Homework 2: Learning on Sequence and Graph Data

Deep Learning (84100343-0)
Autumn 2023
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 Part One: Language modeling with RNN

Language modeling is a central task in NLP and language models can be found at the heart of speech recognition, machine translation, and many other systems. In this homework, you are required to solve a language modeling problem by implementing RNN. A language model is a probability distribution over sequences of words. Given such a sequence $(\mathbf{x}_1, ..., \mathbf{x}_m)$ with length m, it assigns a probability $P(\mathbf{x}_1, ..., \mathbf{x}_m)$ to the whole sequence. In detail, given a vocabulary dictionary of words $(\mathbf{v}_1, ..., \mathbf{v}_m)$ and a sequence of words $(\mathbf{x}_1, ..., \mathbf{x}_t)$, a language model predicts the following word $\mathbf{x}_{t+1}$ by modeling: $P(\mathbf{x}_{t+1} = \mathbf{v}_j | \mathbf{x}_1, ..., \mathbf{x}_t)$ where $\mathbf{v}_j$ is a word in the vocabulary dictionary.

1.1 LSTM Background

Since vanilla RNNs can be tough to train on long sequences due to vanishing and exploding gradients 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 $h_{t-1} \in \mathbb{R}^H$; 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 an 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 elementwise.

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

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

1.2 Dataset

The WikiText dataset is a collection of over 100 million tokens extracted from the set of verified Good and Featured articles on Wikipedia, which is available under the Creative Commons Attribution-ShareAlike License. The WikiText dataset contains a large vocabulary, where WikiText-2 has a vocabulary of 33,278 and WikiText-103 is over 8 times larger. As it is composed of full articles, the dataset is also suited for models that can capture long-term dependencies. We use the small version WiKiText-2 dataset, which contains two splits: training and validation set. The dataset can be found in the `./rnn/data` folder.

1.3 Tasks and Scoring

You need to finish the following tasks on the given dataset and start code:

1. **RNN for Language Modeling**: Construct a kind of RNN with GRU[1] block and train your model from scratch on the *train.txt*. To make this task focus on how RNN works in language modeling, you can use some existing GRU block implementation (e.g. *torch.nn.GRU*). Then validate your model on the *valid.txt*, and report the training and validation curves. [**10pts**]
2. **LSTM Implementation**: Construct a kind of RNN with LSTM[3] block and train your model from scratch. Note that in this task, usage of an existing LSTM implementation **is not allowed** (e.g. *torch.nn.LSTM*). Validate your model, and report the training and validation curves. [**25pts**]
3. **Inference using RNN**: Consider this scenario where you want the RNN language model to complete a sentence. In this case, the language model needs to inference in an auto-regressive manner. You may adjust the ratio of prediction to the given context if you need (e.g., *given 40 words, predict the following 5 words* corresponds to a ratio of 40:5). Please implement the *predict* function and explain your implementation in detail in the report. Also, showcase your auto-regressive predictions on the validation set in your report. [**15pts**]
4. **Question Answering**: In the current setup, our RNN language model produces a word at every timestep as its output. However, an alternate way to pose the problem is to train the network to operate over **characters** (e.g. 'a', 'b', etc.) as opposed to words, so at it every timestep, it receives the previous character as input and tries to predict the next character in the sequence. Can you give one advantage and disadvantage of a language model that uses a character-level RNN? [**5pts**]

Note:

1. Conventionally, we evaluate our language model in terms of **perplexity** (PP) [1]. $\mathrm{PP} = \exp(\text{Average Cross Entropy Loss})$. You need to include this metric in your report.
2. An example for Task 3 is as follows:

   Input = ("The", "player", "progresses", "through", "a", "series", "of")
   Output = ("linear", "missions")

   The RNN model predicts the following 2 words given 7 words. As we want to ensure you understand how auto-regressive inference works, please explain your implementation in detail in the report. You should also present examples where your model predicts very accurately and where your model fails to make correct predictions. Analyzing these phenomena' reasons can give you a deeper understanding of when to use RNNs.

2 Part Two: Graph Neural Networks (GNN) and Node2Vec

Graphs are the basic data structure. In the real world, objects cannot be fully understood without considering their connections with others. 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.

[1] https://en.wikipedia.org/wiki/Perplexity

2.1 Dataset Split in Node Classification

In this part, each data point is an independent sentence. Hence, it is easy to split the dataset (e.g., 80% as the training set and the remaining 20% as the validation set). While in a graph, different nodes are **not independent** because of message passing. In this case, there are two options.

- **Transductive Setting**: the input graph can be observed in all the dataset splits (training, validation, and test set). In other words, we only split the labels.
- **Inductive Setting**: we break the edges between splits to get multiple graphs. These graphs are thus independent.

2.2 Dataset and Library

- We adopt the citation network dataset *Cora* from [8]. In a citation network, nodes represent documents, and edges represent citation links.
- 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.

2.3 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 **Node2Vec**[2] for you. Read through and run `gcn.py`, `gat.py`, `node2vec.py`, and report the performance of these methods. [**20pts**]
- **Task B**: Implement **DeepGCN**[5] **or GIN**[7] (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] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NeurIPS*, 2015.

[2] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, 2016.

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

[8] Z. Yang, W. Cohen, and R. Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.