

Neuroevolutionary Reinforcement Learning for Generalized Helicopter Control

Rogier Koppejan



UNIVERSITEIT VAN AMSTERDAM

Introduction



- Autonomous helicopter control
- Challenging because:
 - **Continuous** and **high-dimensional** state and action spaces
 - Complex, **non-linear** helicopter transition dynamics
 - **Noisy** observations, i.e., we do not know the true state
 - High **risk**: even a small mistake can crash the helicopter
- **Generalized** version of the problem where wind varies

Overview

What will be presented:

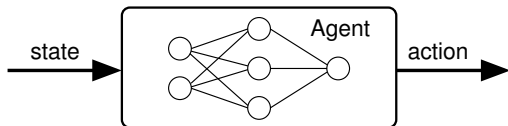
- **Neuroevolutionary** strategies for the helicopter problem
- A **model-free** approach that won the 2008 **RL Competition**
- More complex **model-based** reinforcement learning methods
- Future work: harder generalization of the helicopter domain

Table of Contents

- Background
- Evolving Helicopter Policies
- Learning Helicopter Models
- Model-Free vs. Model-Based Approaches
- Discussion

Policy Representation

- The behavior of the agent is defined by its **policy**, π
- A policy is a mapping from states to actions, $\pi : S \rightarrow A$
- Helicopter policy is represented by an **artificial neural network** (ANN)



Why choose a neural network?

- Concise way of representing a non-linear function
- Very useful in continuous domains
- Models a complex relationship between states and actions

Policy Search

- Network behavior can be altered by modifying the weights
- Goal: find the **optimal policy** π^* that maximizes long-term reward
- Finding the optimal policy (network) is not trivial (large search space)
- Solution: search directly in policy space, i.e., **policy search**
- We will use **neuroevolution** to search for the best policy (network)

Neuroevolution (NE)

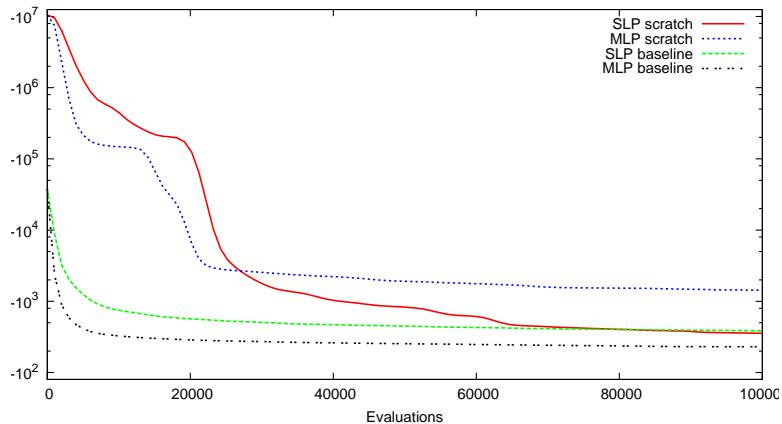
- Stochastic optimization: simulates process of **natural selection**
- Search for a specialized solution in a **population** of ANNs
- Useful for control tasks in continuous and high-dimensional domains



Evolving Helicopter Policies

- Given a **single** SDP with **fixed** wind, can we evolve optimized policies?
- We consider four approaches to evolving helicopter policies:
 - ① Fully connected **single-layer perceptrons**, initial weights set to 0.0
 - ② Fully connected SLPs, initial weights equal to a **baseline controller**
 - ③ Manually designed **multi-layer perceptrons**, starting with zero-weights
 - ④ Manually designed MLP with behavior similar to the baseline controller

Results (Lower is Better)



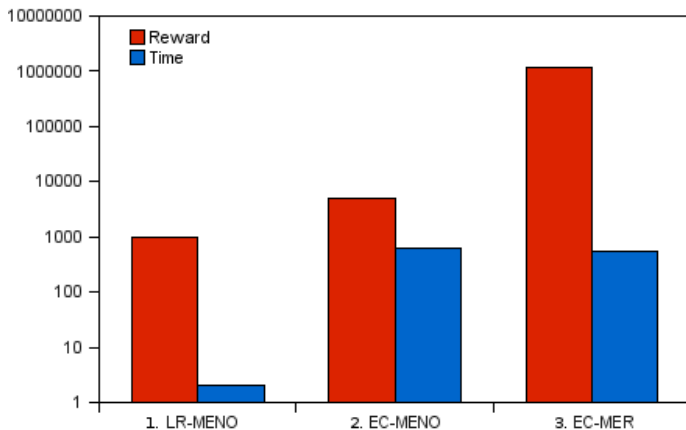
Model-Based Reinforcement Learning

- Model-free approaches have high **sample complexity** (SC)
- Evolving online is infeasible since we are evaluated on cumulative reward
- Reduce SC using a **model** that predicts how the environment responds
- Given observation o_t and action a_t , predict o_{t+1} and reward r_{t+1}
- Thus, a model **simulates** the fitness function required by neuroevolution
- Evolve policies **offline** using a model learned from flight data

Learning Helicopter Models

- Transition function can be represented by a set of linear equations
- Estimate model parameters from flight data such that they:
 - ① Minimize error in predicted **next state** using **linear regression**
 - ② Minimize error in predicted next state with a **genetic algorithm**
 - ③ Minimize error in predicted **reward** also using a GA

Results (Lower is Better)



Model-Free Approach (2008)

Given: a set P of **pre-trained** specialized policies:

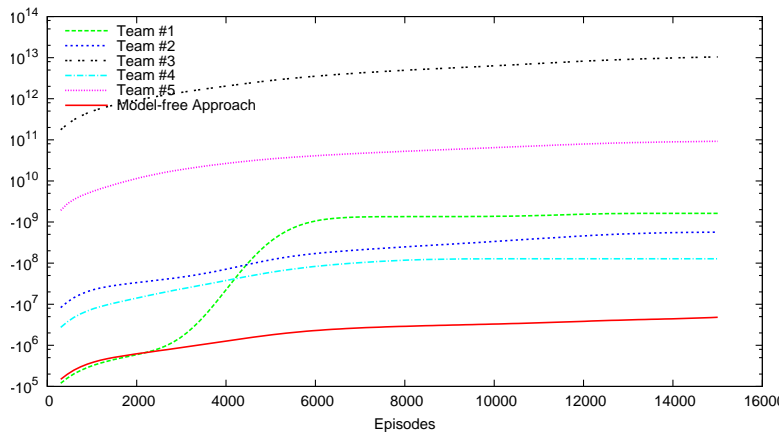
Strategy

- For each policy in P , evaluate it on helicopter
- Select policy that performs best and use it for the remaining episodes

Pros:

- **Sample-efficient**
- No need for an accurate model

2008 Competition Results (Lower is Better)



Model-Based Approach (Post-2008)

Strategy

- Gather flight data using a **generic policy**
- Estimate model from flight data using linear regression
- Evolve policy using the learned model as a fitness function
- Employ specialized policy for the remaining episodes

Incremental Approach

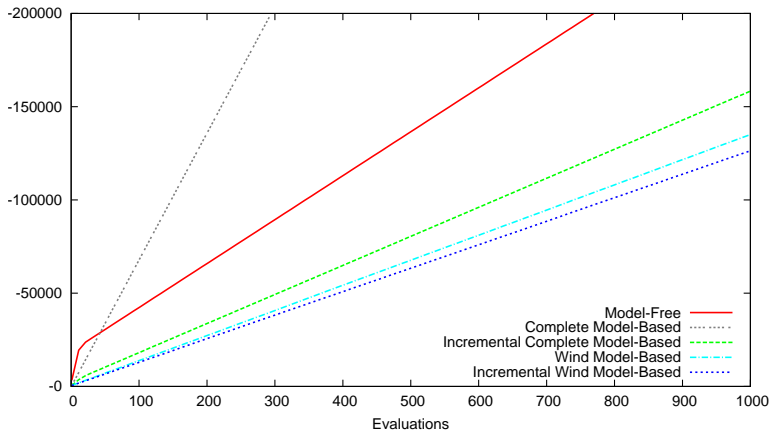
Update model with new flight data and improve policy until:

- ① Current reward is below a predefined **threshold**
- ② Evaluations without improvement exceed some predefined number

Pros:

- **Reliable** *if* we use the incremental setup (noise)
- Better performance than Model-Free approach

Model-Free vs. Model-Based (Lower is Better)



2009 Competition

The 2009 helicopter dynamics differ from previous year and are **unknown**:

- Model-Free approach still works but is **suboptimal**
- Model-Based approach is no longer reliable due to model discrepancy

Ways of altering our model that proved unsuccessful:

- Non-stationary wind speed
- Wind in z-direction

Solution: a **hybrid** and **risk-averse** approach

Hybrid Approach (2009)

A combination of Model-Free and Model-Based RL

Strategy

- Gather flight data using a set of pre-trained policies
- Evolve a few policies offline based on the learned model and evaluate
- Use best policy (pre-trained or evolved) for remaining time

Risk-Aversion

- Check at each step if the agent is in a **dangerous state**
- If so, revert to generic (safe) policy
- Evolve new policy to replace the unsafe policy and select current best

Pros:

- Best of both worlds: sample efficient & near-optimal performance
- **Risk-Averse**: avoid the possibility of crashing

Performance vs. Computational Complexity

- Results of the 2009 proving runs (1 run/approach, 20 SDPs/run)
- Model-Based approach thresholds are set to:
 - ① Minimal negative reward: 300
 - ② Evaluations without improvement: 3

Results

Approach	Avg. Reward	Total Time
Model-Free	-523.10	±15h
Risk-Averse Model-Based	-364.77	±96h
Risk-Averse Hybrid	-223.04	±48h

Table: Average reward and total running time for one proving run.

Conclusions

- NE is effective at finding an optimal helicopter controller
- Models trained to minimize one-step error are the most accurate
- Linear regression is the fastest and most effective way to learn a model
- Model-free strategies are sample efficient but yield sub-optimal performance in the generalized helicopter domain
- Model-based approaches are very powerful when it comes to maximizing cumulative reward but have high computational complexity
- Being risk-averse helps to reducing the possibility of crashing
- Competition is not challenging enough: generic policy never crashes

Future Work

- Currently, exploration is not very challenging, but it is central in RL
- **High risk domains:** increase generalization, exploration becomes hard
- Example: generalize entire model, effectively altering the helicopter type
- Determine risk-aversiveness as a function of the number of test runs

Questions

Manually Designed MLP

