Time Series Analysis & Recurrent Neural Networks

lecturer: Daniel Durstewitz
tutors: Manuel Brenner, Max Thurm, Janik Fechtelpeter
SS2022

Exercise 9

To be uploaded before the exercise group on June 29th, 2022

Task 1: Understanding an RNN The aim of this exercise is to capture a simple dynamical system (in this case a sinusoidal oscillation) with a recurrent neural network (RNN). Therefore, define with the help of the python library pytorch a RNN of the following form:

$$z_t = \tanh(U x_{t-1} + V z_{t-1} + b) \tag{I}$$
$$x_t = W z_t + c, \tag{II}$$

where U, V, W are weight matrices and b and c bias vectors. Depict the RNN graphically (in a flow chart).

Task 2: Training the RNN The file *sinus.pt* contains data of 41 time steps from a two-dimensional sinusoidal osciallation:

$$x_t = \begin{pmatrix} \sin\left(\frac{t}{10}\pi\right) \\ \cos\left(\frac{t}{10}\pi\right) \end{pmatrix}, \quad t = 0, \ldots, 40$$

Complete the template for model* and training loop given in *rnn_template.py*†. In the last section of the script, the trained model generates a new trajectory $\{\hat{x}_t\}$, which is then plotted. The new trajectory is 5 times longer than the original one and should give you an impression if the model learned the oscillation correctly. Try to find the minimum number of hidden units in the RNN (dimension of z_t) such that it reconstructs the dynamics to satisfaction.‡

Task 3: Optimizing Training Gradient descent updates the parameters θ with gradient g and learning rate λ: $\theta \leftarrow \theta - \lambda g$. Observe the influence of the learning rate on the dynamics of the learning process:

1. Plot the losses as a function of number of gradient steps and vary the learning rate in gradient descent. How does the loss behave depending on the learning rate?
2. A scheme to speed up learning is to use *momentum* which keeps a moving average over the past gradients: $v \leftarrow \alpha v - \lambda g$, $\theta \leftarrow \theta + v$. How do the dynamics change when the learning rate is adapted with momentum?
3. How does the adaptive learning rate of the Adam (Kingma and Ba, 2014) optimizer perform (`torch.optim.Adam`) in contrast to stochastic gradient descent (SGD)?
4. Can you identify bifurcations in the learning dynamics from eye-balling the loss curve? Plot the generated trajectories before and after a bifurcation to observe how the optimization changes the network dynamics.

*The `torch.nn.RNN` class is your friend

†Hint: the first dimension of the data tensors should always be time, while the last dimension is features.

‡Around 500 epochs should suffice to reconstruct the oscillation.

Task 4: Mini-batching A common strategy to improve training is mini-batching. From the original data set, the model is trained on a randomly drawn subset in every epoch. For time-series $\{x_t\}_{t=1,\dots,T}$, that means drawing a number of sub-sequences $\{x_t\}_{t=s,\dots,s+l-1}$ of length l that start at random time-points s.

Implement mini-batching into your training procedure by implementing epoch-wise drawing of sub-sequences, and concatenating them along a new dimension in the input vector to RNN. In pytorch, the dimensionality of batched data tensors for RNNs conventionally is (Time, Batch, Features). Be careful not to mess this up. The training should now be faster and require less episodes.