Deep Reinforcement Learning and Artificial Intelligence

Matteo Hessel

Artificial Intelligence

What is AI?

Objective:

- Reproduce in software the salient features of *intelligence*.

Key Features:

- *Logical Reasoning*: deduction of the implications of certain facts.
- *Sequential Decision Making*: reasoning about extended action sequences.
- *Learning*: adaptation and self-improvement.

Artificial Intelligence

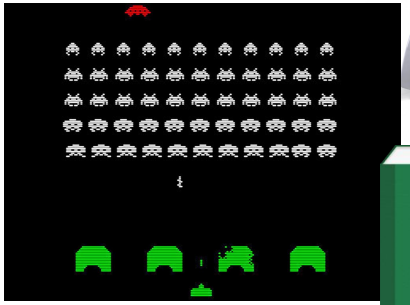
Brief history of AI

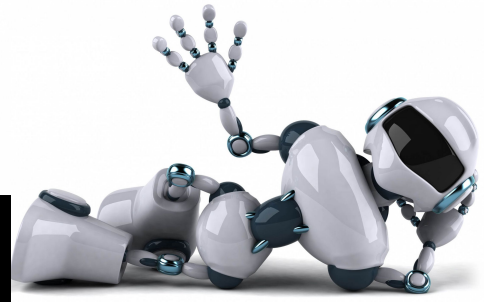
- ‘46 - ENIAC/EDVAC: first GP computers.
- ‘50 - Turing: The Turing Test (*Evaluation*).
- ‘56 - Simon: Automatic Theorem Prover (*Logic*).
- ‘57 - Rosenblatt: The Perceptron (*Learning*).
- ‘57 - Bellman: MDPs (*Sequential Decision Making*).
- ‘80s - ConvNets (*Vision*).
- ‘90s - Graphical Models (*Probabilistic Reasoning*).
- ‘06 - Deep Learning (*Neural networks at Scale*).

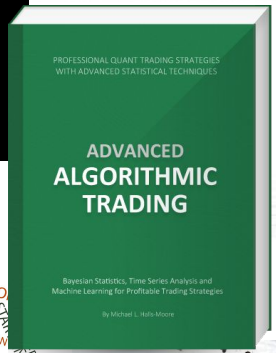
Artificial Intelligence

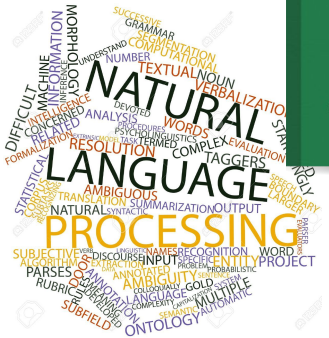
Current applications of AI

- Machine Vision
- Natural Language Processing
- Robotics
- Online advertisement
- Web Search
- Finance and Trading
- Health





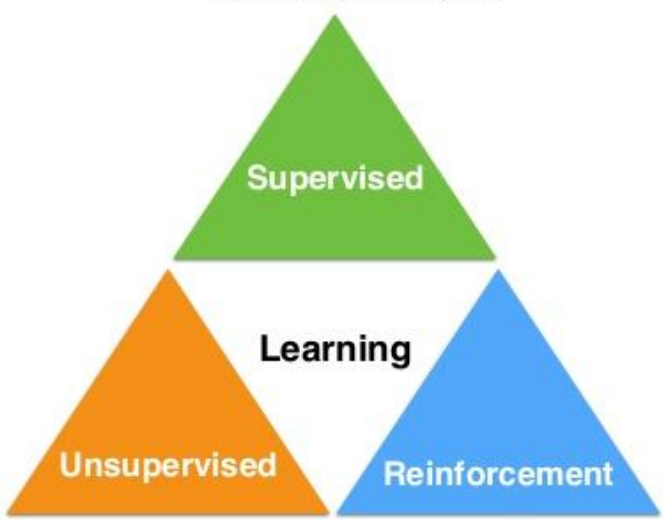




Machine Learning

Putting learning at the center of AI

*A collection of **algorithms** and techniques to design systems that **improve** their performance on a task as they gain **experience** in performing it.*



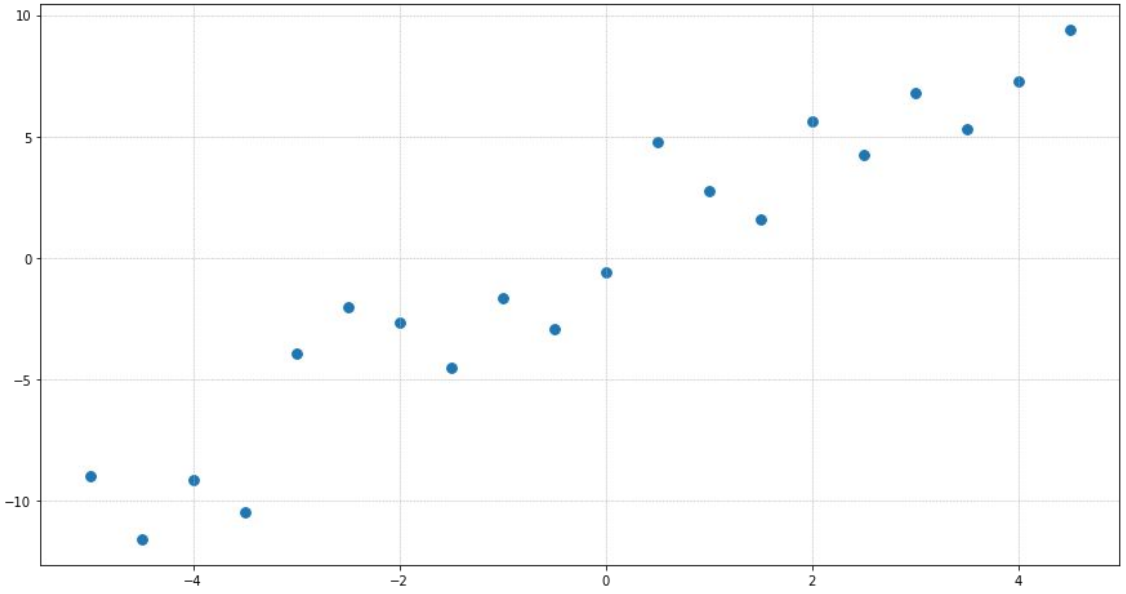
Supervised Learning

What is supervised learning?

- A **supervised learning** system is provided experience in the form of a **dataset**
- In SL this data takes the form of a set of input-output pairs *(x, y)*:
  - where inputs are identified by a set of **attributes** $x = \{x_1, x_2, ..., x_n\}$
  - and the output *y* can be either a **discrete** or **continuous** label
- The system must learn a **function**, that correctly maps inputs x to outputs y

Problem 1: Curve fitting

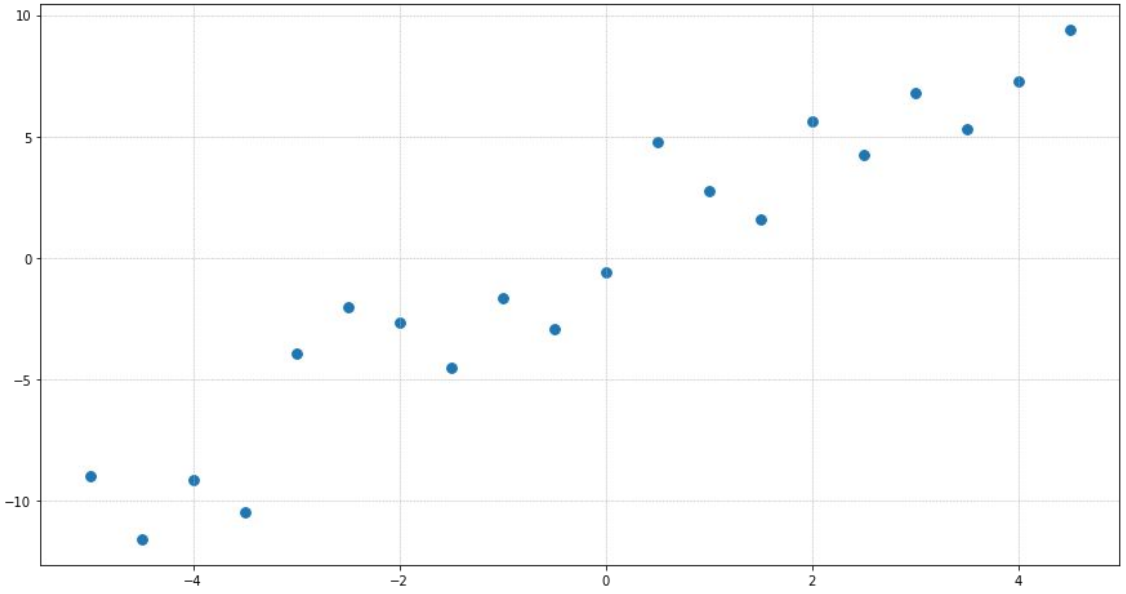
Fitting functions to data points



This is the problem of finding a function that maps as **accurately** as possible inputs *x* to outputs *y*

Problem 1: Curve fitting

Fitting functions to data points

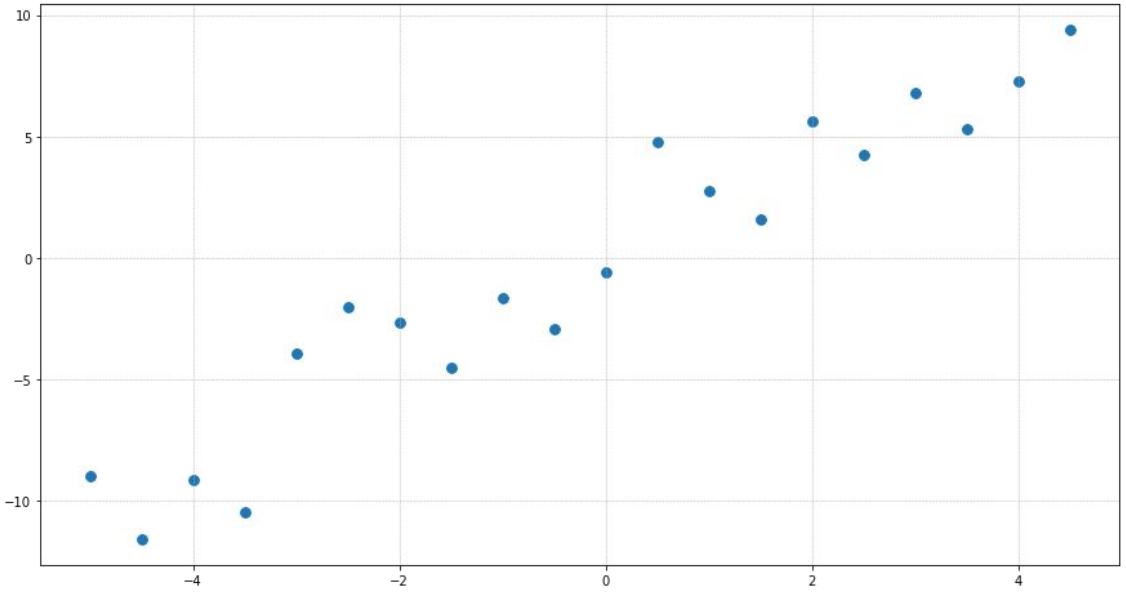


First, you need to specify what **accurate** even means? Typically this is done by a **loss function**:

$$MSE = \sum_i [f(x_i) - y_i]^2$$

Problem 1: Curve fitting

Fitting functions to data points

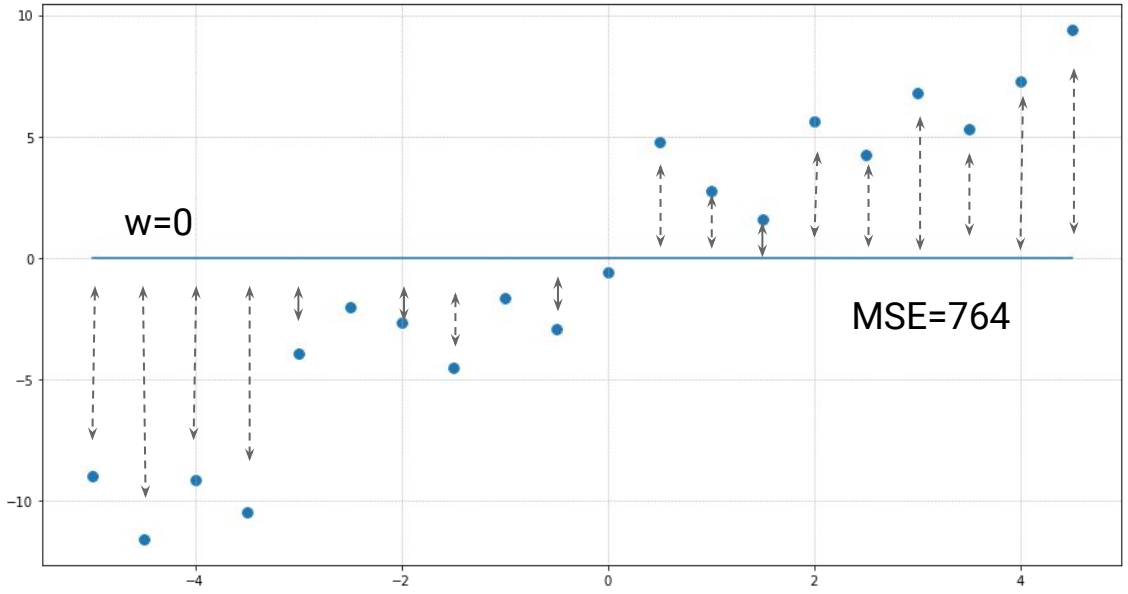


Next, you need to specify among what class functions you are considering, for instance:

$$f(x) = w * x$$

Problem 1: Curve fitting

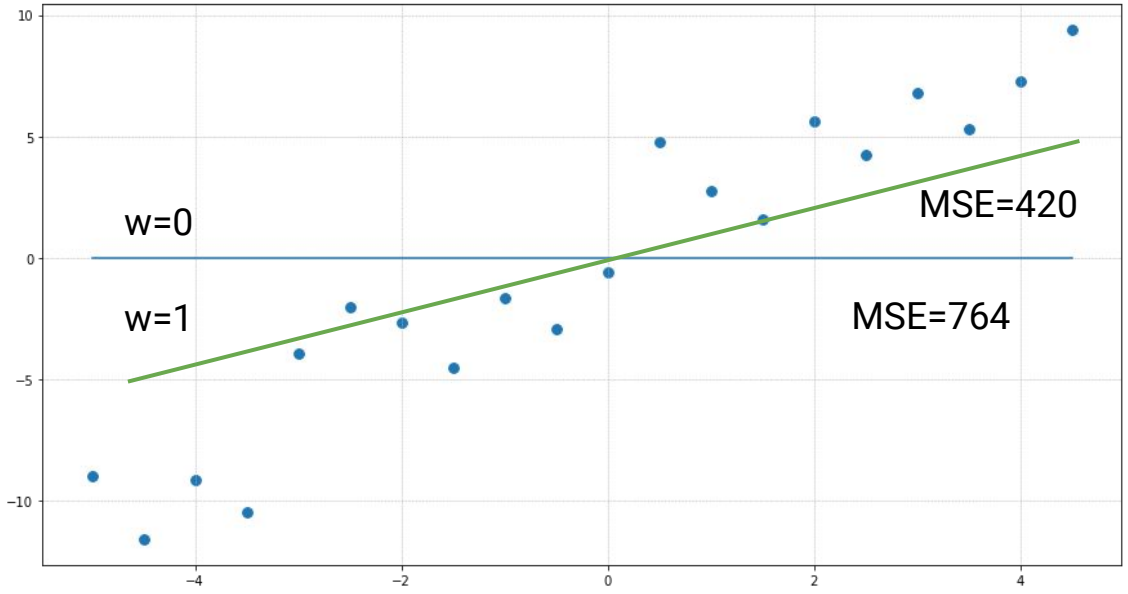
Fitting functions to data points



Then you can search in the selected family of functions and compared them based on the loss function

Problem 1: Curve fitting

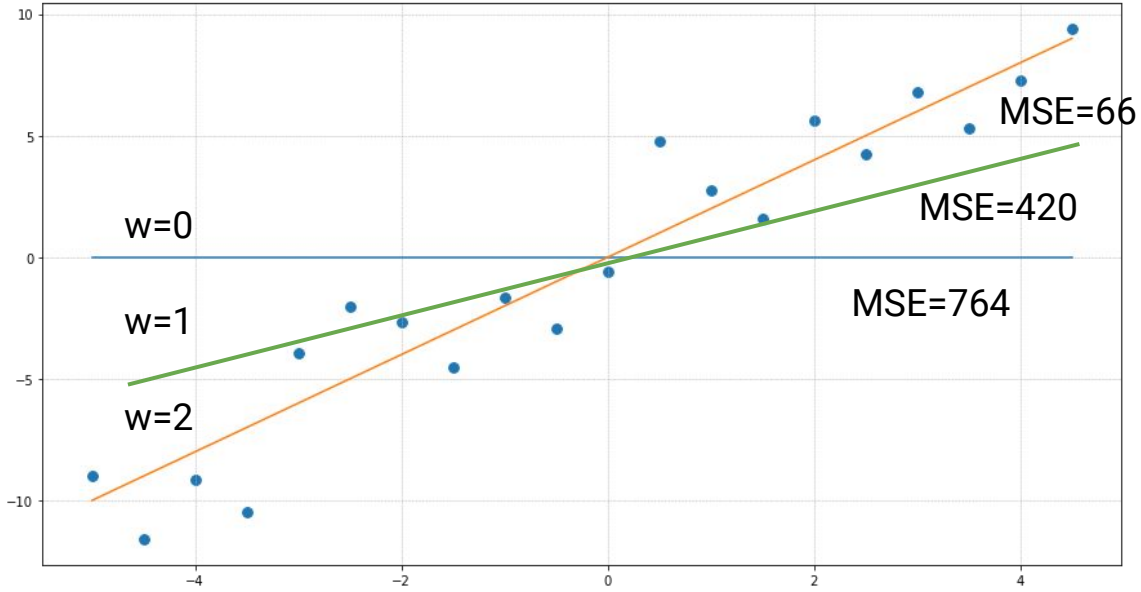
Fitting functions to data points



Then you can search in the selected family of functions and compared them based on the loss function

Problem 1: Curve fitting

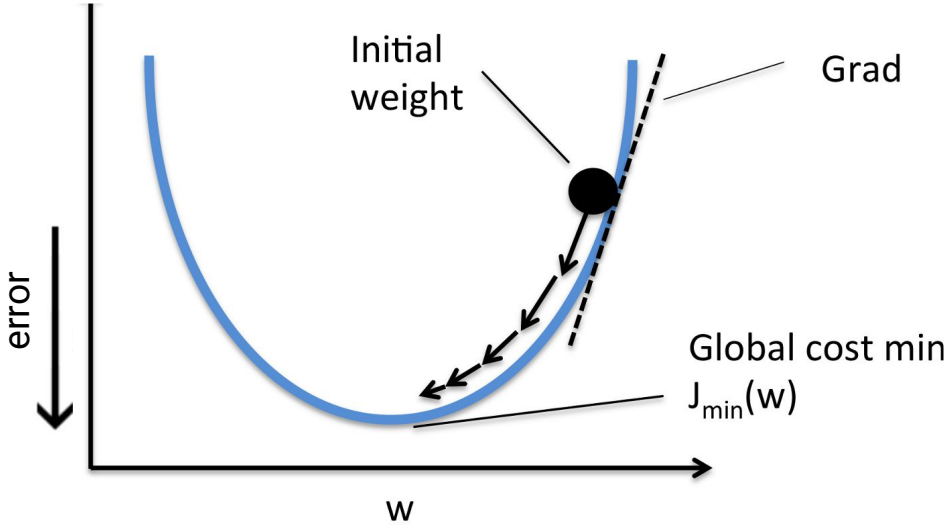
Fitting functions to data points

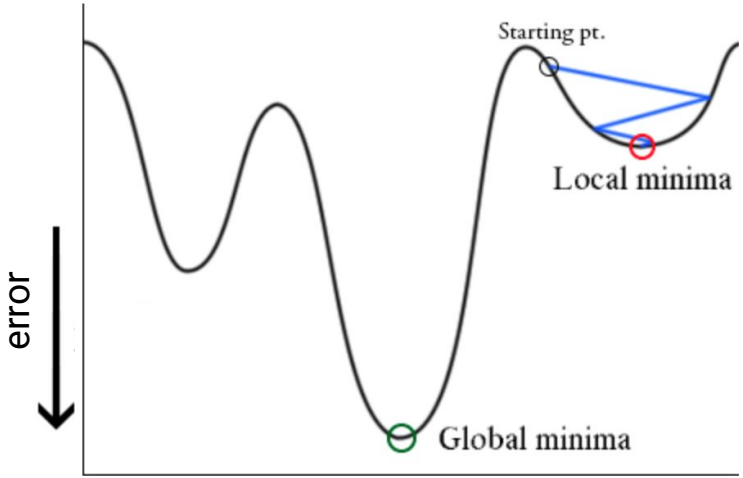


Then you can search in the selected family of functions and compared them based on the loss function

Gradient Descent

The workhorse of AI





Credits: *towardsdatascience.com/deep-learning-personal-notes-part-1-lesson-2*

Problem 2: Generalization

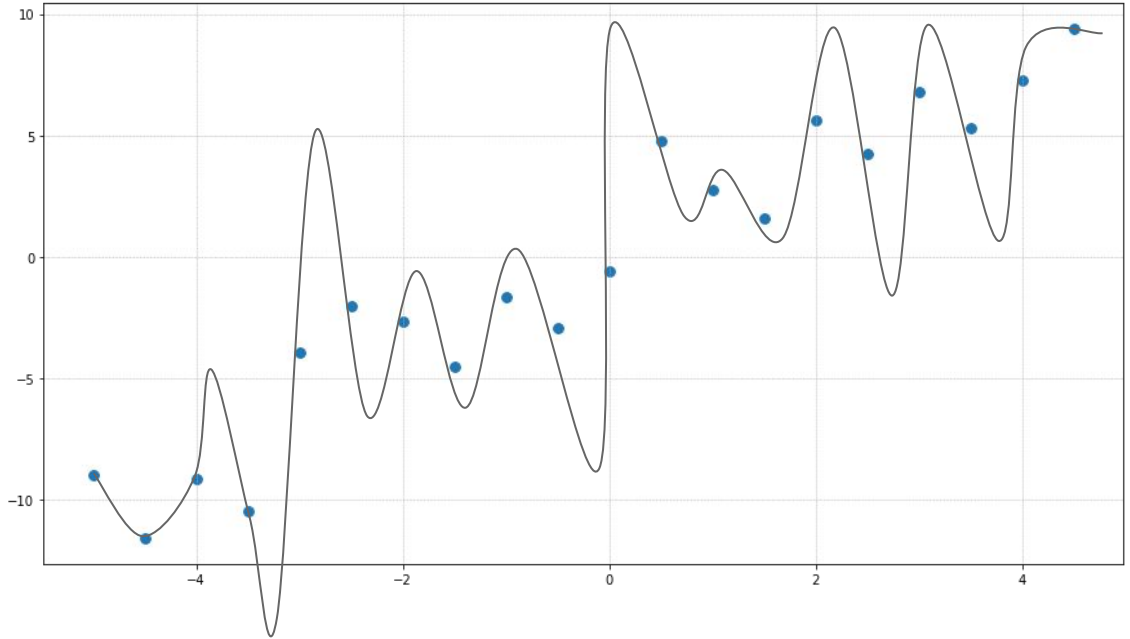
Learning functions that generalize to new data points

What makes it even more difficult is it's combination with **generalization**

The aim is that when **new data** is provided, as long as this data is generated from a similar phenomenon, problem or setting, the system will **generalize what it has learned,** make reasonable predictions for the new data as well.

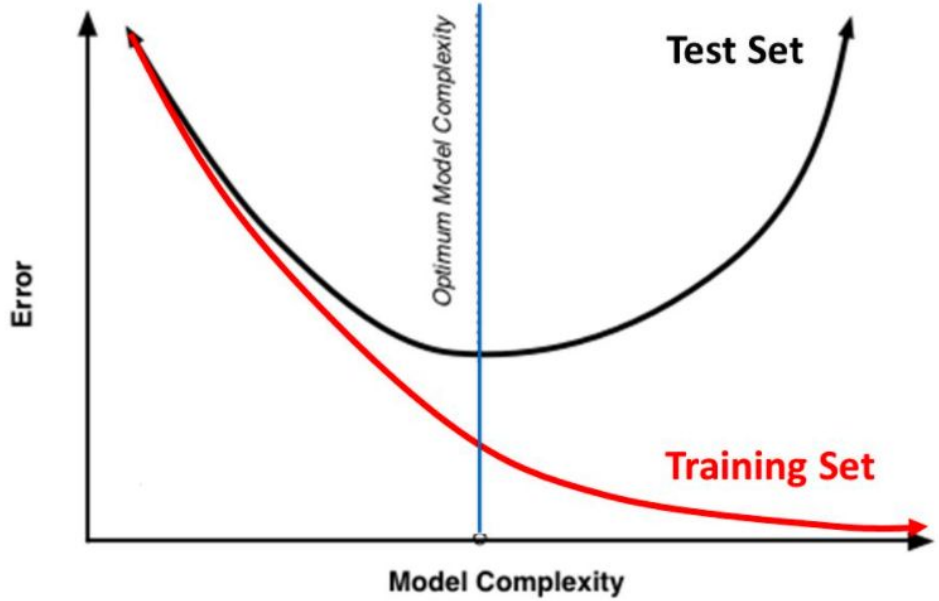
Overfitting

When does generalization fail?



Overfitting

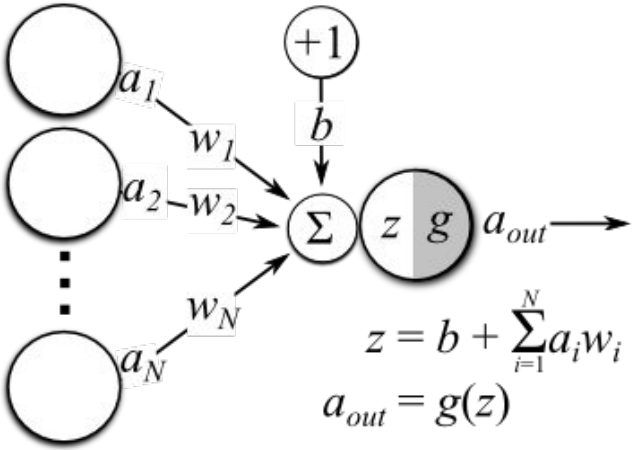
When does generalization fail?

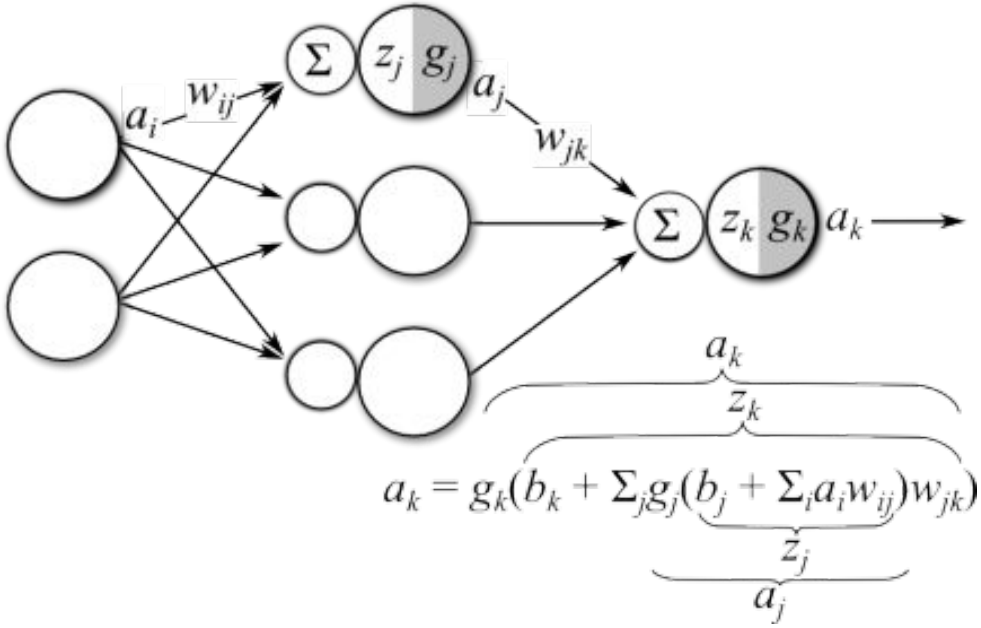


Credits: *analyticsvidhya.com/blog/2018/04/fundamentals-deep-learning-regularization-techniques*

Neural Networks

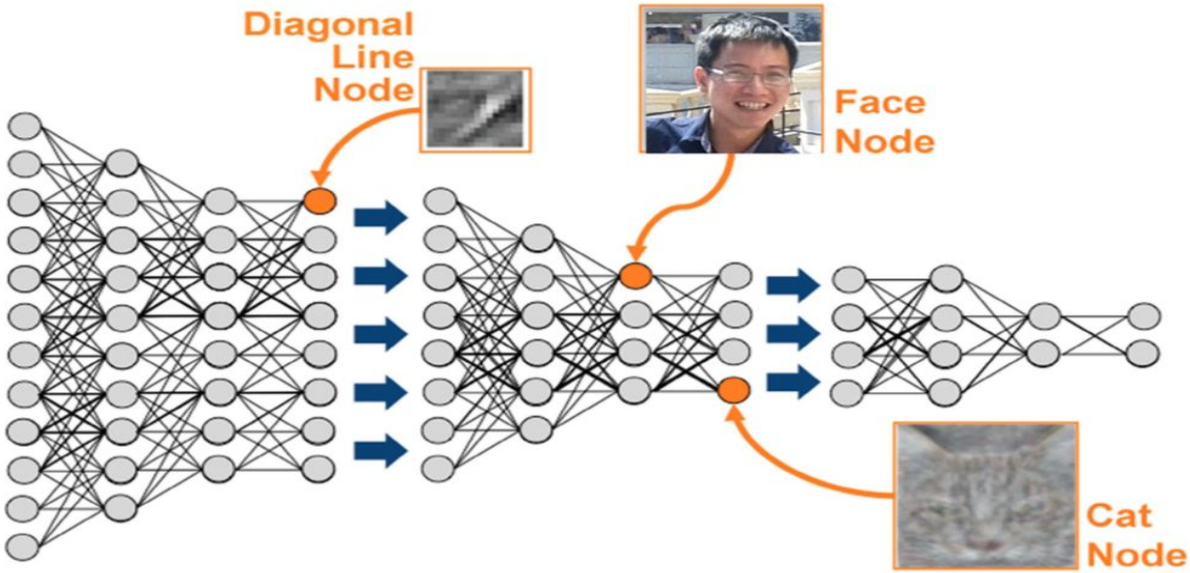
A flexible class of functions





Deep Learning

Neural networks at scale



Credits: *medium.com/your-personal-sim-pt-4-deep-agents-understanding-natural-intelligence*

Deep Learning

How do we achieve good generalization?

Research in Deep Learning focuses on designing

- suitable architectures
- suitable loss functions
- suitable training procedures

Neural networks are incredibly flexible

- They can fit any dataset with very **low error**
- They are **universal** function approximators

Isn’t that bad for generalization?

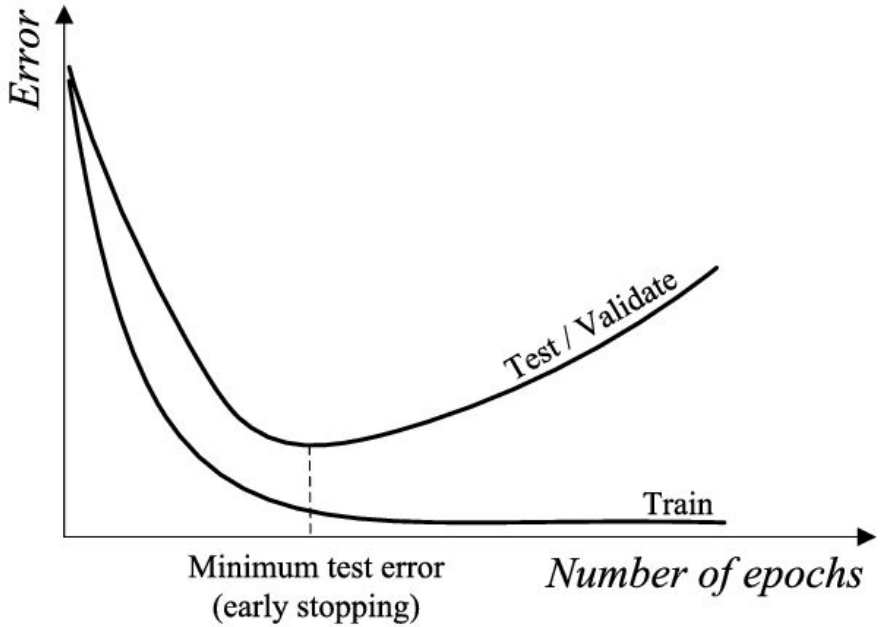
- Yes, but maybe we can work around ti?

Vision

ImageNet

Early stopping

Preventing overfitting in deep learning



Is this all?

Is this all we need for Artificial Intelligence?

“Machines will be capable, within twenty years, of doing any work a man can do”

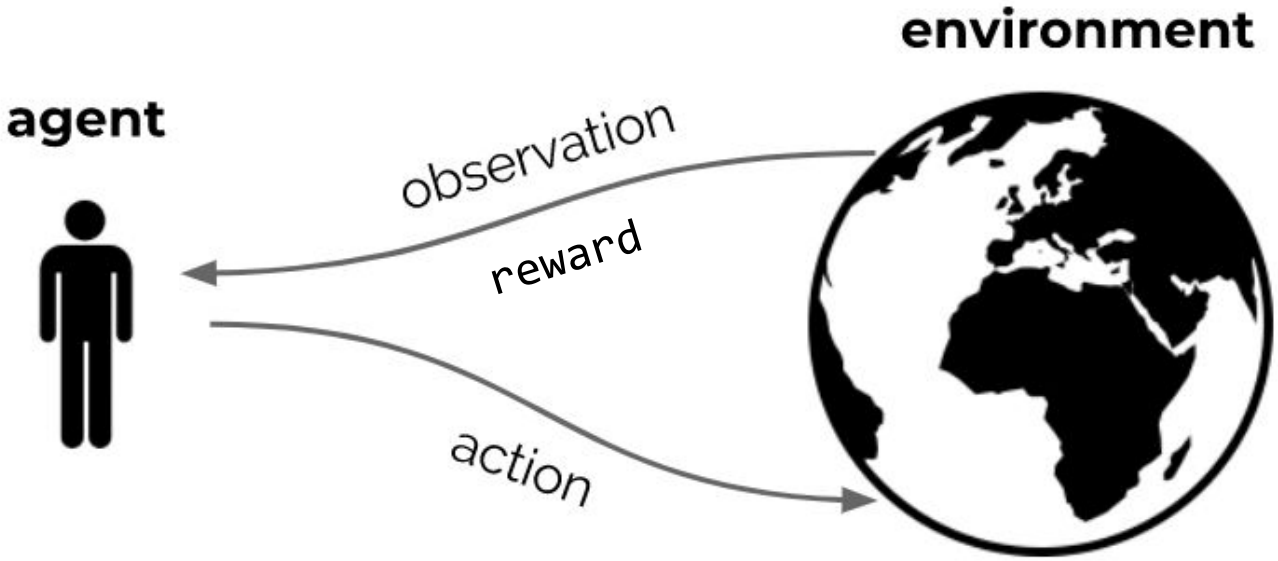
– Herbert Simon, 1956.

Deep learning is only solving two problems:

- Curve fitting
- Generalization

Reinforcement Learning

Agents and Environments



Reinforcement Learning

The objective

The agent's **behaviour** is defined by a **policy** $\pi(A_t|H_t)$, a (possibly stochastic) function that maps the **history** of previous observations H_t onto **actions**

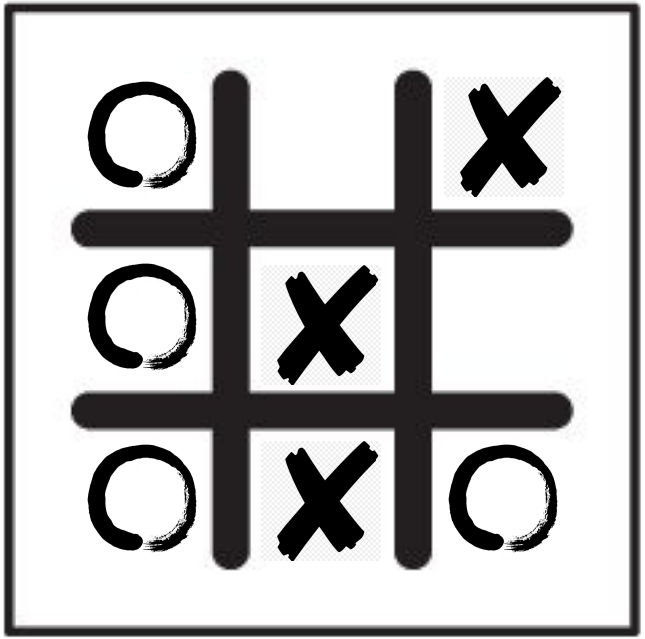
The agent's **objective** is to find an **optimal** policy $\pi^*(A_t|H_t)$ that maximizes the discounted sum of future rewards, or **return:**

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

An environment is said **fully observable** when such optimal policy can depend only on the **last observation** received by the agent

Problem 3: Credit Assignment

How do we assign credit where credit is due?



Value Functions

Learning to predict returns

A **value function** is a learned estimate of the **return** you can expect from some timesteo onwards

- When executing a given policy π
- Averaging across the stochasticity in the environment
- Averaging across the stochasticity in the policy itself

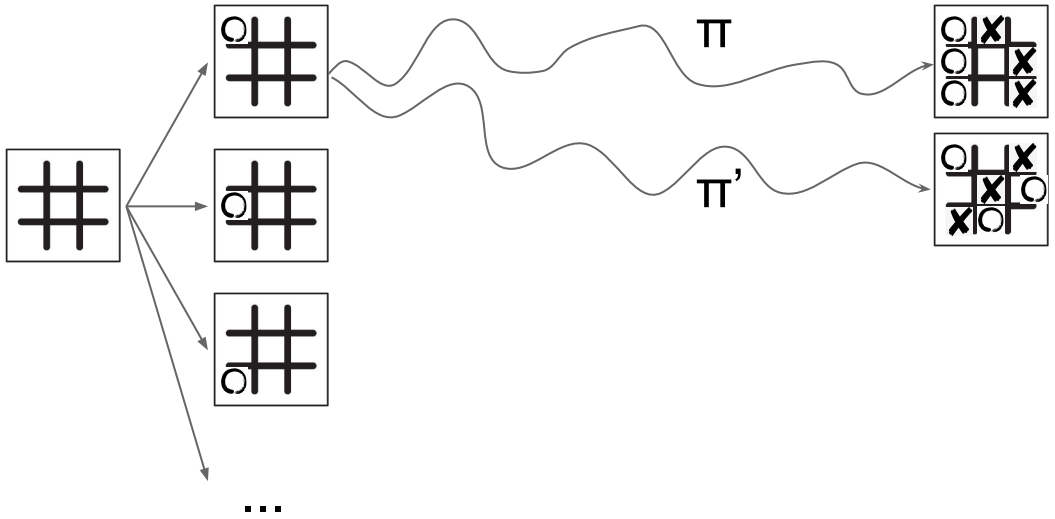
The most useful are **action-value** functions:

- $Q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$
- You can derive a policy by inspection of the values

Value Functions

Learning to predict returns

Action values *Q(s,a)* depend on the policy π that you execute **after** action *a*:



Optimal Value Functions

Not all values were created equal

Optimal action values *Q*(s,a)* measure how much return would the agent collect

- If it first selects action a in state s
- And acts in the **best possible way** thereafter

If I have good predictions of the optimal action values I can get an **optimal policy**

$$\pi^*(s) = argmax_a Q^*(s,a)$$

But how can **estimate** the optimal action values?

Bellman Equations

A recursive decomposition of optimal values

Optimal values admit a **recursive decomposition**:

$$Q^*(S_t, A_t) = E[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a')]$$

This kind of equation is called a **Bellman Equation**.

Solving this equation results in finding the optimal values.

This however is a huge system of non-linear equations, that depend on the dynamics of the environment **which is a priori unknown**

Temporal Difference Learning

Iterative solution of bellman equations

Temporal difference learning algorithms solve for this equation iteratively by

- **Initializing** value estimates at random
- **interacting** with the environment
- applying **small incremental** updates on each transition

  so as to **make current estimates satisfy better the bellman equation**

Temporal Difference Learning

A concrete algorithm for solving bellman equations

$S = \texttt{env.initial\_state()}$

$Q(s, a) = \texttt{random()} \quad \forall s, a$

While True:

$A = \texttt{argmax}_a Q(S, a)$

$R, S' = \texttt{env.step}(A)$

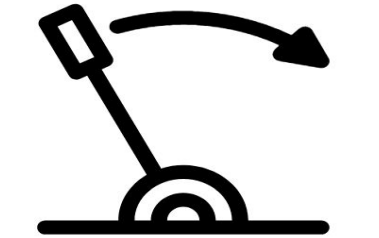
$Q(S, A) \rightarrow R + \gamma \max_{a'} Q(S', a')$

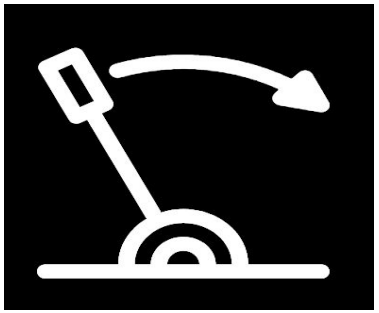
Q-learning

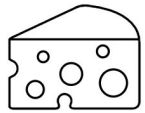
$Q^{new}(S,A) = (1-\alpha)\, Q^{old}(S,A) + \alpha\, [R + \gamma \max_a Q^{old}(S',a) - Q^{old}(S,A)]$

Problem 4: Exploration

The Exploration-Exploitation Dilemma







Temporal Difference Learning

A concrete algorithm for solving bellman equations

$S = \texttt{env.initial\_state()}$

$Q(s, a) = \texttt{random()} \quad \forall s, a$

ε-greedy Q-learning

While True:

$$A = \begin{cases} \texttt{argmax}_a Q(S, a), \texttt{ with prob.} (1 - \epsilon) \\ \texttt{act\_random()}, \texttt{ otherwise} \end{cases}$$

$R, S' = \texttt{env.step}(A)$

$Q(S, A) \rightarrow R + \gamma \max_{a'} Q(S', a')$

Deep Reinforcement Learning

Combining deep learning and reinforcement learning

Values and policies are ultimately just **functions**

Deep learning provides us with a collection of techniques for flexibly **representing** and efficiently **learning** functions

Deep Reinforcement Learning studies how to apply classical Reinforcement Learning ideas to train deep neural network to model values, policies and models, by **converting updates** like the one we saw before **into loss functions**

Why is deep RL hard?

Combining deep learning and reinforcement learning

Reinforcement Learning violates many **assumptions** of deep learning methods

E.g. in supervised learning

- we assume that each input x is **independent** of the next.
- we assume that the distribution of inputs x and labels y is **fixed** during the course of training

Why is deep RL hard?

Combining deep learning and reinforcement learning

For many years people thought that combining deep learning with reinforcement learning was **inherently instable**

Hence a focus on:

- Tabular methods
- Linear methods

For which strong theoretical guarantees of convergence can be more easily found

But using powerful function approximation and exploit **generalization** is essential to scale up reinforcement learning to challenging domains

DQN

Combining deep learning and reinforcement learning

DQN was one of the first agents to successfully apply deep learning to RL

- Implements the **Q-learning** algorithm described before
- But the Q-estimates are the output of a **deep neural network**
- Trained via **gradient descent** to minimize on each transition

$$Loss(s, a, r, s') = [r + \gamma \max_{a'} Q(s', a') - Q(s, a))]^2$$

- Store last N transitions in a big **memory buffer**
- Compute loss and gradient updates on batches of data sampled randomly from the memory buffer

1. RL-Aware Deep Learning

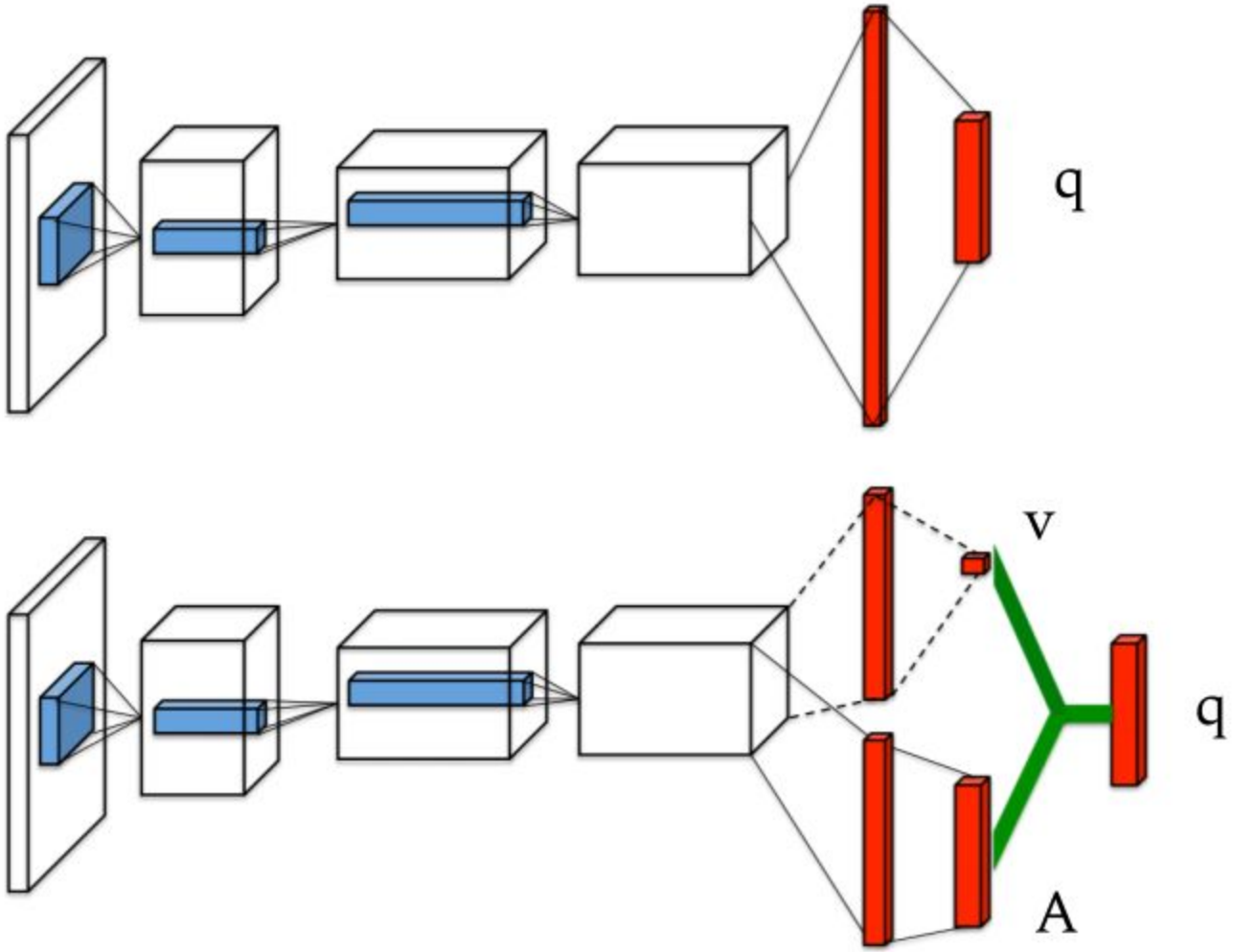
Custom neural network atchitectures for RL

Deep learning methods were largely designed for supervised learning losses

- The type of predictions and functions in RL have specific structures

It is reasonable to assume that just using out of the box the architectures developed for SL is largely suboptimal

Can we design **RL-aware deep learning** architectures?


q
v
q
A

2. Partially observable environments

What do I do if the optimal policy depends on the entire history?

If the environment is **partially observable**

- The optimal policy depends on the entire history of observations the agent has seen up to that moment

The agent must **summarize** the history in a compact representation (the **state**)

- For computational efficiency the state should have a **recursive** form

$$S_{t+1} = f(S_t, O_t)$$

This can be implemented quite easily in deep RL using **recurrent neural networks**

3. Stable off-policy deep RL

Can we learn effectively from data generated from someone else?

In many situations an RL agent during learning might not be very **safe**

- Learns via **trial and error** by **interacting** with the world
- Explores via **random selection** of actions

Imagine this in the context of:

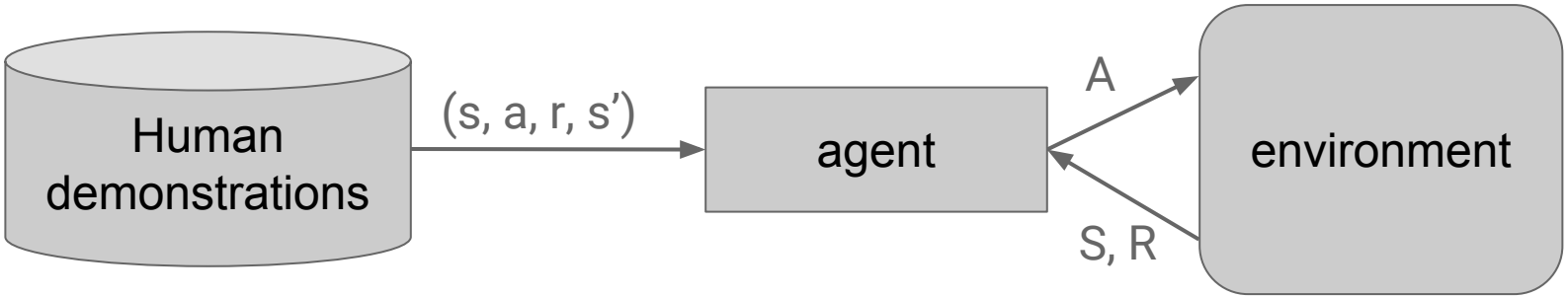
- Robotics
- Autonomous cars

3. Stable off-policy deep RL

Can we learn effectively from data generated from someone else?

For instance an agent could learn from transitions (s, a, r, s')

- where actions were selected by a human



Until the agent has reached **basic competence**

4. How do we scale up Reinforcement Learning?

Much progress in Deep Learning was result of a general principle

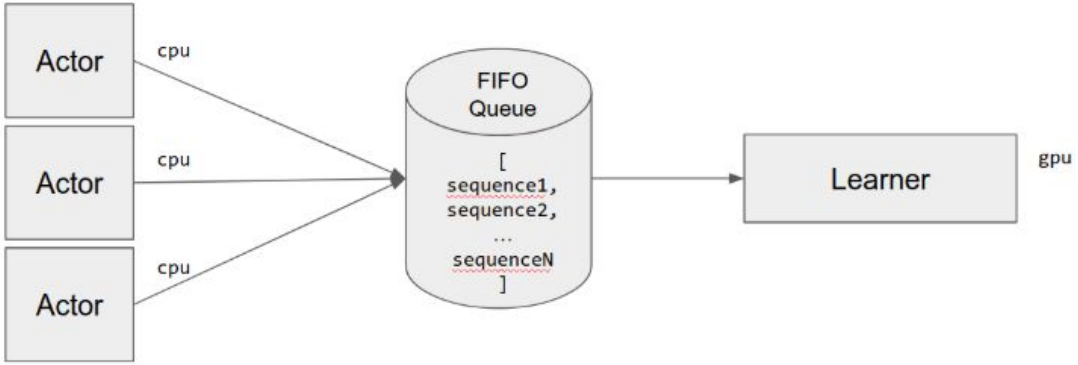
More Data + More Computation
= Better Performance

It is reasonable to assume that a similar principle also holds for RL
In the past few years many agents have been proposed designed for scale

4. How do we scale up Reinforcement Learning

Actor-Learner Decomposition

Most large scale RL agents exploit a decomposition between **acting** and **learning**



Many actors interact in their own copy of the environment, and a **single learner** processes all the experience in bulk using powerful hardware (e.g., GPU/TPU)

5. The “problem” problem

What environments should we train our agents on?

What are suitable tasks/problems to train and improve our agents?

- This is often called **the “problem” problem**

We need tasks that

- are sufficiently **complex** to require intelligence to be solved
- are **not so difficult** that we cannot observe any progress
- they must be **cheap** and **fast** to run experiments on
- they must be **safe**

Video-games

The role of video-games and simulation for AI research

The **video-game** community has been an incredibly prolific source of tasks that share all of these important properties:

- They we're designing to be challenging and fun for humans
- They often come with natural curriculums (levels) of different difficulty
- Video games are cheap to run and can be run faster then real time
- There is no risk of damage to people or assets if things go wrong

Resources

Resources to study Reinforcement Learning and Artificial Intelligence

Videos:

- Advanced topics in Machine Learning - David Silver (Youtube)
- Reinforcement Learning - Hado van Hasselt (Youtube)

Books

- Reinforcement Learning: an introduction - Rich Sutton, 2018 (PDF)
- Algorithms for Reinforcement Learning - Csaba Szepesvari, 2009 (PDF)

Resources

Resources to study Reinforcement Learning and Artificial Intelligence

Open source agents:

- DQN (github/google/dopamine)
- IMPALA (github/deepmind)
- A2C/PPO (github/openai/baselines)

RL libraries

- TRFL

DeepMind

Interested in working at DeepMind?

If you are interested in these topics at DeepMind,
either on the research side or in real world applications of AI
we are always looking for talent

Solve intelligence and use it to make the world a better place

We’re hiring!

Interested in working at DeepMind?

Research Engineering:

- Paris
- Edmonton.

Internships:

- London
- Montreal