Projects Proposals

Self-Organization: Theory and Simulations

Prof. C. Gros
Winter term 2022/23

December 12, 2022

1 Formalities **2**
1.1 Selection / Assignment 2

2 Projects **3**
2.1 Stochastic Predator-Prey Models on Regular Lattices 3
2.2 Swarm Movement 3
2.3 Reinforcement learning with SARSA 4
2.4 Deep learning with MNIST (advanced) 4
2.5 Global Cascades on Random Networks 5
2.6 Epidemic Modeling: the SEIRS Model 5
2.7 Self-Organized Criticality: The Sandpile Model 6
2.8 Car Following Models - Webots Simulation 7

1 Formalities

Projects should be performed by groups of maximally three people.

- The date to hand-in is: **Mon, Jan 30, 2023.**
- Projects are handed in via mail to Oren Neumann in the form of a zip/tar file containing the web presentation. Consult your tutor beforehand to make sure everything works. The final version will be published as a webpage online.
- There is no need to hand in an additional pdf summary.
- All groups present their results at the end of the term. If there is time, selected groups may present their results during the lecture hours, otherwise during the tutorials.
- After the presentation, improvement suggestions can be made. Only projects of presentable quality will be uploaded to the internet. Final date for handing in revised project is **Mon, March 5, 2023**.
- Course requirements are fulfilled only once the project is published on the web.

You are welcome to consult with the person responsible for the project whenever you need help:

Claudius Gros gros@itp.uni-frankfurt.de

Oren Neumann neumann@itp.uni-frankfurt.de

Daniel Nevermann nevermann@itp.uni-frankfurt.de

Elias Fischer fischer@itp.uni-frankfurt.de

1.1 Selection / Assignment

The aim is that groups select different projects. Please send a preference list (a), (b), etc for projects to Oren Neumann. The final assignment of projects to groups will be made in the class on **Tue, Dec 20, 2022**. It is desirable that at least one member of each group is present.

2 Projects

You are welcome to propose your own project. This has to be down however well in advance.

- Contact the tutors, discuss content and workload.
- In case, submit an outline following the style of the projects below.

Your proposal will be included in this list (and reserved), if accepted.

2.1 Stochastic Predator-Prey Models on Regular Lattices

(Claudius Gros)

Predator-prey systems such as the Lotka-Volterra equations can be modeled (1) as contact processes of the form

$$\begin{aligned} A &\xrightarrow{\mu} \emptyset, && \text{predator death} \\ B &\xrightarrow{\sigma} B+B, && \text{prey reproduction (birth)} \\ A+B &\xrightarrow{\lambda} A+A, && \text{predation} \end{aligned}$$

where A/B are predator/prey densities and $\mu, \sigma, \lambda > 0$ the corresponding reaction rates

- Perform the mean-field analysis which leads to the classical Lotka-Volterra system in terms of a set of ordinary differential equations. Consider the case of resource limitation.
- Implement different variants of a predator-prey system on a cellular grid with next-neighbor interactions. Variations of the model may concern the rule of reproduction/spreading or the rules of interaction between both populations (2).
- Generate an online application.

(1) Stochastic population dynamics in spatially extended predator-prey systems

(2) Oscillations and chaos behind predator-prey invasion : mathematical artifact or ecological reality?

2.2 Swarm Movement

(Oren Neumann)

Swarming is observed in nature when groups of animals flock together, often creating complex patterns such as bird flock murmurations (1). this behavior is an emergent phenomenon, arising from the actions of individuals who can only accurately measure the movement of their close neighbors. You are required to reproduce and analyze the behavior of two kinds of swarms, governed by two different equations.

- Insect swarm: each individual tries to minimize their distance to the center of the swarm, where safety is maximized, but also maximize the distance to their neighbors in order to avoid collision.

$$\boldsymbol{v}_n(t+1) = \boldsymbol{v}_n(t) + \frac{1}{N}\sum_{m\neq n}\left(a\frac{\boldsymbol{r}_m - \boldsymbol{r}_n}{|\boldsymbol{r}_m - \boldsymbol{r}_n|} - b\frac{\boldsymbol{r}_m - \boldsymbol{r}_n}{|\boldsymbol{r}_m - \boldsymbol{r}_n|^2}\right) \tag{1}$$

- Bird flocks and fish schools: using the same rules governing insect movement, birds and fish produce more complex movement patterns by adding a velocity matching term, minimizing the difference between their velocity vector and that of their neighbors.

$$\boldsymbol{v}_n(t+1) = \boldsymbol{v}_n(t) + \frac{1}{N}\sum_{m\neq n}\left(a\frac{\boldsymbol{r}_m - \boldsymbol{r}_n}{|\boldsymbol{r}_m - \boldsymbol{r}_n|} - b\frac{\boldsymbol{r}_m - \boldsymbol{r}_n}{|\boldsymbol{r}_m - \boldsymbol{r}_n|^2} + c\frac{\boldsymbol{v}_m - \boldsymbol{v}_n}{|\boldsymbol{r}_m - \boldsymbol{r}_n|^2}\right) \tag{2}$$

Implement a visual simulation of these two movement patterns with particles moving on a 2D torus (periodic boundary conditions. How do you calculate distance on a torus?). Enforce a maximal velocity limit to avoid numerical errors. Add knobs controlling the strength of movement parameters: center attraction, neighbor repulsion and velocity matching. How does the pattern change with these quantities?

(1) Patterns of Flocks of Starlings Mathematically Behave How Magnets Would Move

2.3 Reinforcement learning with SARSA

(Oren Neumann)

Q-learning is a group of machine learning algorithms where an agent learns to solve a problem by predicting the quality Q of available actions (1). One of the basic Q-learning algorithms is SARSA (2), standing for State-Action-Reward-State-Action. This algorithm builds a function Q that estimates the quality of an action a given the current state s, by interacting with the environment and refining its estimation according to the rewards r it receives.

Create a gridworld environment, where an agent needs to walk from a start cell to a goal cell by hopping each turn to one of 4 neighboring cell. Let the agent run through the gridworld many times, updating the Q value of each state-action pair according to the Bellman equation:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right] \tag{3}$$

where α is the called the learning rate and γ is a number approaching 1 from below. This equation uses the available estimation of the Q values of future states to approximate the discounted sum of future rewards, $Q \approx \sum_t r_t \gamma^t$. The rewards r should be determined by you for each environment. To ensure all state-action pairs are visited several times, make the agent moves according to an ϵ-greedy policy, where moves always maximize Q but sometime are made randomly with probability ϵ.

Implement the following environment:

- Vanilla: simple environment where the agent receives a reward of -1 at each time-step and a reward of $+100$ when reaching the goal, terminating the episode.
- Cliff: remove the bottom half of the map, such that moving into it results in a reward of -100 and episode termination.
- Windy: Add wind to the middle are between the start and goal positions, carrying the agent upwards at each time-step (so moving left would be moving up and left). Change the wind force for each column. Next, add a cliff drop in the direction of the wind.
- **Optional:** Create your own idea for an environment.

Run many episodes for each environment until Q values converge. How do the parameters α, γ affect convergence?

Create a static webpage to present your results, with either an interactive interface or a recording of the agent playing.

(1) An introduction to Q-Learning: Reinforcement Learning

(2) Sarsa: On-Policy TD Control

2.4 Deep learning with MNIST (advanced)

(Oren Neumann)

Deep learning is a machine learning subfield where neural networks are trained to fit datasets. The classic training dataset is called MNIST (1), featuring a collection of handwritten digits. Follow the instructions in (2) and set up a training pipeline, training a convolution neural net on a subset of the data and testing its performance on the rest of the data. To do that you will need to read about and explain how convolution neural nets work.

Change the parameters of the network (width, depth) and training hyperparameters (learning rate, dropout) and see how they affect training speed and quality. You should explain what each of those parameters does, and what is the function of each neural network layer. Optional: add skip connections to the network and see if they help it learn faster when the network is deep.

Create a webpage with a summary of your results, including short introductions to the concepts used and training plots.

(1) MNIST dataset

(2) Simple MNIST convnet

2.5 Global Cascades on Random Networks

(Daniel Nevermann)

How can a small, at first seemingly harmless disturbance in a large connected network trigger a complete system failure?

Networks are everywhere around us: financial networks, epidemiological networks or infrastructure networks are just a few examples. Participants of such networks always rely on their neighbors in one way or another and therefore, might also suffer if one of their neighbors is troubled. This can lead to a large cascade of failures that affects the network as a whole.[1] While the stability of an individual in the network may be easy to assess, cascades of failures can be extremely difficult to predict. Particularly dangerous are scenarios, where a system appears stable for a long period of time and withstands many external shocks and then suddenly exhibits a large scale cascade. This is typically referred to as the *robust-yet-fragile* nature of a system.

The goal of this project is to develop an intuition for system-wide cascades of failures from a network theoretical point of view. For that, consider a model of global cascades on random networks introduced by Duncan Watts in 2002 (1). This model explains cascades on undirected sparse random networks in terms of a very basic measure: the connectivity (or coordination number) of the network

$$z = \frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \deg(v),$$

where $\mathcal{V}$ is the set of nodes in the network and deg denotes the degree of a node. Each participant in the network is either *on* or *off*. To begin with, all nodes in the network are off. Then, at time $t = 0$, a small subset of nodes is turned on externally. At the following time-steps, a node switches it's state to on if and only if a certain fraction of its neighbors is on (*threshold rule*).

The following steps should guide you through the project:

- Read up on the model proposed in (1). Try to understand its dynamics and main characteristics. Familiarize yourself with the basic network theoretical concepts required for the formulation of the model (you can refer to (2) for that).
- Summarize the model and its main characteristics. Explain the robust-yet-fragile tendency. How is the stability of a random network related to its average degree z?
- Build a simulation, which visualizes how a cascade evolves in a randomly generated network with coordination number z according to the model in (1). Let the coordination number be a variable parameter and thus show, how it relates to network stability.
- **Optional:** If you are interested to go further, you may discuss the analytical solution of the model or investigate the effect of a scale-free degree-distribution.

(1) A simple model of global cascades on random networks

(2) Complex and Adaptive Dynamical Systems: A Primer

2.6 Epidemic Modeling: the SEIRS Model

(Daniel Nevermann)

The basic SIRS model has been discussed in the lecture. This model specifies three groups in a population of size N: susceptible (S), infected (I), and recovered (R), where normalization demands $S + I + R = 1$ at all times. The dynamics of this model are dictated by

$$\dot{S} = -\beta SI + \omega R, \quad \dot{I} = \beta SI - \gamma I, \quad \dot{R} = \gamma I - \omega R,$$

where β, γ and ω denote the infection rate, recovery rate and the rate at which immunity is lost, respectively. Furthermore, we neglect births and deaths and thus assume a fixed population size (non-vital model). A fundamental assumption in the SIRS model is that an infected individual is itself infectious instantaneously. For most infectious diseases, however, there is a latency period before an infected individual is transmissive (a notable example is COVID-19).

[1] For example, in 1996 a cascade of failures caused a blackout in the western US.

The goal of this project is to study an extention to the SIRS model, the SEIRS model, that includes a fourth group: the exposed (E). An infected individual will first be in the exposed group for a certain incubation time, before moving to the infected group. The dynamics are thus governed by the following equations

$$\dot{S} = -\beta SI + \omega R, \quad \dot{E} = \beta SI - \sigma E, \quad \dot{I} = \sigma E - \gamma I, \quad \dot{R} = \gamma I - \omega R,$$

where σ denotes the incubation rate.

The goal of this project is to compare the SIRS model to the SEIRS model and study both models in detail. The following steps should guide you through the project:

- Understand the SIRS and SEIRS model. You may use (1) and (2) for that, but are encouraged to find further resources on your own.
- Explain both models and their main characteristics in detail.
- Build a simulation that compares the solutions to these two models. Make sure that all significant parameters can be modified by the user. Using your simulation, point out the differences between the two models. Consider vital models (variable population size) as well.
- **Optional:** Build a second simulation, that visualizes the spread of an epidemic according to the SIRS model. You may use this as inspiration.

(1) Modeling infectious epidemics

(2) The SEIRS model for infectious disease dynamics

2.7 Self-Organized Criticality: The Sandpile Model

(Elias Fischer)

The term *self-organized criticality* was cultivated by Per Bak, Chao Tang and Kurt Wiesenfeld in 1987. In the lecture various systems were discussed that show criticality if some system parameter is set to a critical value. A system shows self-organized criticality if its dynamics lead to into a critical state 'on its own' starting from various different initial conditions. As an illustration, Bak, Tang and Wiesenfeld introduced a simple cellular automaton - the sandpile model.

The sandpile model is defined on a finite $N \times N$ grid of cells. A value

$$z(i,j) \in \{0,1,2,3\} \quad i,j = 1,\dots N$$

is associated to each cell, indicating the number of grains of sand on that cell. New grains of sand are dropped onto the grid which fills the cells. If the size of a sandpile exceeds a threshold value $z(i,j) > 3$ it topples and spreads sand to its four direct neighboring grid cells. A neighboring cell of a toppling cell close to its threshold value thus might also topple as a consequence. The result of this process can be an avalanche of sand running over the grid which stops once a new stable configuration is reached. We use open boundary conditions: A toppling cell at the boundary of the grid will cause a grain of sand to drop out of the system. In the sandpile model, the configuration will relax into a steady state in the long term limit.

The following steps should guide you through the project:

- Understand the sandpile model and its main characteristics. Originally the model was proposed in (1), but you may also refer to Chap. 6.4 in (2).
- Summarize the model and its main characteristics. Explain self-organized criticality. What happens to the density of sand in the system?
- Build a simulation, which visualizes the sandpile model. Treat three different ways how new grains of sand are added to the system: sand may only be dropped in the center of the grid, to a randomly selected cell in the grid or to cells the user can select by clicking on the grid.
- **Optional:** You may also implement your own ideas how sand is added to the system.

(1) P. Bak, C. Tang and K. Wiesenfeld: Self-organized criticality

(2) Complex and Adaptive Dynamical Systems: A Primer

2.8 Car Following Models - Webots Simulation

(Drazen Hukic)

The simplest model describing a chain of cars is given by

$$\ddot{x}_{j+1}(t+T) = \lambda \left[\dot{x}_j(t) - \dot{x}_{j+1}(t)\right], \tag{4}$$

where $x_j(t)$ is the position of the j-th car at time t, T is the reaction time, and λ is a reaction constant. It describes a behavior where a driver slows down if the distance to the car in front becomes smaller or speeds up if the distance gets bigger. There are more realistic models where the reaction also depends on the distance, so that the closer two cars are, the stronger the reaction.

Webots is a free open-source 3D robot simulator. The program allows the user to build a world with programmable robots in it. The first step is to build a circular track and simple "car-like" robots. The robots then can drive in a row along the track in a circle. The behavior of the robots is governed by the equations of the car following models. The following steps can be used as guidelines:

- Implement the simple model describing a chain of cars and vary the number of cars, as well as the parameters T and λ. How does the flow of the cars change? Are there congestions, or crashes?
- Discuss the concept of carrying capacity of the road, in dependence of the parameters T and λ.
- Let the car in front change its speed and observe the reaction of the chain.
- Implement a more realistic model and observe the traffic in dependence of the given parameters.

(1) Complex and Adaptive Dynamical Systems: A Primer

(2) Car-following: a historical review