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

Practical Deep Learning, PDL211
O. Azencot
Spring 2021

Assignment 2: **LSTM Autoencoders**
Due 23:59pm, Thursday, May 20, 2021

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. In addition, please provide all the code you wrote/used.

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 $x \rightarrow_{\chi_{\text{enc}}} z \rightarrow_{\chi_{\text{dec}}} \tilde{x}$ 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

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

where id is the identity transformation. In practice, χ_{enc} and χ_{dec} can be implemented using neural networks. For example, *LSTM AE* are 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 $z \in \mathbb{R}^k$, which is typically the last hidden state of the recurrent model. Namely, $z := h_T$ where $h_t = \text{LSTM}(c_{t-1}, h_{t-1}, x_t)$, 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 $\text{MSE}(x_t, \tilde{x}_t)$ or the negative log likelihood (as used in Assignment 1).

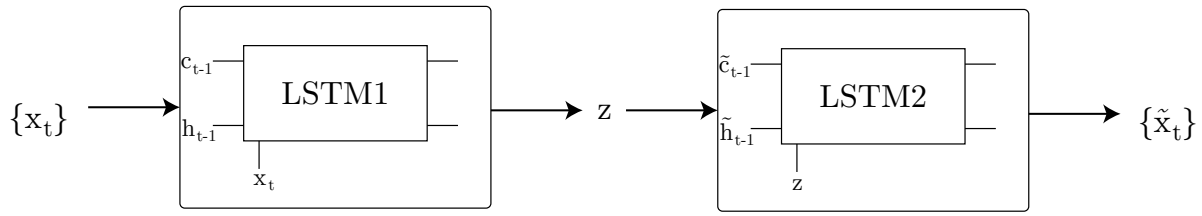

Figure 1: The input sequence $\{x_t\}$ is encoded to a latent representation denoted by z, which is decoded to the output sequence $\{\tilde{x}_t\}$.

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 and centered, i.e., $x_t \in [0, 1]$ and $\frac{1}{T}\sum_t x_t = \frac{1}{2}$.

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`. When treated as a sequence $\{x_t\}$, each x_t is a different pixel of a particular 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 $\{x_{j,t} \in \mathbb{R}\}$ where $j = 1, ..., 10{,}000$, and $t = 1, ..., 50$. 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. 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 the previous task.

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 768 pixels. You will input the image to the autoencoder in a pixel-by-pixel format, i.e., $I = \{x_t\}_{t=1}^{768}$ where $x_t \in [0, 1]$ is a specific pixel. 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. Evaluate the differences in the results when your LSTM components are using fully-connected weights vs. convolutional kernels. Namely, you will have two networks F_{fc} which uses fully connected layers (as we taught in class), and F_{conv} whose transformations are based on convolutions. See `https://github.com/ndrplz/ConvLSTM_pytorch` for an example. Attach to your report three different examples of a **digit** I, and its **reconstructions** I_{fc}, I_{conv} using the fully-connected and convlutional LSTM AEs.

2. Modify the networks F_{fc} and F_{conv} 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 $\text{CE}(y, \tilde{y})$ 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).

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. 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 the previous task.

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 Tips

1. If your computational resources are low, you are allowed to feed MNIST data row by row, instead of pixel by pixel.

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. Please attach all the code you used 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.

5. At the same time, the report should not be too long. $1-2$ pages per task (i.e., Sec. 3.1, 3.2, 3.3) should be just fine.

References

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