졸업작품 발표

농산물 가격(시계열 데이터) 예측 회귀 모델

16101666 안재현

목표 및 제작 배경

1-1 작품의 목표

농산물의 가격 데이터와 거래량의 데이터를 학습하여, 미래(일주일 뒤)의 가격을 예측하는 딥러닝 모델을 구현하고, 2021 농산물 가격예측 AI 경진대회에 작품을 제출하는 것을 목표합니다.

1-2 제작 배경

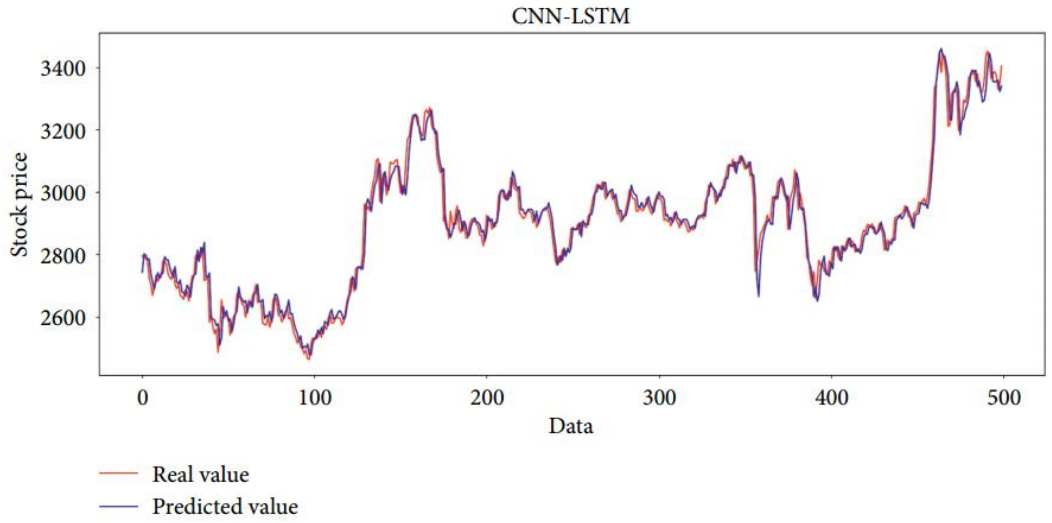
시계열 데이터에서 가장 예측하기 힘든 데이터는 주식 데이터 입니다. 시계열 데이터를 예측하는 기법으로는 보통 과거의 데이터만을 활용하는 모델로 구성이 되어 있습니다. 하지만 난도가 어려운 시계열 데이터의 경우, 많은 데이터 간의 상관관계를 사용하여, 예측하는 것을 확인할 수 있었습니다.

결국 시계열 데이터를 분석하는 것에 있어서 단순 과거정보로만 모델을 구성하는 경우 분명한 한계점이 있는 것을 깨달을 수 있었습니다.
농산물 데이터를 통해, 과거 정보로만 모델을 학습하는 경우의 결과와 상관관계를 사용하여 모델을 구성하는 결과를 비교하여, 데이터에 따른 모델들의 우수성을 평가합니다. 즉 최신 딥러닝 기법 및 기존 딥러닝 조합 모델을 사용하여, 시계열 데이터를 예측하는 것에 목표가 존재합니다. 이를 위해, 공공데이터 중 가격에 대한 직접적인 데이터가 존재하는, 농산물에 거래량 및 가격에 대한 데이터로 시계열 데이터의 가격을 예측합니다.

제약 조건

1-3 구현과정에서 예측되는 현실적 제약조건 기술

다음날의 종가를 직접 예측하는 경우, 올바른 교육 데이터와 손실함수, 최적화 기법이 사용되었음에도, 모델이 교육이 될 때 예측하는 데이터가 1-time lagging 되는 현상이 발생합니다



이를 해결하여 교육하기 위해서는, 최대한 학습 데이터와 정답데이터의 직접적인 상관관계를 덜 주도록 해야 합니다. 방법으로는 1. 입력데이터를 수정하여 직접적인 가격 데이터가 아닌, 가격의 변화율 데이터로 가공하여 사용하는 방법, 2. 정답데이터를 수정하여, 정답데이터를 가격 데이터가 아닌, 상승 및 하락으로 예측하게 하는 방법, 3. 다음 날을 예측하는 것이 아닌, 3일 뒤 혹은 일주일 뒤의 가격을 예측하는 방법입니다. 이는 결국 학습 데이터와 정답데이터의 차이가 존재하도록 만드는 방법입니다. 이번 농산물 데이터의 가격을 예측할 때는 일주일 뒤의 데이터를 예측하는 것을 목표합니다.

설계 목표 (Model 1)

목표하였던 모델은 총 두 가지로 아래와 같습니다.

1. CNN-LSTM-Attention (단일 종목 시계열 예측)
2. DTML [3] (상관관계를 이용한 다 종목 시계열 예측)

단일 종목 시계열 예측으로, 과거 정보를 사용하여 예측하는 모델로 구현한 것은 CNN–LSTM–Attention 입니다.

CNN : 시계열 데이터의 특징을 잘 파악하기 위해 사용, 가령 일주일 뒤의 데이터를 예측한다고 할 때, 커널 사이즈는 7으로 설정.

LSTM : 다음 데이터를 예측할 때, 이전의 정보들을 참고하기 위한 RNN계열의 Layer이며,
일반 LSTM Layer를 사용하는 것보다, 더 성능이 좋은 Bidirectional(양방향) LSTM을 사용함. 또한,
LSTM 단일 층이 아닌, LSTM Layer를 쌓은 Stack 형식의 LSTM을 사용.

Attention : LSTM layer의 Last hidden state를 가장 잘 표현하고 있는 이전의 hidden state를 사용하여, 더 좋은 결과를 도출.

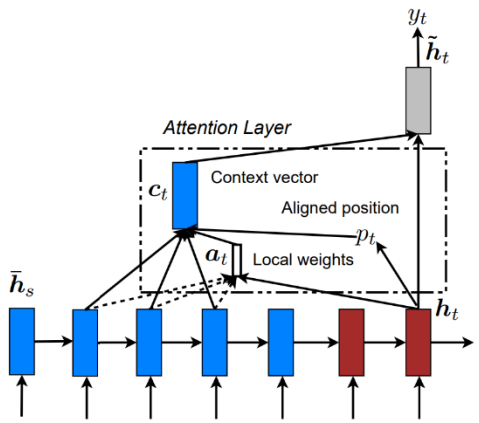


그림 : Attention을 사용한 seq2seq

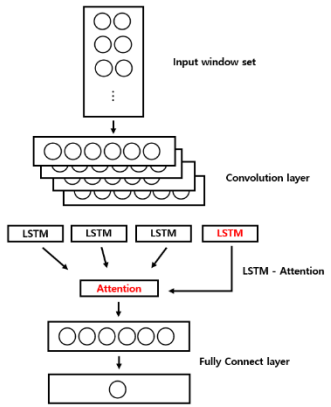


그림 : 모델의 구성 도면

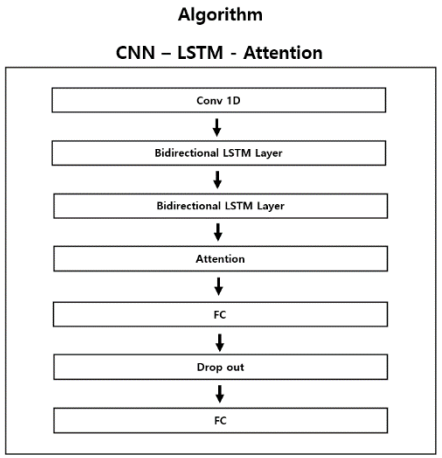


그림 : 모델의 알고리즘

설계 목표 (Model 2)

상관관계를 이용한 시계열 예측의 모델은 DTML입니다.

DTML : Accurate Multivariate Stock Movement Prediction via Data-Axis Transformer with Multi-Level Contexts (2021)

상관관계를 사용하여 예측하는 최신 모델은 21년 8월에 KDD 21에서 발표된 DTML[3]이 존재합니다. 해당 모델은 주가 데이터의 등락을 예측하며, 해당 모델은 **직접적인 수익을 내기 때문에 코드는 공개되어 있지 않으며**, 논문에는 모델의 구조만 표현하고 있습니다.

LSTM Layer : 이전의 정보들을 참고하기 위한 RNN계열의 Layer

Attention : LSTM layer의 Last hidden state를 가장 잘 표현하고 있는 이전의 hidden state를 사용하여, 더 좋은 결과를 도출.

Norm : Layer를 정규화, (hidden state의 범위가 다르기 때문)

Multi Context : Local Context와 Global Context를 합침 (is not concatenate)

Self Attention : key, query가 동일한 Attention Layer(scaled dot product Attention)

Add & Norm : Layer 정규화 및 이전 Layer의 input 정보 입력

Feed Forward : 기본적인 Fully Connected Layer

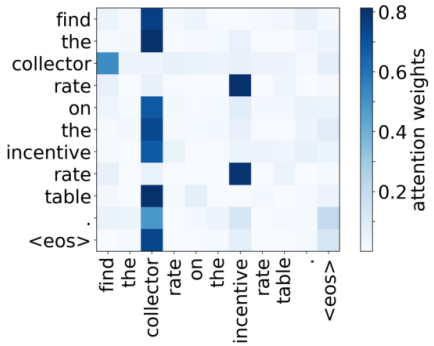


그림 : Self Attention Weights

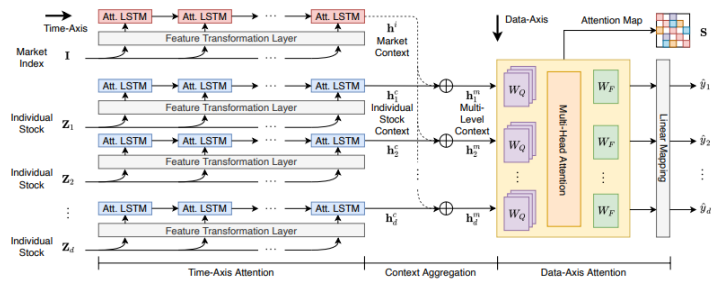


그림 : 모델의 구성 도면

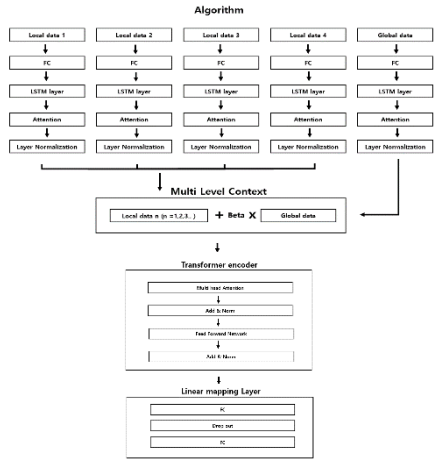


그림 : 모델의 알고리즘

실험 환경 설정

- 실험환경

모든 실험은 GPU 활용을 위해 Colab Pro 환경에서 실행하였습니다.

Tensorflow ver : 2.6.0

numpy ver : 1.19.5

Python ver : 3.7.12

- 자료수집

농 넷에서 제공되는 전국도매시장 거래정보 데이터 API [7]를 활용하여,
자료를 수집할 수 있었으며, 수집한 데이터는 산지별, 도매가격, 소매가격을 포함한 데이터이며, 이후, 전 처리 과정을 거쳐, 날짜 별 평균 거래량, 평균 가격을 구할 수 있었습니다.
총 21개의 데이터를 산출하였으며, 품목 16개 품종 5개입니다.
품목 : 배추, 무, 양파, 건고추, 마늘, 대파, 얼갈이배추, 양배추, 깻잎, 시금치, 미나리, 당근, 파프리카, 새송이, 팽이버섯, 토마토 품종 : 청상추, 백다다기, 애호박, 캠벨얼리, 샤인마스캇

- Train, Validation, Test Data

데이터의 수집 기간은 2016년 ~ 2021년이며,
70% train data, 20% validation data, 10% test data set
실제 검증은 private data를 통해, 실제 미래에 대한 검증이 이루어지기 때문에,
test set의 크기를 비교적 작게 설정했습니다.

- 데이터 예외

일요일에 거래되는 거래량이 0이기 때문에, 이는 입력데이터의 2차 전 처리 과정에서 제외해주었습니다.
또한, 건고추의 가격이 잠시 폭등할 시기가 존재하였는데, 보통의 거래금액이 2만 원인 것을 보아 10만 원이 넘는 금액은 예외상황으로 판단하여 건고추의 가격 전체 범위를 1000~30000 사이의 가격만 표현되도록 전 처리 했습니다.

Parameter setting (Model 1) Best Condition

loss function : mse
optimizer : Adam
Learning rate : 0.005
Convolution layer filters = 64
Convolution layer kernel_size = 6
Time_step(window_size) = 50(1주,2주), 60(4주)
Batch_size = 200
dropout : 0.2
(Stacking LSTM)
Number of hidden units in LSTM1 layer = 512
Number of hidden units in LSTM2 layer = 128
units of Attention Weights = 128
Epochs = 150, 300

Parameter setting (Model 2) Best Condition

loss function : mse
optimizer : Adam
Learning rate : 0.005, 0.001
SECTOR_NUM = 21
Number of hidden units in LSTM layer = 128
Feed Forward (FC) Units = 128
Linear mapping (FC) Units = 40
drop out = 0.3
Number of Head = 4
Time_step(window_size) = 50
Batch_size = 10
dropout : 0.3, 0.2
Epochs = 150, 300

실험 결과

- Train 과정 및 각 모델 Loss 값 검출

Model 2의 Train 과정 중 Validation Loss와 Train Loss는 0.029까지 관측되었으며, Model 1의 Train Loss와 Validation Loss는 각각 개별 종목마다 다르게, 관측되어 이를 표(Table 1)에 정리했습니다. DTML 모델을 사용하여, 일주일 뒤의 가격을 예측하도록 교육을 진행했습니다. 개별 모델을 통해서 교육할 때보다 Validation data set의 Loss 값이 확연하게 줄어드는 것을 확인하지 못했으며, 이에 대한 이유는 Multi head attention weights에서 찾을 수 있었습니다.

예측기간	1주	2주	4주
배추	5 10	5 15	10 30
무	28 64	56 57	116 158
양파	22 15	17 22	26 42
건고추	55 92	59 105	69 138
마늘	42 42	33 30	58 87
대파	44 48	43 54	85 77
얼갈이배추	34 71	32 71	42 135
양배추	34 45	31 49	92 148
깻잎	52 139	42 103	66 187
시금치	17 31	27 28	24 47
미나리	57 78	56 65	64 85
당근	13 19	11 26	23 30
파프리카	23 47	22 33	38 74
새송이	43 48	40 52	62 87
팽이버섯	50 90	47 79	81 165
토마토	19 44	17 49	36 93
청상추	49 73	45 44	40 54
백다다기	45 87	45 79	40 104
애호박	31 57	25 57	43 74
캠벨얼리	12 11	11 16	13 15
샤인마스캇	31 43	17 42	42 53

Table 1 : 개별 종목별 Train Loss, Validation Loss (단위 0.001)

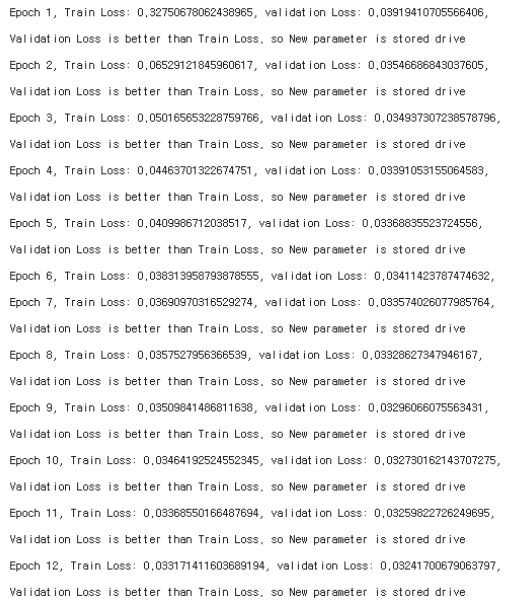


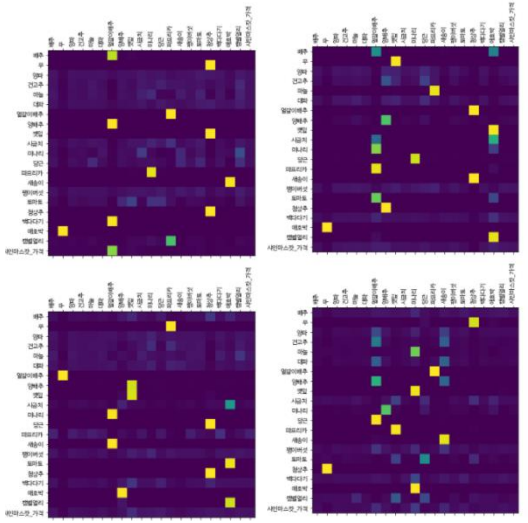
그림 : Model 2 Train process

실험 결과

- Multi head attention weights 시각화 (Model 2의 결과)

농산물 데이터 간의 상관관계가 존재하는 종목들도 몇 가지 존재하였으나 상관관계가 존재하지 않는 데이터들이 더 많으며 one hot vector에 치우쳐진 값들도 보이는 것을 관측할 수 있습니다.

또한, Multi head Attention의 연산 Scaled-Dot product Attention의 스케일링 값을 더 크게 하여서 one hot vector의 형태를 최대한 만들지 않도록 했음에도 불구하고, 농산물 간의 상관관계가 잘 표현되지 않은 것으로 보아 농산물 간의 가격에는 상관관계가 크기 없는 것으로 확인했습니다. (Fig. 15)



(Fig. 15)

- 가격 예측 (1주 후, 2주 후, 4주 후) (Model 1의 결과)

Model 1의 최상의 교육 결과를 위하여, 각 하이퍼 파라미터를 수정하며 교육을 진행하였습니다.
또한, 가장 교육이 잘되었을 때의 모델 파라미터 값들을 저장하여 새로운 데이터가 들어오더라도 이를 해당 파라미터를 불러와 예측합니다

2020-10-26+1week	624.2708	421.9819	1065.283	9011.511	5176.727	2085.542	683.8459	815.429	5899.809	2132.346	2602.988	1307.467	4252.274	2622.975	1566.051	2907.373	2247.986	1293.814	1425.479	3009.388	9145.539
2020-10-26+2week	301.8274	475.2282	1030.666	4933.192	5141.586	2023.768	920.022	843.5209	4929.625	2019.576	2500.306	1177.054	3997.873	2520.394	2068.934	2946.148	3639.793	1367.589	1867.672	3934.625	10829.61
2020-10-26+4week	534.0474	774.6703	1063.164	50397.03	5027.77	1888.636	981.3871	889.8675	4816.152	2205.241	2895.928	1490.543	4346.397	2721.918	1904.294	2835.903	2046.405	1301.169	1244.321	3591.302	10595.81
2020-10-27+1week	644.0701	373.0099	1066.482	9099.382	5191.217	2080.571	683.7409	814.2609	5872.849	2146.789	2560.059	1262.155	4252.097	2596.034	1525.06	2890.774	2259.439	1258.016	1397.261	3014.473	9105.161
2020-10-27+2week	327.6723	466.6609	1029.401	5027.778	5156.87	2034.777	912.9201	833.952	4983.152	1977.03	2482.846	1137.162	3988.31	2479.292	2067.062	3027.102	3581.943	1341.53	1895.734	3922.982	10903.19
2020-10-27+4week	539.1606	811.4738	1053.936	50496.73	4999.865	1951.717	978.3563	911.5924	4944.928	2197.054	2872.106	1527.274	4331.81	2723.258	1938.615	2861.241	2018.493	1304.543	1306.85	3618.815	11045.76
2020-10-28+1week	659.2155	357.2172	1066.254	9169.967	5202.216	2072.002	683.1201	849.7888	5864.838	2160.834	2521.189	1266.736	4243.24	2549.259	1538.62	2876.375	2273.846	1245.825	1415.182	3016.138	9067.2
2020-10-28+2week	355.5594	456.6596	1029.709	5014.497	5170.007	2042.214	910.1475	837.6043	5047.577	1939.272	2505.747	1110.19	3960.011	2433.385	2058.445	3060.171	3536.963	1318.495	1918.368	3915.526	10987.56
2020-10-28+4week	547.3049	844.1542	1043.943	50497.17	4966.056	1979.811	976.8163	926.928	5129.901	2182.923	2849.33	1570.781	4310.296	2714.376	1962.986	2892.494	2044.354	1306.322	1344.426	3631.027	11507.27
2020-10-29+1week	670.3791	346.695	1069.575	9255.625	5207.967	2068.129	673.5817	872.797	5755.749	2173.6	2469.186	1258.688	4221.082	2494.399	1551.024	2904.095	2287.979	1245.652	1461.713	3058.855	9034.875
2020-10-29+2week	382.5301	445.0148	1030.924	5062.491	5184.057	2052.746	901.4167	834.8411	4947.747	1899.757	2566.632	1092.772	3931.416	2394.361	2041.459	3055.54	3500.706	1295.73	1924.174	3896.386	11069.83
2020-10-29+4week	556.8278	876.9648	1033.46	50550.24	4928.039	1963.934	977.9686	931.9304	5333.737	2154.53	2825.643	1623.109	4296.381	2713.195	1971.419	2923.287	2354.662	1353.21	1393.676	3639.511	11851.07
2020-10-30+1week	677.5078	345.562	1074.848	9146.499	5194.879	2072.306	673.2443	891.8829	5585.364	2184.013	2414.536	1235.505	4197.488	2446.54	1592.718	2918.328	2297.311	1240.208	1506.267	3115.034	9006.811
2020-10-30+2week	401.6241	432.9857	1031.828	4977.24	5190.458	2059.327	895.5386	835.6244	4745.492	1858.728	2662.092	1076.432	3912.788	2361.245	2021.276	3024.699	3458.793	1274.753	1918.038	3861.923	11136.08
2020-10-30+4week	569.124	900.3286	1025.823	50471.32	4886.957	1963.338	974.187	934.0122	5747.703	2098.989	2803.676	1661.063	4284.943	2701.496	1973.866	2955.551	2905.019	1389.393	1435.114	3590.317	12035.78
2020-10-31+1week	684.3678	343.8887	1079.864	9048.548	5182.289	2081.217	682.1548	915.6168	5422.776	2203.59	2377.069	1220.123	4168.612	2427.82	1599.951	2932.532	2309.084	1223.798	1508.842	3229.318	8983.527
2020-10-31+2week	423.8254	426.7737	1035.795	4902.534	5184.311	2067.515	879.4236	836.3337	4576.648	1817.662	2824.756	1064.633	3897.157	2343.086	1994.246	3028.632	3445.854	1248.34	1901.713	3847.596	11198.99
2020-10-31+4week	576.5618	904.0436	1018.341	50482.98	4850.757	1921.906	972.4772	939.2139	5991.532	2011.875	2758.77	1694.036	4277.242	2696.627	1955.418	2996.478	3363.23	1422.179	1488.966	3549.024	12216.11

결과[결론 도출]

- **농산물 가격예측 경진대회 출전**

Model 1을 사용하여 "한국 농수산 식품유통공사" 에서 주최한 "농산물 가격예측 경진대회" 에 출전하여 농산물 가격예측 경진대회에 제출하였으며,
private data set으로 23/199 등을 기록하고 있습니다.

- **결과 도출**

농산물 데이터의 경우, 품종 별 상관관계를 찾아 가격을 예측하는 Model2보다 과거 데이터만으로 예측하는 Model1이 더 좋은 성능을 보였습니다.
시계열 데이터를 예측하는 딥러닝 기법은 다양하게 발전되고 있으며 데이터의 특성에 따라 모델의 특징에 따라, 성능 차이를 보이는 것을 확인하였습니다. 그리고, 과거 데이터가 중요한 시계열 데이터의 경우 여러 데이터 간 상관관계를 사용하여 예측하는 모델보다는, 과거 데이터만 위주로 교육한 모델이 성능이 더 우수한 것을 확인할 수 있었습니다. 시계열 데이터의 적용사례는 다양하게 발전될 것이며, 특히 다양한 요인에 의해 지배를 받고 있어 예측의 난이도가 가장 어려운 주가 데이터 또한 예측이 가능한 수준까지 발전될 것으로 예상합니다.

- **발전사항**

추가적인 발전사항으로, 상관관계를 사용한 모델들이 발전된다면,
언젠가 가격의 예측을 할 때 제품에 들어가는 원자재의 가격들과 시장의 흐름(지수)을 넣어 가장 연관성이 깊은 데이터를 찾아내는 것도 가능하게 될 것입니다. 기존의 DTML 모델은 과거 지수 데이터와 개별 주식들의 과거 데이터를 사용하여 주가의 등락을 예측하였는데, 이를 제품의 원자재 가격과 환율 및 지수 데이터 등 주식가격과 상관관계가 존재할 법한 다양한 입력데이터를 넣고 주가를 예측해보는 것으로 발전될 수 있습니다.

또한, 주가 데이터 뿐만 아닌, 암호화폐의 가격을 예측하려고 하는 시도 또한 늘어나는 추세입니다. 임의로 Global Context를 만들거나, Global Context로 비트 코인의 시세를 넣고 다른 2세대 3세대의 암호화폐의 가격 등락을 예측하는 모델을 만들어 내는 것으로도 발전될 수 있습니다.

참고 문헌

Ⅵ. 참 고 문 헌

[1] Wenjie Lu, Jiazheng Li, Yifan Li, Aijun Sun, and Jingyang Wang A CNN-LSTM-Based Model to Forecast Stock Prices", Hindawi Complexity (2020)

[2] Yaping Hao and Qiang Gao, "Predicting the Trend of Stock Market Index Using the Hybrid Neural Network Based on Multiple Time Scale Feature Learning" MDPI (2020)

[3] Jaemin Yoo, Yejun Soun, Yong-chan Park and U Kang "Accurate Multivariate Stock Movement Prediction via Data-Axis Transformer with Multi-Level Contexts " KDD 21 (2021)

[4] Minh-Thang Luong, Hieu Pham, and Christopher D Manning "Effective approaches to attentionbased neural machine translation" arXiv preprint arXiv:1508.04025, (2015)

[5] Ilya Sutskever, Oriol Vinyals and Quoc V. Le "Sequence to Sequence Learning with Neural Networks", NIPS (2014)

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin "Attention is all you need " NIPS (2017)

[7] https://www.nongnet.or.kr/anss/dmndSplyInfo.do (농넷)

[8] https://www.tensorflow.org/

추후 캡스톤디자인 관련 계획

1. 한국 가스 공사 주최 가스 공급량 수요예측 모델 개발

2. 딥러닝 논문 학습 및 모델 구현

   Transformer 및 시계열 데이터 관련된 논문 학습

3. DTML 모델을 사용한 다른 데이터 셋 적용

 - 암호화폐 데이터 크롤링
 - DTML 모델 수정 필요 (classification로의 변형 및 Global Context 코드 수정)

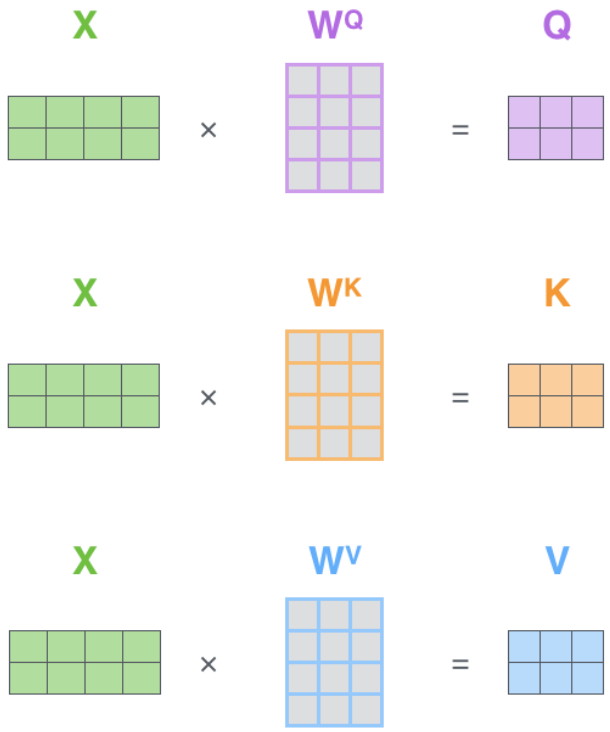
참고 사항

Transformer 주요 Layer

Self Attention

Q : 비교할 대상
K : 비교 당하는 대상
V : Attention Score를 구한 뒤 곱에 사용

행렬로써 한번에 계산이 가능
모든 입력에 대한 임베딩 벡터를 하나의 행렬로 만들어 버림
-> Wq, Wk, Wv는 하나의 Head에서 공유되기 때문



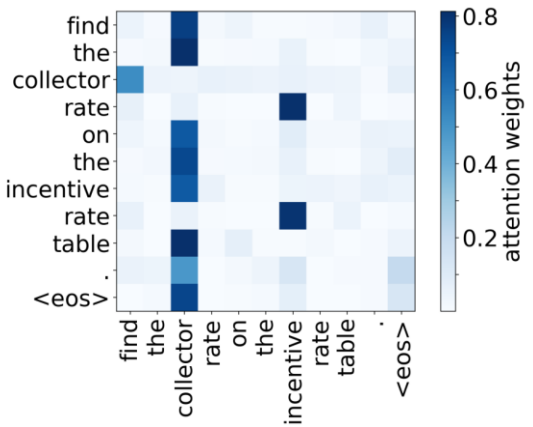
출처: Alammar

계산을 된 값을
root(임베딩 벡터의 차원)으로 나눔

Scaled Attention score를 만듬

Value Vactor와 곱해서 output을 만듬

Ex) Self Attention output



$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

Multi head Attention

배워야 할 개념

1. Input Embedding
2. Positional Encoding
3. Multi-Head Attention
4. Add & Norm
5. Feed Forward
6. Masked Multi-Head Attention

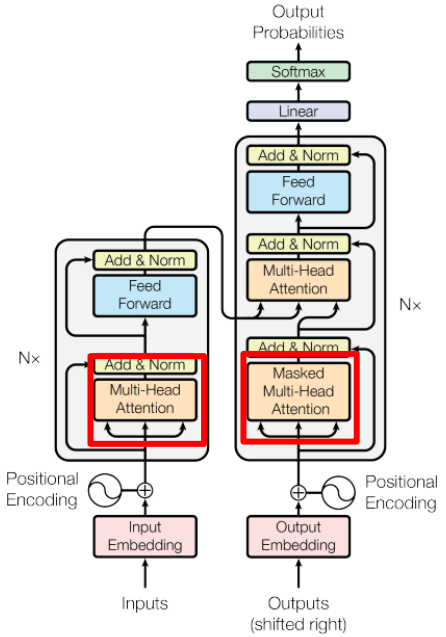


Figure 1: The Transformer - model architecture.

Multi-Head Attention이란 ?

Attention : (대부분) Self Attention

Multi-Head : 여러 개의 Attention

-> 여러 개의 Attention을 사용한 Layer

1. Self Attention의 과정
2. 여러 개의 Attention 연산을 한번에 처리하는 방법

Self Attention : Scaled dot product attention

dot product attention = 루 옹 어텐션

Multi head Attention

여러 개의 Attention 연산을 한번에 처리하는 방법

Embedding이란:

결국 단어가 입력되었을 때 고정된 크기의 벡터 값으로 변환해주는 Layer 입니다.

Ex) 고정된 벡터의 크기가 "8" 일 경우

"I" -> (0.1, 0.3, 0.2, 0.4 0.62, 0.34, 0.54,0.23)
"am" -> (0.4, 0.5, 0.32, 0.1, 0.52, 0.54, 0.42, 0.57)
"a" -> (0.15, 0.33, 0.64, 0.32, 0.4, 0.77, 0.85, 0.89)
"boy" -> (0.13, 0.30, 0.44, 0.52, 0.7, 0.5, 0.97,0.13)

Embedding Vector를 사용

-> 하나의 단어에 대한 설명이 Embedding Vector의 차원으로 변경되므로

Embedding Vector를 내가 원하는 Head의 수 만큼으로 나눠서, 해결한다.

위의 예시에서, Head의 개수가 2이라고 할 경우

8/2 = 4

하나의 Attention Vector당 4개의 Embedding 벡터를 가져간다.

Multi head Attention

여러 개의 Attention 연산을 한번에 처리하는 방법

Basic) Head

“I” -〉 (0.1, 0.3, 0.2, 0.4 0.62, 0.34, 0.54,0.23)
“am” -〉 (0.4, 0.5, 0.32, 0.1, 0.52, 0.54, 0.42, 0.57)
“a” -〉 (0.15, 0.33, 0.64, 0.32, 0.4, 0.77, 0.85, 0.89)
“boy” -〉 (0.13, 0.30, 0.44, 0.52, 0.7, 0.5, 0.97,0.13)

Multi - Head

(Head 개수 : 2개)
(embedding size / head_num = multi head embedding dim)

head1

“I” -〉 (0.1, 0.3, 0.2, 0.4)
“am” -〉 (0.4, 0.5, 0.32, 0.1)
“a” -〉 (0.15, 0.33, 0.64, 0.32)
“boy” -〉 (0.13, 0.30, 0.44, 0.52)

head2

“I” -〉 (0.62, 0.34, 0.54,0.23)
“am” -〉 (0.52, 0.54, 0.42, 0.57)
“a” -〉 (0.4, 0.77, 0.85, 0.89)
“boy” -〉 (0.7, 0.5, 0.97,0.13)

입력 embedding dim을 Head의 개수 만큼 나누게 된다면,

동일한 데이터 내부에서 Head의 개수만큼의 Self-Attention이 가능해집니다,

Self Attention 1개(그림 왼쪽) vs Self-Attention 2개(그림 오른쪽)

1. Self-Attention의 개수가 늘어나기 때문에, Size가 기존과 달라지는가?
2. Self-Attention의 개수가 늘어났을 때를 시각화

Multi head Attention

Basic) Head

“I” -〉 (0.1, 0.3, 0.2, 0.4 0.62, 0.34, 0.54,0.23)
“am” -〉 (0.4, 0.5, 0.32, 0.1, 0.52, 0.54, 0.42, 0.57)
“a” -〉 (0.15, 0.33, 0.64, 0.32, 0.4, 0.77, 0.85, 0.89)
“boy” -〉 (0.13, 0.30, 0.44, 0.52, 0.7, 0.5, 0.97,0.13)

Multi - Head

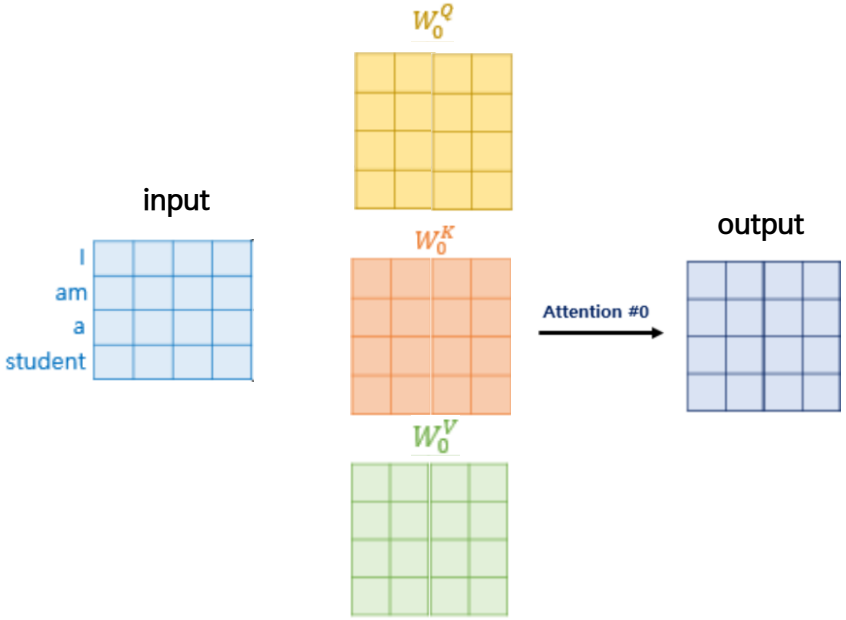
head1

“I” -〉 (0.1, 0.3, 0.2, 0.4)
“am” -〉 (0.4, 0.5, 0.32, 0.1)
“a” -〉 (0.15, 0.33, 0.64, 0.32)
“boy” -〉 (0.13, 0.30, 0.44, 0.52)

head2

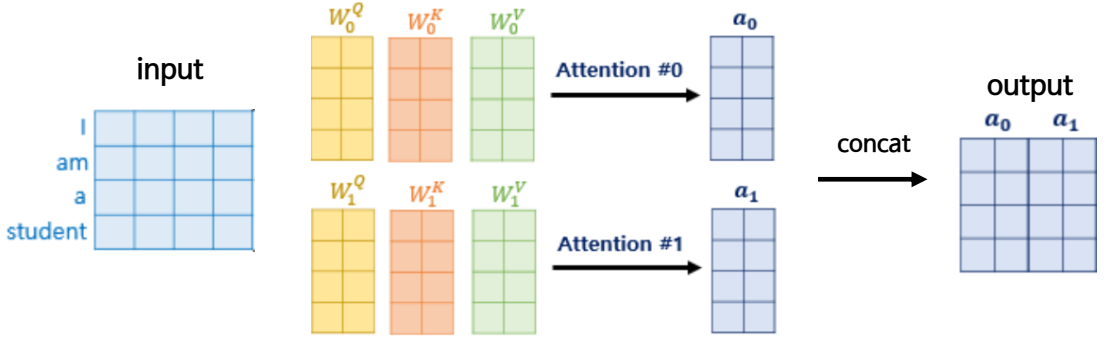
“I” -〉 (0.62, 0.34, 0.54,0.23)
“am” -〉 (0.52, 0.54, 0.42, 0.57)
“a” -〉 (0.4, 0.77, 0.85, 0.89)
“boy” -〉 (0.7, 0.5, 0.97,0.13)

Self Attention 1개(그림 왼쪽)



vs

Self-Attention 2개(그림 오른쪽)



Transformer

배워야 할 개념

1. Input Embedding
2. Positional Encoding
3. Multi-Head Attention
4. Add & Norm
5. Feed Forward
6. Masked Multi-Head Attention

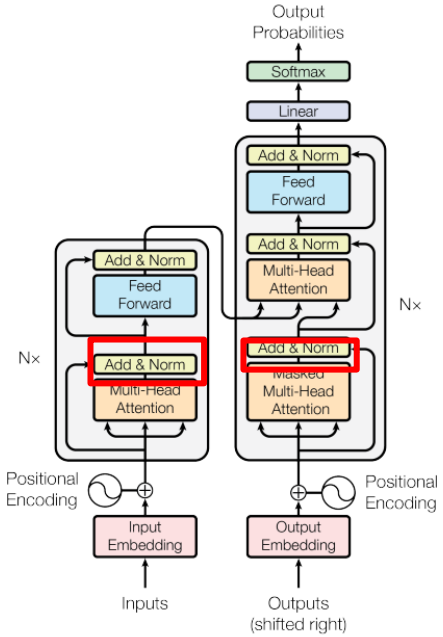


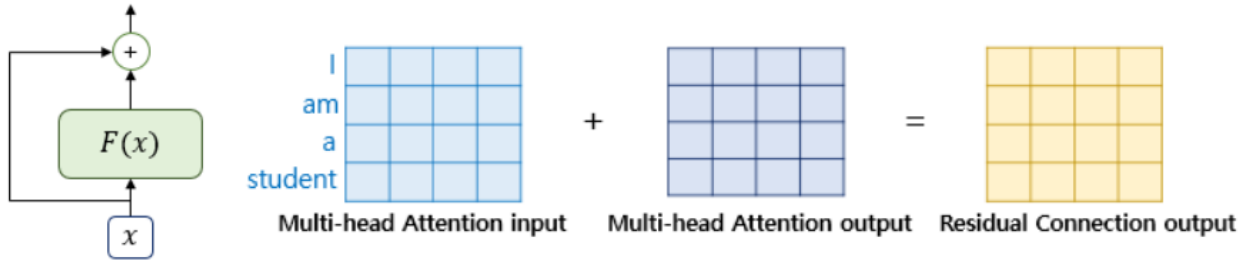
Figure 1: The Transformer - model architecture.

Add & Norm 이란 ?

Add : Residual connection

Norm : Layer Normalization

$$H(x) = x + F(x)$$



Residual connection : 입력을 출력에 더해주는 과정

Why : 어텐션 연산과 더불어, 병렬적인 과정의 연산이 이어지므로,
기존 데이터의 속성을 잃을 수 있는 문제가 생기는데,
이를 해결해주어서, 교육을 더 잘 될 수 있도록 도움을 줌

주의) 입력과 출력의 크기가 동일 해야함
(Resnet에 사용되는 개념)

Transformer

배워야 할 개념

1. Input Embedding
2. Positional Encoding
3. Multi-Head Attention
4. Add & Norm
5. Feed Forward
6. Masked Multi-Head Attention

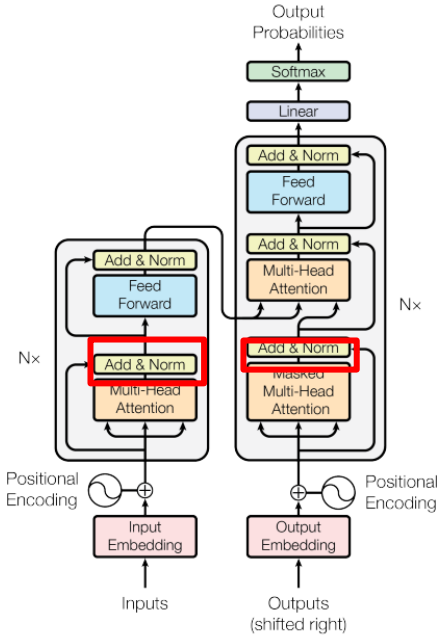


Figure 1: The Transformer - model architecture.

Add & Norm 이란 ?

Add : Residual connection

Norm : Layer Normalization

$$\hat{x}_{i,k} = \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

$$ln_i = \gamma \hat{x}_i + \beta = LayerNorm(x_i)$$

Layer Normalization : 정규화 + 학습가능한 파라미터 (감마, 베타)
각 단어의 차원을 각각 정규화를 해주어서,
학습을 더욱 잘 될 수 있도록 도와줍니다.

(감마의 초기 값 : 1, 베타의 초기 값 : 0)

$$LN = LayerNorm(x + Sublayer(x))$$

DTML (KDD 21)

Stock Prediction

Research Track Paper KDD '21, August 14–18, 2021, Virtual Event, Singapore

Accurate Multivariate Stock Movement Prediction via Data-Axis Transformer with Multi-Level Contexts

Jaemin Yoo
Seoul National University
Seoul, South Korea
jaeminyoo@snu.ac.kr

Yejun Soun
Seoul National University
DeepTrade Inc.
Seoul, South Korea
sony7819@snu.ac.kr

Yong-chan Park
Seoul National University
Seoul, South Korea
wjdakf3948@snu.ac.kr

U Kang
Seoul National University
DeepTrade Inc.
Seoul, South Korea
ukang@snu.ac.kr

Stock Prediction

1 INTRODUCTION

How can we efficiently correlate multiple stocks for accurate stock movement prediction? Stock movement prediction is one of the core applications of financial data mining, which has received growing interest in data mining and machine learning communities due to its substantial impact on financial markets [9, 25, 31]. The problem is to predict the movement (rise or fall) of stock prices at a future moment. The potential of the problem is unquestionable, as accurate predictions can lead to the enormous profit of investment.

It is challenging to achieve high accuracy of stock movement prediction, since stock prices are inherently random; no clear patterns exist unlike in other time series such as temperature or traffic. On the other hand, most stocks can be clustered as sectors by the industries that they belong to [3]. Stocks in the same sector share a similar trend even though their prices are perturbed randomly in a short-term manner, and such correlations can be a reliable evidence for investors. For instance, one can buy or sell a stock considering the prices of other stocks in the same sector, based on the belief that their movements will coincide in the future.

주가의 움직임을 높은 확률로 예측하는 것이 목표이며,

그 근거로 사용되는 조건 중 1개는

대부분의 주가들은 산업에 따라 Sector가 구분되며,
움직임을 기간에 따라 구분을 두면 Short-term은 random하게 움직이지만,
Long term은 결국 Sector의 트랜드에 따라가는 것을 볼 수 있습니다.

Stock Prediction

Most previous works that utilize the correlations between stocks rely on pre-defined lists of sectors [15, 18]. However, using a fixed list of sectors makes the following limitations. First, one loses the dynamic property of stock correlations that naturally change over time, especially when training data span over a long period. Second, a prediction model cannot be applied to stocks that have no information of sectors or whose sectors are ambiguous. Third, the performance of predictions relies heavily on the quality of sector information rather than the ability of a prediction model.

그동안의 상관관계를 이용하여 주가를 예측하는 모델들의 단점을 기술

1. 정해진 섹터만 가능하며, 장기간 학습할 경우 동적인 속성을 잃게 됩니다.
(장기간 학습 시 예측 데이터는 학습했던 데이터와 특징이 달라진 데이터로 변경)

2. 업종이 애매하면 섹터의 고정이 불가능합니다.
(일정 섹터의 주식들만 예측이 가능)

3. 예측의 성능은 모델의 성능보다는 섹터의 정보에 크게 의존합니다.
(해당 데이터로 다른 모델에 넣어도 비슷한 성능을 냄)

Stock Prediction

In this work, we design an end-to-end framework that learns the correlations between stocks for accurate stock movement prediction; the quality of correlations is measured by how much it contributes to improving the prediction accuracy. Specifically, we aim to address the following challenges that arise from the properties of stock prices. First, multiple features at each time step should be used together to capture accurate correlations. For instance, the variability of stock prices in a day can be given to a model by including both the highest and lowest prices as features. Second, the global movement of a market should also be considered, since the relationship between stocks is determined not only by their local movements, but also by the global trend. Third, the learned correlations should be asymmetric, reflecting the different degrees of information diffusion in a market.

데이터들의 상관관계를 이용하여, 주가의 움직임을 예측하는 모델을 제안합니다.

상관관계의 기준 : 예측 정확도의 향상에 얼마만큼 기여하는지

1. 주가의 단일 특성(종가)만을 사용하는 것이 아닌, 고가, 저가 등의 다른 특성을 같이 입력합니다.

2. 종목 (local trend) 만을 학습하는 것이 아닌, 종목들의 전체 트렌드(global trend)를 같이 학습
Ex) 삼성전자, 하이닉스 등의 데이터 + 지수(코스피 100) 데이터도 입력

3. Global trend가 끼치는 영향이 각 종목마다 다르기 때문에, 반영 정도를 다르게 학습 시켜야 합니다.

Stock Prediction

We propose DTML (Data-axis Transformer with Multi-Level contexts), an end-to-end framework that automatically correlates multiple stocks for accurate stock movement prediction. The main ideas of DTML are as follows. First, DTML generates each stock's comprehensive context vector that summarizes multivariate historical prices by temporal attention. Second, DTML extends the generated contexts into multi-level by combining it with the global movement of the market. Third, DTML learns asymmetric and dynamic attention scores from the multi-level contexts using the transformer encoder, which calculates different query and key vectors for multi-head attention between stocks.

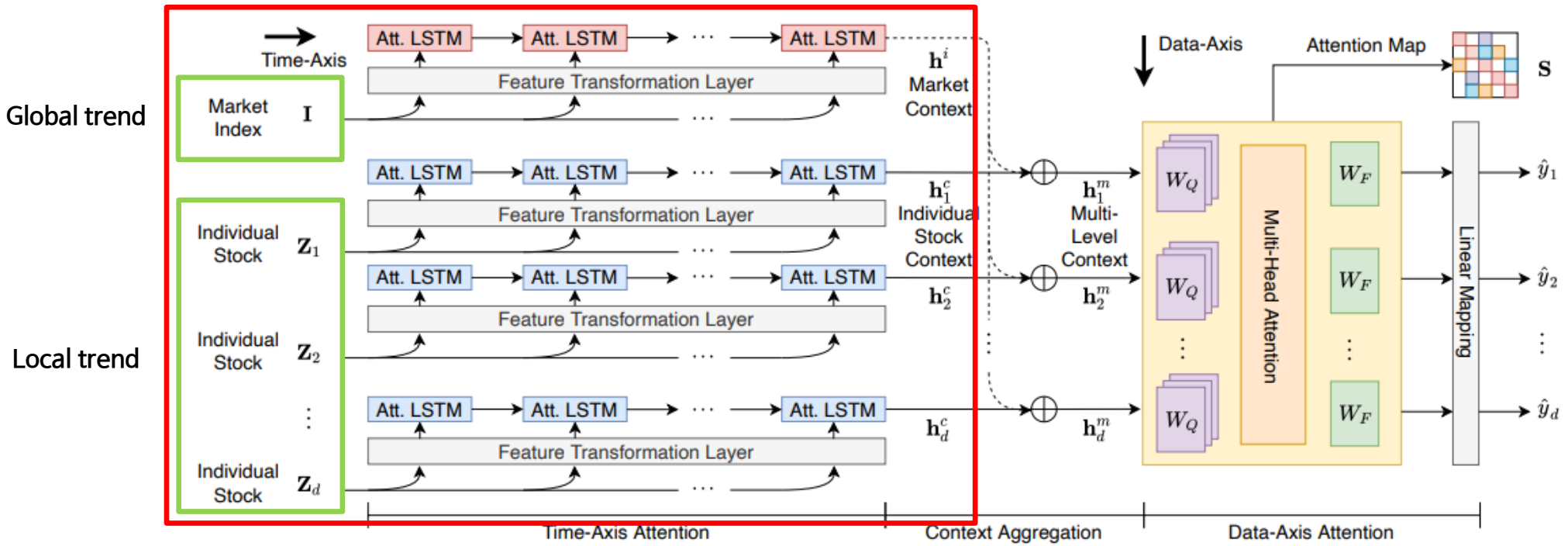
여기서 제안하는 모델 DTML의 경우
여러가지의 데이터를 가지고 와서, 각 데이터 간의 상관관계를 체크 한 뒤 예측을 실시합니다.

메인 아이디어 :

1. 주가 종목의 과거 데이터를 이용하여, 이를 요약한 context vector 만들기(LSTM -Attention)
2. Multi – level context vector 만들기
(Multi – level 이란 : local context vector + global context vector)

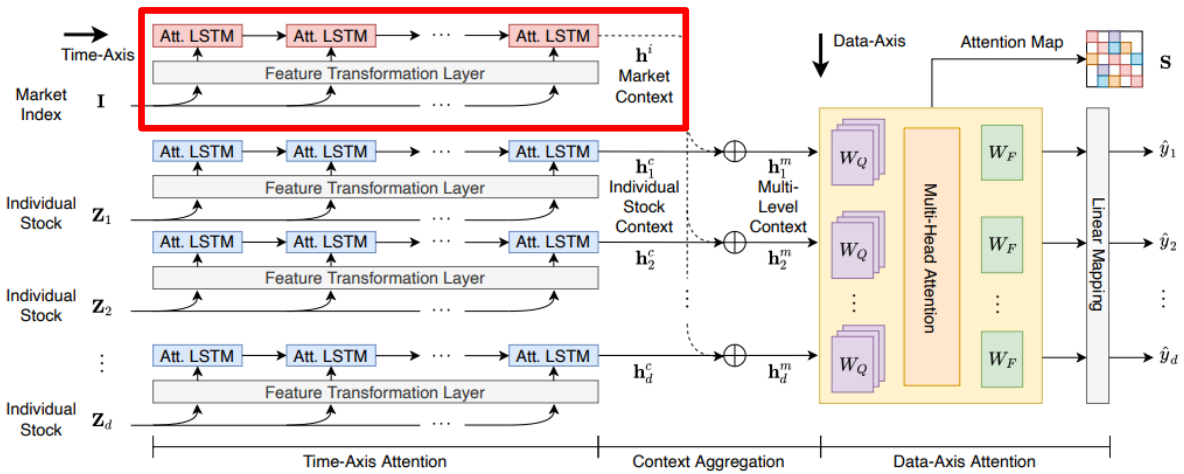
3. 위에서 얻은 Multi-level context vector가 입력으로 사용되어,
Transformer encoder구조를 사용한 동적인 Attention score를 만들어서 입력을 예측하고,
마지막에 Fully connected layer(activation = sigmoid)사용하여, 상승 및 하락 예측

Stock Prediction

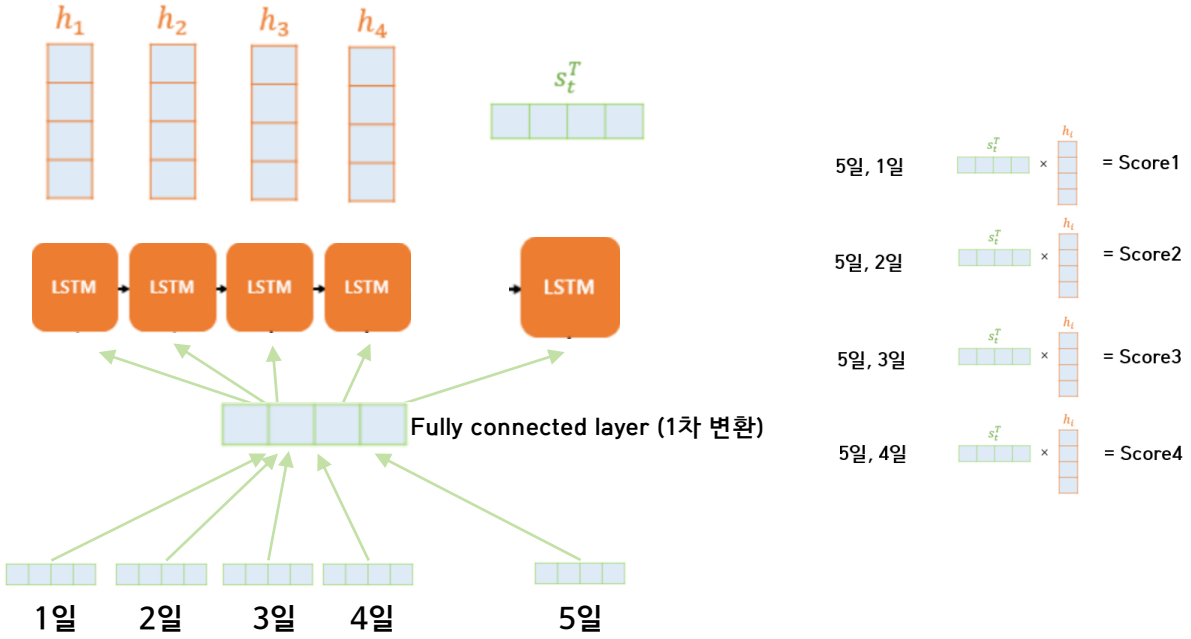


메인 아이디어 : 1. 주가 종목의 과거 데이터를 이용하여, 이를 요약한 context vector 만들기(LSTM -Attention)

Stock Prediction



Input :



Fully connected layer

Input 데이터를 그대로 사용하지 않고,
Fully connected layer에 한번 넣어서 사용한다.

-> LSTM Layer에 입력으로 들어가기 전에

복잡성을 늘리지 않으면서,

학습을 더욱 잘되게 할 수 있기 때문에 사용됨

Attention (Dot-product Attention(루 옹 어텐션))

Qurey : St (제일 마지막 입력된 5일째의 데이터)
Key : h1, h2, h3, h4

제일 마지막으로 입력되는 데이터가 Qurey로 설정됩니다.
가장 많은 과거 데이터를 함축하고 있기 때문

각 주식 종목마다 context vector를 만들게 됩니다.

Stock Prediction

개별 주식(주가) : Local context

시장의 흐름(지수) : global context

-〉 Layer Normalization

각각의 context vector 마다

Layer Normalization을 시도한다.

-〉 각각의 주식들은 실제 가격이
정규화 되어있지 않기 때문에,
정규화를 해주는 과정이 필요

Ex) 삼성전자 range : 40000 ~ 90000

하이닉스 range : 80000 ~ 150000

...

Context Normalization. The context vectors generated by the attention LSTM have values of diverse ranges, because each stock has its own range of features and pattern of historical prices. Such diversity makes subsequent modules unstable, such as the multi-level context aggregation (Section 3.3) or data-axis self-attention (Section 3.4). We thus introduce a context normalization, which is a variant of the layer normalization [1]:

$$h^c_{ui} = \gamma_{ui} \frac{\tilde{h}^c_{ui} - \text{mean}(\tilde{h}^c_{ui})}{\text{std}(\tilde{h}^c_{ui})} + \beta_{ui}, \tag{3}$$

where i is the index of an element in a context vector, mean$(\cdot)$ and std$(\cdot)$ are computed for all stocks and elements, and γ_{ui} and β_{ui} are learnable parameters for each pair (u, i).

Norm : Layer Normalization

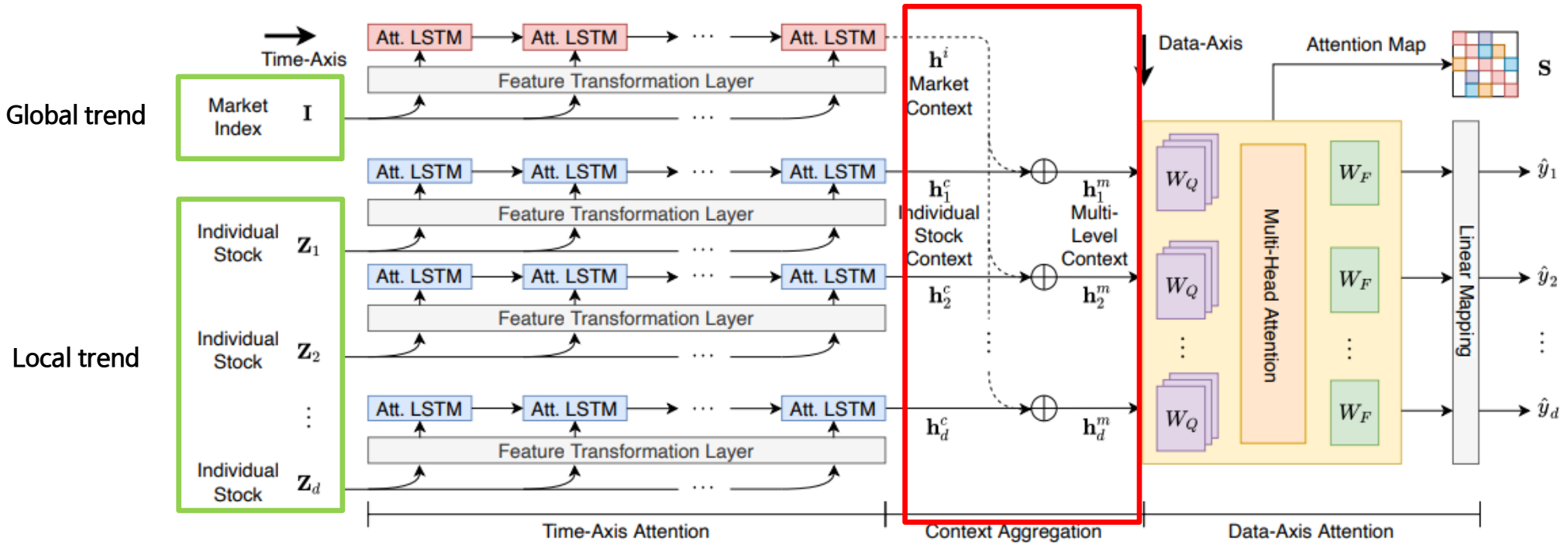
$$\hat{x}_{i,k} = \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}}$$

$$ln_i = \gamma \hat{x}_i + \beta = LayerNorm(x_i)$$

Layer Normalization : 정규화 + 학습가능한 파라미터 (감마, 베타)
각 단어의 차원을 각각 정규화를 해주어서,
학습을 더욱 잘 될 수 있도록 도와줍니다.

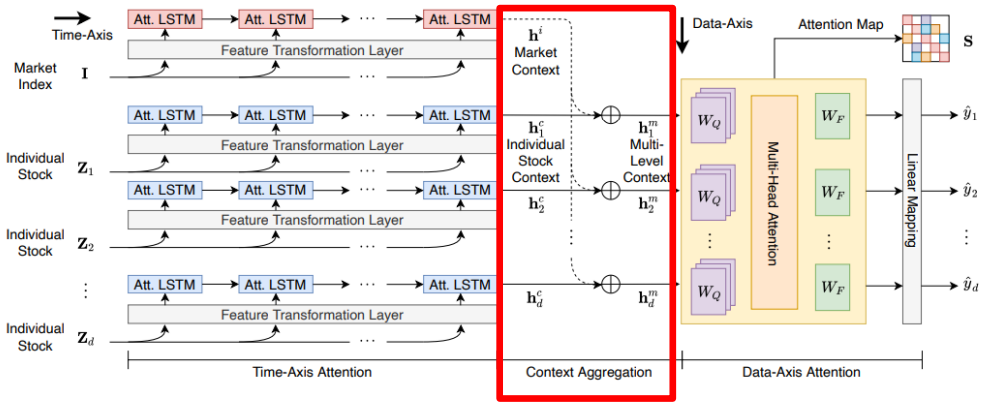
(감마의 초기 값 : 1, 베타의 초기 값 : 0)

Stock Prediction



메인 아이디어 : 2. Multi – level context vector 만들기
(Multi – level 이란 : local context vector + global context vector)

Stock Prediction



Multi-Level Contexts. Then, we generate a multi-level context $\mathbf{h}_u^m$ for each stock u by using the global market context $\mathbf{h}^i$ as base knowledge of all correlations:

$$\mathbf{h}_u^m = \mathbf{h}_u^c + \beta \mathbf{h}^i, \tag{4}$$

where β is a hyperparameter that determines the weight of $\mathbf{h}^i$. As a result, the correlation between two stocks is determined not only by their local movements, but also by the relationship to the global market context $\mathbf{h}^i$.

메인 아이디어 : 2. Multi – level context vector 만들기
(Multi – level 이란 : local context vector + global context vector)

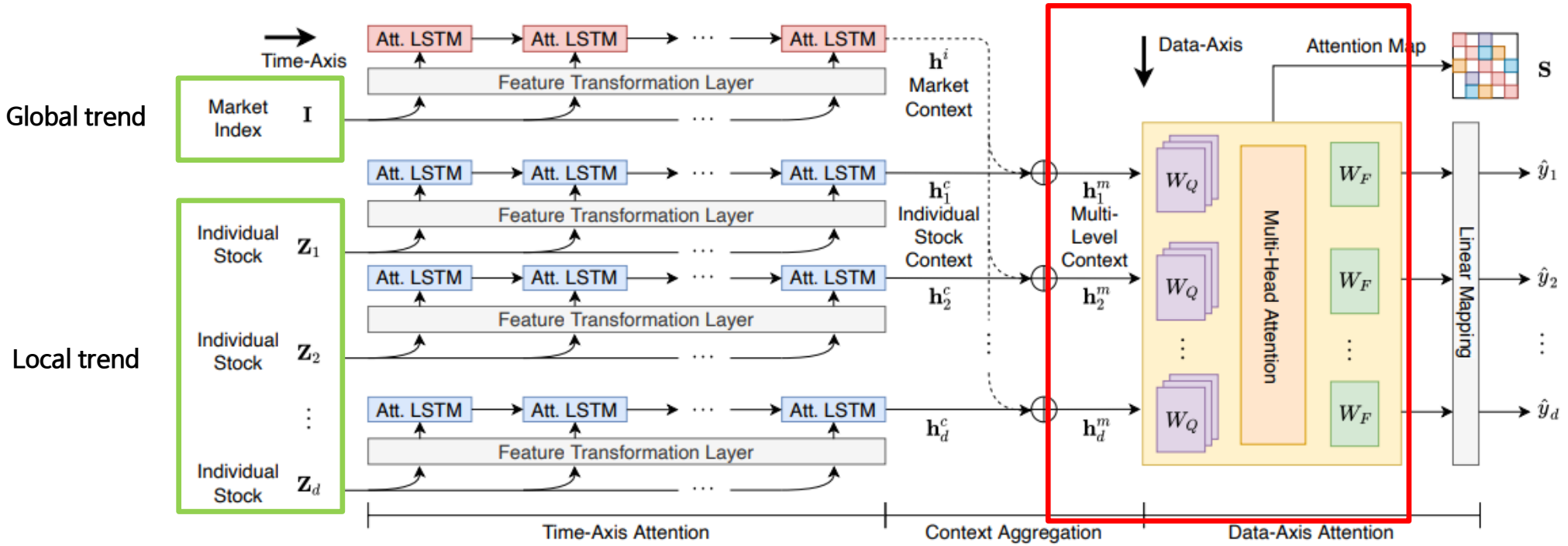
$$\mathbf{h}_u^m = \mathbf{h}_u^c + \beta \mathbf{h}^i,$$

Multi-context vector = local context vector + beta * global context vector

Beta : hyperparameter로 결정하며,

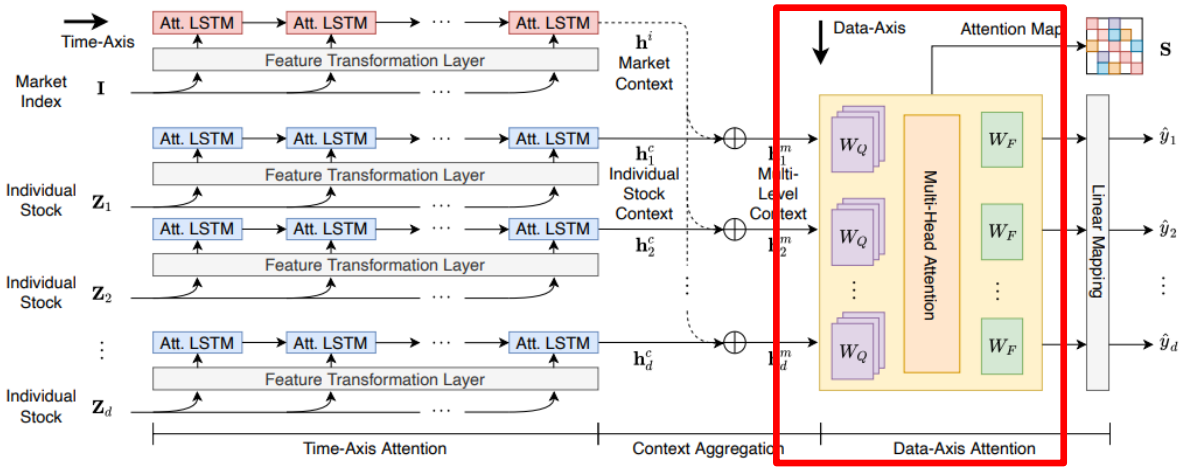
Beta의 비율에 따라서, 얼마나 시장의 흐름(global context vector)을 반영할 것인지 결정한다고 합니다.

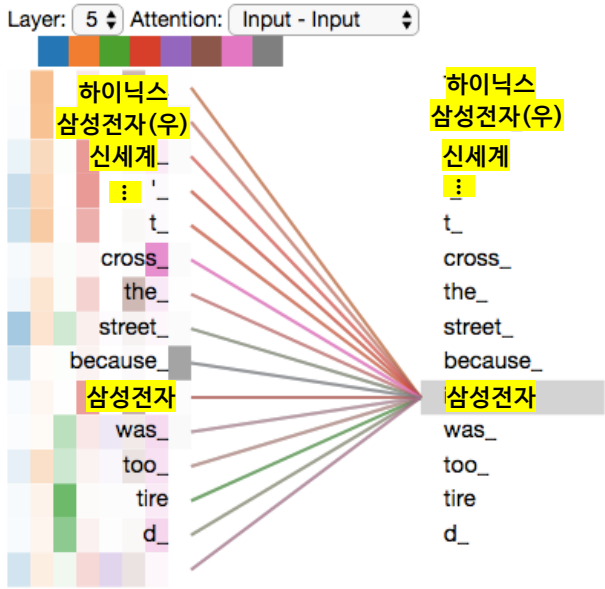
Stock Prediction



메인 아이디어 : 3. 위에서 얻은 Multi-level context vector가 입력으로 사용되어,
Transformer encoder구조를 사용하며 내부에 있는 Multi head Attention을 통해
개별 주식들의 상관관계를 파악하고, Feed-Forward Network를 통해서 output을 만듭니다.

Stock Prediction





Self-Attention. To apply the self-attention with respect to the data-axis, we first build a multi-level context matrix $\mathbf{H} \in \mathbb{R}^{d \times h}$ by stacking $\{\mathbf{h}_u^m\}_u$ for $u \in [1, d]$, where d is the number of stocks and h is the length of context vectors. We then generate query, key, and value matrices by learnable weights of size $h \times h$:

$$\mathbf{Q} = \mathbf{H}\mathbf{W}_{\mathrm{q}} \qquad \mathbf{K} = \mathbf{H}\mathbf{W}_{\mathrm{k}} \qquad \mathbf{V} = \mathbf{H}\mathbf{W}_{\mathrm{v}}. \tag{6}$$

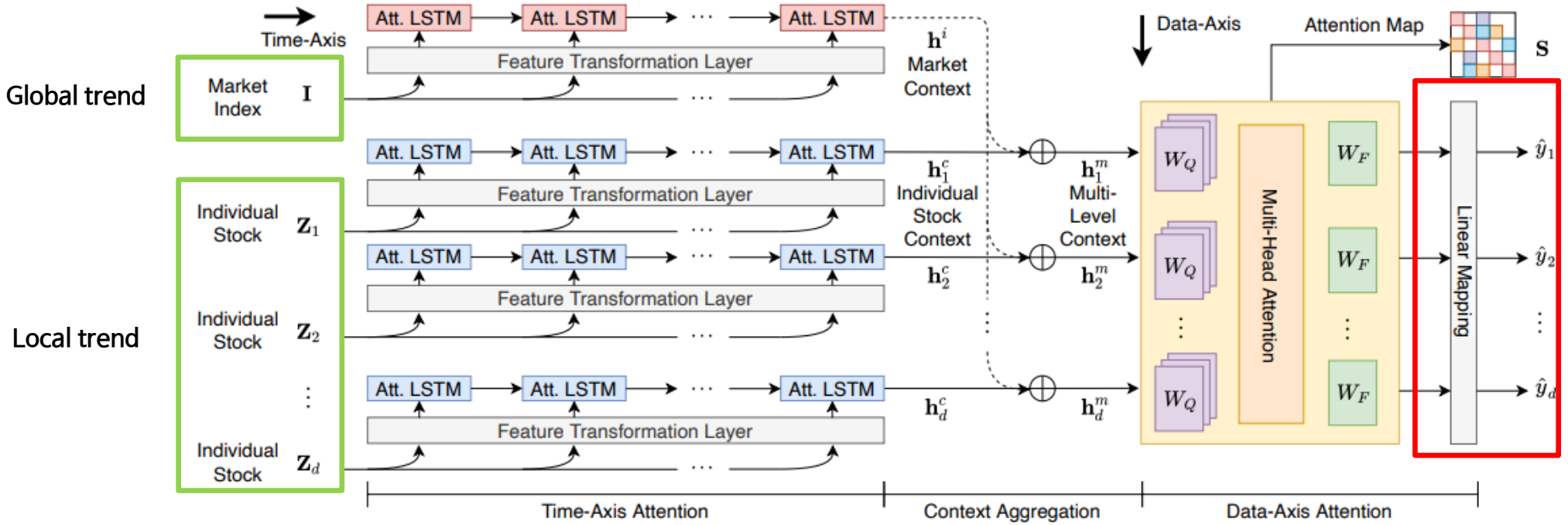
Then, we compute the attention scores from the query and key vectors, and aggregate the value vectors as follows:

$$\tilde{\mathbf{H}} = \mathbf{S}\mathbf{V} \quad \text{where} \quad \mathbf{S} = \mathrm{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^{\top}}{\sqrt{h}}\right). \tag{7}$$

Nonlinear Transformation. We update the aggregated contexts with residual connections as follows:

$$\mathbf{H}_{\mathrm{p}} = \tanh(\mathbf{H} + \tilde{\mathbf{H}} + \mathrm{MLP}(\mathbf{H} + \tilde{\mathbf{H}})), \tag{8}$$

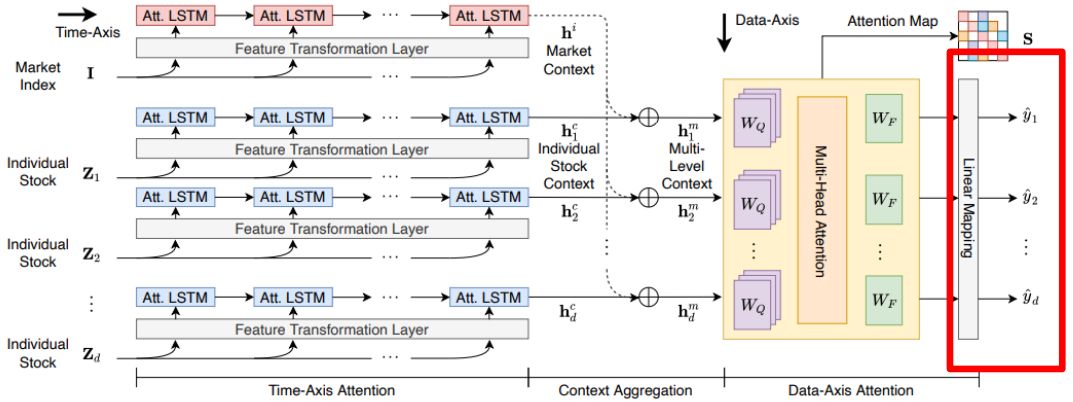
Stock Prediction



최종 Output :

Transformer encode의 output은 결국 입력의 크기와 동일하게 바뀌며,
해당 output을 최종적으로 이진 분류를 통해, 상승 및 하락을 예측합니다.

Stock Prediction



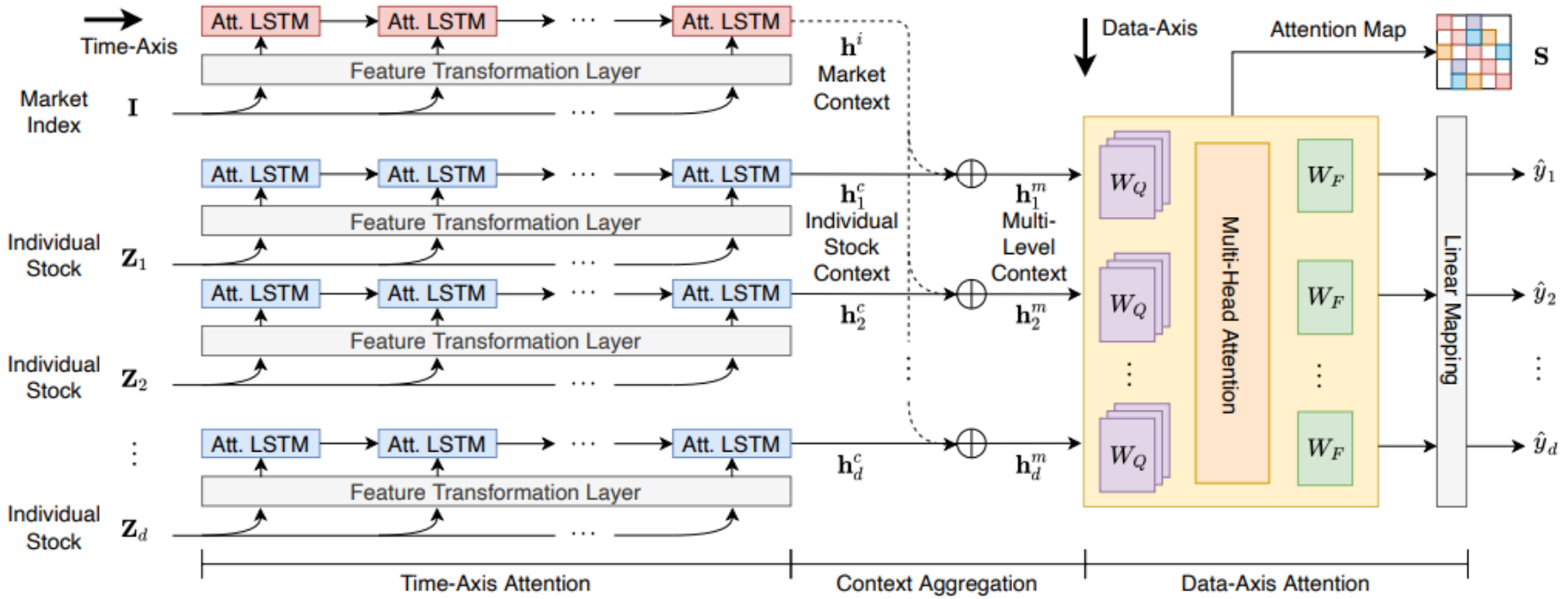
Final Prediction. We lastly apply a single linear layer to the transformed contexts to produce the final predictions as

$$\hat{\mathbf{y}} = \sigma(\mathbf{H}_{\mathrm{p}}\mathbf{W}_{\mathrm{p}} + \mathbf{b}_{\mathrm{p}}). \quad (9)$$

We apply the logistic sigmoid function σ to interpret each element $\hat{y}_u$ for stock u as a probability and use it directly as the output of DTML for stock movement prediction.

Sigmoid : 이진 분류 activation function으로 사용되며, 입력 값을 0 ~ 1사이로 결정
0.5이상의 결과가 나온다면 1, 0.5미만의 결과가 나온다면 0으로 해석합니다.

Stock Prediction



최종 정리 : 데이터가 예측되는 범위는 코스피 100이라고 가정.

코스피 100에서의 기업 중 선별된 기업 전부를
상승 혹은 하락 예측 (선별의 기준점은 나와있지 않음, 개수만 나옴)

Stock Prediction

Hyperparameters. We search the hyperparameters of DTML as follows: the window size w in $\{10, 15\}$, the market context weight β in $\{0.01, 0.1, 1\}$, the hidden layer size h in $\{64, 128\}$, the number of epochs in $\{100, 200\}$, and the learning rate in $\{0.001, 0.0001\}$. We set the strength λ of selective regularization to 1 and the dropout rate to 0.15. We use the Adam optimizer [14] for the training with the early stopping by the validation accuracy. For competitors, we use the default settings in their public implementations.

하루를 예측할 때 window size = 10 or 15 / global context의 반영 비율 (0.01, 0.1, 1)

Model	ACL18 (US)		KDD17 (US)		NDX100 (US)	
	ACC	MCC	ACC	MCC	ACC	MCC
LSTM [24]	0.4987 ± 0.0127	0.0337 ± 0.0398	0.5118 ± 0.0066	0.0187 ± 0.0110	0.5263 ± 0.0003	0.0037 ± 0.0049
ALSTM [31]	0.4919 ± 0.0142	0.0142 ± 0.0275	0.5166 ± 0.0041	0.0316 ± 0.0119	0.5260 ± 0.0007	0.0028 ± 0.0084
StockNet [31]	0.5285 ± 0.0020	0.0187 ± 0.0011	0.5193 ± 0.0001	0.0335 ± 0.0050	0.5392 ± 0.0016	0.0253 ± 0.0102
Adv-ALSTM [9]	0.5380 ± 0.0177	0.0830 ± 0.0353	0.5169 ± 0.0058	0.0333 ± 0.0137	0.5404 ± 0.0003	0.0046 ± 0.0090
DTML (proposed)	**0.5744 ± 0.0194**	**0.1910 ± 0.0315**	**0.5353 ± 0.0075**	**0.0733 ± 0.0195**	**0.5406 ± 0.0037**	**0.0310 ± 0.0193**

Model	CSI300 (China)		NI225 (Japan)		FTSE100 (UK)	
	ACC	MCC	ACC	MCC	ACC	MCC
LSTM [24]	0.5367 ± 0.0038	0.0722 ± 0.0050	0.5079 ± 0.0079	0.0148 ± 0.0162	0.5096 ± 0.0065	0.0187 ± 0.0129
ALSTM [31]	0.5315 ± 0.0036	0.0625 ± 0.0076	0.5060 ± 0.0066	0.0125 ± 0.0139	0.5106 ± 0.0038	0.0231 ± 0.0077
StockNet [31]	0.5254 ± 0.0029	0.0445 ± 0.0117	0.5015 ± 0.0054	0.0050 ± 0.0118	0.5036 ± 0.0095	0.0134 ± 0.0135
Adv-ALSTM [9]	0.5337 ± 0.0050	0.0668 ± 0.0084	0.5160 ± 0.0103	0.0340 ± 0.0201	0.5066 ± 0.0067	0.0155 ± 0.0140
DTML (proposed)	**0.5442 ± 0.0035**	**0.0826 ± 0.0074**	**0.5276 ± 0.0103**	**0.0626 ± 0.0230**	**0.5208 ± 0.0121**	**0.0502 ± 0.0214**

최종 결과