Homework 2: Learning on Sequence and Graph Data

Deep Learning (84100343-0)
Autumn 2024
Tsinghua University

Important notes:

- This homework contains two parts: Recurrent Neural Networks (RNN) [**55pts**] and Graph Neural Networks (GNN) [**45pts**], and will account for **15pts** in your final score.
- Please carefully read the guidelines for submission in Sec 3. DO NOT submit your checkpoint files, result files, or data. Your report should be concise and NO MORE THAN 10 pages.

1 Modeling World Dynamics with Recurrent Neural Networks

In sequential environments with complex, high-dimensional observations (e.g., video frames from simulations or games), constructing a predictive model of the world's dynamics allows us to understand and simulate how states evolve over time. Such a world model can be particularly useful in applications that involve prediction, planning, and simulation.

In this assignment, you will develop a world model using Recurrent Neural Networks (RNNs) and, more specifically, Long Short-Term Memory (LSTM) networks. RNNs are especially suited for learning temporal dependencies and handling long-term dependencies by maintaining an internal state across time steps. Here, the world model will use a sequence of observed states and actions to predict subsequent states, capturing the underlying dynamics of a simulated environment. The dataset for this task can be downloaded via: `https://cloud.tsinghua.edu.cn/f/18096a7cee674e76922c/`.

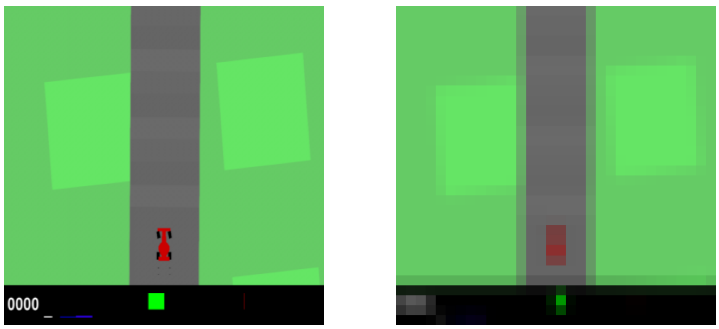
1.1 Problem Setup

In this assignment, you will work with the CarRacing environment[1], a continuous control task where the agent's objective is to navigate a simulated racetrack from a top-down view. The environment provides high-dimensional RGB images as observations, depicting the car's position relative to the track and surroundings. Your goal is to build a model capable of **predicting the next frame** in the sequence based on the **current image** (state) and **action** taken, mimicking the role of a **World Model**. This predictive model will assist in understanding the environment's dynamics by learning to anticipate the outcomes of given actions, ultimately enabling better control strategies for navigation.

Data Collection In the `CarRacing` environment, data collection involves driving a car along a racetrack while recording sequences of observations (images) and actions. Each sequence consists of a series of time steps, where each time step captures the current state as an RGB image, the corresponding action taken, and the resulting next state. Each observation is a 200×200 RGB image that provides a top-down view of the track, capturing track boundaries, road markers, and surrounding terrain. Actions are recorded as continuous 3-dimensional vectors that control the steering, throttle, and brake. This data is stored as a list of sequences, with each sequence containing a series of $(\text{state}, \text{action})$ pairs that represent the car's interaction with the environment across multiple time steps.

Preprocessing Given the high dimensionality of raw RGB images, each state image is first downsampled to a 32×32 resolution, reducing computational overhead while retaining the key features

necessary for navigation. These downsampled images and their corresponding actions are then organized into fixed-length sub-sequences, each containing a specific number of consecutive time steps for model training. This approach ensures a consistent input shape across all data batches, enabling efficient training and inference by the model while preserving the essential spatial and temporal structure of the data.



(a) raw image (b) downsampled image

Figure 1: Illustration of the CarRacing environment's input processing, showing the original high-resolution scene and a downsampled version, used for efficient model input while retaining essential task features for navigation.

1.2 LSTM Background

Vanilla RNNs can be tough to train on long sequences due to gradient vanishing or gradient exploding caused by repeated matrix multiplication. A common variant is the Long-Short Term Memory (LSTM[3]) RNN. LSTMs solve this problem by replacing the simple update rule of the vanilla RNN with a gating mechanism as follows:

Similar to the vanilla RNN, at each timestep we receive an input $x_t \in \mathbb{R}^D$ and the previous hidden state $h_{t-1} \in \mathbb{R}^H$; Moreover, the LSTM also maintains an H-dimensional cell state, so we also receive the previous cell state $c_{t-1} \in \mathbb{R}^H$. The learnable parameters of the LSTM are an input-to-hidden matrix $W_x \in \mathbb{R}^{4H \times D}$, a hidden-to-hidden matrix $W_h \in \mathbb{R}^{4H \times H}$ and a bias vector $b \in \mathbb{R}^{4H}$.

At each timestep we first compute the activation vector $a \in \mathbb{R}^{4H}$ as $a = W_x x_t + W_h h_{t-1} + b$. We then divide this into four vectors $a_i, a_f, a_o, a_g \in \mathbb{R}^H$ where a_i consists of the first H elements of a, a_f is the next H elements of a, etc. We then compute the input gate $g \in \mathbb{R}^H$, forget gate $f \in \mathbb{R}^H$, output gate $o \in \mathbb{R}^H$ and block input $g \in \mathbb{R}^H$ as:

$$i = \sigma(a_i), f = \sigma(a_f), o = \sigma(a_o), g = \tanh(a_g),$$

where σ is the sigmoid function and $\tanh$ is the hyperbolic tangent, both applied element-wisely.

Finally we compute the next cell state c_t and next hidden state h_t as:

$$c_t = f \odot c_{t-1} + i \odot g, h_t = o \odot \tanh(c_t).$$

1.3 Tasks and Scoring

Your objective is to design, train, and evaluate a RNN-based world model capable of predicting future states in a sequential environment. Preprocessed data has been provided, so the focus is on the model design, training, evaluation, and analysis. The assignment is worth 55 points in total, which can be broken down as follows:

1. **Define and train an LSTM-based world model**
   Implement an LSTM-based world model using the built-in class `torch.nn.LSTM` layer in Pytorch. The model should take the current state (preprocessed image) and the action vector as input, and the next predicted state as output. Task relevant metrics include training and validation loss over

epochs. Display a plot tracking training and validation loss over epochs. Note that as the input to the model contains an image, you can use a small CNN network to downsample the image to a lower dimension to facilitate computational efficiency. **[15pts]**

2. **Implement an LSTM layer from scratch**
   Build an LSTM-based world model without using any built-in LSTM layers. Implement the LSTM layer manually, including gate calculations and state updates. Train this custom LSTM model on the provided dataset, and track training and validation loss over epochs. Display the learning curve and compare the performance with the built-in LSTM model. **[15pts]**
3. **Evaluate Model with Two Rollout Strategies**
   For a more in-depth analysis, compare two types of **rollout predictions** on a selected sequence from the test set:
   (a) **Teacher Forcing**: Start with the initial state, and at each step, use the true next state as the input to predict the following state. This approach simulates how the model performs when given accurate state information at each step.
   (b) **Autoregressive Rollout**: Start with the initial state, but at each step, feed the model's own prediction from the previous step as the input to predict the next state. Now that our model does not predict the next action, you can feed the ground truth action to the model at each step. This approach tests the model's ability to predict future states based solely on its own past predictions.

   Display a side-by-side comparison of the ground truth vs predicted frames for each rollout strategy over at least 10 steps, and analyze how the two strategies affect the model's stability and accuracy across multiple time steps. **[15pts]**
4. **Question Answering**
   Suggest at least two potential improvements to help the model perform better in the **Autoregressive Rollout**. These could include changes to the model architecture, training approach, or data preprocessing. Explain why each improvement might address the limitations observed in the Autoregressive Rollout. Note that implementation of these approaches is not required. **[10pts]**

2 Part Two: Graph Neural Networks (GNN)

Graphs are a basic type of data structure. In the real world, objects cannot be fully understood without considering their connections with others. Furthermore, graphs' edges are usually sparse, which makes it challenging to apply dense-input deep networks directly. Graph models in deep learning are specialized to get stronger expressiveness on graph information and have been used in antibacterial discovery, recommendation systems, and physics simulations.

2.1 Background

The QM9 dataset [7] is a popular benchmark in computational chemistry and machine learning, specifically designed for molecular property prediction tasks. It contains a collection of small organic molecules and includes various chemical and physical properties, as depicted in figure 2. The dataset is widely used in fields like materials science, molecular physics, and quantum chemistry to develop and test machine learning models that can predict molecular characteristics. In this task, we will focus on a **graph-level prediction task**, i.e. predicting the **dipole moment**, which is a crucial property in molecular chemistry that measures the separation of charge in a molecule.

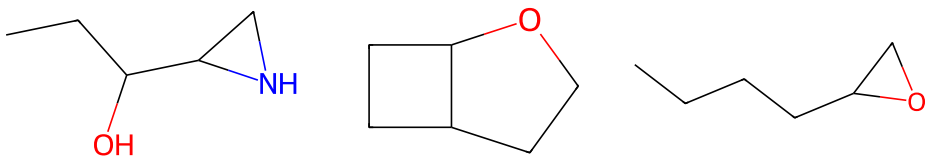


Figure 2: Some sample moleculars in the QM9 dataset.

We will use Pytorch Geometric (PyG) in this assignment. You can install it according to the tutorial. Besides, it's **strongly** recommended to read through PyG's documentation and code.

2.2 Tasks and Scoring

You need to finish the following tasks on the given dataset:

- **Task A**: We provide implementations of **GCN** [4], **GAT** [6] and **GrahpSAGE** [2] for you. Read through and run `gcn.py`, `gat.py`, `graphsage.py`, and report the performance of these methods. Plot and analyze the training and validation loss curves over epochs. Calculate the R^2 score to measure how well the model's predictions align with the true values on the test set. This score ranges from 0 to 1, where values closer to 1 indicate better fit. The R^2 score is calculated as : [**20pts**]

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

  where:

  - y_i is the true value of the target variable for the i^{th} sample,
  - $\hat{y}_i$ is the predicted value for the i^{th} sample,
  - $\bar{y}$ is the mean of all true values y_i.

- **Task B**: Implement **DeepGCN** [5] **or GIN** [8] manually (You **only** need to implement **one** of them to get full grades), and report the performance. (**Hint**: PyG has implemented basic layers for you) [**25pts**]

3 Submit Format

Submit your *code and report* as an Archive (zip or tar). Your code should only contain .py files, not checkpoint files, result files, or data. The report is supposed to cover your **question answering**, the model **technical details**, **experimental results**, and necessary **references**. The length of your report is restricted to 10 pages.

References

[1] D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In *NeurIPS*, 2018.

[2] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, 2017.

[3] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[4] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[5] G. Li, M. Muller, A. Thabet, and B. Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

[6] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. In *ICLR*, 2018.

[7] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

[8] K. Xu, W. Hu, and Leskovec. How powerful are graph neural networks? In *ICLR*, 2019.