"Introduction to Deep Learning" Homework II ("practical") version 1.0

Artem Chernodub, Ph.D.
Grammarly
Ukrainian Catholic University, Faculty of Applied Sciences
chernodub@ucu.edu.ua

June 17, 2021

1 Introduction

The goal of Homework II is to learn the internal structure and training pipeline of Recurrent Neural Networks *under the hood*. The outcome is a source code written in Python for PyTorch[1]. You are asked to prepare the code for RNN's forward pass on matrix level "from scratch".

The maximum score for the Homework II is 40% of total score, submission deadline is July 6, 9:00 AM CET. The penalty for missing the deadline: up to one week – minus 50% of scores, more than one week – minus 100% of scores.

2 The problem

You need to implement train and inference for *AlarmworkNet* neural network from the Homework I [Che21]. *AlarmworkNet* (Fig. 2) is a modified version of Simple Recurrent Network [PMB13] (or "Elman Network"). Its design was inspired by *ClockworkRNN* [Kou+14]. Since its hidden layer actually contains several recurrent layers where each layer processes the sequential data at its own clock rate, such a modification makes the architecture competitive with gated neural networks like LSTMs.

[1] Jupyter Notebooks are appreciated but not required.

To test RNN network's capabilities for solving long-term dependencies, we prepared synthetic problem called "Adding task" [PMB13], [HS97]. The data for it is generated artificially as follows (Fig. 1):

channel 1	0.15	-0.21	0.25	....	-0.05	0.1
channel 2	0	1	0	....	1	0

Figure 1: Input sequence $\tilde{\mathbf{x}}$ for adding task. Target value $\mathbf{t} = -0.26$.

- Each sequence $\tilde{\mathbf{x}}$ of length L contains two-dimensional input vectors.
- The values in the first channel are sampled from the uniform distribution in range $[-0.25; 0.25]$.
- All values in the second channel are zeros except of ones at two positions which were sampled randomly sampled from the ranges $[0; L/2]$ and $[L/2 + 1; L]$, respectively.
- Target values $\mathbf{t}$ are sums of the values in the first channel which are marked by ones in the second channel.

The neural network's goal is to predict the target value $\mathbf{t}$ after processing the input sequence $\tilde{\mathbf{x}}$. Prediction is assumed to be successful if $|\mathbf{y} - \mathbf{t}| < 0.1$, where $\mathbf{y}$ is neural network's output. We calculate accuracy for dataset containing N_{total} samples as rate of successful cases:

$$accuracy = \frac{N_{successful}}{N_{total}} * 100\%. \tag{1}$$

The longer the generated sequences are, the longer the dependencies need to be learned and the more difficult the task is, and vice versa.

3 The method

Your task is to implement *AlarmworkNet* from scratch, you need to code forward pass only. The homework package contains PyTorch implementation of the training pipeline for Adding task problem where out-of-the-box *SimpleRNN* is used as a model by default. So, you need to modify *class AlarmworkRNN* in order to make it workable.

There are two source files:

- *generate_data_adding_problem.py* - generates synthetic dataset for training and evaluation for pre-defined sequence length.
- *train_rnn.py* - contains main training loop and slots for code to be implemented;

RNNs are trained for 50 epochs, batch size $N_{batch} = 20$, hidden state size $P = 50$. Training is done by Stochastic Gradient Descent (SGD), learning rate $\alpha = 0.01$, momentum $\mu = 0.9$.

4 Tasks for Homework II

1. Generate the data for sequence lengths $L = \{10, 50, 70, 100\}$ (1% of total score).
2. Add PyTorch out-of-the-box LSTM neural network to the pipeline (4% of total score).
3. Implement forward pass for layers $Layer_{rec1}$, $Layer_{rec2}$ and $Layer_{out}$ in a vector form, add them to *forward* method. You may use PyTorch class *torch.nn.Parameter* to code weights and biases (15% of total score).
4. Implement forward pass for $Layer_{out}$ in a scalar form. Compare it's running speed with model's version where $Layer_{out}$ is in a vector form and report the results of comparison (10% of total score).
5. Implement network's weights' initialization, all weight matrices $\mathbf{W}$ must be filled according to Xavier method [GB10], all biases $\mathbf{b}$ must be filled by zeros. Using Pytorch out-of-the-box initialization functions is not allowed (5% of total score).

6. Report models' performance for sequence lengths $L = \{10, 50, 70, 100\}$. Your code must plot the table with accuracy values for each model ($SimpleRNN$, $LSTM$ and $AlarmworkNet$) and each sequence length (5% of total score).

References

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[GB10] Xavier Glorot and Yoshua Bengio. "Understanding the Difficulty of Training Deep Feedforward Neural Networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[PMB13] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the Difficulty of Training Recurrent Neural Networks". In: *International conference on machine learning*. PMLR. 2013, pp. 1310–1318.

[Kou+14] Jan Koutnik et al. "A Clockwork RNN". In: *International Conference on Machine Learning*. PMLR. 2014, pp. 1863–1871.

[Che21] Artem Chernodub. *Introduction to Deep Learning, Homework I.* 2021, p. 2.

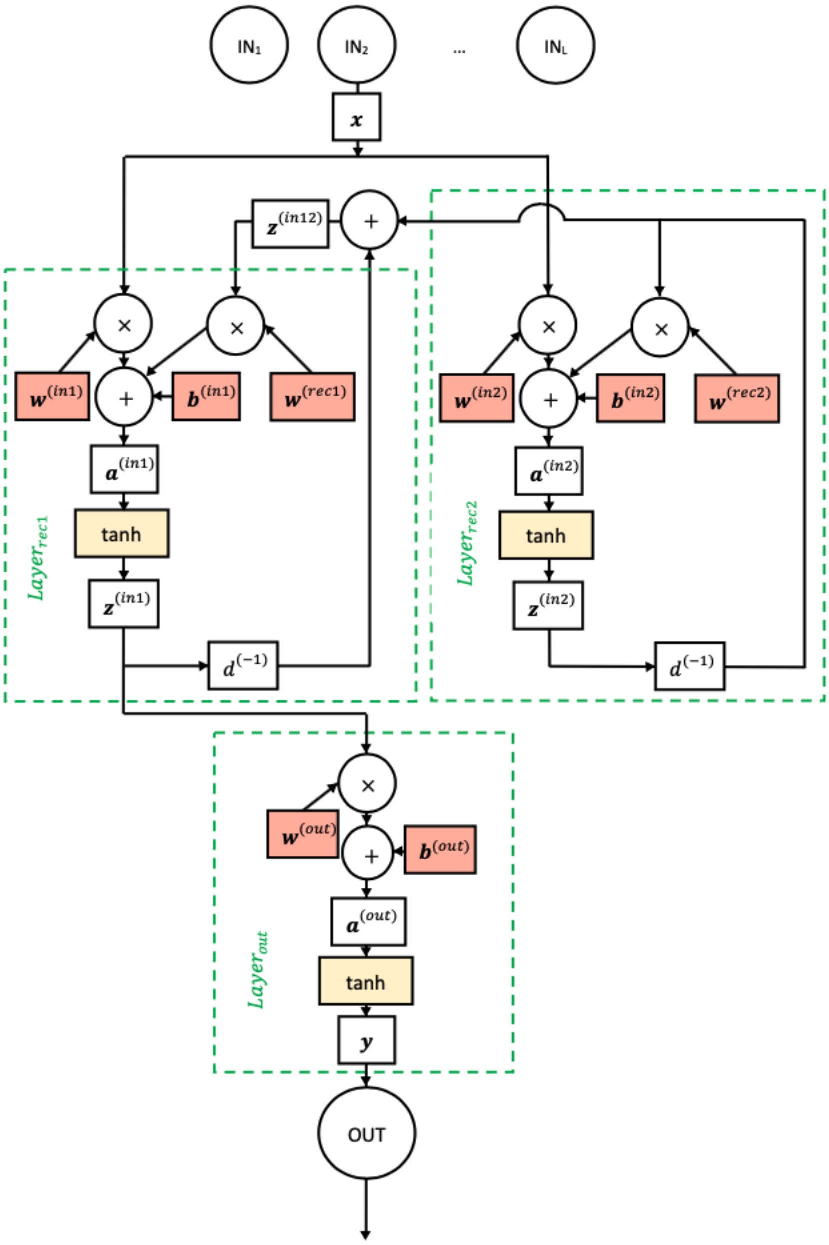

Figure 2: AlarmworkNet's computational graph, time step indices are not shown on the scheme. Layers $Layer_{rec1}$ and $Layer_{rec2}$ works in parallel, but $Layer_{rec2}$ works each second time step only. If $Layer_{rec2}$ is not working on particular time step l, it copies $Layer_{rec2}$'s output values from the previous time step $l-1$. Also, here $d^{(-1)}$ denotes single time step delay, it's the same as $z^{(-1)}$ on some other schemes