TP2: Semi Supervised Learning (SSL)
Graphs in ML

AL HOUCEINE KILANI

December 17, 2019

Questions

2.1. Explain what are the edge types z_{ij}.

Solution: z_{ij} is a latent representation learned by the network of the interaction between node i and node j. It is a one hot encoded vector (in the paper) but it can be a continuous vector if we want to model some "mixture" of interactions. (although this can be modeled as a whole new interaction type by increasing the space/dimensionnality of $\mathbf{z}$).
We then sample from the distribution z learned (a multinomial one) to start the decoding part.

2.2. In the NRI model, explain how the encoder and the decoder work.

Solution:
The encoder :
We first suppose that they are all related to each others (fully connected graph). We then use node embeddings (atoms), pairwaise in the first pass (first MLP), to infer hidden states (edge embeddings). The second pass (second MLP) aims to integrate the information gathered on the other pair of atoms and thus leading it to use the information gathered on the whole graph. In the end, after going through a softmax layer, we infer binary digits 0 or 1 corresponding to the type of interaction between every two nodes.
The decoder :
The goal of the decoder is to infer the next state using the current state and the learned graph structure. The problem with this assumption (Markovian assumption) is that it does not actually hold. So in order to remediate to that, we use the previous hidden states for the inference step too (hidden states given by a GRU unit added to the edge to vertex decoding mechanism of the GNN).
So in the end the decoder uses the infered edge types that were output by the encoder, the current state and given the previous hidden state (output of GRU). But in addition to this architecture, and to avoid the decoder degenerating and drifting using his own errored predictions, we feed him the true atom states during a given nuber of steps (burn-in phase), and then we begin to infer using previous states.

2.3. Explain the LSTM baseline used for joint trajectory prediction. Why is it important to have a "burn-in" phase?

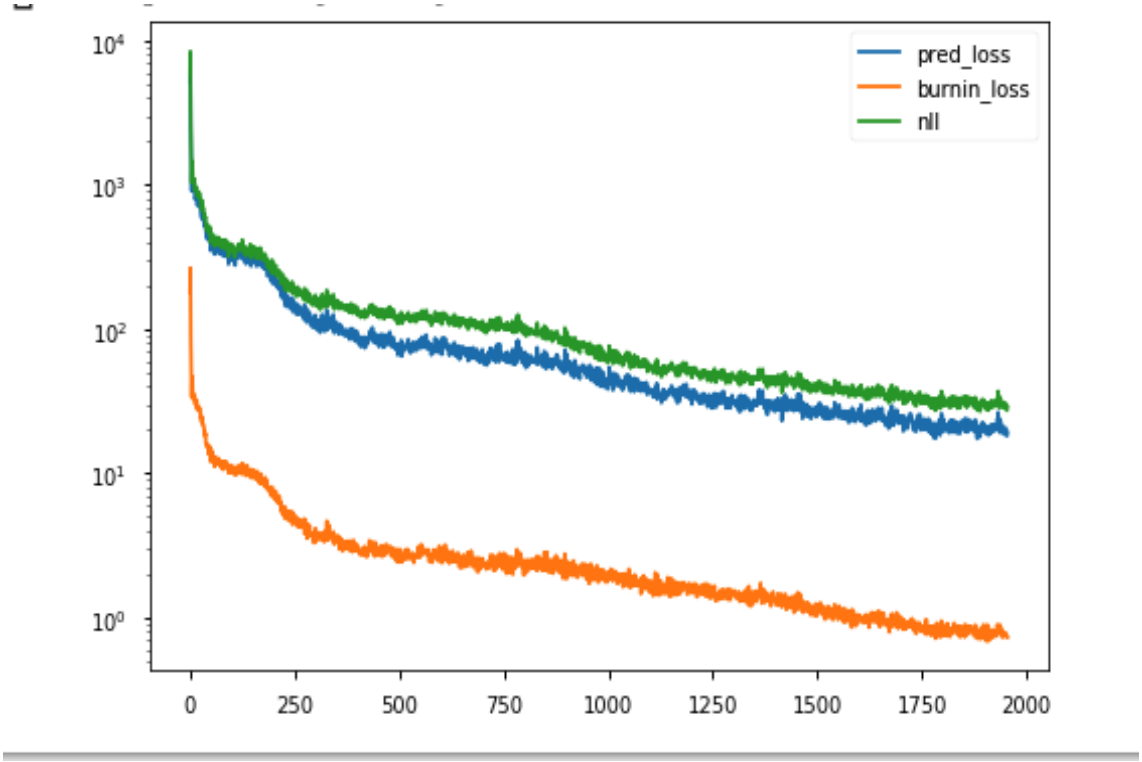

Figure 1: Evolution of the losses

Solution:
This baseline takes the input, encodes it using a 2 layer MLP then goes through and LSTM. This output of the LSTM goes through another 2 layer MLP for a decoding. This decoding operation gives a δ (state vector difference) that is added to the input thus leading to infer the next position and the next velocity of the atom.
The LSTM here is not fed the whole trajectory since the start as we do for the NRI. So if we gave one encoded input to the LSTM and used its output to get the next prediction, it will drift away and it will not manage to capture the dynamics and interaction of atoms and the comparison of performance between LSTM and NRI will not be exactly objective. Thus a burn-in phase is needed (where we feed the LSTM the true next output) in order to force it to "guess" the dynamics during it and computing the loss between the prediction and the real target in this phase. Then comes the phase where we try to guess (like the NRI) the next state using the last prediction.

2.4. Consider the training of the LSTM baseline. Notice that the negative loglikelihood is lower after the burn-in than before. Why is this surprising? Why is this happening?

Solution: Cf Figure 1
Since we use previous state prediction to infer the next state during the prediction step (after the burnin), errors will accumulate and make it drift thus leading to an increase in the loss, something we do not see here in the burnin phase where we feed the correct target to the LSTM.

2.5. Consider the problem of trajectory prediction. What are the advantages of the NRI model with respect to the LSTM baseline?

Solution: In the case of trajectory prediction, The LSTM fails to accurately predict the states in the long term as it lacks this "inference of the dynamics" step that the NRI first does with its encoder.

2.6. Consider the training the of NRI model. What do you notice about the edge accuracy during training? Why is this surprising?

Solution: The edge accuracy increases. But what I find surprising is that it is pretty high since the start (90% in the first epoch) when we would tend to believe that it would be pretty low and start increasing when the epochs will get going. I believe that this is due to the graph dynamics being so simple that is accurately gets it in the first steps and do not struggle infering it.

2.7. What do you expect to happen with the NRI model when there is no interaction between the objects?

Solution: In this case, I would tend to say that the NRI would perform poorly because he is forced to infer dynamics between the objects. And since there are no interactions in reality, it will model some inexisting edges that will lead to bad predictions of the trajectory. One solution to that may be to "hard code" one edge type to represent "non-edge" (as it is said in the last paragraph in the paper). But In this use case, using an LSTM will be more straghforward and logical I think.