**Pusan National University Computer Science and Engineering Technical Report 2022-0X**

**수요예측 알고리즘 경량화 및 서비스 구현**



저자1 서준호

저자2 윤정현

저자3 김민태

지도교수 박진선 교수님

목 차

[1. 서론 1](#_Toc115419178)

[1.1. 연구 배경 1](#_Toc115419179)

[1.2. 기존 문제점 1](#_Toc115419180)

[1.3. 연구 목표 2](#_Toc115419181)

[2. 연구 배경 2](#_Toc115419182)

[2.1. 알고리즘 개발 환경 2](#_Toc115419183)

[3. 연구 내용 3](#_Toc115419184)

[3.1. 데이터 수집 및 전처리 3](#_Toc115419185)

[3.1.1. 데이터의 주기성 3](#_Toc115419186)

[3.1.2. 여러 종류의 데이터 추가하기 5](#_Toc115419187)

[3.1.3. 데이터 나누기 7](#_Toc115419188)

[3.2. 수요 예측 알고리즘 7](#_Toc115419189)

[3.2.1. LSTM 8](#_Toc115419190)

[3.2.2. Convolutional neural network 8](#_Toc115419191)

[3.2.3. 주식 가격 예측에서 회귀 모델의 한계점 11](#_Toc115419192)

[3.3. 분류 모델과 Transformer 12](#_Toc115419193)

[3.3.1. Time2Vec 13](#_Toc115419194)

[3.3.2. Transformer Encoder 14](#_Toc115419195)

[3.4. Knowledge Distillation와 경량화 16](#_Toc115419196)

[3.5. 최종 모델 구성 16](#_Toc115419197)

[3.6. React-query 17](#_Toc115419198)

[3.6.1. Client State 17](#_Toc115419199)

[3.6.2. react-query 상태 18](#_Toc115419200)

[3.6.3. React query가 해결해주는 문제 18](#_Toc115419201)

[3.7. 네이버 검색어 트렌드 API 19](#_Toc115419202)

[3.7.1. 과제 도중 발생한 문제 1 20](#_Toc115419203)

[3.7.2. 과제 도중 발생한 문제 2 21](#_Toc115419204)

[4. 연구 결과 분석 및 평가 22](#_Toc115419205)

[4.1. 모델 간 성능 및 비용 비교 22](#_Toc115419206)

[5. 결론 및 향후 연구 방향 23](#_Toc115419207)

[5.1. 알고리즘 구현 및 경량화 23](#_Toc115419208)

[5.2. 웹 서비스 구현 24](#_Toc115419209)

[6. 참고 문헌 24](#_Toc115419210)

# 서론

## 연구 배경



그림 AI와 주식 [1]

2010년 후반기부터 가장 빠르게 발전한 IT 분야 중 하나는 AI 라고 할 수 있다. AI는 현재 모든 분야에서 각광받는 기술이다. 그 중에서도 금융, 특히 주식 시장은 다른 어느 영역에서 보다 AI 기술을 빠르게 받아드리고 있다. 주식 시장에서 인공지능을 활용하려는 가장 큰 이유는 깊게 생각할 필요도 없이 바로 미래의 주가에 대한 예측일 것이다.

주식은 그 자체로 수요에 대한 데이터라고 말할 수 있다. 주식 가격을 결정하는 가장 기본적인 요인은 수요와 공급이고, 그 균형점에서 주식 가격이 형성된다.

따라서, 주식 가격 데이터를 AI에게 학습시켜 미래의 주식 가격을 예측하는 연구를 진행하고자 하였다.

## 기존 문제점

기존의 수요 예측 알고리즘들은 매우 큰 데이터 셋을 학습하는 것을 전제로 개발되어 상당히 복잡하고 거대하기 때문에 상용화하기에는 학습 비용이 많이 들었다.

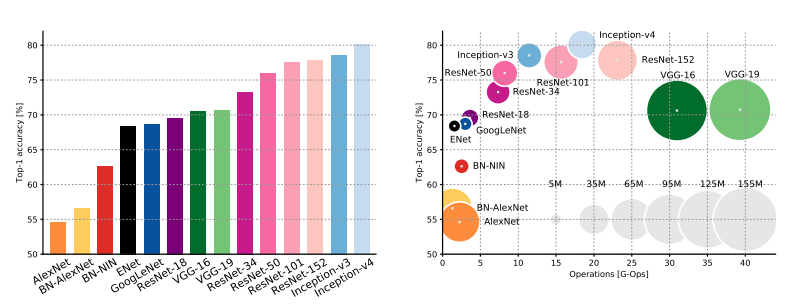


그림 CNN 기반 딥 러닝 모델들의 Parameter 수에 따른 Top-1 Accuracy 분석 차트 [2]

그리고 최근의 IoT 기기들이나 모바일 기기들 같이 상대적으로 저 성능의 디바이스들에게도 딥 러닝 모델을 탑재하려는 움직임들이 늘어나고 있다.

따라서, 이를 경량화 하여 적절한 모델 크기를 가지고 일정 수준의 정확도를 보장하기 위한 경량화를 진행하는 것을 연구의 핵심 목표로 둘 것이다.

## 연구 목표

크게 3가지 연구 목표를 가지고 진행하였다.

* 정해진 데이터 셋에 맞는 수요 예측 알고리즘을 조사 및 구현
* 알고리즘 경량화
* 웹 서비스 기획 및 Back-end/Front-end 구현

# 연구 배경

## 알고리즘 개발 환경

알고리즘 개발을 위해 다음과 같은 라이브러리들을 사용하였다.

* **Numpy**: Python에서의 다양한 수학/과학적 연산과 행렬 같은 다차원 데이터를 쉽게 처리할 수 있도록 지원하는 라이브러리이다.
* **Pandas**: 데이터 조작 및 분석을 위한 오픈소스 라이브러리이다. DataFrame이라는 강력한 자료구조를 제공한다.
* **Matplotlib**: 데이터 시각화를 위한 라이브러리이다.
* **FinanceDataReader**: 금융 데이터를 읽어올 수 있는 오픈소스 라이브러리이다. 코드 한 줄로 원하는 주식 종목의 데이터를 불러오고, 시각화를 할 수 있다. 불러온 데이터는 Pandas가 제공하는 DataFrame 자료구조를 사용한다.
* **TensorFlow**: 엔드 투 엔드 머신 러닝 플랫폼으로써, Keras라는 강력한 딥 러닝 머신 개발용 라이브러리를 제공하고, 그 외에 머신 러닝 개발에 필요한 다양한 기능을 제공한다. Keras를 통해 별도의 구현없이 다양한 Layer들을 사용할 수 있다.

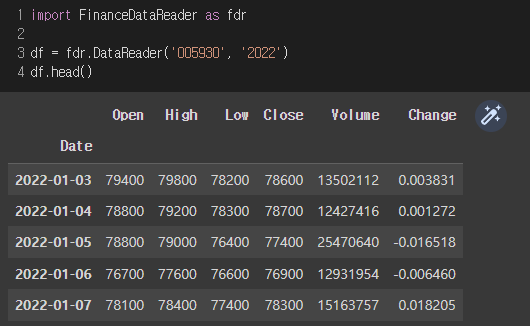


그림 FinanceDataReader를 이용한 주식 데이터 불러오기

# 연구 내용

## 데이터 수집 및 전처리

### 데이터의 주기성

FinanceDataReader를 통해 불러온 주식 데이터를 Matplotlib을 통해 시각화 하면 다음과 같은 형태를 띈다.

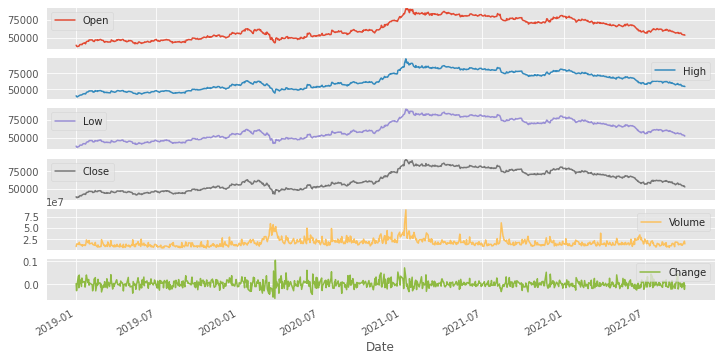


그림 4 FinanceDataReader를 통해 불러온 삼성전자 2019년 ~ 2022년7월 데이터

현재 불러온 데이터는 삼성전자(005930)의 2019년 1월부터 2022년 9월 29일 까지의 주식 가격 데이터이다. 그래프를 보면 일정하지 않은 흐름으로 주식 가격이 시간에 따라 변화하는 것을 확인할 수 있다. 시계열 데이터에 대한 예측 모델을 만들고 싶을 때는 이 데이터가 최대한 주기성을 갖는 것이 매우 중요하다. 이를 위해 일별 가격 데이터가 아닌, 일별 가격 변화량으로 바꿔보았다.

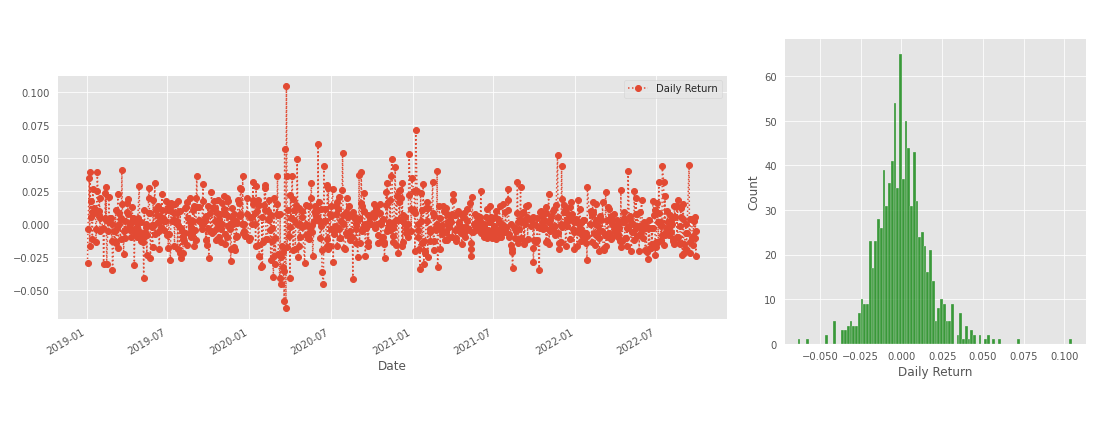


그림 종가 변화량의 점도표 (왼쪽)와 Histogram 그래프 (오른쪽)

이전 그림 보다 조금 더 주기적이고 정규화 된 그래프가 나왔다. 조금 더 정확한 비교를 위해 Dickey-Fuller Test를 통해 데이터의 주기성을 확인하였다.

|  |  |  |
| --- | --- | --- |
|  | Close(종가) | Change(종가 변화량) |
| Test statistic | -1.653828 | -29.870081 |
| p-value | 0.4550446 | 0.000000 |
| Critical Value (1%) | -3.437446 | -3.437446 |
| Critical Value (5%) | -2.864673 | -2.864673 |
| Critical Value (10%) | -2.568438 | -2.568438 |

표 종가와 종가 변화량 데이터에 대한 Dickey-Fuller Test 결과

종가 데이터보다 종가 변화량 데이터가 Test Statistics나 p-value 모두 훨씬 낮게 나온 것을 확인할 수 있다. 따라서 Close 데이터처럼 raw 데이터를 그대로 사용하기보다, 일별 변화량으로 데이터를 가공하여 사용하는 것이 학습에 더 도움을 줄 것이라는 결론이 나왔다.

### 여러 종류의 데이터 추가하기

현재의 주식 가격을 결정하는데 영향을 미치는 데이터에는 다양한 경제 지표들이 있다. 그 중에서도 우리나라의 경기를 직접적으로 나타내는 대표적인 지표는 경기선행지수인데, 그 중에서 동행지수순환변동치와 선행지수순환변동치를 추가하였다.

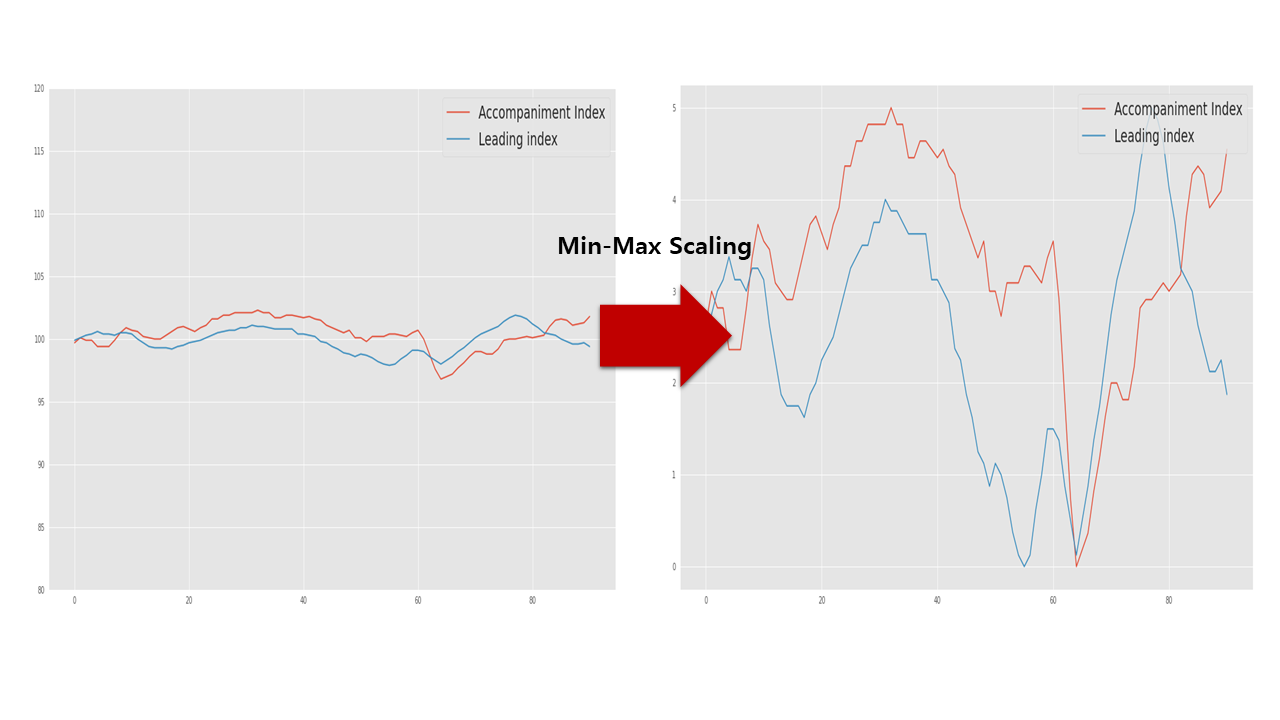


그림 경기선행지수와 Min-Max Scaling

그림의 왼쪽 그래프는 경기선행지수의 Raw Data를 나타낸다. 이 상태 그대로 학습에 사용되기에는 너무 값들의 격차가 작다. 따라서, Min-Max Scaling을 통해 데이터의 스케일을 재조정해주었고, 종가의 변화량 데이터와의 스케일을 맞추기 위해 5를 다시 곱하여서 오른쪽의 그래프와 같이 데이터를 처리하였다.

그리고 미국연방준비은행(FRB)에서 운영하는 데이터 제공 사이트 FRED에서는 다양한 미국의 경제 지표, 선행 지표, 후행 지표 등을 제공한다. 이 중에서 사용자들이 가장 많이 본 데이터들 위주로 추가하였다.

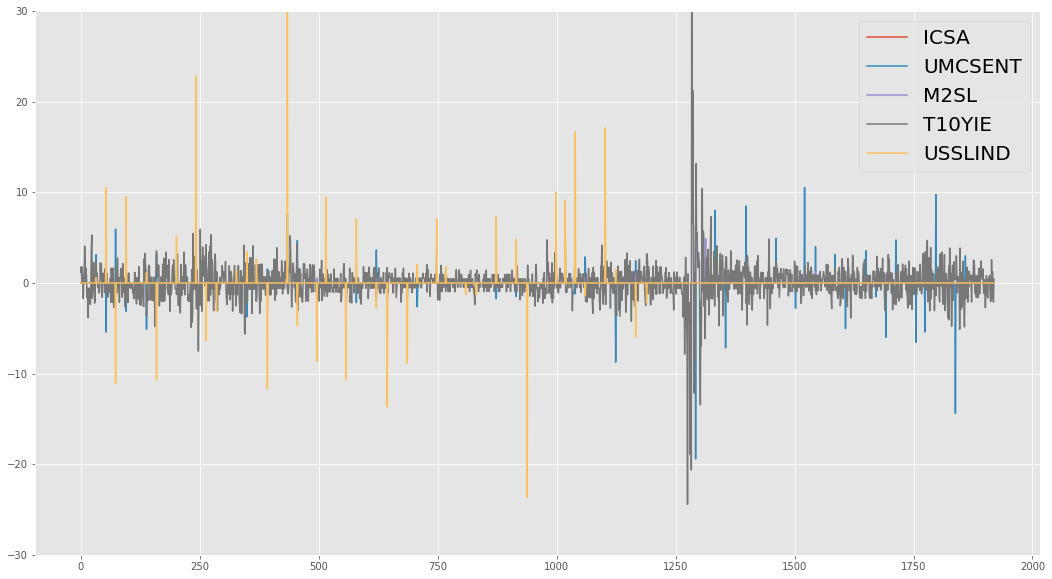


그림 미국 경제 선행 지수

테이블이(가) 표시된 사진

자동 생성된 설명

표 총 데이터의 column과 각각의 통계 수치

### 데이터 나누기

3.1.2에서 완성한 데이터를 Sliding Window 알고리즘으로 일정하게 나눌 것이다. 연구에서는 30일 간격으로 데이터를 나누고, 나눈 데이터의 다음 날 종가를 정답 데이터로 만들었다. Sliding Window를 통해 데이터 하나가 하루를 나타내는 것이 아닌, 데이터 하나를 충분한 시간 간격을 데이터로 만들어 모델이 시계열 데이터의 학습을 더 잘 할 수 있게 만들었다. 연구에서는 시간 간격(Window Size)을 30일로 두었다. Sliding Window를 진행할 때, 학습에 거의 영향을 주지 않거나, 도움이 되지 않는 feature들(Open, High, Low)은 제외하였다.

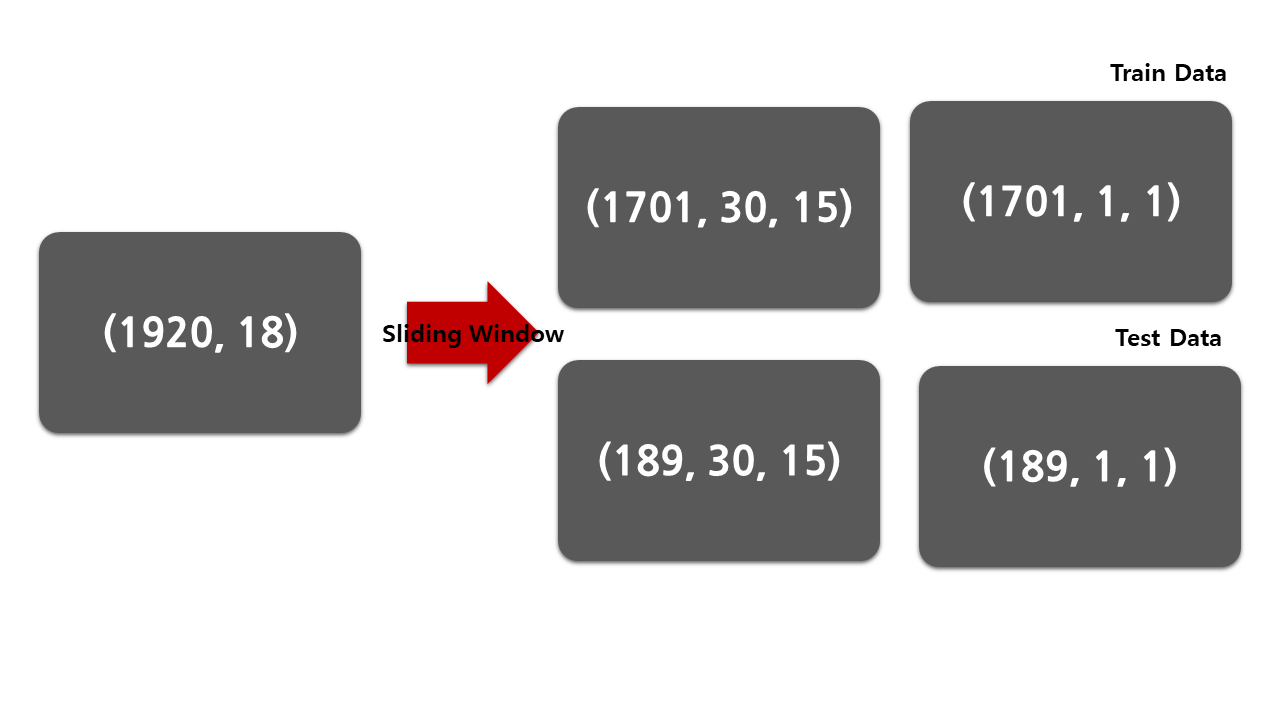


그림 나누기 전과 나눈 후의 데이터 shape

Sliding Window를 진행하고 난 후, 학습용과 테스트용으로 데이터를 따로 나누었는데, 테스트용은 학습용의 10% 정도 크기로 구성하였다.

## 수요 예측 알고리즘

주식 데이터는 시계열 데이터이다. 현재 사용되는 딥러닝 모델 중, 시계열 데이터를 다루는 Sequence model에는 대표적으로 RNN, GRU, LSTM 등이 있다.

### LSTM

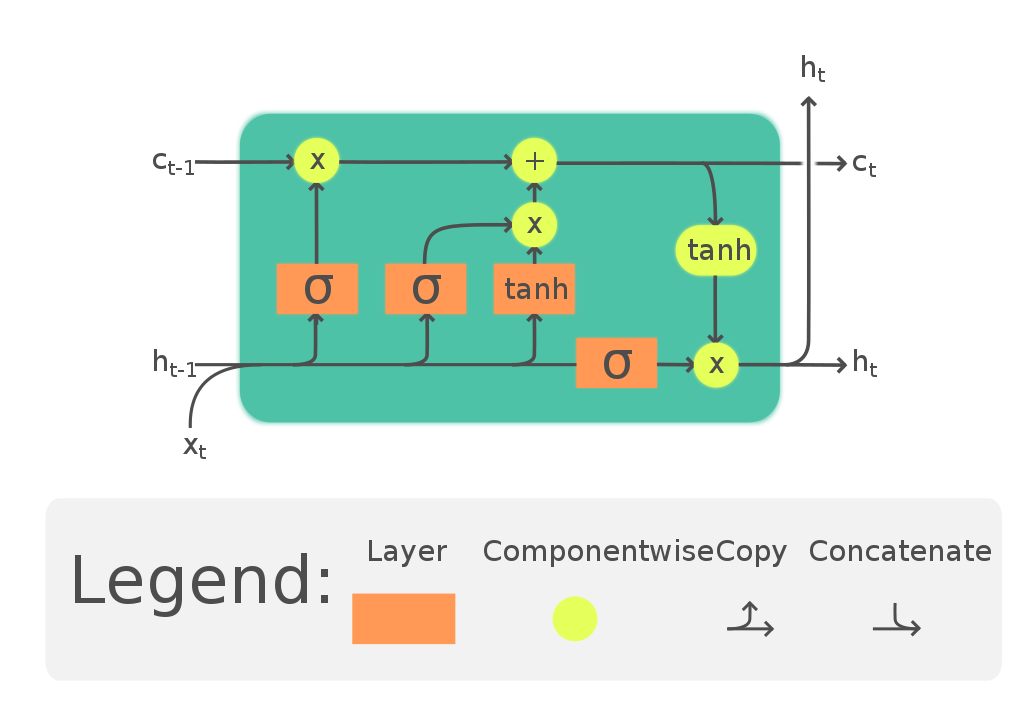


그림 Long short-term Memory (LSTM)의 구조 [3]

많은 sequential model들 중에서 LSTM을 활용하고자 하였다. 그 이유는 RNN에는 Long-term dependencies가 없다는 문제가 있기 때문이다. 즉, 특정 데이터의 위치가 출력과 멀어지면 멀어질 수록 예측에 반영되기가 힘들다. 주식 데이터는 상당히 시간 간격이 긴 정보들을 포함하고 있기 때문에, 꽤 먼 과거의 데이터도 어느정도는 현재의 가격과 연관이 있다.

LSTM과 RNN의 차이점 중 하나는 Cell State의 유무이다. RNN과 달리 LSTM은 Cell State라는 또 다른 상태가 있고, 매 time step 마다 forget gate를 통해 이전의 cell state에서 얼마나 보존할 지 결정한다. 따라서 이를 해결하기 위해 Long Short-Term Memory(LSTM) 모델을 사용하기로 하였다.

### Convolutional neural network

현재 모델의 입력으로 들어가는 데이터는 시계열 데이터이다. 주식 데이터에서 사용할 정보의 종류 개수를 라고 하면 크기의 행렬 형태로 들어오게 된다.

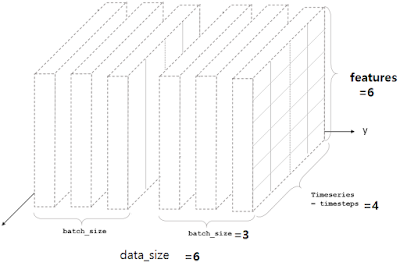


그림 LSTM 입력 데이터의 예 [4]

이 데이터를 바로 LSTM에 넣어서 학습시키는 것 보다, 유용한 정보만 뽑아서 학습을 시키면 정확도가 더 올라갈 것이라는 생각을 하였다. 그래서 우리는 모델의 위층에 Convolutional Layer를 추가하여 이 문제를 해결하고자 하였다.

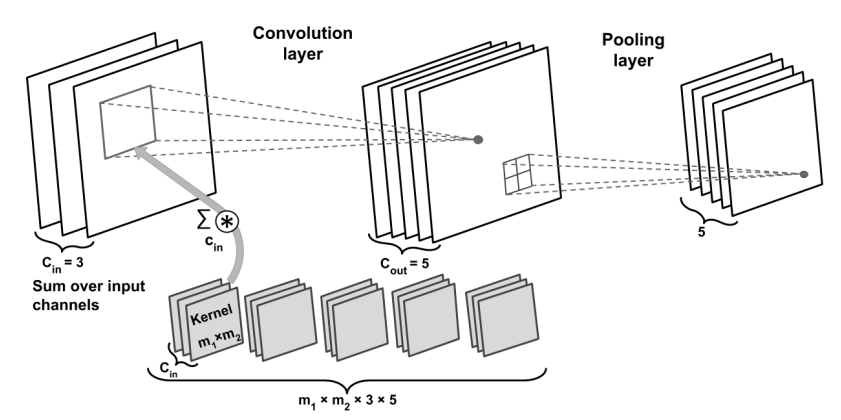


그림 Convolutional neural network의 구조

Convolutional layer에서는 raw data와 filter(kernel)을 convolution 연산하여 데이터들의 상관관계를 뽑아낸다. 과제에서는 주식 데이터들의 시간을 기준으로 정보를 추출하기 위해 raw data를 작은 window size로 묶은 후, 행렬의 형태로 만든다. 이렇게 처리한 데이터는 각 feature들이 합성된 형태의 matrix로 나타내어진다. 이를 여러 번 반복하여 raw data보다 일반적으로 더 유용한 데이터들 만이 추출된다.

Convolutional layer를 통과한 데이터는 pooling layer를 통해 feature들을 더 낮은 차원의 행렬로 만들어준다. 이 과정을 통해 데이터들을 조금 더 밀착시킬 수 있고 이는 outlier data의 영향을 더 적게 하여 모델의 성능을 끌어올릴 수 있다.

마지막으로 Flatten Layer를 추가해 고차원의 데이터들을 1차원으로 바꿔준다.

Flatten Layer의 결과로 나오는 데이터들은 시계열 데이터가 아니기 때문에, LSTM Layer의 입력으로 넣을 수 없다. 따라서, 이를 해결하기 위해 Convolutional Layer를 TensorFlow의 Time Distributed Layer로 감싸서 이 문제를 해결하였다.

테이블이(가) 표시된 사진

자동 생성된 설명

그림 완성된 CNN-LSTM의 전체 Layer 구성

### 주식 가격 예측에서 회귀 모델의 한계점

연구 중반까지 이 모델을 통해 서비스를 구현하려고 했으나, 문제점이 발생하였다.



그림 CNN-LSM 모델의 테스트 결과 그래프

그림 7번은 해당 모델로 학습을 시킨 후, Test data로 예측을 진행한 그림이다. 여기서 보라색 선은 실제 가격이고, 검은색 선은 예측 가격이다. 검은 색 선이 보라색 선을 하루만큼 옆으로 이동한 것과 같은 모습인데, 이는 AI에게 내일 주식 가격을 예측하게 하였는데, 오늘 가격과 동일하다고 답하는 것과 같다.

이런 현상이 일어나는 가장 큰 원인은, 손실 함수 때문이라고 추정된다. 그림 8은 해당 학습에서의 손실 정도를 그래프화 한 것이다.

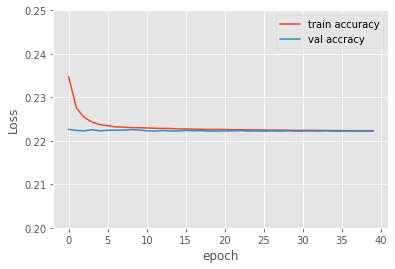
 이때 사용한 손실 함수는 Mean Squared Error이다. 그림 8을 보면, 손실 값이 학습 횟수에 상관없이 일정한 것을 알 수 있다. 즉, 매 학습마다 손실이 가장 적은 경우는 내일 주식가격과 오늘 주식 가격이 똑같은 경우이다. 그래서 학습이 전혀 진행되지 않고 일정한 손실 정도가 나타나는 모습이다.

그림 CNN-LSTM 모델의 학습 Loss그래프

## 분류 모델과 Transformer

3.2의 문제점에 대한 해결책으로 연속적인 가격이 아니라, 내일 주식 가격이 오를 지 안 오를 지만을 예측하는 분류 모델을 사용하고자 하였다.

회귀 분석에서 통계적 분류로 문제가 바뀌었기 때문에, 정답 데이터를 인코딩 해줘야 한다. 원래 자연어 학습에서는 원-핫 인코딩을 통해 정답 데이터를 원-핫 벡터로 표현한다. 우리가 학습시켜야 할 데이터는 주식 데이터이고, 이는 실수 데이터이기 때문에, 단순하게 종가의 변화율이 5% 이상인 경우, -5% 이하인 경우, -5% ~ 5%인 경우 3가지로 나누고 정답 데이터를 이에 맞게 바꾸었다.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Date | Close | Change | Down | Place | Up |
| 2015-01-02 | 26600 | -0.811808 | 1.0 | 0.0 | 0.0 |
| 2015-01-05 | 26660 | 1.264881 | 0.0 | 0.0 | 1.0 |
| 2015-01-06 | 25900 | 0.955180 | 0.0 | 0.0 | 1.0 |
| 2015-01-07 | 26140 | 0.218341 | 0.0 | 1.0 | 0.0 |
| 2015-01-08 | 26280 | -0.726216 | 1.0 | 0.0 | 0.0 |

표 라벨링한 데이터의 예시

Labeling된 정답 데이터와 함께 주식 데이터를 학습시키기 위하여 예측 모델을 CNN-LSTM에서 Transformer로 바꾸었다.

Transformer는 2017년 구글에서 발표한 논문(Ashish, et al. [5])에서 나온 모델이다. 이 모델은 자연어 학습 및 분류에 주로 사용되지만, 주식 데이터를 학습할 수 있게 바꾸어 사용하고자 하였다.

연구에서 사용된 Transformer 모델의 경우 크게 3가지 layer 층으로 구성된다.

1. Time2Vec
2. Encoder
3. Global Average Pooling / Dense

### Time2Vec

Transformer의 정답 데이터는 준비가 되었지만, 입력 데이터도 따로 인코딩 과정을 거쳐야 한다. 시계열 데이터를 처리할 때 시간이 중요한 feature이지만, Transformer를 통해 처리해야할 때는 이야기가 다르다. Transformer는 LSTM처럼 순차적으로 입력을 받는 방식이 아니기 때문에, 시간 정보를 다른 방식으로 전달해야 한다. 자연어를 입력어로 받는 Transformer의 경우 Positional Encoding 기법을 사용하여 문장 내 단어들의 위치 정보를 벡터화 하여 입력 데이터에 embedding하는 방식을 사용한다.

시계열 데이터의 경우 단어의 위치가 아니라 시간 정보를 추출해야 한다. 이를 위해 **Time2Vec**이라는 인코딩 방식을 사용하였다. (Kazemi, et al. [6]) Time2Vec에서는 입력 데이터가 주기적 패턴과 비주기적 패턴이 포함되어 있다고 가정한다. 그리고 시간의 스케일을 조절해도 그 패턴이 변하지 않는다고 가정한다.

논문에서는 시간 벡터를 다음과 같이 표현한다.

은 시간 벡터의 비주기적 feature를 나타내고, 나머지는 주기적인 feature를 나타낸다. 해당 식을 보면, 선형적인 특징을 가지고 있는 것을 알 수 있는데, 이는 비 주기적인 데이터라도 선형인 경향이 있다고 가정하는 것이다. 그리고 는 이런 비주기적 특징을 나타내는 데이터를 주기적으로 바꿔주는 함수이다. Kazemi, et al. [6] 에서는 sine 함수를 사용하였을 때, Time2Vec을 사용하는 모델들의 performance가 가장 좋았다. 따라서 본 연구에서도 sine 함수를 사용하였다.

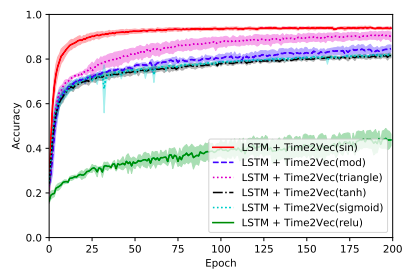


그림 Comparing different activation functions for Time2Vec on Event-MNIS [6]

### Transformer Encoder

Transformer Encoder는 다음과 같이 구성하였다. [7]

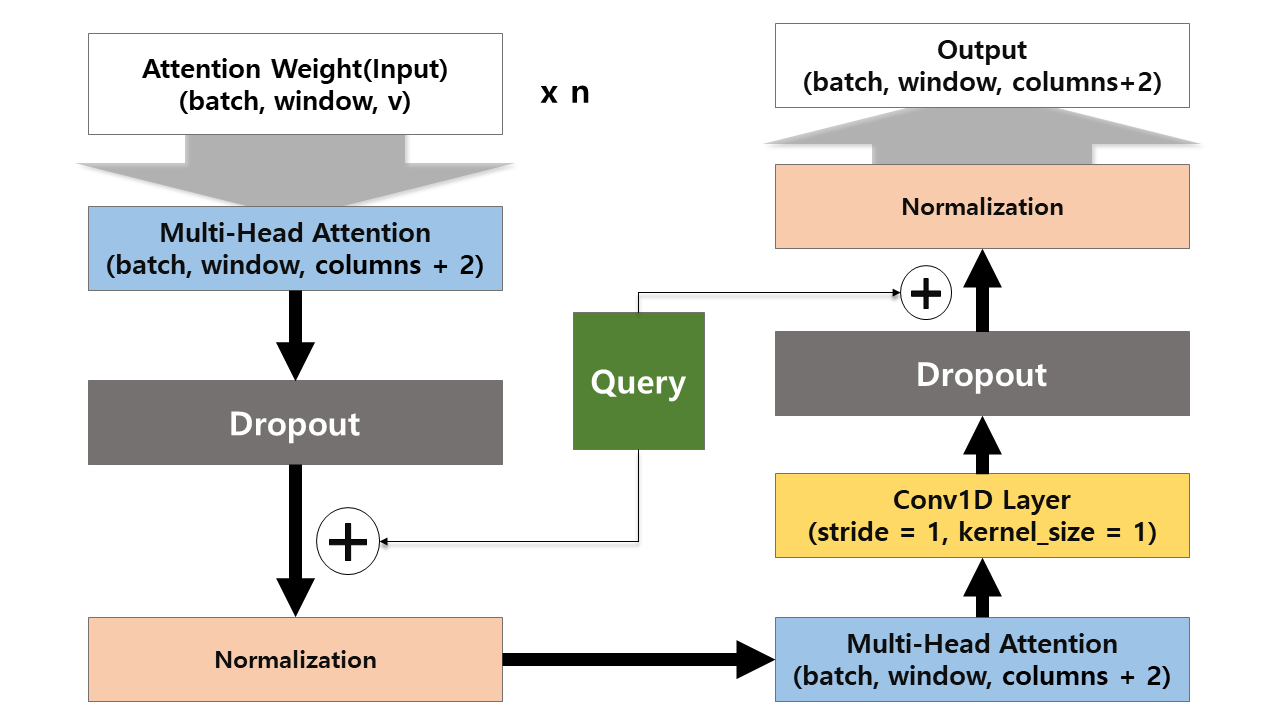


그림 Transformer Encoder내부의 Layer 구성 도식

먼저, Multi-head Attention Layer는 n개의 Single-head Attention Layer에서 나온 Attention weights가 Dense Layer에 통과하면서 진행된다. 그 후, Dropout Layer를 통과할 때, Query 데이터와 residual connection을 진행함으로써, 기존에 학습한 정보를 보존하고 추가적으로 학습하는 정보를 합치는 구조로 구성하였다. Query 데이터는 Attention weights의 첫번째 데이터로, 종가의 변화량에 대한 정보를 가지고 있다. 이어서 Convolution 1D Layer를 지나는데, 이때 stride와 kernel size를 모두 1로 설정하여서 사실상 dense layer와 역할이 동일하다.

Time2Vec과 Transformer encoder layer로 구성된 전체적인 모델 구조는 다음과 같다.

테이블이(가) 표시된 사진

자동 생성된 설명

그림 Transformer 모델의 전체 Layer 구성

Time2Vec을 통해 transformer encoder의 Input으로 들어갈 시간 정보를 embedding한 후, transformer encoder layer를 총 3번 지나가게 된다. 그리고 생성된 모든 feature를 종합하여 1차원 벡터로 만들기 위해 Global average pooling을 진행하였다. 이후, Dense Layer를 통해 최종적으로 Down, Place, Up 총 3가지의 결과를 도출할 수 있도록 구성하였다.

## Knowledge Distillation와 경량화

Knowledge distillation의 목적은 미리 잘 학습된 큰 네트워크인 Teacher network의 정보를, 실제 배포하고자 하는 모델인 Student network에 전달하는 것이다. Teacher network의 결과를 Student network에 전달하고, Student network는 전달받은 정보를 모방하도록 학습시킨다. 결국, 빠르지만 성능은 낮은 Student의 정확도를 Teacher만큼의 큰 비용을 들이지 않고 올릴 수 있다.

Loss Function의 parameter를 조절하여 Teacher network로부터의 정보를 얼마나 받아드릴 건지 결정할 수 있다. Loss Function의 수식은 다음과 같다. [8]

|  |  |
| --- | --- |
| Parameter | Description |
|  | SoftMax function |
|  | Categorical cross-entropy loss function |
|  | KL divergence |
|  | Temperature |
|  | Balancing parameter |

표 Loss Function의 Parameters

이때 는 SoftMax function이고, 는 SoftMax function의 temperature이다. 는 Categorical cross-entropy loss function을 사용하였다. 해당 수식에서 를 통해 Student와 Teacher 모델 간의 학습 distillation 정도를 조절할 수 있고, 를 통해 SoftMax의 output scale을 조절할 수 있다.

결과로 나온 Loss 값으로 optimizer를 통해 Student의 Weight를 업데이트 시켜주면 train step 1회를 마치게 되는 것이다.

## 최종 모델 구성

최종적으로 다음의 도식으로 전체적인 수요 예측 모델을 구성하였다.

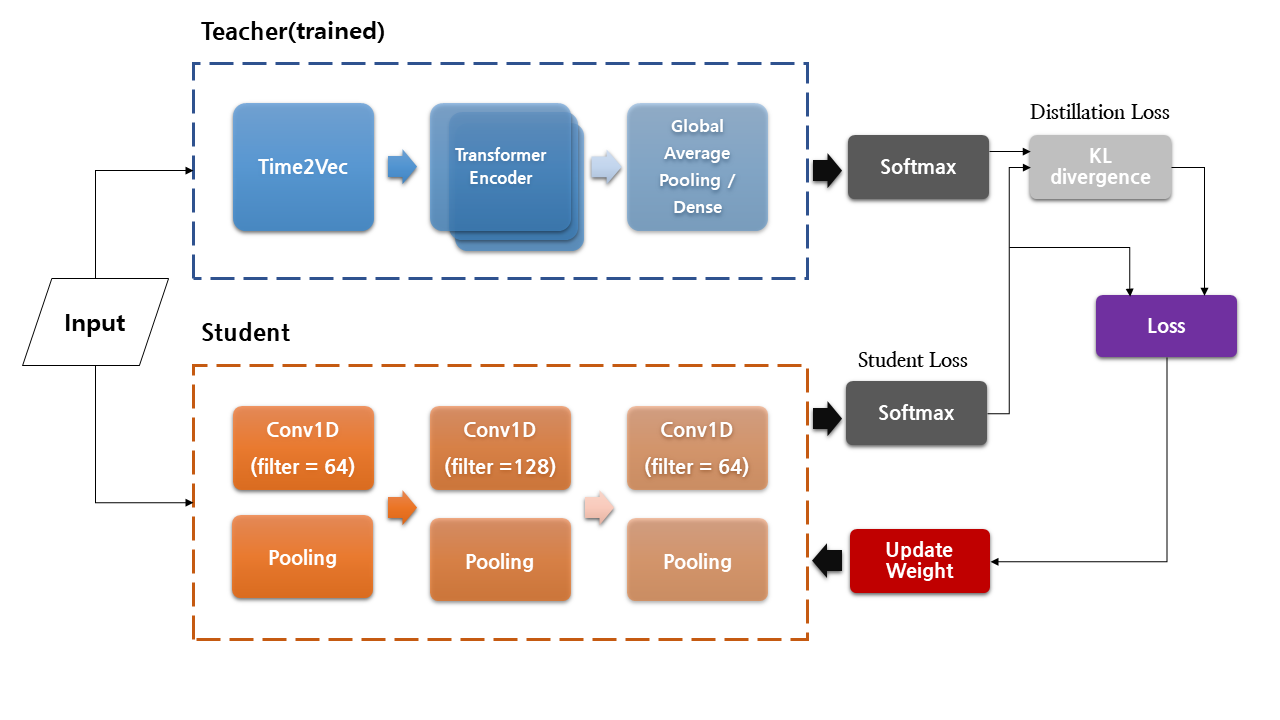


그림 최종 모델 구성 도식

## React-query

리액트 쿼리는 서버에서 가져온 데이터를 웹 브라우저 앱에서 사용하기 쉽게 도와주는 기술이다. 서버는 클라이언트에게 데이터베이스에 있는 정보를 전달해주는 역할을 하는데, 여기서 서버는 API 서버, 클라이언트는 웹 브라우저에서 실행되는 앱을 의미한다. 데이터베이스에서 가져온 데이터를 클라이언트에 보여주기 위해 우리는 ajax를 이용하는데, 이때 서버에서 가져오는 데이터를 server state라고 이야기한다.

서버는 특정 시점에 클라이언트의 요청에 대해 데이터베이스에서 유저 정보를 가져와 서버의 상태 값을 만들어낸다. 데이터베이스에 있는 값을 그대로 클라이언트에게 전달할 수도 있고, 요청에 담긴 특정 값을 이용해 정보를 가공해서 메모리에 들고 있다. 그리고 이 정보를 클라이언트에게 전달해준다.

### Client State

Client state는 크게 두가지로 나눌 수 있다.

1. Client에서 자체적으로 만드는 state(최초 데이터의 발생지가 클라이언트)
2. Server에서 전달받은 값으로 만드는 state (최초 데이터의 발생지가 서버)

첫번째인 클라이언트가 자체적으로 만드는 state는 대게 UI를 담당하는 부분으로 모달이 열리고 닫았는지, 어떤 버튼이 클릭되었는지, 지금 창이 리사이징 되고 있는지에 대한 메타 정보를 담은 상태 값이다.

두번째는 server state -> client state로 가져오는 부분에 대한 내용이다. 리액트는 서버의 상태 값을 받아오는데 정해진 모범 답안이라는 것이 없다.

컴포넌트의 생명 주기를 파악한 후 적절한 시점에 ajax 호출을 하고 서버에서 데이터를 받아온다. 그리고 useState를 사용할 경우 데이터를 불러와 setState호출을 통해 응답 당시의 server state를 component state로 wrapping한다.

### react-query 상태

fresh: 새롭게 추가된 쿼리 & 만료되지 않은 쿼리 ➜ 컴포넌트가 마운트, 업데이트되어도 데이터 재요청 ❌

fetching: 요청 중인 쿼리

stale: 만료된 쿼리 ➜ 컴포넌트가 마운트, 업데이트되면 데이터 재요청 ⭕

inactive: 비활성화된 쿼리 ➜ 특정 시간이 지나면 가비지 컬렉터에 의해 제거

### React query가 해결해주는 문제

* 간편한 server state 수급 방식

앱이 간단하다면 useState와 contextAPI만 사용하더라도 대부분의 client state를 다룰 수 있다. 하지만 복잡도가 올라가고 성능 향상에 대한 필요성이 생기는 시점이 예상된다면, 이미 잘 만들어진 상태 관리 도구를 선택하는게 합리적일 지도 모른다. React query는 hook 기반의 로직들로 되어있어 해당 훅을 사용하는 컴포넌트에서 상태 값의 변경을 간편하게 파악하여 리렌더링을 유발하게 해준다.

* 캐시

React query를 사용하기 전까지 state라는 용어는 암묵적으로 client의 state를 가리켰다. React query가 server state라는 개념을 만든 것이 아니라 client state만 신경쓰기에도 프론트 엔드에서 해줘야 할 작업들이 많았는데, 만약 10초 뒤에 앱에서 표현하는 데이터가 더 이상 유효하지 않은 낡은 데이터라고 한다면(주식, 코인)클라이언트에서는 polling방식으로 주기적으로 데이터를 받아오거나 (setInterval) 실시간성이 중요한 데이터라면 웹 소켓을 통해 서버의 상태 값이 변경되었을 때 서버에게 통지받아야 한다. 그런데 react query는 데이터 캐시 처리를 간편하게 할 수 있는 인터페이스를 제공한다.

1. 몇 초 이후에는 데이터가 유효하지 않은 것으로 간주하고 데이터를 다시 불러온다.
2. 데이터에 변경점이 있는 경우에만 리렌더링을 유발한다.
3. 유저가 탭을 이동했다가 다시 돌아왔을 때 데이터를 다시 불러온다.
4. 데이터를 다시 호출할 때 응답이 오기 전까지는 이전 데이터를 계속 보여준다. 필요에 따라서는 로딩바와 같은 대안 UI를 보여주기 위해 loading state를 기본적으로 제공한다.

텍스트, 스크린샷, 화면, 은색이(가) 표시된 사진

자동 생성된 설명

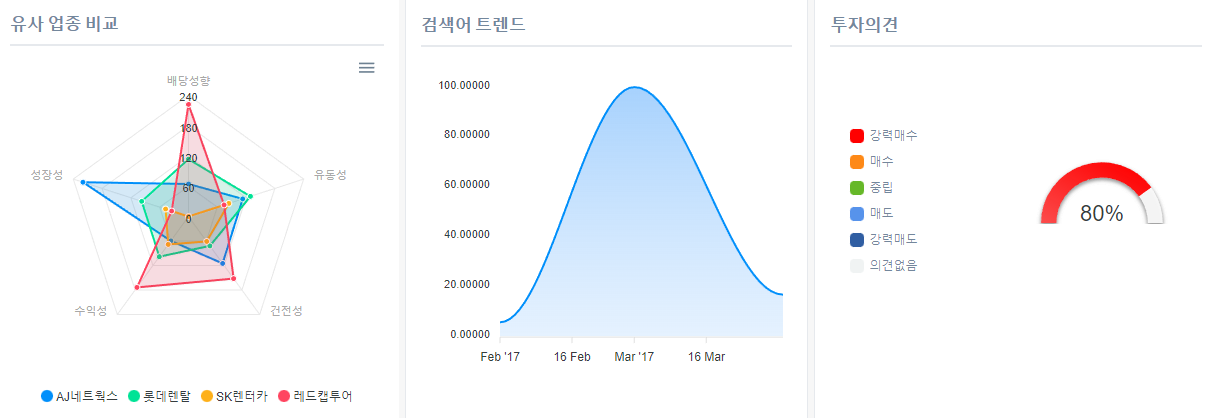
Figure 1react query를 이용한 데이터 패칭 결과

## 네이버 검색어 트렌드 API

주가에 있어 외부요인 또한 중요하다. 따라서 네이버에서 제공하는 Open API를 사용하였다.

통합 검색어 트렌드 API는 네이버 데이터 랩의 검색어 트렌드를 API로 실행할 수 있게 하는 RESTful API이다. 주제어로 묶은 검색어들에 대한 네이버 통합검색에서의 검색 추이 데이터를 JSON형식으로 반환한다. API를 호출할 때는 주제어와 검색어, 검색조건을 JSON 형식의 데이터로 전달한다. 통합 검색어 트렌드 API는 비로그인 방식 오픈 API이다.비로그인 방식 Open API는 네이버 Open API를 호출할 때 HTTP 요청 헤더에 클라이언트 아이디와 클라이언트 시크릿 값만 전송해 사용할 수 있는 오픈 API이다.

|  |  |  |
| --- | --- | --- |
| Parameter | 타입 | 설명 |
| startDate | string | 조회 기간 시작 날짜(yyyy-mm-dd 형식). 2016년 1월 1일부터 조회할 수 있습니다. |
| endDate | string | 조회 기간 종료 날짜(yyyy-mm-dd 형식) |
| timeUnit | string | 구간단위 - date: 일간 - week: 주간 - month: 월간 |
| keywordGroups | array (JSON) | 주제어와 주제어에 해당하는 검색어 묶음 쌍의 배열. 최대 5개의 쌍을 배열로 설정할 수 있습니다. |
| keywordGroups.groupName | string | 주제어. 검색어 묶음을 대표하는 이름입니다. |
| keywordGroups.keywords | array string) | 주제어에 해당하는 검색어. 최대 20개의 검색어를 배열로 설정할 수 있습니다. |
| device | string | 범위, 검색환경에 따른 조건입니다. - 설정 안함: 모든 환경 - pc: PC에서검색추이 - mo: 모바일에서 검색추이 |



타 금융사이트는 가독성, 가시성이 부족했다. 따라서 기업의 이슈나 분석을 어떻게 효과적으로 전달할 수 있을지 고민 끝에 각종 지표를 그래프 형식으로 나타냈다.

### 과제 도중 발생한 문제 1

**const** fetchInvest = () => {

**return** axios.get(`/api/weather/**${**code**}**`);

}

**const** {data,isLoading} = useQuery('invest',fetchInvest);

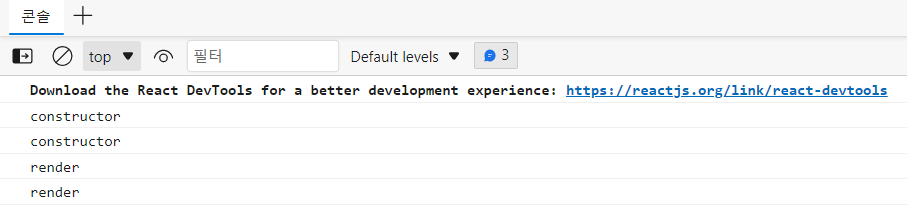
컴포넌트가 마운트 될 때 axios를 이용하여 데이터를 받아오려고 하였으나

{data.data.foreign} 이부분에서

Cannot read property ‘data’ of undefined 라는 TypeError 가 발생하였다. 데이터가 fetch되기 전에 DOM을 그리려다가 에러가 난 것을 금방 알았고 react-query에서 제공하는 loading state를 두어서 해결할 수 있었다. 저 구문에서 에러가 발생한 이유는 data는 어떤 객체도 갖지 않기 때문에 접근이 불가능했고 <>{**undefined**}</>는 렌더링 될 것이고 이는 별 문제없지만, <>{data.data.foreign}</>

는 <>{undefined.foreifn}</>로 렌더링 될 것이고 렌더링 되는 과정에서 에러가 발생한다. 말 그대로 undefined의 foreign을 알 수 없는 셈이다.

### 과제 도중 발생한 문제 2



React를 가지고 콘솔을 찍거나 차트를 그릴 때 2번씩 찍히는 문제가 발생하였다. 이는index.js에서 우리의 컴포넌트가 <StrictMode>로 감싸져 있기 때문이었다.

StrictMode란 애플리케이션에서 잠재적인 문제를 알아내기 위한 도구이다. 개발 모드에서만 활성화되고, 프로덕션 빌드에는 영향을 주지 않는다. 리액트 공식 문서 의Strict Mode 하단에 예상치 못한 부작용 검사 부분을 확인하면 알 수 있다. 렌더링 단계 생명주기 메서드들(setState, update function….)은 여러 번 호출될 수 있기 때문에, 부작용을 포함하지 않는 것이 중요하다. 그렇지 않으면 메모리 누수 등 다양한 문제를 일으킬 수 있다. 하지만 이런 문제들은 예측한대로 동작하지 않기 때문에 발견하기 어렵다. 그래서 Strict 모드가 자동으로 부작용을 찾아 주진 못하지만, 몇몇 함수들을 의도적으로 이중 호출하여서 발견할 수 있게 한다.

# 연구 결과 분석 및 평가

## 모델 간 성능 및 비용 비교

3.1에서 준비한 데이터로 3.2 ~ 3.5에서 구현한 Teacher 모델(Transformer), Student 모델(CNN), Distillation을 적용한 모델을 학습시켰다. Teacher 모델은 KOSPI 상위 종목 10개(시가 총액 기준)의 2015년 1월부터 2022년 7월까지의 데이터를 미리 학습시켰고, Student 모델은 사전 학습을 하지 않았다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Transformer (Teacher)** | **CNN (Student)** | **CNN with Distiller** |
| **epoch** | 15 | 15 | 15 |
| **batch** | 32 | 32 | 32 |
| **time(sec)** | 694.47 | 5.22 | 80.58 |
| **accuracy** | 0.4324 | 0.2432 | 0.3621 |

표 모델 별 걸린 시간 및 정확도

진행 결과 Teacher 모델의 정확도와 Student 모델의 정확도는 대략 80% 정도 차이가 났다. 하지만, Distillation을 통해 Teacher의 정보를 Student에 학습시킨 결과, 정확도가 50% 정도 상승하였다.

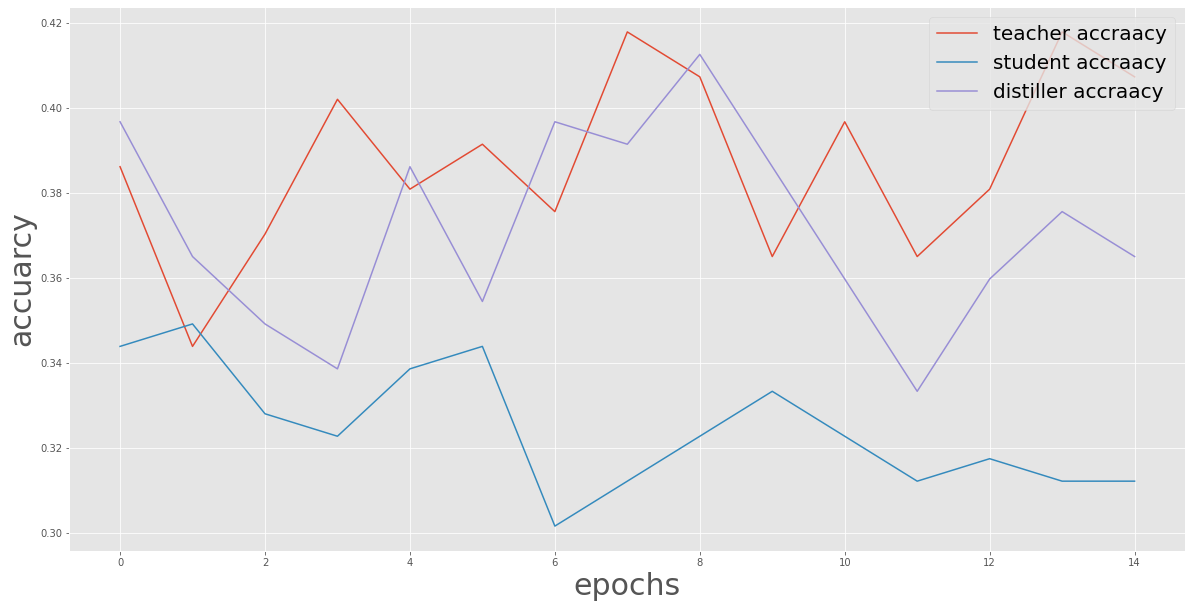


그림 모델 별 Accuracy (Categorical Accuracy)

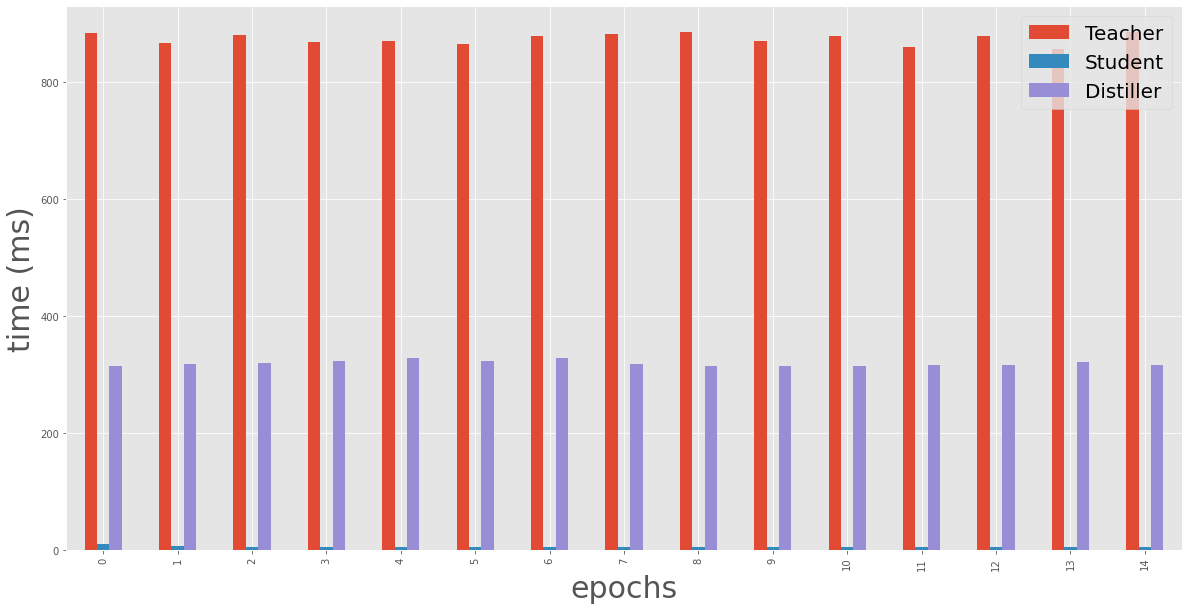


그림 모델 별 학습 진행 시간

그리고, 학습 진행 시간을 보면 Teacher 모델이 Student 모델보다 약 138배 더 오래 걸렸다. 하지만, Distillation을 통해 학습시켰을 때는 Teacher 모델보다 8배 정도 빨랐다.

# 결론 및 향후 연구 방향

## 알고리즘 구현 및 경량화

본 연구에서는 시계열 데이터, 그 중에서 주식 가격을 딥 러닝 모델을 이용해 학습하고, 이를 서비스화 하였다. 주식 데이터와 다양한 경제지표 데이터를 수집하고 데이터들의 특성을 분석하여 알맞게 가공하였다. 학습 과정에서 회귀 모델의 한계점을 파악하고, 분류 모델로 바꾸었다. Transformer라는 비용이 많이 들어가는 모델을 Knowledge Distillation을 통해 CNN같은 비교적 비용이 적은 모델로 Transformer 모델의 정확도를 어느정도 따라가면서, 학습에 걸리는 시간을 줄일 수 있었다.

하지만, 주식에 대한 Domain Knowledge의 부족으로 주식 종목별로 적용 가능한 적절한 경제 지표나 다른 데이터들을 추가하지 못하였고, 모델에 최적화된 데이터 전처리를 하지 못하였다. 그리고, Teacher 모델이 충분히 학습이 안되어 정확도가 안정적이지 않았다.

따라서, 주식 테마나 관련 주식, 또는 수익 상관관계가 높은 주식 데이터를 추가하거나 다른 경제 지표들을 추가하고, 모델에 맞는 데이터 전처리 기법을 사용하여 모델에 최적화된 데이터를 얻을 수 있다면 모델의 정확도가 향상될 수 있을 것이다. 그리고 모델의 parameter를 조절하거나 Layer 층을 조절하여 최적화하여 모델의 전체적인 비용을 감소하고 동시에 Distillation의 비용 감소 및 성능 향상을 이끌어낼 수 있을 것이다.

마지막으로, 지금은 서버에서 Distillation의 연산이 이루어지지만, 연산에 따른 비용이 충분히 작아진다면 이를 클라이언트의 웹 브라우저에서 진행하여 서버의 비용을 클라이언트들에게 분산시킴으로써 알고리즘 경량화라는 연구 목표에 더더욱 다가갈 수 있을 것으로 기대한다.

## 웹 서비스 구현

키움 증권에서 제공하는 open API를 사용하여 사이트에서 매수/매도 기능까지 추가하려고 했지만, TensorFlow와 Flask를 사용하려면 64bit 환경에서 해야 했고, 키움 증권 API는 32bit에서만 사용이 가능했다. 아직 해결방법을 찾지 못했지만, 추후 해결할 수 있게 노력할 것이다

# 참고 문헌

[1] “주린이'를 도와라... AI 투자 열풍, 국내 로보어드바이저 앱 5종 소개”, Available: http://www.aitimes.com/news/articleView.html?idxno=136379

[2] Alfredo Canziani, Adam Paszke and Eugenio Culurciello “AN ANALYSIS OF DEEP NEURAL NETWORK MODELS FOR PRACTICAL APPLICATIONS”, pp. 2, May 2016

[3] “Long short-term memory – Wikipedia”, Available: <https://en.wikipedia.org/>

[4] “Keras LSTM 입력 포맷의 이해 Understanding Input shapes in LSTM | Keras”, Available: <https://swlock.blogspot.com/2019/04/keras-lstm-understanding-input-and.html>

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin, “Attention Is All You Need", pp. 3, 5, Jun 2017

[6] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart and Marcus Brubaker, “Time2Vec: Learning a Vector Representation of Time", pp. 5, Jul 2019

[7]” Stock predictions with state-of-the-art Transformer and Time Embeddings”, Available: <https://towardsdatascience.com/stock-predictions-with-state-of-the-art-transformer-and-time-embeddings-3a4485237de6>

[8] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, “Distilling the Knowledge in a Neural Network", pp. 3, Mar 2015