Task Planning for Unmanned Vehicle Delivery

Team 24
ZHU HAOYU, WANG GUANDING, ZENG CONG

Our project focuses on the path planning of unmanned vehicles (UVs) for parcels delivery across different communities within an urban environment. We construct an attention based encoder-decoder network to be our policy network.

1. Code explanation

Overall, our network consists of two main components: an encoder (GraphAttention -Encoder) and a decoder (OPDecoder). The top-level module, AttentionNet, integrates these components and outputs the final action probability distribution. The key building blocks are EncoderLayer, GraphAttentionEncoder, Glimpse, OPDecoder, and AttentionNet.

Detail:

- EncoderLayer :
  We first linearly project each node's raw features (x, y, r) into a 128-dimensional space. The projected representations are then processed by a multi-head attention layer followed by a feed-forward layer. Residual connections and batch normalization are applied after both the attention and the feed-forward sublayers, yielding the final embedding for each point.

- GraphAttentionEncoder:
  This module stacks/calls the EncoderLayer to embed all points in sequence, and then aggregates them by taking the mean over the node embeddings to obtain a global graph embedding.

- Glimpse:
  This is a simplified multi-head attention operation in which the context serves as a single query that attends over all feasible nodes. Specifically, we concatenate the embedding of the robot's previous node $h_{(t-1)}$, the global graph embedding h_g, and the remaining distance D_t; the concatenated vector is then projected to 128 dimensions to form the context vector. Using this context as the query, we apply multi-head attention over the embeddings of all valid nodes (masking out invalid nodes), yielding updated context-aligned similarities for all points.

- OPDecoder:
  At each time step, we invoke Glimpse to perform multi-head attention. We then apply a single-head attention over all nodes (no mask at this stage) to obtain the final similarity scores. These scores are clipped via tanh clipping function into the range $[-10,10]$. Afterwards, invalid nodes are set to negative infinity and a softmax is applied to produce the action probability distribution.

- AttentionNet:
  This module packages the encoder and decoder. Inputs include the raw map information, the robot's current position, the remaining path length, and the invalid point mask. The output is the probability of moving to each candidate point under the current state.

2. Existing Libraries

- torch
  Our policy network is primarily implemented with torch.nn. We construct the linear layers, multi-head attention, feed-forward sublayers, batch normalization, and the tanh clipping function using modules from torch.nn.

3. Reflections/Lessons Learned

- Mask invalid action before attention
  We initially applied masking only right before the final softmax to filter out invalid actions. During training, this performed is bad. By comparing the probabilities before and after masking, we found that the agent selected invalid actions with a probability of nearly 40%, and this tendency even increased as training progressed. The policy turn to chose a single high-reward location while ignoring nearby points. To address this, we masked invalid nodes within the multi-head attention itself. We found this works.
  We learned that, for RL problems with invalid actions, masking only at the end may limited does not help the agent to learn which actions are feasible. The resulting gap between the learned (unmasked) policy and the executed (masked) policy prevents effective learning. We need to close this gap by making feasibility awareness part of the network (e.g., feasibility masking inside attention) or by put it into the loss function.