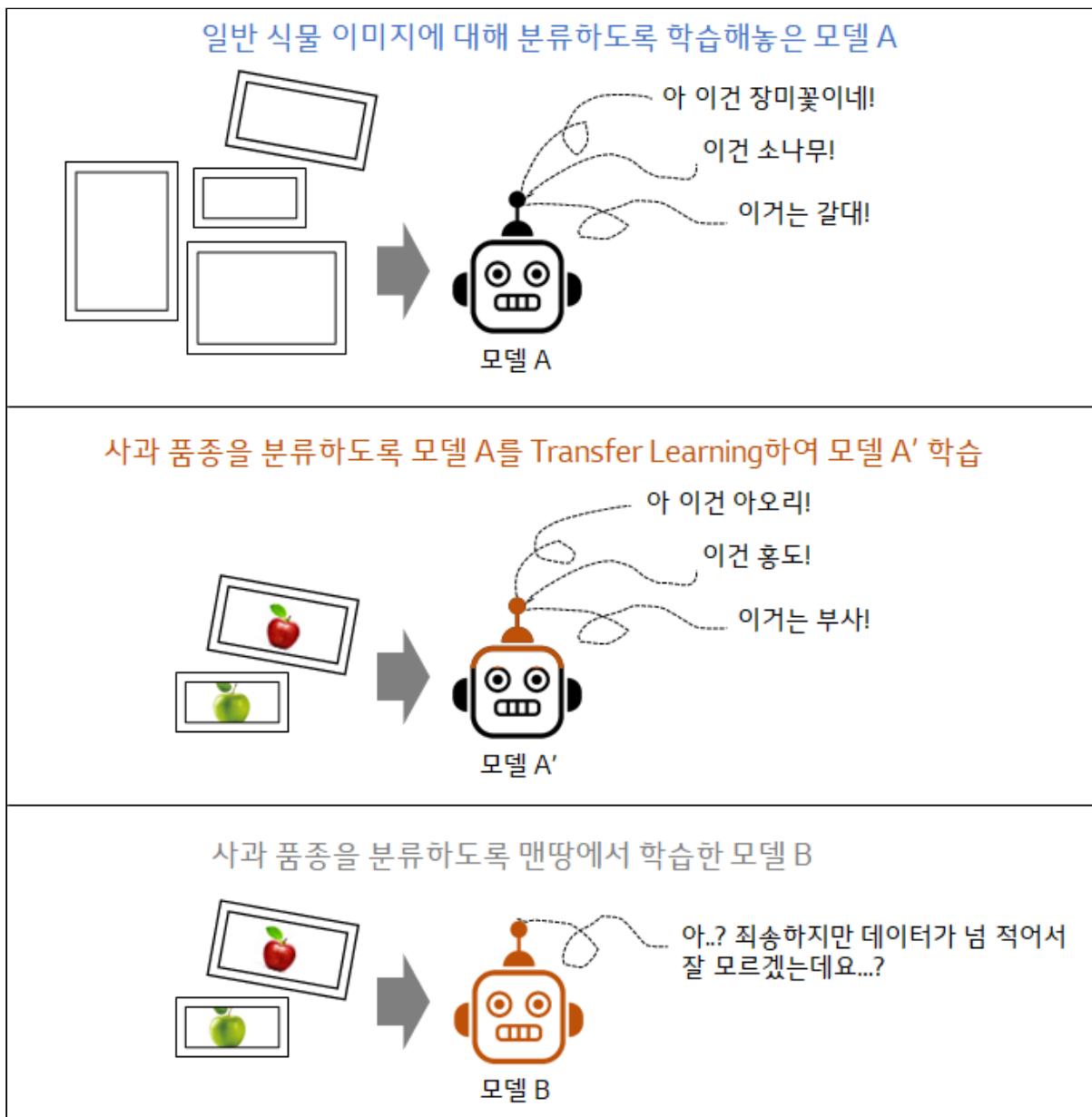
하지만 엄연히 환경이 다르고 데이터의 패턴이 다르기 때문에 그 모델을 그대로 활용해서는 좋은 성능이 나올 수 없죠.

Transfer Learning은 하나의 모델이 이미 배워놓은 지식은 잘 유지하면서도, 새로운 태스크에 대해 필요한 지식을 추가로 습득할 수 있도록 합니다.

이미 만들어 놓은 모델의 아키텍처를 새 태스크에 맞게 조금 수정하거나 추가하는 작업을 한 뒤

새로운 태스크에 대한 학습 데이터를 이어서 학습시킨다면 처음부터 학습한 모델보다 좋은 성능을 낼 수 있습니다.

새로운 학습데이터가 기존만큼 많지 않아도 말이죠.



[그림 6. 사과 품종 분류를 위한 Transfer Learning 예]

예를 들어 사과 농장으로부터 의뢰를 받아 사과 사진으로 사과 품종을 자동 분류하는 과제를 수행한다고 가정하겠습니다.

하지만 확보한 품종별 사과 사진이 몇 장 없는 어려운 상황입니다.

이대로 학습한다면 몇 장 없는 이미지로부터 모델이 충분히 특징을 학습하지 못하여 분류를 제대로 수행하지 못하게 됩니다. (모델 B)

하지만 약간 갈래는 다른듯 하나 일반 식물의 종류를 구분하는 AI 모델(모델 A)을 만든 적은 있습니다.

만들어둔 모델에 어떻게든 약간 숟가락을 얹어서 사과도 잘 분류하게 할 수 없을까요?

우선 모델 A의 마지막 부분은 일반 식물들을 분류하도록 되어있을테니, 사과 품종 카테고리 수에 맞게 분류할 수 있도록 인공신경망을 조금 수정해주어야 합니다.

이후 주어진 소량의 사과 이미지 데이터를 여기에 추가로 학습시켜서 새로운 모델 A'를 만들어냅니다.

모델 A는 기존에 '식물 이미지' 데이터의 일반적인 특징(색상, 각도, 선, 열매, 잎사귀 등..)을 학습한 적이 있으니,

Transfer Learning을 적용한 모델 A'가 처음부터 소량의 사과만 학습한 모델 B보다는 더 적은 학습량으로 더 많은 지식을 가질 수 있는 것이죠.

7.2.1 Catastrophic forgetting : 치명적인 기억상실!

태스크에 맞게 수정하여 추가로 학습하면 모든 것이 좋아질 듯 하지만 이것이 만능 해결책은 아닙니다.

딥러닝 모델이 새로운 정보를 학습할 때 이전에 배웠던 정보를 완전히 잊어 버리는 경향이 발생할 수 있는데, 이를 Catastrophic forgetting이라고 합니다.

사과 분류 모델의 예시로 돌아가봅시다.

모델 A'가 몇 장 없는 사과 데이터에 대해 너무 여러번 반복학습하다 보면 모델 A가 이전에 학습했던 기본적인 식물 이미지의 특징들을 잊어버리게 되는 것이지요.

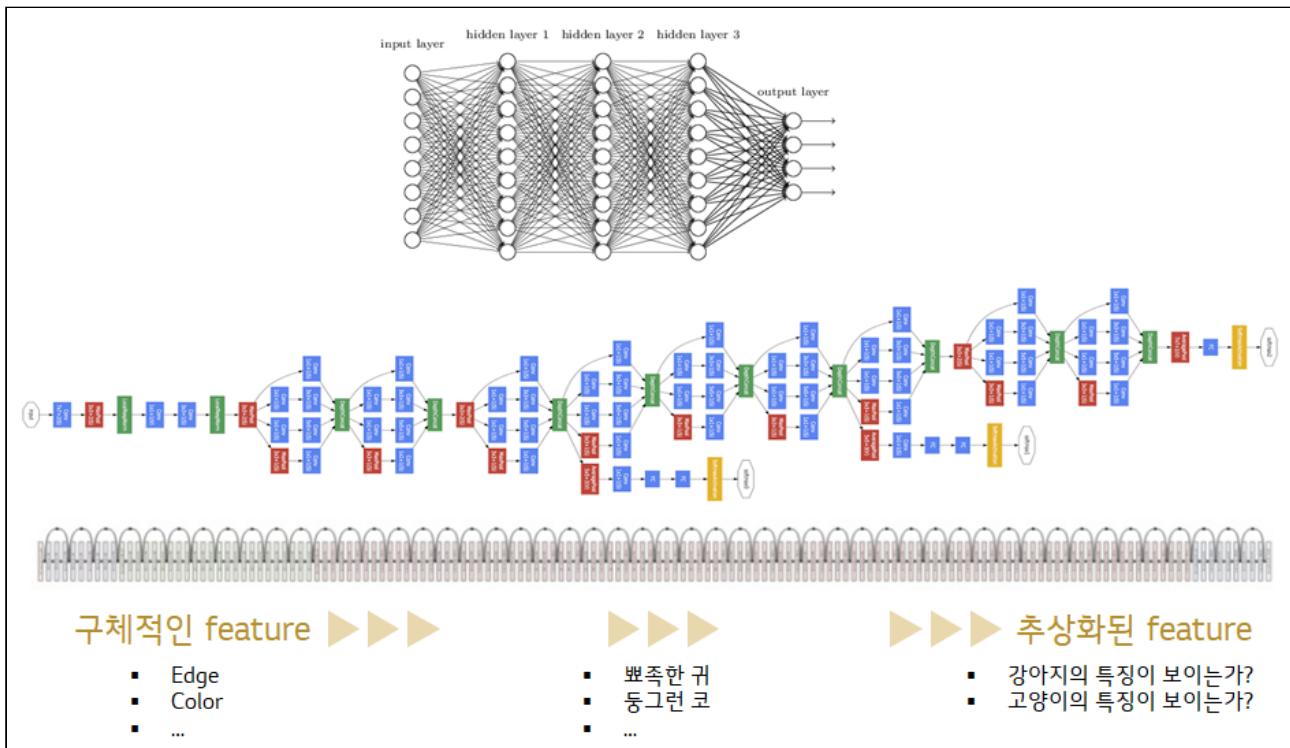
전이(Transfer)가 필요 이상으로 과하게 일어났다고 볼 수 있겠습니다.

Catastrophic forgetting이 발생하지 않도록 하면서도 새로운 데이터를 잘 배우기 위해서는 다음과 같은 기법을 활용해야 합니다,

7.2.2 더 나은 Transfer Learning을 위한 방법

일반적인 딥러닝 모델은 두 층(layer) 이상의 인공신경망으로 구성되어 있습니다.

이 신경망은 초반에는 데이터의 구체적이고 기본적인 특징을, 후반부로 갈수록 특정 태스크를 위한 추상적이고 개념적인 특징을 학습하게 됩니다.



[그림 7. 강아지와 고양이를 분류하는 인공신경망 예]

이미지든 텍스트든 이미 학습해두었던 모델은 해당 데이터의 기본적인 특징을 어느정도 알고 있다고 볼 수 있습니다.

따라서 새 태스크에 기 보유 모델을 Transfer Learning 하려 할 땐 기본 특징 학습은 건너 뛰고 바로 태스크를 위한 학습으로 가는 것이 좋습니다.

이를 위해 후반부의 신경망 층에 대해서만 파라미터를 학습하고, 전반부의 파라미터는 학습되지 않도록 고정해놓는 기법을 적용할 수 있습니다.

이를 레이어 동결(Layer freezing)이라 부릅니다.

새로운 태스크가 기 학습한 태스크와 크게 상이하지 않은 경우라면 대부분을 동결시켜도 되고,

태스크가 많이 상이한 경우라면 동결을 풀어서 조금 더 앞쪽의 층까지 학습할 수 있도록 설정할 수도 있습니다.

또는 다 동결시켜놓았다가 학습이 진행될수록 후반부부터 서서히 동결을 푸는 방법(Gradual Unfreezing)도 적용할 수 있으니 상황에 맞게 유연하게 대처해 볼 수 있습니다.

이외에 층마다 Learning rate(학습률)의 차별을 두는 것도 한 방법입니다. (Discriminative fine-tuning)

Learning rate이란 한 번에 인공신경망의 파라미터를 얼마나 업데이트 시킬지에 대한 정도로, 사람이 설정하는 값(hyperparameter)입니다.

전반부의 인공신경망은 기본적인 특징을 이미 배워놨으니 새롭게 고칠 것이 별로 없을 것이고

따라서 새로운 태스크에 적용할 때 Learning rate을 작게 설정하는 것이 좋습니다.

반면 후반부의 인공신경망은 특정 태스크를 위한 부분으로, 새롭게 배워나가야 하는 부분이 많을테니

Learning rate을 크게 설정하여 빠르게 성큼성큼 학습하는 편이 좋습니다.

이외에도 Transfer Learning을 효과적으로 하기 위한 다양한 기법이 있으니, 상황에 맞는 기법을 활용할 수 있어야합니다.

7.3 Transfer Learning 모델 이용

Transfer Learning은 보통 오픈도메인 데이터에 대해 만들어놓은 모델을 특정 도메인의 태스크에 적용하는 식으로 활용합니다.

일반적인 내용을 두루두루 넓게 학습해놓은 모델의 사전 지식을 구체적인 하위 영역에 활용하는 셈입니다.

7.3.1 컴퓨터 비전에서의 Transfer Learning

이미지를 입력 데이터로 처리하는 경우라면 이미지넷([참고\(see page 0\)](#)) 데이터로 학습된 모델에 Transfer Learning을 적용하여 좋은 성능을 낼 수 있습니다.

이미지넷은 무려 120만 장의 학습 이미지를 1,000개 카테고리로 분류하는 과제이기 때문에, 여기에 대해 학습한 모델은 이미지를 꽤나 잘 배웠다고 볼 수 있습니다.

특히 이미지넷 데이터 인식 대회에서 우승을 했던 GoogleNet(2014 우승)이나 ResNet(2015 우승) 등은 지금까지도 널리 활용되는 기본 모델이지요.



[그림 8. ResNet의 구성]

이외에도 다양한 특장점을 지닌 모델들의 아키텍처가 전부 논문으로 공개되어있으며, 이 중 대부분은 깃헙 등에 학습된 모델이 오픈되어있습니다.

이미지 처리를 하는 과제를 진행중이라면, 밑바닥부터 학습시키기보다는 이런 자료를 활용하여 Transfer Learning 하는 것이 좋겠죠?

7.3.2 자연어 이해(NLU)에서의 Transfer Learning

텍스트란 특허나 함축적인 데이터 타입이기 때문에 Transfer Learning을 활용하는 것이 매우 효과적입니다.

위키백과와 같은 방대하고 일반적인 지식에 대해서 학습한 적이 있는 모델이라면 일반적인 어휘의 의미를 대강 알고 있는 모델이라고 볼 수 있습니다.

이런 모델이 공개되어있다면 Transfer Learning에 활용하는 것이 좋겠지요.

자사에서 만들어 공개한 KorQuAD(Korean Question Answering Dataset)는 한국어 위키백과를 스스로 한 오픈도메인 질의응답 데이터인데요,



[그림 9. LG CNS가 만들고 공개한 한국어 표준 질의응답 데이터 KorQuAD ([바로가기](#)⁴²)]

자연어 질의응답이란 인공지능이 수행하기 상당히 어렵고 복잡한 과제 중 하나입니다. 따라서 다량의 학습 데이터를 필요로 합니다.

하지만 KorQuAD로 학습해둔 모델이 있다면 콜센터나 챗봇 등 특정 고객사의 질의응답 데이터가 부족해도 상대적으로 좋은 성능을 얻을 수 있습니다.

7.4 마무리

여러 번 언급하지만, 딥러닝은 데이터로부터 패턴을 학습하여 의사결정하는 모델이기 때문에 양질의 데이터가 충분한 환경이라면 아무 문제 없습니다.

하지만 현실적으로 우리는 데이터가 부족한 상황에 놓이는 경우가 대부분입니다.

과거 경험에 이어 소량의 데이터를 추가 학습하여 좋은 성능을 짜내보자! 하는 것이 Transfer Learning이라는 AI 재활용 기법입니다.

AI 공부를 하려고 이론을 배우거나 튜토리얼 코드를 실습할 때 필요한 것이 아닌, 현장을 위한 기술이죠.

그래서 분류(Classification)와 같이 어느정도 기술수준이 성숙된 과제에서는 Transfer Learning을 얼마나 잘 적용할 수 있느냐가 인기있는 후속 연구주제입니다.

AI 모델을 제대로 학습시키기 여려운 만큼, 한 번 만들어 놓았다면 알뜰살뜰하게 끝까지 잘 쓸 수 있어야겠습니다.

이번 시간은 딥러닝 모델을 재활용하는 Transfer Learning에 대해 설명드렸습니다.

⁴² <https://korquad.github.io/>

Transfer Learning은 일반적인 오픈도메인 대해 사전학습해놓은 모델을 특정 close-domain의 태스크에 적용할 때도 활용할 수 있는데요,

다음 시간에는 이러한 몸풀기 학습이라 할 수 있는 '**AI 사전학습 및 자기주도학습(Pre-Training and Self-supervised Learning)**'에 대해 자세히 알아보겠습니다.

감사합니다 😊

참고자료

- 무인편의점 '스마트 GS25' 가 보니, 동아일보, <https://www.donga.com/news//article/all/20180928/92185150/3>
- Transfer learning & fine-tuning, Keras, https://keras.io/guides/transfer_learning/
- 위키피디아 - Transfer learning, https://en.wikipedia.org/wiki/Transfer_learning
- 위키피디아 - Catastrophic inference, https://en.wikipedia.org/wiki/Catastrophic_interference
- Universal Language Model Fine-tuning for Test Classification, J. Howard & S. Ruder, 2018, <https://arxiv.org/abs/1801.06146>
- 자사 딥러닝 실무과정 교재보기, 김명지, [\[교재보기\] 딥러닝 실무⁴³](#)
- Going Deeper with Convolutions, C. Szegedy, et al., 2014, <https://arxiv.org/abs/1409.4842>
- Deep Residual Learning for Image Recognition, K. He, et al., 2015, <https://arxiv.org/abs/1512.03385>
- KorQuAD, [https://korquad.github.io⁴⁴](https://korquad.github.io)

⁴³ <http://wire.lgcns.com/confluence/pages/viewpage.action?pageId=73005264>

⁴⁴ <https://korquad.github.io>

8 [8편] 준비된 인공지능, Pre-trained AI

안녕하세요, CTO AI 빅데이터연구소입니다.

한 달에 두 번씩 **AI 테크레터**를 통해 인공지능 지식을 임직원 여러분들께 공유드리고 있습니다.

모든 CNSer가 이해하실 수 있도록 쉽게 작성하려고 하니, 상세 기술에 대한 궁금증이 생기시면 댓글이나 이메일을 통해 언제든 연락 바랍니다 😊

본 업로드는 [TECH wiki AI게시판](#)(see page 7)에서 연재됩니다.

작성 : CTO AI 빅데이터연구소 AI기술팀 [김명지 팀장/총괄 CONSULTANT](#)/[언어AI LAB](#)⁴⁵

- Pre-training: 니가 뭘 요청할지 몰라서 일단 다 공부해놨어(see page 120)
 - 대규모 데이터에 대한 Pre-training(see page 121)
 - 시각 데이터에 대한 사전학습(see page 121)
 - 언어 데이터에 대한 사전학습(see page 123)
 - Self-Supervised Learning: 나 혼자 어떻게든 해볼게(see page 125)
 - 예: 이미지 데이터를 위한 Self-Supervised Learning(see page 129)
 - 예: 텍스트 데이터를 위한 Self-Supervised Learning(see page 131)
 - 예: Google BERT(Bidirectional Encoder Representations from Transformers)(see page 132)
 - 마무리(see page 133)
-

특정 태스크에 인공지능을 적용하는 것은 쉬운 일이 아닙니다.

태스크를 위한 인공지능 학습용 데이터를 마련하고 라벨을 붙여야 하고,

최적의 모델을 만들기 위해서는 다양한 하이퍼파라미터 설정을 조합하여 실험을 반복해야 합니다.

이렇게 만들어진 모델이 새로운 데이터에 대해서도 좋은 성능을 보이는지 확인하고, 개선해나가는 것도 중요하지요.

매번 아키텍처를 고민하고 학습하는 것은 힘들기에, 지난시간에 Transfer Learning(전이학습)에 대해 소개하였습니다.

Transfer Learning은 한 번 만든 딥러닝 모델을 다른 유사 태스크에 바꾸어 재활용하는 방법입니다.

오늘은 미리 준비된 인공지능 모델, 인공지능 사전학습(Pre-training)에 대해 알아보겠습니다.

지난 시간까지의 내용이 궁금하신 분은 ★[AI Tech Letter](#)(see page 7)★를 확인하시기 바랍니다.

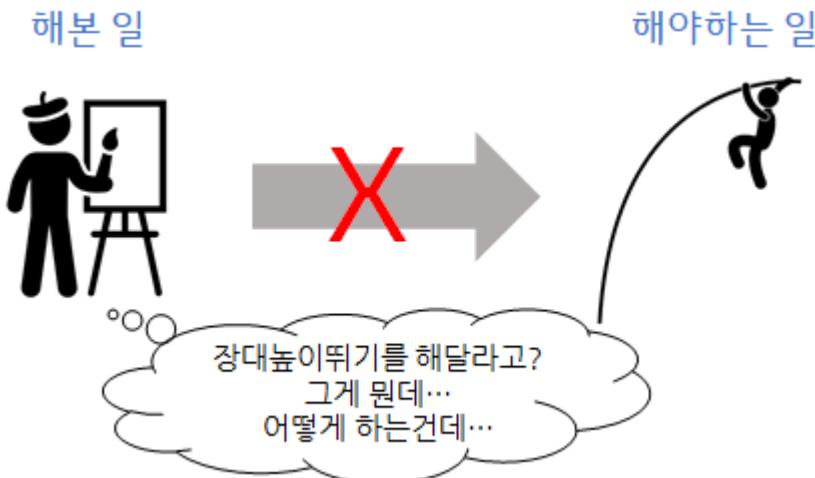
⁴⁵<https://wire.lgcns.com/confluence/display/~78628>

8.1 Pre-training: 니가 뭘 요청할지 몰라서 일단 다 공부해놨어

Transfer Learning을 써보려고 해도, 비슷한 태스크에 대해 만들어놓은 딥러닝 모델이 없다면 재활용을 할 수 없습니다.

전혀 다른 것에 대해 학습한 모델이라면 Transfer Learning을 해도 효과가 없을 가능성이 매우 큽니다.

웹툰 작가에게 갑자기 장대높이뛰기 경기에 나가달라고 요구하는 것과 같이, 모델간 사전지식을 잘 활용하기 어렵겠죠.



[그림 1. 예체능이라고 다 같은 계열이 아니듯이 AI라고 해서 두루두루 다 되는 건 아닙니다.]

그렇다면 지식을 전수하는 Transfer Learning은 적용할 수 있는 경우가 많지 않겠는데요?

Transfer Learning은 아주 유사한, 한정된 태스크끼리만 가능한 걸까요?

전혀 다른 태스크간에는 지식을 전수하기 쉽지 않습니다.

하지만 전이 학습을 염두에 두고 다방면으로 활용할 수 있는 모델을 미리 만들어놓을 수 있습니다.

여러 태스크에 활용하기 위해 여러 지식을 미리 두루두루 학습해놓은 인공지능을 만드는 것이지요.

이러한 학습을 **Pre-training**, 또는 **사전학습**이라고 하며 이러한 용도의 모델을 **Pre-trained Model**, 즉 **사전학습 모델**이라고 합니다.

사전학습은 보통 특정 데이터타입에 대한 일반적인 지식을 두루 배워놓는 것을 목표로 합니다.

텍스트 사전학습 모델은 언어의 일반적인 의미와 구조에 대해, 이미지 사전학습 모델은 이미지의 일반적인 특징, 색채, 형태 등에 대해 배웁니다.

그래서 향후 어떤 텍스트나 이미지 관련 태스크를 수행한다 해도 기본 지식을 바탕으로 잘 적용할 수 있습니다.

예를 들어 기본적인 웨이트 트레이닝, 유산소 운동, 스트레칭 등을 통해 몸 쓰는 법을 배웠다면 어떤 운동 종목이든 적응이 수월한 것처럼요.