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

Deep Learning, PDL221
O. Azencot
Winter 2024

Assignment 2: **LSTM Autoencoders**
Due on Jan. 16, 2025 at 5pm

In this home assignment, you will learn about LSTM autoencoders, their implementation in python, and their applications in the context of regression problems. Please include in your submission a **detailed** report that describes how you solve each of the tasks, and several examples of the results you obtain as detailed below. You should aim for a report with a total of 8 pages. In addition, please provide **all** the code you wrote/used in .py files (excluding library code such as Numpy/pyTorch).

1 Background

Autoencoders (AE) are a specific type of neural networks which are used to extract the "important" information from a given input. See Chap. 14 in [1] for a detailed discussion of various autoencoder architectures and approaches. Let x denote an input signal, a typical AE model finds a transformation $x \to_{\chi_{\text{enc}}} z \to_{\chi_{\text{dec}}} \tilde{x}$ such that $\tilde{x} \approx x$, where χ_{enc} and χ_{dec} are the *encoder* and *decoder*, respectively. The fixed-size vector z is usually called the *context* of the input. The general formulation for an AE reads

$$z = \chi_{\text{enc}}(x) \ , \quad \tilde{x} = \chi_{\text{dec}}(z) \ , \quad \text{s.t.} \quad \chi_{\text{dec}} \circ \chi_{\text{enc}} = \text{id} \ , \tag{1}$$

where id is the identity transformation. In practice, χ_{enc} and χ_{dec} can be implemented using neural networks. For example, *LSTM AE* is a particular realization of Eq. (1), where the input is a time series $\{x_t\}$, and the encoder and decoder are LSTM networks.

Let us describe an LSTM AE more thoroughly. Given an input sequence $\{x_t \in \mathbb{R}^m\}_{t=1}^T$, the encoding LSTM network χ_{enc} generates a latent representation of the sequence $z \in \mathbb{R}^k$ (a single vector), which is typically the last hidden state of the recurrent model. Namely, $z := h_T$ where $(c_t, h_t) = \text{LSTM}(c_{t-1}, h_{t-1}, x_t)$, and c_{t-1} is the previous cell state. Then, the output sequence $\{\tilde{x}_t \in \mathbb{R}^m\}_{t=1}^T$ is produced by a different decoding LSTM network χ_{dec} which takes z as input. That is, $\tilde{x}_t = \sigma(U\tilde{h}_t)$ and $\tilde{h}_t = \text{LSTM}(\tilde{c}_{t-1}, \tilde{h}_{t-1}, z)$. Notice that the same z is used as an input throughout the entire sequence. We show in Fig. 1 a schematic illustration of this architecture. To train the network, you can use a loss function that is based on the mean square error $\text{MSE}(x_t, \tilde{x}_t)$.

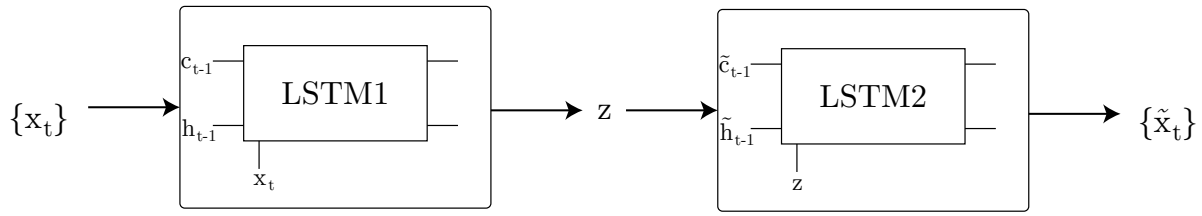

Figure 1: The input sequence $\{x_t\}$ is encoded to a latent representation denoted by z, which is decoded to the output sequence $\{\tilde{x}_t\}$.

2 Data

You will study the behavior of your LSTM AE while experimenting with a few datasets (one synthetic and two real-world). Specifically, we will focus on the MNIST digit database, the S&P500 stock prices, and a random synthetic one-dimensional signal you will create. For all the datasets, you should verify the data segments are normalized, i.e., $x_t \in [0, 1]$ (min-max normalization).

MNIST. MNIST is a well-known dataset that can be downloaded from `http://yann.lecun.com/exdb/mnist/`. Most existing deep learning frameworks have built-in methods for downloading and pre-processing MNIST. For instance, `torchvision` of `pyTorch`, see `https://pytorch.org/vision/stable/datasets.html#mnist`. MNIST images are not a natural time series, however, they can be treated as a time series in the following way. Specifically, when considered as a sequence $\{x_t\}$, each x_t is a different pixel of a given MNIST image.

S&P500 stock prices. We consider the stock prices of S&P500 over the years 2014-2017. The data is available via `http://www.kaggle.com` (you may need to create an account on this website) in the link `https://www.kaggle.com/gauravmehta13/sp-500-stock-prices`, and we will also provide it in the home assignment description. Notice there is a related notebook that shows how to work with this dataset.

Synthetic Data. In addition to the above datasets, you will also create your own synthetic data. To this end, you will randomly generate a total of 10,000 input sequences of length 50 each. Thus, you will have a database of samples $\{x_{j,t} \in [0, 1]\}$ where $j = 1, ..., 10{,}000$, and $t = 1, ..., 50$. We would like the data to be not completely random. To this end, you will post-process each of 10k sequences in the following way. Sample a random integer $i \in [20, 30]$, and multiply the values in $[i - 5, i + 5]$ by 0.1. For instance, say for $j = 1$ you sampled $i = 21$, then you would modify the sequence $\{x_{1,t}\}$ by $\{x_{1,1}, x_{1,2}, \ldots, 0.1 \cdot x_{1,16}, \ldots, 0.1 \cdot x_{1,26}, \ldots, x_{1,50}\}$. Split the data using the usual convention 60%, 20%, 20% for the training, validation, and test data, respectively.

3 Specific Tasks

Implement an LSTM AE model based on the illustration in Fig. 1. Design your code so that the following parameters of the network can be easily changed by the user: input size (e.g., 10), hidden state size (e.g., 64). In addition, each of the scripts you prepare below should support the following training parameters (check out `argparse`): #epochs (e.g., 1000), optimizer (e.g., `Adam`), learning rate (e.g., 1×10^{-3}), gradient clipping (e.g., 1), batch size (e.g., 128), hidden state size (e.g., 64). You can add more parameters, and you can also add application specific parameters.

3.1 Warm-up with the synthetic data

Create a script with the name `lstm_ae_toy.py` that includes the training and evaluation code of your LSTM AE on the synthetic dataset you created in Sec. 2. Please implement the following tasks.

1. Attach to your report two or three examples from the synthetic dataset, showing a **graph** of signal value vs. time for each example.

2. Train the LSTM AE such that it reconstructs the input sequence $\{x_t\}$. Namely, your network $F(\{x_t\})$ should output the sequence $\{\tilde{x}_t\}$ such that $\tilde{x}_t \approx x_t$ for every t. Please perform a grid search of the hyperparameters: hidden state size, learning rate, and gradient clipping (do not train more than 20-30 models). You should choose the range for each hyperparameter. Also, you can choose the other hyperparameters (e.g., optimizer) arbitrarily. Attach to your report two examples of the original **signal** and its **reconstruction** in the same format as described in Task 1. You should make sure that your network is **not** learning the identity function. One way to achieve this is to choose the hidden size of the network to smaller than the sequence length.

3.2 Reconstructing and classifying MNIST images

Create a script with the name `lstm_ae_mnist.py` that includes the training and evaluation code of your LSTM AE on the MNIST dataset. Every image I in MNIST is a data instance of a total of 784 pixels. You will input the image to the autoencoder in a row-by-row format, i.e., $I = \{x_t\}_{t=1}^{28}$ where $x_t \in [0,1]^{28}$ is a specific row. Please implement the following tasks:

1. Train the LSTM AE such that it reconstructs the image, i.e., $I \approx \tilde{I} = F(I)$ where F is the LSTM AE network you implemented. Show a comparison of the output images with respect to the original images. Attach to your report three different examples of a **digit** I, and its **reconstructions** $\tilde{I}$.

2. Modify the network F to allow the classification of images. Namely, in addition to the reconstructed signal $\tilde{I}$, each network should also output a vector of size 10 which consists the probability distribution of the class of the image (similar to what you did in Ex.1). For example, let $\tilde{h}_T$ denote the last hidden state of the decoder LSTM, then we define $\tilde{y} = U\tilde{h}_T \in \mathbb{R}^{10}$ to be the probability distribution, and we penalize the network with the additional loss term $\mathrm{CE}(y, \tilde{y})$ where CE is the cross-entropy loss function. Attach to your report two graphs

containing the **loss** vs. **epochs** and **accuracy** vs. **epochs** for the two architectures (i.e., each graph includes two plots). Your tests should produce an accuracy of at least 98% on the test set.

3. Repeat Task 2 (prediction) when the input to the network is pixel-by-pixel (instead of row-by-row). Compare the results you obtain in both settings. If your computational resources are limited, it is okay to train your network for 5 epochs and report the resulting accuracy.

3.3 Reconstructing and predicting the S&P500 index

Create a script with the name `lstm_ae_snp500.py` that includes the training and evaluation code of your LSTM AE on the S&P500 stock dataset. Make sure the data sequences you work with have the same length. You can omit stocks with missing data to that end. Read carefully the description in `https://www.kaggle.com/gauravmehta13/sp-500-stock-prices`, and the attached notebook. Please implement the following tasks:

1. Attach to your report the graphs of the daily max value for the stocks `AMZN` and `GOOGL`. The x-axis of the graph is the date and the y-axis is the **daily maximum** value of the particular stock.

2. Train the LSTM AE such that it reconstructs the stocks' prices. One approach to train your network is by using cross-validation, i.e., re-training by setting randomly selected train and test sets, and taking the best performing model. Attach to your report examples of three different stocks and their **reconstructed** signal, using the same format as in Task 1. You are allowed split the sequence into n sub-sequences.

3. Modify the network from the previous task to also perform prediction. Let $\{x_t\}$ be the input sequence of a particular stock (e.g., `AMZN`). You will create input pairs in the following way: $(x_t, y_t) = (x_t, x_{t+1})$ for every $t \in [1, T-1]$. Then, in addition to penalizing the reconstruction penalty, e.g., the MSE$(x_t, \tilde{x}_t)$, you will also penalize for the prediction error, e.g., MSE$(\tilde{y}_t, y_t = x_{t+1})$, where $\tilde{y}_t$ is an additional output of the network for every time t. Attach to your reporet two graph of the **training loss vs. time** and **prediction loss vs. time**.

4. Evaluate the model from Task 3 in the context of multi-step prediction. Specifically, multi-step prediction is performed in the following way: given a test sequence $\{x_t\}_{t=1}^{T}$, you will feed the network with the input $\{x_t\}_{t=1}^{T/2}$ (assuming T is even). The network's output provides an approximation of $x_{T/2+1}$. Then, feed the network the input sequence $\{x_t\}_{t=2}^{T/2+1}$ and compute the output $x_{T/2+2}$. Repeat the process until you have all missing values in the sequence. Compare your multi-step prediction results to your previous one-step predictions.

4 Additional Comments

1. To improve your results, you are allowed to dynamically change the learning rate via pyTorch schedulers. Check out `https://pytorch.org/docs/stable/optim.html`.

2. You are allowed to use neural models that are built-in in pyTorch/Tensorflow such as `LSTM` or `LSTMCell`, see e.g., `https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html`.

3. 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.

4. 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.

5. At the same time, the report should not be too long. Please aim for a 8 page document.

6. Regression problems as the ones you solve in this home assignment are different from classification problems. In particular, while in classification problems the accuracy can be measured and easily compared, there is no such unified metric for regression problems. One way to evaluate the performance of your network is to plot the original signal and its reconstruction. A very good result is such that these signals are almost indistinguishable from one another. Similarly, a good result is such that the reconstructed signal is missing some small details of the original signal.

References

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