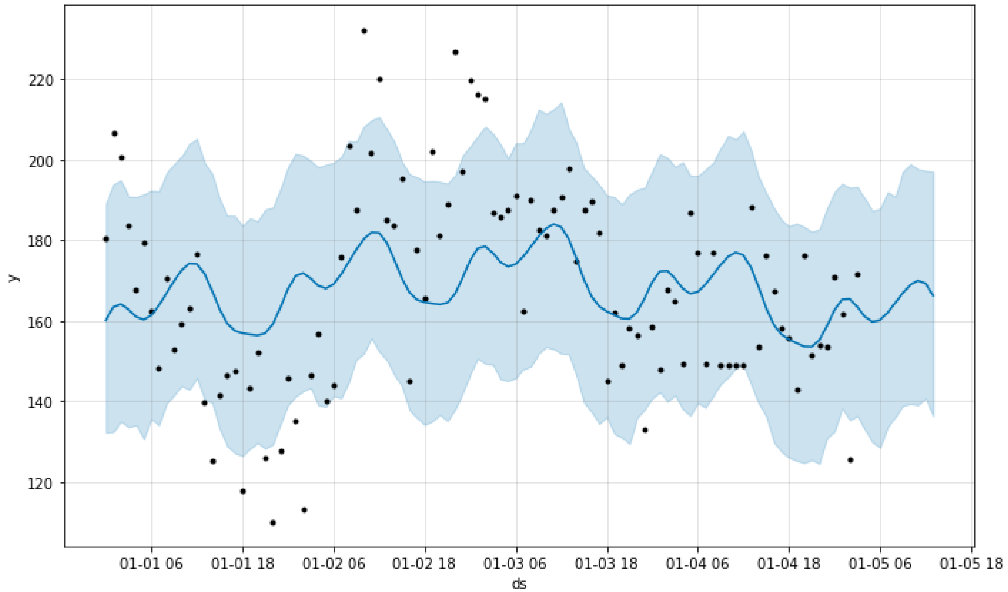
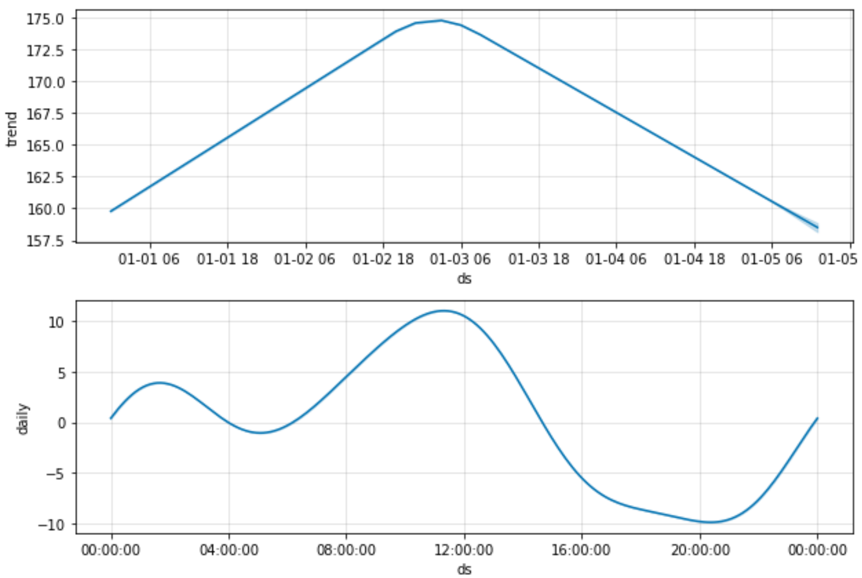
Multistep prediction

Related information

- Time series usually have yearly seasonality, weekly seasonality and daily seasonality
- *y_hat = trend + [[daily_seasonality] +[weekly_seasonality] + [yearly_seasonality]]*
- Example:

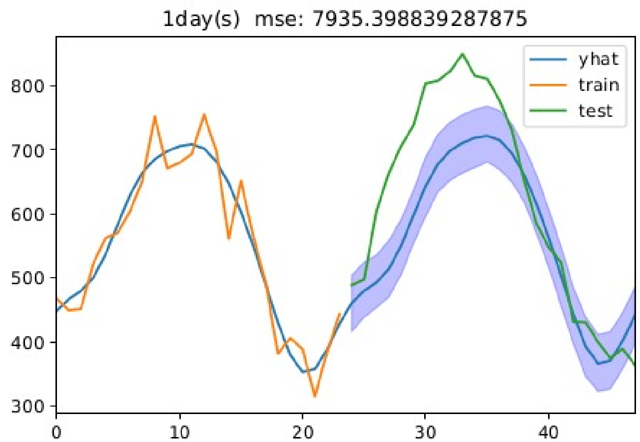


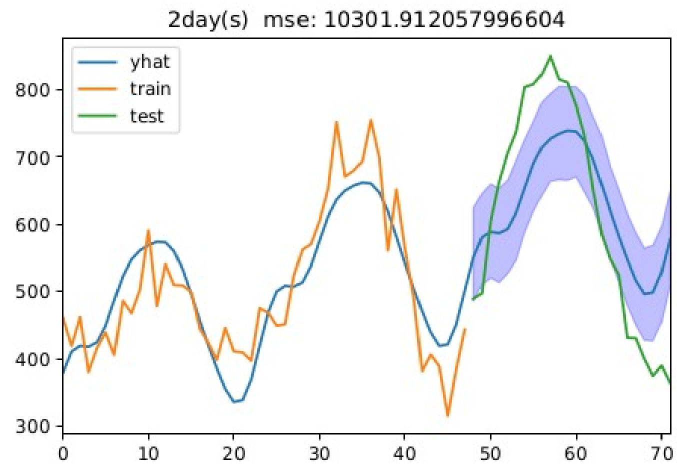


- Tool: *Prophet https://facebook.github.io/prophet/docs/quick_start.html*

Overview of my ideas

- ***N*** period points before current point ***C*** must have a trend,using this trend to predict ***M*** period points later (***trend(N)-->valueOf(M)***)
- Set ***lowerN*** and ***upperN*** , then let N differs in [***lowerN, upperN***], different trends can be get, so we get multi ***valueOf(M)*** sequences.
- Give the ***valueOf(M)*** ***weights***, make weighted summation, finally got the ***hat_valueOf(M)***





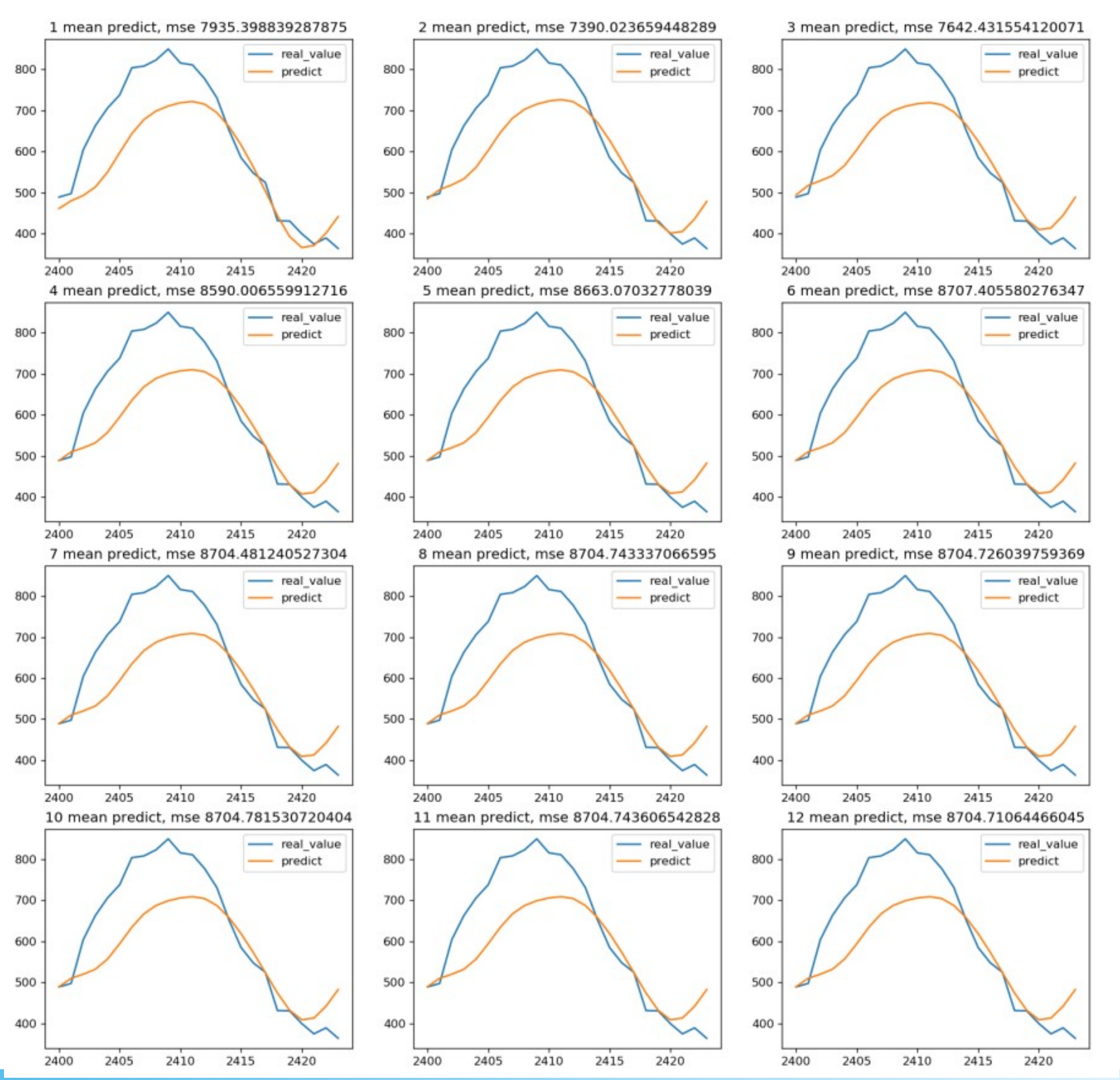
- More>>

Pseudocode

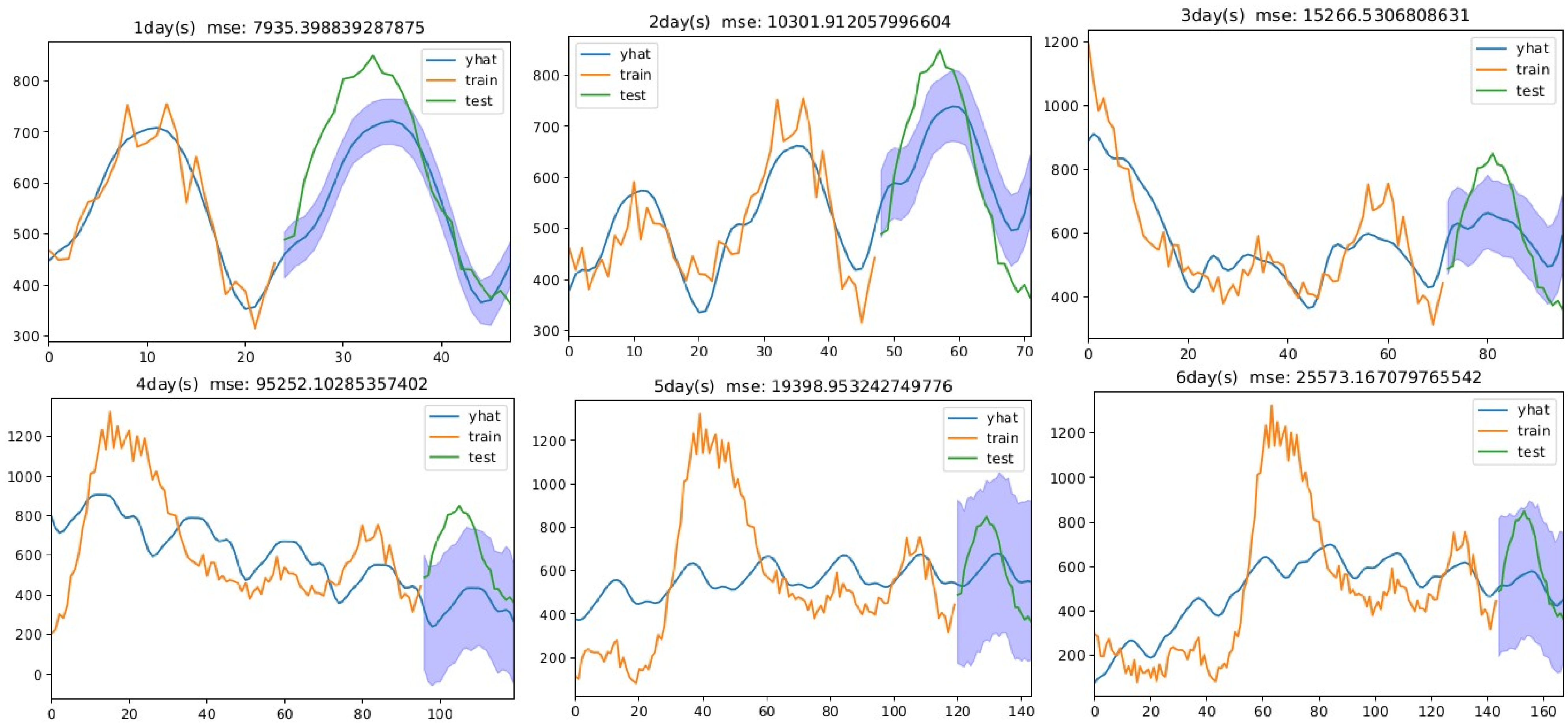
- **For *N* in** range(***lowerN***, ***upperN***, step):
  - ValueOfM_pre_list = []
  - Trend = get_trend(***N***_points_sequence)
  - ValueOfM_pre = next_M_points_predict(Trend, ***C***, ***M***)
  - ValueOfM_pre_list.append(ValueOfM_pre)
- **For *weight* in** weights, ValueOfM_pre in ValueOfM_pre_list:
  - ValueOfM_hat += weight * ValueOfM_pre
- **Finally** get the predict of next ***M*** periods values ValueOfM_hat

Result obtained 1

- *C* = 2015-04-11 00:00:00
- *N* in range([24,24*12+1,24])
- *M* = 24



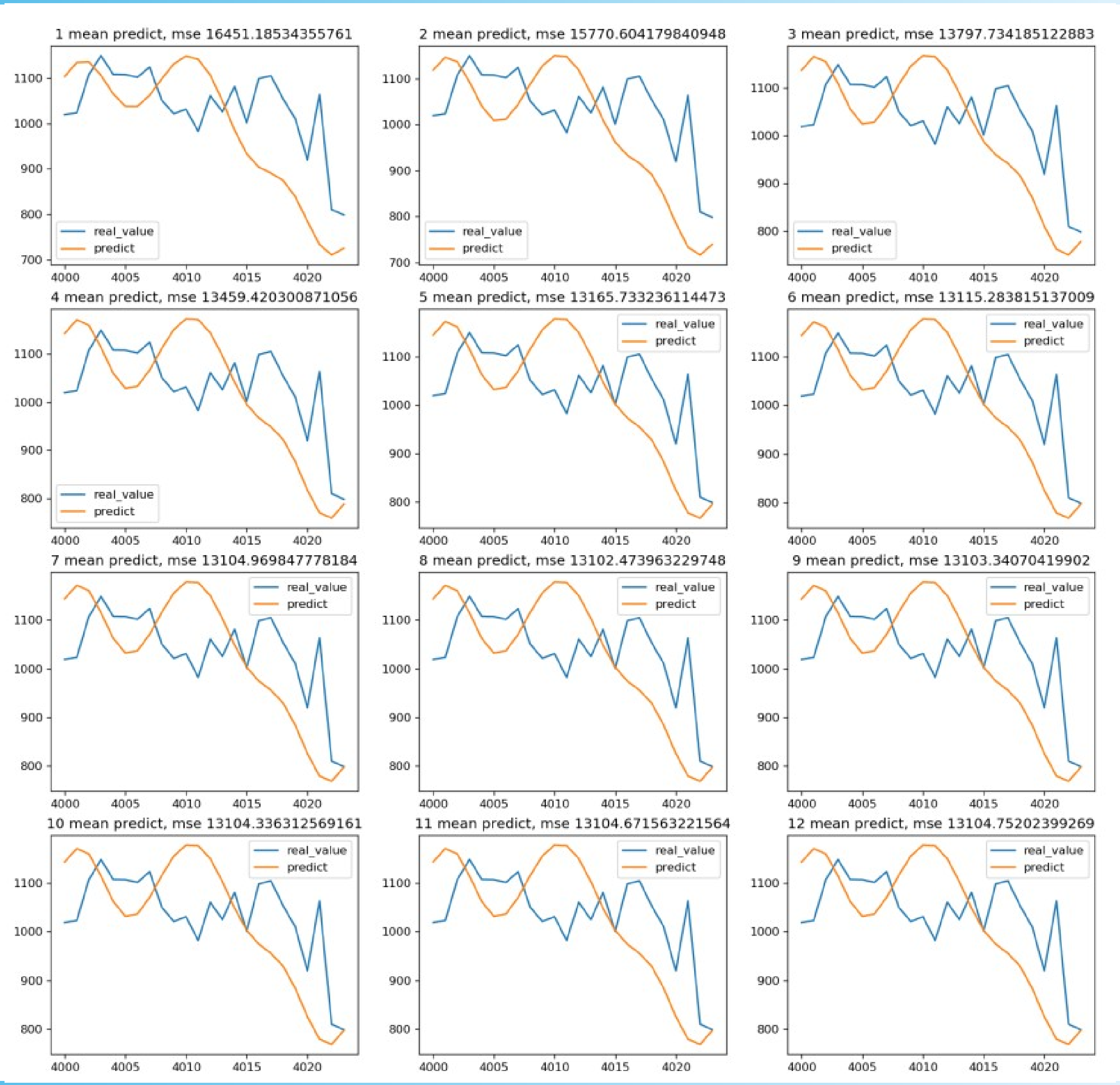
Result obtained 1 details



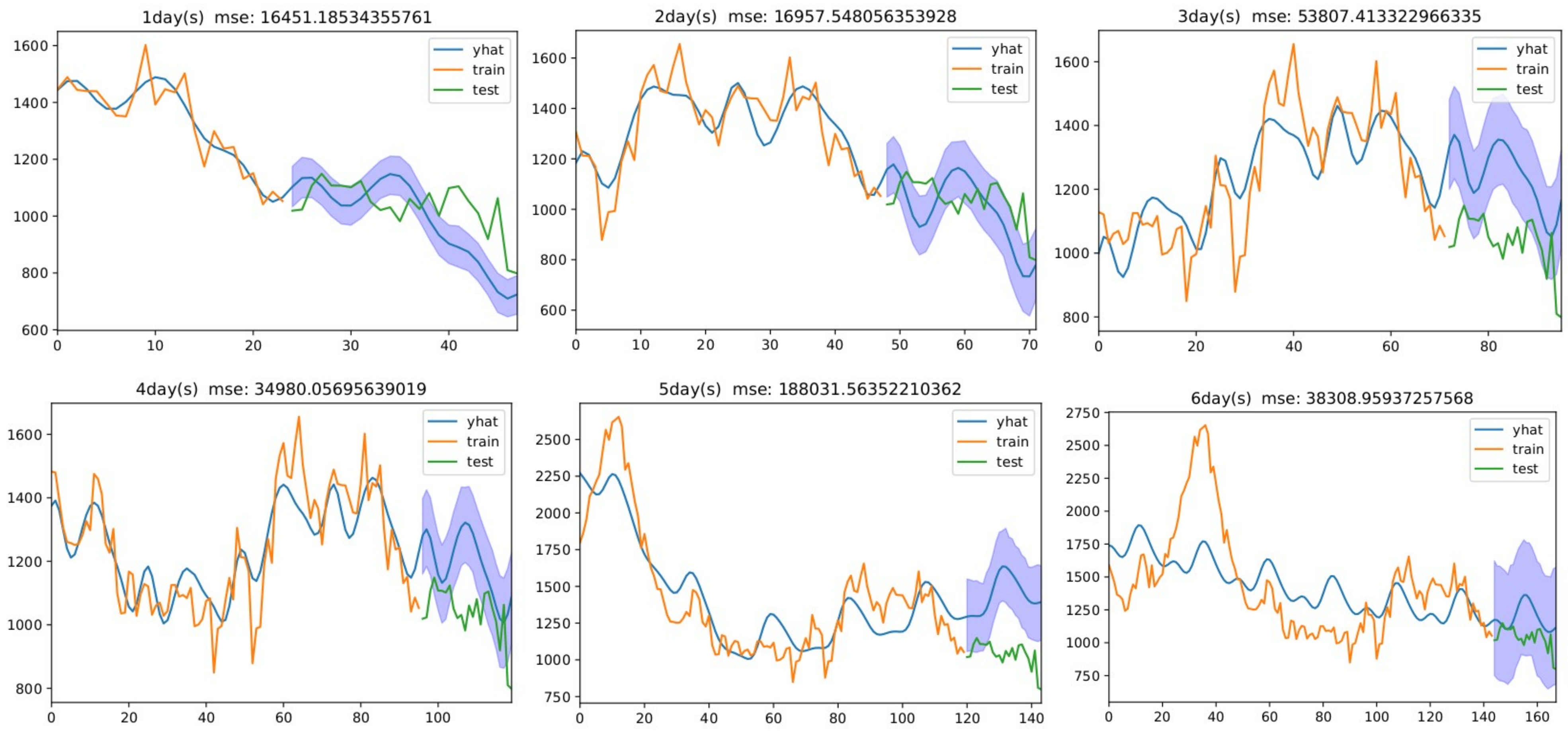
......

Result obtained 2

- ***C*** = 2015-06-16 16:00:00
- ***N*** in range([24,24*12+1,24])
- ***M*** = 24



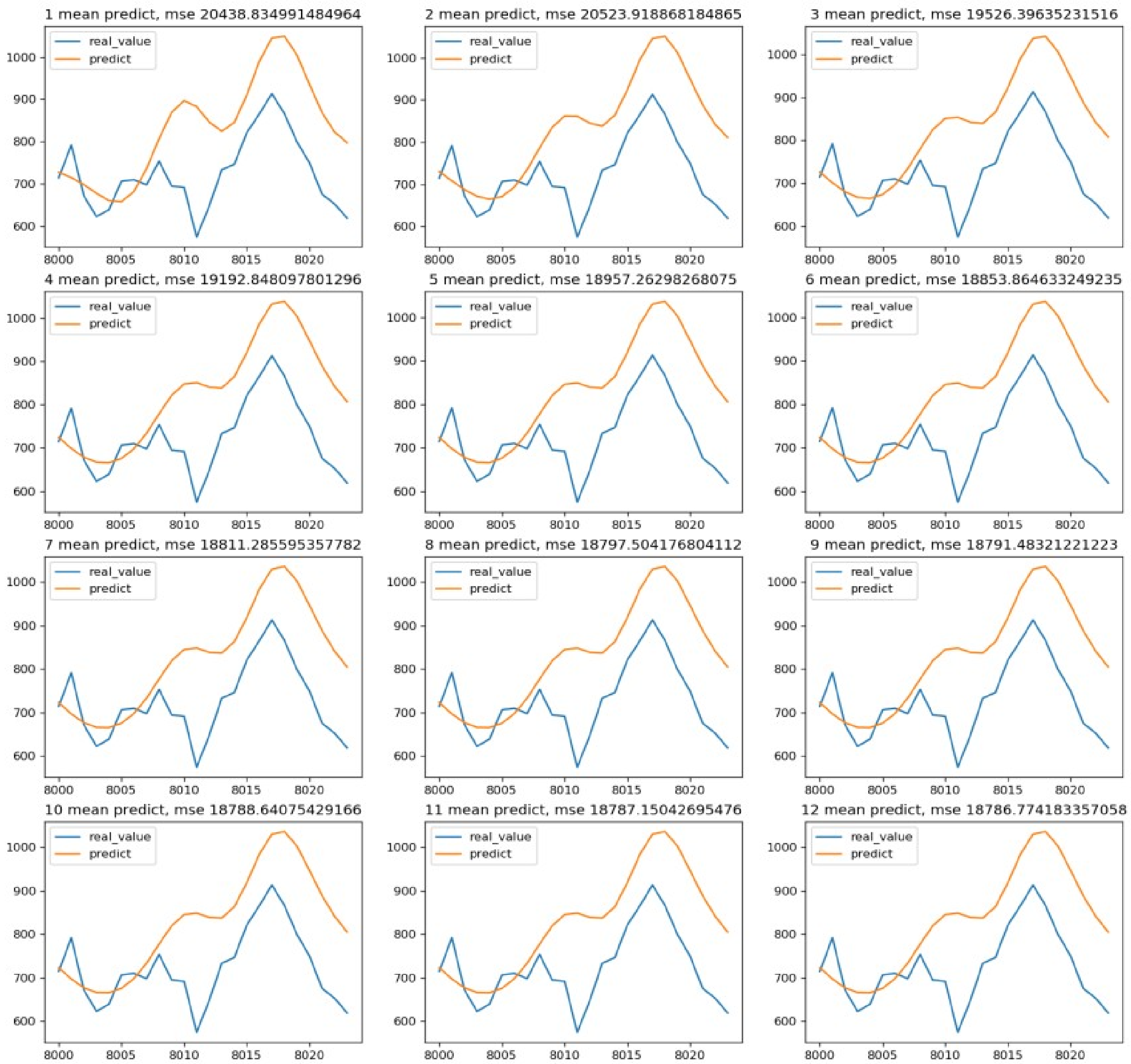
Result obtained 2 details



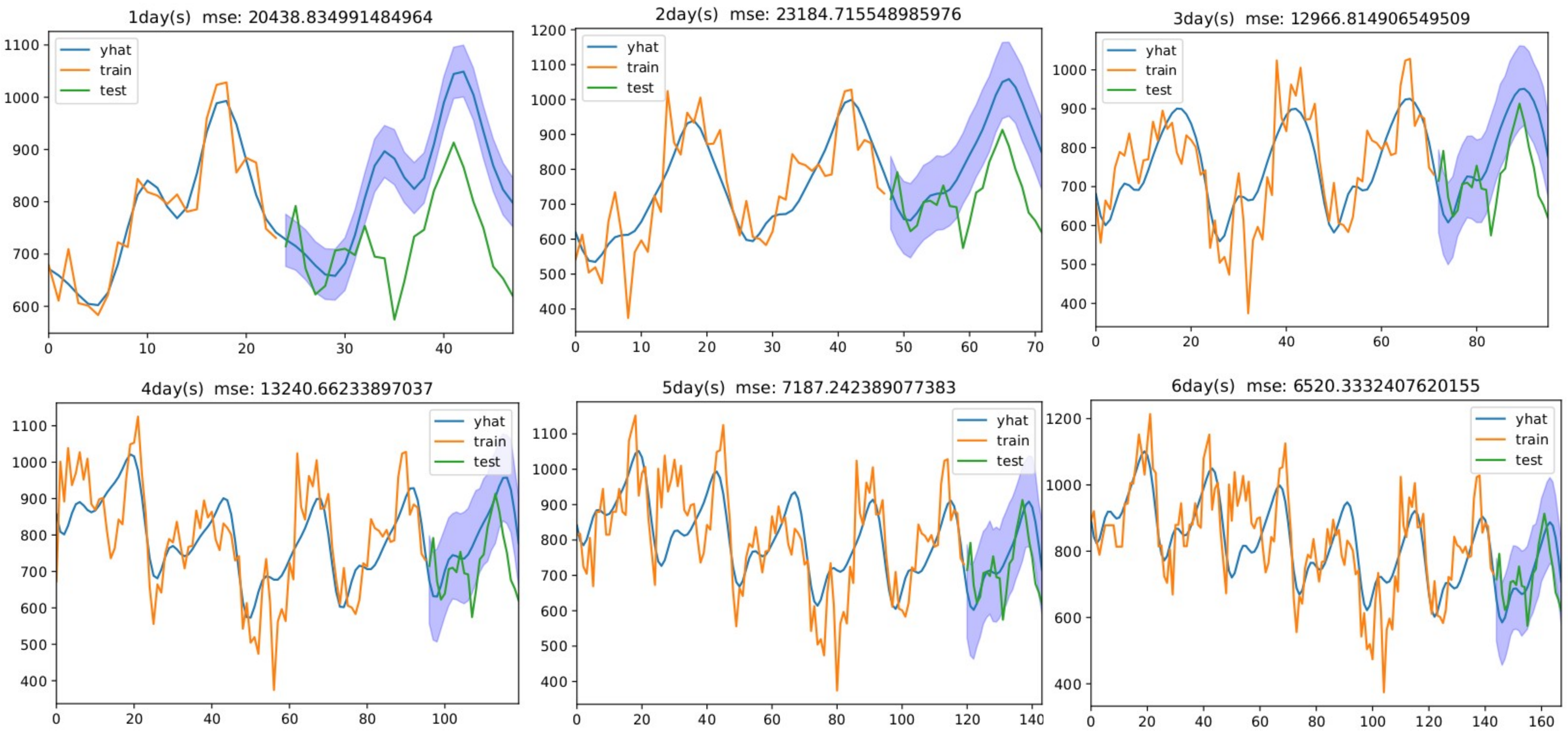
......

Result obtained 3

- ***C*** = 2015-11-30 08:00:00
- ***N*** in range([24,24*12+1,24])
- ***M*** = 24



Result obtained 3 details



……

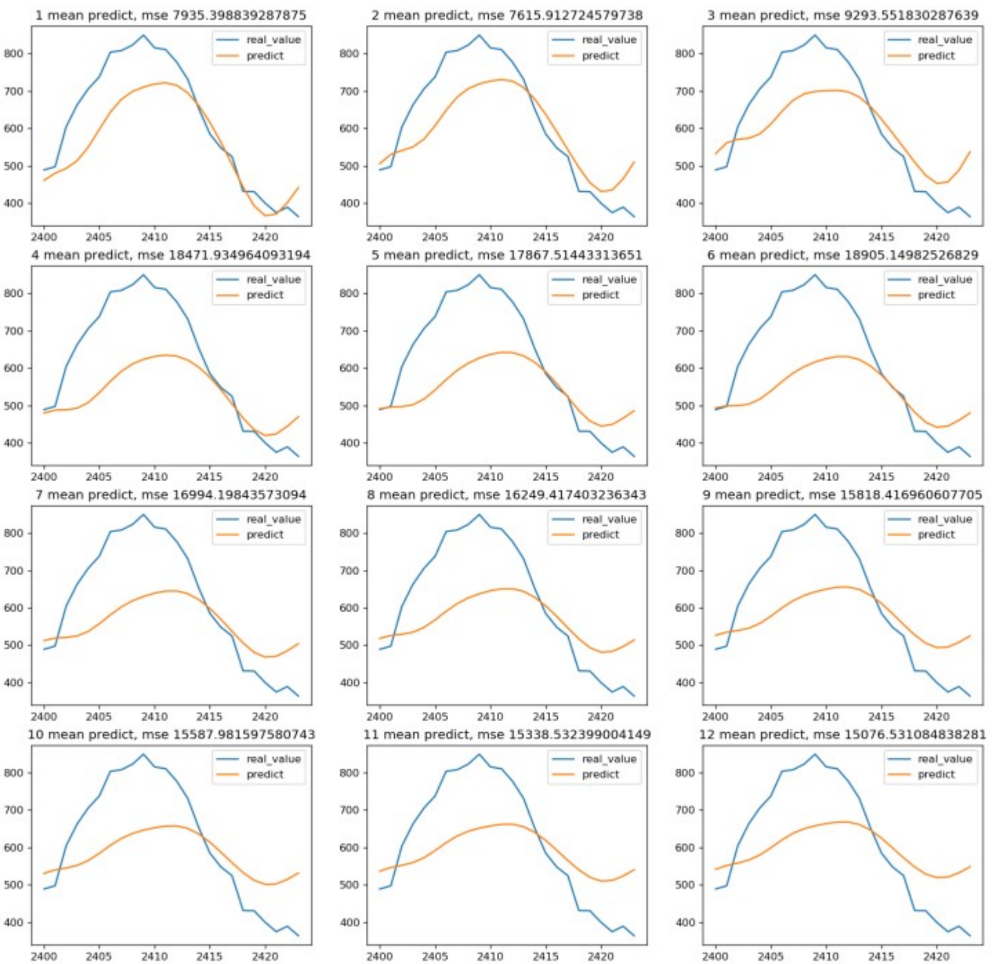
Weight strategies

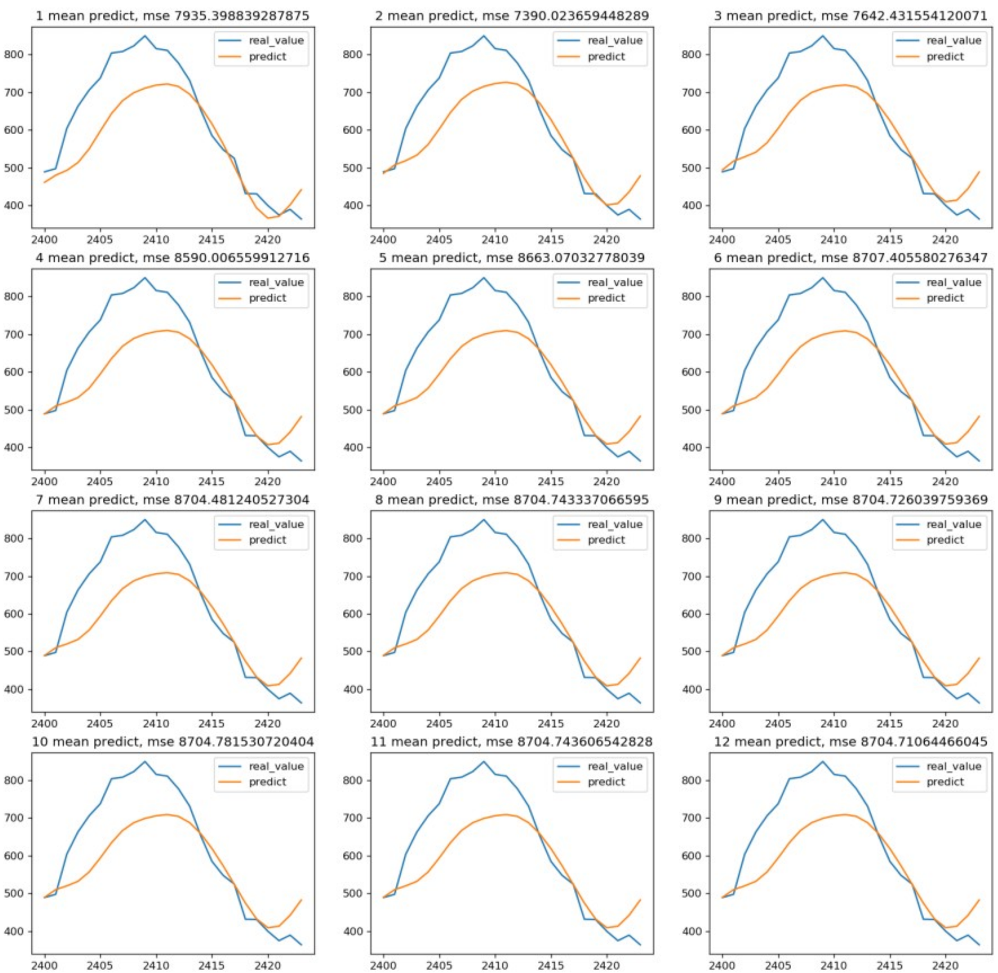
- Let ***n*** = Number of next period points prediction sequences
  - Strategy1: share the same weight 1/n
  - Strategy2: e^i / sum(e^j) (i , j in range(n))
- Code:

```
In [6]: def get_weights(n,weight_type='same'):
            if weight_type is 'same':
                return np.array(n*[1.0/n])
            elif weight_type is 'softmax':
                denominator = np.sum(np.exp([i for i in range(n)]))
        #         print(denominator)
                numerator = np.exp(np.abs(np.sort([-i for i in range(n)])))
        #         print(numerator)
                return numerator / denominator
            else:
                pass
        print('same weight\n',get_weights(12,'same'),'\n')
        print('softmax weight\n',get_weights(12,'softmax'))

        same weight
         [0.08333333 0.08333333 0.08333333 0.08333333 0.08333333 0.08333333
         0.08333333 0.08333333 0.08333333 0.08333333 0.08333333 0.08333333]

        softmax weight
         [6.32124443e-01 2.32545587e-01 8.55487405e-02 3.14716228e-02
         1.15777630e-02 4.25922099e-03 1.56687984e-03 5.76422879e-04
         2.12054127e-04 7.80103536e-05 2.86984053e-05 1.05575533e-05]
```

Comparison of two strategies

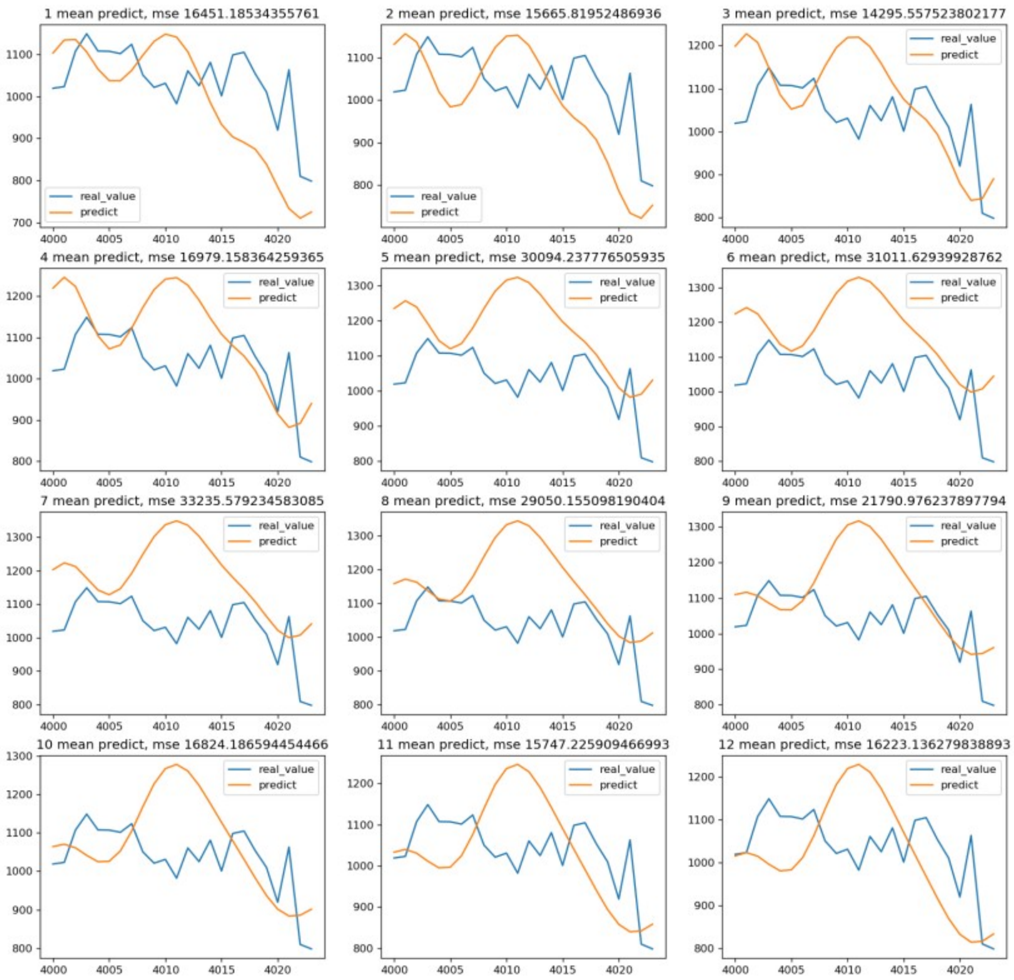
‘same’ VS ‘softmax’

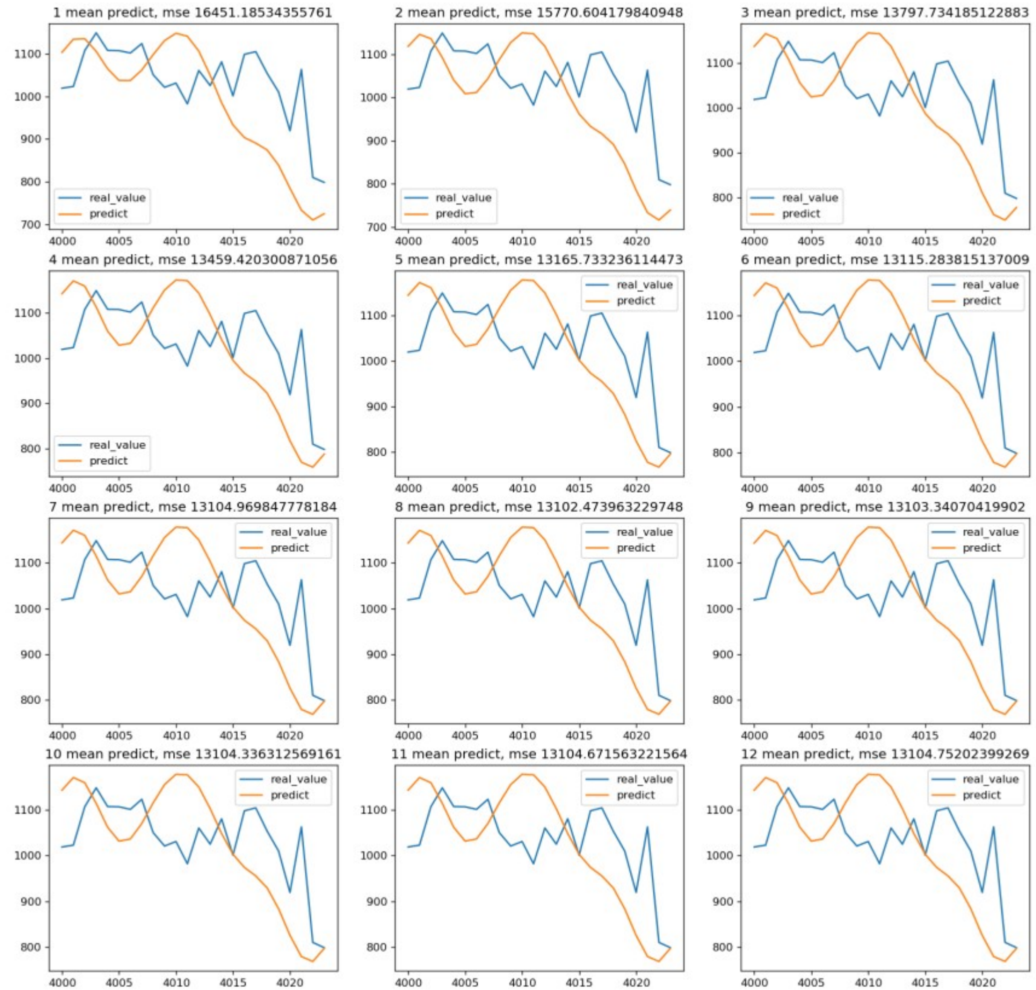




Comparison of two strategies

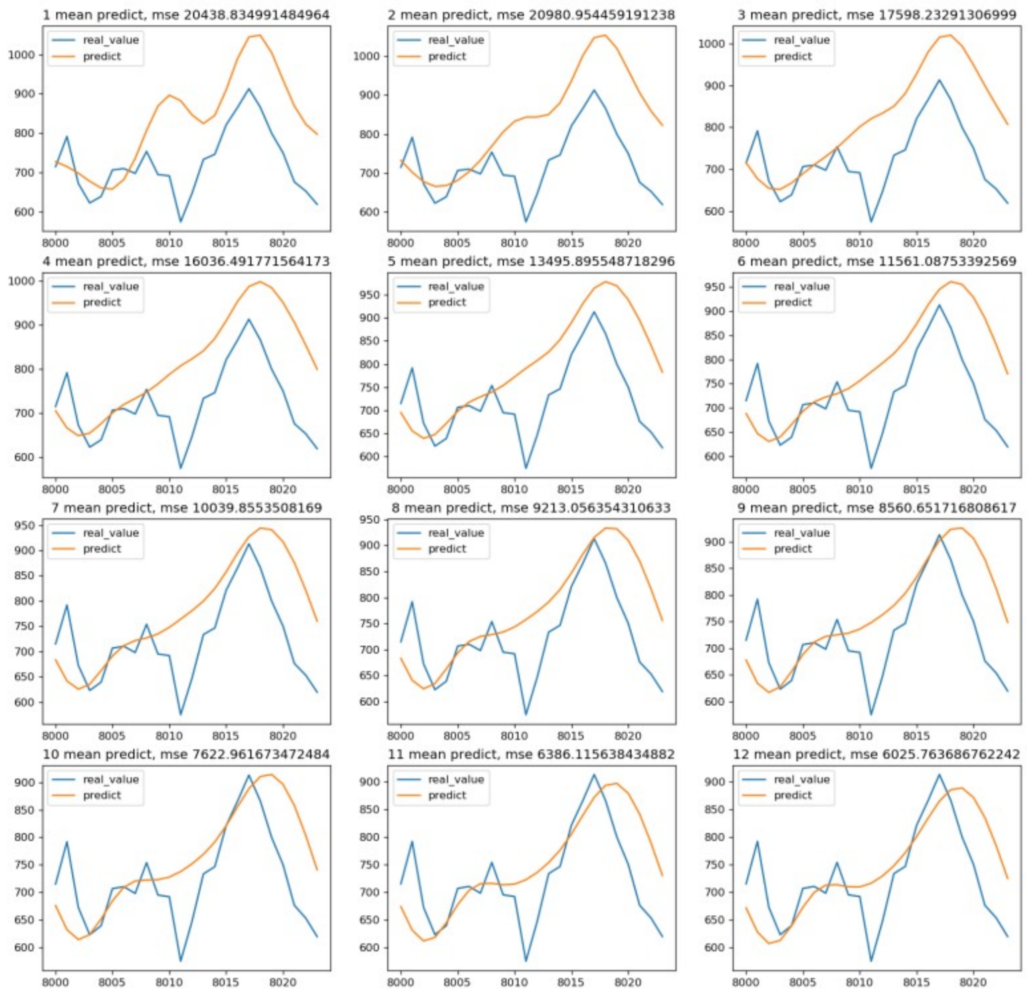
- ‘same’ VS ‘softmax’

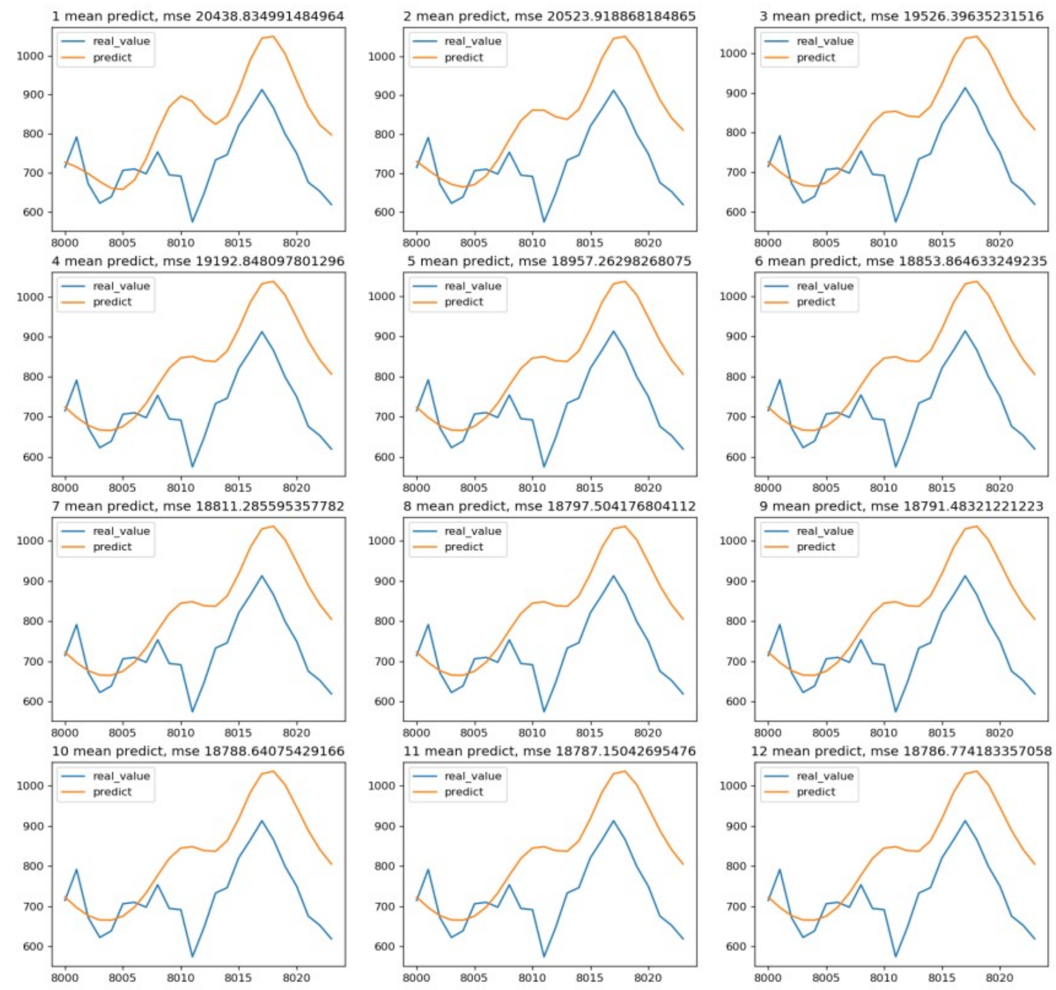




Comparison of two strategies

- ‘same’ VS ‘softmax’



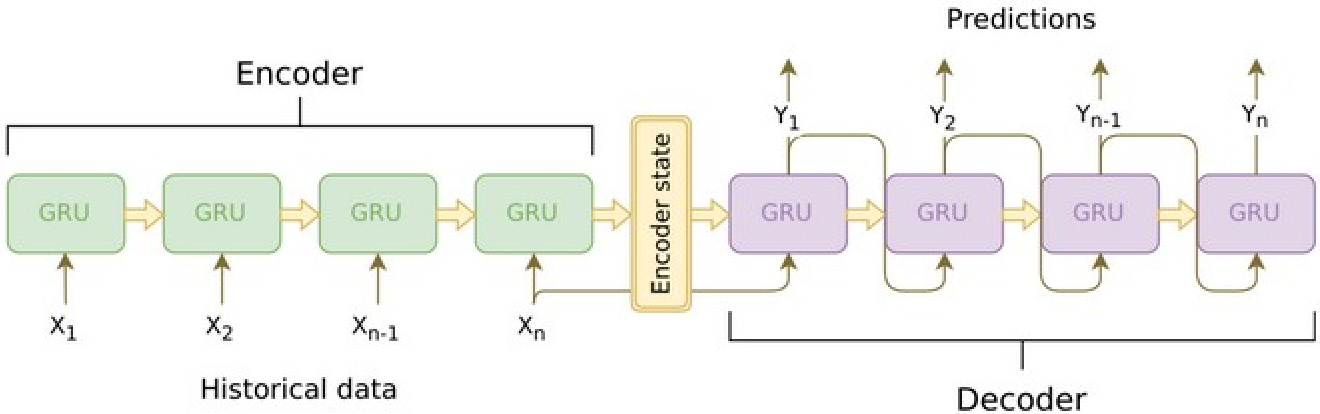


Advantages and disadvantages

- Advantages:
  - The model is very simple, just focus on the data of the recent Npoints, it only takes very little training data and time.
  - Models can 'remember' the impact of data points far away from themselves on current data trends
  - The effects of recent and distant trends on current data trends can be adjusted with different weights
- Disdvantages:
  - Finding the optimal weight assignment strategy may be difficult
  - The trend of different Npoints sequence may be quite different, which is unfavorable for the prediction of the results.

Solution assumption

- Improve weights assignment strategy
- Construct a supervised learning model to 'learn' weights
- Using Encoder, Decoder(seq2seq)
  - Mapping a fixed length sequence to a consecutive fixed length sequence(with GRU or LSTM layers)



- Demo implementation
  - https://www.kaggle.com/c/web-traffic-time-series-forecasting/discussion/43795#latest-567361

Improved weight strategy

- Lower pass filter

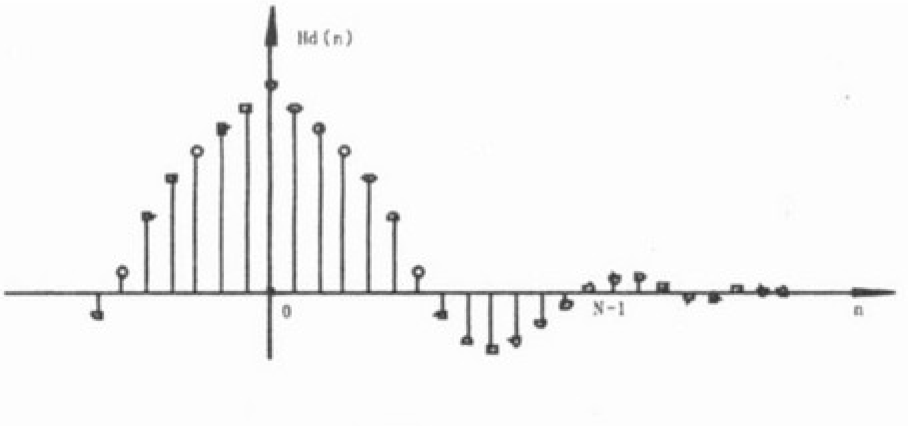
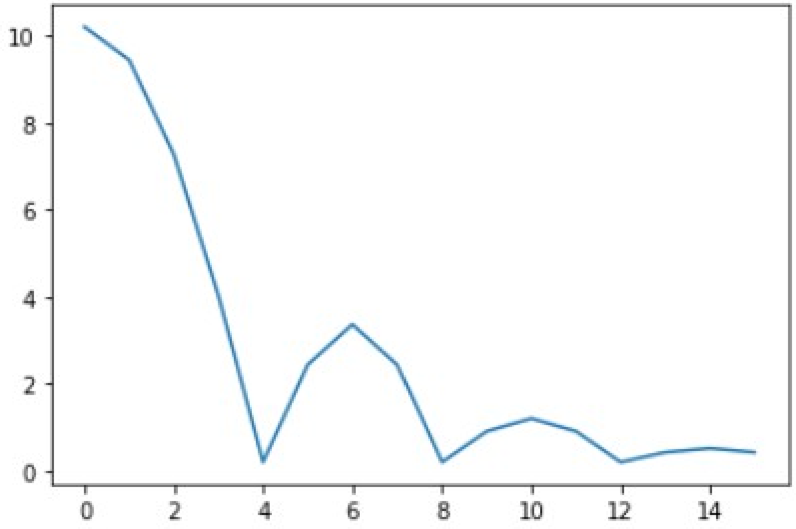


图 1　理想低通滤波器冲激响应图

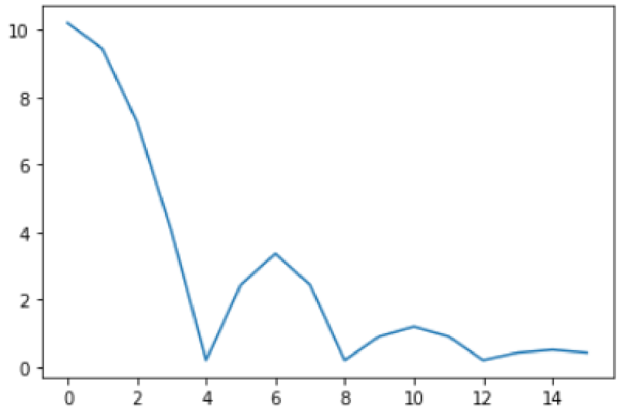


Improved weight strategy

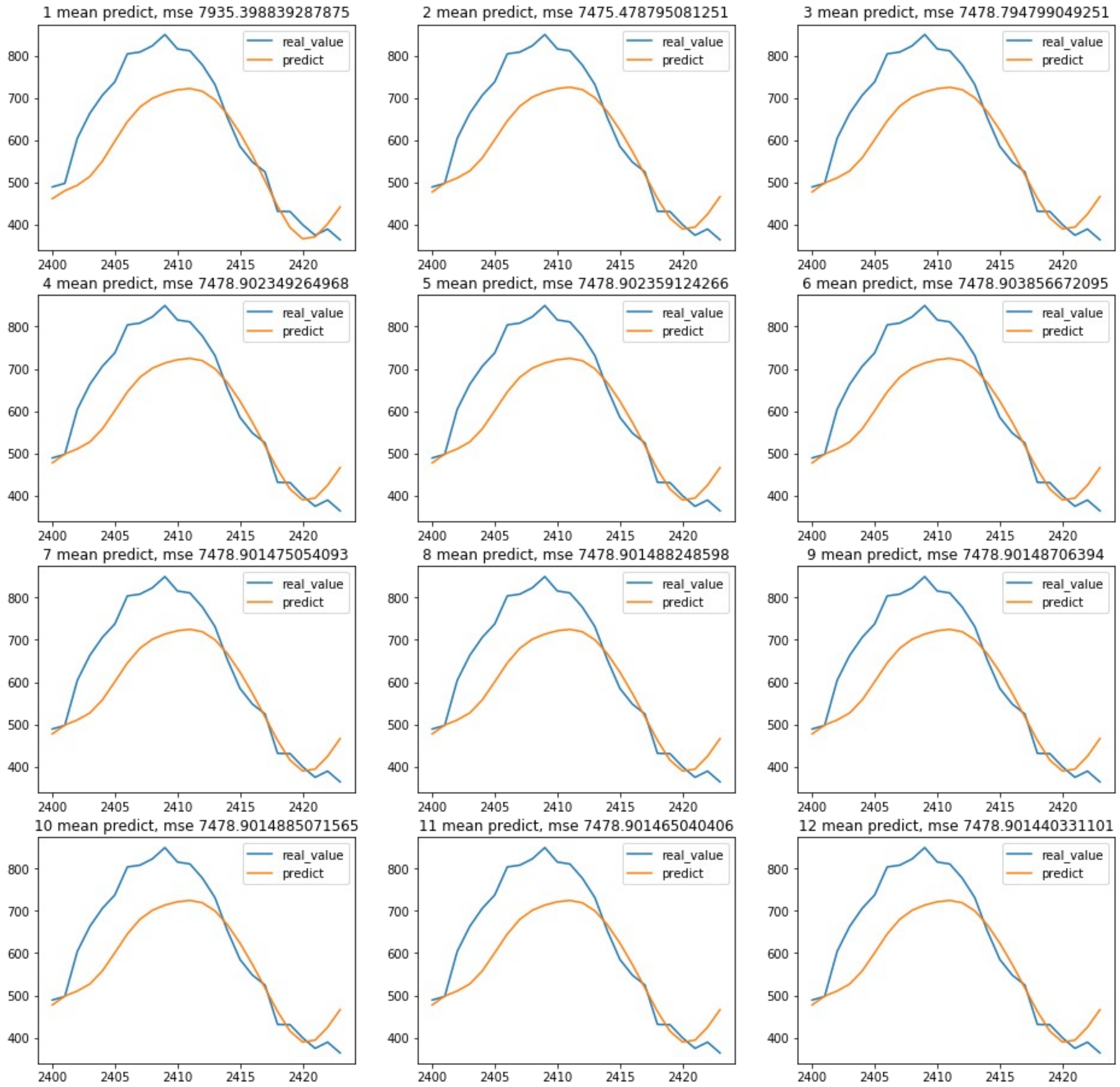
- Lower pass filter

In [155]:
```
def lower_pass_filter(n,period,amplitude,shift):
    res = []
    for i in range(int(n / period)+1):
        weight_decay = (1 / (10 ** 0.5)) ** i
        if i == 0:
            res.append(shift + weight_decay *amplitude * np.cos([i*(np.pi/2)/period for i in range(period)] ))
        else:
            res.append(shift + weight_decay *amplitude * np.cos([-np.pi/2 + i*(np.pi)/period for i in range(period)
    return res
res = np.array(lower_pass_filter(12,4,10,0.2)).reshape(-1)
plt.plot(res)
denominator = np.sum(np.exp(res))
numerator = np.exp(res)
res = numerator / denominator
res
```

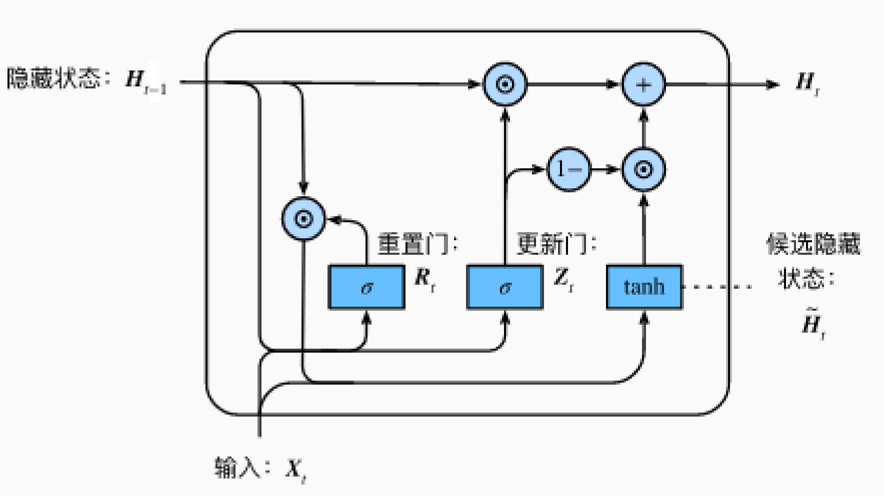
Out[155]:
```
array([6.55658680e-01, 3.06260386e-01, 3.50476359e-02, 1.36680332e-03,
       2.97668580e-05, 2.78512685e-04, 7.03222462e-04, 2.78512685e-04,
       2.97668580e-05, 6.03706108e-05, 8.09147093e-05, 6.03706108e-05,
       2.97668580e-05, 3.72258133e-05, 4.08384237e-05, 3.72258133e-05])
```



Improved results



GRU(gated recurrent unit)



- Reset Gate & Update Gate

$$\begin{aligned} \boldsymbol{R}_t &= \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xr} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hr} + \boldsymbol{b}_r), \\ \boldsymbol{Z}_t &= \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xz} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hz} + \boldsymbol{b}_z), \end{aligned}$$

- Candidate hidden state

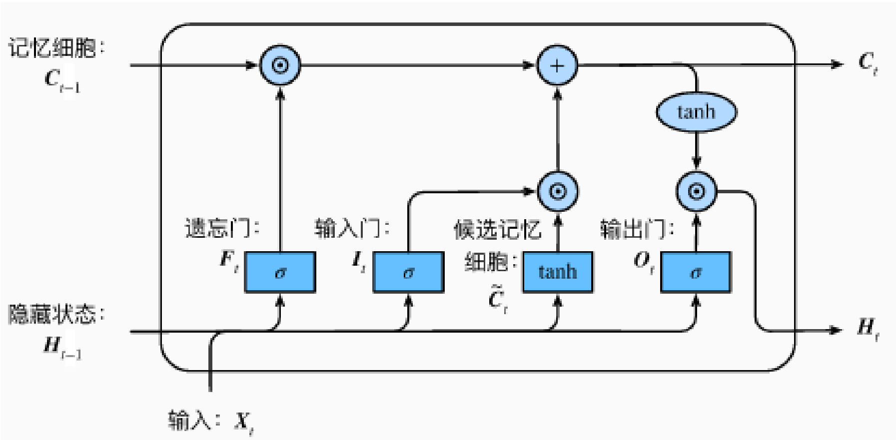
$$\tilde{\boldsymbol{H}}_t = \tanh(\boldsymbol{X}_t \boldsymbol{W}_{xh} + (\boldsymbol{R}_t \odot \boldsymbol{H}_{t-1}) \boldsymbol{W}_{hh} + \boldsymbol{b}_h),$$

- Hidden state

$$\boldsymbol{H}_t = \boldsymbol{Z}_t \odot \boldsymbol{H}_{t-1} + (1 - \boldsymbol{Z}_t) \odot \tilde{\boldsymbol{H}}_t.$$

- Summary
  - Reset gate can be used to discard some historical information unrelated to prediction
  - Reset gate helps capture short-term dependencies in the time series
  - Update gate helps capture long-term dependencies in time series
- References
  - https://arxiv.org/pdf/1409.1259.pdf
  - https://arxiv.org/pdf/1412.3555.pdf

LSTM(long short-term memory)



- Input, Forget, Output Gate

$$
\begin{aligned}
\boldsymbol{I}_t &= \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i),\\
\boldsymbol{F}_t &= \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f),\\
\boldsymbol{O}_t &= \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{ho} + \boldsymbol{b}_o),
\end{aligned}
$$

- Candidate Memory Cell

$$\tilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{X}_t \boldsymbol{W}_{xc} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hc} + \boldsymbol{b}_c),$$

- Memory Cell

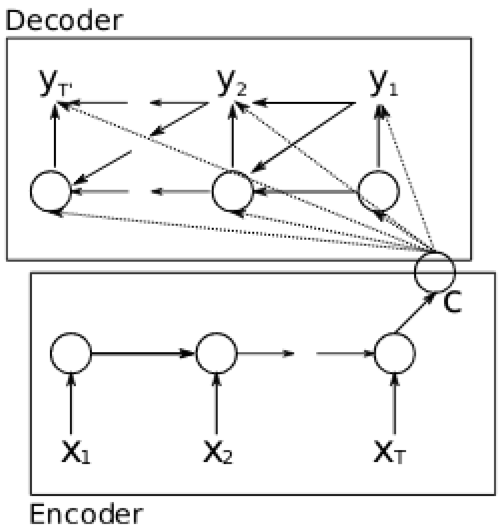
$$\boldsymbol{C}_t = \boldsymbol{F}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \odot \tilde{\boldsymbol{C}}_t.$$

- Hidden state

$$\boldsymbol{H}_t = \boldsymbol{O}_t \odot \tanh(\boldsymbol{C}_t).$$

- Summary
  - The hidden layer output of LSTM includes ***Ht*** and ***Ct***. Only ***Ht*** is passed to the output layer
  - LSTM network can cope with the gradient attenuation problem in the cyclic neural network and better capture the dependence of the time step distance in the time series.
- References
  - https://link.springer.com/content/pdf/10.1007%2F978-3-642-24797-2.pdf

Encoder&Decoder(seq2seq)



- Encoder

$$\boldsymbol{h}_t = f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$$

$$\boldsymbol{c} = q(\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T).$$

- Decoder

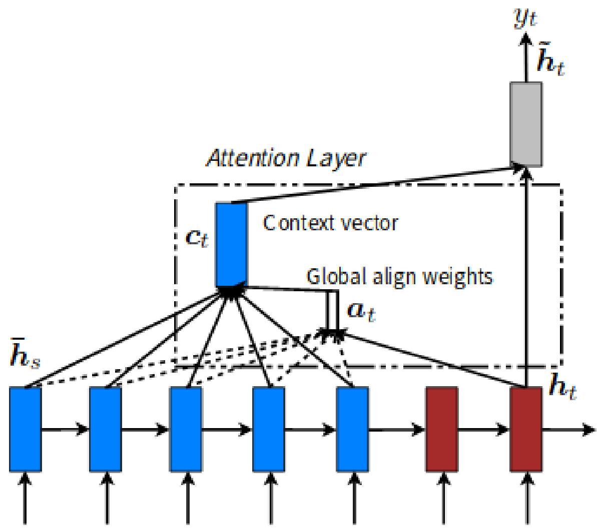
$$\boldsymbol{s}_{t'} = g(y_{t'-1}, \boldsymbol{c}, \boldsymbol{s}_{t'-1}).$$

- ***How to choose c ????***

- Summary
  - Encoder transforms the hidden state of each time step into a background variable through a custom function **q**
  - Decoder $P(y_{t'} \mid y_1, \ldots, y_{t'-1}, \boldsymbol{c})$
- References
  - https://arxiv.org/pdf/1406.1078.pdf
  - https://arxiv.org/pdf/1409.3215.pdf

Attention mechanism in seq2seq

- The attention should assigned differently by different steps.
  - Global Attention



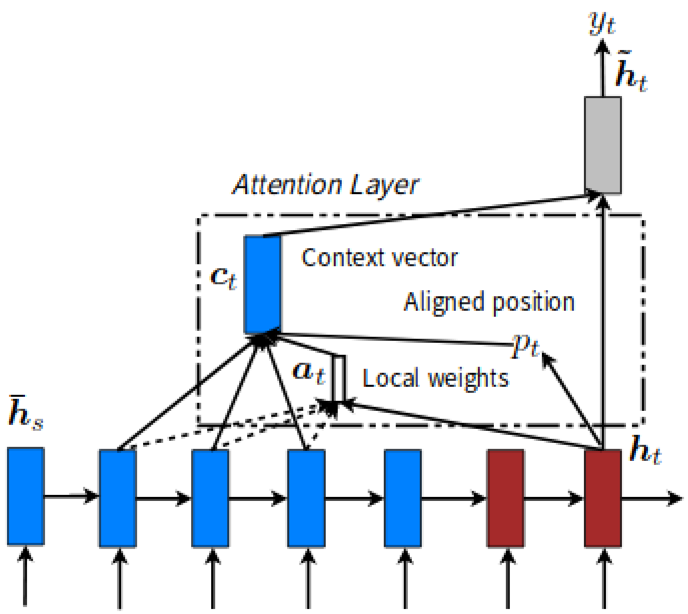
$$\begin{aligned}\boldsymbol{a}_t(s) &= \operatorname{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) \\ &= \frac{\exp\left(\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'} \exp\left(\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)}\end{aligned}$$

$$\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s & \textit{dot} \\ \boldsymbol{h}_t^\top \boldsymbol{W}_{\boldsymbol{a}} \bar{\boldsymbol{h}}_s & \textit{general} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W}_{\boldsymbol{a}}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right) & \textit{concat} \end{cases}$$

Figure 2: **Global attentional model** – at each time step t, the model infers a *variable-length* alignment weight vector $\boldsymbol{a}_t$ based on the current target state $\boldsymbol{h}_t$ and all source states $\bar{\boldsymbol{h}}_s$. A global context vector $\boldsymbol{c}_t$ is then computed as the weighted average, according to $\boldsymbol{a}_t$, over all the source states.

Attention mechanism in seq2seq

- The attention should assigned differently by different steps.
  - Local Attention



$$p_t = S \cdot \mathrm{sigmoid}(\boldsymbol{v}_p^\top \tanh(\boldsymbol{W_p h}_t)), \quad (9)$$

$\boldsymbol{W_p}$ and $\boldsymbol{v}_p$ are the model parameters which will be learned to predict positions. S is the source sentence length. As a result of sigmoid, $p_t \in [0, S]$. To favor alignment points near p_t, we place a Gaussian distribution centered around p_t. Specifically, our alignment weights are now defined as:

$$\boldsymbol{a}_t(s) = \mathrm{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) \exp\left(-\frac{(s - p_t)^2}{2\sigma^2}\right) \quad (10)$$

We use the same align function as in Eq. (7) and the standard deviation is empirically set as $\sigma = \frac{D}{2}$. Note that p_t is a *real* nummber; whereas s is an *integer* within the window centered at p_t.[10]

Attention mechanism in seq2seq

- The attention should assigned differently by different steps.
  - Input-feeding Approach

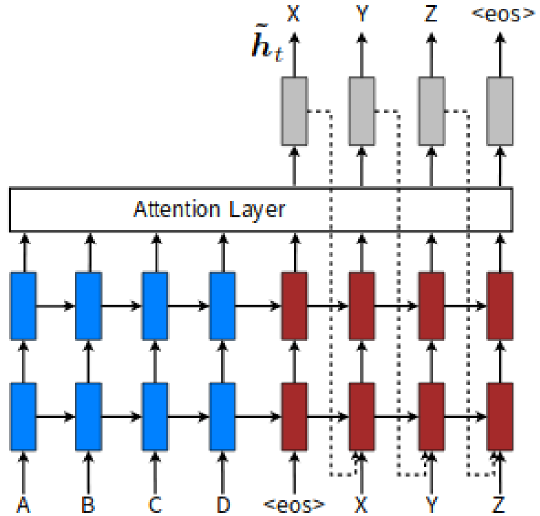


Figure 4: **Input-feeding approach** – Attentional vectors $\tilde{\boldsymbol{h}}_t$ are fed as inputs to the next time steps to inform the model about past alignment decisions.

- References
  - https://arxiv.org/pdf/1508.04025.pdf

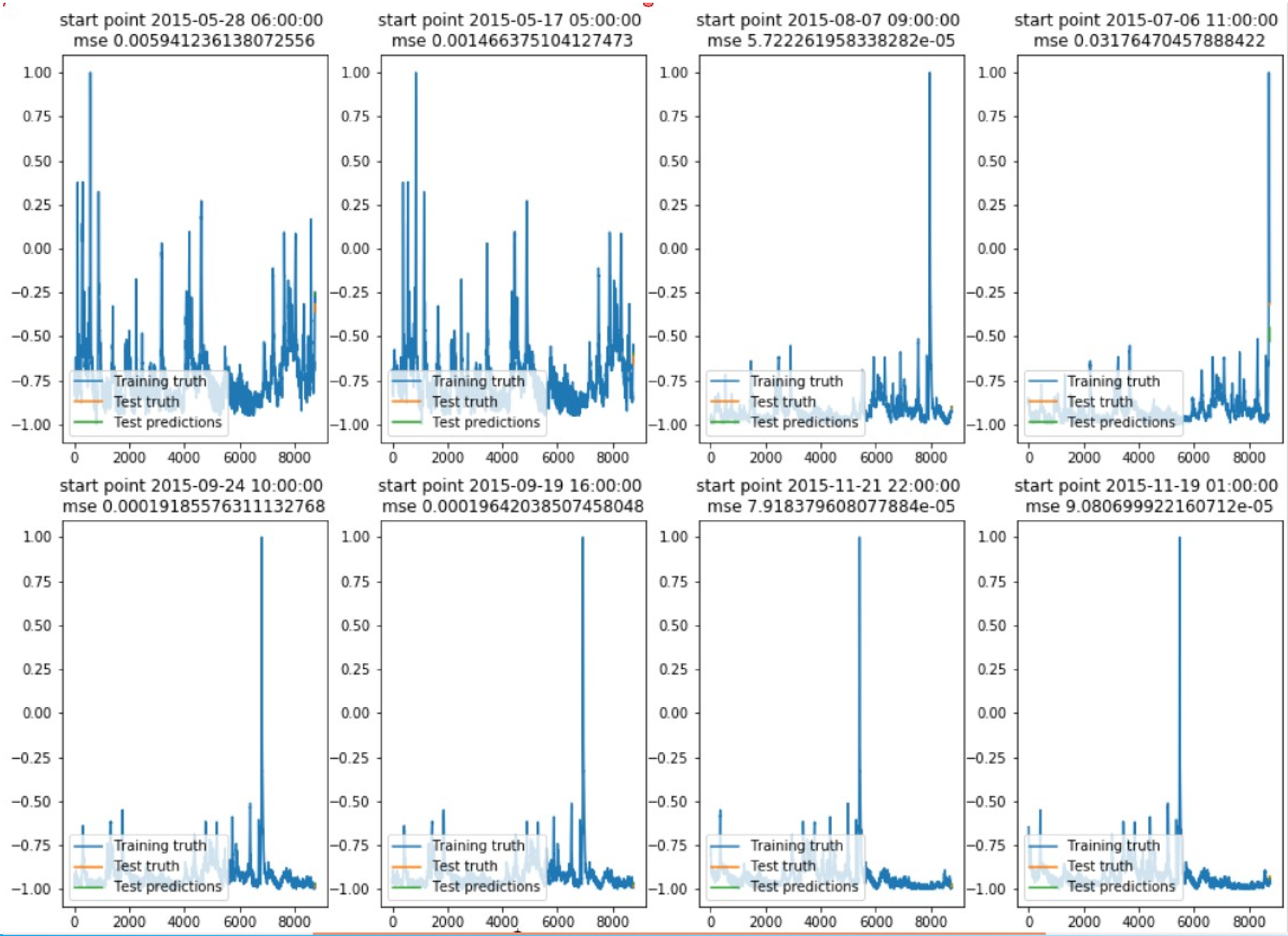
Using seq2seq

- Data preprocessing: MinMaxScaler((-1,1))
- Make data into: sequences → sequences

	current_before_3	current_before_2	current_before_1	current	current_next_1	current_next_2
3	-0.964450	-0.961909	-0.96248	-0.964150	-0.965671	-0.964560
4	-0.961909	-0.962480	-0.96415	-0.965671	-0.964560	-0.966197

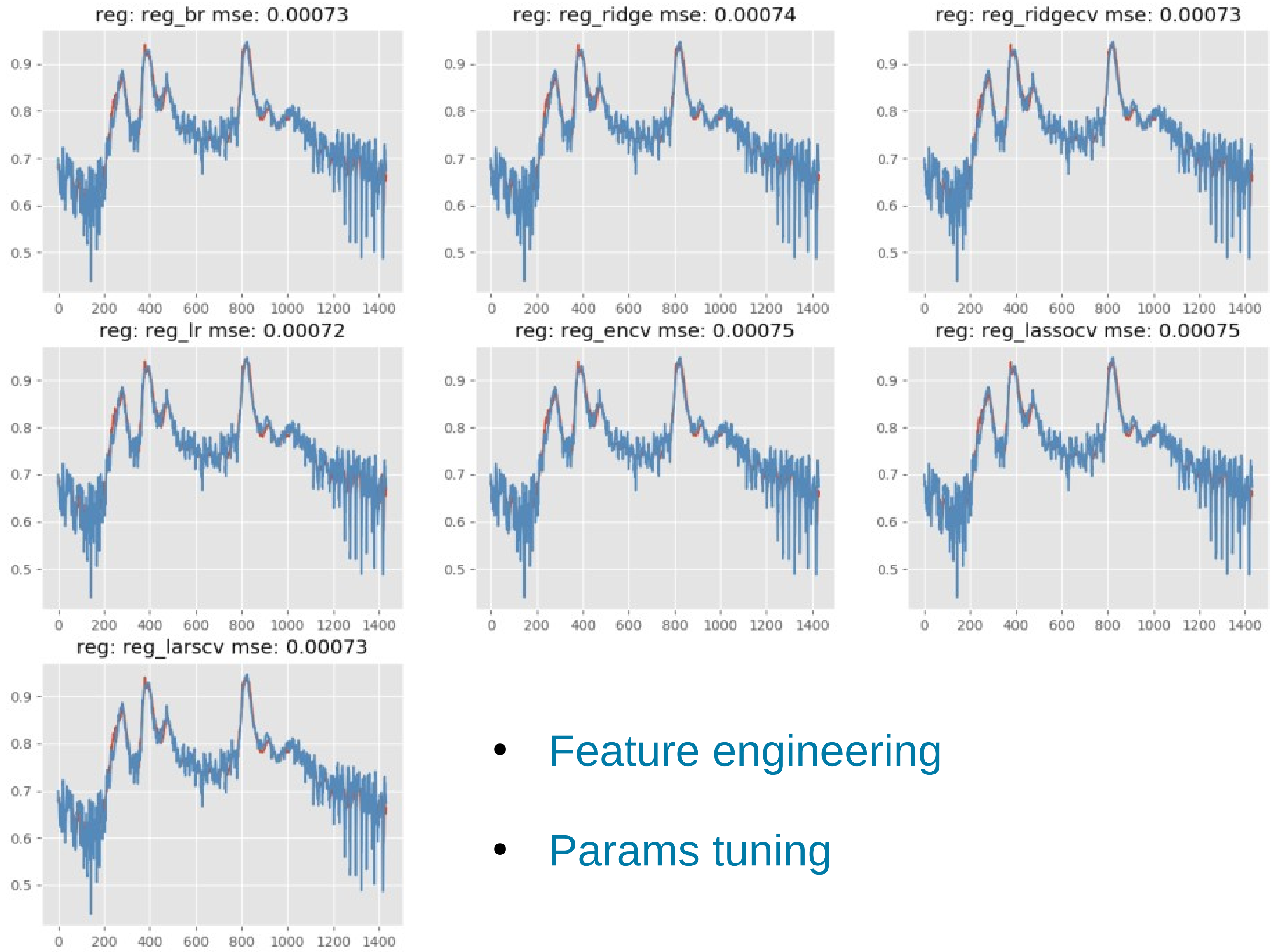
- We can build data set seq(len:***N***) → seq(len:***M***) in any ***N***&***M***
- Have a try with ***N*** = 3, ***M*** = 3

Result obtained



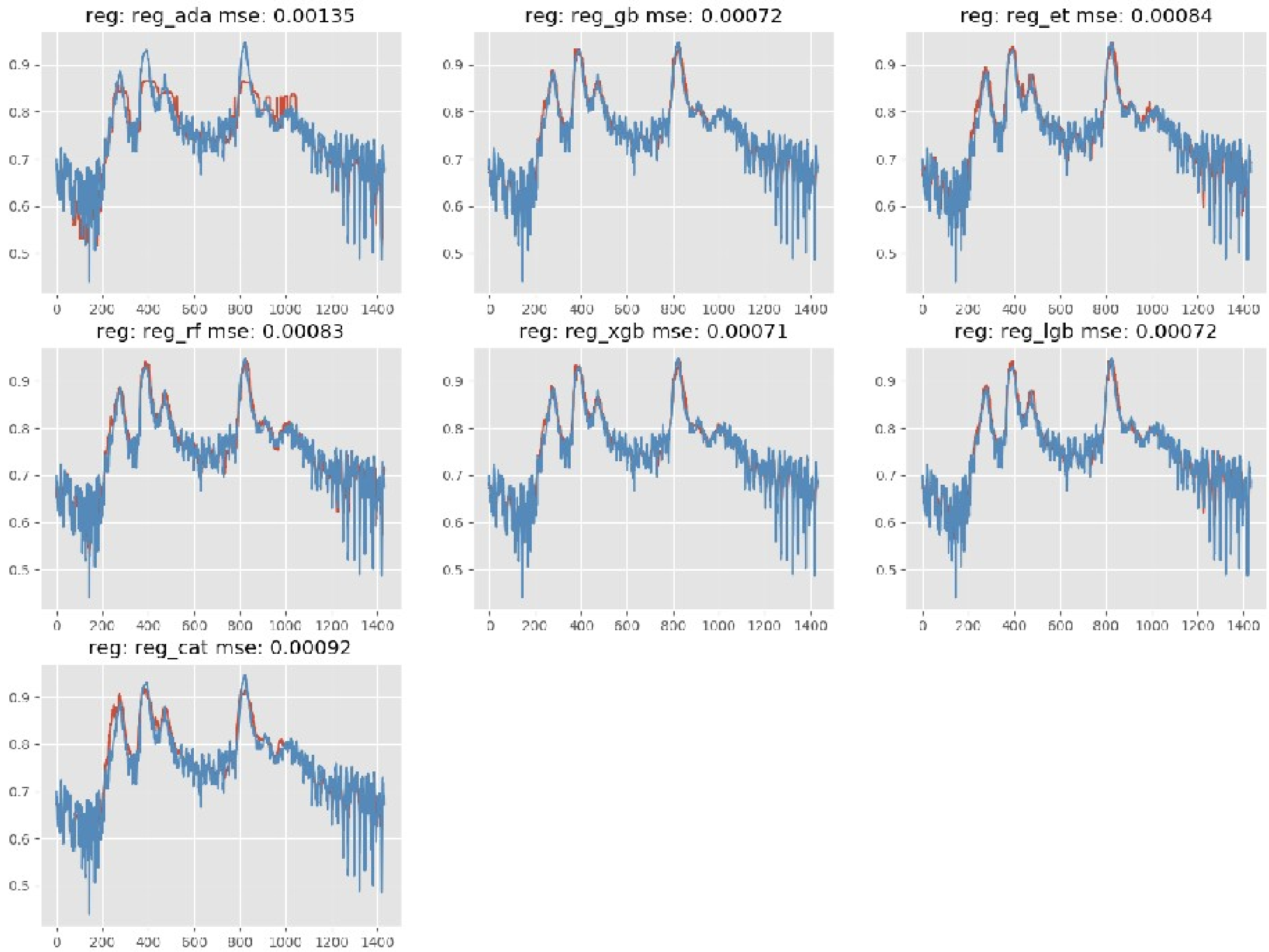
Single step prediction

Linear model

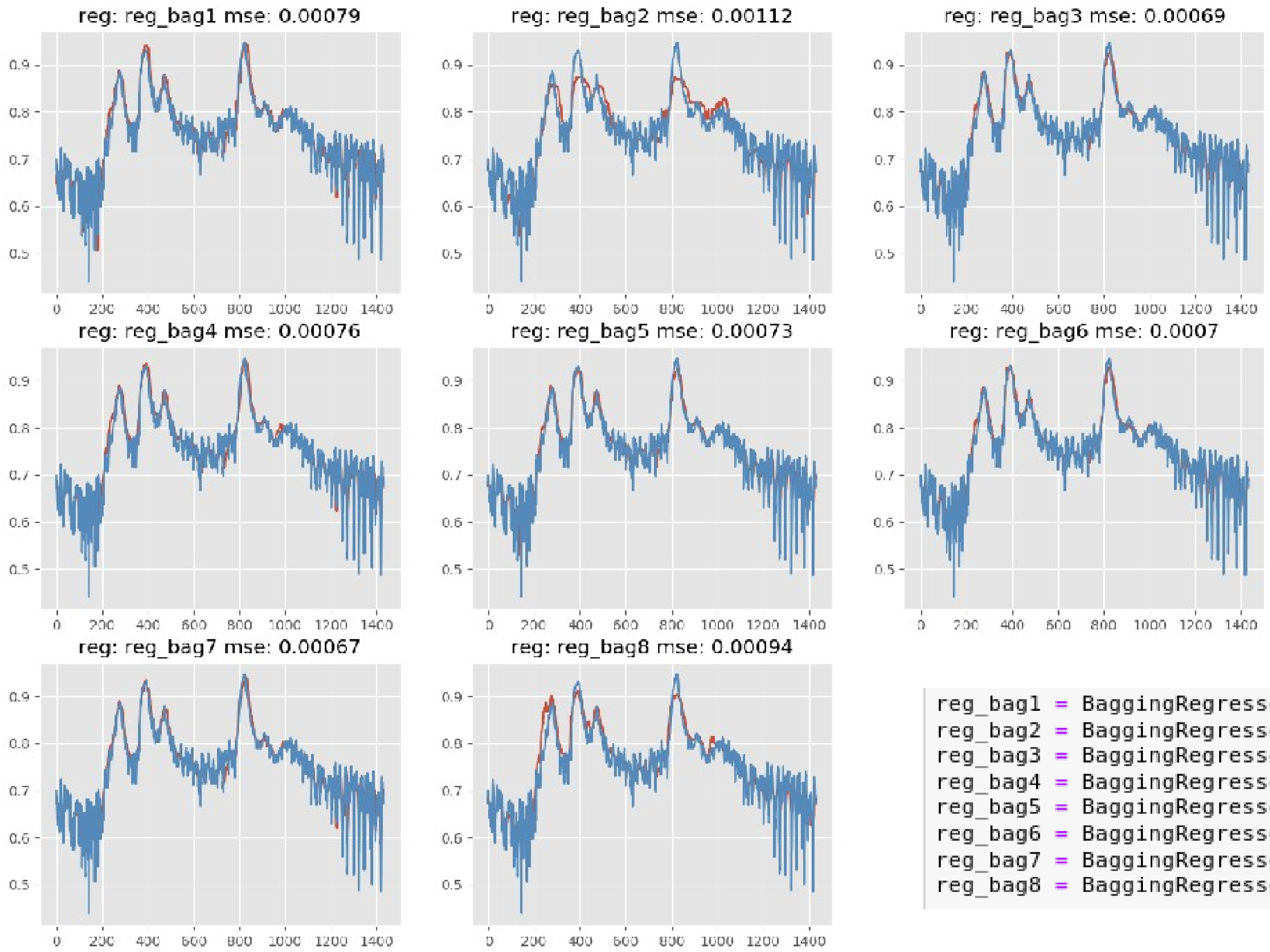


- Feature engineering
- Params tuning

Tree Based model

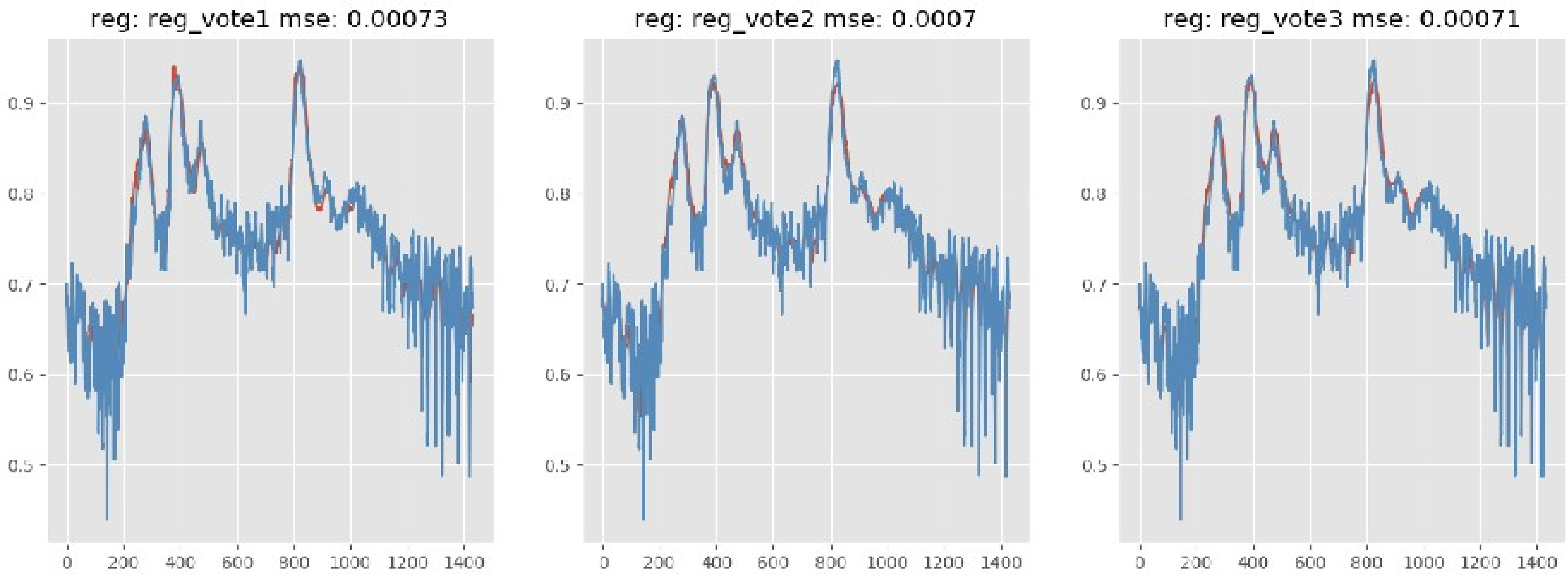


Bagging




```
reg_bag1 = BaggingRegressor()
reg_bag2 = BaggingRegressor(base_estimator=reg_ada)
reg_bag3 = BaggingRegressor(base_estimator=reg_gb)
reg_bag4 = BaggingRegressor(base_estimator=reg_et)
reg_bag5 = BaggingRegressor(base_estimator=reg_rf)
reg_bag6 = BaggingRegressor(base_estimator=reg_xgb)
reg_bag7 = BaggingRegressor(base_estimator=reg_lgb)
reg_bag8 = BaggingRegressor(base_estimator=reg_cat)
```

Voting




```python
linear_models = []
linear_models.append(('reg_br',reg_br))
linear_models.append(('reg_ridge',reg_ridge))
linear_models.append(('reg_ridgecv',reg_ridgecv))
linear_models.append(('reg_lr',reg_lr))
linear_models.append(('reg_encv',reg_encv))
linear_models.append(('reg_lassocv',reg_lassocv))
linear_models.append(('reg_larscv',reg_larscv))
reg_vote1 = VotingRegressor(estimators=linear_models)
```

```python
tree_based_models = []
tree_based_models.append(('reg_ada',reg_ada))
tree_based_models.append(('reg_gb',reg_gb))
tree_based_models.append(('reg_et',reg_et))
tree_based_models.append(('reg_rf',reg_rf))
tree_based_models.append(('reg_xgb',reg_xgb))
tree_based_models.append(('reg_lgb',reg_lgb))
tree_based_models.append(('reg_cat',reg_cat))
reg_vote2 = VotingRegressor(estimators=tree_based_models)
```

```python
bagging_models = []
bagging_models.append(('reg_bag1',reg_bag1))
bagging_models.append(('reg_bag2',reg_bag2))
bagging_models.append(('reg_bag3',reg_bag3))
bagging_models.append(('reg_bag4',reg_bag4))
bagging_models.append(('reg_bag5',reg_bag5))
bagging_models.append(('reg_bag6',reg_bag6))
bagging_models.append(('reg_bag7',reg_bag7))
bagging_models.append(('reg_bag8',reg_bag8))
reg_vote3 = VotingRegressor(estimators=bagging_models)
```