Optimisation of Reinforcement Learning using Evolutionary Algorithms

Mario Fokken

Master Thesis

24.04.2025

Carl von Ossietzky University of Oldenburg

Contents

1. Introduction

Introduction

- Agent systems have become more important in powergrids
- These agents i.a. use DRL algorithms
- Network's architecture has big impact on performance

Problem:

- Network architecture is currently done by hand
  - Standard architecture not optimised
  - Knowledge and thinking needed

➡ Idea:

Neural Architecture Search (NAS) algorithms may find an architecture with better performance in a reasonable time in the ARL context.

2. Basics

Agents

- Autonomous computer system
- Functions:
  1. Sensing
  2. Processing
  3. Acting

- Five classes:
  - Simple reflex
  - Model-based reflex
  - Goal-based
  - Utility-based
  - Learning

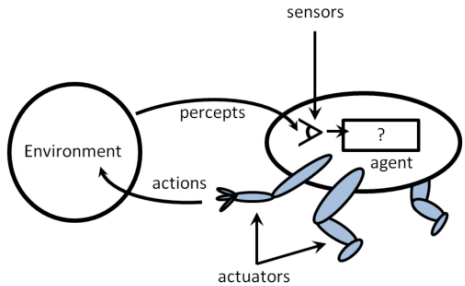


Figure 1: A depiction of an agent with its core functions after Russell and Norvig *

[1], [2], [3]

Deep Learning

- Refers to the use of *neural networks*
- Based upon the human brain

-> Multiple layers of interconnected Neurons

- Input layer gets data
- Hidden layers used for calculation
- Output layer gives result

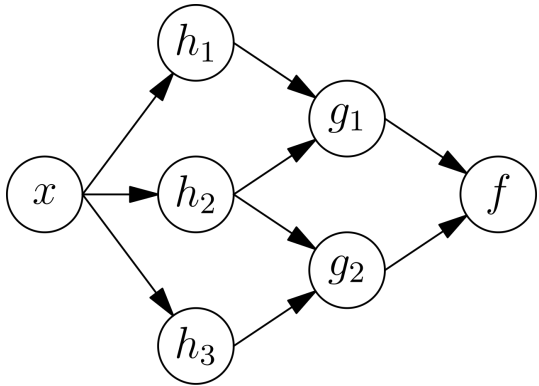


Figure 2: A basic representation of a neural network *

[4], [5]

Deep Learning (ii)

- A neuron has at least one input as well as one output
- Each connection has a weight
- Each neuron has an activation function
  - e.g. $\tanh(x)$ or $\mathrm{ReLU}(x) = \max(0, x)$
- Formula:

$$h_1 = A\left(\sum_{c=1}^{n}(i_c * w_c)\right)$$

- Learning is done by changing weights

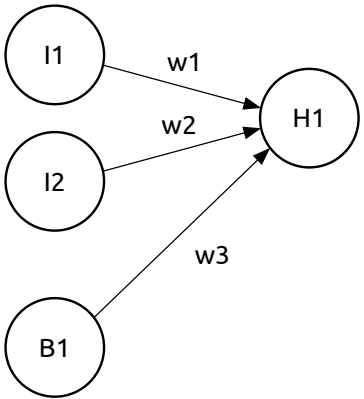


Figure 3: A single hidden neuron with its inputs

[4], [5]

Reinforcement Learning

The Bellman principle of optimality is a central part of reinforcement learning:

> "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision."
>
> — Richard Bellman

or, as equation:

$$V^{\pi^*}(x) = \max_a \left[R_{x(a)} + \gamma \sum_y P_{\mathrm{xy}}(a) V^{\pi^*}(y) \right]$$

➡ An optimal policy is achievable by doing the best action in each state [6]

Reinforcement Learning (ii)

1. Agent interacts with the environment
2. Agent receives reward based on its actions
3. Agent tries to maximise reward

➡ Agent learns

Two approaches:

- On-Policy: learning, acting in 1 policy
- Off-Policy: 1 learning, 1 acting policy

➡ On-Policy is generally more stable and simpler, but cannot reuse data samples, hence less data efficient

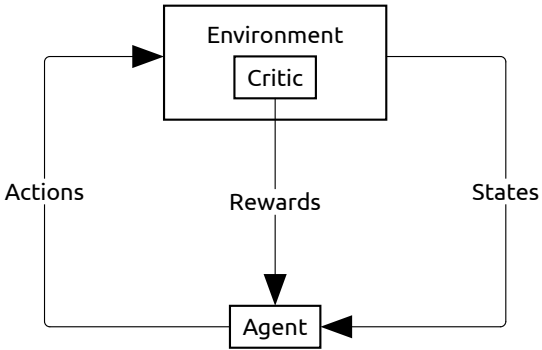


Figure 4: The feedback-loop of RL algorithms

[7]

Soft Actor-Critic

- Off-Policy, actor-critic RL algorithm
- Not only maximises the expected reward, but also the entropy $H(\pi)$ of the policy:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \Big(r_t + \alpha H(\pi(\cdot\,|s_t)) \Big)\right]$$

- Concurrently learns a stochastic policy π_θ and two Q-functions Q_{ϕ_1} and Q_{ϕ_2}
  - Stochastic policy (actor): represents the probability of taking action a in state s
  - Q-Function (critic): using the Bellman equation, estimates long-term reward for taking action a in state s and following policy π afterwards

[8], [9]

Neuroevolution

- Uses an evolutionary process to evolve neural networks:

1. Create population of individuals
2. Rate each individual's performance at the task with a fitness score
3. Select the fittest individuals
4. Create new individuals with the selected ones
5. Repeat steps 2-5 until satisfied

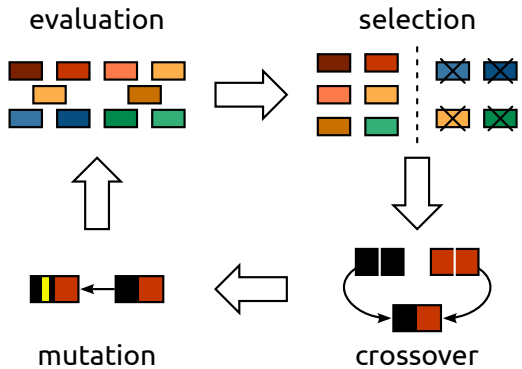


Figure 5: A diagram showing the steps of a genetic algorithm *

[10]

NEAT

- NeuroEvolution of Augmenting Topologies
- Developed by Kenneth O. Stanley and Risto Miikkulainen in 2001
- TWEANN (Topology and Weight Evolving Artificial Neural Networks) algorithm
- Upgrades the known TWEANN algorithms with 3 key features

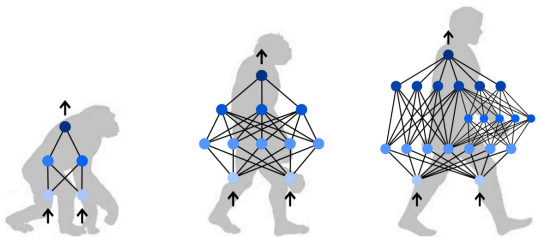


Figure 6: Evolution of a network *

[11]

NEAT (ii)

- Genetic Encoding
- Historical Markings
  - Allows for crossover of different topologies
- Speciation
  - New, unoptimised structures are bad for fitness
  - Grouping similar individuals and compare in group
- Minimising Dimensionality
  - Starts with simple networks and increases complexity

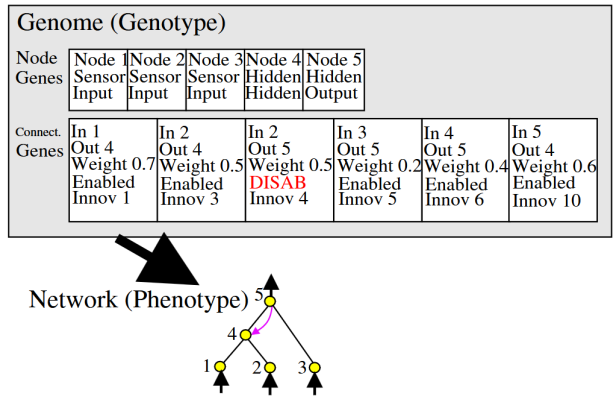


Figure 7: The scheme NEAT uses for its genomes

[11]

Bayesian optimisation

- Strategy for global optimisation of black-box function $f(x)$
- Surrogate function:
  best guess of the form of $f(x)$
- Acquisition function:
  directs the exploration of the space

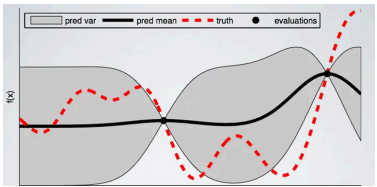


Figure 8: BO with 2 evaluations *

Bayesian Optimisation

for $n = 1, 2, \ldots$ **do**

select new x_{n+1} by optimising acquisition function α:

$$x_{n+1} = \arg\max_x \alpha(x; \mathcal{D}_n)$$

query objective function to obtain

$$y_{n+1} = f(x_{x+1})$$

augment data set

$$\mathcal{D}_n = \{\mathcal{D}_n, (x_{n+1}, y_{n+1})\}$$

update surrogate function

end for

[12]

Neural Architecture Search

- Tries to automate the design of neural network architectures
- Several possible approaches:
  - Reinforcement learning
  - Evolutionary algorithms
  - Gradient-based optimisation
- NAS was successfully used in several applications and studies
- But most NAS applications focus on image classification tasks

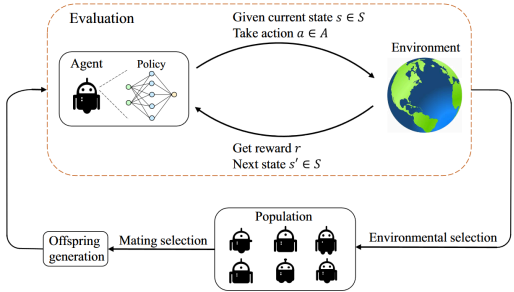


Figure 10: How an evolutionary algorithm is normally used to optimise RL

[13], [14], [15], [16], [17]

3. Stack

Stack

- palaestrAI
- NEAT
  - pytorch-neat by ddehueck
- RL algorithm
  - minimal-nas by nicklashansen
- Bayesian Optimisation
  - BayesianOptimization by bayesian-optimization

Why these? Others either

- had no usable implementation
- were optimised for image classification
- did not use PyTorch

4. Implementation

Concept

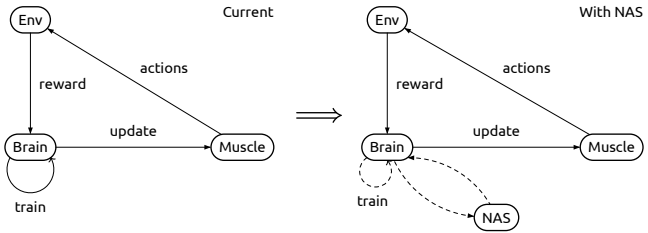


Figure 11: Adding NAS to the palaestrAI learning cycle

- ‘Big Loop’ not possible

➡ Break up NAS algorithm to run within learning loop

- After the NAS finishes, use best performing network to continue with ‘normally’

NEAT

0. Population is generated during setup
1. Network is taken from population and run several times
2. Resulting rewards are used to determine fitness (average)
3. When every network has been run, select the best performing networks
4. Generate new population
5. Go to 1. until 'satisfied'
   i.e. fitness over specified threshold or reaching a number of generations

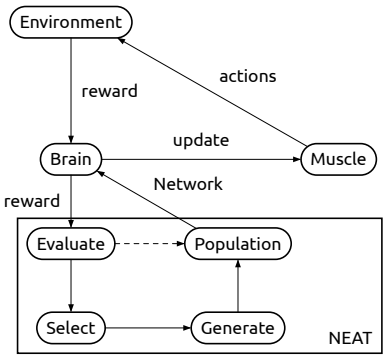


Figure 12: NEAT concept

RL

1. RL generates a set of ‘actions’ out of

```
0: 2, 1: 4, 2: 8, ..., 8: 128,
9: 'tanh', 10: 'ReLu', 11: 'EOS'
```

2. Actions are used to create a network
3. Network is run through the cycle and trained ‘normally’
4. Loss and rewards are used as feedback to the RL algorithm

- Repeats until a set amount of iterations finishes
- Select best performing network

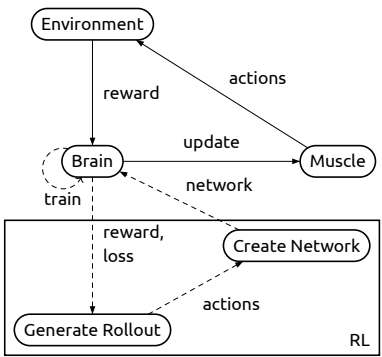


Figure 13: RL approach

BO

- Black-box function is made of an encoding of the network
  - e.g. a, b, c with bounds (0, 256) representing the hidden layers

1. BO algorithms proposes a set of parameters
2. Parameters are used to create a network
3. Network is run through the cycle and trained ‘normally’
4. BO uses the rewards to improve model

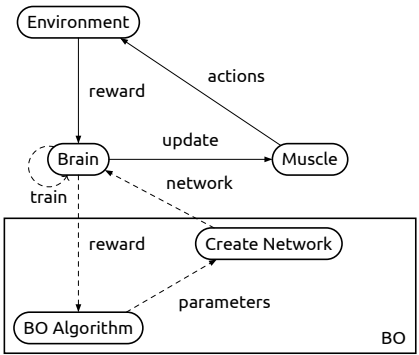
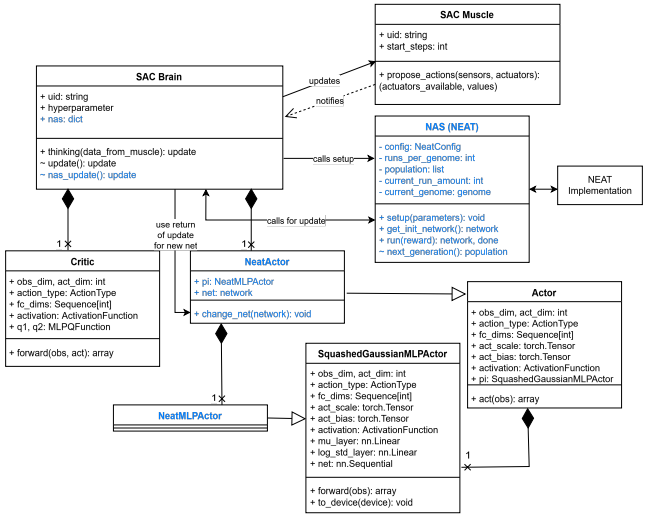
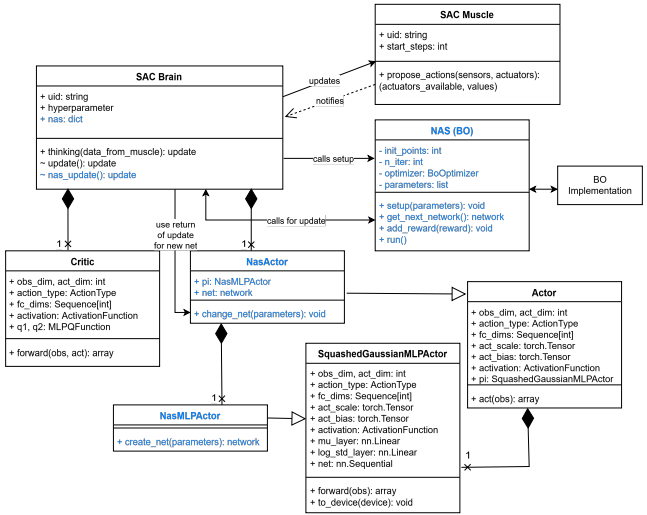


Figure 14: BO approach

UML


SAC Muscle
+ uid: string
+ start_steps: int
+ propose_actions(sensors, actuators): (actuators_available, values)
SAC Brain
+ uid: string
+ hyperparameter
+ nas: dict
+ thinking(data_from_muscle): update
~ update(): update
~ nas_update(): update
updates
notifies
calls setup
NAS (NEAT)
- config: NeatConfig
- runs_per_genome: int
- population: list
- current_run_amount: int
- current_genome: genome
+ setup(parameters): void
+ get_init_network(): network
+ run(reward): network, done
~ next_generation(): population
NEAT Implementation
use return of update for new net
calls for update
1
1
Critic
+ obs_dim, act_dim: int
+ action_type: ActionType
+ fc_dims: Sequence[int]
+ activation: ActivationFunction
+ q1, q2: MLPQFunction
+ forward(obs, act): array
NeatActor
+ pi: NeatMLPActor
+ net: network
+ change_net(network): void
Actor
+ obs_dim, act_dim: int
+ action_type: ActionType
+ fc_dims: Sequence[int]
+ act_scale: torch.Tensor
+ act_bias: torch.Tensor
+ activation: ActivationFunction
+ pi: SquashedGaussianMLPActor
+ act(obs): array
1
NeatMLPActor
SquashedGaussianMLPActor
+ obs_dim, act_dim: int
+ action_type: ActionType
+ fc_dims: Sequence[int]
+ act_scale: torch.Tensor
+ act_bias: torch.Tensor
+ activation: ActivationFunction
+ mu_layer: nn.Linear
+ log_std_layer: nn.Linear
+ net: nn.Sequential
+ forward(obs): array
+ to_device(device): void
1

UML (ii)


SAC Muscle
+ uid: string
+ start_steps: int
+ propose_actions(sensors, actuators): (actuators_available, values)
updates
notifies
SAC Brain
+ uid: string
+ hyperparameter
+ nas: dict
+ thinking(data_from_muscle): update
~ update(): update
~ nas_update(): update
calls setup
NAS (BO)
- init_points: int
- n_iter: int
- optimizer: BoOptimizer
- parameters: list
+ setup(parameters): void
+ get_next_network(): network
+ add_reward(reward): void
+ run()
BO Implementation
use return of update for new net
calls for update
1
1
Critic
+ obs_dim, act_dim: int
+ action_type: ActionType
+ fc_dims: Sequence[int]
+ activation: ActivationFunction
+ q1, q2: MLPQFunction
+ forward(obs, act): array
NasActor
+ pi: NasMLPActor
+ net: network
+ change_net(parameters): void
Actor
+ obs_dim, act_dim: int
+ action_type: ActionType
+ fc_dims: Sequence[int]
+ act_scale: torch.Tensor
+ act_bias: torch.Tensor
+ activation: ActivationFunction
+ pi: SquashedGaussianMLPActor
+ act(obs): array
1
NasMLPActor
+ create_net(parameters): network
SquashedGaussianMLPActor
+ obs_dim, act_dim: int
+ action_type: ActionType
+ fc_dims: Sequence[int]
+ act_scale: torch.Tensor
+ act_bias: torch.Tensor
+ activation: ActivationFunction
+ mu_layer: nn.Linear
+ log_std_layer: nn.Linear
+ net: nn.Sequential
+ forward(obs): array
+ to_device(device): void
1

5. Experiment

Experiment

- All 3 NAS and baseline were tested
- CIGRE Medium Voltage Net is used
- NAS agent as defender, no attacker
- Task: keeping the power grid stable

1st Experiment:

- ARL Defender Objective
- 3 episodes à 100 days simulated
- step size of 900

2nd Experiment:

- COHDARL Defender Objective
- 15 Mio. sec. (ca. 174 days) per episode

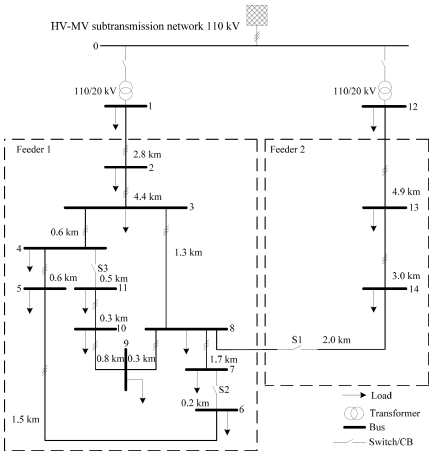


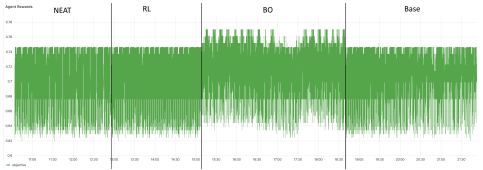
Figure 17: CIGRE Medium Voltage Net

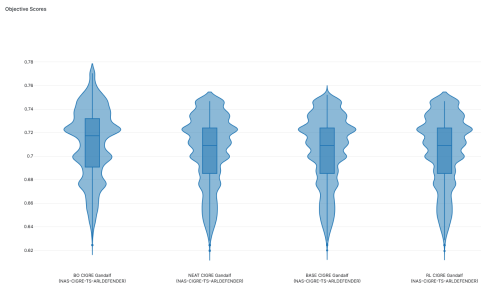
[18]

Result Experiment 1

- RL takes 2:11h, NEAT 2:21h, Base 3:12h, BO 3:31h
- NEAT and RL perform similarly to Baseline (all median reward of 0.709)
- BO performs better (median reward 0.717)

➡ Hypothesis is true, NAS algorithms can perform better in a reasonable time

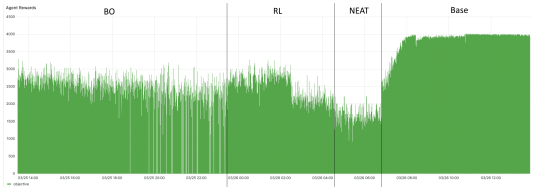


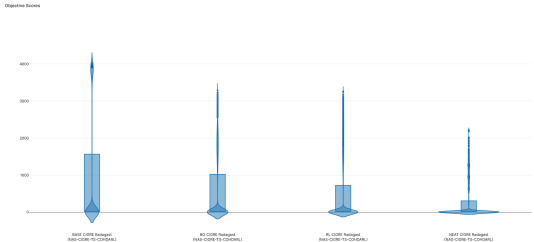


Result Experiment 2

- NEAT takes 2:14h, RL 5:03h, Base 7:02h, BO 9:57h
- NEAT performs worst (1.598)
- Base performs best (3.778)
- Baseline only one that “learns”

➡ Hypothesis cannot be confirmed, NAS methods do not perform





Evaluation

- Contrary results: 1st experiment agrees to hypothesis, 2nd does not
- In both experiments, the NAS methods do not learn
  - Baseline also does not learn in 1st experiment
    ➡ Experiment might not work as intended
  - Either implementation of NAS methods is not correct or the task is not suitable for NAS methods
    ➡ More research needed
- NEAT had a problem in the 2nd experiment when fitted all actuators
  - Stopped after first step; due to the unique network structure in NEAT's impl.
  - Fewer actuators were used, hence the significantly worse performance

6. Conclusion

Further Work

- BO approach has big room for improvement
  - Black-box function used has 6 parameters, each between 0 and 256
  - Baseline used 2 hidden layers with 48 features

  ➡ Improving function may speed up learning and perform better
- Changing the NEAT implementation's network to one more compatible
- Duration of the NAS could be set automatically
- Other algorithms could be tested

[19]

Conclusion

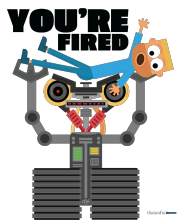
- Goal of thesis was to confirm the hypothesis :

> a NAS algorithm can improve upon the performance of a user-picked architecture in a reasonable amount of time in the ARL architecture

- The hypothesis was confirmed in the 1st experiment, but not in the 2nd
- 1st experiment did not show a learning baseline, thus 2nd is more meaningful

➡ The hypothesis cannot be confirmed,
further research may lead to satisfactory results

Summary



- Hypothesis: a NAS algorithm can perform better than manually picked architectures in a reasonable time in the ARL architecture
- Three NAS algorithms were tested:
  - NEAT (Evolutionary Algorithm)
  - An reinforcement learning approach (“Minimal-NAS”)
  - Bayesian optimisation
- 1st exp. showed BO performing better, but may not be meaningful since no agent learnt
- 2nd exp. showed no NAS method performing, with Baseline only one learning

➡ Hypothesis could not be proven, further research needed

Bibliography

[1] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. pearson, 2016.

[2] M. Wooldridge, 'Intelligent agents', *Multiagent systems: A modern approach to distributed artificial intelligence*, vol. 1, pp. 27–73, 1999.

[3] M. Wooldridge, *An introduction to multiagent systems*. John wiley & sons, 2009.

[4] J. A. Anderson, *An introduction to neural networks*. MIT press, 1995.

[5] J. Heaton, 'Introduction to the Math of Neural Networks (Beta-1)', *Heaton Research Inc*, 2011.

[6] R. Bellman, 'The theory of dynamic programming', *Bulletin of the American Mathematical Society*, vol. 60, no. 6, pp. 503–515, 1954.

[7] E. Puiutta and E. M. Veith, 'Explainable reinforcement learning: A survey', in *International cross-domain conference for machine learning and knowledge extraction*, 2020, pp. 77–95.

[8] T. Haarnoja *et al.*, 'Soft actor-critic algorithms and applications', *arXiv preprint arXiv:1812.05905*, 2018.

[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, 'Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor', in *International conference on machine learning*, 2018, pp. 1861–1870.

[10] K. O. Stanley, 'Neuroevolution: A different kind of deep learning – oreilly.com'. 2017.

[11] K. O. Stanley and R. Miikkulainen, 'Evolving Neural Networks through Augmenting Topologies', *Evolutionary Computation*, vol. 10, no. 2, pp. 99–127, 2002, doi: 10.1162/106365602320169811.

Bibliography (ii)

[12] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, 'Taking the human out of the loop: A review of Bayesian optimization', *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[13] H. Bai, R. Cheng, and Y. Jin, 'Evolutionary Reinforcement Learning: A Survey', *Intelligent Computing*, vol. 2, no. , p. 25, 2023, doi: 10.34133/icomputing.0025.

[14] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. G. Yen, and K. C. Tan, 'A survey on evolutionary neural architecture search', *IEEE transactions on neural networks and learning systems*, vol. 34, no. 2, pp. 550–570, 2021.

[15] T. Elsken, J. H. Metzen, and F. Hutter, 'Neural architecture search: A survey', *Journal of Machine Learning Research*, vol. 20, no. 55, pp. 1–21, 2019.

[16] Y. Fu, Z. Yu, Y. Zhang, and Y. Lin, 'Auto-agent-distiller: Towards efficient deep reinforcement learning agents via neural architecture search', *arXiv preprint arXiv:2012.13091*, 2020.

[17] N. Mazyavkina, S. Moustafa, I. Trofimov, and E. Burnaev, 'Optimizing the neural architecture of reinforcement learning agents', in *Intelligent Computing: Proceedings of the 2021 Computing Conference, Volume 2*, 2021, pp. 591–606.

[18] K. Rudion, A. Orths, Z. Styczynski, and K. Strunz, 'Design of benchmark of medium voltage distribution network for investigation of DG integration', in *2006 IEEE Power Engineering Society General Meeting*, 2006, p. 6. doi: 10.1109/PES.2006.1709447.

[19] T. Kavzoglu and others, 'Determining optimum structure for artificial neural networks', in *Proceedings of the 25th Annual Technical Conference and Exhibition of the Remote Sensing Society*, 1999, pp. 675–682.

Image Sources

Figure 1: www.researchgate.net/figure/ntelligent-Agents-Russell-and-Norvig-2009_fig2_303545602

Figure 2: commons.wikimedia.org/wiki/File:Ann_dependency_(graph).svg

Figure 5: www.strong.io/blog/evolutionary-optimization

Figure 8: miro.medium.com/v2/resize:fit:720/format:webp/1

Figure 22: ih1.redbubble.net/image.4720339522.3301/bg

Figure 23: giphy.com/gifs/spongebob-season-3-spongebob-squarepants-xTeV7ycHGuPnKX92cE