Pytorch Implementation: Influence-aware Memory Architectures for Deep Reinforcement Learning

Reproduction Blog

In this blog we are going to be writing about our experience reproducing the influence-aware memory (IAM) architecture for deep reinforcement learning paper. The goal of this reproduction is to set-up the model and test it with the ware-house example.

Goal of the paper

This paper aims to improve training and convergence difficulties of RNN's with IAM. IAM achieves this by limiting the input of the recurrent layers to variables that influence the hidden state information.

In addition, the model the paper proposes allows information to flow without being stored in the RNN's internal memory, so that not every piece of information used for estimating Q values is fed back into the network for the next prediction.

The model proposed stands out from LSTMs with two key features:

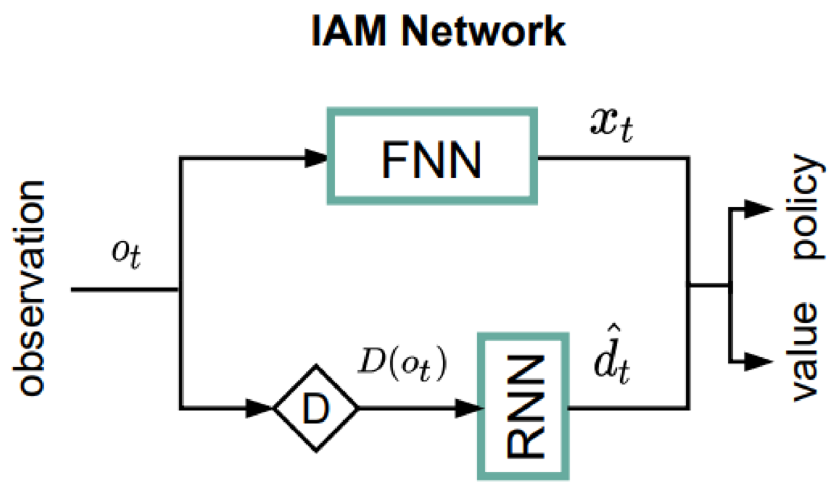
1. The input of the RNN is restricted to a subset of observation variables which in principle should contain sufficient information to estimate the hidden state.
2. There is a feedforward connection parallel to the recurrent layers, through which the information that is important for estimating Q values but that does not need to be memorized can flow.

Related to this is a variant of the memory architecture proposed, which implements a spatial attention mechanism (Xu et al., 2015) to provide the network with a layer of dynamic weights. This type of attention is different from the temporal attention mechanism that is used in seq2seq models, which allows the RNN to condition multiple past internal memories to make predictions. The spatial attention used here filters out a fraction of the information that comes in with the observation.

The memory architecture proposed uses some theoretical insights developed by the framework of influence-based abstraction (IBA).

Influence-aware Memory

Start writing here about the pytorch code implementation of the network.



```
def add code in blocks like these:


def train(model: Pix2PixOptimizer, data_loader: DataLoader,
     no_epochs: int, save_path: os.path, start: int):
 no_images = len(data_loader)
 for i in range(start - 1, no_epochs):
   cumulative_generator_loss = 0.0
   cumulative_discriminator_loss = 0.0
   for j, data in enumerate(data_loader):
     model.set_input(data)
     gloss, dloss = model.optimize()
     gloss = gloss.item()
     dloss = dloss.item()      cumulative_generator_loss +=
gloss
     cumulative_discriminator_loss += dloss
print_percentage(j, no_images)     # Save last image of
epoch
     if j == (no_images - 1):
       save_image(i, model, data)  # Print average losses
  print_avg_losses(i, no_epochs, cumulative_generator_loss,
  cumulative_discriminator_loss, no_images)  # Save network
every ten epochs
  if (i + 1) % 10 == 0:
    save_network(model, i, save_path)
```

Experiments

Warehouse

Traffic Control

Flickering Atari

Results

Did we achieve the original paper results? compare in table here.

Conclusion

Was the original paper implementable in pytorch? Answer here.

References:

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. Show, attend and tell: Neural image caption generation with visual attention. In Proc. of the 32nd International Conference on Machine Learning, pp. 2048–2057, 2015.