Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting

Code Review

송규헌

0. Code Review

- Official github : https://github.com/zhouhaoyi/Informer2020/tree/main

  └ /models/ : attn.py, decoder.py, embed.py, encoder.py, model.py

- Code example in Colab : https://colab.research.google.com/drive/1_X7O2BkFLvqyCdZzDZvV2MB0aAvYALLC

  └ modify

```
!git clone https://github.com/zhouhaoyi/Informer2020.git
!git clone https://github.com/zhouhaoyi/ETDataset.git
!ls
```

to

```
!git clone https://github.com/skier-song9/Informer2020.git
!git clone https://github.com/zhouhaoyi/ETDataset.git
!ls
```

0. Code Review

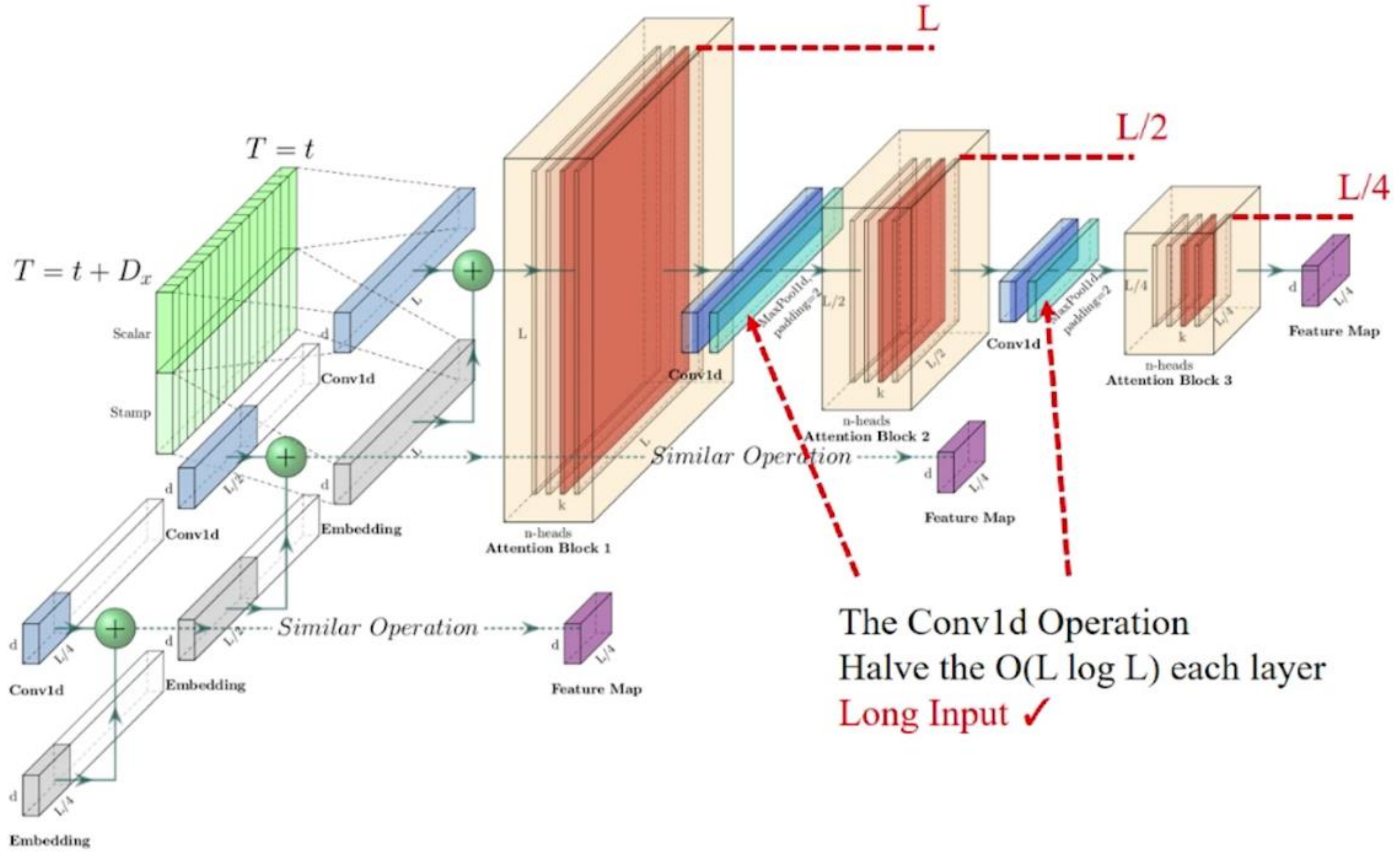
Edit

modify

```
args = dotdict()
args.model = 'informer'
...
```

to

```
args = dotdict()
args.model = 'informerstack'
...
```



AAAI-21 presentation : https://slideslive.com/38948878/informer-beyond-efficient-transformer-for-long-sequence-timeseries-forecasting

---

modify

```
args.e_layers = 2 # num of encoder layers
args.d_layers = 1 # num of decoder layers
```

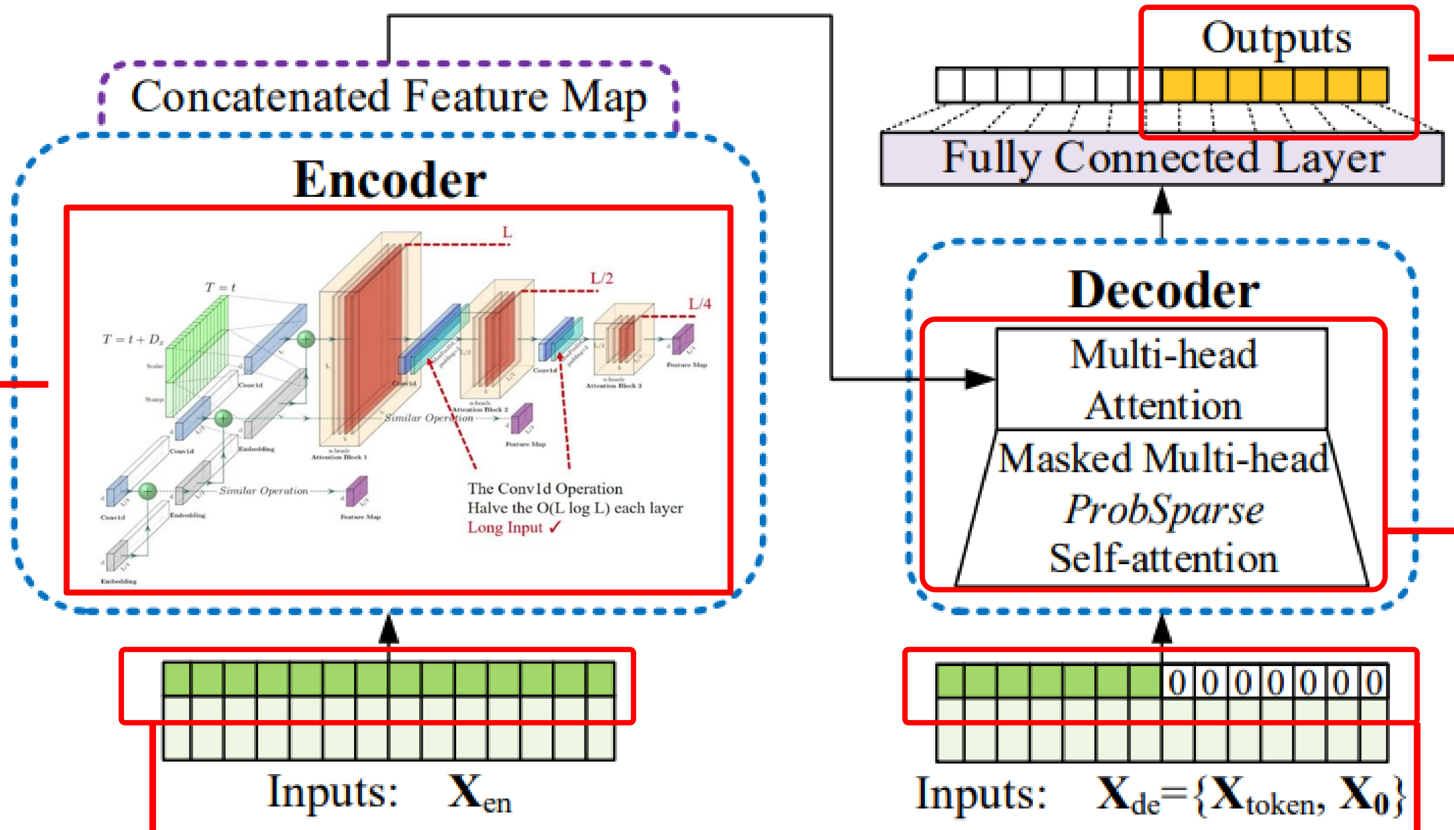
to

```
args.e_layers = 2 # num of encoder layers
args.s_layers = [3,2,1] # num of encoder layers in stacking encoder replicas
args.d_layers = 1 # num of decoder layers
```

1. Set Parameters

Variable_name	Value	Description
data	ETTh1	Electricity Tranformer Temperature per hour
features	M	- M : multivariate predict multivariate - S : univariate predict univariate - MS : multivariate predict univariate
freq	h	freq for time features encoding e.g. s:secondly, t:minutely, h:hourly, d:daily, b:business days, w:weekly, m:monthly
seq_length	96	Length of input sequence for **encoder** (In this case, 96 hours of ETT data)
label_len	48	Length of X^t_{token} input sequence for **decoder**
pred_len(=out_len)	24	Length of X^t_0 prediction period (In this case, goal is to predict next 24 hours of ETT)
enc_in, dec_in, c_out	7	Number of input features(columns), c_out refers to number of output features
factor	5	Probsparse attention factor = **c** $(u = c \cdot lnL_Q,\ U^* = c \cdot lnL_k)$
d_model	512	Model size
s_layers e_layers	[3,2,1] 3	When '**Informerstack**', number of encoder layers in stacking encoder replicas. When 'Informer', number of encoder layers
n_heads, d_layers, d_ff, dropout, activation	8, 1, 2048, 0.05, 'gelu'	Same parameters as Vanilla Transformer
batch_size	32	Batch size
attn	'prob'	'prob' : use ProbSparse self attention 'full' : use Scaled Dot-product self attention like Vanilla Transformer
embed	'timeF'	Global time stamp embedding changes according to 'timeF', 'fixed', 'learned'

1. Set Parameters



[Encoder Stack Replicas]

- 3 replicas (s_layers=3)
- each has 3, 2, 1 encoder layers respectively

[Prediction]

- pred_len = 24
- $Y^t = \{y_1, \dots, y_{L_y} \mid y_i^t \in \mathbb{R}^{d_y}\}$
  $(L_y = 24, \quad d_y = 7)$

[Decoder Layer]

- 1 layer (d_layers=1)
- masked ProbSparse attention + Full attention

[Encoder Input]

- ETT1h data has 7 features
- $X^t = \{x_1, \dots, x_{L_x} \mid x_i^t \in \mathbb{R}^{d_x}\}$
  $(L_x = 96, \quad d_x = 512)$

[Decoder Input]

- predict all 7 features
- label_len = 48
- pred_len = 24
- $X_{de}^t = Concat(X_{token}^t, X_0^t) \in \mathbb{R}^{(L_{token}+L_y)\times d_{model}}$
  $(L_{token} = 48, \quad L_y = 24,$
  $de = L_{token} + L_y = 72)$

2. Informer – Embedding

Embed Module

```
# Embedding
self.enc_embedding = DataEmbedding(enc_in, d_model, embed, freq, dropout)
self.dec_embedding = DataEmbedding(dec_in, d_model, embed, freq, dropout)
```

DataEmbedding

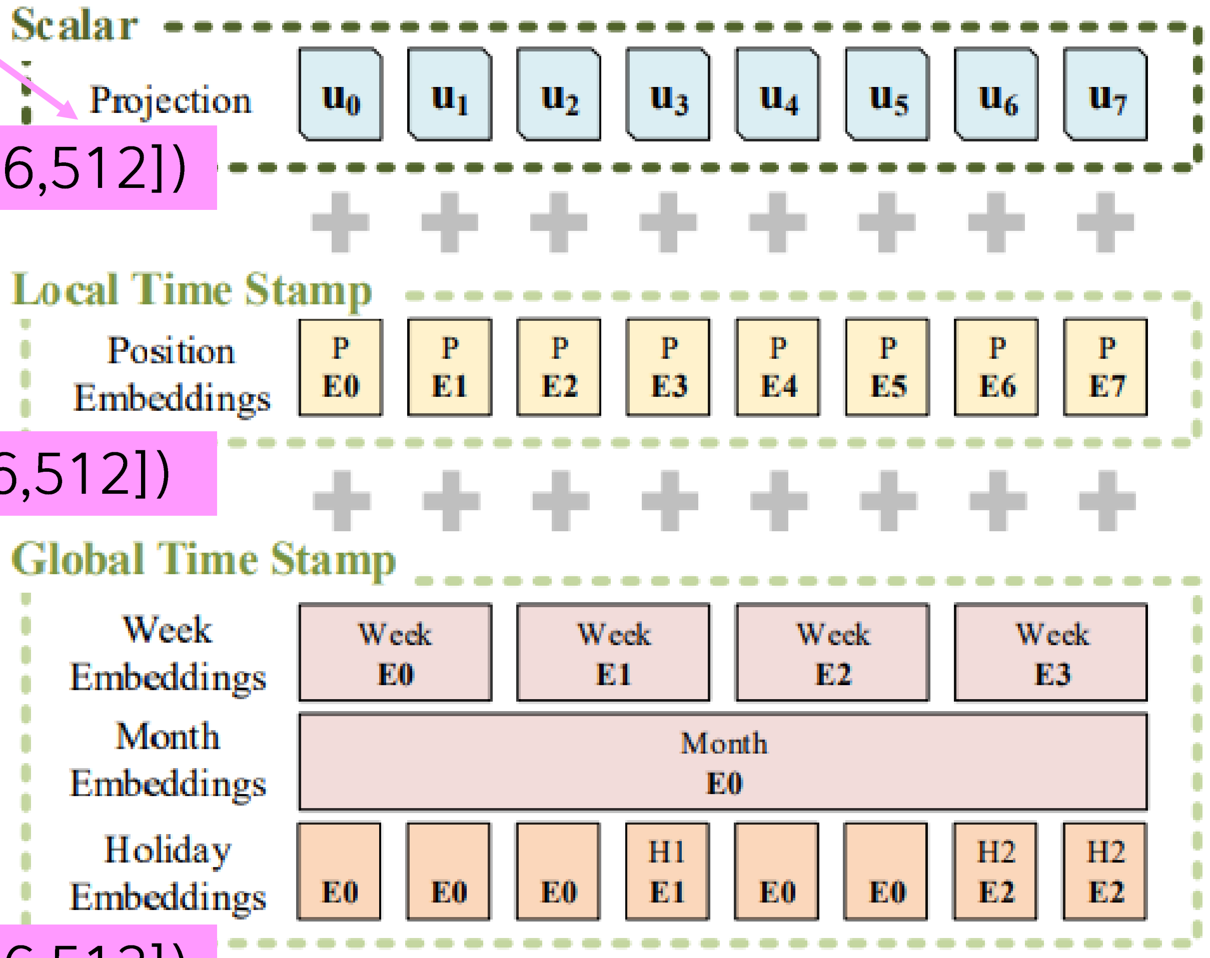
```
def forward(self, x, x_mark):
    x = self.value_embedding(x) + self.position_embedding(x) + self.temporal_embedding(x_mark)
    return self.dropout(x)
```

d_model
seq_length
batch_size

torch.Size([32,96,512])

torch.Size([1,96,512])

torch.Size([32,96,512])



- value_embedding = TokenEmbedding class
  : Conv1d(kernel_size=3)

- position_embedding = PositionalEmbedding class
  : Same as Vanilla Transformer

- temporal_embedding =
  if embed(embed_type) == ‘timeF’:
    use TimeFeatureEmbedding class
  else:
    use TemporalEmbedding & FixedEmbedding class

2. Informer – Embedding

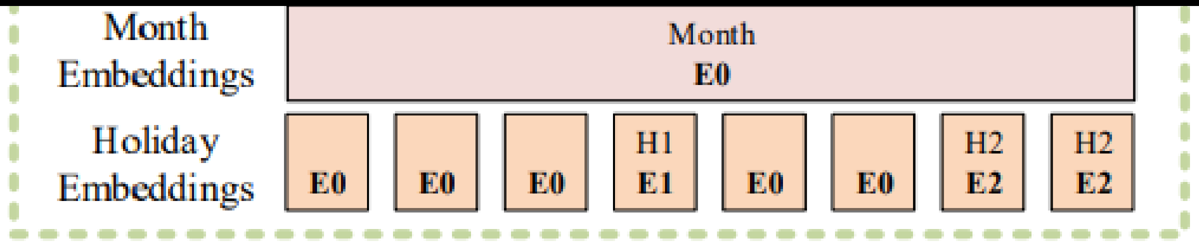
Embed Module

```
# Embedding
self.enc_embedding = DataEmbedding(enc_in, d_model, embed, freq, dropout)
self.dec_embedding = DataEmbedding(dec_in, d_model, embed, freq, dropout)
```

DataEmbedding

```
class TokenEmbedding(nn.Module):
    def __init__(self, c_in, d_model):
        super(TokenEmbedding, self).__init__()
        padding = 1 if torch.__version__>='1.5.0' else 2
        self.tokenConv = nn.Conv1d(in_channels=c_in, out_channels=d_model,
                                   kernel_size=3, padding=padding, padding_mode='circular')
        for m in self.modules():
            if isinstance(m, nn.Conv1d):
                nn.init.kaiming_normal_(m.weight,mode='fan_in',nonlinearity='leaky_relu')

    def forward(self, x):
        x = self.tokenConv(x.permute(0, 2, 1)).transpose(1,2)
        return x
```




```
    use TimeFeatureEmbedding class
else:
    use TemporalEmbedding & FixedEmbedding class
```

2. Informer – Embedding

Embed Module

```python
class PositionalEmbedding(nn.Module):
    def __init__(self, d_model, max_len=5000):
        super(PositionalEmbedding, self).__init__()
        # Compute the positional encodings once in log space.
        pe = torch.zeros(max_len, d_model).float()
        pe.require_grad = False

        position = torch.arange(0, max_len).float().unsqueeze(1)
        div_term = (torch.arange(0, d_model, 2).float() * -(math.log(10000.0) / d_model)).exp()

        pe[:, 0::2] = torch.sin(position * div_term)
        pe[:, 1::2] = torch.cos(position * div_term)

        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)

    def forward(self, x):
        return self.pe[:, :x.size(1)]
```


```
else:
  use TemporalEmbedding & FixedEmbedding class
```

2. Informer – Embedding

Model train, validation, test codes

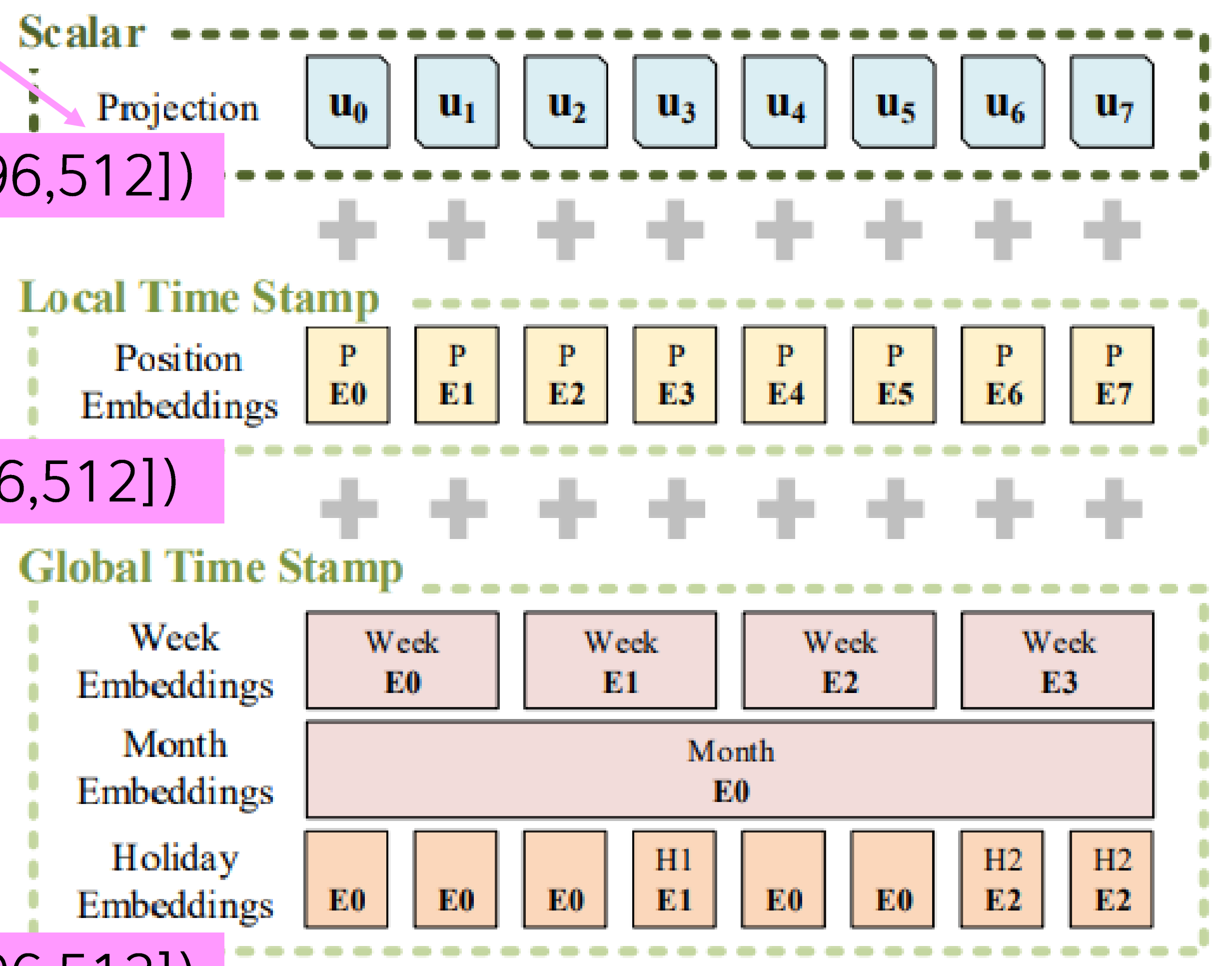
```
for i, (batch_x,batch_y,batch_x_mark,batch_y_mark) in enumerate(train_loader):
```

DataEmbedding

```
def forward(self, x, x_mark):
    x = self.value_embedding(x) + self.position_embedding(x) + self.temporal_embedding(x_mark)
    return self.dropout(x)
```

d_model
seq_length
batch_size

torch.Size([32,96,512])



torch.Size([1,96,512])

torch.Size([32,96,512])

```
- value_embedding = TokenEmbedding class
  : Conv1d(kernel_size=3)
```

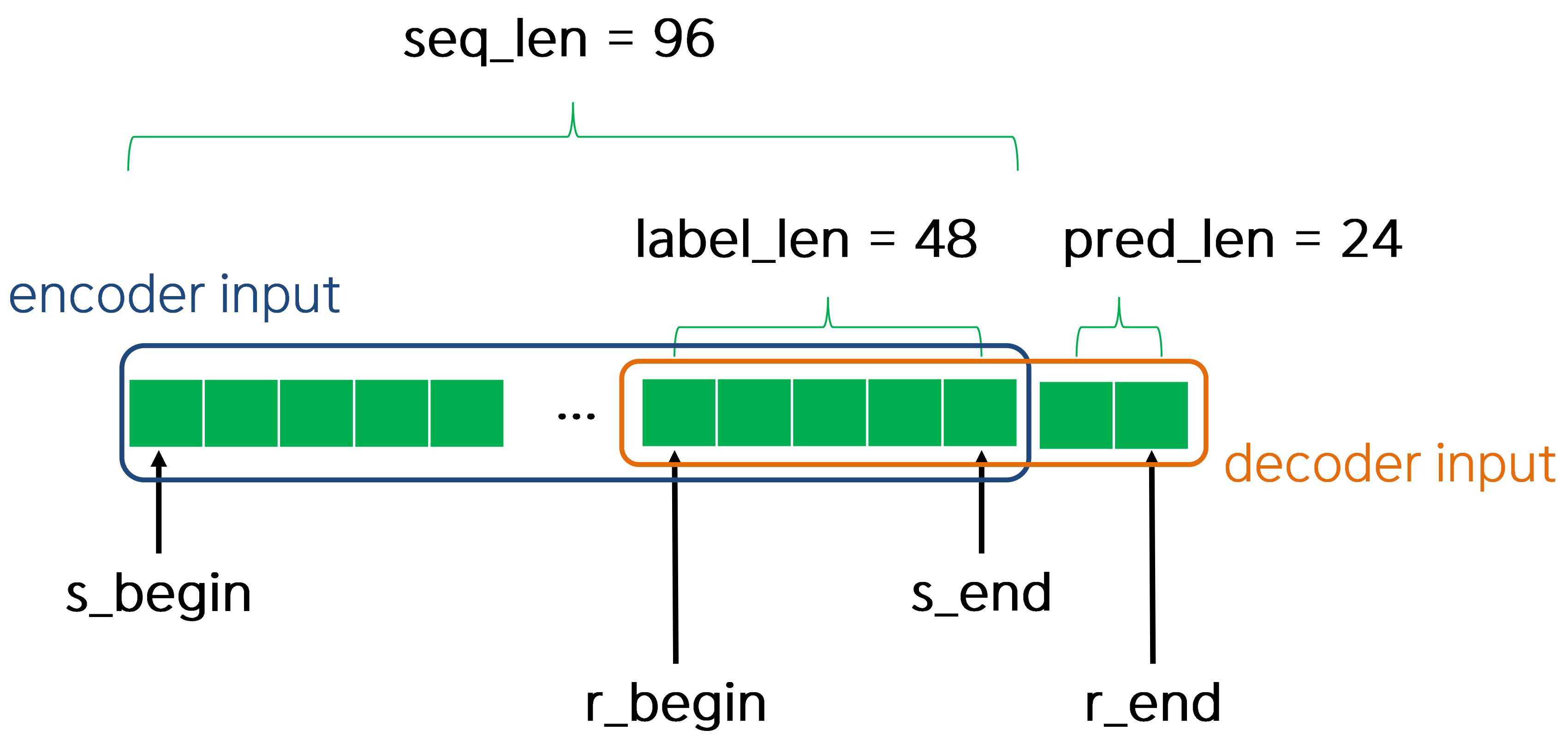
```
- position_embedding = PositionalEmbedding class
  : Same as Vanilla Transformer
```

```
- temporal_embedding =
  if embed(embed_type) == 'timeF':
    use TimeFeatureEmbedding class
  else:
    use TemporalEmbedding & FixedEmbedding class
```

2. Informer – Embedding

└ /datas/ : dataloader.py

```
class Dataset_ETT_hour(Dataset):
  ...
  ...
  def __getitem__(self, index):
    s_begin = index
    s_end = s_begin + self.seq_len
    r_begin = s_end - self.label_len
    r_end = r_begin + self.label_len + self.pred_len
    seq_x = self.data_x[s_begin:s_end]
    if self.inverse:
      seq_y = np.concatenate([self.data_x[r_begin:r_begin+self.label_len],
            self.data_y[r_begin+self.label_len:r_end]], 0)
    else:
      seq_y = self.data_y[r_begin:r_end]
      seq_x_mark = self.data_stamp[s_begin:s_end]
      seq_y_mark = self.data_stamp[r_begin:r_end]
    return seq_x, seq_y, seq_x_mark, seq_y_mark
  ...
```



__read_data__() 함수에서 utils.timefeatures.time_features() 함수를 사용하여 시간 정보 추출

```
for i, (batch_x,batch_y,batch_x_mark,batch_y_mark) in enumerate(train_loader):
```

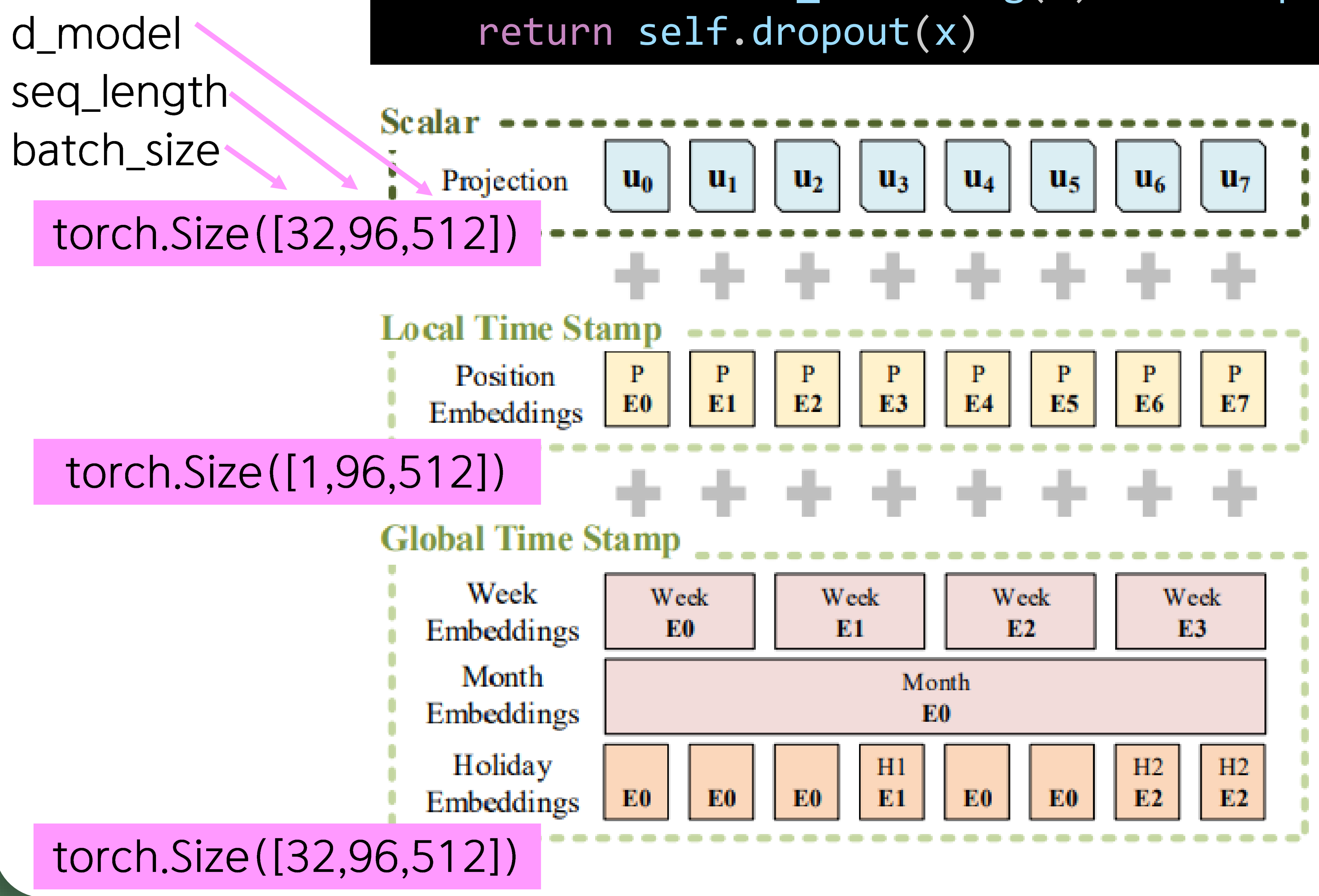
2. Informer – Embedding

Embed Module

```
# Embedding
self.enc_embedding = DataEmbedding(enc_in, d_model, embed, freq, dropout)
self.dec_embedding = DataEmbedding(dec_in, d_model, embed, freq, dropout)
```

DataEmbedding

```
def forward(self, x, x_mark):
    x = self.value_embedding(x) + self.position_embedding(x) + self.temporal_embedding(x_mark)
    return self.dropout(x)
```



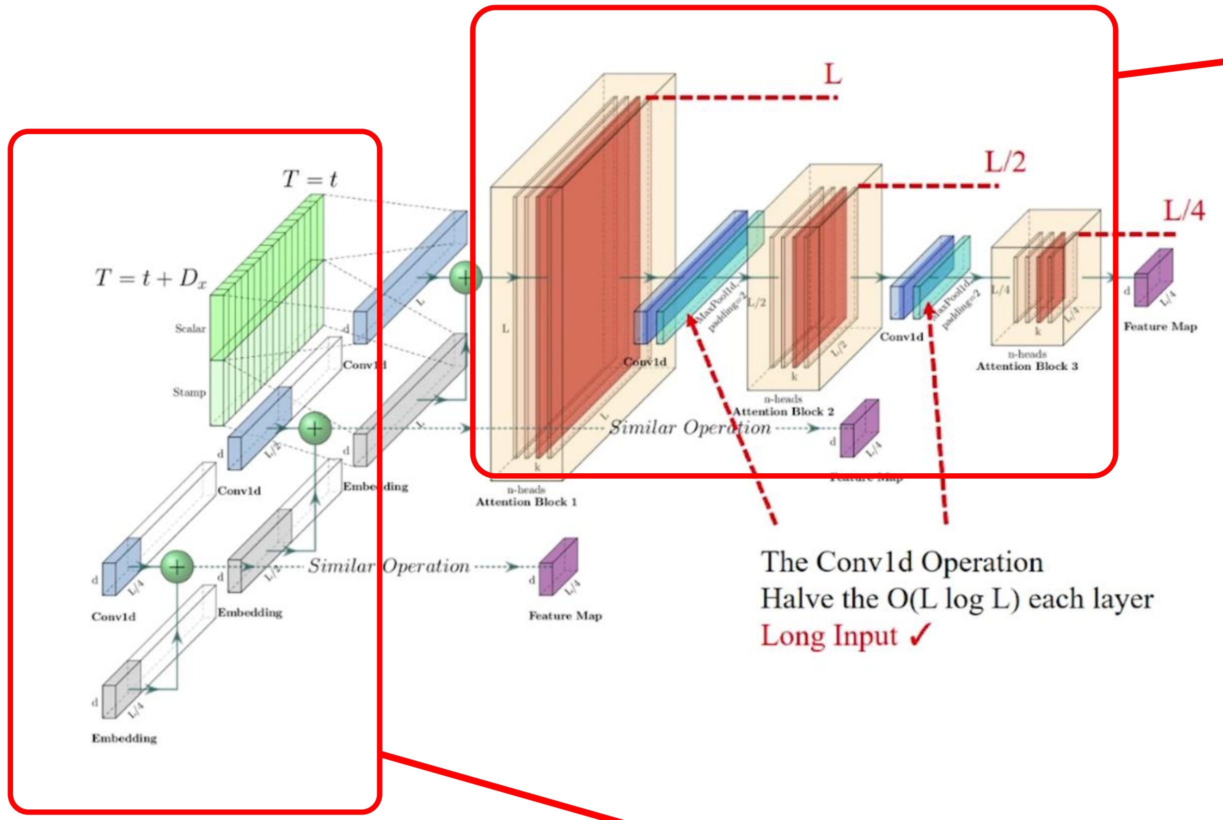
- value_embedding = TokenEmbedding class
  : Conv1d(kernel_size=3)

- position_embedding = PositionalEmbedding class
  : Same as Vanilla Transformer

- temporal_embedding =
  if embed(embed_type) == ‘timeF’:
    use TimeFeatureEmbedding class
  else:
    use TemporalEmbedding & FixedEmbedding class

3. Informer
– Encoder

Encoder Layer
= Attention + Conv1d

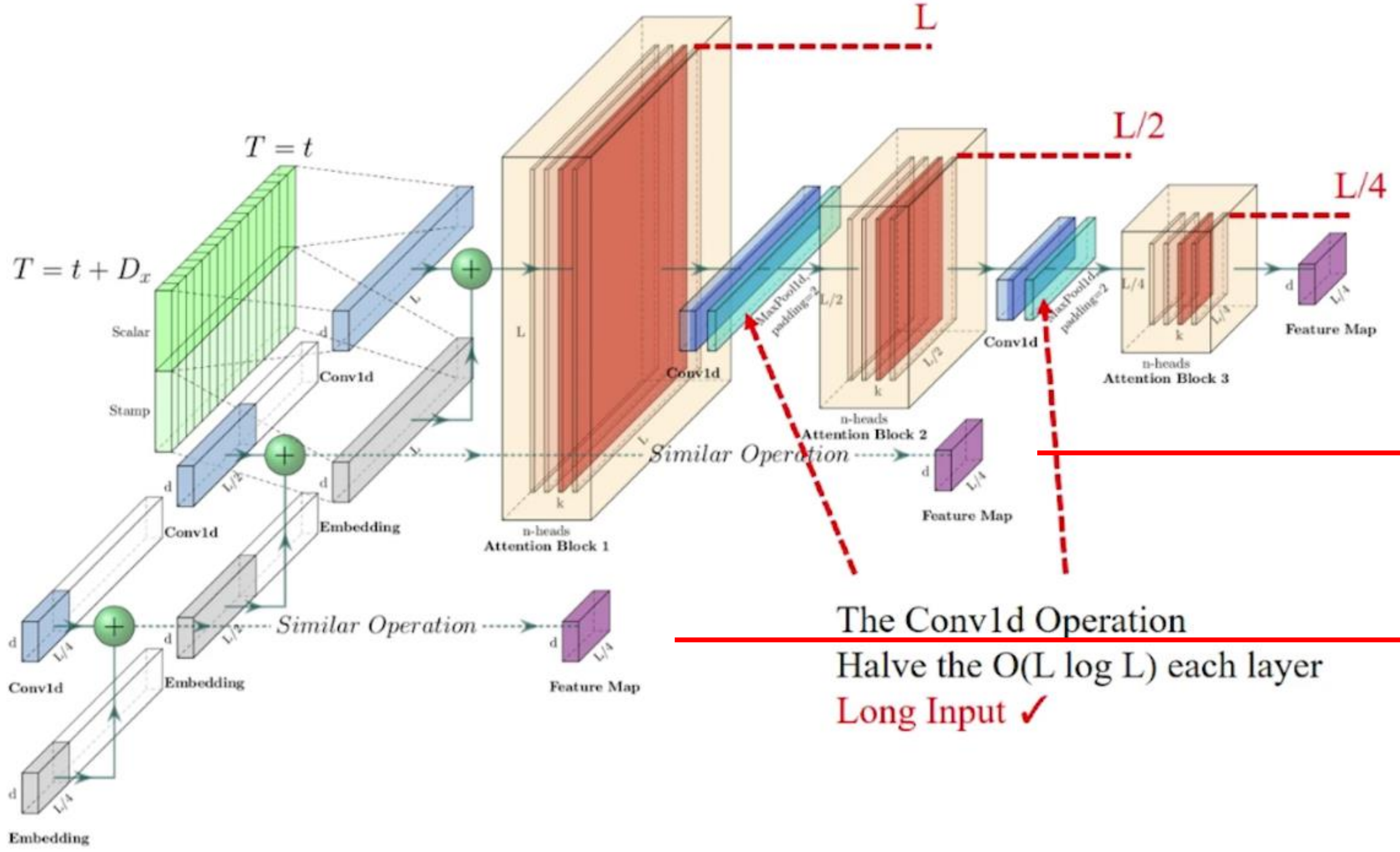


Stacking Layer Replicas

```
# Encoder
inp_lens = list(range(len(e_layers)))
encoders = [
  Encoder(
    [
      EncoderLayer(
        AttentionLayer(Attn(False, factor, attention_dropout=dropout,
          output_attention=output_attention),
          d_model, n_heads, mix=False),
          d_model,
          d_ff,
          dropout=dropout,
          activation=activation
       ) for l in range(el)
    ],
    [
      ConvLayer(
            d_model
      ) for l in range(el-1)
    ] if distil else None,
    norm_layer=torch.nn.LayerNorm(d_model)
  ) for el in e_layers]
self.encoder = EncoderStack(encoders, inp_lens)
```

3. Informer
– Encoder

EncoderStack



Stack1 – 3 encoder layers

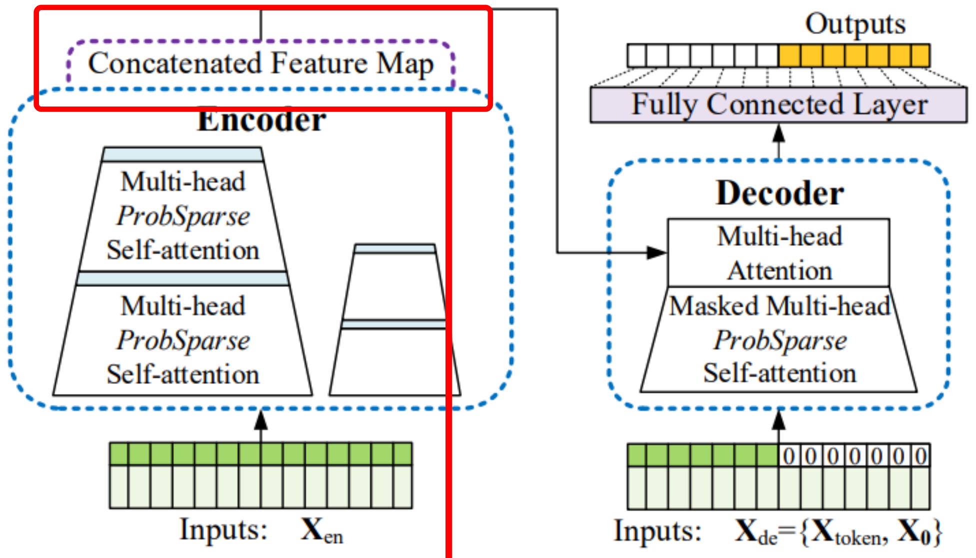
torch.Size([32,24,512])

Stack2 – 2 encoder layers

torch.Size([32,24,512])

Stack3 – 1 encoder layers

torch.Size([32,24,512])



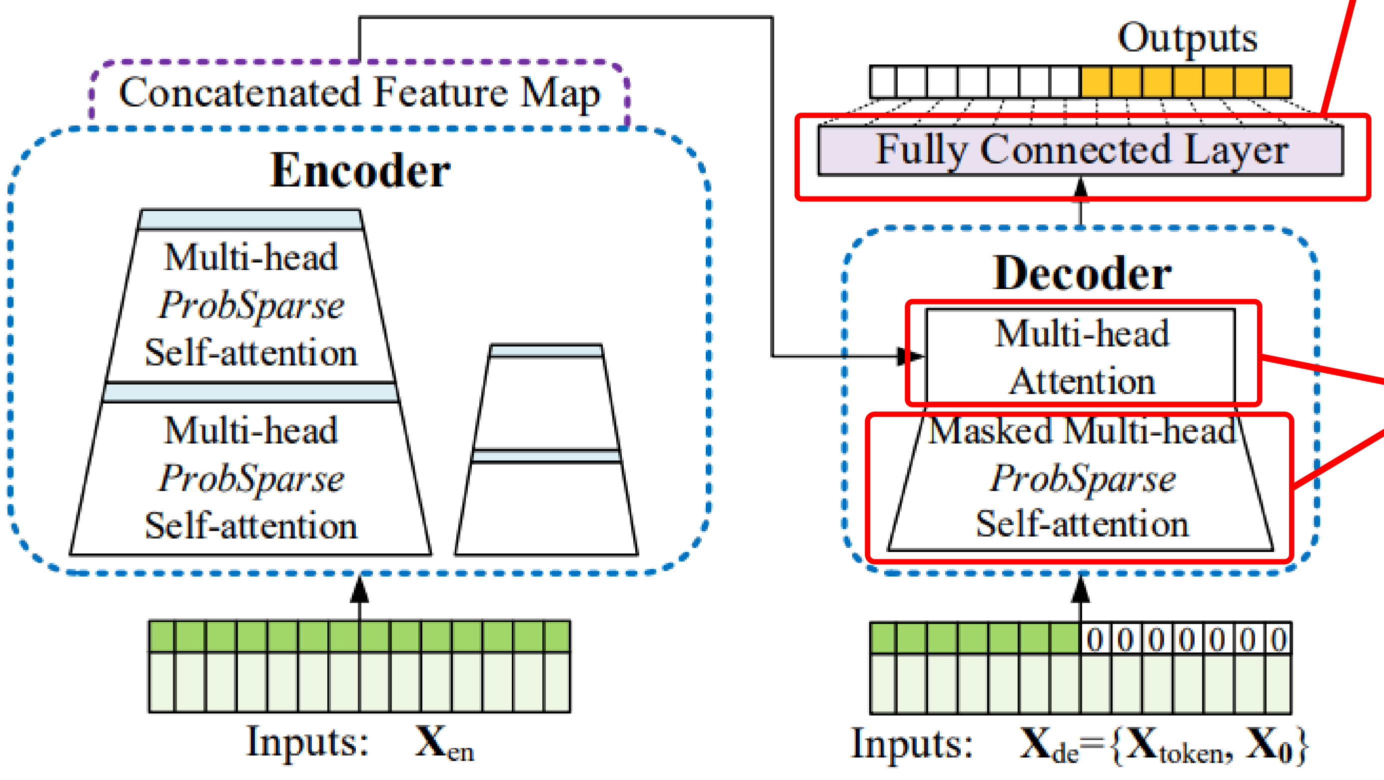
concat

torch.Size([32,72,512])

4. Informer – Decoder

Data

Conv1ds with kernel_size=1
act as FFNN



torch.Size([32,72,512])

```
y = self.dropout(self.activation(self.conv1(y.transpose(-1,1))))
y = self.dropout(self.conv2(y).transpose(-1,1))
dec_out = y
```

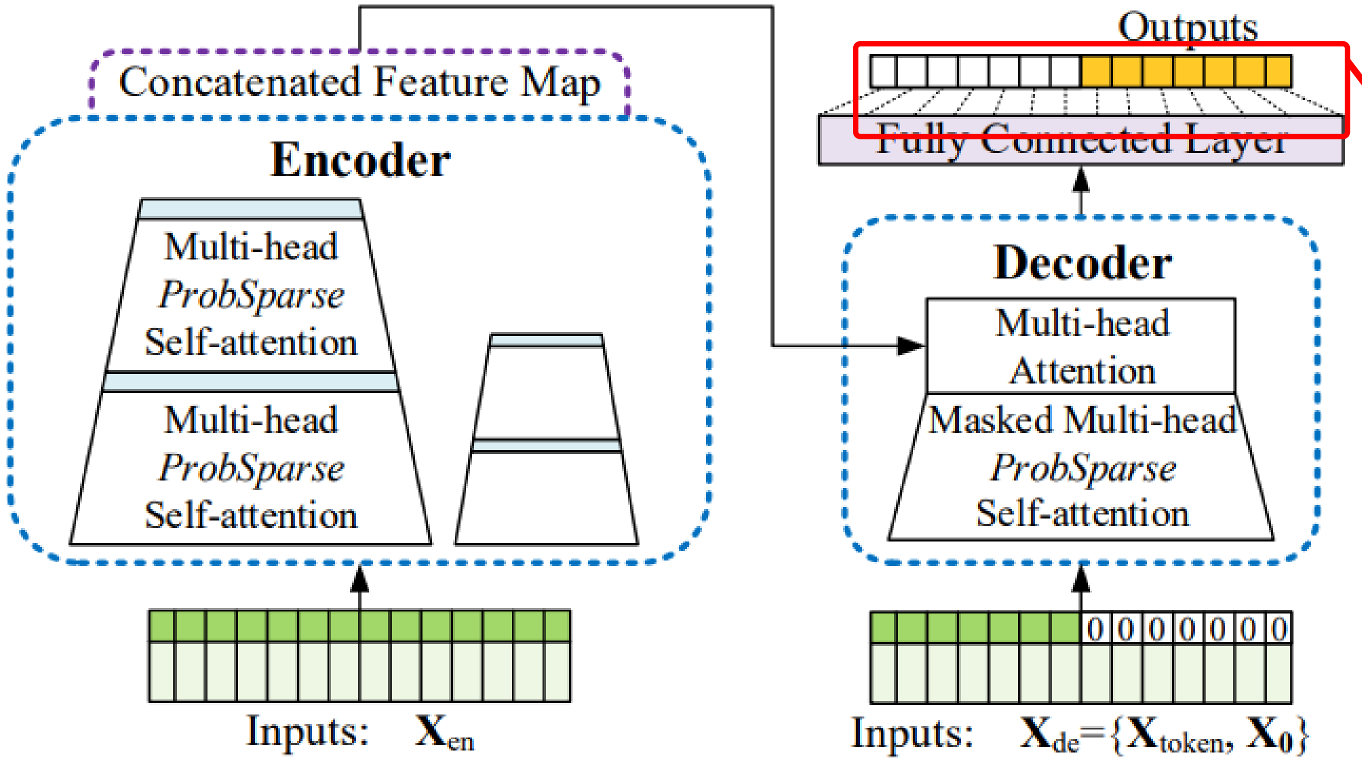
```
# Decoder
self.decoder = Decoder(
  [
    DecoderLayer(
      AttentionLayer(Attn(True, factor, attention_dropout=dropout,
        output_attention=False),
      d_model, n_heads, mix=mix),
      AttentionLayer(FullAttention(False, factor,
        attention_dropout=dropout, output_attention=False),
      d_model, n_heads, mix=False),
      d_model,
      d_ff,
      dropout=dropout,
      activation=activation,
    ) for l in range(d_layers)
  ],
  norm_layer=torch.nn.LayerNorm(d_model)
)
```

4. Informer – Decoder

Data

```
y = self.dropout(self.activation(self.conv1(y.transpose(-1,1))))
y = self.dropout(self.conv2(y).transpose(-1,1))
dec_out = y
```

torch.Size([32,72,512])




```
...
self.projection = nn.Linear(d_model, c_out, bias=True)
...

dec_out = self.projection(dec_out)
...
```

torch.Size([32,72,7])

label_len+pred_len c_out

final_prediction =

```
return dec_out[:,-self.pred_len:,:]
```

torch.Size([32,24,7])

pred_len

5. Log

```
InformerStack
<class 'models.embed.TokenEmbedding'> : torch.Size([32, 96, 512])
<class 'models.embed.PositionalEmbedding'> : torch.Size([1, 96, 512])
<class 'models.embed.TimeFeatureEmbedding'> : torch.Size([32, 96, 512])
<class 'models.embed.DataEmbedding'> : torch.Size([32, 96, 512])
encoder embedding out : torch.Size([32, 96, 512])
```

Encoder Embedding

```
<class 'models.attn.AttentionLayer'> : torch.Size([32, 96, 512]),
<class 'models.encoder.EncoderLayer'> : torch.Size([32, 96, 512])
<class 'models.encoder.ConvLayer'> : torch.Size([32, 48, 512])
<class 'models.attn.AttentionLayer'> : torch.Size([32, 48, 512]),
<class 'models.encoder.EncoderLayer'> : torch.Size([32, 48, 512])
<class 'models.encoder.ConvLayer'> : torch.Size([32, 24, 512])
<class 'models.attn.AttentionLayer'> : torch.Size([32, 24, 512]),
<class 'models.encoder.EncoderLayer'> : torch.Size([32, 24, 512])
<class 'models.encoder.Encoder'> : torch.Size([32, 24, 512])
<class 'models.attn.AttentionLayer'> : torch.Size([32, 48, 512]),
<class 'models.encoder.EncoderLayer'> : torch.Size([32, 48, 512])
<class 'models.encoder.ConvLayer'> : torch.Size([32, 24, 512])
<class 'models.attn.AttentionLayer'> : torch.Size([32, 24, 512]),
<class 'models.encoder.EncoderLayer'> : torch.Size([32, 24, 512])
<class 'models.encoder.Encoder'> : torch.Size([32, 24, 512])
<class 'models.attn.AttentionLayer'> : torch.Size([32, 24, 512]),
<class 'models.encoder.EncoderLayer'> : torch.Size([32, 24, 512])
<class 'models.encoder.Encoder'> : torch.Size([32, 24, 512])
<class 'models.encoder.EncoderStack'> : torch.Size([32, 72, 512])
encoder out : torch.Size([32, 72, 512])
```

EncoderStack

```
<class 'models.embed.TokenEmbedding'> : torch.Size([32, 72, 512])
<class 'models.embed.PositionalEmbedding'> : torch.Size([1, 72, 512])
<class 'models.embed.TimeFeatureEmbedding'> : torch.Size([32, 72, 512])
<class 'models.embed.DataEmbedding'> : torch.Size([32, 72, 512])
```

Decoder Embedding

```
decoder embedding out : torch.Size([32, 72, 512])
<class 'models.attn.AttentionLayer'> : torch.Size([32, 72, 512]),
<class 'models.attn.AttentionLayer'> : torch.Size([32, 72, 512]),
<class 'models.decoder.DecoderLayer'> : torch.Size([32, 72, 512])
<class 'models.decoder.Decoder'> : torch.Size([32, 72, 512])
decoder out : torch.Size([32, 72, 512])
decoder out projection : torch.Size([32, 72, 7])
```

Decoder

```
<class 'models.model.InformerStack'> : torch.Size([32, 24, 7])
--------------end--------------
```

6. more...

```
import os

# set saved model path
setting = 'informer_ETTh1_ftM_sl96_ll48_pl24_dm512_nh8_el2_dl1_df2048_atprob_fc5_ebtimeF_dtTrue_mxTrue_exp_0'
# path = os.path.join(args.checkpoints,setting,'checkpoint.pth')
```

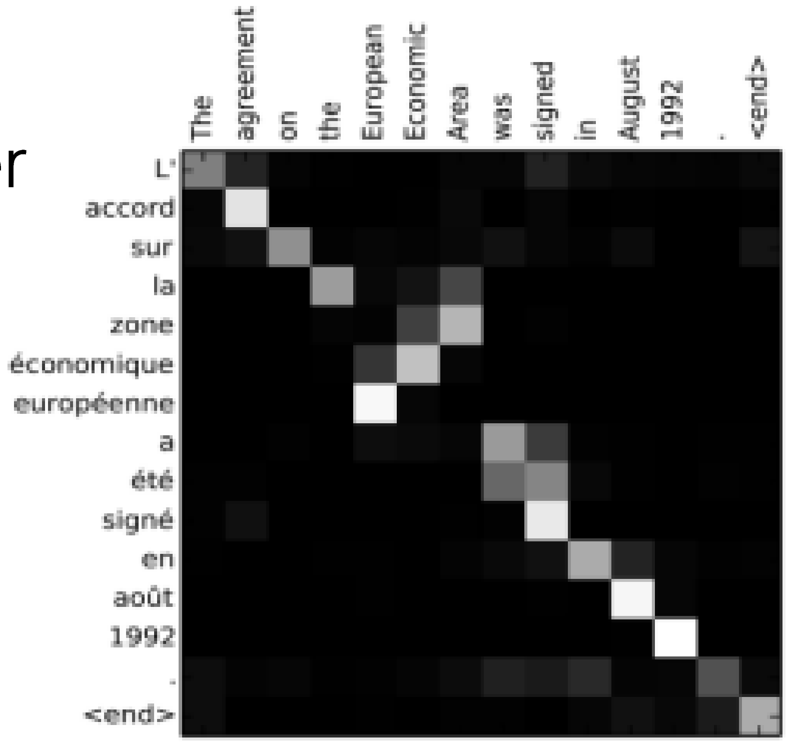
```
import os

# set saved model path
# setting = 'informer_ETTh1_ftM_sl96_ll48_pl24_dm512_nh8_el2_dl1_df2048_atprob_fc5_ebtimeF_dtTrue_mxTrue_exp_0'
# path = os.path.join(args.checkpoints,setting,'checkpoint.pth')
```

이후 코드에 등장하는 모든 setting 변수를 주석처리 해줘야 에러가 안 납니다.

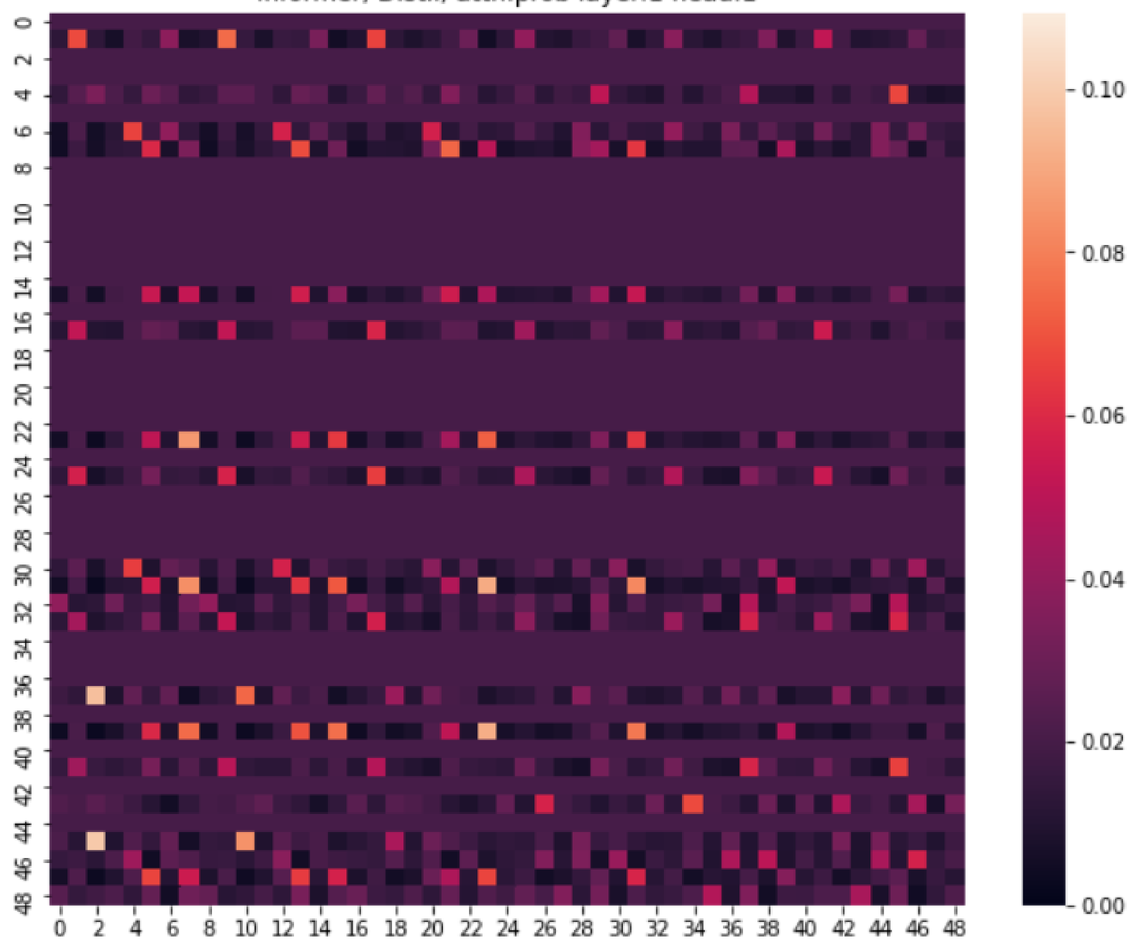
6. more...

attention of vanilla transformer

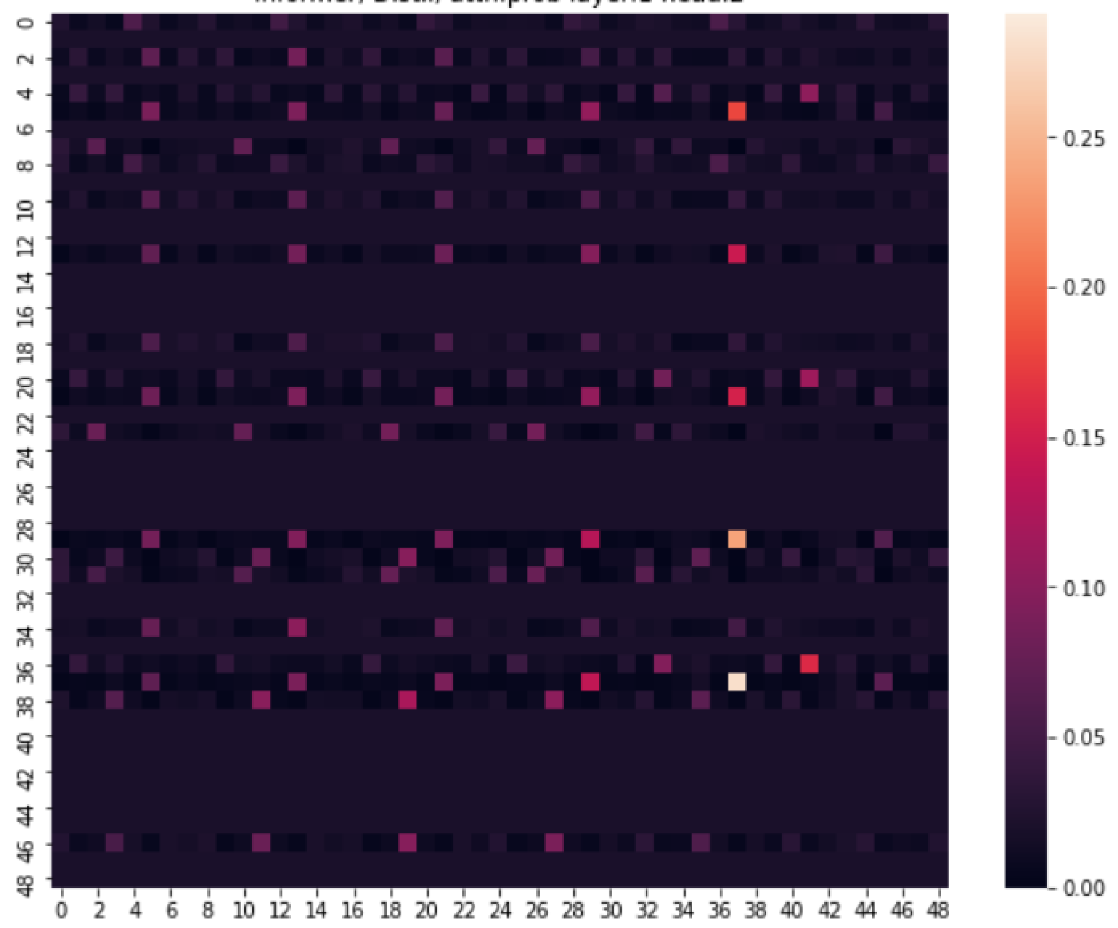


Attention visualization

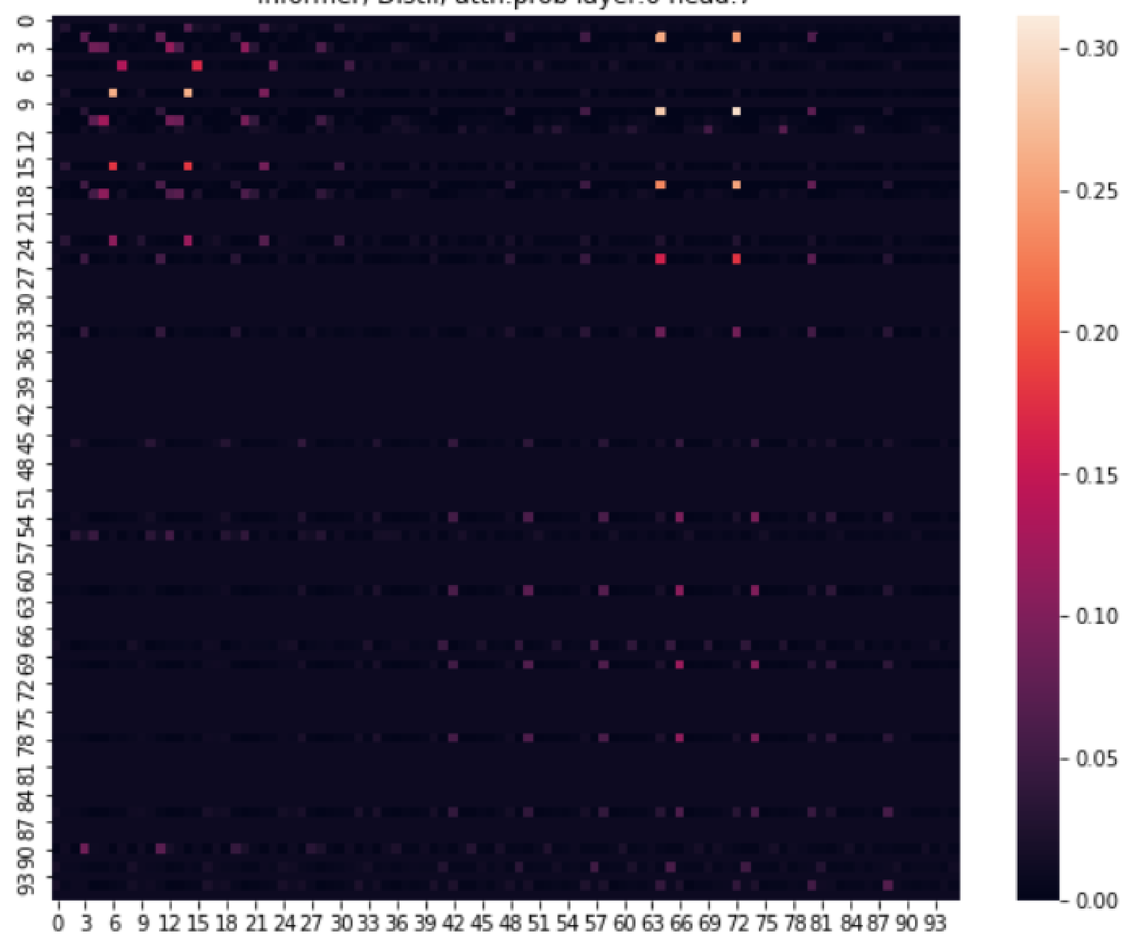
Informer, Distil, attn:prob layer:1 head:1



Informer, Distil, attn:prob layer:1 head:2



Informer, Distil, attn:prob layer:0 head:7



Model : Informer (e_layers=2, d_layers=1)

출처

- references

code example : https://colab.research.google.com/drive/1_X7O2BkFLvqyCdZzDZvV2MB0aAvYALLC#scrollTo=6mx2dnwY9dWi

official github : https://github.com/cookieminions/Informer2020/tree/dev

paper review reference : http://dsba.korea.ac.kr/seminar/?mod=document&pageid=1&keyword=informer&uid=1823

AAAI-21 presentation : https://slideslive.com/38948878/informer-beyond-efficient-transformer-for-long-sequence-timeseries-forecasting