EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Computer Science Department
Neuro-Cognitive Modeling

**Winter Term 2024/2025**

# Recurrent and Generative Neural Networks

**Exercise Sheet 06**

Release: January 20, 2025 Deadline: February 3, 2025 (11:59 am)

**General remarks:**

- Download the file `exercisesheet06.zip` from the lecture site (ILIAS). This archive contains files (Python scripts), which are required for the exercises.
- Ideally, the exercises should be completed in teams of two students. Larger teams are not allowed.
- **Add a brief documentation (pdf) to your submission**. The documentation should contain your **names**, **matriculation numbers**, and **email adresses** as well as protocols of your experiments, parameter choices, and a discussion of the results.
- Set up a Conda environment using the provided `environment.yml` file (included in the `exercisesheet06.zip` archive).
- When questions arise start a forum discussion in ILIAS or contact us via email:
  Fedor Scholz (`fedor.scholz@uni-tuebingen.de`)
  Manuel Traub (`manuel.traub@uni-tuebingen.de`)

## Exercise 1: Model Predictive Control [50 points]

In this exercise, you will implement Model Predictive Control (MPC) using the Cross-Entropy Method (CEM) [1, 2] to safely control and land a spaceship in a simple simulation.

### (a) CEM Implementation [30 points]

In this task, you will apply a random planner to the `SimpleControlGym` environment in `simple_control_env.py` and then replace it with your CEM-based planner.

**Setup and Overview** Run the provided random planner by executing:

`python run.py --render`

This will control the agent in the environment and render its behavior. The environment, `SimpleControlGym`, includes a ground truth transition model, `SimpleControlModel` (in `models.py`), derived directly from the environment. Using this ground truth ensures that poor performance is due to issues in your CEM implementation, not in the transition model. When implemented correctly, the agent should head towards the orange goal area. It may move slightly within the goal after reaching it, which is expected.

**Task** Implement the Cross-Entropy Method (CEM) in `planners.py`. Your CEM class should inherit from the `Planner` base class. Ensure your implementation uses a single loop to calculate and update samples. Use the provided hyperparameters in `run.py`. Apply the `policy_handler` function to each sampled action to ensure actions stay within the valid range defined by the environment. Add the option `--cem` to the command-line interface to enable your CEM implementation instead of the `RandomPlanner`.

## (b) Transition Model Implementation [10 points]

After verifying your CEM implementation, you will now apply it to the LunarLander environment.

**Setup and Overview** Familiarize yourself with the `LunarLander` environment by consulting its documentation [1]. Pay special attention to the structure of continuous actions and observations. Note: In this exercise, the leg contact information is omitted, as it cannot be predicted solely from position and velocity.

Customize the behavior of `run.py` with the following arguments:

- `--lunarlander`: Switch to the `LunarLander` environment.
- `--render`: Enable rendering (omit this for faster training).
- `--record`: Save environment images to your hard disk for later inspection.
- `--seed n`: Set the random seed to `n` for reproducibility.

Saved models are stored in the directory: `models/{model_id}/{epoch}.pth` Adjust the `state_path` variable in the code to load a specific checkpoint for control.

**Task** Implement a neural network-based transition model called `ComplexControlModel` in `models.py`. Define and standardize the inputs. Ensure the values are approximately standard normally distributed and explain your approach. Specify and format the outputs required to predict the next state. Explain why a recurrent neural network (RNN) is unnecessary for this task.

## (c) Transition Model Training and MPC-CEM [10 points]

In this task, you will train your transition model for the LunarLander environment and apply your CEM implementation to control the rocket.

[1] https://gymnasium.farama.org/environments/box2d/lunar_lander/

**Training the Transition Model** Train your `ComplexControlModel` by running:

```
python run.py --train
```

Select and configure an appropriate optimizer (e.g., Adam or SGD). Experiment with suitable hyperparameters such as learning rate, batch size, and number of epochs.

We use the observations from the environment as labels (targets) during training. You may implement a dataset and dataloader to train on batches, though this is optional. We utilize the heuristic provided by the environment to generate valid actions during training.

**Applying MPC-CEM** Use your trained transition model with CEM in the `LunarLander` environment:

```
python run.py --lunarlander --cem
```

Aim to land the rocket reliably between the flags. A perfect success rate is not required! If necessary, adjust the criterion or hyperparameters of your CEM planner to improve performance.

# Exercise 2: Loci [50 points]

The slot-based location and identity tracking system Loci was published in [3]. Its architecture consists of an encoder, temporal prediction, and decoder pipeline with unique modular structure. The program that you will be running is based on Loci-Looped (Loci-L), which contains a novel internal processing loop, which is able to compute adaptive information fusion. It is based on this paper currently available on ArXiv: Traub et al., Looping LOCI: Developing Object Permanence from Videos. In this bonus exercise you will run Loci and explore its abilities in a simple dataset using a pre-trained model.

## Instructions to get the code running

Download the BOUNCINGBALLS datasets from `https://nc.mlcloud.uni-tuebingen.de/index.php/s/nfoe5Q9RPXx4Btd` and place them into `data/data/video/BOUNCINGBALLS`. A pretrained Loci-Looped models can be found in `out/pretrained1/bouncingballs/net_final.pt`. Run the `eval.sh` script to evaluate the pretrained models with `sh eval.sh`. At the very beginning of the script `scripts/evaluation_bb.py` a section SETTINGS is included. Here you can control some of the model parameters and run evaluations with different settings. The program will start evaluating the BouncingBalls dataset then in the default setting.

## Output generated by the program

The results that are generated by the program contain performance videos as well as some performance statistics. All of this can be found in the sub-folders of
`out/pretrained1/bouncingballs/results1`.
The `-n` flag enables you to generate also other folders, e.g., `-n 2` generates the results in
`out/pretrained1/bouncingballs/results2`.

There are four sub-folders in `net_final/test`.

Folder `statistics` provides txt files about the performance statistics. There are various ones - the most simple and well-known is found in the first line: the mean squared error in pixel space (mse). Furthermore, there are evaluations on the object level for visual similarity (LPIPS, PSNR, SSIM) and mask accuracy (ARI) over time. All these measure report the performance between test step 20 and 30; 80 steps are run in total.

The three folders `open`, `vidpred_auto`, `vidpred_black` contain videos generated during the evaluations.

The differences between the three settings are:

- open: the ground truth image from the simulator is continuously provided as input.
- vidpred_auto: from time step 20 onward, Loci's own output is fed-back as input.
- vidpred_black: from time step 20 onward, Loci receives a black screen as input.

The evaluations are done with a pre-trained Loci-L network. The training setup can be found in `cfg/bouncingballs/bb-level1-run1.json`. In short for the bouncingballs dataset: Training proceeded for 1000 epochs. The inner loop started to be trained after 40k updates of pure outer-loop training. Moreover, also after 40k updates, simulated blackouts are induced. When a blackout occurs, the input frame from the simulator as well as the respective prediction error are set to zero, while the learning target image still stays as before. The blackout rate is initialized to 10% and then continuously increased to 45%. Moreover, the backpropagation through time steps progressively increase from 1 to 3. The 400k update training took about 20 hours.

## (a) Evaluate the default settings [20 Points]

Contrast the performance for the three evaluation cases (open vs. auto vs. black). Describe which differences you can notice in the videos and evaluate the performance measure (report a few key values also).

## (b) Run Loci-L with just 2 teacher forcing steps [10 Points]

Report the effect on performance including potential behavior changes observable in the videos.

## (c) Run Loci-L with eight slots [10 Points]

Report the effect on performance including potential behavior changes observable in the videos.

## (d) Run Loci-L with the inner loop disabled [10 Points]

Report the effect on performance including potential behavior changes observable in the videos.

# Literatur


[1] R. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and computing in applied probability*, vol. 1, no. 2, pp. 127–190, 1999.

[2] C. Pinneri, S. Sawant, S. Blaes, J. Achterhold, J. Stueckler, M. Rolinek, and G. Martius, "Sample-efficient cross-entropy method for real-time planning," *arXiv preprint arXiv:2008.06389*, 2020.

[3] M. Traub, S. Otte, T. Menge, M. Karlbauer, J. Thuemmel, and M. V. Butz, "Learning what and where: Disentangling location and identity tracking without supervision," in *The Eleventh International Conference on Learning Representations*, 2022.