Running environment: Python 3.6 env\_graph\_convnet

Spatio-temporal graph convolutional networks:

Datasets:

* PeMSD7\_V\_{%num\_route%}.csv : Historical Speed Records with shape of [len\_seq \* num\_road] (len\_seq = day\_slot \* num\_dates).
* PeMSD7\_W\_{%num\_route%}.csv : Weighted Adjacency Matrix with shape of [num\_road \* num\_road].

Note: please replace the %num\_route% with the number of routes in your dataset. '\*.csv' should not contain any index or header in the file.

228\* 228 W

Day\_slot \*num\_dates = 12672

According to the paper,

All the tests use 60 minutes as the historical time window, a.k.a. 12 observed data points (M =12) are used to forecast traffic conditions in the next 15, 30, and 45 minutes (H =3, 6, 9).

Evaluation Metrics:

Mean Absolute Errors (MAE)

Mean Absolute Percentage Errors (MAPE)

Root Mean Squared Errors (RMSE)

Train the models by minimizing the mean square error using RMSprop for 50 epochs with batch size as 50. The initial learning rate is 10^-3 with a decay rate of 0.7 after every 5 epochs.

Clearly, all the metrics, the smaller, the better.

Training model‘s result:

Epoch  0, Step   0: [52037.711, 687.885]

Epoch  0 Training Time 289.294s

Time Step 3: MAPE  8.778%, 8.217%; MAE  3.417, 3.319; RMSE 5.765,  5.623.

Time Step 6: MAPE 11.842%, 11.297%; MAE  4.677, 4.592; RMSE 8.080, 7.980.

Time Step 9: MAPE 13.919%, 13.251%; MAE  5.518, 5.414; RMSE 9.549, 9.463.

Epoch  0 Inference Time 256.922s

Epoch  1, Step   0: [1287.680, 680.710]

Epoch  1 Training Time 311.116s

Time Step 3: MAPE  8.325%, 7.876%; MAE  3.225, 3.131; RMSE 5.475,  5.396.

Time Step 6: MAPE 11.760%, 11.400%; MAE  4.483, 4.401; RMSE 7.926, 7.905.

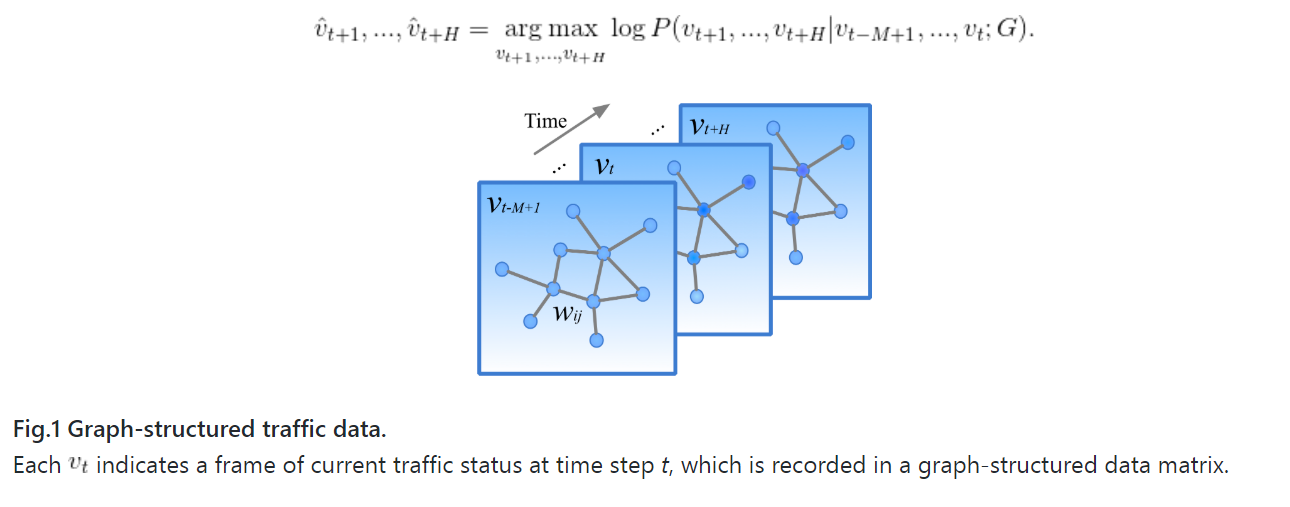
Time Step 9: MAPE 13.919%, 13.881%; MAE  5.347, 5.254; RMSE 9.501, 9.487.

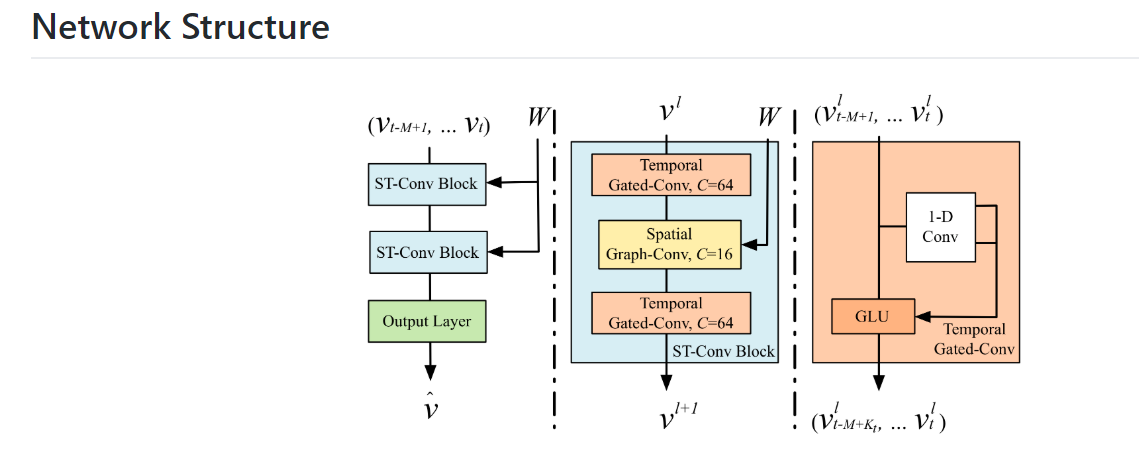
Epoch  1 Inference Time 260.327s

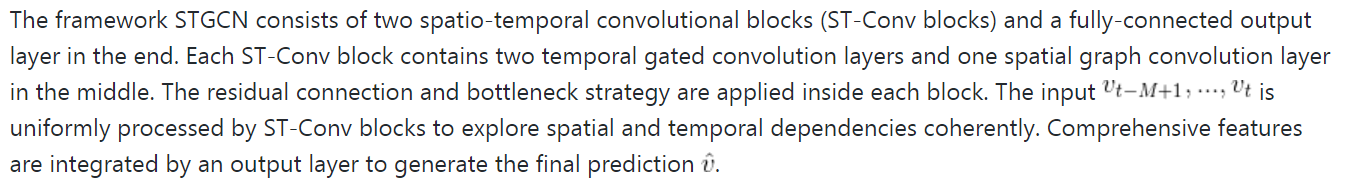
Analyze the paper:

We propose a novel deep learning framework, **STGCN**, to tackle time series prediction problem in traffic domain. Instead of applying regular convolutional and recurrent units, we formulate the problem on graphs and build the model with complete convolutional structures. To the best of our knowledge, it is the first time that to apply purely convolutional structures to extract spatio-temporal features simultaneously from graph-structured time series in a traffic study.

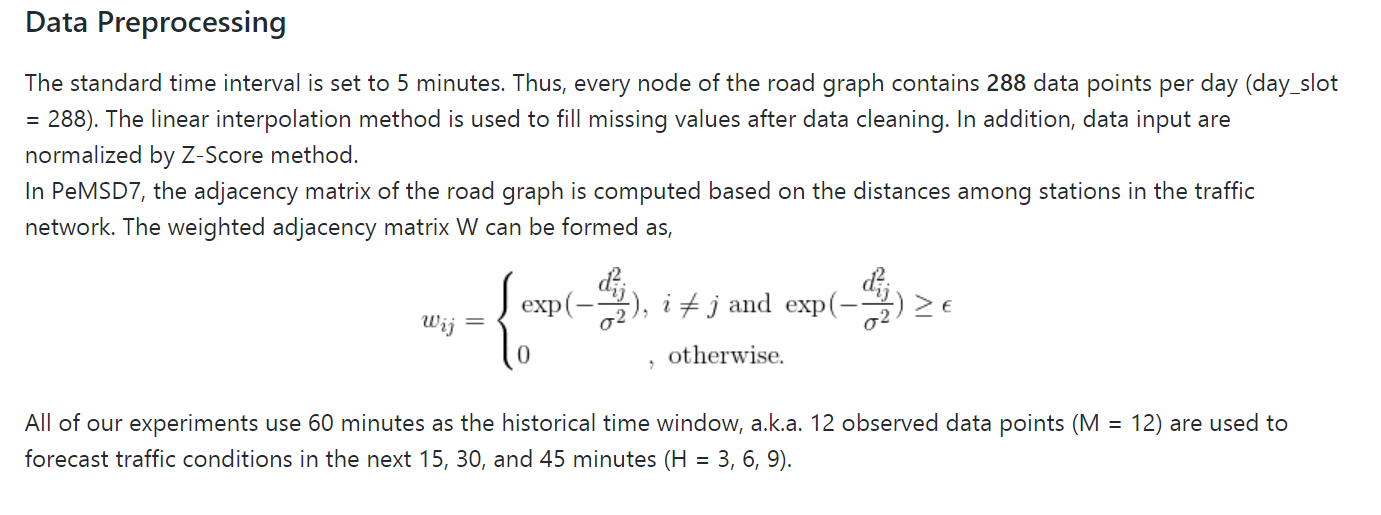
Traffic forecast is a typical time-series prediction problem, i.e. predicting the most likely traffic measurements (e.g. speed or traffic flow) in the next H time steps given the previous M observations from traffic network G as







What will be the residual connection and bottleneck strategy?



(60/5)824 =288

Dig into the code: (Data preprocessing)

N\_route = 228

N\_his = 12

N\_pred =9

Ks =3 kt =3

Blocks = [[1,32,64], [64,32,128]]

W = PeMSD7\_W\_228.csv

Data\_file = PeMSD7\_V\_228.csv

n\_train = 34, n\_val = 5, n\_test =5

Data generation process:

Data\_gen(data\_file, (n\_train, n\_val, n\_test), n\_route, n\_his + n\_pred =21)

Return : dict, dataset that contains training, validation and test with stats

N\_pred = 9, means the next 45 mins, 5 min as one slot per day by default.

Involves seq\_gen method:

Seq\_gen(len\_seq, data\_seq, offset, n\_frame, n\_route, day\_slot, C\_0=1)

Len\_seq: the length of target date sequence

Data\_seq: np.ndarray source data / time-series

Offset: the starting index of different dataset type

n\_frame: the number of frame within a standard sequence unit

n\_route: the number of routes in the graph

day\_slot = 288, the number of time slots per day

C\_0 = the size of input channel.

Seq\_train = seq\_gen(n\_train, data\_seq, 0, n\_frame, n\_route, 288)

Equivalent to a window (268\*228) go along y-axis, each step move unit step

1. Calculate n\_slot = 288-21+ 1 = 268
2. Tmp\_seq = np.zeros((34\*268, 21,228,1))

Seq\_train.shape = (9112,21,228,1)

Train\_block-> the first 34 days, each block contains 268\* 21\*228 observations

Train\_seq = (34\*268,21,228,1)

Validation\_block -> the following 5 days, each block contains 268\*21\*228 observations

Val\_seq =(5\*268,21,228,1)=(1340,21,228,1)

Test\_seq = (5\*268,21,228,1) =(1340,21,228,1)

Then, use Train\_seq’s stats, x\_stats[‘mean’] x\_stats[‘std’] to z\_score normalize

(Train\_seq, Val\_seq, Test\_seq)

Here x\_stats[‘mean’] and x\_stats[‘std’] are scalar, to get them, use np metrics, works on the whole dataset of Train\_seq, not by columns.

Dataset is a dictionary, that contains ‘train’ -> ndarray (9112,21,228,1)

‘val’ -> ndarray (1340,21,228,1)

‘test’ -> ndarray (1340,21,228,1)

Mean scalar value from train\_seq

Std scalar value from train\_seq

The table should have 288\*(34+5+5)=12672 entries

Model\_train(Dataset, blocks, args)

Models.layers.py (Analyze the equations in this paper)

Figure out what’s the graph in this work for (fixed G !!!!!!)

The W (weighted adjacency matrix of G is fixed and applied to all ST-Conv Block)

Model\_train(Dataset, blocks, args, sum\_path =’./output/tensorboard’)

:param blocks: list, channel configs of st\_conv blocks

n, n\_his, n\_pred = args.n\_route(228) , args.n\_his(12) args.n\_pred(9)

Ks, Kt = 3,3

Batch\_size, epoch inf\_mode, opt = 200, 2, ‘merge’, ‘RMSProp’

Opt(‘RMSProp’: minimizing the mean square error)

Placeholder for model training:

X = tf.placeholder(tf.float32, [None,n\_his+1, n,1], name=’data\_input’)

Why n\_his +1?

Keep\_prob = tf.placeholder(tf.float32, name=’keep\_prob’)

#Define model loss

train\_loss, pred = build\_model(X, n\_his, Ks, Kt, blocks, keep\_prob)

:param Ks: int, kernel size of spatial convolution

:param Kt: int, kernel size of temporal convolution

x = X[:,0:n\_his, :,:]

k\_0 = n\_his: kernel size of temporal convolution in the output layer.

For i, channels in enumerate(blocks):

x = st\_conv\_block(x, Ks, Kt, channels, i, keep\_prob, act\_func = ‘GLU’)

k\_0 -= 2\*(Ks -1) /\*k\_0 = k\_0 -2\*2 each time, each time dimension reduce 4\*/

/\* channels = [[1,32,64], [64,32,128]]

Enumerate(blocks) return 0 , [1,32,64] 1, [64,32,128]

k\_0 = 12-4-4 =4

In the output\_layer

#maps multi-steps to one.

With tf.variable\_scope(f’{scope}\_in’):

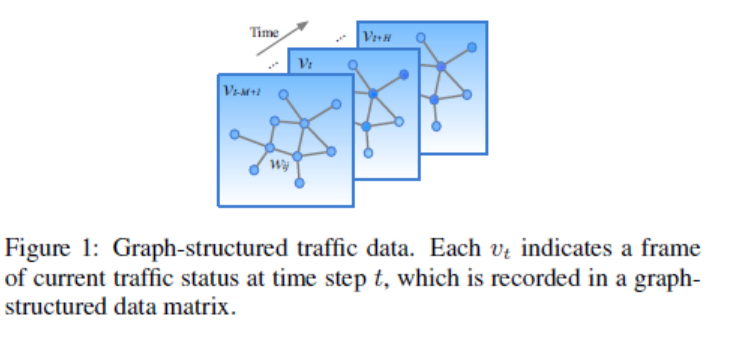
x\_i = temporal\_conv\_layer(x, T, channel, channel, act\_func = act\_func)

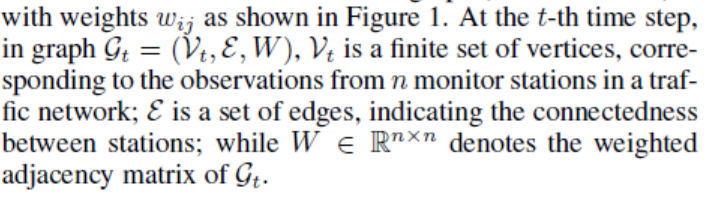
x\_ln = layer\_norm(x\_i, f’layer\_norm\_{scope}’)

with tf.variable\_scope(f’{scope}\_out’):

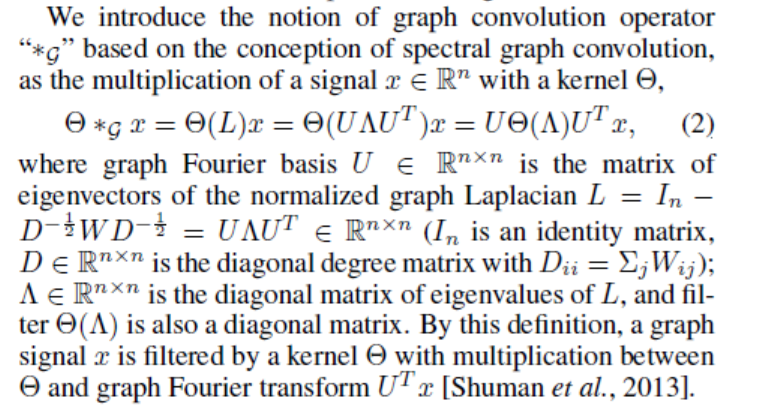
x\_o = temporal\_conv\_layer(x\_ln, 1, channel, channel, act\_func = ‘sigmoid’)

Paper reading:

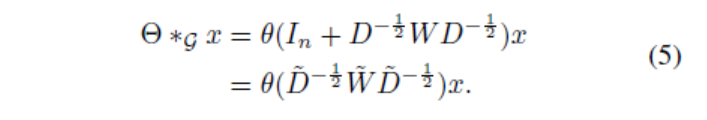




This paper follows GCN- spectral graph convolution,



In the DATASET, each row represents at one time stamp, each entry represents a node in the G\_t.



Mapping to GCN layers:

|  |
| --- |
| def gconv(x, theta, Ks, c\_in, c\_out): |
|  | ''' |
|  | Spectral-based graph convolution function. |
|  | :param x: tensor, [batch\_size, n\_route, c\_in]. |
|  | :param theta: tensor, [Ks\*c\_in, c\_out], trainable kernel parameters. |
|  | :param Ks: int, kernel size of graph convolution. |
|  | :param c\_in: int, size of input channel. |
|  | :param c\_out: int, size of output channel. |
|  | :return: tensor, [batch\_size, n\_route, c\_out]. |
|  | ''' |
|  | # graph kernel: tensor, [n\_route, Ks\*n\_route] |
|  | kernel = tf.get\_collection('graph\_kernel')[0] |
|  | n = tf.shape(kernel)[0] |
|  | # x -> [batch\_size, c\_in, n\_route] -> [batch\_size\*c\_in, n\_route] |
|  | x\_tmp = tf.reshape(tf.transpose(x, [0, 2, 1]), [-1, n]) |
|  | # x\_mul = x\_tmp \* ker -> [batch\_size\*c\_in, Ks\*n\_route] -> [batch\_size, c\_in, Ks, n\_route] |
|  | x\_mul = tf.reshape(tf.matmul(x\_tmp, kernel), [-1, c\_in, Ks, n]) |
|  | # x\_ker -> [batch\_size, n\_route, c\_in, K\_s] -> [batch\_size\*n\_route, c\_in\*Ks] |
|  | x\_ker = tf.reshape(tf.transpose(x\_mul, [0, 3, 1, 2]), [-1, c\_in \* Ks]) |
|  | # x\_gconv -> [batch\_size\*n\_route, c\_out] -> [batch\_size, n\_route, c\_out] |
|  | x\_gconv = tf.reshape(tf.matmul(x\_ker, theta), [-1, n, c\_out]) |
|  | return x\_gconv |

GCN essentially have two matmul operations, one is with kernel, to get x\_ker

One is with theta [Ks\*c\_in, c\_out], trainable parameters.

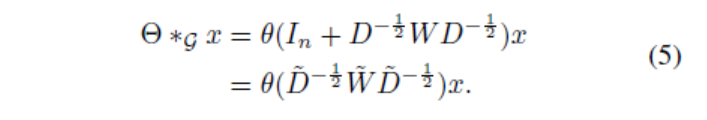
Ks maps to how many GCN layers

Spatial Graph-Conv part use gconv layer.

tf.get\_collection(‘graph\_kernel’)[0]

The list of values in the collection with the given name, or an empty list if no value has been added to that collection. The list contains the values in the order under which they were collected.

#graph kernel tensor [n\_route, Ks\*n\_route] , precomputed from



Temporal\_conv\_layer(x, Kt, c\_in, c\_out, act\_func = ‘relu’)

:param x: tensor, [batch\_size, time\_step, n\_route, c\_in]

(?, 12, 228,1)

:param Kt: int, kernel size of temporal convolution.

(3)

:param c\_in: int, size of input channel.

(1)

:param c\_out: int, size of output channel.

(32)

|  |
| --- |
|  |
| def temporal\_conv\_layer(x, Kt, c\_in, c\_out, act\_func='relu'): |
|  | ''' |
|  | Temporal convolution layer. |
|  | :param x: tensor, [batch\_size, time\_step, n\_route, c\_in]. |
|  | :param Kt: int, kernel size of temporal convolution. |
|  | :param c\_in: int, size of input channel. |
|  | :param c\_out: int, size of output channel. |
|  | :param act\_func: str, activation function. |
|  | :return: tensor, [batch\_size, time\_step-Kt+1, n\_route, c\_out]. |
|  | ''' |
|  | \_, T, n, \_ = x.get\_shape().as\_list() |
|  |  |
|  | if c\_in > c\_out: |
|  | w\_input = tf.get\_variable('wt\_input', shape=[1, 1, c\_in, c\_out], dtype=tf.float32) |
|  | tf.add\_to\_collection(name='weight\_decay', value=tf.nn.l2\_loss(w\_input)) |
|  | x\_input = tf.nn.conv2d(x, w\_input, strides=[1, 1, 1, 1], padding='SAME') |
|  | elif c\_in < c\_out: |
|  | # if the size of input channel is less than the output, |
|  | # padding x to the same size of output channel. |
|  | # Note, \_.get\_shape() cannot convert a partially known TensorShape to a Tensor. |
|  | x\_input = tf.concat([x, tf.zeros([tf.shape(x)[0], T, n, c\_out - c\_in])], axis=3)  # x\_input (?,12,228,32) |
|  | else: |
|  | x\_input = x |
|  |  |
|  | # keep the original input for residual connection. |
|  | x\_input = x\_input[:, Kt - 1:T, :, :]  # x\_input just remove the first 2 rows observation  # prepare temporal convolution  # (?, 10, 228, 32) |
|  |  |
|  | if act\_func == 'GLU': |
|  | # gated liner unit |
|  | wt = tf.get\_variable(name='wt', shape=[Kt, 1, c\_in, 2 \* c\_out], dtype=tf.float32)  # (3,1,1,64) |
|  | tf.add\_to\_collection(name='weight\_decay', value=tf.nn.l2\_loss(wt)) |
|  | bt = tf.get\_variable(name='bt', initializer=tf.zeros([2 \* c\_out]), dtype=tf.float32) |
|  | x\_conv = tf.nn.conv2d(x, wt, strides=[1, 1, 1, 1], padding='VALID') + bt  #(?,10,228,64) |
|  | return (x\_conv[:, :, :, 0:c\_out] + x\_input) \* tf.nn.sigmoid(x\_conv[:, :, :, -c\_out:])  #(?,10,228,32) |
|  | else: |
|  | wt = tf.get\_variable(name='wt', shape=[Kt, 1, c\_in, c\_out], dtype=tf.float32) |
|  | tf.add\_to\_collection(name='weight\_decay', value=tf.nn.l2\_loss(wt)) |
|  | bt = tf.get\_variable(name='bt', initializer=tf.zeros([c\_out]), dtype=tf.float32) |
|  | x\_conv = tf.nn.conv2d(x, wt, strides=[1, 1, 1, 1], padding='VALID') + bt |
|  | if act\_func == 'linear': |
|  | return x\_conv |
|  | elif act\_func == 'sigmoid': |
|  | return tf.nn.sigmoid(x\_conv) |
|  | elif act\_func == 'relu': |
|  | return tf.nn.relu(x\_conv + x\_input) |
|  | else: |
|  | raise ValueError(f'ERROR: activation function "{act\_func}" is not defined.') |
|  |  |

The temporal convolution explores Kt neighbors of input elements without padding which leading to shorten the length of sequences by Kt -1 each time.

The input can be regarded as a length-M sequence with C\_i channels.

wt = tf.get\_variable(name='wt', shape=[Kt, 1, c\_in, 2 \* c\_out], dtype=tf.float32)

(convolution kernel \tau)

The created or existing Variable (or PartitionedVariable, if a partitioner was used).

For one ST-Conv block, the \tau\_0 and \tau\_1, \theta are all learning parameters.

Output\_layer:

|  |
| --- |
|  |
| def output\_layer(x, T, scope, act\_func='GLU'): |
|  | ''' |
|  | Output layer: temporal convolution layers attach with one fully connected layer, |
|  | which map outputs of the last st\_conv block to a single-step prediction. |
|  | :param x: tensor, [batch\_size, time\_step, n\_route, channel]. |
|  | :param T: int, kernel size of temporal convolution. |
|  | :param scope: str, variable scope. |
|  | :param act\_func: str, activation function. |
|  | :return: tensor, [batch\_size, 1, n\_route, 1]. |
|  | ''' |
|  | \_, \_, n, channel = x.get\_shape().as\_list() |
|  |  |
|  | # maps multi-steps to one. |
|  | with tf.variable\_scope(f'{scope}\_in'): |
|  | x\_i = temporal\_conv\_layer(x, T, channel, channel, act\_func=act\_func) |
|  | x\_ln = layer\_norm(x\_i, f'layer\_norm\_{scope}') |
|  | with tf.variable\_scope(f'{scope}\_out'): |
|  | x\_o = temporal\_conv\_layer(x\_ln, 1, channel, channel, act\_func='sigmoid') |
|  | # maps multi-channels to one. |
|  | x\_fc = fully\_con\_layer(x\_o, n, channel, scope) |
|  | return x\_fc |
|  |  |

:param x (?,4,228,128)

Output\_layer:

Temporal\_conv\_layer

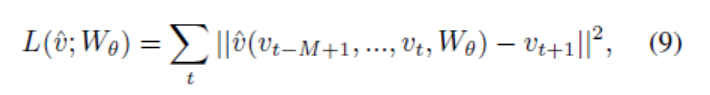
Layer\_norm

Temporal\_conv\_layer

=> (?,1,228,128)

Use the fully\_con\_layer to do the prediction:

* (?,1,228,1)



Dataset description:

PeMSD7 dataset in the weekdays of May and June of 2012

Dataset is a dictionary, that contains ‘train’ -> ndarray (9112,21,228,1)

‘val’ -> ndarray (1340,21,228,1)

‘test’ -> ndarray (1340,21,228,1)

Mean scalar value from train\_seq

Std scalar value from train\_seq

The table should have 288\*(34+5+5)=12672 entries

STGCN(Cheb) -> set the k\_s (spatial convolution kernel size as 3)

STGCN(1st) -> set the k\_s (spatial convolution kernel size as 1)