Time Series Analysis & Recurrent Neural Networks

lecturer: Daniel Durstewitz
tutors: Georgia Koppe, Leonard Bereska
WS2019/2020

Exercise 8

To be uploaded before the exercise group on December 18th, 2019

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

$$z_t = \tanh(W_{xz}x_{t-1} + W_{zz}z_{t-1} + b_z) \tag{I}$$

$$x_t = W_{zx}z_t + b_x, \tag{II}$$

where W_{xz}, W_{zz} and W_{zx} are dense matrices and b_z and b_x bias vectors. The file *sinus.pt* contains data of 21 time steps from a one-dimensional sinusoidal osciallation ($\{x_t\}_{t=1,...,21}$). Choose a suitable number of hidden units in the RNN (dimension of z_t) to fit the RNN to the data. A template for the training loop in pytorch is given in the file *rnn_template.py*.

Task 1: Learning Dynamics
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 gradient steps and vary the learning rate in the optimizer wrapper for stochastic gradient descent `tc.optim.SGD`. 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 (option of `tc.optim.SGD`)?

3. How does the adaptive learning rate of the Adam (Kingma and Ba, 2014) optimizer perform (`tc.optim.Adam`) in contrast to stochastic gradient descent (SGD)?

Can you identify bifurcations in the learning dynamics from eye-balling the loss curve? Plot the freely running network for each gradient step of the optimization to observe how the optimization changes the network dynamics.

Task 2: Reservoir Computing
In the training loop as given above, the gradients are implicitly backpropagated for all model parameters and through all time steps. An alternative approach is to initialize a network with sufficiently rich dynamics and only train a linear output layer to fit the observations.

1. Initialize the weights W_{xz} and W_{zz} of the network by drawing from a 1. standard normal, 2. uniform (in the interval (0, 1)) distribution or a by a 3. random orthogonal* matrix (`tc.nn.init.normal_`, `tc.nn.init.uniform_` and `tc.nn.init.orthogonal_`) and plot the dynamics *before* any training.

2. Change the to-be-optimized parameters in an optimizer (of your choice) to only contain the output layer weights W_{zx}. (*e.g.* `tc.optim.SGD(model.output_layer.parameters())`). Can you recover the oscillation? Which initialization works best?

*probability distribution given by the respective invariant measure https://arxiv.org/pdf/math-ph/0609050.pdf