Spatio-Temporal Graph Convolutional Networks and Vertical Federated learning

1 Spatio-Temporal Graph Convolutional Networks

Supposing we have graph $\mathcal{G}$ and data point that evolve in time X_m^t with m indexing the nodes on the graph,the vector of which we will denote $\mathcal{X}^t$ that evolves in time slices of δt, that is a full dataset:

$$\mathcal{D} = \{\mathcal{X}_i^t : t = n\delta t \quad \text{for} \quad n \leq k \quad \text{and} \quad i \leq \text{Num Samples}\}$$

which the model works with. We can use STGCNs to predict how the graph data evolves and make predictions. The STGCN block is constructed so to exploit both spatial and temporal relations and consists of:

- Temporal gated convolution over time of features for all nodes separately.
- Graph convolution of the output over the nodes at fixed time.
- Temporal gated convolution over time of features for all nodes separately again.

The output of which we feed into fully connected layers or other STGC blocks

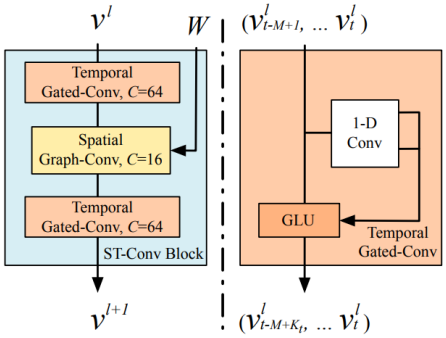

Figure 1: STCG and Temporal gated blocks

1.1 Example:

Consider now for an example usage that we know $\mathcal{G}$ that is we have some number n of nodes, which represent speed cameras, and we know the edges as we know the distances between them from which we can calculate a corresponding Adjugate and hence Edge index matrices.

We present the following problem: Given some previous data $\{\mathcal{X}_t : t = n\delta t \quad \text{for} \quad n = 0, 1, 2...m\}$ we want to predict $\{\mathcal{X}_t : t = n\delta t \quad \text{for} \quad n = m + 1, 1, 2...l\}$ for some $m, l \leq k$.

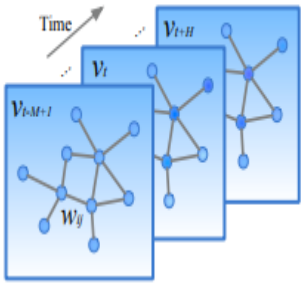

Figure 2: Temporal evolution of data

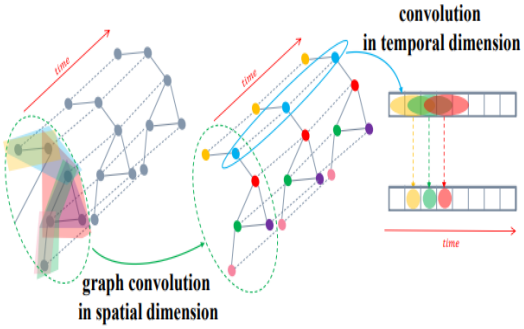

Figure 3: STGCN layer

A good dataset to test above is: PeMSD7, 'PeMSD7 is traffic data in District 7 of California consisting of the traffic speed of 228 sensors while the period is from May to June in 2012 (only weekdays) with a time interval of 5 minutes.'. In our implementation we pool to values in every 30 minutes to speed up the learning.

The implementation of the architecture above with a single layer in Pytorch is in Listing 1.

We could significantly improve above if we instead of only working in δt intervals we for example feed into the fully connected layers three different outputs of the STGC, for the δt, hourly, and weekly time steps.

```
class TemporalGatedConv1d(nn.Module):
    def __init__(self, in_channels, out_channels, kernel_size, stride=1, padding=1):
        super(TemporalGatedConv1d, self).__init__()
        self.conv_gate = nn.Conv1d(in_channels, out_channels, kernel_size, stride=stride, pa
        self.conv_linear = nn.Conv1d(in_channels, out_channels, kernel_size, stride=stride,
    def forward(self, x):
        gate_output = torch.sigmoid(self.conv_gate(x))
        linear_output = self.conv_linear(x)
        gated_output = gate_output * F.relu(linear_output)
        return gated_output
class SpatialGraphConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(SpatialGraphConv, self).__init__()
        self.conv = GCNConv(in_channels, out_channels)
    def forward(self, x):
        return F.relu(self.conv(x, edge_index))
class TemporalGraphConvNet(nn.Module):
    def __init__(self):
        super(TemporalGraphConvNet, self).__init__()
        self.temporal_gated_conv1 = TemporalGatedConv1d(in_channels=num_features, out_channe
        self.spatial_graph_conv = SpatialGraphConv(in_channels=64, out_channels=16)
        self.temporal_gated_conv2 = TemporalGatedConv1d(in_channels=16, out_channels=64, ker
        self.linear_layer1 = nn.Linear(64*num_time_steps*num_features, 200)
        self.linear_layer2 = nn.Linear(200, forecast_horizon)
    def forward(self, x):
        batch_size, num_time_steps, num_nodes, num_features = x.size()
        x = x.view(batch_size  * num_nodes,num_features, num_time_steps)
        x = self.temporal_gated_conv1(x)
        x = x.view(batch_size*num_time_steps*num_features, num_nodes,64)
        x = self.spatial_graph_conv(x)  #
        x = x.view(-1,  16, num_time_steps)
        x = self.temporal_gated_conv2(x)
        x = x.view(batch_size*num_nodes,64*num_time_steps*num_features)
        x = F.relu(self.linear_layer1(x))
        x = self.linear_layer2(x)
        x = x.view(batch_size,forecast_horizon,num_nodes  )
        return x
```

Listing 1: STGC

2 Horizontal and Vertical Federated learning

Keeping data centrally is not always optimal due to privacy or storage concerns, hence given a server $\mathcal{S}$ running a model $\mathcal{M}$ and a set of clients $\{\mathcal{C}_n\}_n$ each client downloads the model and trains it on its own data developing into different models M_n sending the updates, outputs weights, to the server which usually averages over the weights to update $\mathcal{M}$. After enough iterations the clients download the new update model. This is usually known as (Horizontal) federated learning.

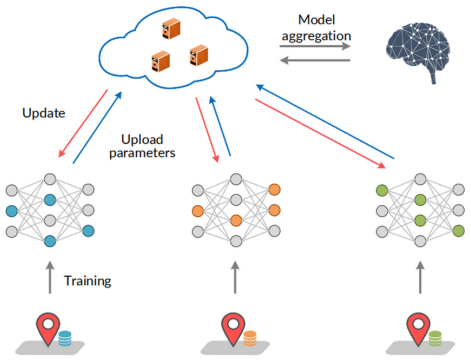

Figure 4: Algorithm sketch

The process above makes a lot of assumption most importantly it assumes that the data on each of clients is of the same type, and that the targets are kept locally for the training set however this is clearly not always true. Consider if a patient goes to a local general hospital and the hospital produces data D_1 on him, now the patient also goes to a specialist and produces new data D_2, the two hospitals will not always keep track of the exact same features. We say that The data is not distributed Horizontally but vertically.

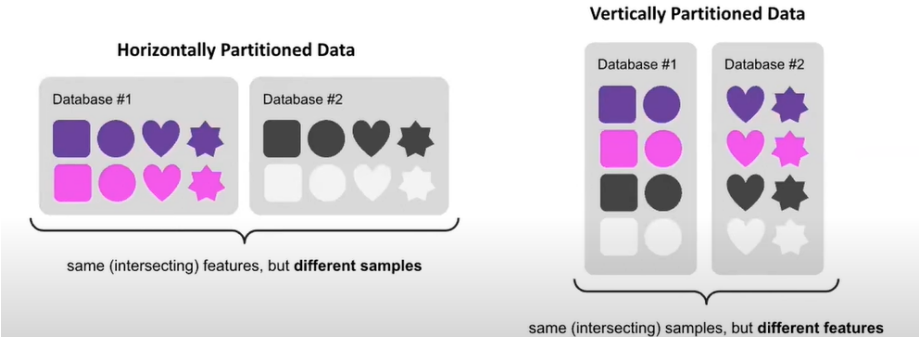

Figure 5: Here color will represent different ids, that is in the example different patients, and the shapes different features.

Now consider if instead locally for each client $\mathcal{C}_n$ we use an appropriate model M_n for the data, this will have some outputs $M_n(x_n) = y_n$, known as embeddings where x_n are the inputs. We send these embeddings to the server which runs them through its own model $\mathcal{M}$ and commutes the loss function as it only contains the targets. Through backpropagation know we find the gradients which are sent back to update the local models.

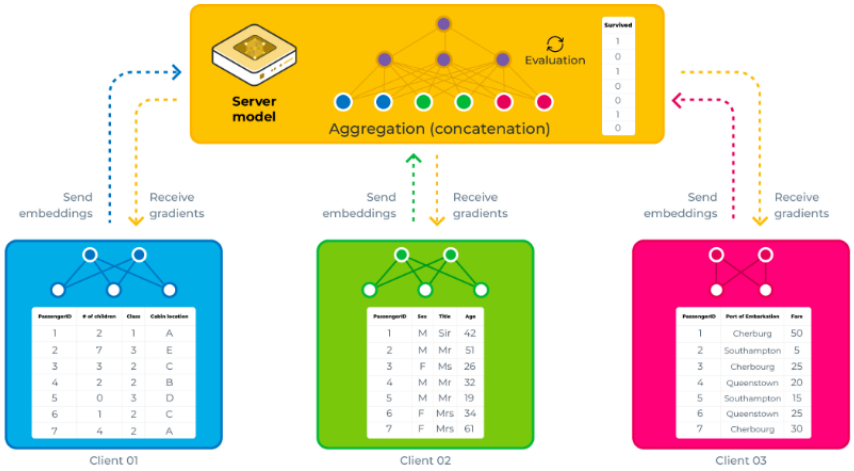

Figure 6: Diagram of architecture

The method above still have some different issues namely:

- From the weight or gradient held on the server it is sometimes possible to reconstruct client data, this problem can be adverted by adding noise or different methods in which you update the server model, for example in the horizontal case instead of averaging weights, that is summing all of the weight and dividing by n, number of clients, we could calculate $w = \sum_i X_i w_i$ with $\sum_i X_i = 1$, where X_i's are uniformly or normally distributed. The area of differential privacy mathematically guarantees protection of data, more widely this is an area of interest in information theory.

- In both the horizontal and vertical cases, clients will label user data by different methods say email, names or phone numbers to have the user ids mach they do need to reveal some information to each other, but this can be sensitive information as well.

The methods above can be applied through packages such as Flower and PyVertical

3 STGC Networks and Vertical Federated learning for Traffic forecasting

Suppose in some city called Machinetown we have a local government collecting at certain locations, nodes, precise traffic data, now in Machinetown the government has an incentive to make the traffic more predicable.

Now suppose in the town there are multiple ride sharing and taxi private companies, legally obliged to keep their data private. These companies would have an incentive to be able to known about the traffic situation.

So suppose The Machinetown government makes them a following offer, in exchange for the precise traffic prediction the companies would locally train the model and send the output the government run server.

In this framework a vertical approach would be appropriate as the local and company data would be of different shapes.

The Machinetown authorities collect data for certain nodes, we could pool the companies data to the graph form as well, that is saying for example this car took this long to go from being closes to node A to node B. Now we have spatial temporal data that we could use STGCN's on to train the data locally that is each company and the government puts their data into STGC layer(s) with the output of each being feed into the governments server, which combines the feature in fully connected layers to make wanted predictions.

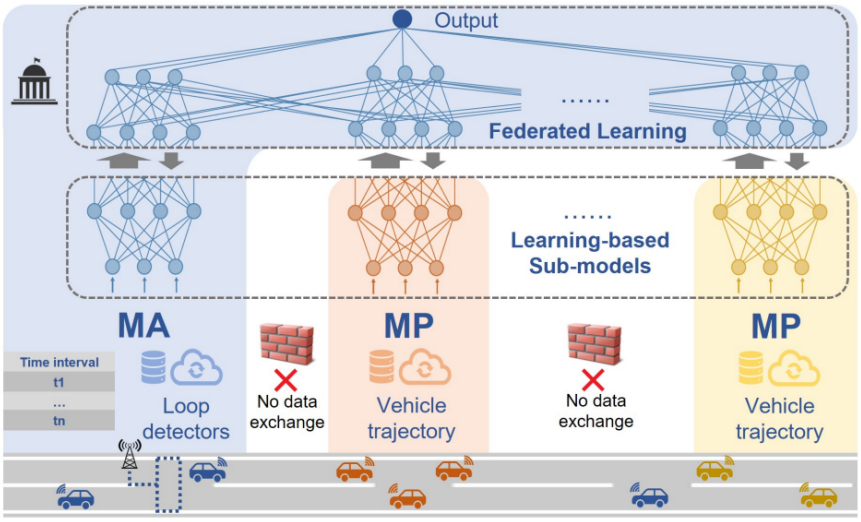

Figure 7: Diagram of architecture, here MA is the government, and MPs are the taxi and ride sharing comapanies.

Some references:

- https://arxiv.org/pdf/2401.11836
- https://arxiv.org/pdf/1709.04875
- https://arxiv.org/pdf/2302.12973