אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Sequential Modeling
O. Azencot
Winter 2023

Assignment 1: **The Phase Portrait of Neural Networks**
deadline: December 4 at 5pm

In this home assignment, you will learn about recurrent neural networks, their implementation in python, and their forecasting capabilities. Submission is in pairs. Your work should include two separate files: 1) a zip file named `code.zip` containing ALL the code you used (even if you think it is not an important piece of code), excluding library code such as Numpy/pyTorch. 2) a single pdf file named `hw1.pdf` that contains your report. Your report should be a **detailed** description of your approach to solve each of the tasks, as well as several examples of the results you obtain as detailed below. You should aim for a report with a total of 5 pages. Please submit your work via Moodle. For questions, use the Slack channel, or contact me via email.

1 Background

Neural networks are parametric models which typically try to approximate some high dimensional and complex function f, mapping inputs x to outputs y. For instance, x can be an image of a cat, and y is its label, and the network tries to fit $y = f(x)$. This is done by solving an optimization problem, trying to minimize a distance function $|y - f(x)|$, also called the loss function. In practice, we usually have in the supervised setting a data set of data samples $\{(x^i, y^i)\}_{i=1}^N$ for which we want that $y^i \approx f(x^i)$ for all i. Fitting the model f to the dataset is called training, and it is achieved by gradient descent approaches. After the model finished training, we can use it on the test set which includes samples that are not available during training. In machine learning, algorithms are typically compared with respect to the error measures they obtain on the test set. For a brief tutorial on neural networks, please see the following tutorial, and this code example.

Recurrent neural networks (RNN) are designed to handle time series data $x_{1:T} := x_1, \ldots, x_T$. To encode memory in the system, RNNs use a hidden state h_t, computed from the previous hidden state and the current input. Formally,

$$h_{t+1} = \sigma(Wh_{t-1} + Ux_t) \ ,$$

where W is the hidden-to-hidden matrix, U is the input-to-hidden matrix, and σ is a nonlinear function (e.g., ReLU), see also [1]. For a brief tutorial on recurrent neural network, please see the following tutorial. You can also check this tutorial which gives a forecasting example solved with RNN modules (e.g., LSTM).

2 Data

We will work with simulated data related to the **harmonic oscillator** equation, see [2, Chap. 5]. The governing linear differential equation is given by

$$\frac{\mathrm{d}^2 x}{\mathrm{d}\, t^2} = -\omega^2 x \ , \tag{1}$$

where we take $\omega = 1$. To generate the train and test sets, you will need to numerically solve Eq. (1). A stable scheme to obtain solutions is given by *Verlet method*. Given an initial point x_0, initial velocity v_0, and a time step $0 < \Delta t \ll 1$, we can approximate the solution x_n at time n via

$$\text{set} \quad x_1 = x_0 + v_0 \Delta t - \frac{1}{2} x_0 \Delta t^2 \ , \tag{2}$$

$$\text{iterate} \quad x_{t+1} = 2x_t - x_{t-1} - x_t \Delta t^2 \ . \tag{3}$$

You can take $v_0 = 0$. Using the numerical scheme above, you can construct 500 simulated sequences, each of length 20. That is, generate a collection $\{x_t^i\}$ where $i = 1, \ldots, 500$ is the sample index (based on a specific initial condition x_0^i) and $t = 1, \ldots, 20$ is the time index. The initial conditions x_0^i should be randomly sampled from the unit two-dimensional box $[0, 1] \times [0, 1]$, sampled uniformly. The train set consists of 80% of the generated sequences, and the rest is the test set. Such a split can be achieved via random sampling. You may also read this tutorial for a simple example of solving the pendulum equation using Verlet method in python.

3 Forecasting via RNN

The forecasting problem is defined as follows. Given a time series $x_{1:t} := \{x_1, \ldots, x_t\}$, the goal is to faithfully predict the next item in the sequence using a function f. That is, we require $x_{t+1} \approx f(x_{1:t})$ for any t. We model f using a recurrent neural network, and denote $\tilde{x}_{t+1} = f(x_{1:t})$. The network is trained using the standard mean squared error (MSE) loss for any index t

$$\mathrm{MSE}(\tilde{x}_t, x_t) = |\tilde{x}_t - x_t|_2^2 = \sum_j (\tilde{x}_t[j] - x_t[j])^2 \ , \tag{4}$$

where $x_t[j]$ is the j-th entry in the vector x_t. Using pyTorch, the forecasting problem can be easily solved with the RNN module. You may also read this tutorial for a simple example of forecasting using RNN and pyTorch. You are free to choose the hyper-parameters (e.g., hidden layer size), as long as forecasting results are reasonable.

4 Tasks

1. Generate the harmonic oscillator dataset as described in Sec. 2. Plot the phase portrait of the generated data.

2. Implement an RNN forecasting model, following the descriptions in Sec. 1, 3 for the harmonic oscillator data described in Sec. 2. Your model should predict the next state x_{t+1} from its

previous observations $x_{1:t}$. Compute the prediction error on the train and test sets per epoch, and produce their plot. Specifically, train the RNN model you have implemented for 100 epochs, at the end of each epoch record the prediction error on the train and test sets. Produce a single graph containing both prediction errors as a function of the epoch.

3. Using the trained model, plot the phase portrait of the learned space. To do that, sample densely $10,000$ points in the unit square $[0,1] \times [0,1]$ using the uniform distribution, predict their trajectories using the model, and plot their corresponding trajectories. The horizon of predicted trajectories is 20.

5 Additional Comments

1. You are allowed to use neural models that are built-in in pyTorch/Tensorflow.
2. You should not attach code to the report. Instead, please attach ALL the code you used (even if you think it is not an important piece of code) and the report to a compressed zip package.
3. A significant portion of the grade is dedicated to the neatness and descriptiveness of the report. You should make all the figures and discussions to be as clear as possible. In particular, the axes ticks and labels, as well as the legend, and etc. should be clear without zooming in. Finally, you should describe the architecture you designed and implemented for every task.
4. At the same time, the report should not be too long. Please aim for a 5 page document.

References

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning.* MIT press, 2016.

[2] Steven H Strogatz. *Nonlinear dynamics and chaos: with applications to physics, biology, chemistry, and engineering.* CRC press, 2018.