Time Series Analysis & Recurrent Neural Networks

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

Exercise 10

To be uploaded before the exercise group on July 6, 2022

Task 1: Backpropagation Through Time Spoiler Given $T \in \mathbb{N}$, a univariate RNN

$$z_t = \phi(u z_{t-1} + v x_{t-1}), \quad y_t = w z_t$$

with a differentiable function ϕ, where $\forall t \in \{0, \ldots, T\} : u, v, w, x_t, y_t, z_t \in \mathbb{R}$, and a squared error loss

$$\mathcal{L}(u, v, w) = \sum_{t=1}^{T} (x_t - y_t)^2,$$

analytically find formulas for the gradients

$$\nabla_u \mathcal{L}, \nabla_v \mathcal{L}, \nabla_w \mathcal{L}.$$

Find conditions for the gradients such that they neither vanish nor explode, that is

$$\lim_{T \to \infty} \nabla_u \mathcal{L} \notin \{0, \pm\infty\}.$$

Task 2: The influence of noise on the data The file *sinus.pt* contains data of 41 time steps from a two-dimensional sinusoidal osciallation with Gaussian white noise:

$$x_t = \begin{pmatrix} \sin\left(\frac{t}{10}\pi\right) + \varepsilon(t) \\ \cos\left(\frac{t}{10}\pi\right) + \varepsilon(t) \end{pmatrix}, \quad t = 0, \ldots, 40, \quad \varepsilon(t) \sim \mathcal{N}(0, 0.2)$$

Using too few hidden units can make the model unable to learn the dynamics, whereas too many can lead to overfitting (it reproduces the noise, not the underlying oscillation). Because it's often hard to find the number of parameters that's just right, one usually starts with a big number and employs some tricks. Using your RNN from last week on the noisy data, [*], find a number of hidden units where the RNN clearly overfits the data [†]. Compare the performance of the following strategies to mitigate overfitting:

a) L1-Regularization: add $\alpha(\sum_j |\theta_j|)$ to the loss function, where θ_j are all the parameters of the model, and try out different values for α, starting at 0.1. To access θ_j, use `model.parameters()`.

b) L2-Regularization: add $\alpha(\sum_j (\theta_j)^2)$ to the loss function and try out different values for α, starting at 0.1. Equivalently, you can set the "weight decay"-argument of the Adam optimizer to α.

c) Dropout: set the "dropout" argument to the `torch.nn.RNN` class to $d \in (0, 1)$. This will set a fraction of d of the total weights to zero in every gradient step. *Important*: before training, set `model.train()`, and afterwards, `model.eval()`. This will set dropout on and off, you don't want dropout when evaluating the model!

Find arguments (briefly) why each of the techniques counteract overfitting.

[*] Use the Adam optimizer and mini-batching

[†] Here, you can deduce that from the prediction plot.