UNIVERSITÄT PASSAU

Search-Based Software Engineering

Neuroevolution

Patric Feldmeier
Chair of Software Engineering II

Neuroevolution

Neuro evolution

Neuro evolution

Neural networks

- Mimic a human brain to solve complex tasks

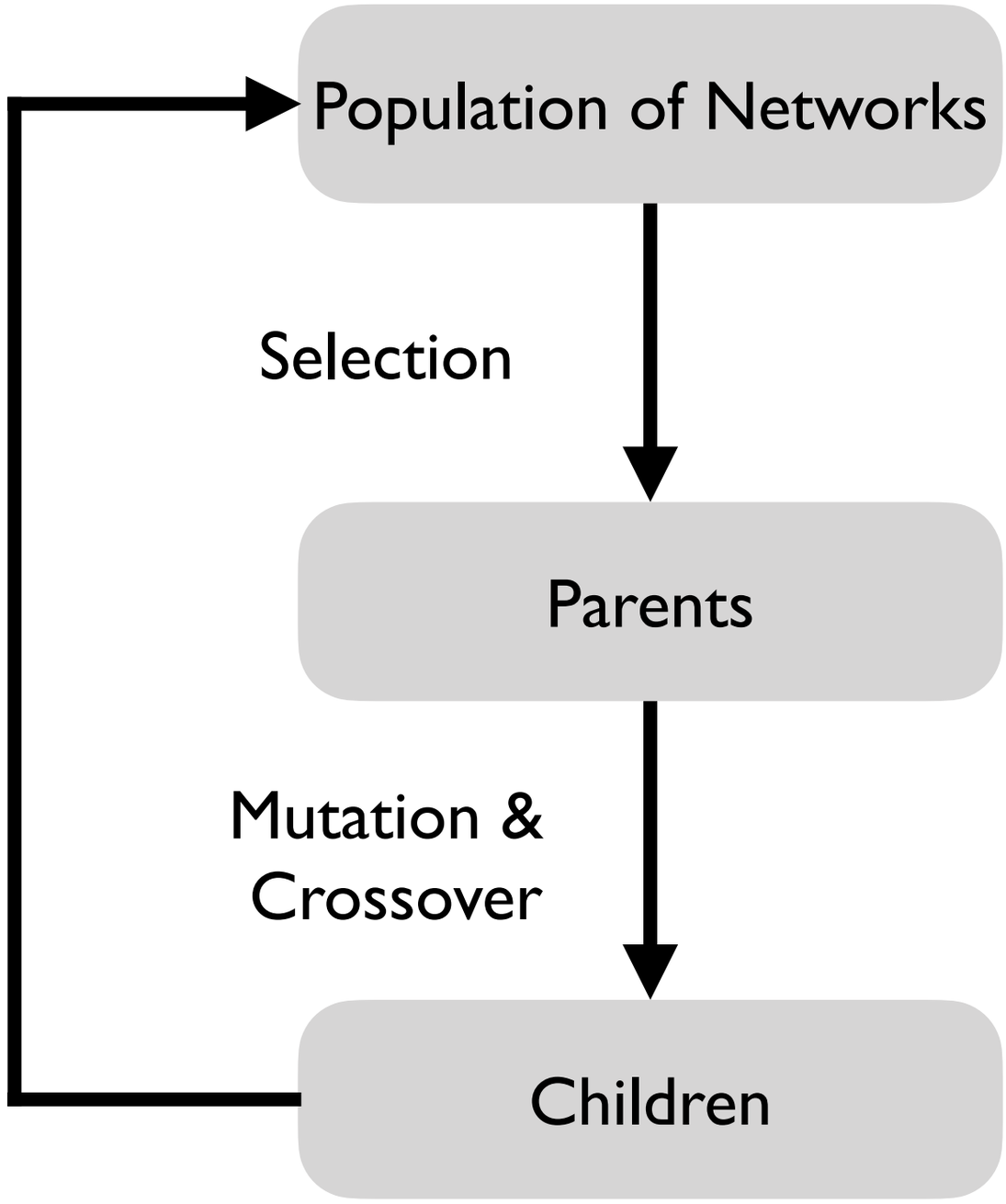
- Must be optimised for each task individually

Neuro evolution

Neural networks

- Mimic a human brain to solve complex tasks

- Must be optimised for each task individually



Example: Super Mario

Classic Nintendo Games are
(Computationally) Hard

Greg Aloupis* Erik D. Demaine† Alan Guo†‡ Giovanni Viglietta§

February 10, 2015

Abstract

We prove NP-hardness results for five of Nintendo's largest video game franchises: Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokémon. Our results apply to generalized versions of Super Mario Bros. 1–3, The Lost Levels, and Super Mario World; Donkey Kong Country 1–3; all Legend of Zelda games; all Metroid games; and all Pokémon role-playing games. In addition, we prove PSPACE-completeness of the Donkey Kong Country games and several Legend of Zelda games.

1 Introduction

A series of recent papers have analyzed the computational complexity of playing many different video games [1, 4, 5, 6], but the most well-known classic Nintendo games have yet to be included among these results. In this paper, we analyze some of the best-known Nintendo games of all time: Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokémon. We prove that it is NP-hard,

Gen 34 species 14 genome 14 (37%)
Fitness: 35 Max Fitness: 4322

A
B
X
Y
Up
Down
Left
Right

Neuroevolution-based Game Testing

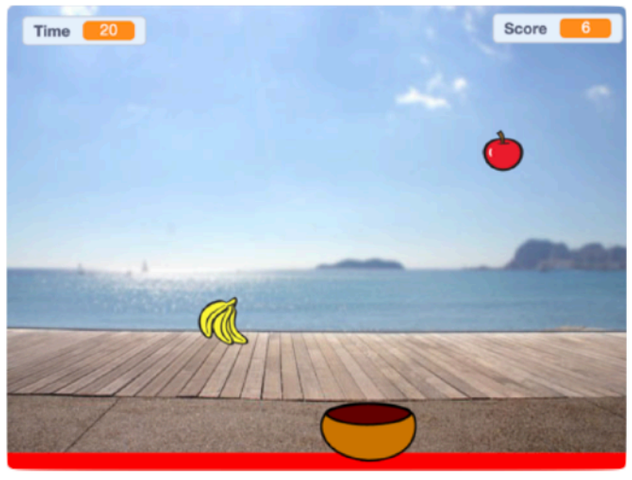
Neuroevolution-Based Generation of Tests and Oracles for Games

Patric Feldmeier
University of Passau
Germany

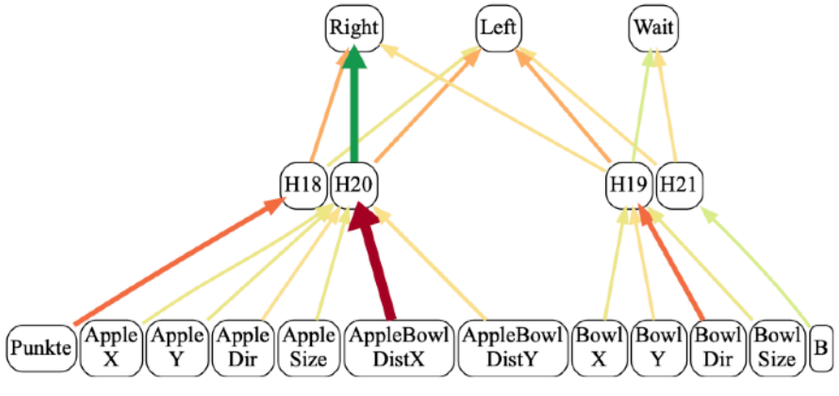
Gordon Fraser
University of Passau
Germany

ABSTRACT

Game-like programs have become increasingly popular in many software engineering domains such as mobile apps, web applications, or programming education. However, creating tests for programs that have the purpose of challenging human players is a daunting task for automatic test generators. Even if test generation succeeds in finding a relevant sequence of events to exercise a program, the randomized nature of games means that it may neither be possible to reproduce the exact program behavior underlying this sequence, nor to create test assertions checking if observed randomized game behavior is correct. To overcome these problems, we propose NEATEST, a novel test generator based on the *NeuroEvolution of Augmenting Topologies* (NEAT) algorithm. NEATEST systematically explores a program's statements, and creates neural networks that operate the program in order to reliably reach each statement—that is, NEATEST learns to play the game in a way to reliably cover different parts of the code. As the networks learn the actual game behavior, they can also serve as test oracles by evaluating how surprising the observed behavior of a program under test is compared to a supposedly correct version of the program. We evaluate this approach in the context of SCRATCH, an educational programming environment. Our empirical study on 25 non-trivial SCRATCH games demonstrates that our approach can successfully train neural networks that are not only far more resilient to random influences than traditional test suites consisting of static input sequences, but are also highly effective with an average mutation score of more than 65%.

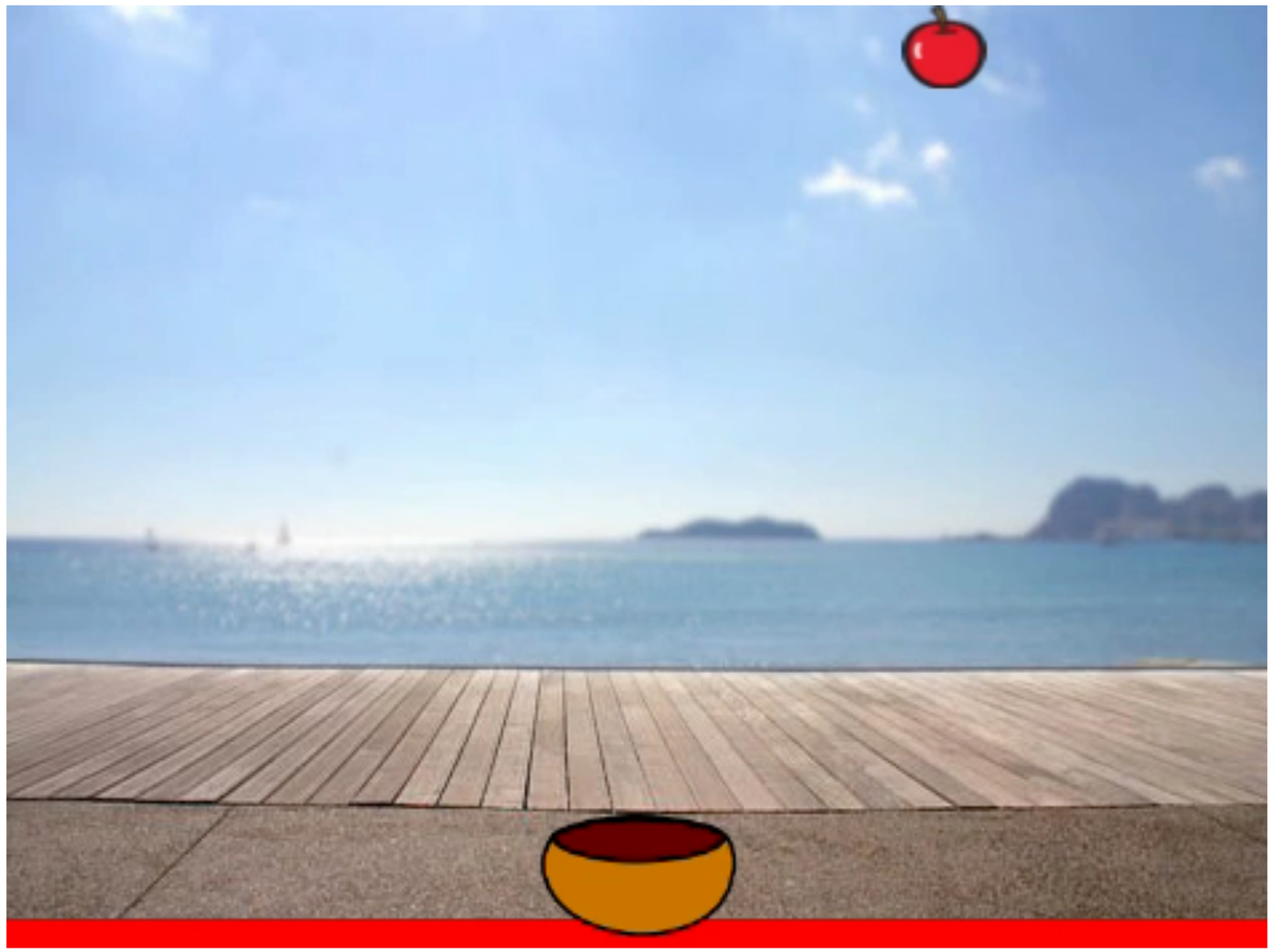


(a) FruitCatching example game.

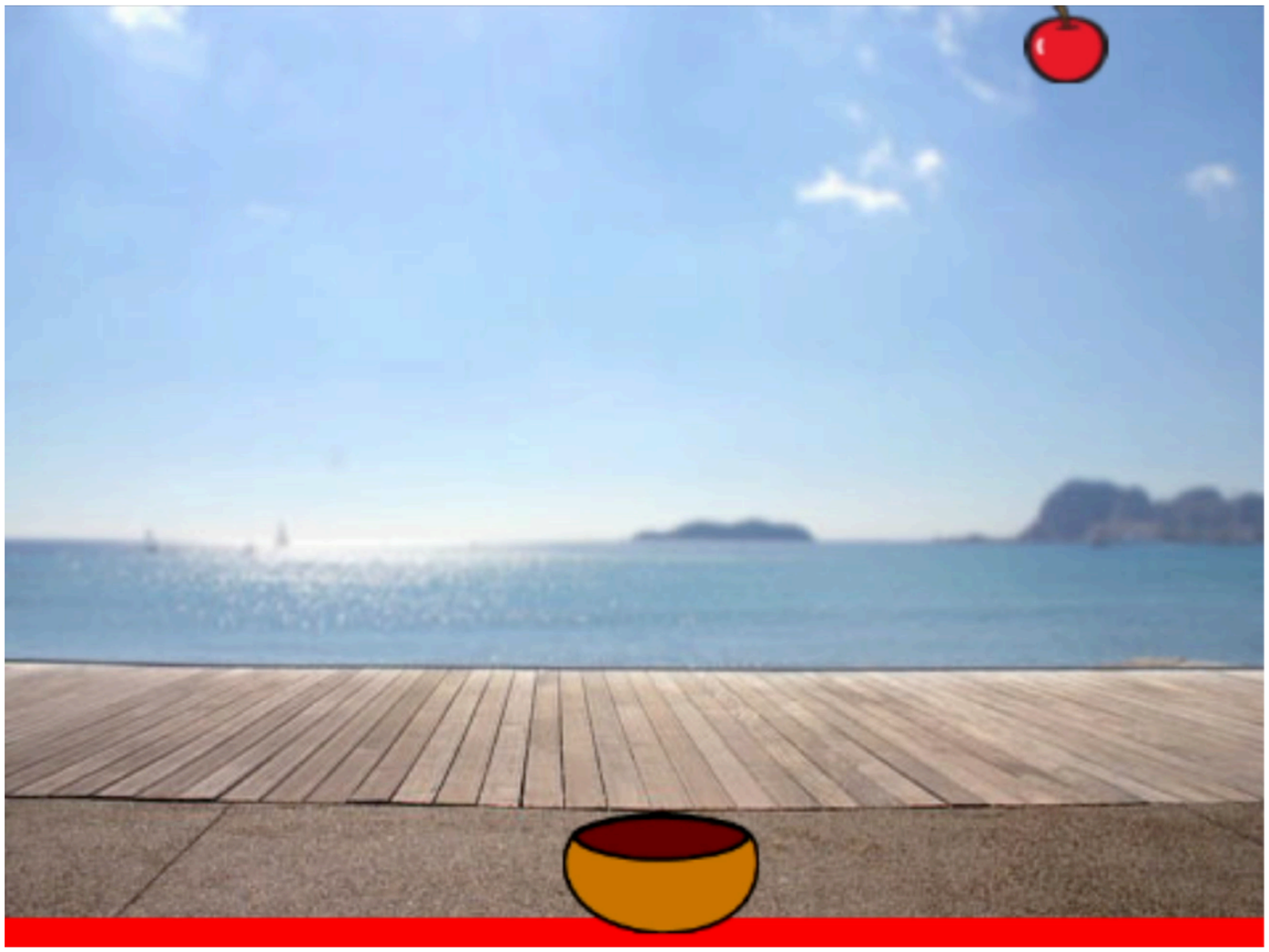


(b) Example of an optimized network with excluded regression head. The width and brightness of connections represent their importance.

Testing Games

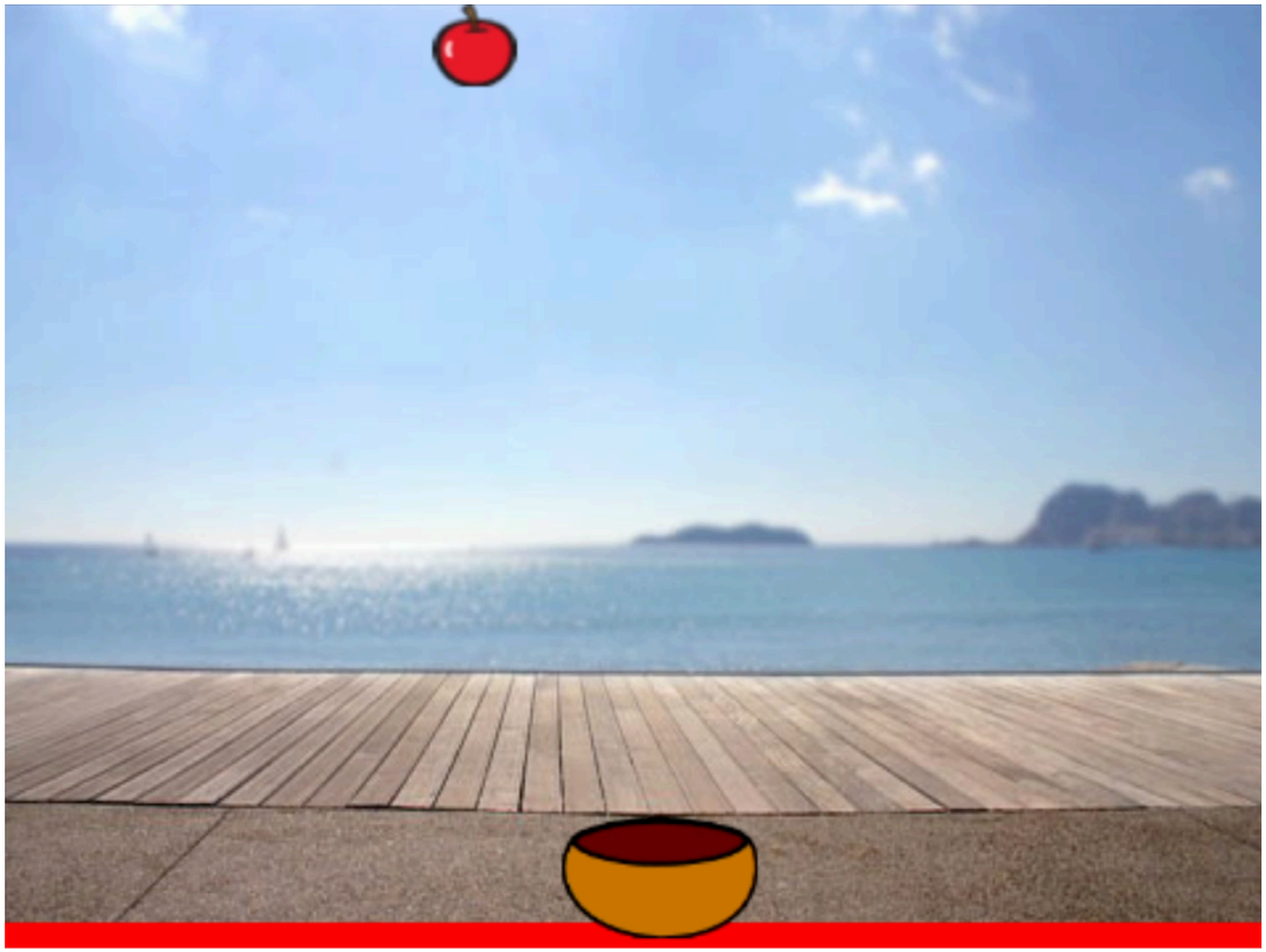


Testing Games



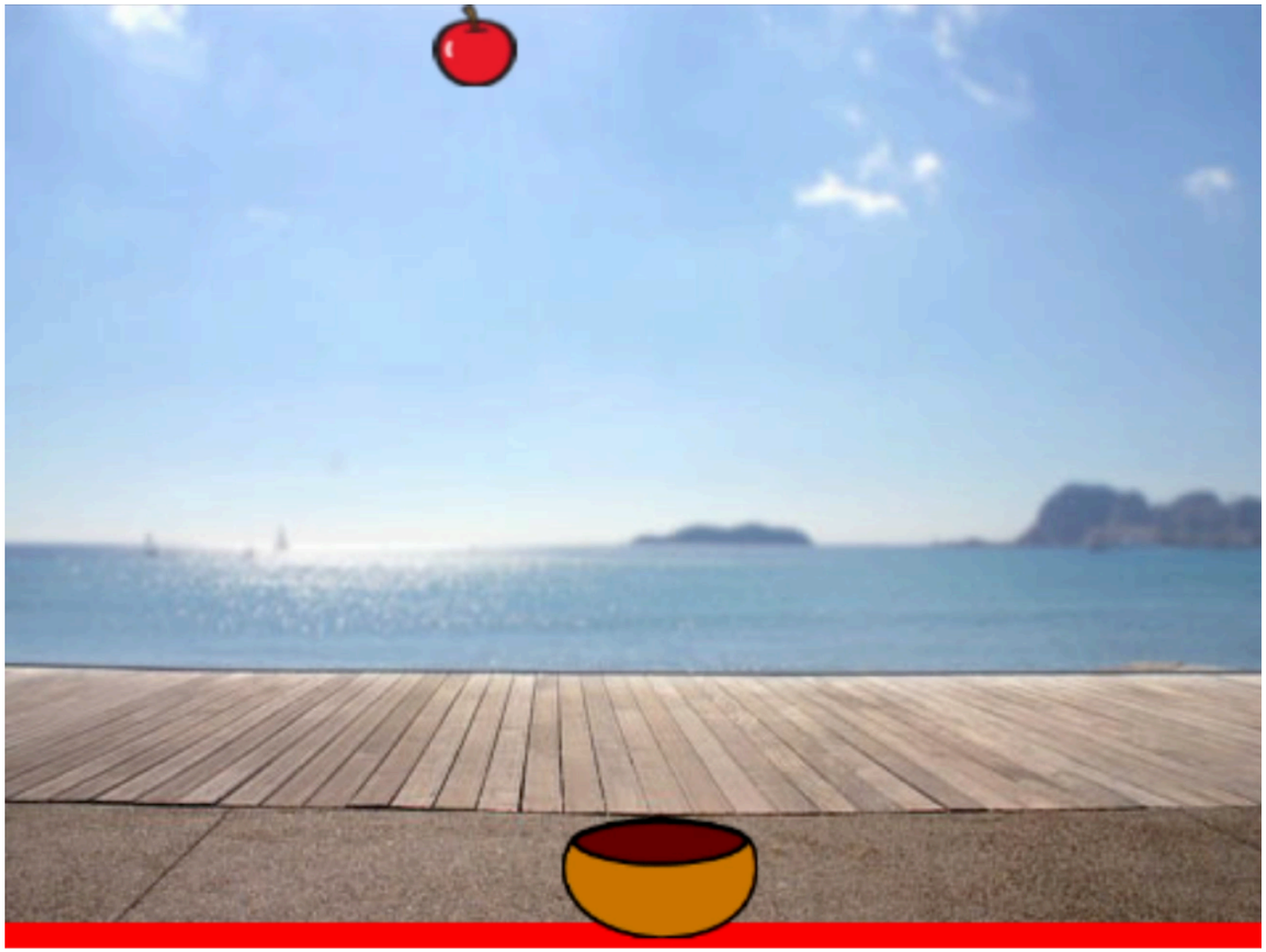
Challenges of Game Testing

Randomisation



Challenges of Game Testing

Randomisation

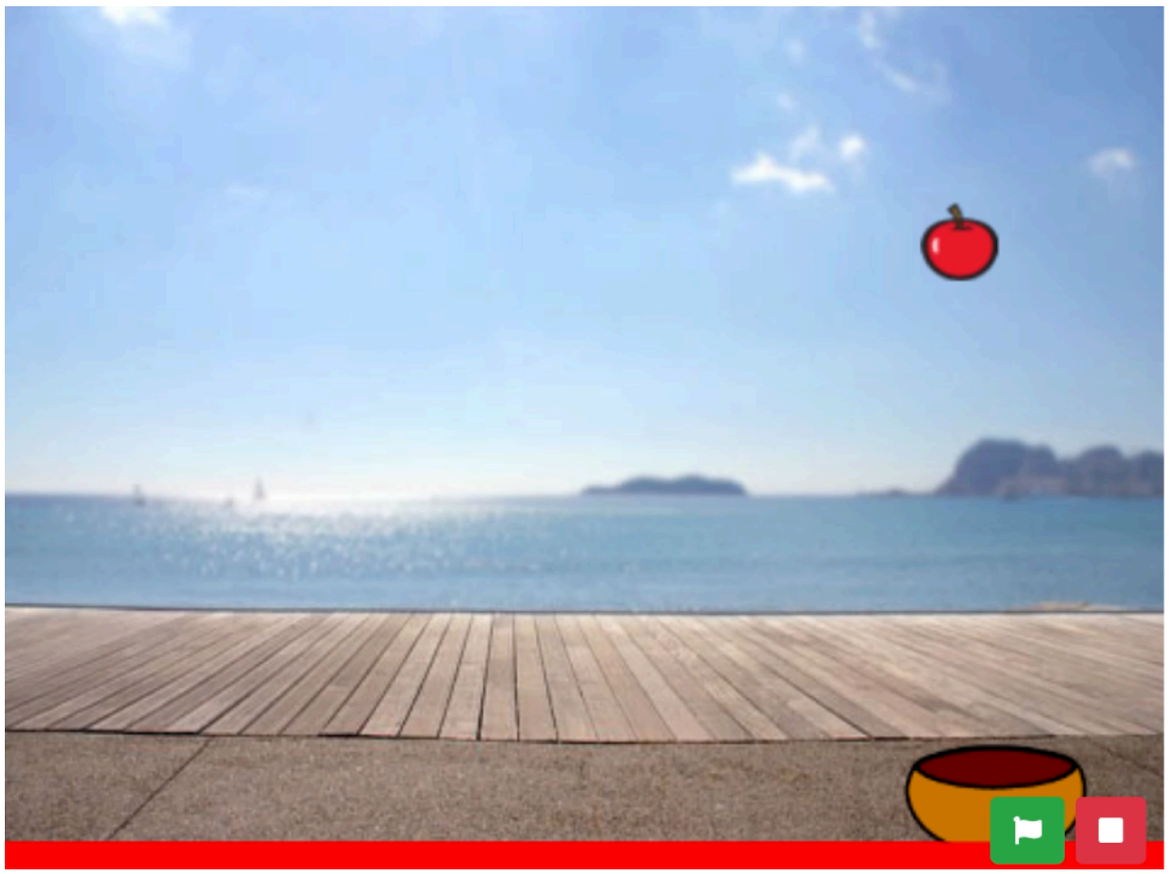


Challenges of Game Testing

Randomisation

Challenging statements

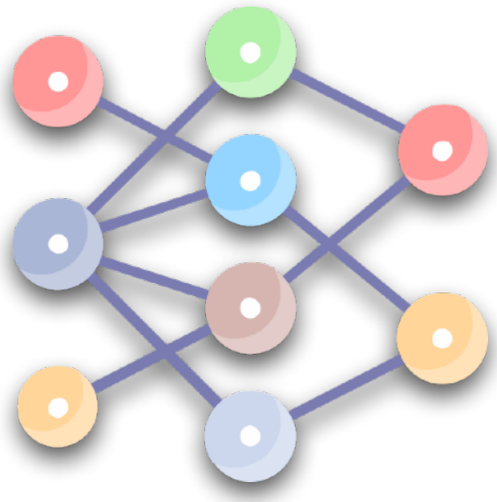
Neatest

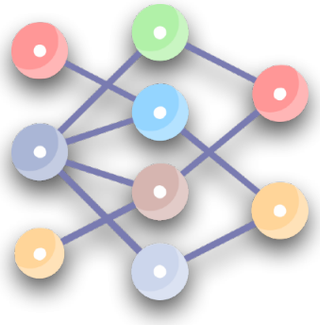


Learn to play

Validate behaviour

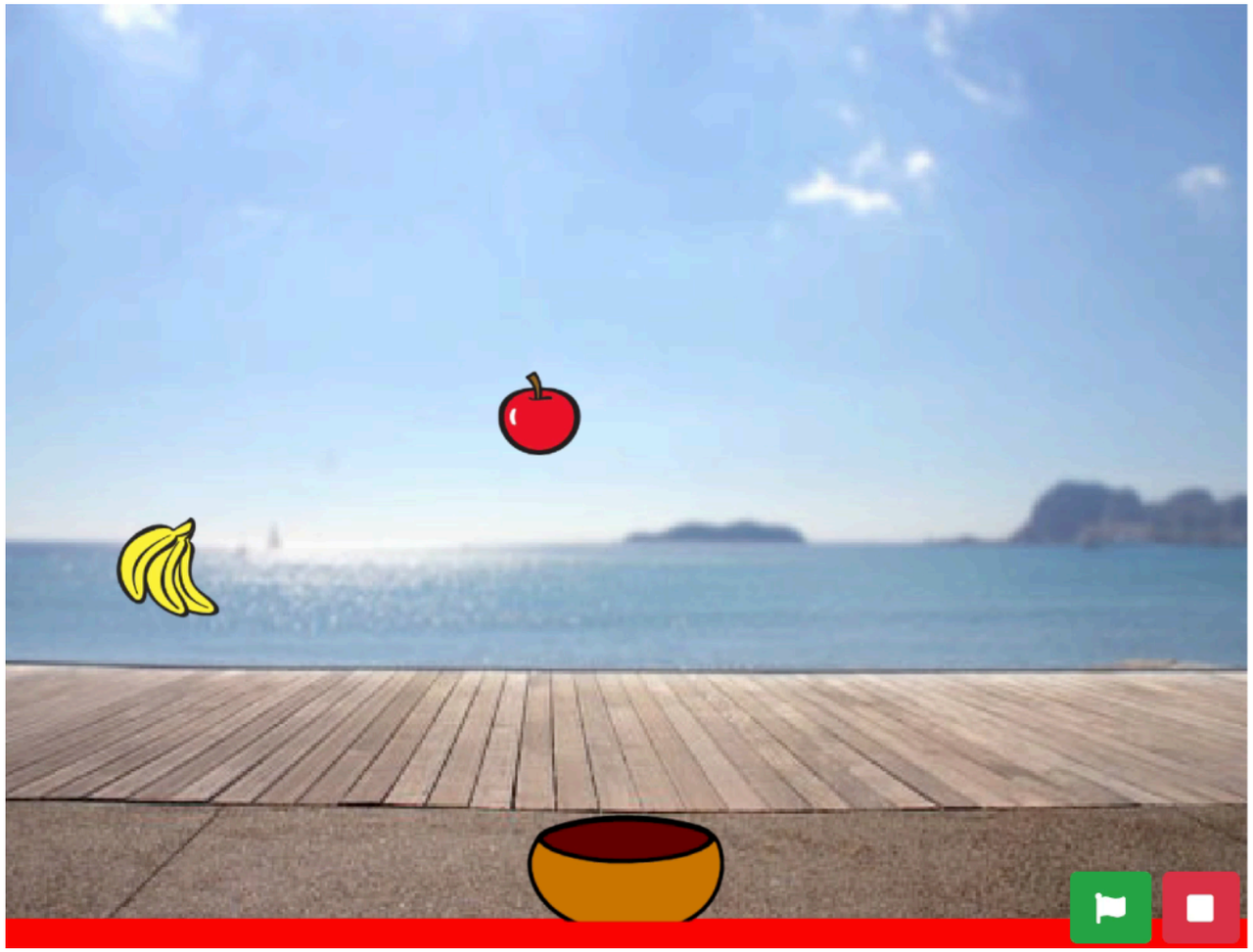
Dynamic Test Suites

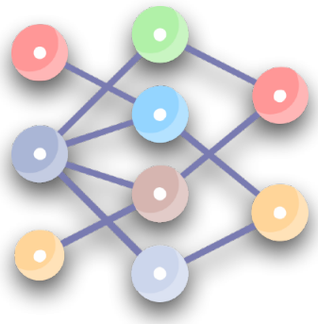




Generating Dynamic Test Suites

Select a target statement

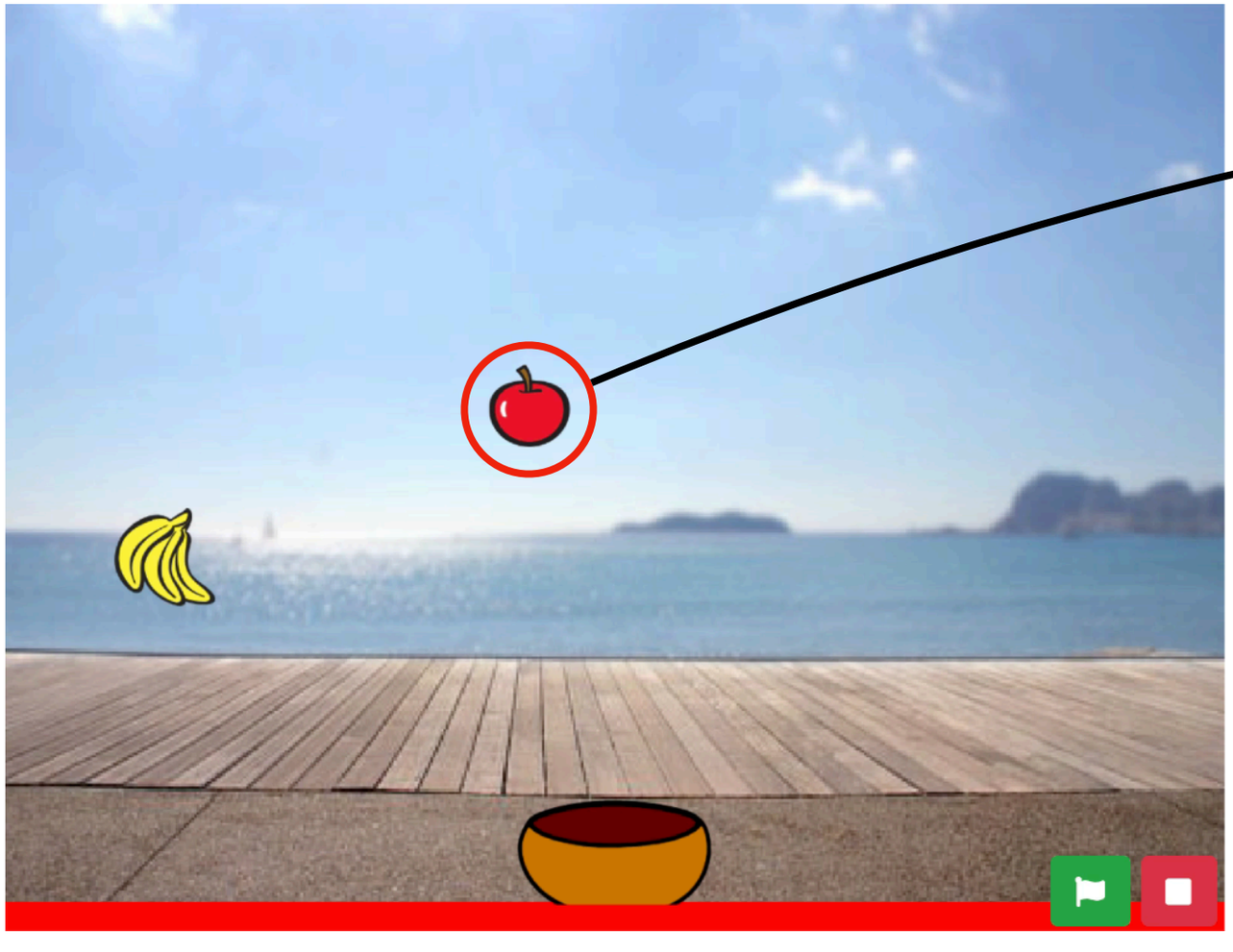




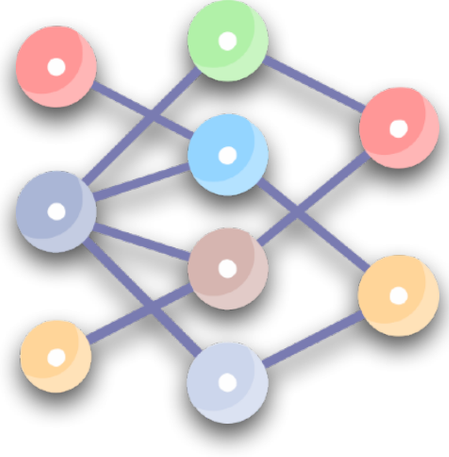
Generating Dynamic Test Suites

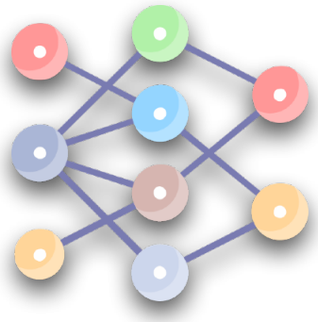
Optimise networks to cover target statement

➡ Fitness = distance to target statement



Neuroevolution



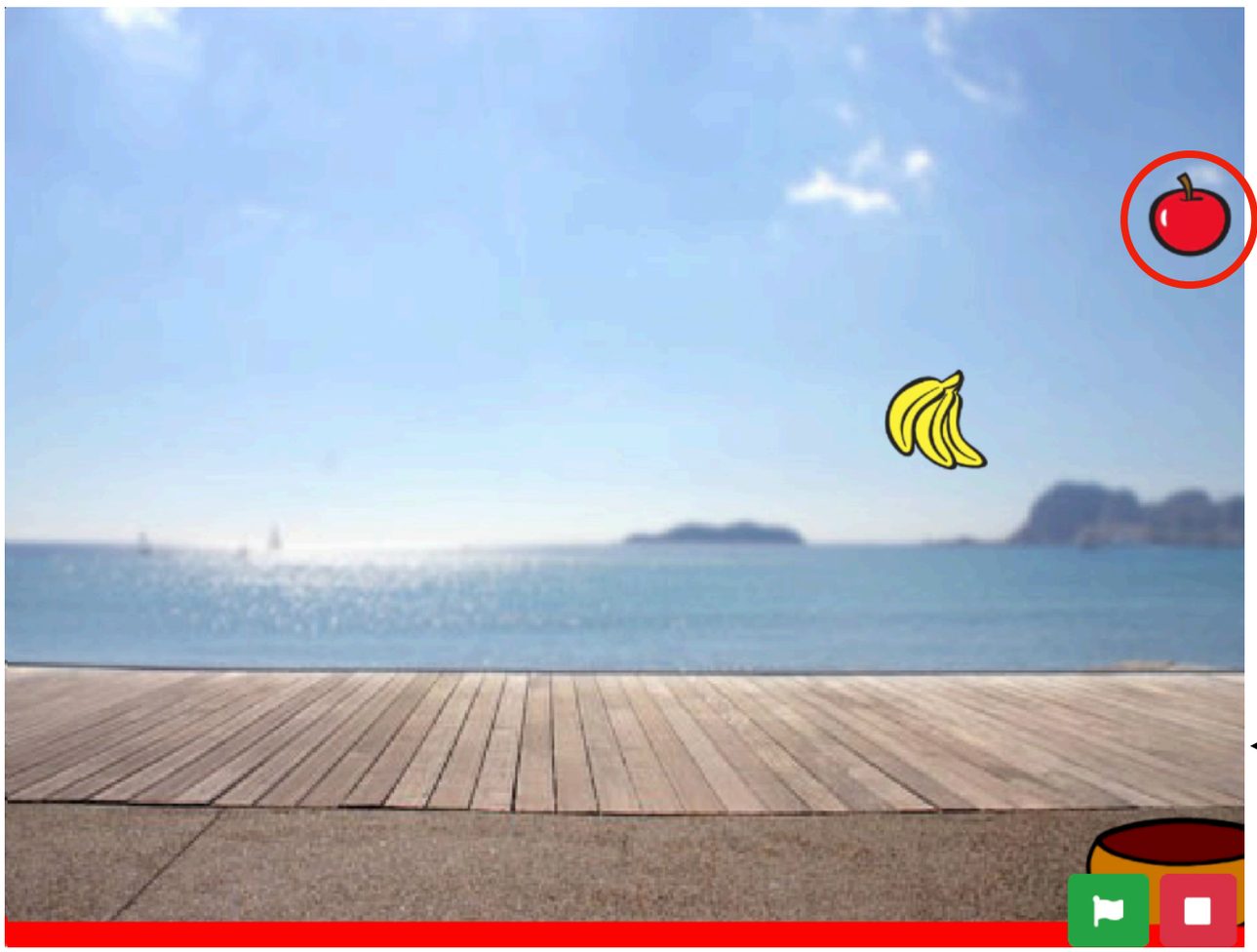


Generating Dynamic Test Suites

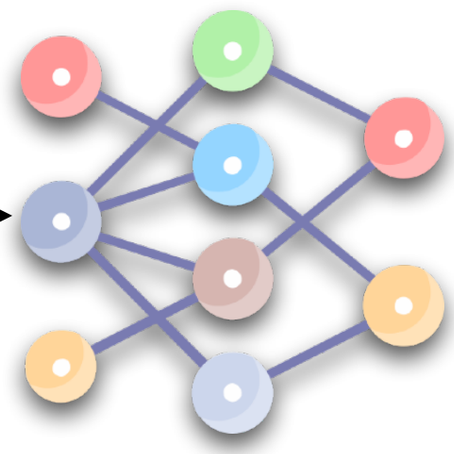
Validate and improve robustness of networks

➡ Cover 

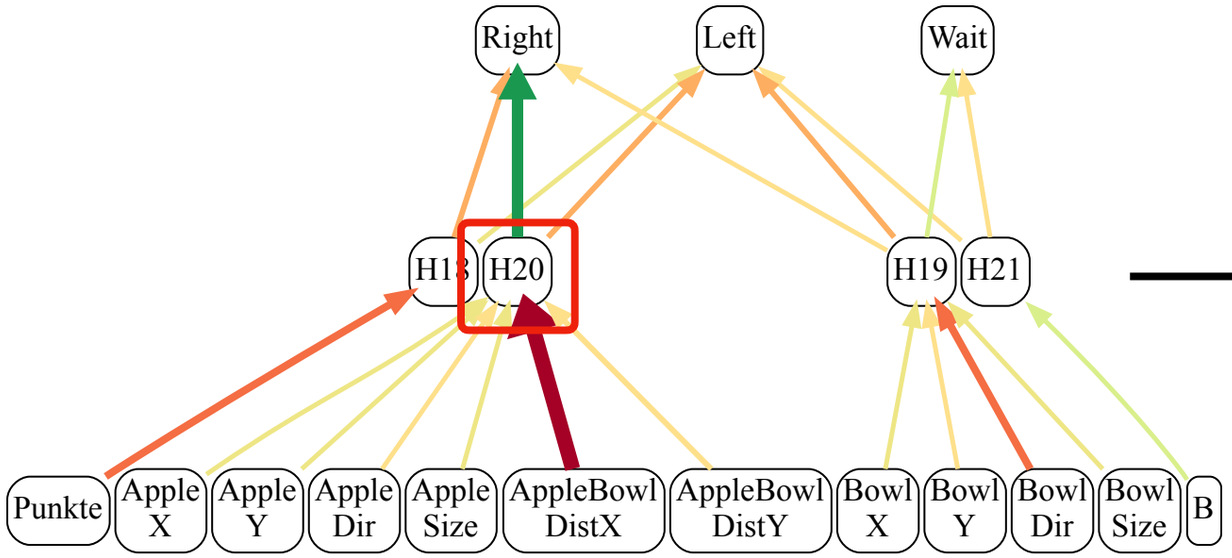
multiple times with different seeds

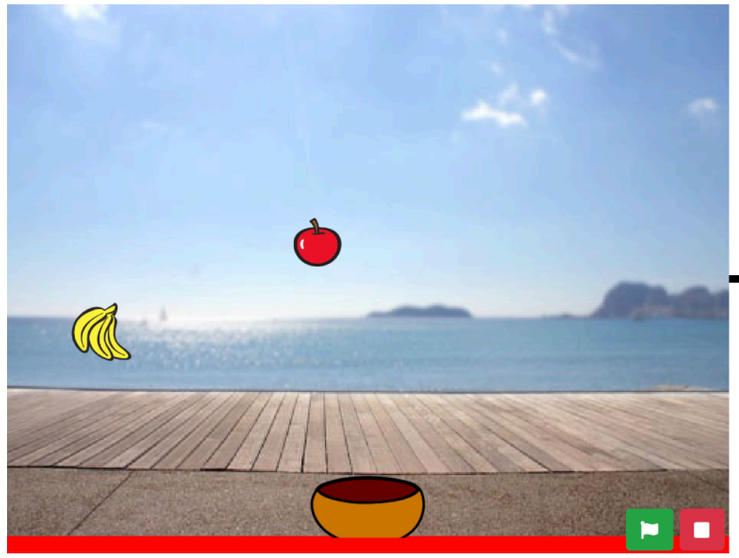


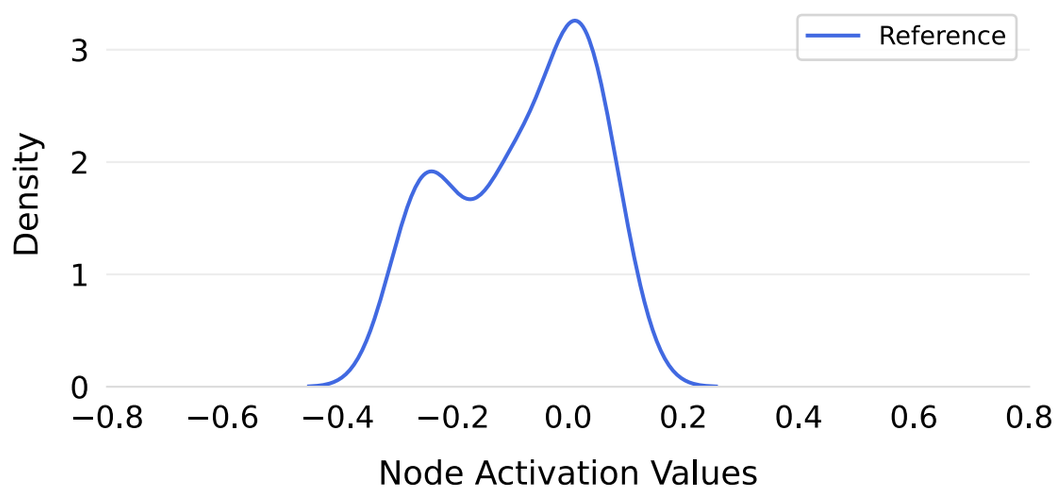
Neuroevolution



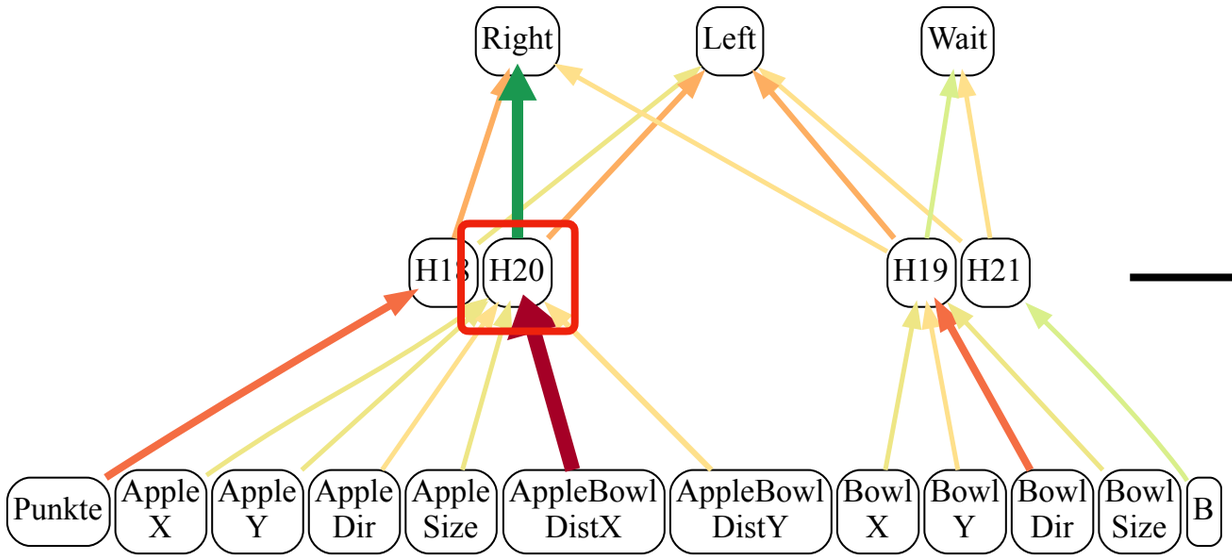
Test Oracle Based Surprise Adequacy

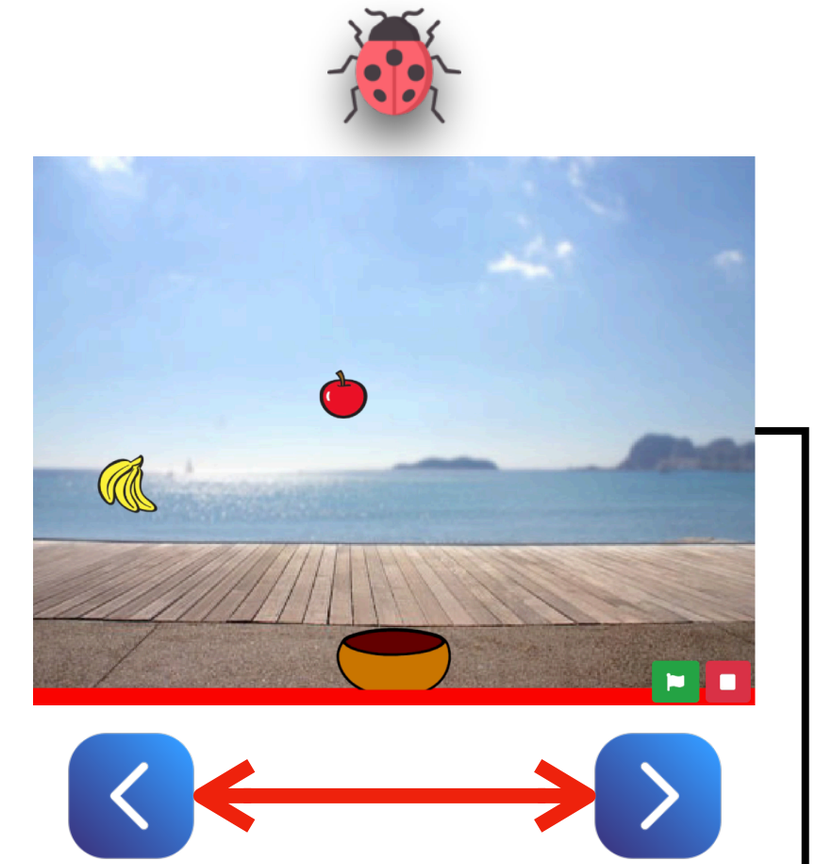


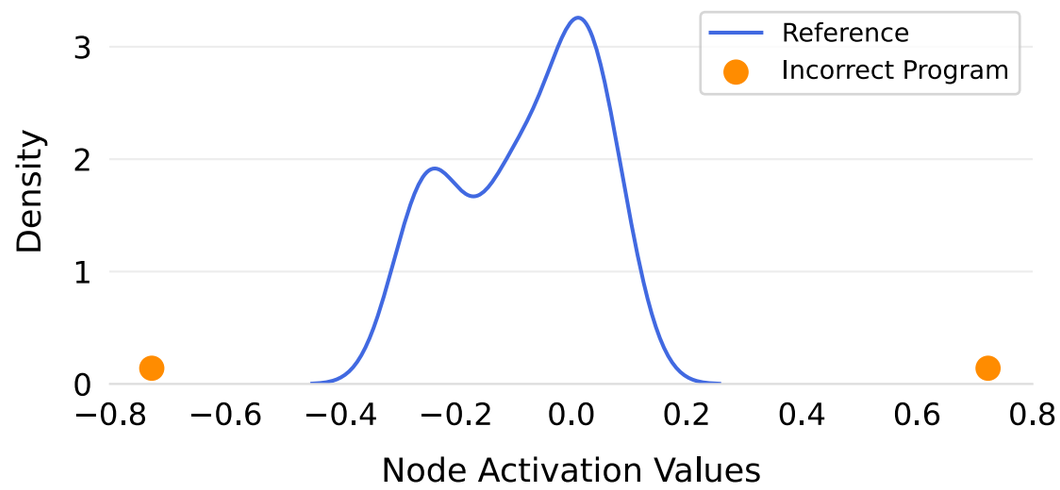




Test Oracle Based Surprise Adequacy





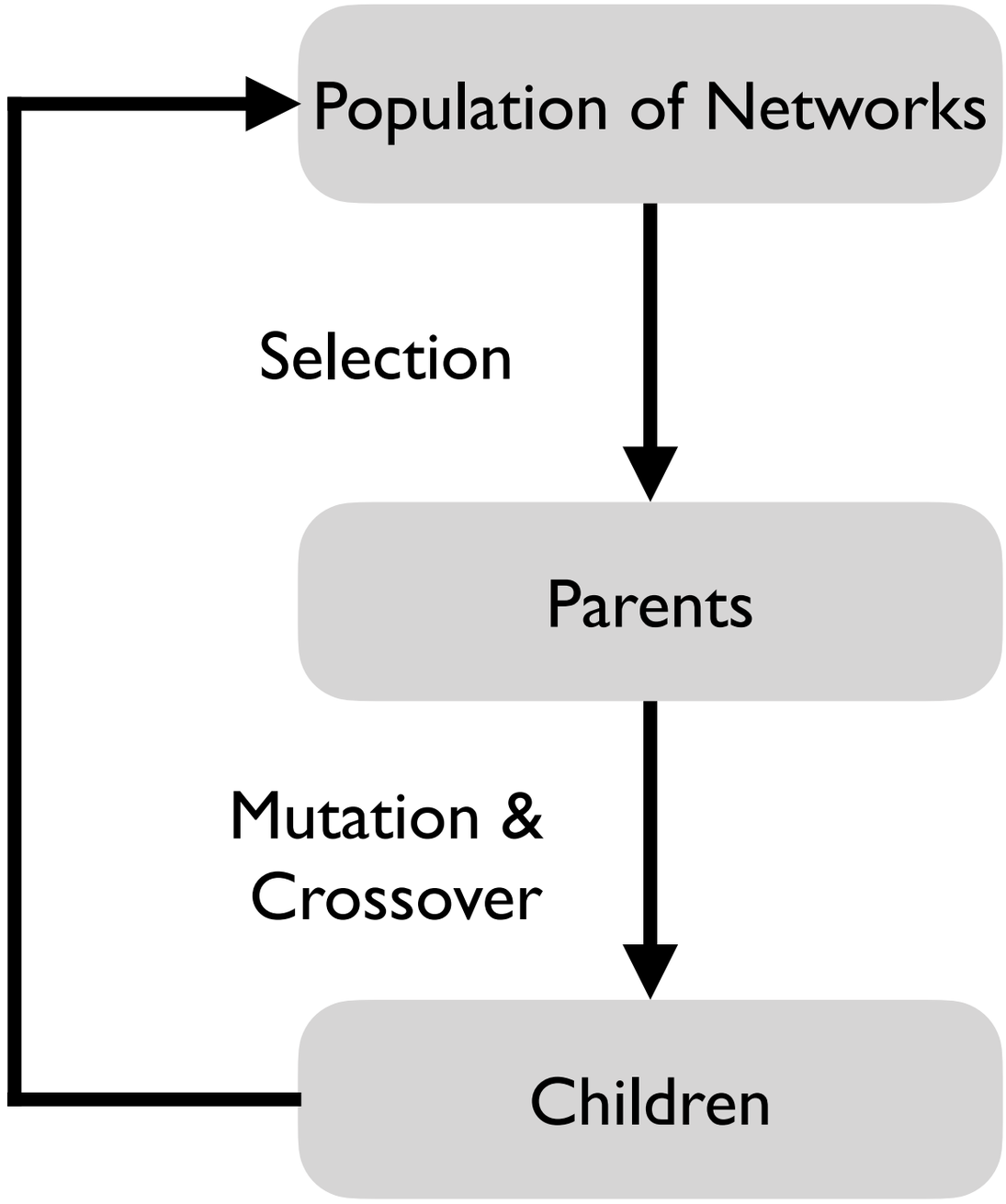


Neuro evolution

Neural networks

- Mimic a human brain to solve complex tasks

- Must be optimised for each task individually

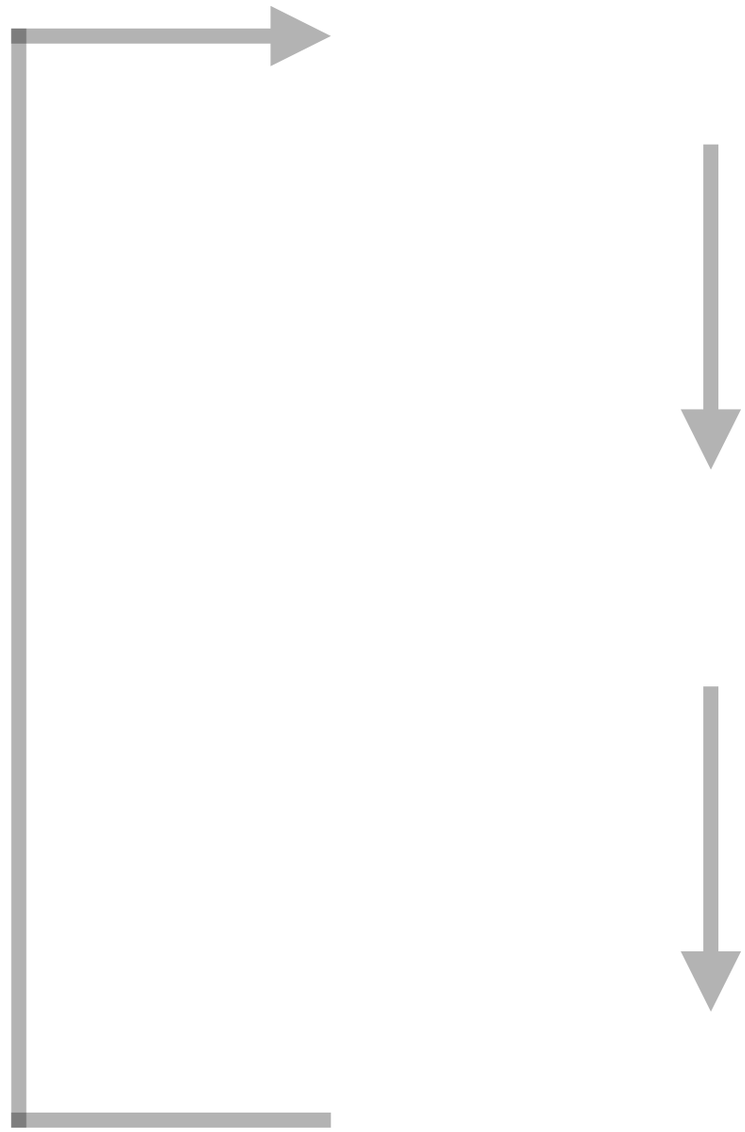


Neuro

Neural networks

- Mimic a human brain to solve complex tasks

- Must be optimised for each task individually



Neural Networks

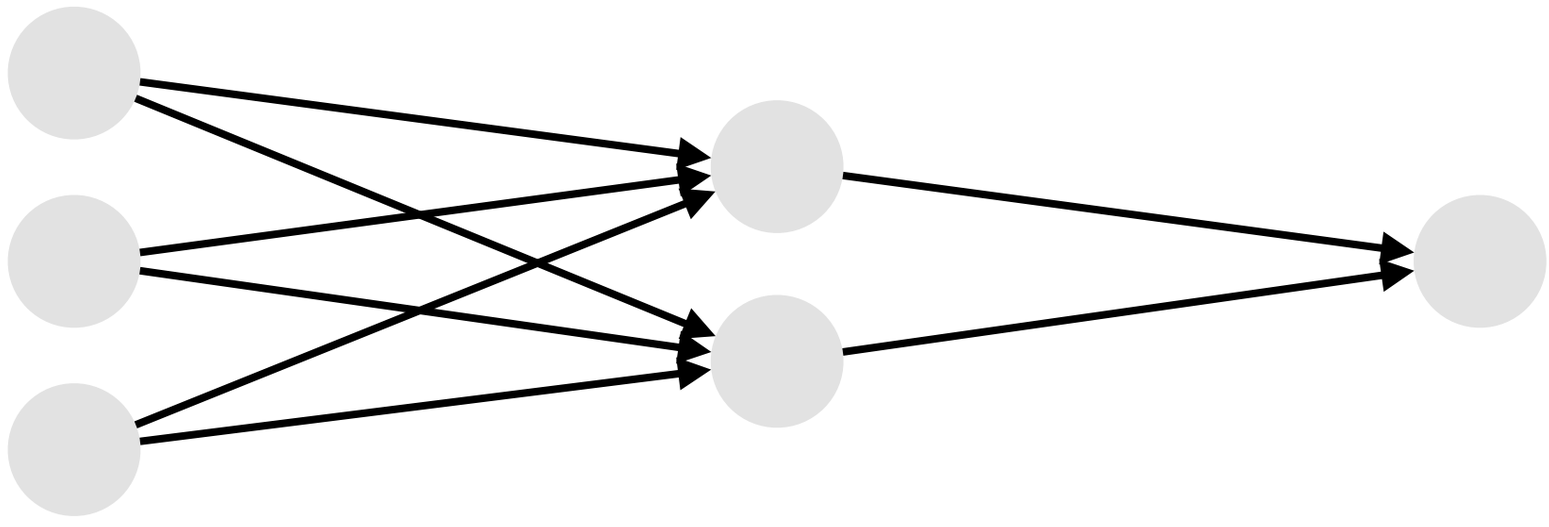
- Idea is to solve complex tasks by mimicking a human brain

Neural Networks

- Idea is to solve complex tasks by mimicking a human brain
- Neurons/Nodes represent brain cells

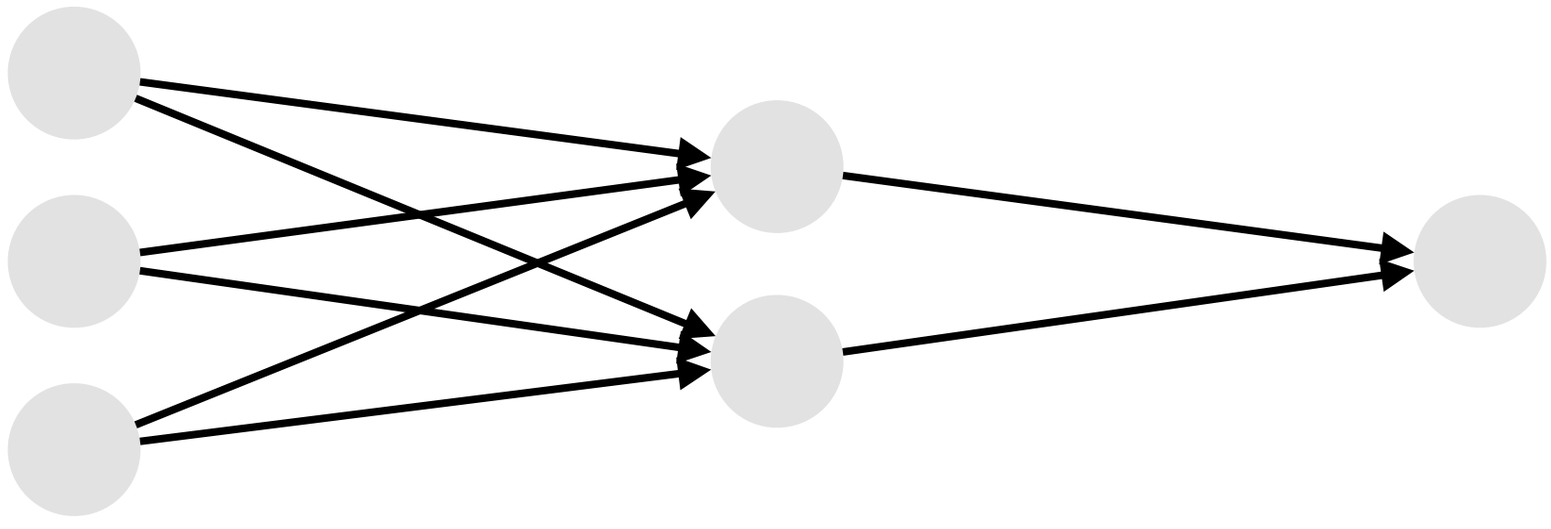
Neural Networks

- Idea is to solve complex tasks by mimicking a human brain
- Neurons/Nodes represent brain cells
- Weighted Connections link nodes with each other



Neural Networks

- Idea is to solve complex tasks by mimicking a human brain
- Neurons/Nodes represent brain cells
- Weighted Connections link nodes with each other
- Layers combine neurons into groups

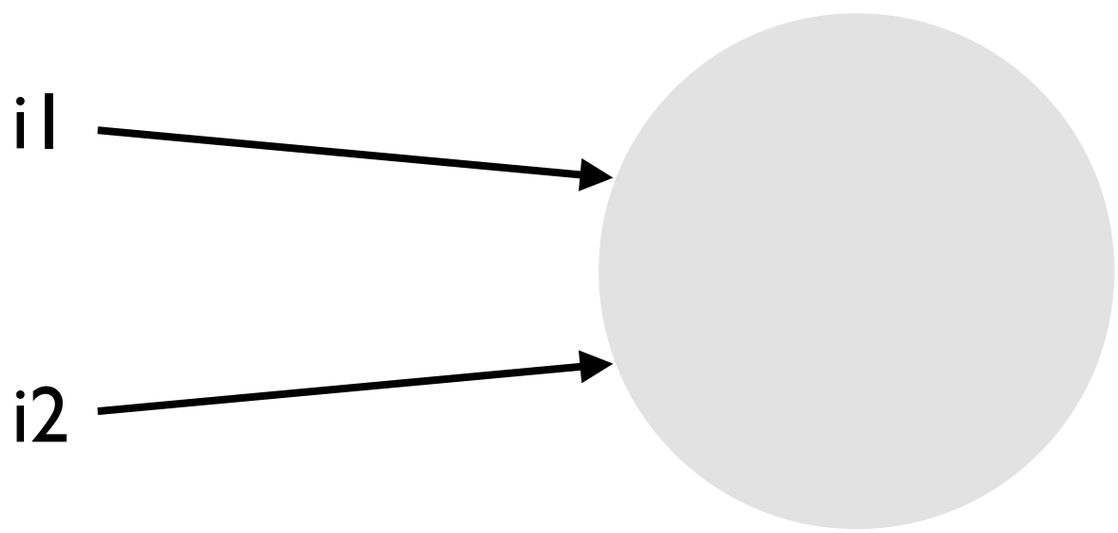


Neurons/Nodes

- Central processing units of neural networks

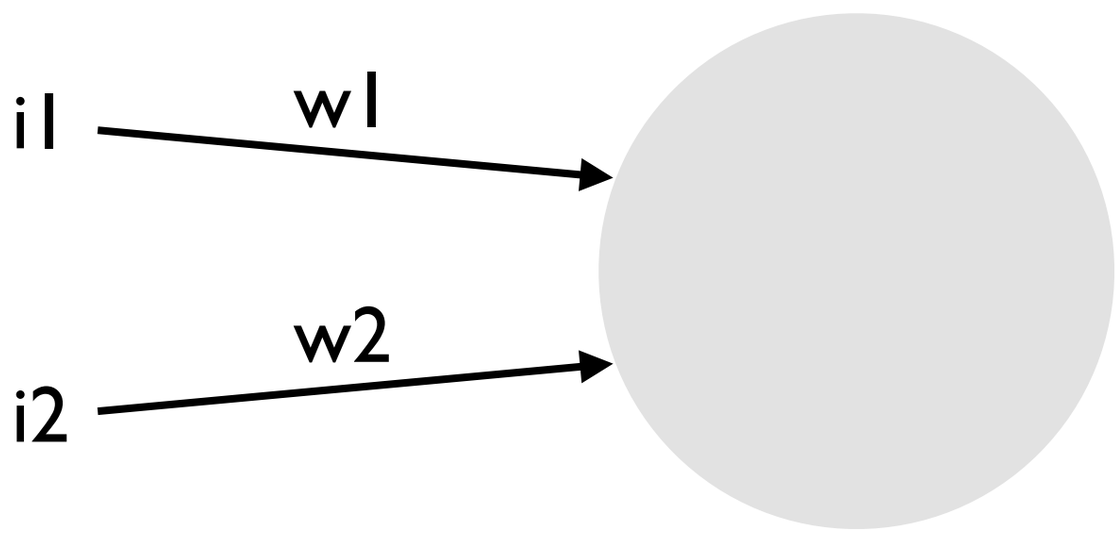
Neurons/Nodes

- Central processing units of neural networks
- Receive input i from previous nodes or an external source



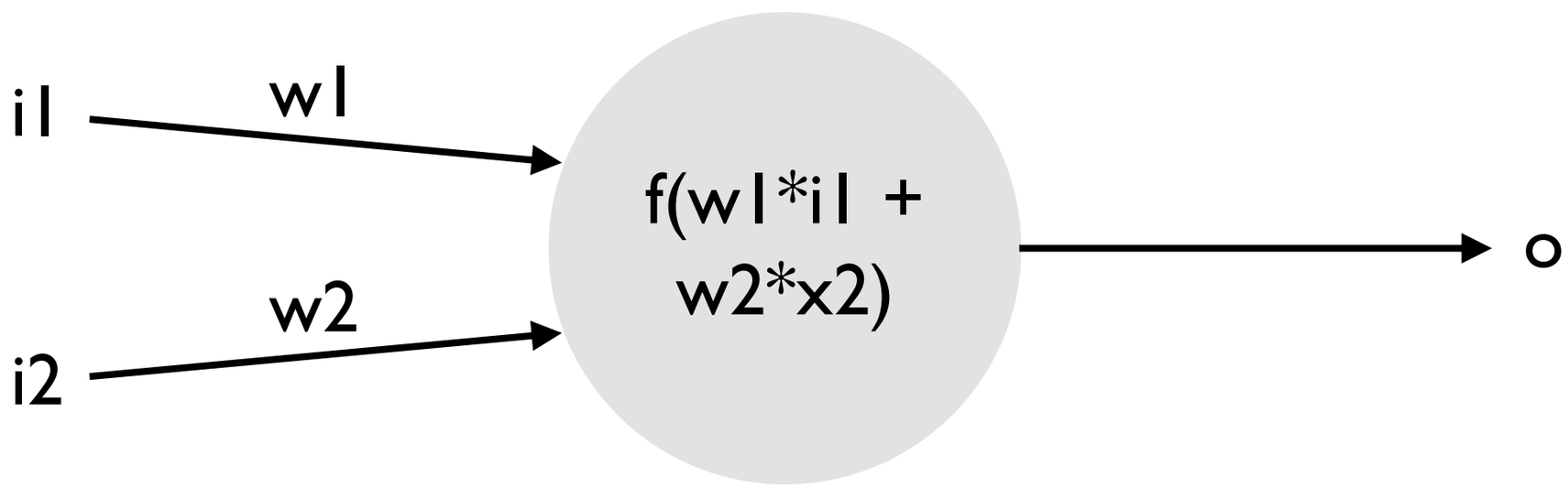
Neurons/Nodes

- Central processing units of neural networks
- Receive input i from previous nodes or an external source
- Each input is associated with a weight w



Neurons/Nodes

- Central processing units of neural networks
- Receive input i from previous nodes or an external source
- Each input is associated with a weight w
- An output o is produced by applying an activation function f to the weighted sum of the node's inputs.

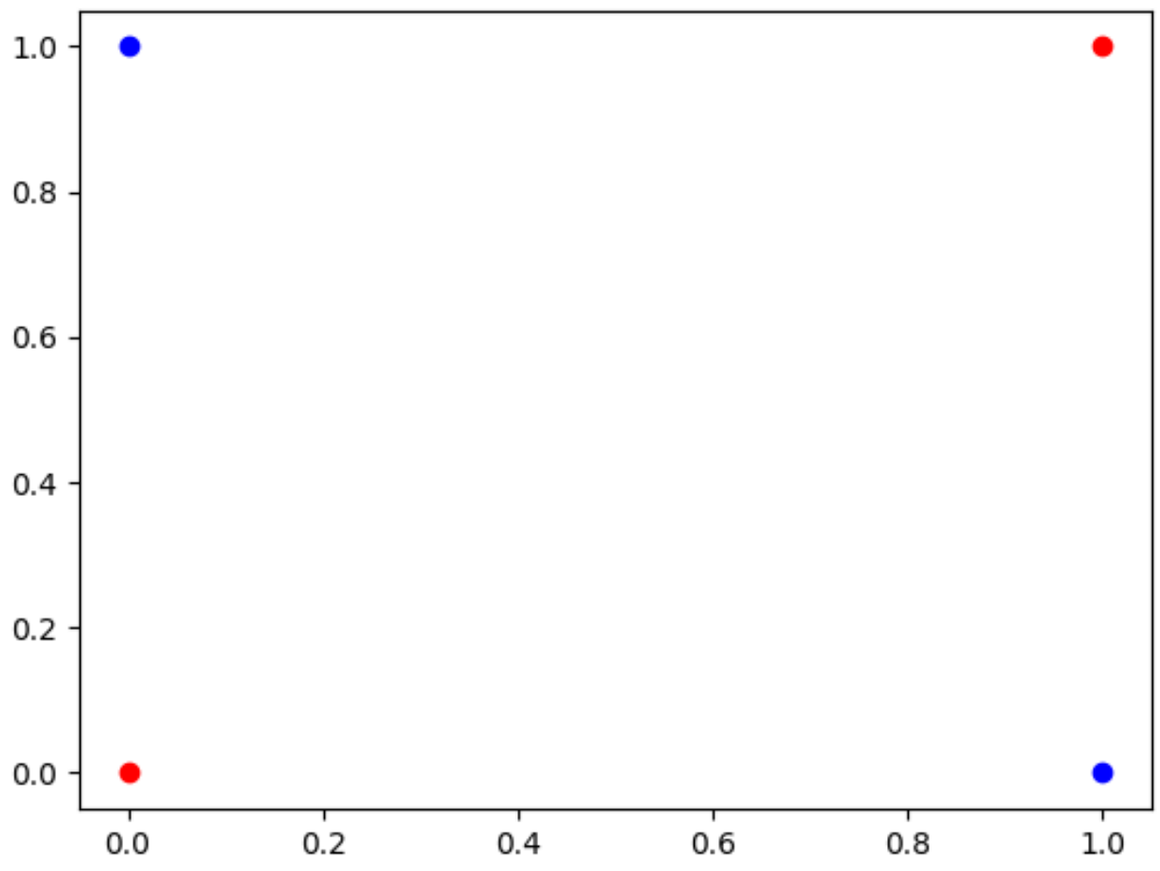


Activation Functions

- Neurons should learn representations of the input data
- To represent non-linear real-world data, we apply non-linear activation functions.

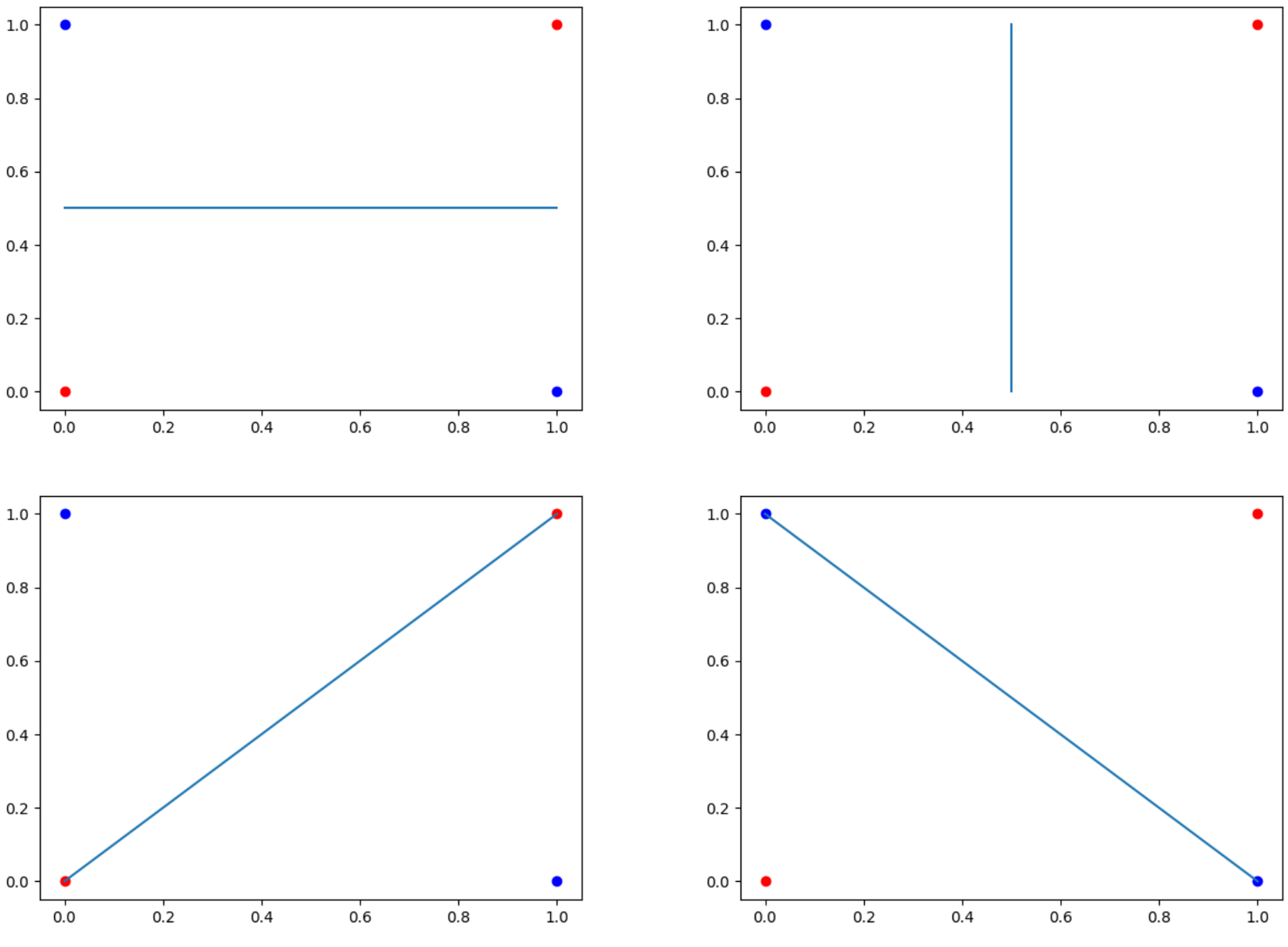
Activation Functions

Non-linearity of XOR



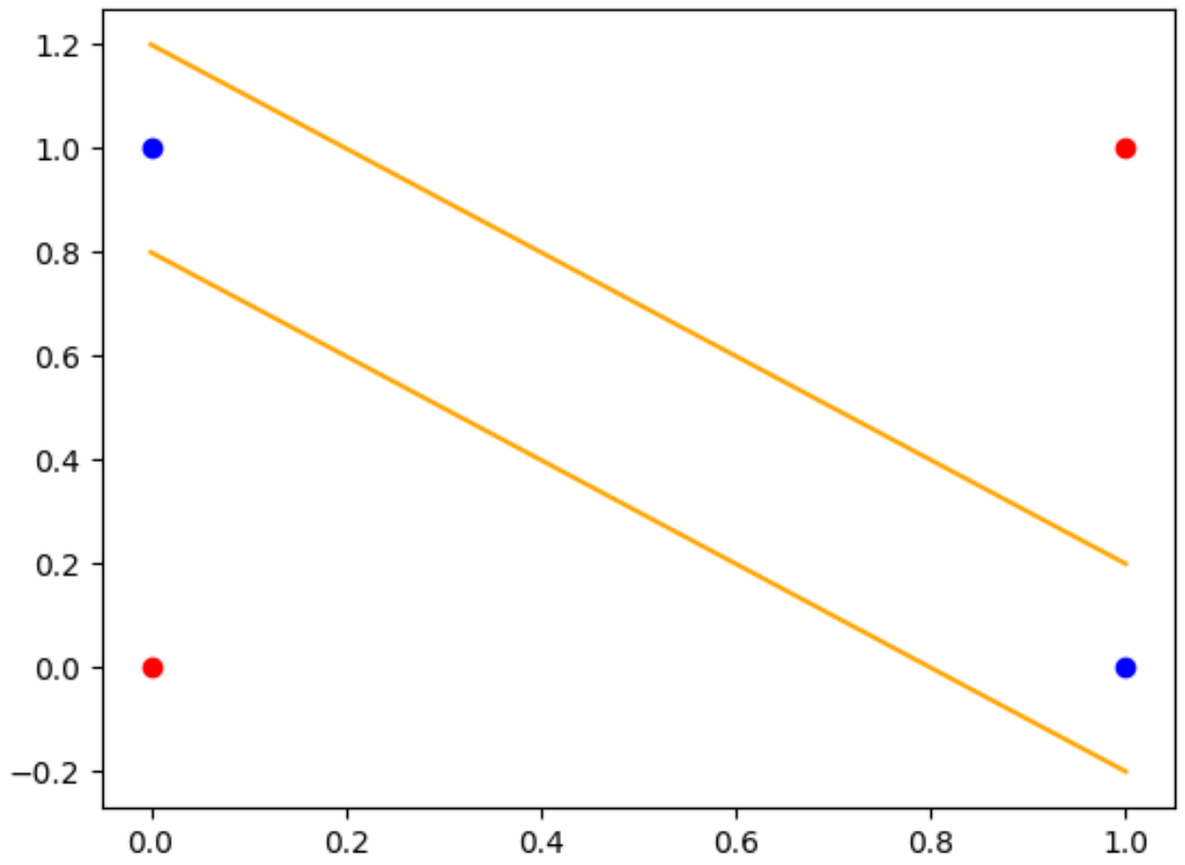
Activation Functions

Non-linearity of XOR



Activation Functions

Non-linearity of XOR

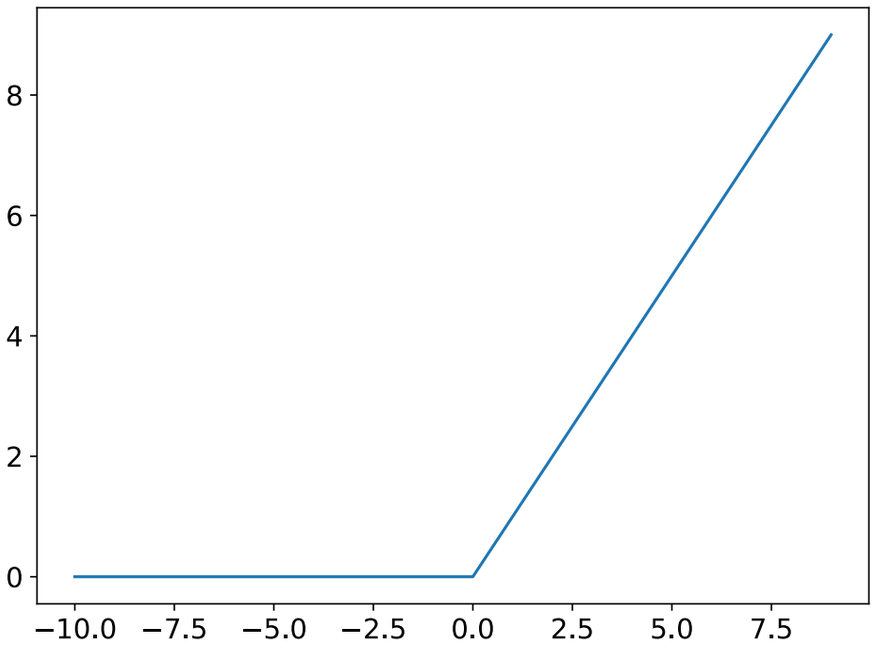


Activation Functions

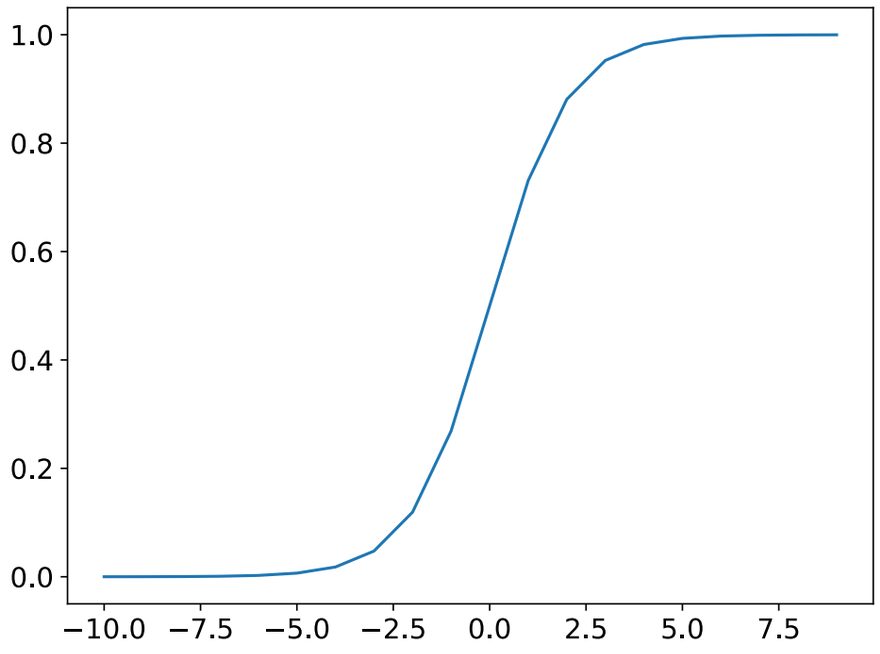
- Neurons should learn representations of the input data
- To represent non-linear real-world data, we apply non-linear activation functions.
- Activation functions take a single parameter and apply a fixed non-linear operation
- Usually we do not apply activation functions at input nodes as we are interested in the raw input signal

Hidden Layer Activation Functions

Relu: $f(x) = max(0,x)$



Sigmoid: $f(x) = \frac{1}{1 + e^{-x}}$



Output Layer Activation Functions

- Selected based on the task that should be solved
- Regression problem
- Classification problem
  - ➡ Binary classification
  - ➡ Multi-class classification
  - ➡ Multi-label classification

Output Layer Activation Functions

- Regression tasks predict continuous values
  - ➡ Prediction of house prises
  - ➡ Prediction of temperature
  - ➡ Estimated duration of a software project
- Linear activation function with single output node
  - ➡ $f(x) = x$

Output Layer Activation Functions

- Binary classification tasks assign a single label out of two mutually exclusive classes to an input
  - ➡ Positive/Negative movie review
  - ➡ Will it rain tomorrow?
  - ➡ Is there a cat or a dog in the picture?
- Sigmoid activation function and a single output node
  - ➡ $$f(x) = \frac{1}{1 + e^{-x}}$$

Output Layer Activation Functions

- Multi-class classification tasks assign a single label out of many mutually exclusive classes to an input

  ➡ Assign genres to movies

  ➡ Recognise handwritten digits

  ➡ Is there a dog or a cat or a horse in the picture?

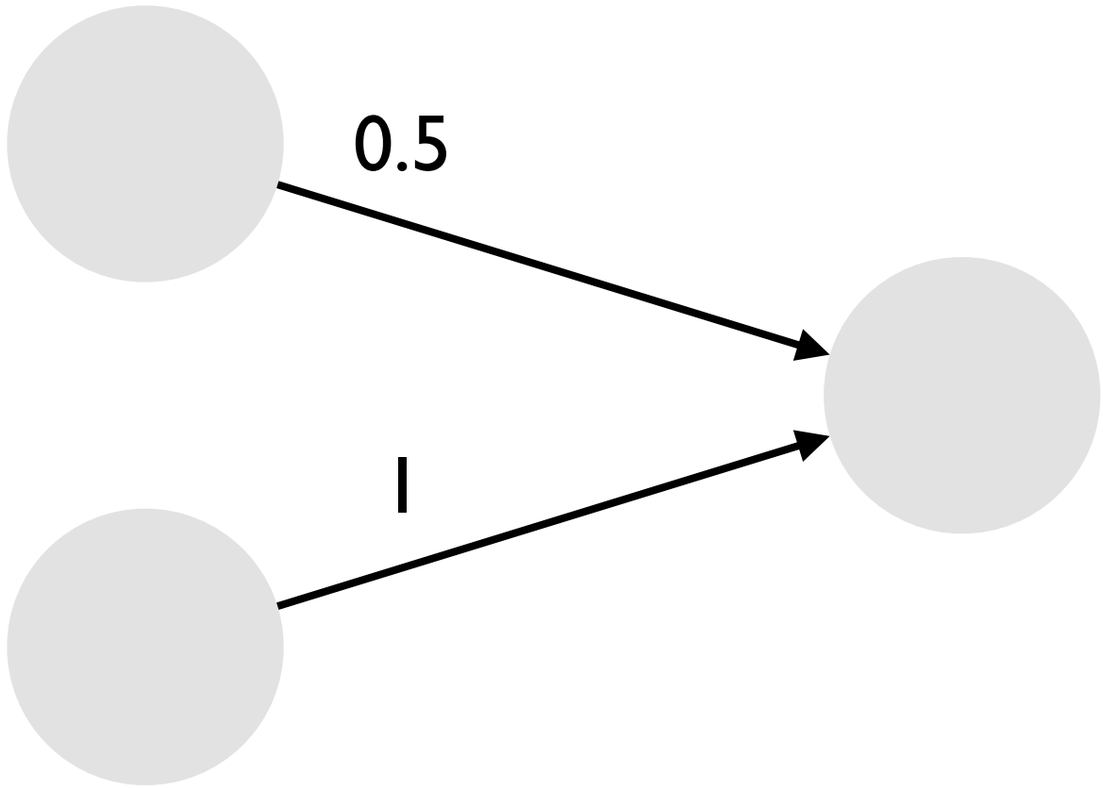
- Softmax activation function and one output node per class

  ➡ $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$ with x representing the vector of node values over all output nodes N

Output Layer Activation Functions

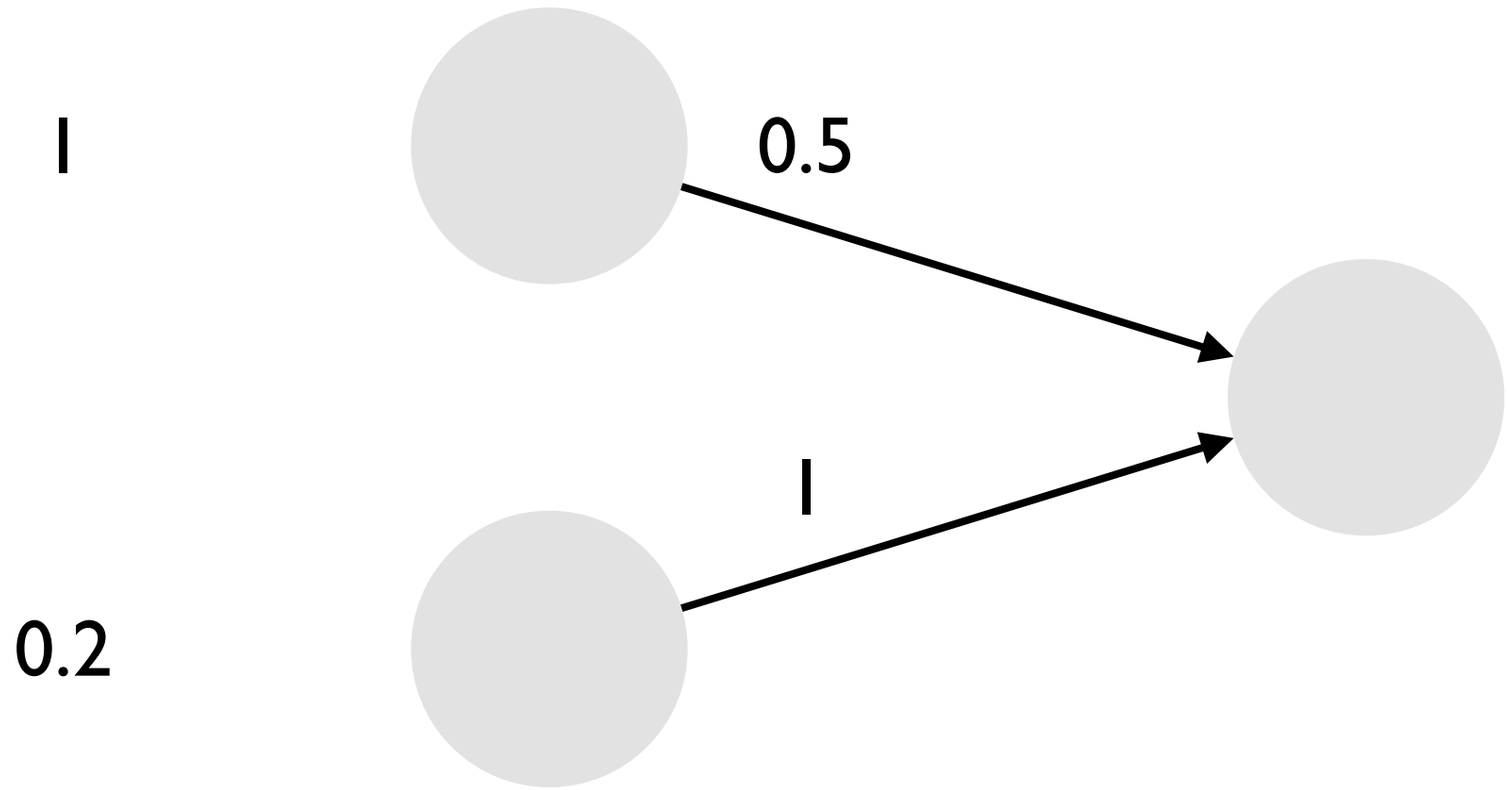
- Multi-label classification tasks assign multiple label out of two or more inclusive classes to an input
  - ➡ Predict authors of an article
  - ➡ Predict weather conditions
  - ➡ Recognise animals in a picture
- Sigmoid activation function and one output node per class
  - ➡ $$f(x) = \frac{1}{1 + e^{-x}}$$

Example of Network Activation



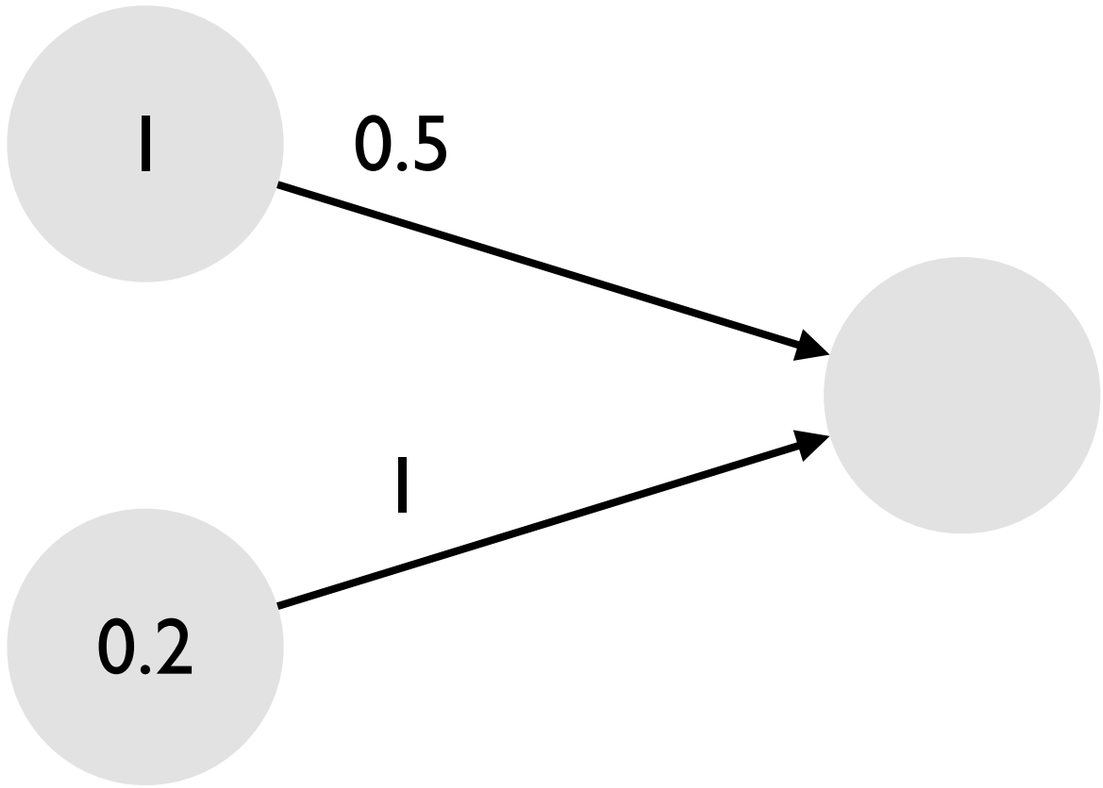
Example of Network Activation

Load inputs into the input layer



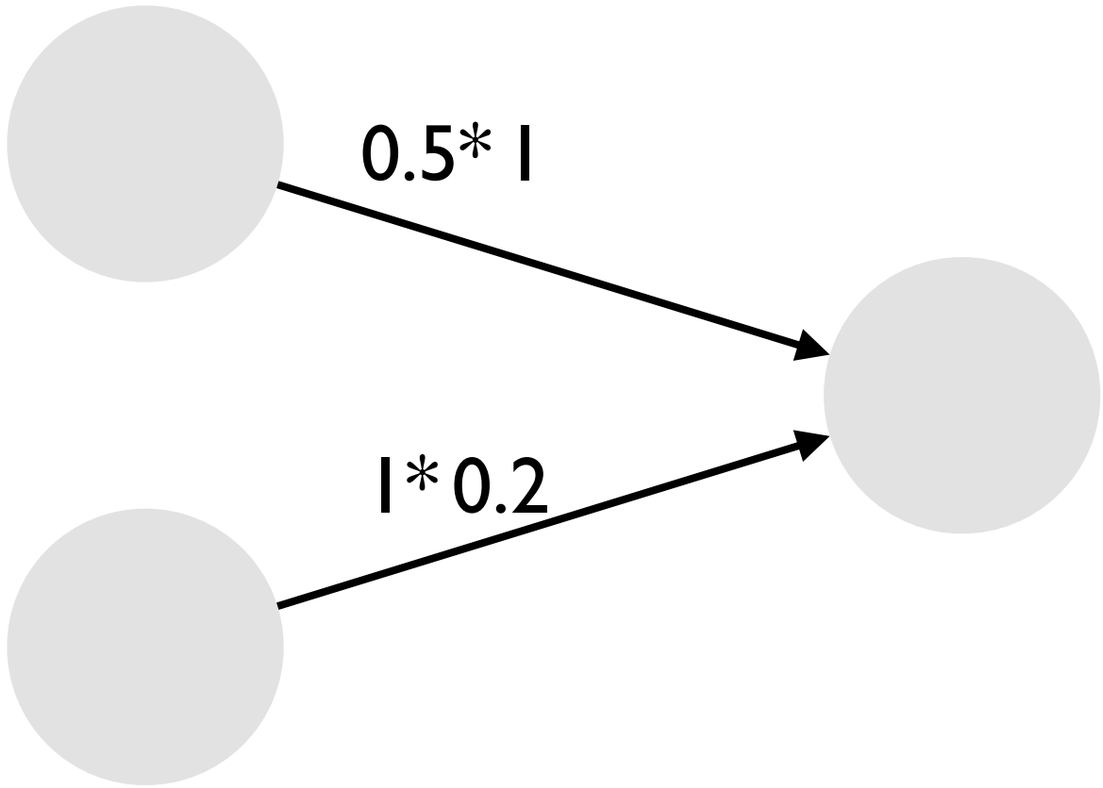
Example of Network Activation

Load inputs into the input layer



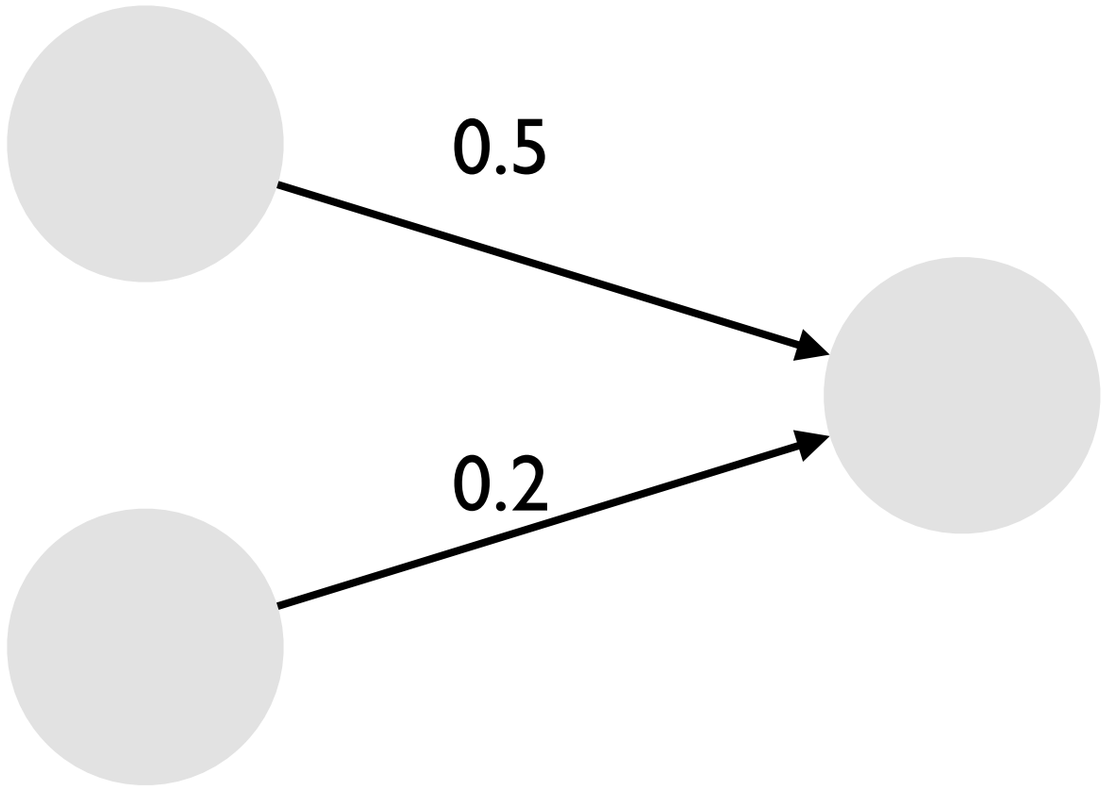
Example of Network Activation

Propagate inputs and calculate the weighted sum



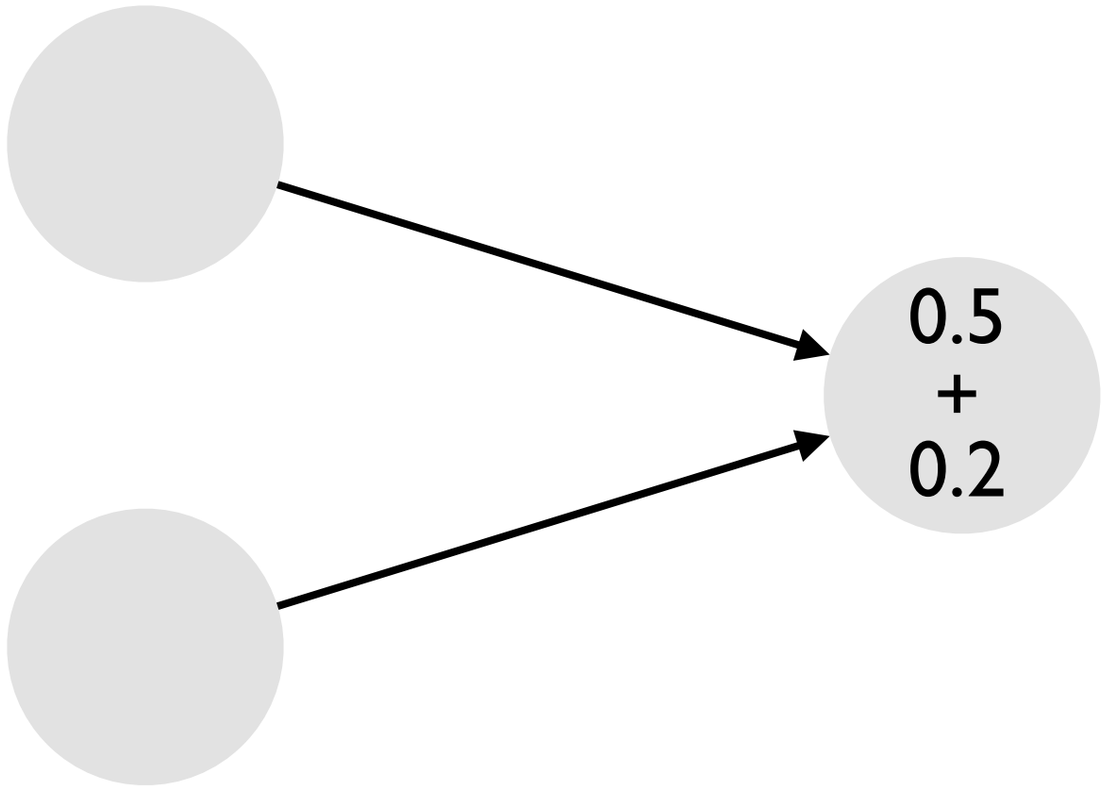
Example of Network Activation

Propagate inputs and calculate the weighted sum



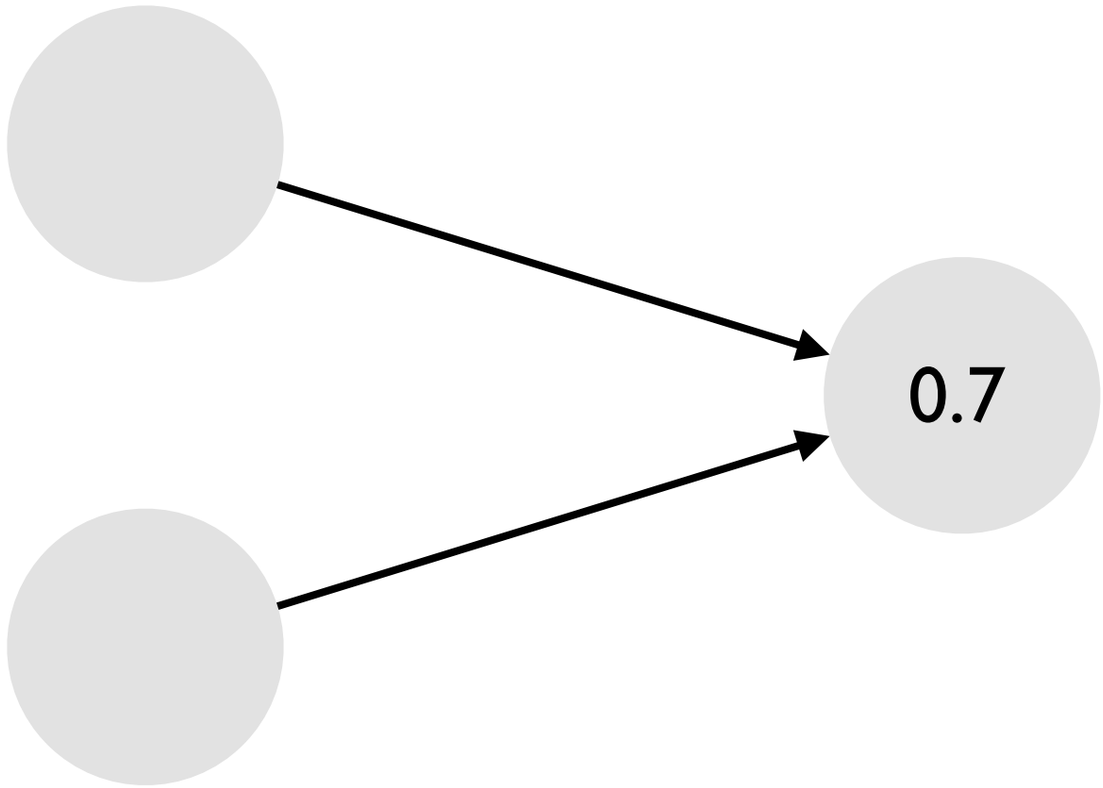
Example of Network Activation

Propagate inputs and calculate the weighted sum



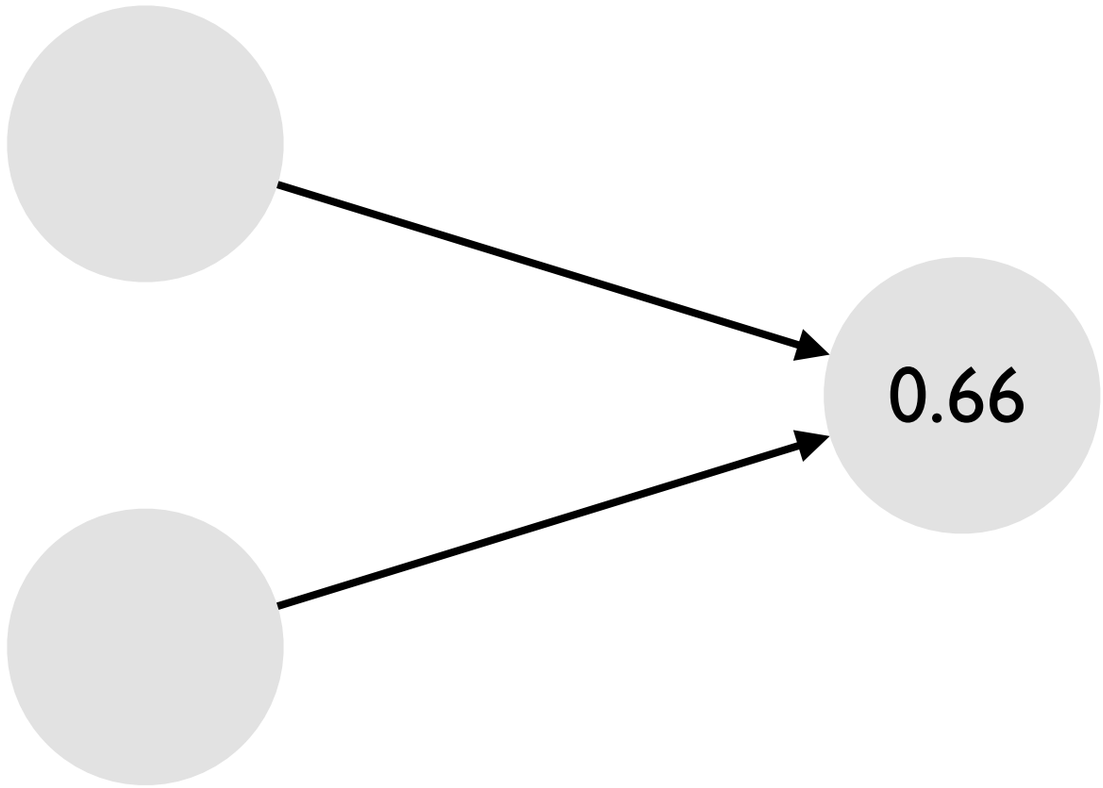
Example of Network Activation

Propagate inputs and calculate the weighted sum



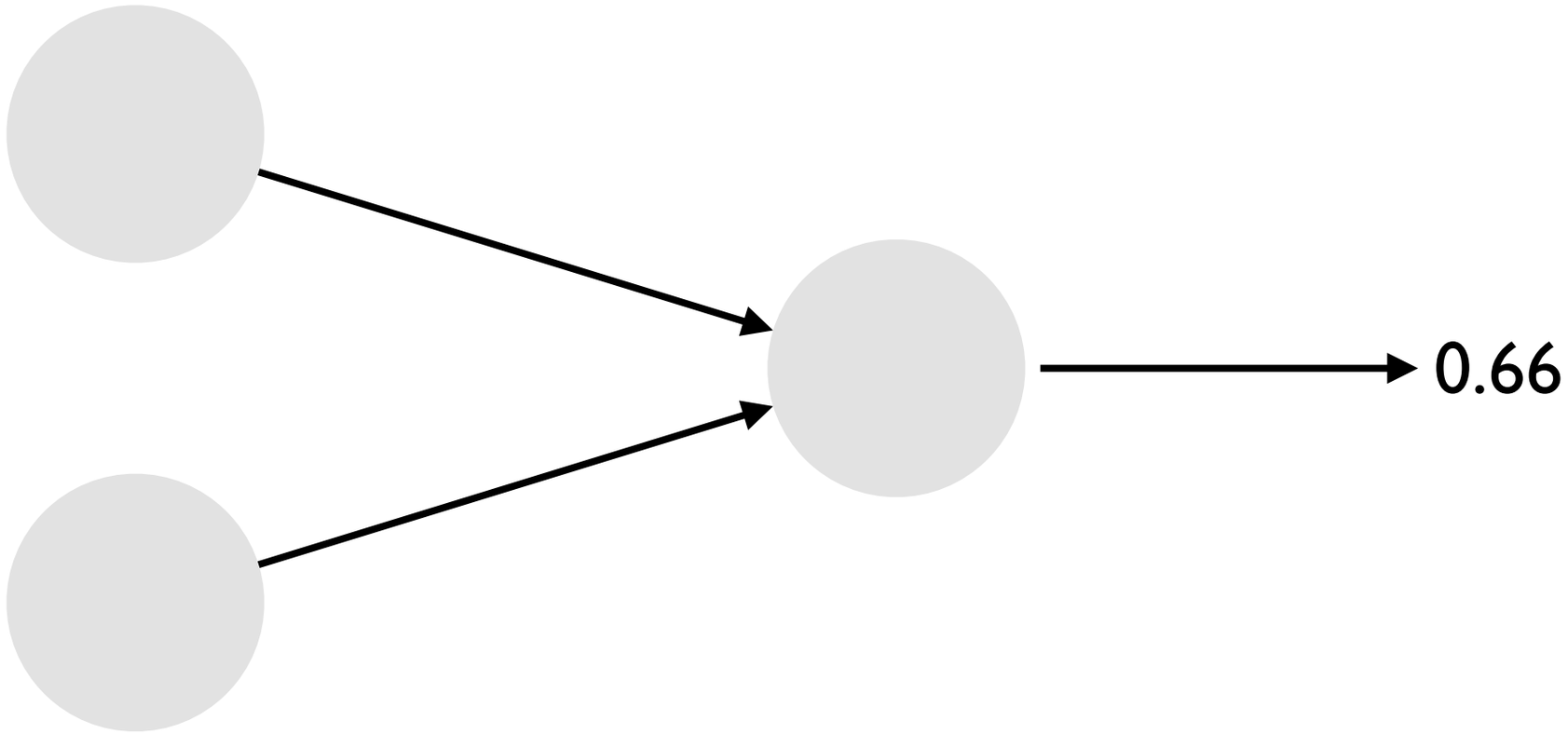
Example of Network Activation

Apply activation function



Example of Network Activation

Propagate neuron output deeper into the network

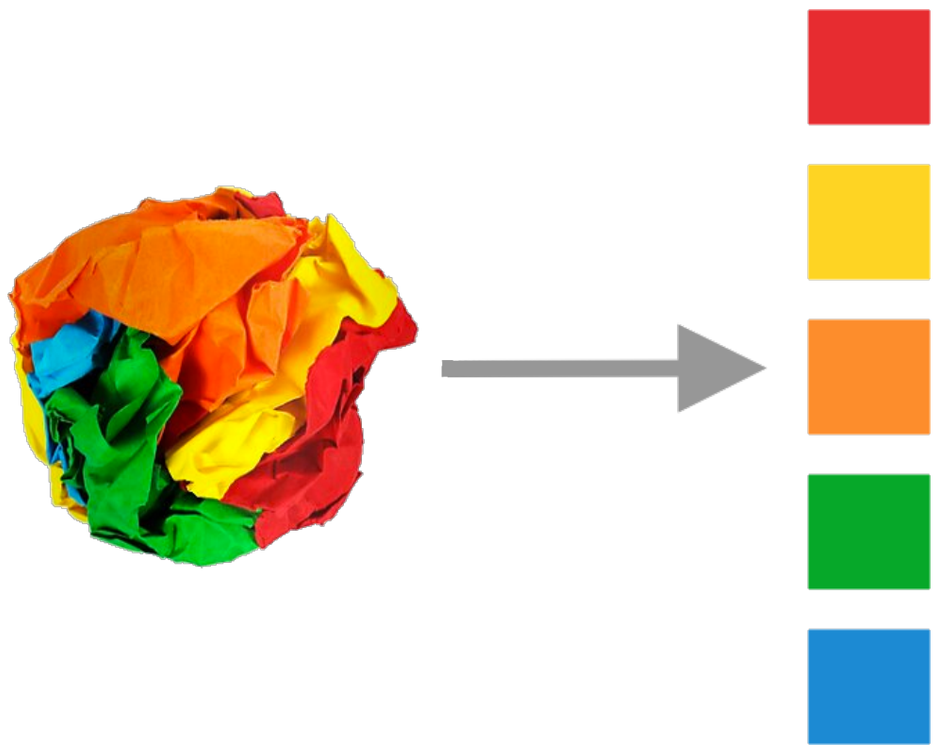


Neural Network Interpretations

Neural Network Interpretations

Geometric interpretation

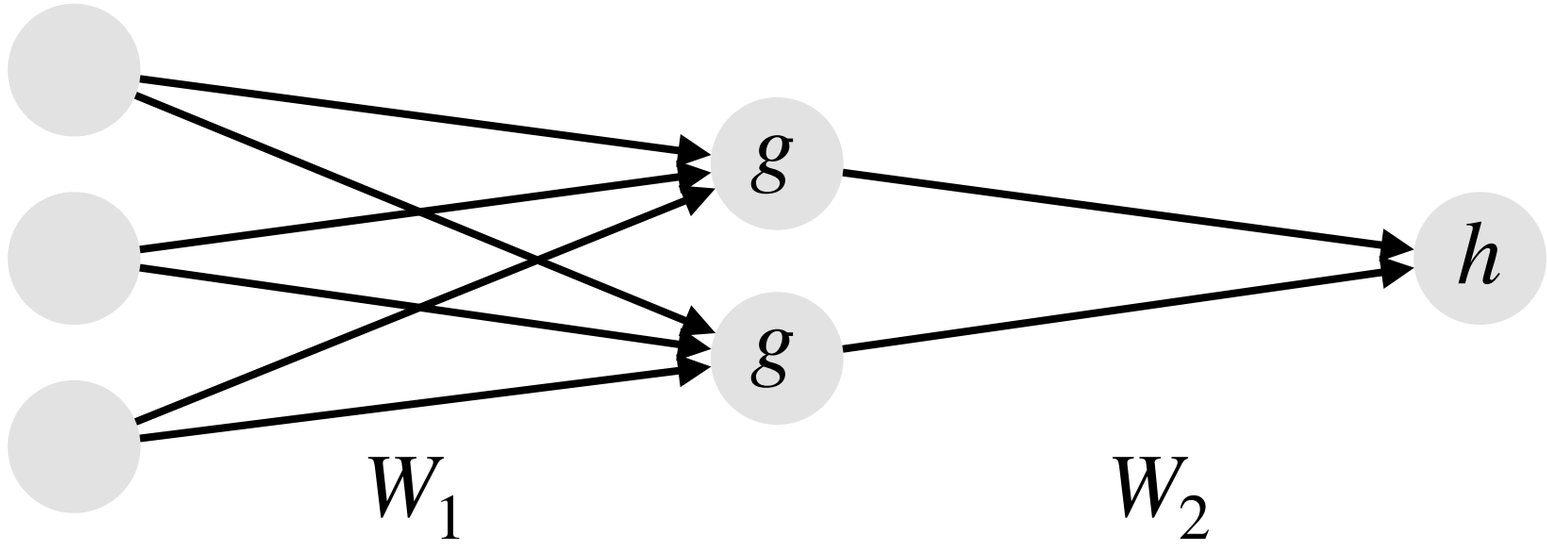
- Chains of tensor operations
- Geometric transformations of input data



Neural Network Interpretations

Function approximators

$$\hat{f}(x, W) \approx f(x)$$



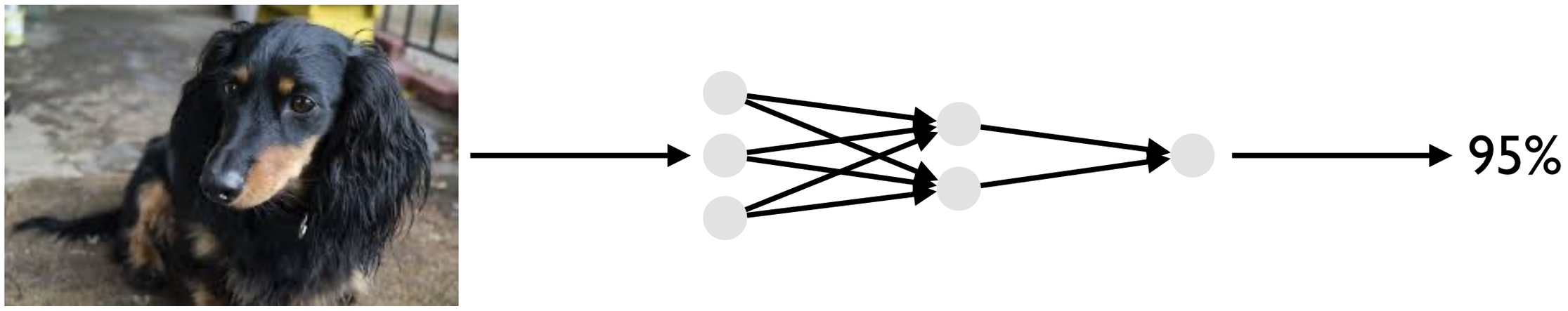
Input Layer | Hidden Layer | Output Layer

x | $g(x, W_1)$ | $h(g(x, W_1), W_2) = \hat{f}(x, W)$

Neural Network Interpretations

Probability distribution approximators

Is a dog in the image? $\Rightarrow y$



Input	Conditional Probability	Interpretation
x	$P(y \mid x)$	$y = True$

No Free Lunch Theorem

If an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems.

Wolpert & Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation, 1997.

No Free Lunch Theorem

If an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems.

Wolpert & Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation, 1997.

➡ We need to optimise neural networks for each task individually

Neural Network Optimisation

- Hyperparameter optimisation:
  - ➡ Network architecture (number of layers and nodes)
  - ➡ Activation functions
  - ➡ Learning method & parameter
- Parameter optimisation:
  - ➡ Optimisation of connection weights

Hyperparameter Optimisation

- Based on experience as well as trial and error
- Problem: tedious, time-intensive and often manual task
- Solution: train parameters and hyper parameters jointly

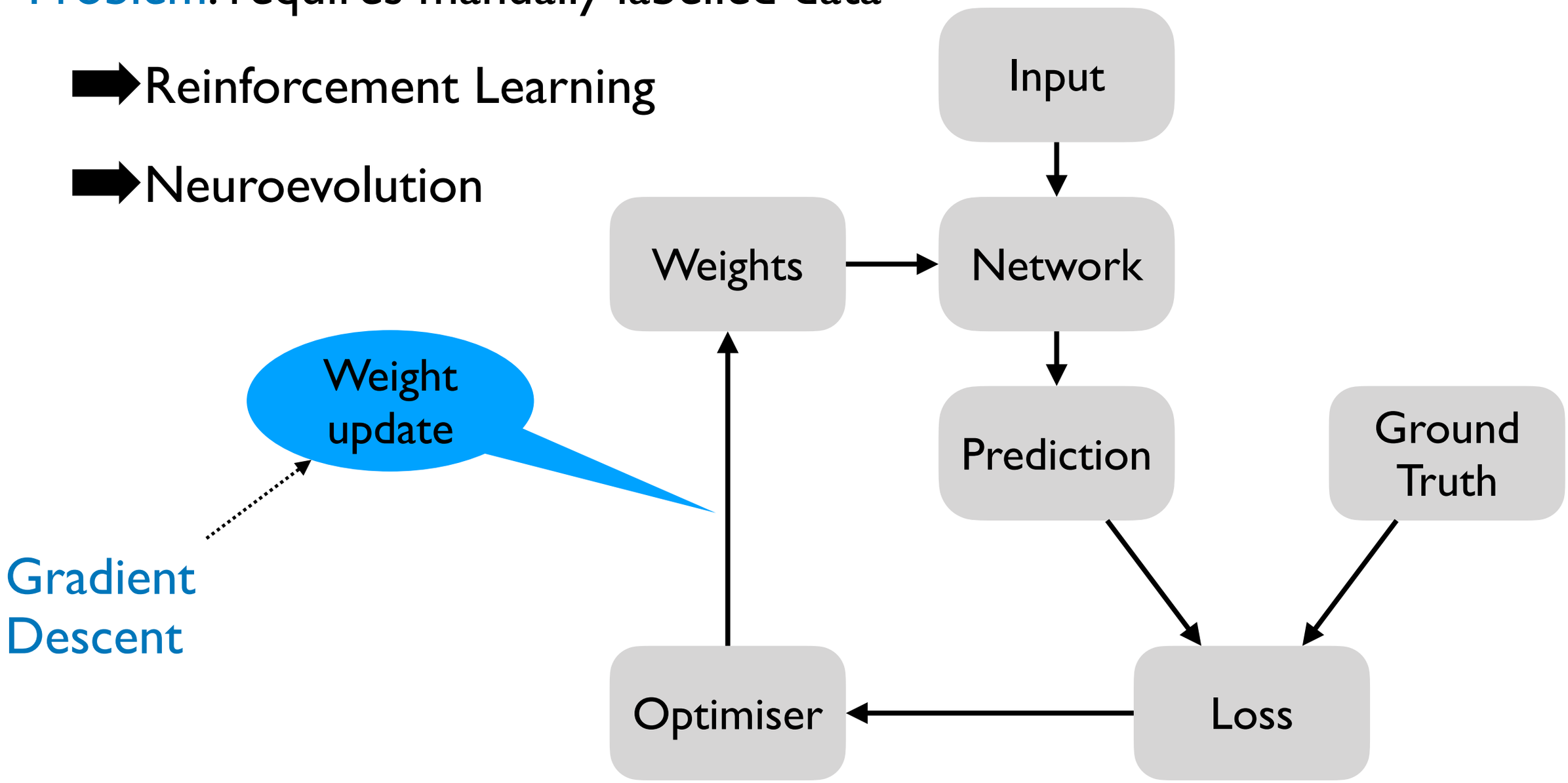
➡ Neuroevolution

Supervised Parameter Optimisation

Problem: requires manually labelled data

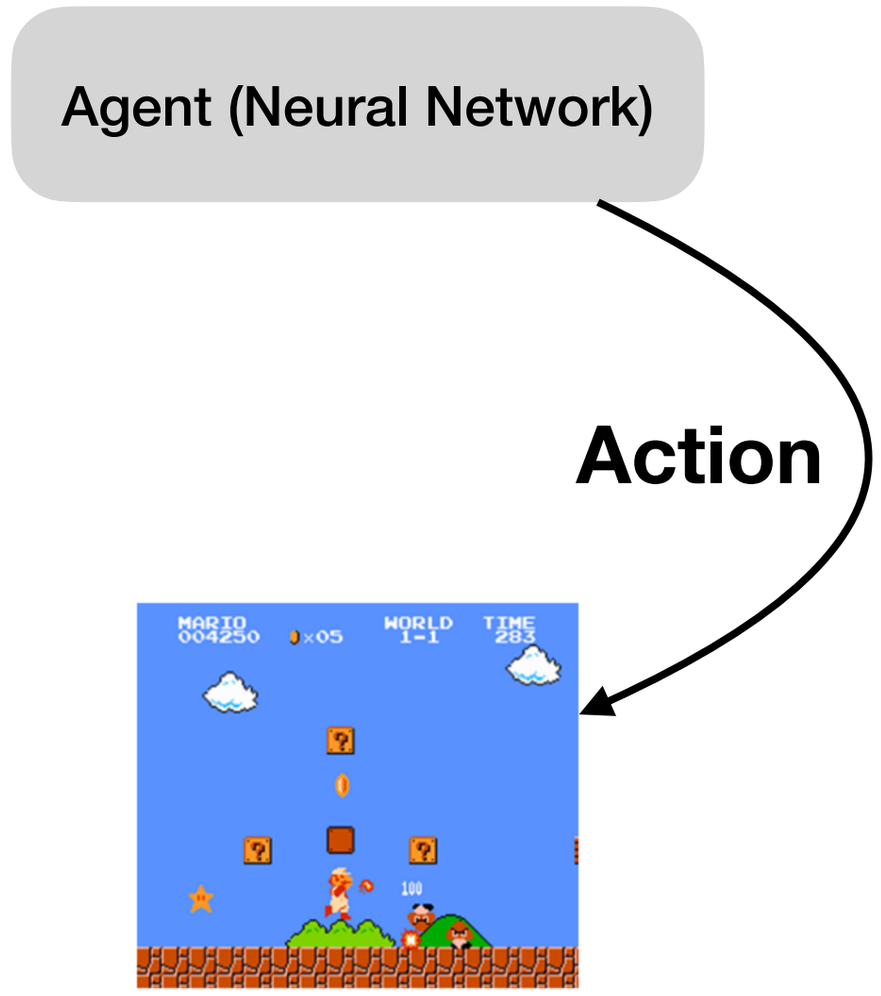
➡ Reinforcement Learning

➡ Neuroevolution



Reinforcement Learning

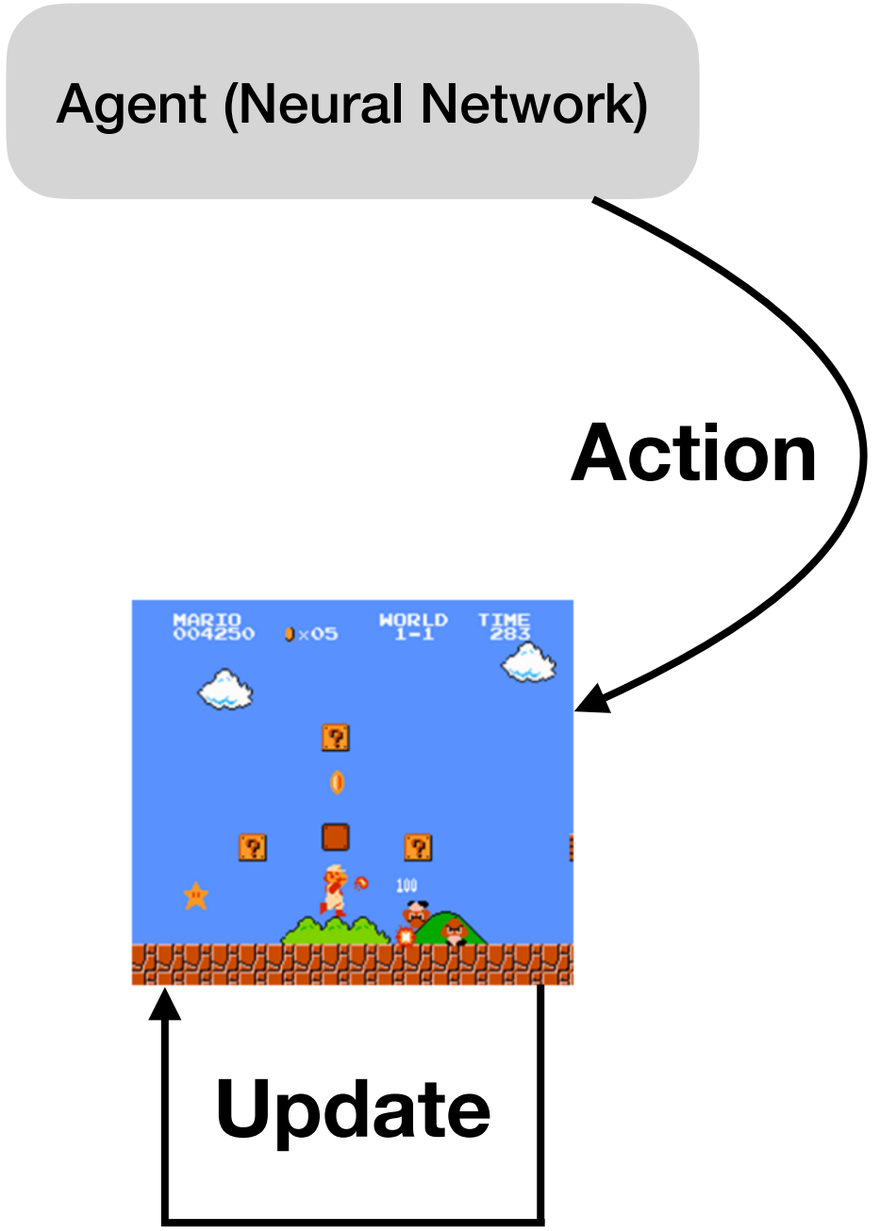
- An agent is placed into an environment in which, it can perform certain actions.



Reinforcement Learning

- An agent is placed into an environment in which, it can perform certain actions.

- Based on the selected action, the environment is updated.



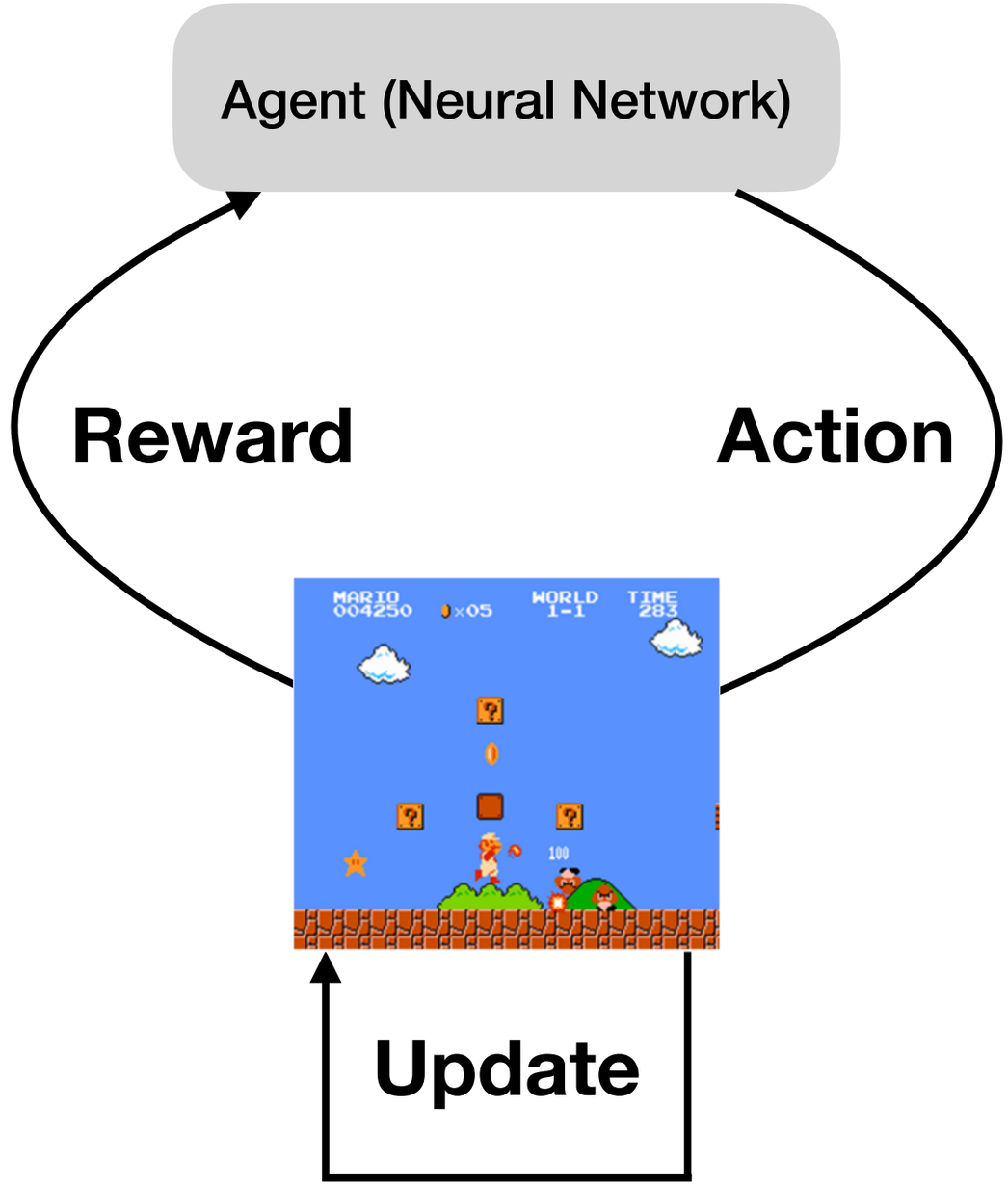
Reinforcement Learning

- An agent is placed into an environment in which, it can perform certain actions.

- Based on the selected action, the environment is updated.

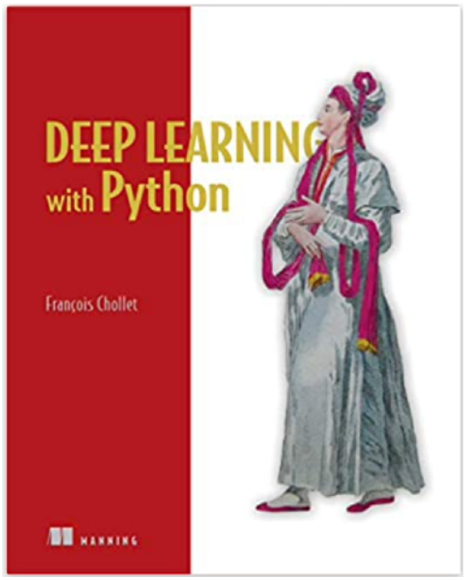
- The agent is assigned a reward (fitness value) based on the reached state after applying one or several actions.

➡ Has to represent the intended goal as closely as possible!

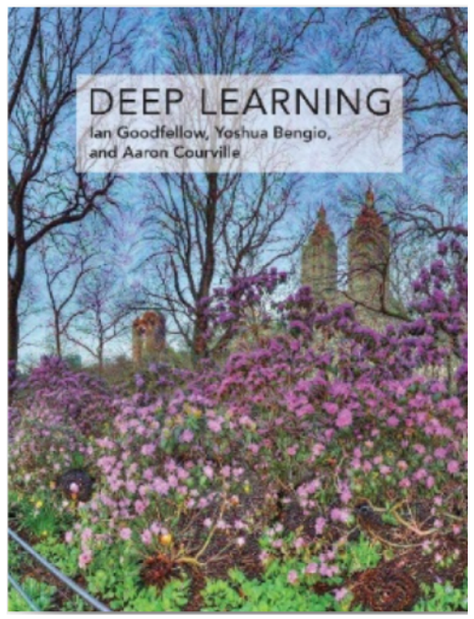


More on Neural Networks

- Deep Learning with Python
  - ➡ François Chollet
  - ➡ Beginner level, hands-on



- Deep Learning
  - ➡ Ian Goodfellow, Yoshua Bengio, Aaron Courville
  - ➡ Advanced, comprehensive, mathematical



Neuro

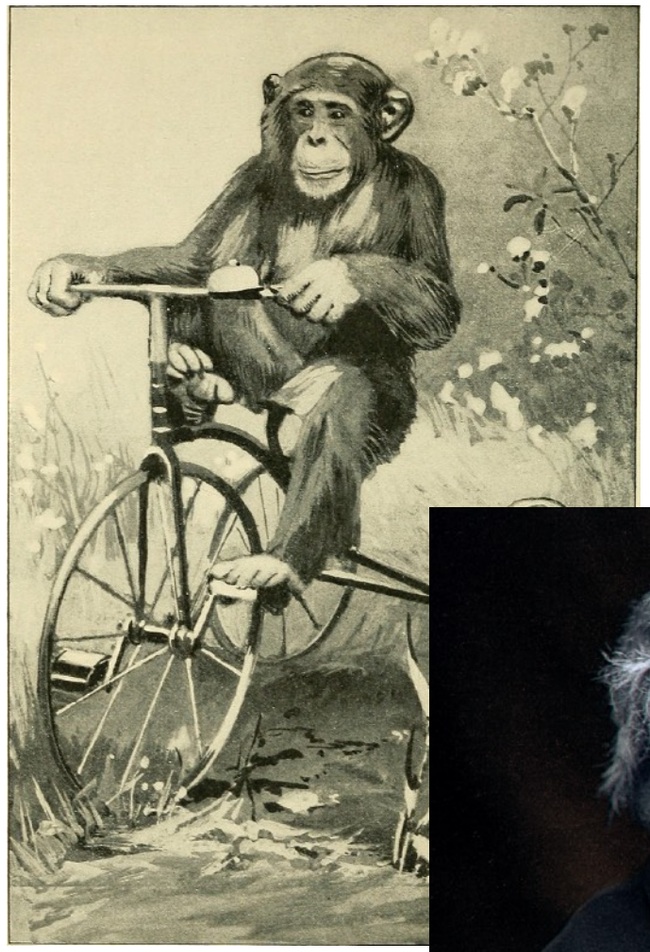
Neural networks

- Mimic a human brain to solve complex tasks

- Must be optimised for each task individually

Neuro

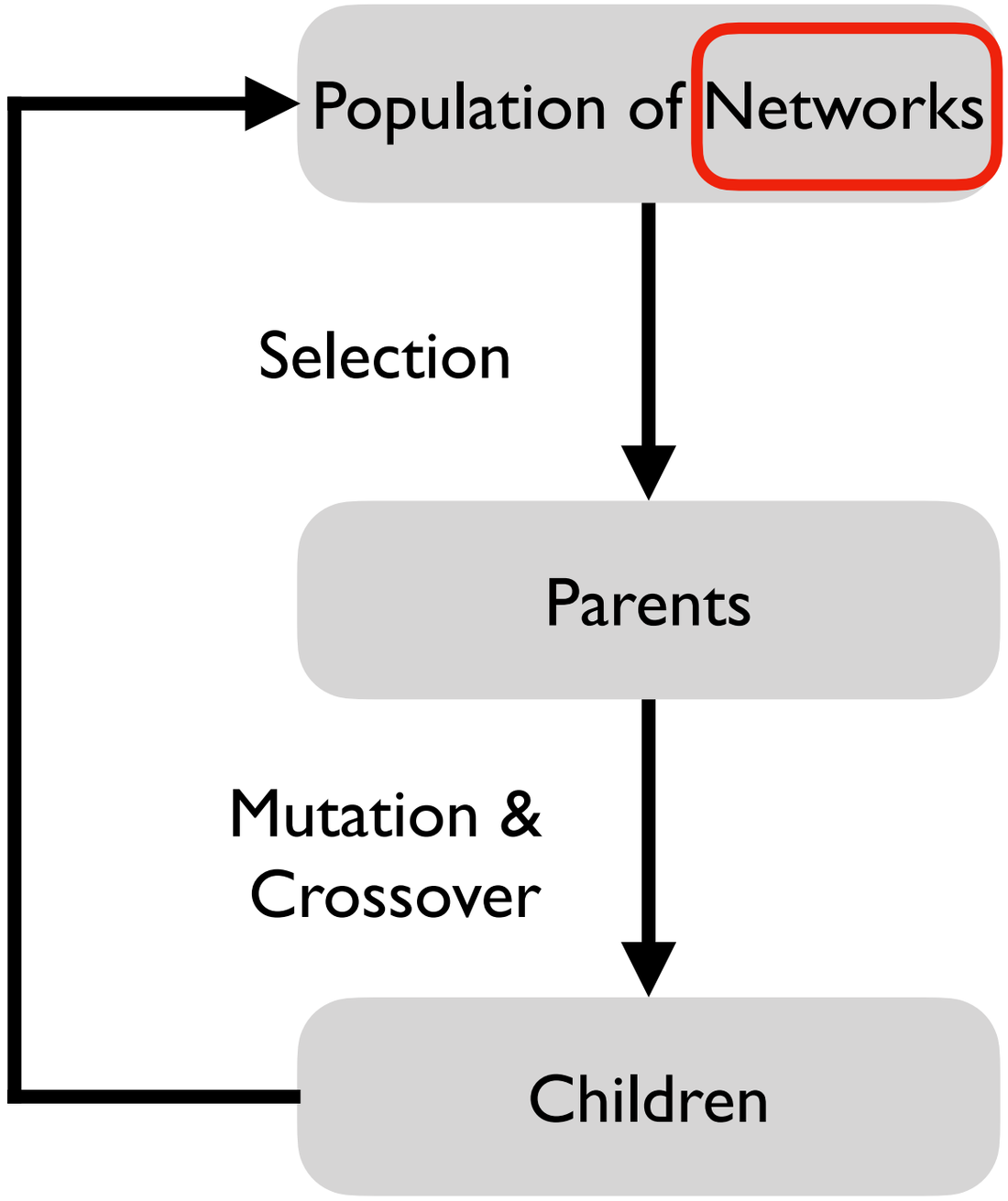
evolution



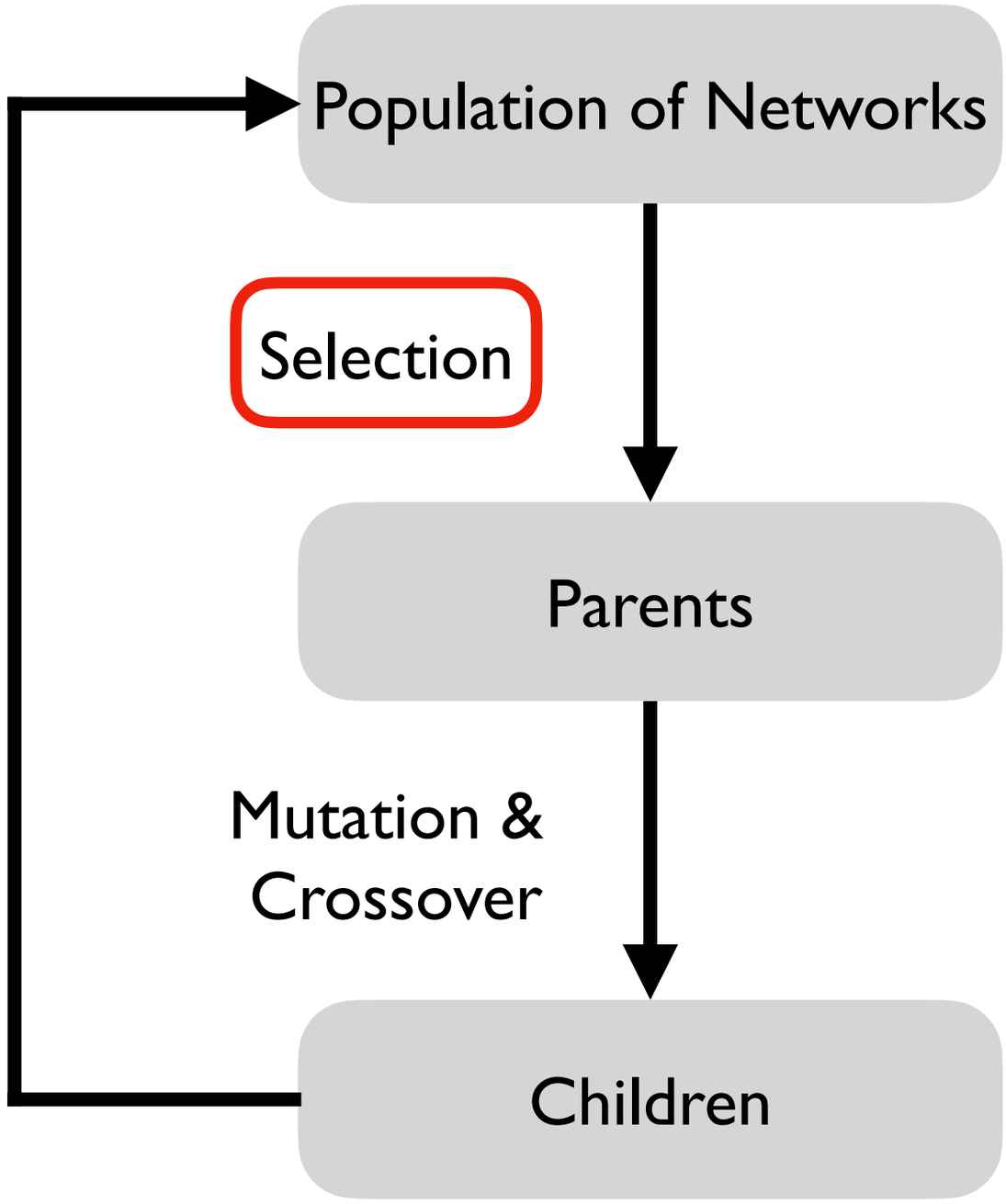
Neural networks

- Mimic a human brain to solve complex tasks
- Must be optimised for each task individually

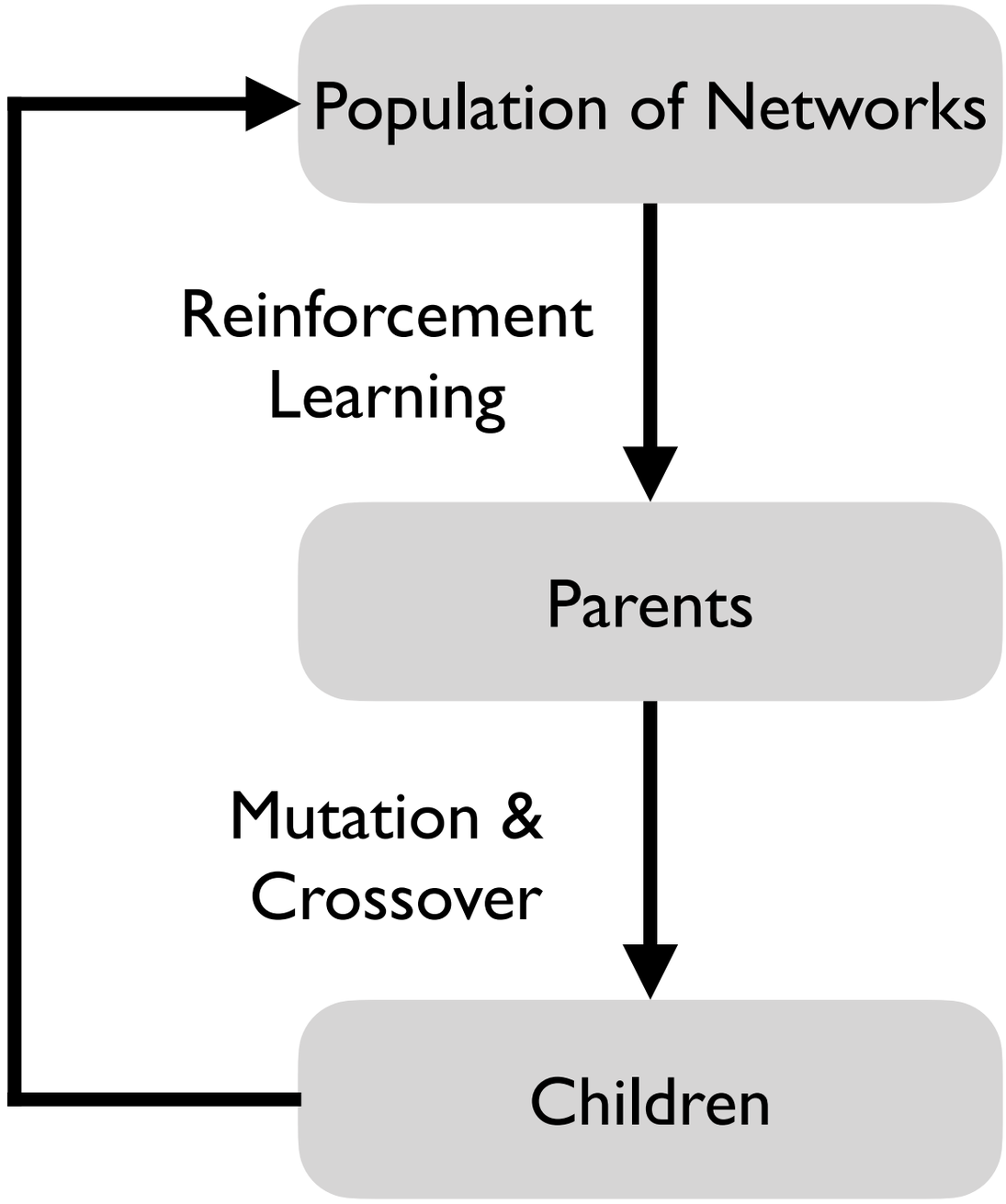
evolution


Population of Networks
Selection
Parents
Mutation &
Crossover
Children

evolution


Population of Networks
Selection
Parents
Mutation &
Crossover
Children

evolution


Population of Networks
Reinforcement Learning
Parents
Mutation & Crossover
Children

SANE

Efficient Reinforcement Learning through Symbiotic Evolution

DAVID E. MORIARTY AND RISTO MIIKKULAINEN moriarty,risto@cs.utexas.edu
Department of Computer Sciences, The University of Texas at Austin. Austin, TX 78712

Received October 20, 1994 ; Revised October 6, 1995

Editor: Leslie Pack Kaelbling

Abstract. This article presents a new reinforcement learning method called SANE (Symbiotic, Adaptive Neuro-Evolution), which evolves a population of neurons through genetic algorithms to form a neural network capable of performing a task. Symbiotic evolution promotes both cooperation and specialization, which results in a fast, efficient genetic search and discourages convergence to suboptimal solutions. In the inverted pendulum problem, SANE formed effective networks 9 to 16 times faster than the Adaptive Heuristic Critic and 2 times faster than Q-learning and the GENITOR neuro-evolution approach without loss of generalization. Such efficient learning, combined with few domain assumptions, make SANE a promising approach to a broad range of reinforcement learning problems, including many real-world applications.

Keywords: Neuro-Evolution, Reinforcement Learning, Genetic Algorithms, Neural Networks.

1. Introduction

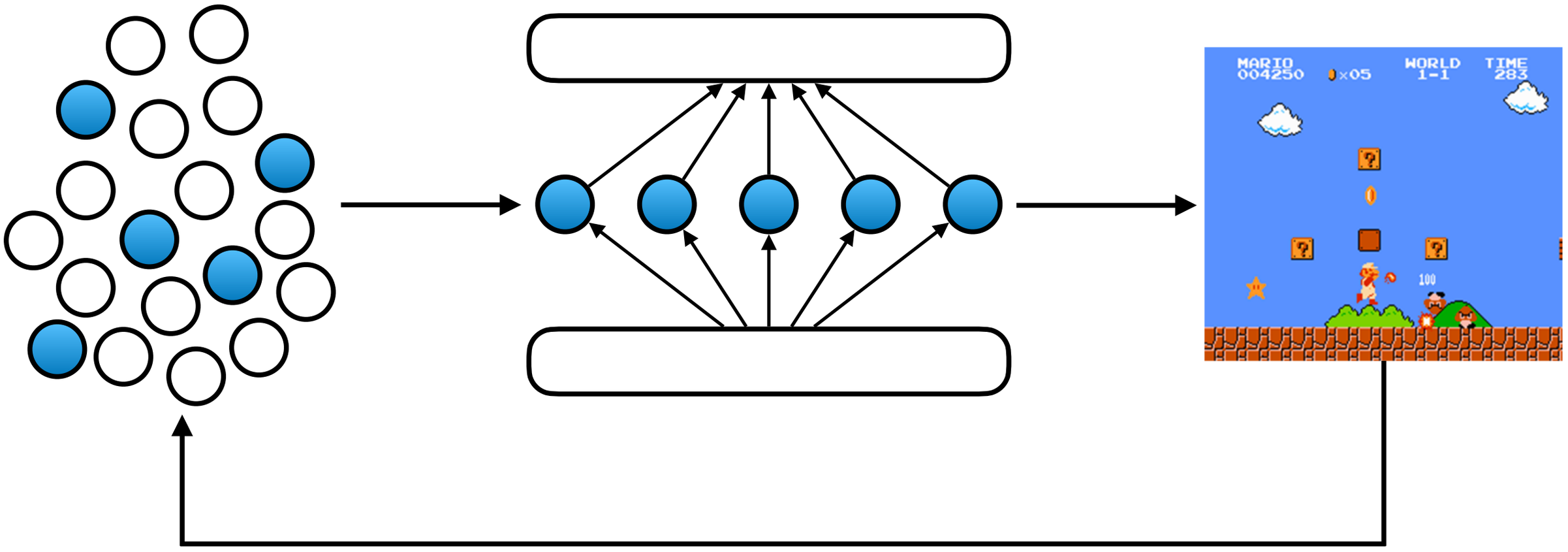
Learning effective decision policies is a difficult problem that appears in many real-world tasks including control, scheduling, and routing. Standard supervised learning techniques are often not applicable in such tasks, because the domain information necessary to generate the target outputs is either unavailable or costly to obtain. In reinforcement learning, agents learn from signals that provide some measure of performance and which may be delivered after a sequence of decisions

SANE

Population of hidden neurons

Subpopulation forms neural network

Network evaluation in problem environment



Reward