

# 심화 알고리즘

## -M5 Forecasting-

현예성 김민석 최성웅  
문구영 이노아 강호석

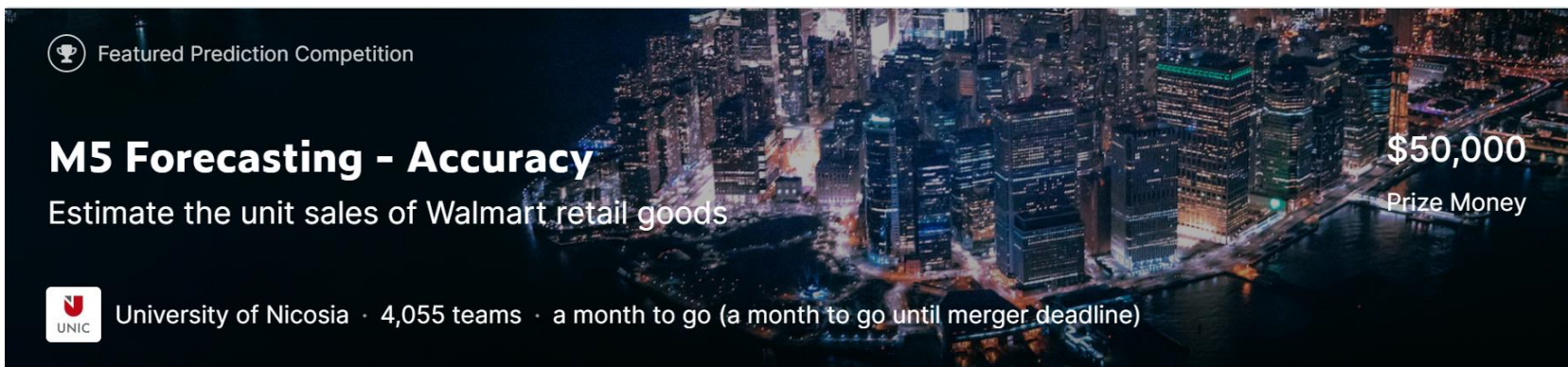
# INDEX


---

1. EDA

2. Light GBM

3. LSTM




 Featured Prediction Competition

# M5 Forecasting - Accuracy

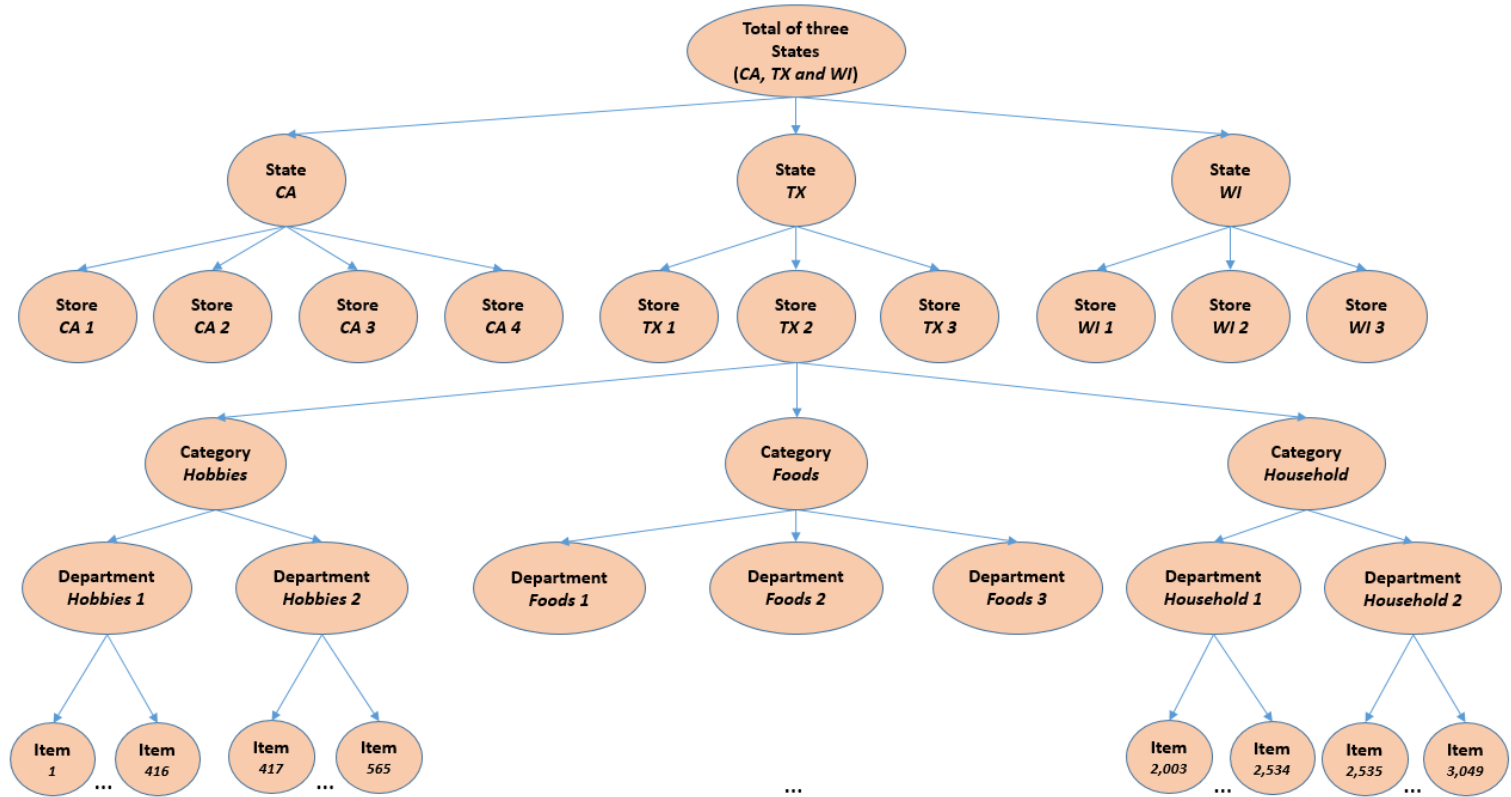
Estimate the unit sales of Walmart retail goods

**\$50,000**  
Prize Money

 University of Nicosia · 4,055 teams · a month to go (a month to go until merger deadline)

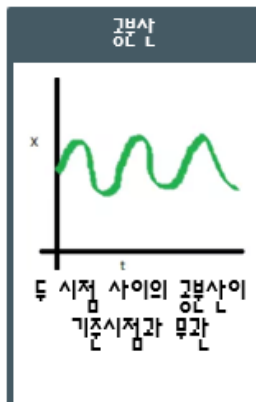
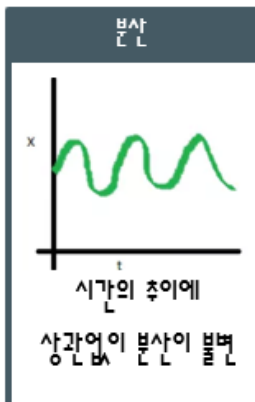
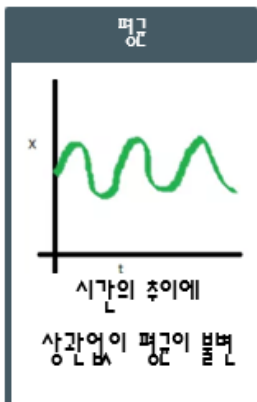
## Files

- `calendar.csv` - Contains information about the dates on which the products are sold.
- `sales_train_validation.csv` - Contains the historical daily unit sales data per product and store [`d_1` - `d_1913`]
- `sample_submission.csv` - The correct format for submissions. Reference the [Evaluation](#) tab for more info.
- `sell_prices.csv` - Contains information about the price of the products sold per store and date.
- `sales_train_evaluation.csv` - Available once month before competition deadline. Will include sales [`d_1` - `d_1941`]

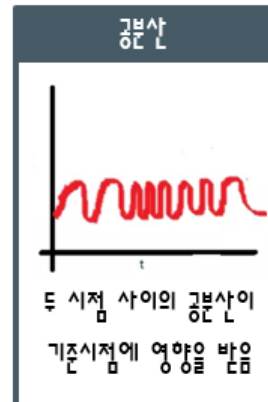
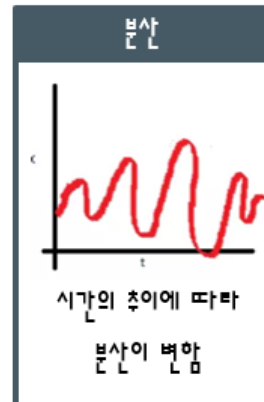
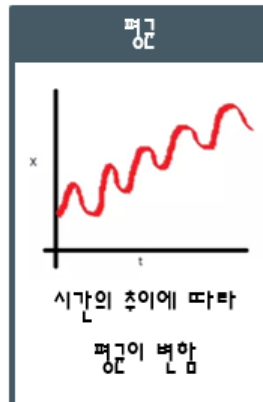


# Time Series

## Stationary Series

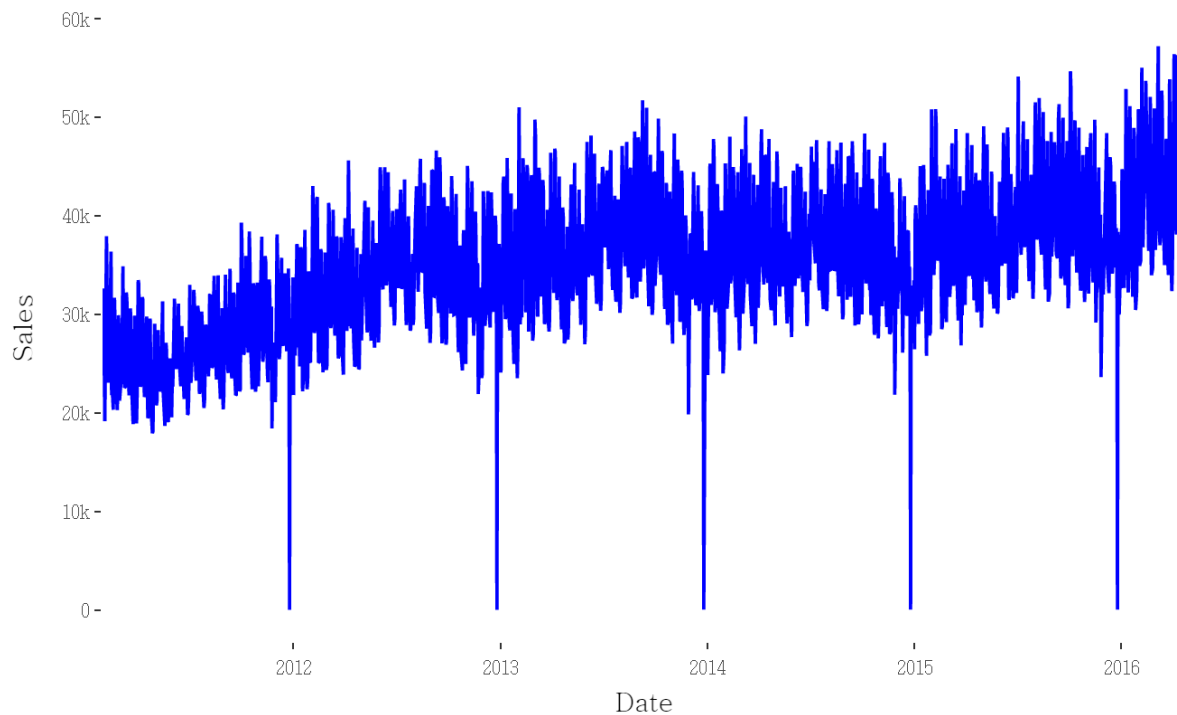


## Non-Stationary Series



# EDA

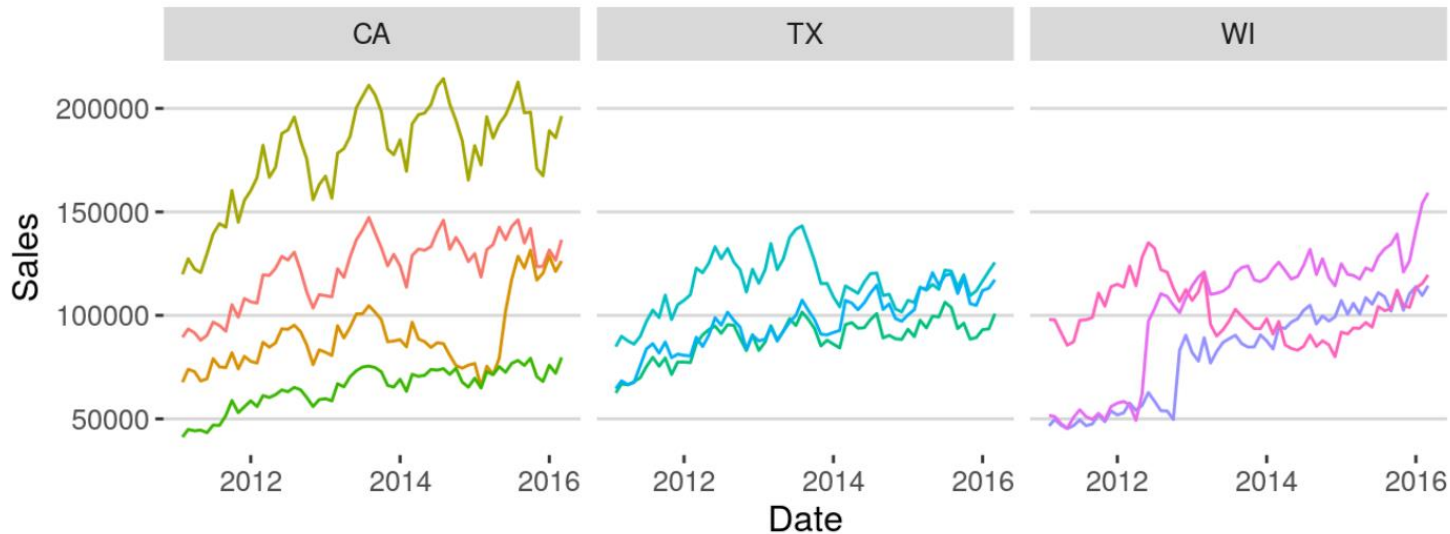
## total aggregate sales



## Monthly sales per State



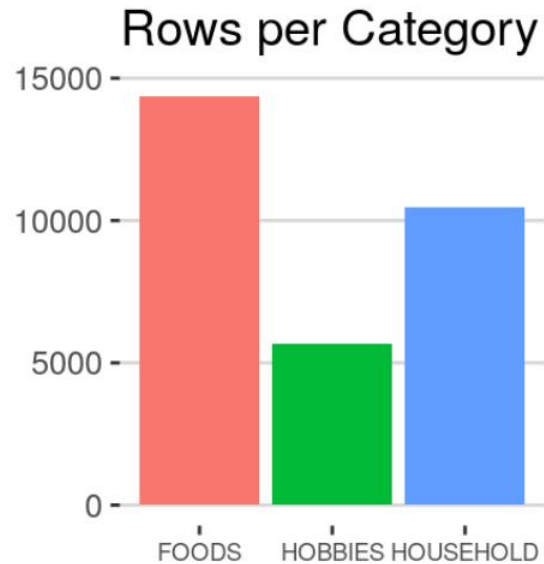
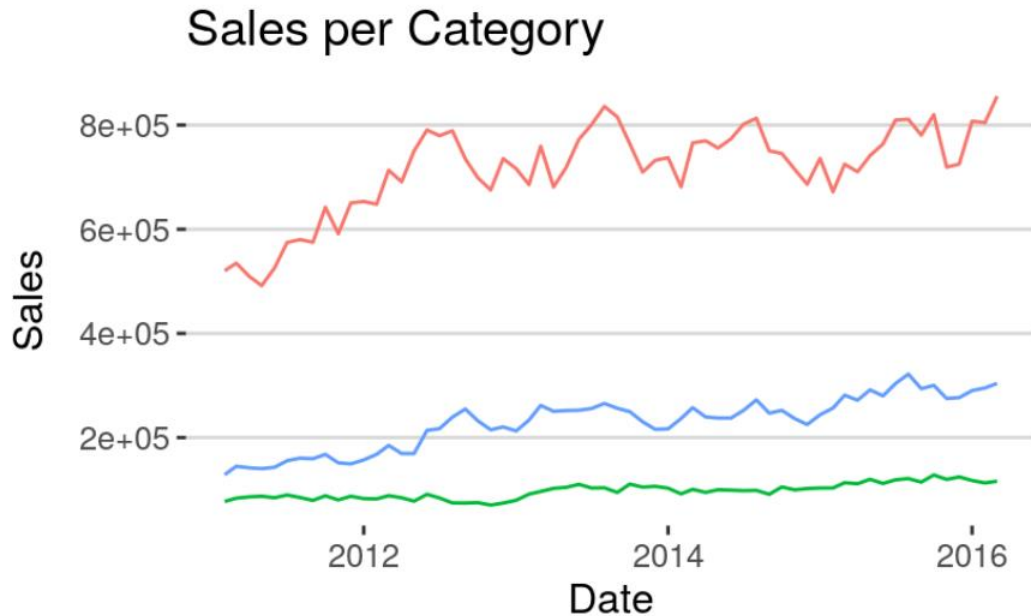
## Sales per Store



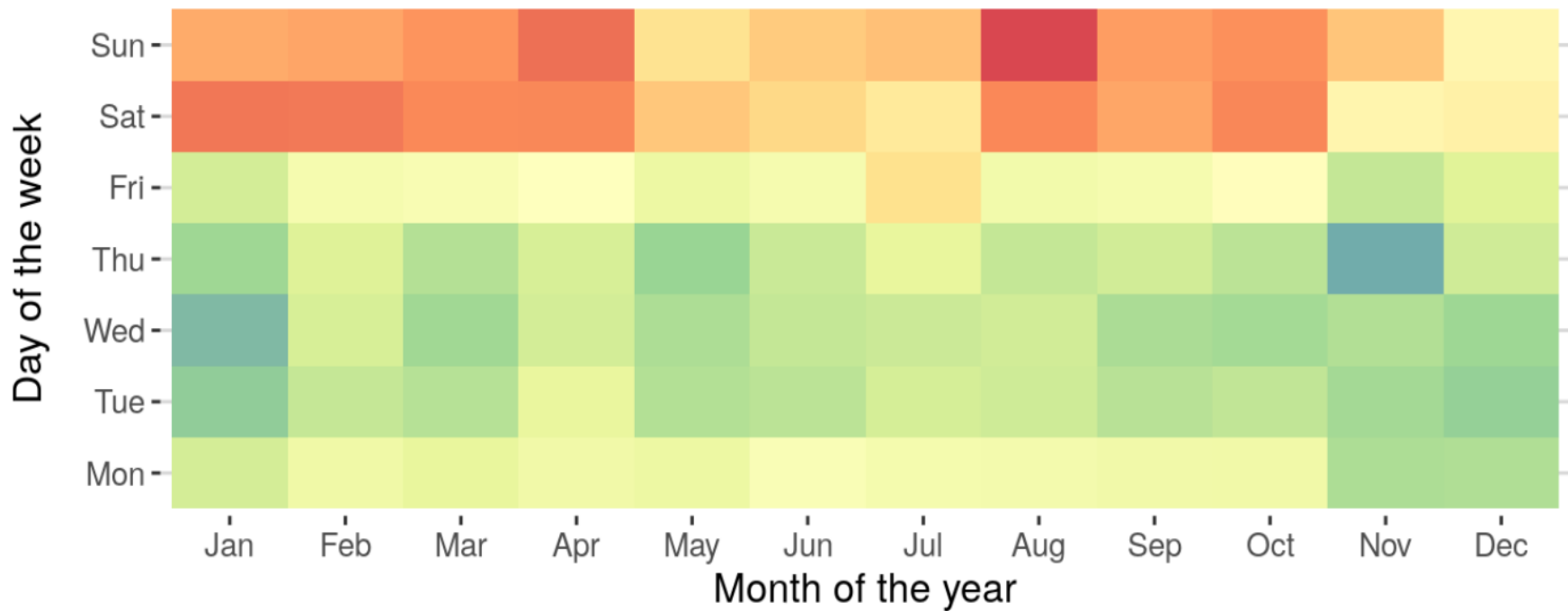
Store ID

CA_1	CA_3	TX_1	TX_3	WI_2
CA_2	CA_4	TX_2	WI_1	WI_3

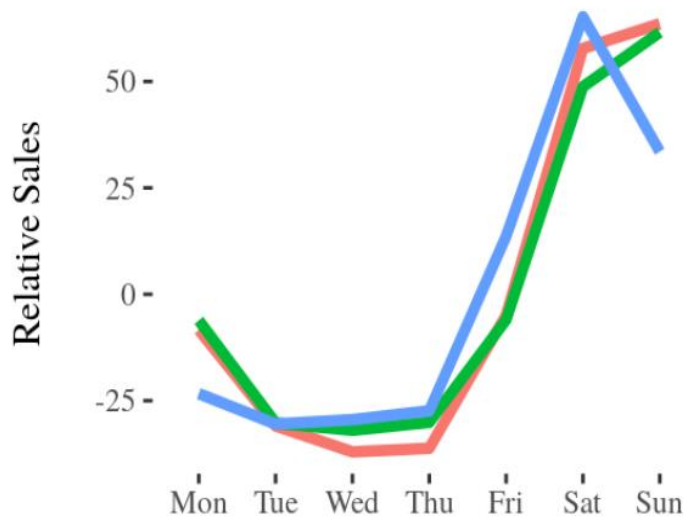




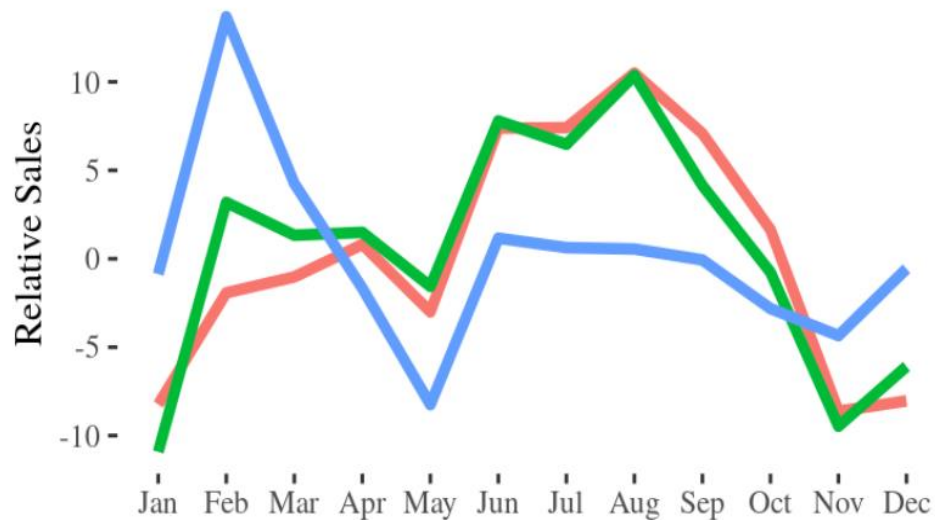
# EDA



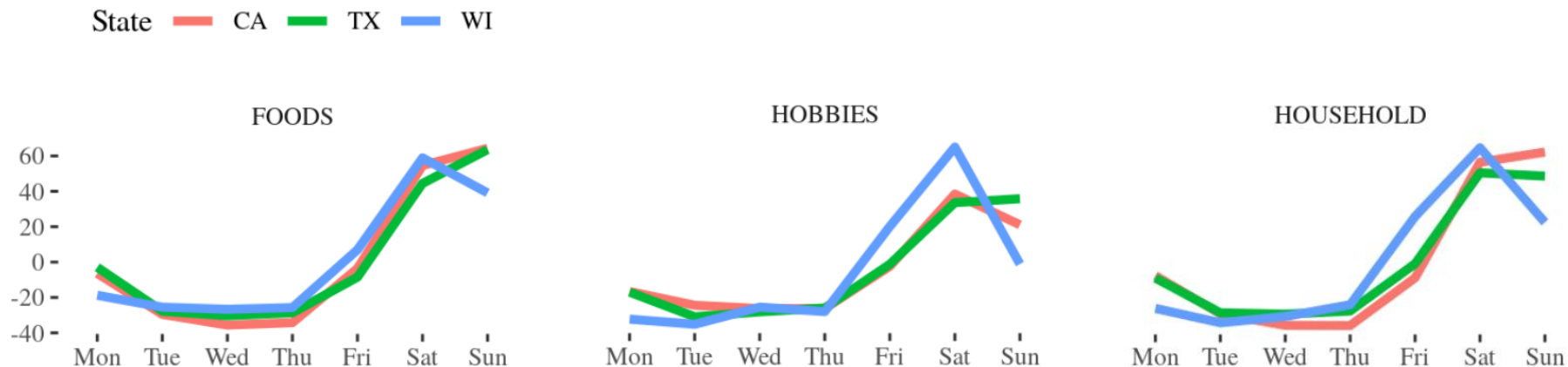
## Weekly Seasonality



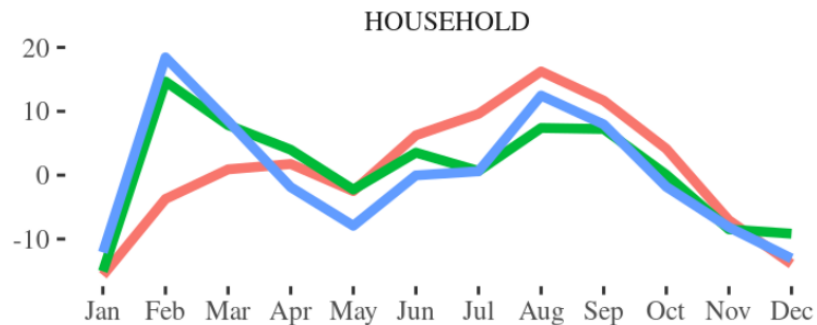
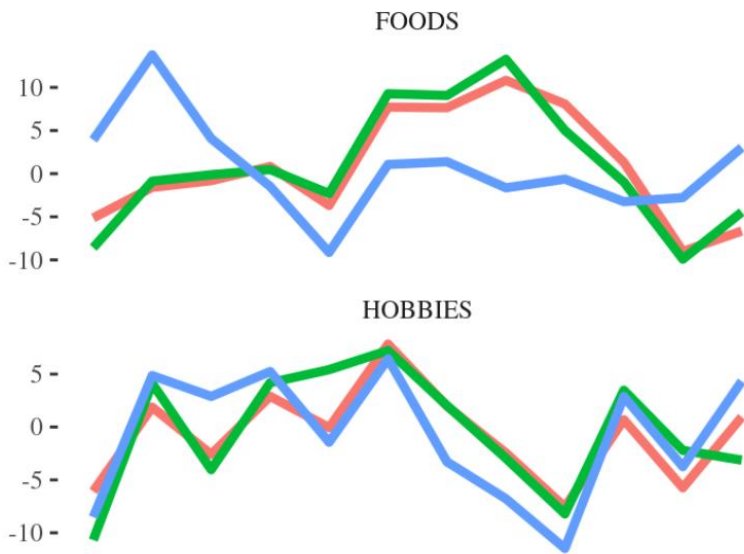
## Monthly Seasonality



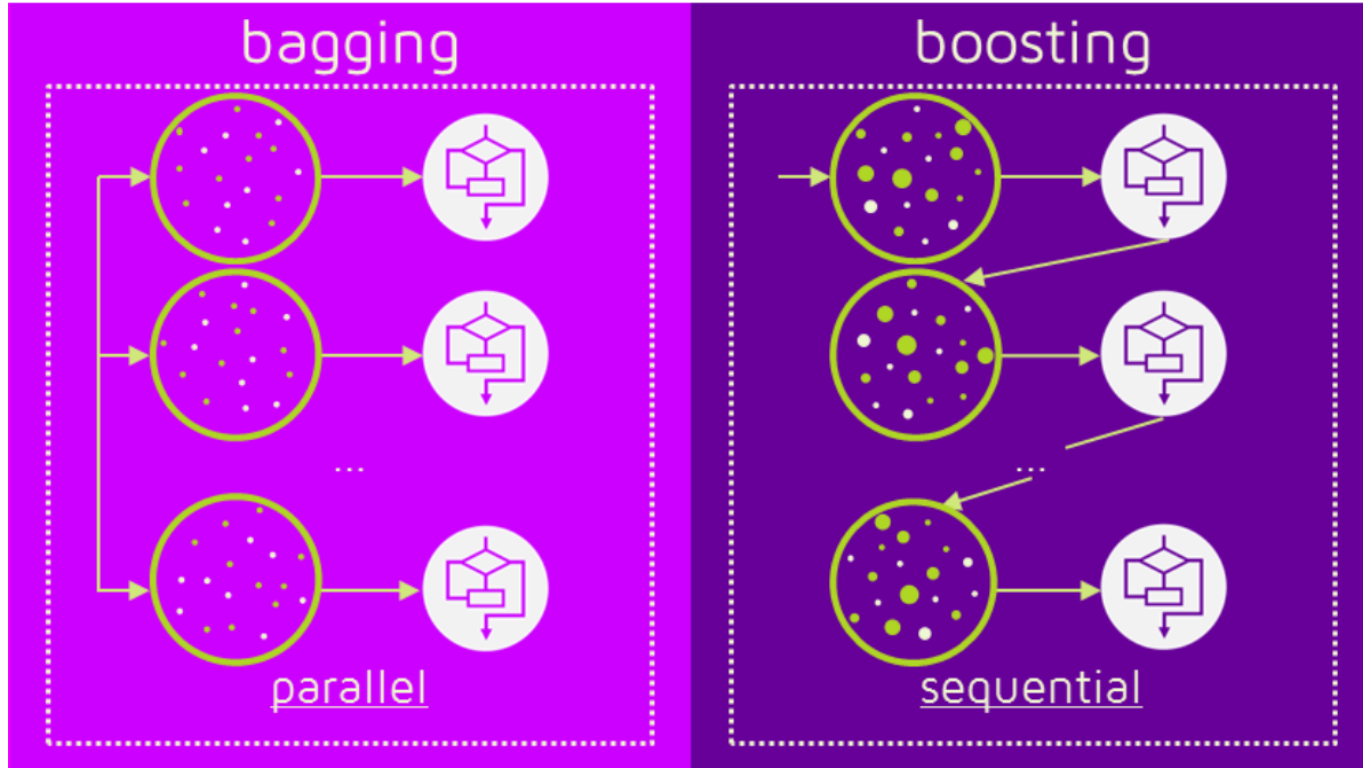
## Weekly Seasonality



## Monthly Seasonality by Category & State



# Boosting



# Boosting

Overfitting 문제 개선 - Bagging

단일 모델의 성능 개선 - Boosting / bias 감소에 초점 - 모수 조정이 중요 / outlier에 취약

- **Ada Boost**

데이터셋에서 샘플을 추출하여 여러 분류기에 적용해서 학습시킨다.  
시행 결과 잘못 분류된 데이터를 집중적으로 학습하여 다음 fitting에 활용.  
Noise나 Outlier가 심한 데이터의 경우, 문제가 발생할 수 있다.

- **Gradient Boost**

Gradient Descent를 Ada Boost에 적용한 기법.  
Outlier, Noise 문제를 해결할 수 있으나, 연산량이 많아짐

- **XG Boost**

Gradient Boost의 많은 연산량을 CPU 분산처리기법 등을 통해 소화하는 방식.

# Light GBM

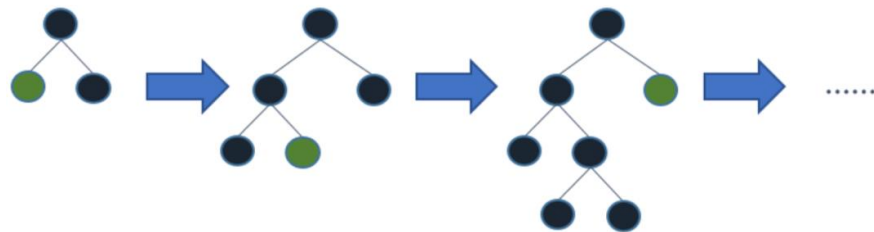
- 학습 속도가 느린 XGBoost의 단점을 보완

GOSS(Gradient-based One-Sided Sampling) / EFB(Exclusive Feature Bundling) 사용

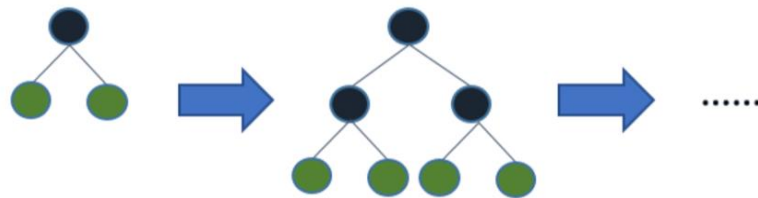
- 대용량 데이터 처리가 가능
- 타 모델에 비해 적은 메모리 사용
- 정보의 손실을 줄일 수 있음

## 리프 중심 트리 분할

: 최대 손실 값을 가지는 리프 노드 지속적 분할



Leaf-wise tree growth



Level-wise tree growth

## 균형 트리 분할



# 모델 적합

	id	item_id	dept_id	store_id	cat_id	state_id	d	sales	date	wm_yr_wk	...	month	year	event_name_1	event_type_1	event_name_2	event_type_2	snap_CA	snap_TX	snap_WI	sell_price
0	HOBBIES_1_002_CA_1_validation	1	0	0	0	0	d_250	0.0	2011-10-05	11136	...	10	2011	0	0	0	0	1.0	1.0	1.0	3.97
1	HOBBIES_1_002_CA_1_validation	1	0	0	0	0	d_251	0.0	2011-10-06	11136	...	10	2011	0	0	0	0	1.0	1.0	1.0	3.97
2	HOBBIES_1_002_CA_1_validation	1	0	0	0	0	d_252	0.0	2011-10-07	11136	...	10	2011	0	0	0	0	1.0	1.0	0.0	3.97
3	HOBBIES_1_004_CA_1_validation	3	0	0	0	0	d_250	0.0	2011-10-05	11136	...	10	2011	0	0	0	0	1.0	1.0	1.0	4.34
4	HOBBIES_1_004_CA_1_validation	3	0	0	0	0	d_251	4.0	2011-10-06	11136	...	10	2011	0	0	0	0	1.0	1.0	1.0	4.34

5 rows × 22 columns

# 모델 적합

## 변수 추가

lag_7	lag_28	rmean_7_7	rmean_28_7	rmean_7_28	rmean_28_28	week	quarter	mday
0.0	0.0	0.000000	0.142857	0.178571	0.285714	48	4	29
0.0	0.0	0.000000	0.000000	0.142857	0.285714	48	4	30
0.0	1.0	0.000000	0.142857	0.142857	0.321429	48	4	1
0.0	0.0	0.000000	0.142857	0.142857	0.321429	48	4	2
1.0	1.0	1.428571	1.428571	1.607143	1.785714	48	4	29

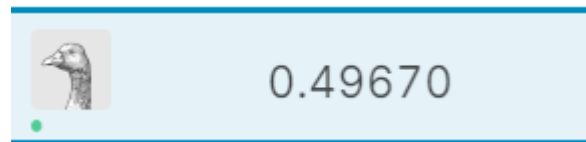
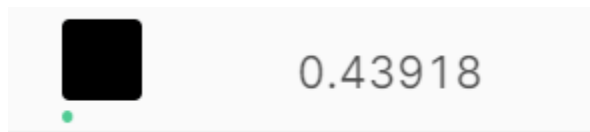
# 모델 적합

---

```
params = {  
    "objective" : "poisson", #poisson regression  
    "metric" : "rmse",  
    "force_row_wise" : True, #learning direction (relatively small columns)  
    "learning_rate" : 0.075,  
    "sub_row" : 0.75,  
    "bagging_freq" : 1, #0 means disable bagging; k means perform bagging at every k iteration  
    "lambda_l2" : 0.1, #L2 regularization  
    "metric": ["rmse"],  
    'verbosity': 1, # Info  
    'num_iterations' : 1200,  
    'num_leaves' : 128,  
    "min_data_in_leaf" : 100,  
}
```

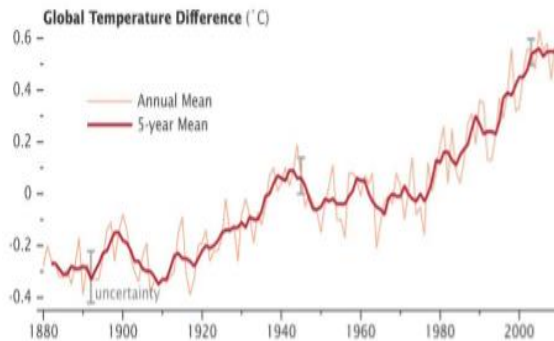
# 결과

	id	F1	F2	F3	F4	F5	F6	F7	F8	F9	...	F19	F20	F21	F22
0	FOODS_1_001_CA_1_validation	0.900208	0.850305	0.835608	0.808511	1.068399	1.240130	1.336629	1.036806	1.005700	...	1.092689	1.365811	1.295096	1.031251
1	FOODS_1_001_CA_2_validation	0.826054	0.846310	0.782934	1.054960	1.074849	1.252265	1.328782	0.880883	0.909217	...	1.181405	1.557242	1.386192	0.970276
2	FOODS_1_001_CA_3_validation	1.224390	1.119274	0.986613	0.970576	1.063056	1.244171	1.236825	1.119709	1.136489	...	1.119759	1.659144	1.779672	1.205213
3	FOODS_1_001_CA_4_validation	0.401395	0.349750	0.346810	0.351312	0.421821	0.429025	0.513803	0.390084	0.421533	...	0.446922	0.482879	0.506504	0.376197
4	FOODS_1_001_TX_1_validation	0.220046	0.218440	0.224843	0.227519	0.220456	0.218982	0.261482	0.561683	0.514396	...	0.410909	0.429856	0.445446	0.311021
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
60975	HOUSEHOLD_2_516_TX_2_evaluation	0.283826	0.260213	0.288633	0.284374	0.353788	0.432488	0.343618	0.238760	0.241598	...	0.292015	0.364561	0.354640	0.247517
60976	HOUSEHOLD_2_516_TX_3_evaluation	0.167784	0.155329	0.171575	0.161759	0.198939	0.226477	0.171531	0.124789	0.121023	...	0.161521	0.179582	0.165588	0.139283
60977	HOUSEHOLD_2_516_WI_1_evaluation	0.091761	0.084142	0.083611	0.091436	0.103181	0.113265	0.104594	0.096435	0.092222	...	0.129596	0.143583	0.136628	0.099333
60978	HOUSEHOLD_2_516_WI_2_evaluation	0.042751	0.040404	0.040279	0.087703	0.106358	0.104421	0.097441	0.090797	0.090362	...	0.133645	0.132977	0.121293	0.105357
60979	HOUSEHOLD_2_516_WI_3_evaluation	0.054448	0.052615	0.052839	0.057376	0.074295	0.062670	0.062580	0.100624	0.098897	...	0.144387	0.149870	0.138397	0.115388

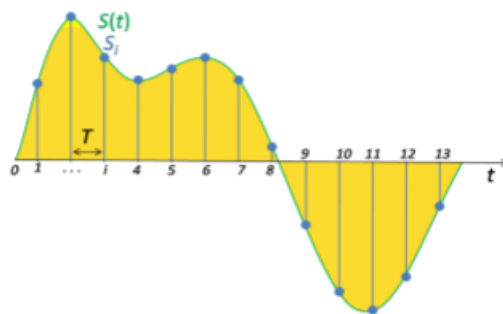


# 순차 데이터 (Sequential Data)

순서가 의미를 가지며, 순서가 달라질 경우 의미가 손상되는 데이터



세계 기온 변화  
(Temporal Sequence)



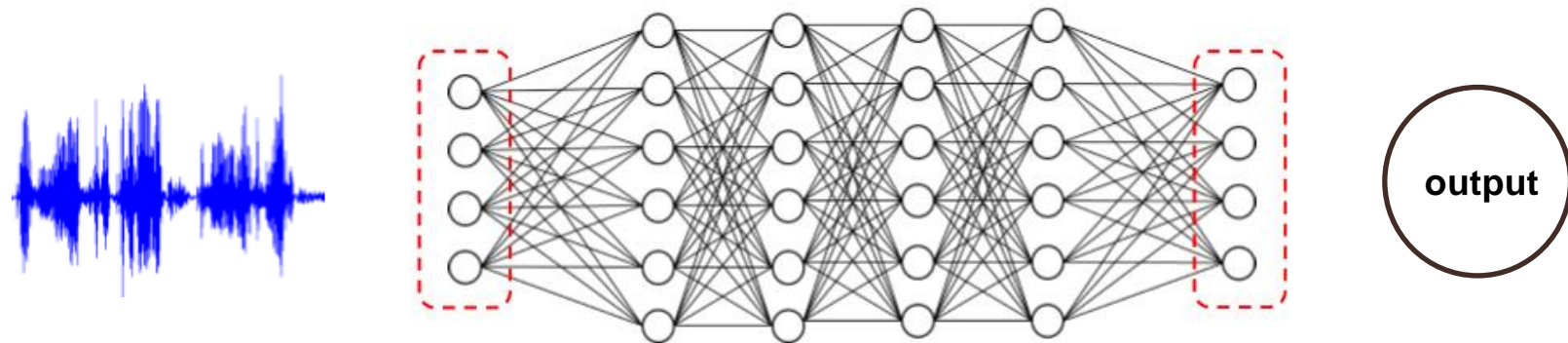
샘플링된 소리 신호  
(Time series)



주식 가격 시계열 데이터  
(Time series)

시간적 의미가 있는 경우 Temporal Sequence, **일정한 시간차라면 Time Series**

# 심층 신경망과 순차 데이터



**Fixed-Length Vector?**

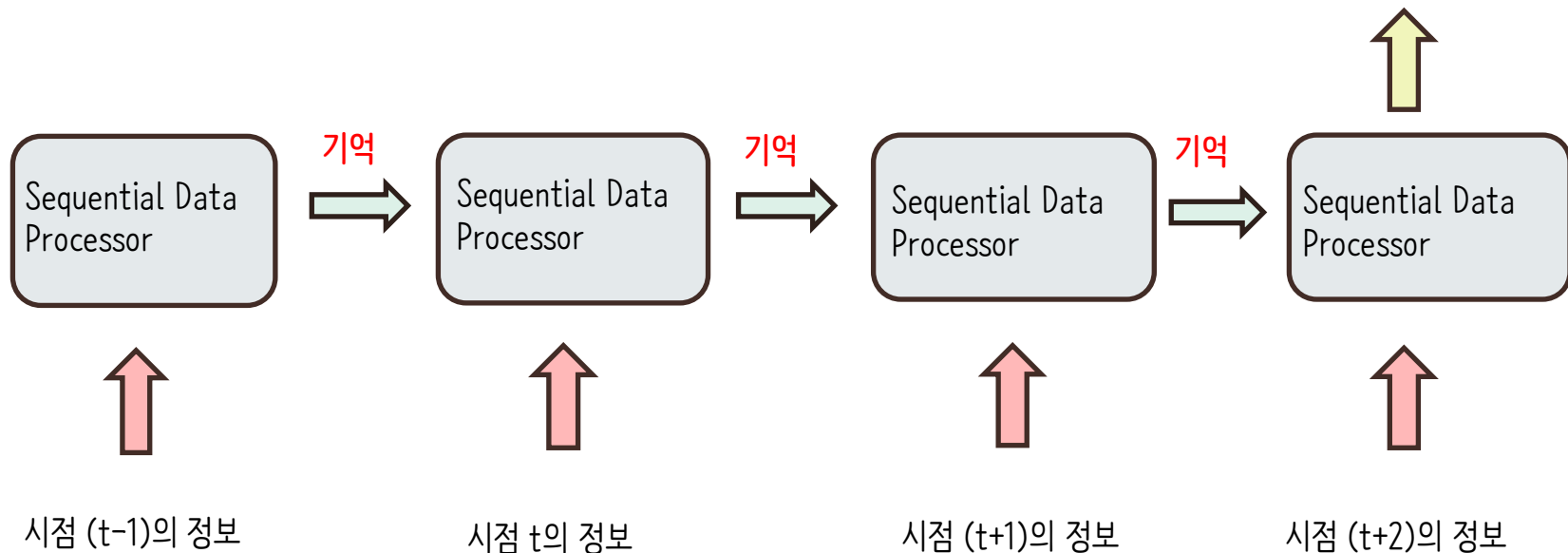
**입력되는 시계열의 길이는 매번 다름**

**데이터는 시간 순서에 의존적**

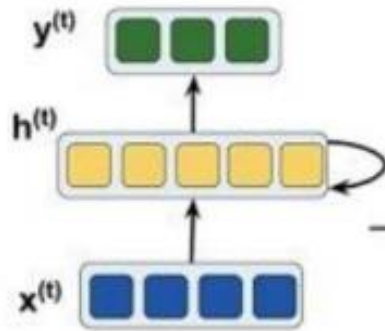
**One-Hot vector**

**무수히 많은 클래스가 필요**

# 기억 시스템(Memory System)



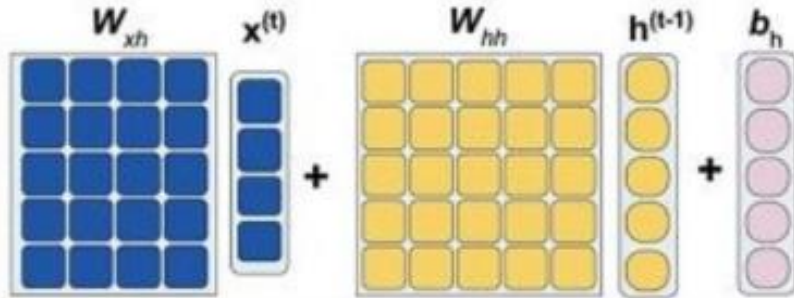
올바른 출력을 내려면, 입력을 받을 때 마다 그 내용을 '기억'할 수 있어야 한다.



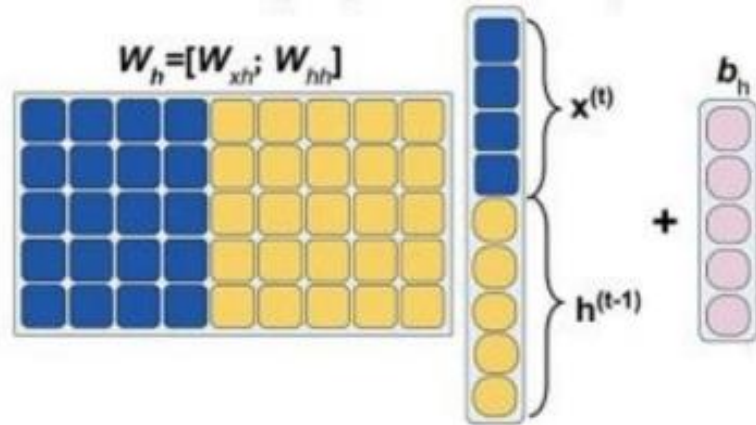
최종 출력:  

$$y^{(t)} = \phi_y(W_{hy} h^{(t)} + b_y)$$

공식 1:  $h^{(t)} = \phi_h(W_{xh} x^{(t)} + W_{hh} h^{(t-1)} + b_h)$

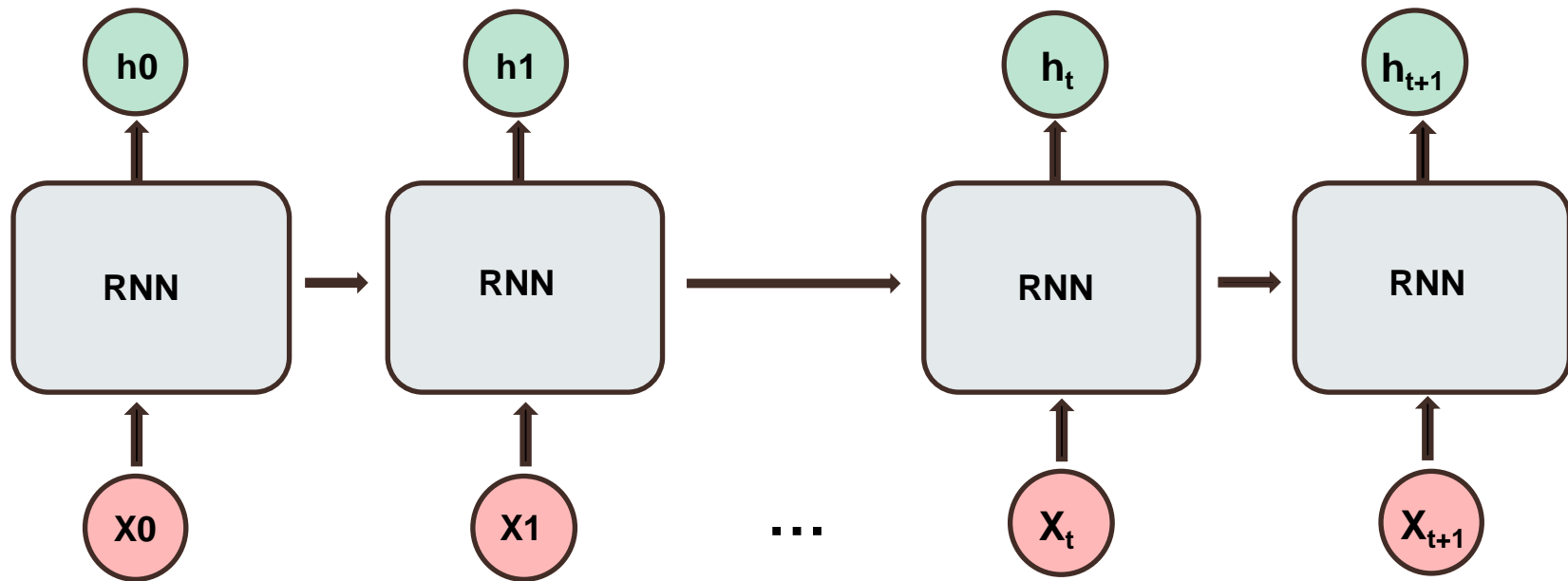


공식 2:  $h^{(t)} = \phi_h(W_h [x^{(t)}; h^{(t-1)}]^T + b_h)$

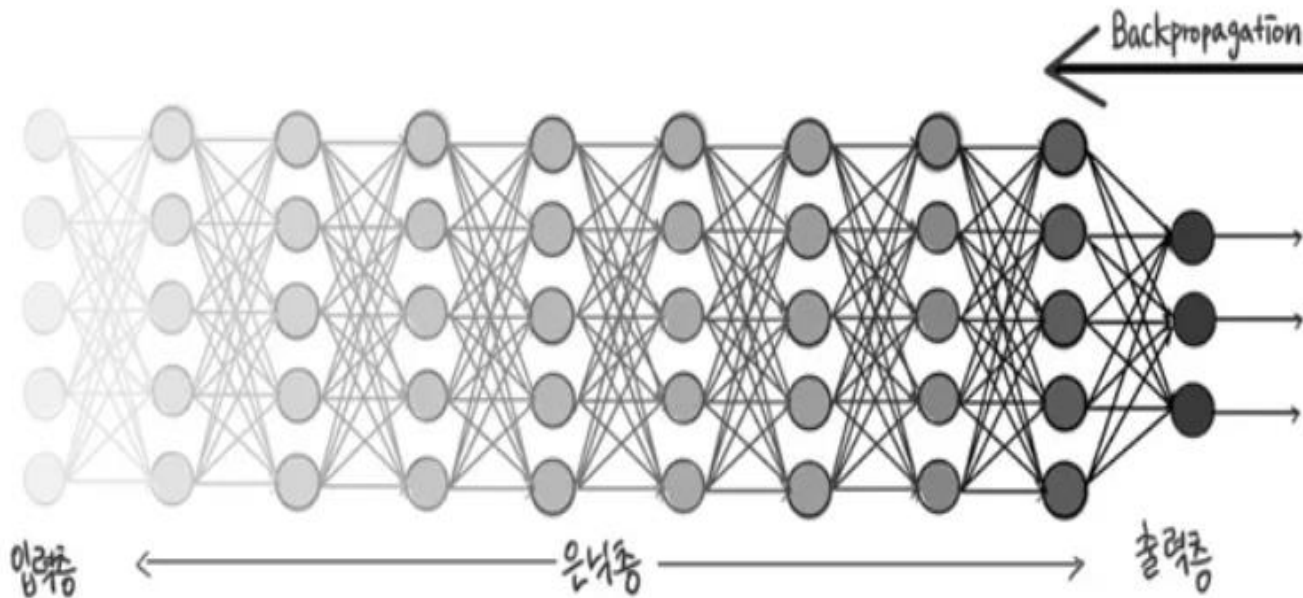




# 순환 신경망 (RNN)의 한계



어떤 입력의 정보가 **사용되는 시점의 차이가 많이 날 경우**, 학습능력이 저하된다.



장기 기억에 있어 취약

오래된 시점의 노드가 담고 있는 정보, 신호가 제대로 전파되지 않음 (정보 손실)

# Back Propagation Through Time

$$\frac{\partial L^{(t)}}{\partial \mathbf{w}_{hh}} = \frac{\partial L^{(t)}}{\partial \mathbf{y}^{(t)}} \times \frac{\partial \mathbf{y}^{(t)}}{\partial \mathbf{h}^{(t)}} \times \left( \sum_{k=1}^t \boxed{\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}}} \times \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{w}_{hh}} \right) \quad \bullet \quad \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(k)}} = \prod_{i=k+1}^t \frac{\partial \mathbf{h}^{(i)}}{\partial \mathbf{h}^{(i-1)}}$$

전체 손실  $L$ 은  $t=T$ 까지 각 타임 스텝의 모든 손실 함수의 합

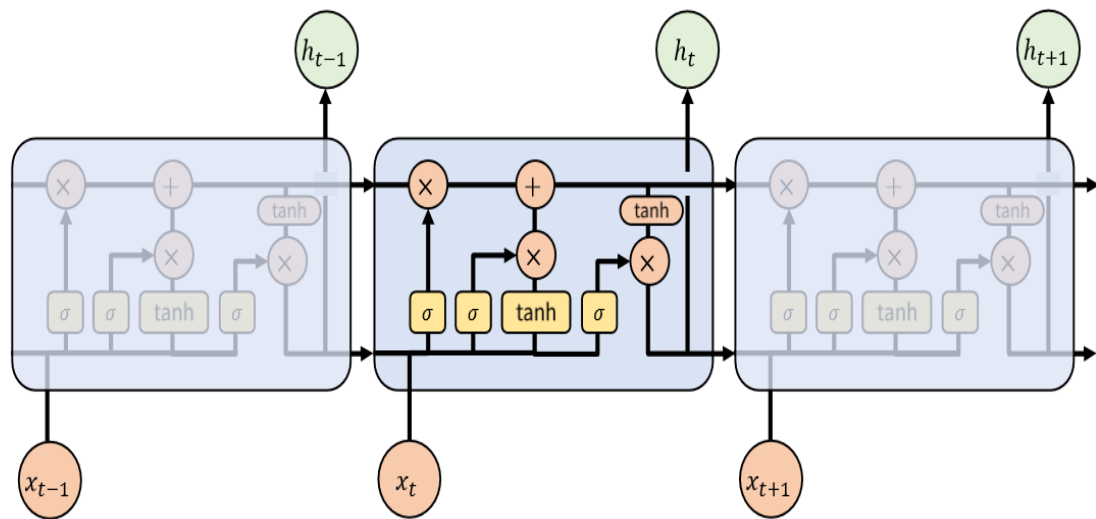
타임 스텝  $t$ 에서의 손실은 모든 이전 타임 스텝  $1 \sim (t-1)$ 의 은닉층에 의존

기울기 소실 / 폭주 문제

$(t-k)$ 개의 곱셈 : 가중치  $w$ 가  $(t-k)$ 번 곱해져  $w_{(t-k)}$ 가 된다

$|w| > 1$  : 폭주 /  $|w| < 1$  : 소실

# Long Short-Term Memory



다음 층으로 기억된 값을 넘길 지, 넘기지 않을 지를 관리

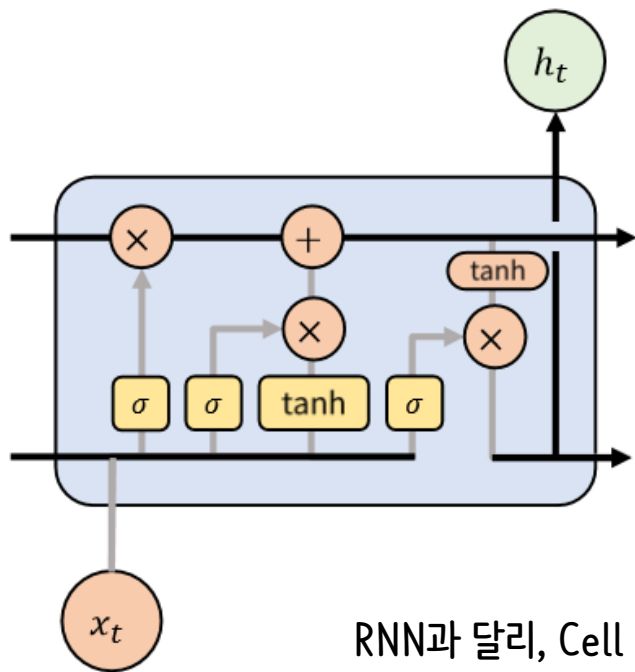
과거의 신호를 전파

기울기 소실 문제를 극복

기억할 것은 오래 기억하고, 잊을 것은 빨리 잊어버리는 능력

삭제, 입력, 출력 게이트로 구성

# Cell, Hidden State

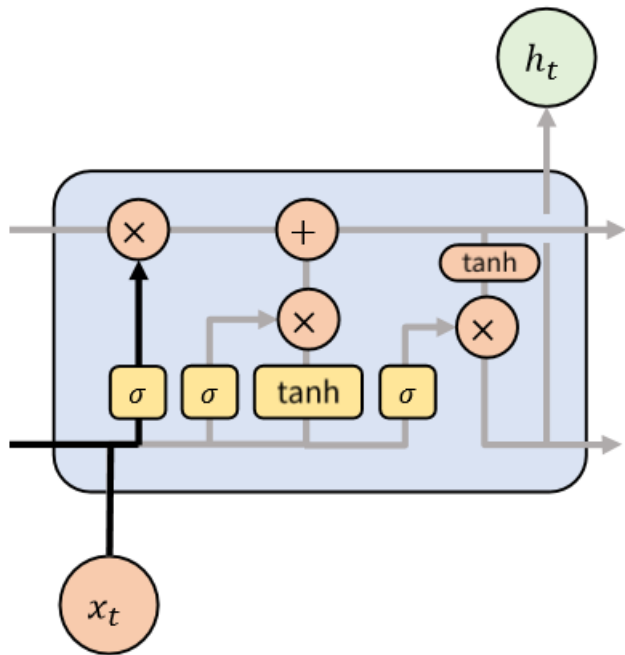


Cell State : 기억을 오랫동안 유지할 수 있는 구조  
새로운 특징을 덧셈으로 받음

Hidden State : 계층의 출력  
다음 타임 스텝으로 넘기는 정보

RNN과 달리, Cell State가 있어서 '기억'에 관한 부분을 전담

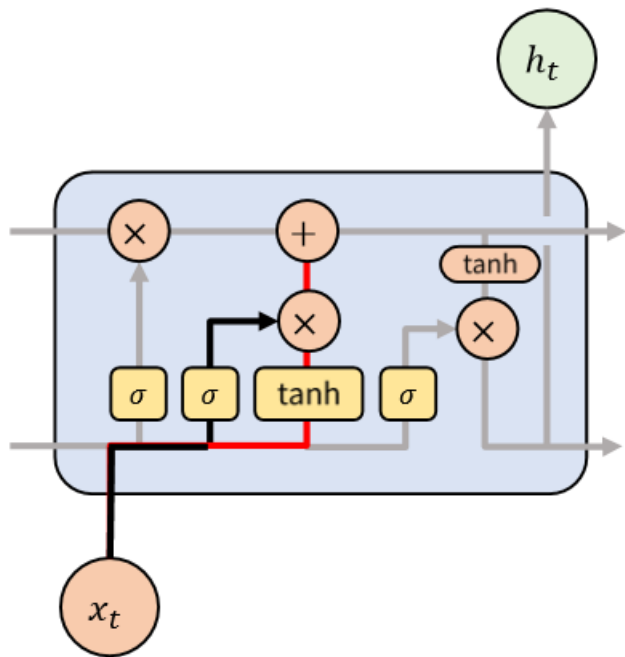
# Forget Gate



## Forget Gate

- Sigmoid 활성화 함수, 0~1의 값
- Cell State에 그 값을 곱해주어서 **얼만큼 잊을지** 결정
- 메모리 셀이 무한정 성장하지 않도록 셀 상태 설정
- **통과할 정보, 억제할 정보**를 결정

# Input Gate



## Input Gate

- Sigmoid 활성화 함수, 0~1의 출력값
- 새롭게 추출한 특징을 얼마나 사용할 지 결정
- 셀 상태를 업데이트

$\mathbf{f}_t = \sigma(\mathbf{w}_{xf} \mathbf{x}^{(t)} + \mathbf{w}_{hf} \mathbf{h}^{(t-1)} + \mathbf{b}_f)$  : 셀 상태를 다시 설정

$\mathbf{i}_t = \sigma(\mathbf{w}_{xi} \mathbf{x}^{(t)} + \mathbf{w}_{hi} \mathbf{h}^{(t-1)} + \mathbf{b}_i)$

$\mathbf{g}_t$  : 입력 노드

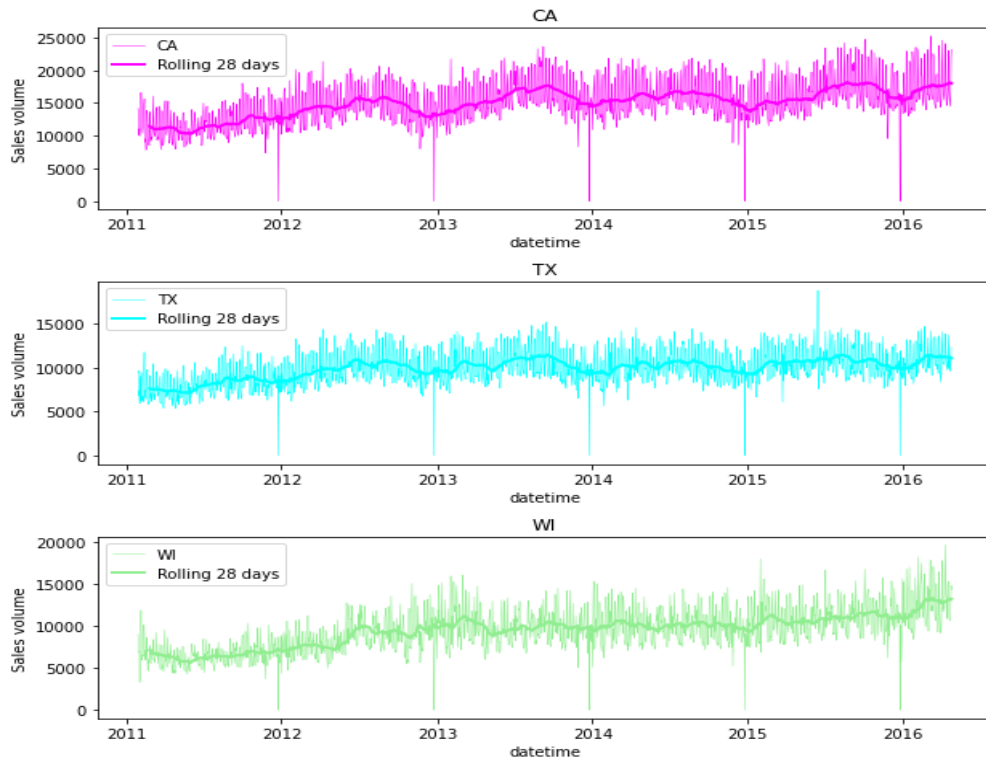
$\mathbf{C}^{(t)} = (\mathbf{C}^{(t-1)} \odot \mathbf{f}_t) + (\mathbf{i}_t \odot \mathbf{g}_t)$  : t에서의 셀 상태

$\mathbf{O}_t = \sigma(\mathbf{w}_{xo} \mathbf{x}^{(t)} + \mathbf{w}_{ho} \mathbf{h}^{(t-1)} + \mathbf{b}_o)$  : 은닉 유닛의 출력 업데이트

$\mathbf{h}_t = \mathbf{O}_t \odot \tanh(\mathbf{C}^{(t)})$  : 현재 타임스텝의 은닉 유닛의 출력



# Moving Average for Stationary



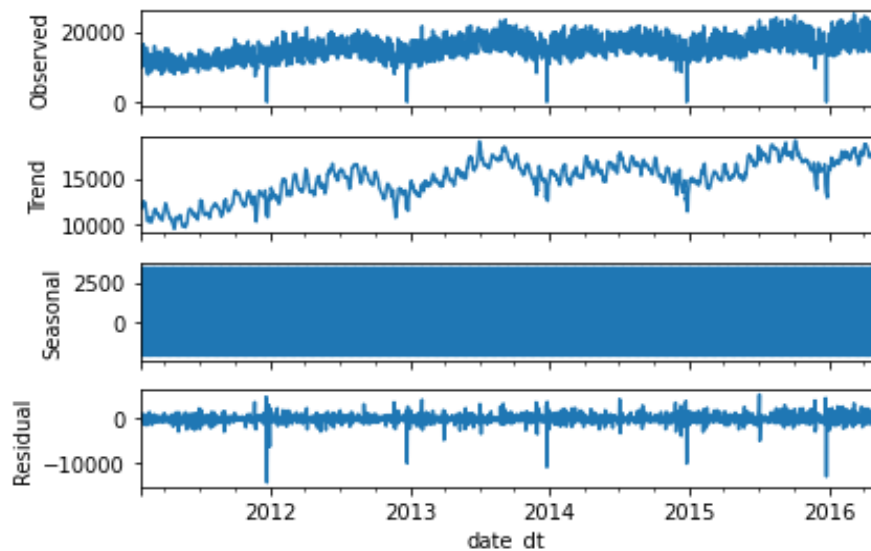
```
state_group[state_col[i]].rolling(28).mean()  
>> 28일 간격의 이동평균
```

```
from statsmodels.tsa.seasonal import seasonal_decompose

result_CA = seasonal_decompose(state_group['CA'])

fig = plt.figure(figsize=(20,15))
fig = result_CA.plot()
```

<Figure size 1440x1080 with 0 Axes>



$$y_t = S_t + T_t + R_t$$

$y_t$  : data

$S_t$  : 계절성분

$T_t$  : 추세-주기 성분

$R_t$  : 나머지 성분 (Residual)

# Data

```
store_group_minmax_scaled.head()
```

	date	CA_1	CA_2	CA_3	CA_4	TX_1	TX_2	TX_3	WI_1	WI_2	...	month	year	d	event_name_1	event_type_1	event
0	2011-01-29	0.478530	<a href="#">0.434718</a>	<a href="#">0.295064</a>	<a href="#">0.326010</a>	<a href="#">0.297974</a>	<a href="#">0.558541</a>	<a href="#">0.277108</a>	0.391594	<a href="#">0.219526</a>	...	1	2011	d_1	NaN	NaN	
1	2011-01-30	0.442181	0.351632	0.308553	0.385641	<a href="#">0.332889</a>	0.571300	<a href="#">0.272445</a>	<a href="#">0.317681</a>	<a href="#">0.172943</a>	...	1	2011	d_2	NaN	NaN	
2	2011-01-31	<a href="#">0.174755</a>	<a href="#">0.180082</a>	0.148835	<a href="#">0.232248</a>	<a href="#">0.102345</a>	0.390273	0.120676	<a href="#">0.226087</a>	<a href="#">0.186332</a>	...	1	2011	d_3	NaN	NaN	
3	2011-02-01	<a href="#">0.221690</a>	<a href="#">0.217730</a>	<a href="#">0.217351</a>	<a href="#">0.253433</a>	<a href="#">0.218550</a>	<a href="#">0.423747</a>	<a href="#">0.109794</a>	<a href="#">0.181014</a>	<a href="#">0.256625</a>	...	2	2011	d_4	NaN	NaN	
4	2011-02-02	0.137607	0.146884	0.153740	<a href="#">0.291095</a>	0.068230	0.354398	0.023708	0.000000	0.068759	...	2	2011	d_5	NaN	NaN	

```
target_store = distinct_store_names[x]

# scaled data 설정
data_scaled = store_group_minmax_scaled[target_store].ravel().reshape(-1,1)

# train, test split
train_data, test_data = data_scaled[:len(data_scaled) - 56], data_scaled[len(data_scaled) - 56 : ]
```

# 시계열 Generator

```
n_input = 28
```

```
n_features=1
```

```
train_generator = TimeseriesGenerator(train_data, train_data, length=n_input, batch_size=1)
test_generator = TimeseriesGenerator(test_data, test_data, length=n_input, batch_size=1)
```

28개의 과거 시점 매출액으로 현 시점의  
매출액을 예측

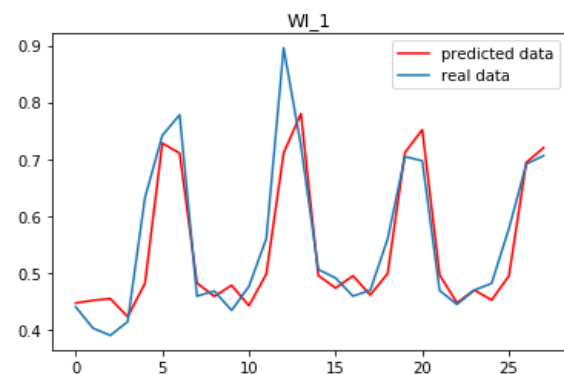
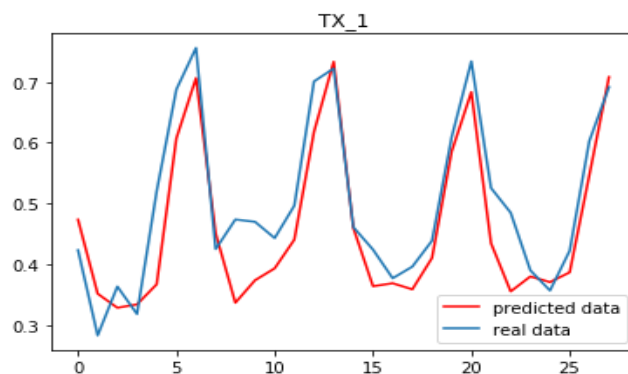
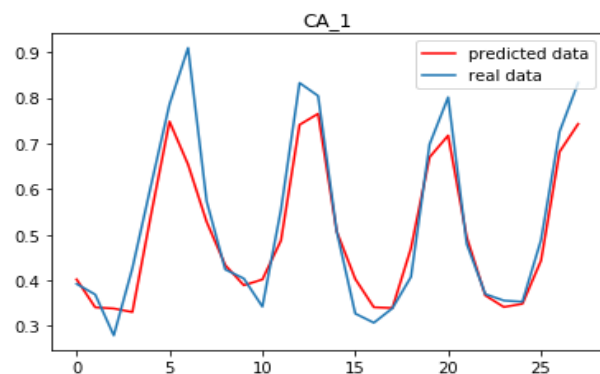
$(y_{t-1}, y_{t-2}, \dots, y_{t-28})$  : 특성변수 >>> 은닉층 >>>  $y_t$  : 목적변수

```

model = Sequential()
model.add(LSTM(4, activation = 'tanh', input_shape=(n_input, 1)))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='adam')
model.summary()

history = model.fit_generator(train_generator, epochs=32).history
yhat = model.predict(test_generator)
print(y_hat)

```




# 최종 발표 전 계획

---

**1. 시계열 분해를 통한 예측**

**2. Stacking Model**

**3. 변수 및 hyper-parameter 조정**

A stylized illustration of a person from the chest up, wearing a grey suit jacket, a white shirt, and a dark tie. The person's face is partially visible, showing a large, open mouth with a red tongue. A large, black-outlined speech bubble originates from the mouth. Inside the speech bubble, the text "Do you have any question?" is written. The word "question?" is in a larger, pink font, while "Do you have any" is in a smaller, grey font. In the bottom right corner of the image, the text "Thank you for your attention." is written in a white, sans-serif font.

Do you  
have any  
question?

Thank you  
for your attention.