Unsupervised learning

B. Mehlig, Department of Physics, University of Gothenburg, Sweden

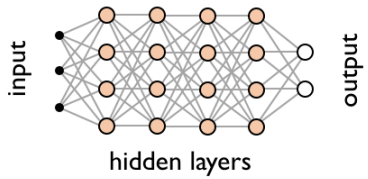
1. Introduction
2. Neural networks and deep learning (repetition)
3. Unsupervised learning
4. Reinforcement learning

Bernhard Mehlig, *Machine learning with neural networks,* Cambridge University Press (2021)

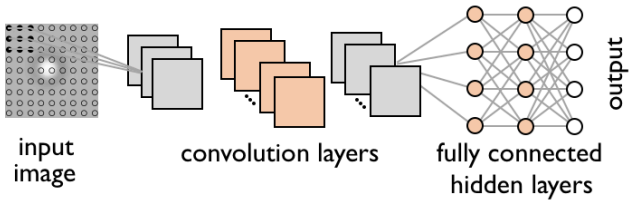
Introduction

Supervised

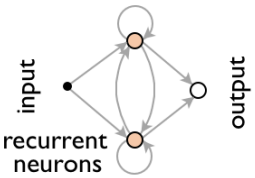
Neural net



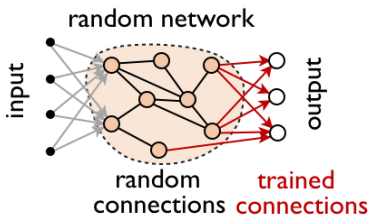
Convolutional neural net



Recurrent neural net

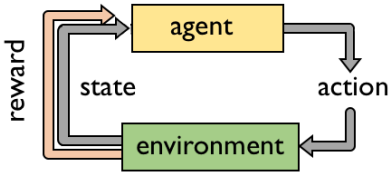


Reservoir computing



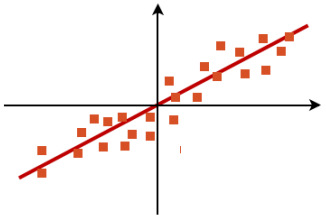
Semisupervised

Reinforcement learning

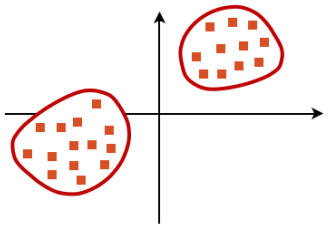


Unsupervised

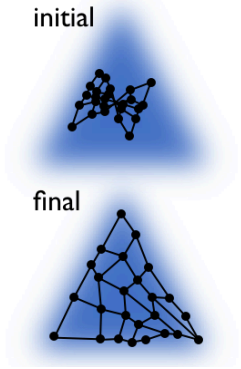
Principal-component analysis



Clustering



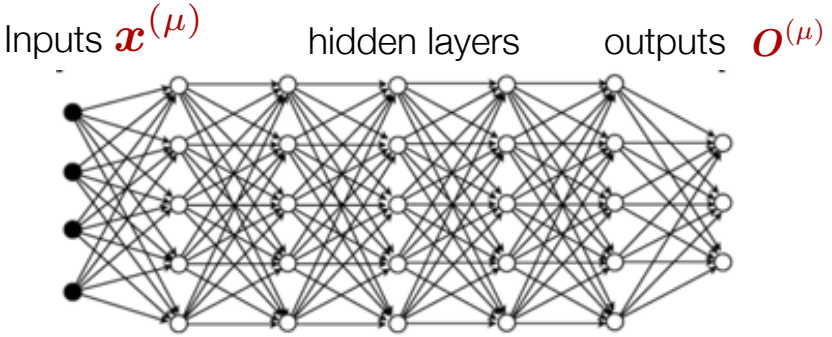
Self-organizing map



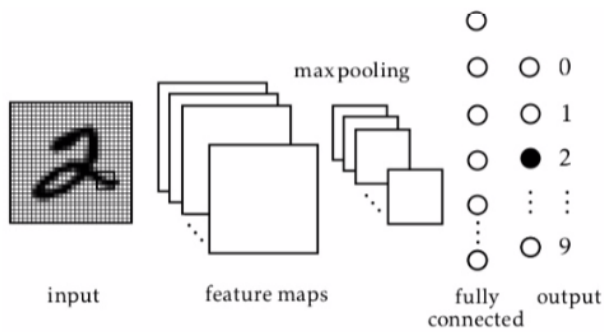
Deep neural networks

Rosenblatt, Psychological Review **65** (1958) 386

Connect neurons into networks that can perform computing tasks. Adjust parameters (weights and thresholds) to minimise output error $H = \frac{1}{2}\sum_{\mu,i}\left[t_i^{(\mu)} - O_i(\boldsymbol{x}^{(\mu)})\right]^2$.



Deep neural net (many hidden layers).



Convolutional neural net

Krizhevsky, Sutskever & Hinton (2012)

Inputs ($\mu = 1, \ldots, p$)

$$\boldsymbol{x}^{(\mu)} = \begin{bmatrix} x_1^{(\mu)} \\ \vdots \\ x_N^{(\mu)} \end{bmatrix}$$

Output for input $\boldsymbol{x}^{(\mu)}$

$$\boldsymbol{O}^{(\mu)} = \begin{bmatrix} O_1^{(\mu)} \\ \vdots \\ O_M^{(\mu)} \end{bmatrix}$$

Target for $\boldsymbol{x}^{(\mu)}$

$$\boldsymbol{t}^{(\mu)} = \begin{bmatrix} t_1^{(\mu)} \\ \vdots \\ t_M^{(\mu)} \end{bmatrix}$$

input dimension N - large
size of data set p - very large
number of layers L - large

Training — supervised learning

Rosenblatt, Psychological Review **65** (1958) 386

Determine network parameters (weights w_{ij} and thresholds θ_i by minimising H .

Learning rule for weights $w'_{mn} = w_{mn} + \delta w_{mn}$ with $\delta w_{mn} = -\eta \partial H / \partial w_{mn}$.

Example

$$H = \tfrac{1}{2} \sum_{i\mu} [t_i^{(\mu)} - O_i(\boldsymbol{x}^{(\mu)})]^2 \quad \text{with} \quad O_i(\boldsymbol{x}^{(\mu)}) = g(\underbrace{\textstyle\sum_j w_{ij} x_j^{(\mu)} - \theta_i}_{= b_i^{(\mu)}}) \; .$$

Use chain rule:

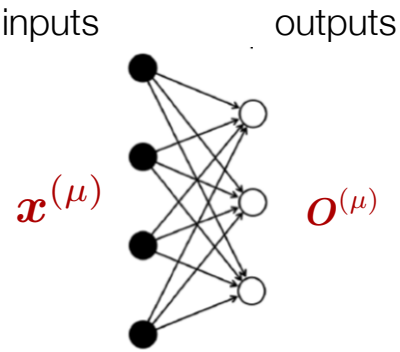
$$\frac{\partial H}{\partial w_{mn}} = -\sum_{i\mu} (t_i^{(\mu)} - O_i^{(\mu)}) \frac{\partial O_i^{(\mu)}}{\partial w_{mn}}$$

$$\frac{\partial O_i^{(\mu)}}{\partial w_{mn}} = g'(b_m^{(\mu)}) \delta_{mi} x_m^{(\mu)} \quad \text{with} \quad \delta_{mi} = \begin{cases} 1 & m = i \\ 0 & m \neq i \end{cases} \; .$$

Resulting learning rule:

$$\delta w_{mn} = \eta (t_m^{(\mu)} - O_m^{(\mu)}) x_n^{(\mu)}$$

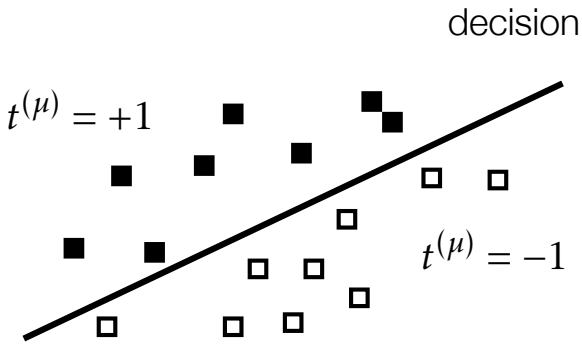
Essentially Hebb's rule, $\delta w_{mn} = \eta t_m^{(\mu)} x_n^{(\mu)}$.

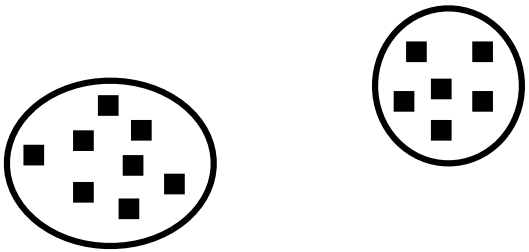


Unsupervised learning

Unsupervised learning

Unsupervised learning: learning without labels (targets)





Binary classification problem $t^{(\mu)} = \pm 1$.

Input distribution $P_{\text{data}}(\boldsymbol{x})$.

Supervised learning finds decision boundaries for labelled data $[\boldsymbol{x}^{(\mu)}, t^{(\mu)}]$, $\mu = 1, \ldots, p$.

.

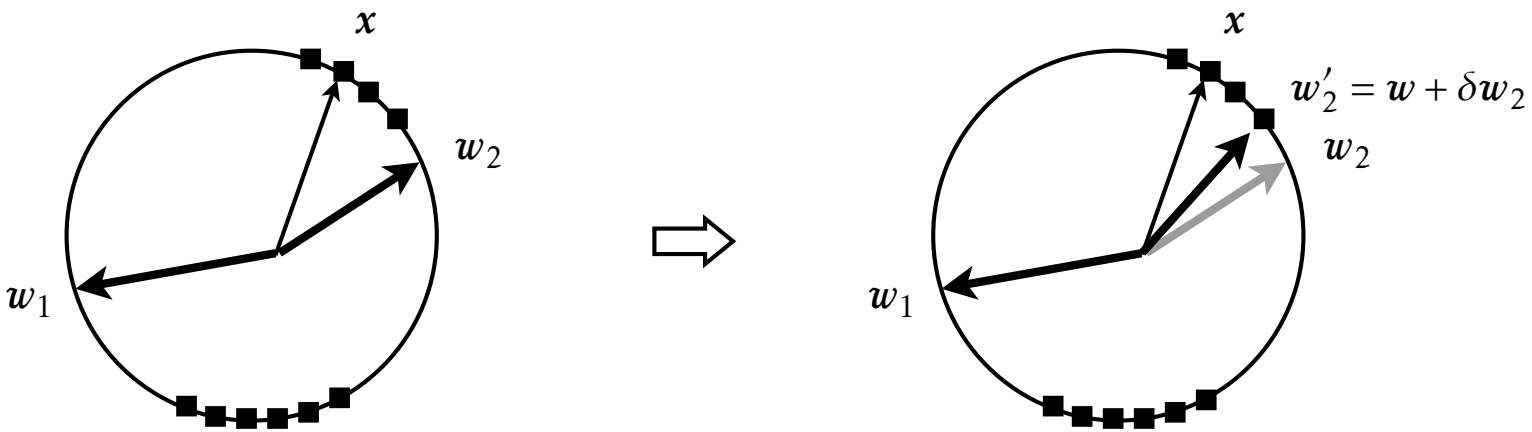
Unsupervised learning can identify clusters in the input data $\boldsymbol{x}^{(\mu)}$, $\mu = 1, \ldots, p$, drawn from input distribution $P_{\text{data}}(\boldsymbol{x})$

Competitive learning

Learning rule. $\delta \boldsymbol{w}_m = \eta y_m (\boldsymbol{x} - \boldsymbol{w}_m)$ with $y_m = \begin{cases} 1 & m \text{ is winning neuron } (\boldsymbol{w}_m \text{ closest to } \boldsymbol{x}), \\ 0 & \text{otherwise.} \end{cases}$

Closely related to Hebb's rule $\delta w_{mn} = \eta y_m x_n$. Second term is weight decay.

Example: patterns $\boldsymbol{x}$ and weights $\boldsymbol{w}_i$ one unit circle.

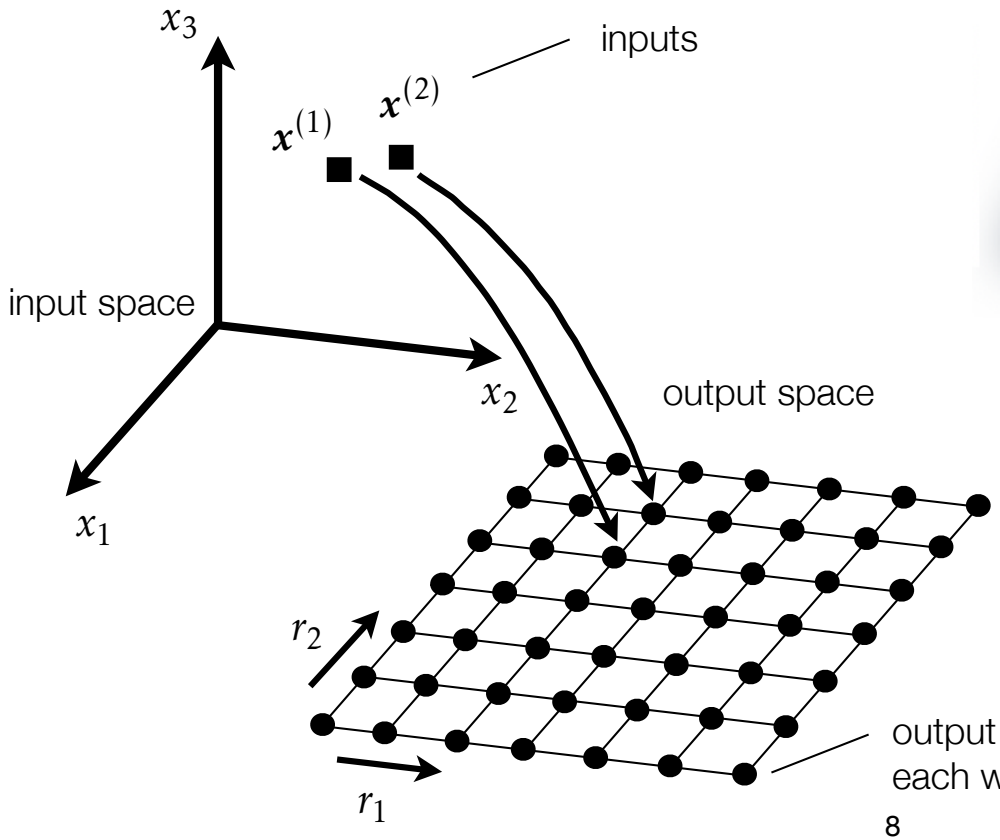


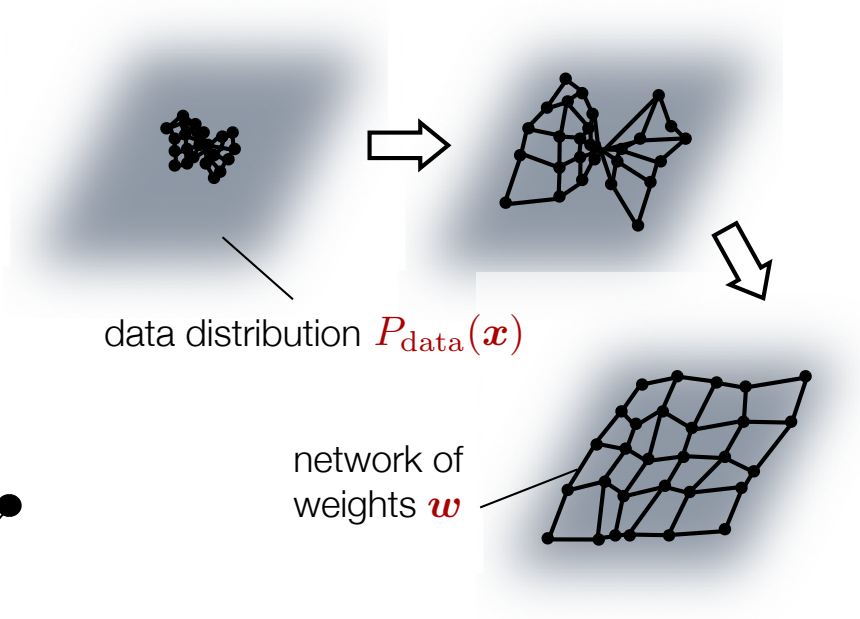
Competitive learning finds clusters in input data.

Self-organising map

Topographic map: non-linear representation of high-dimensional input distribution that preserves distance.

Idea: self-organising topographic map develops in response to the inputs it receives.
Competitive learning rule with distance information





Topographic map for MNIST digits

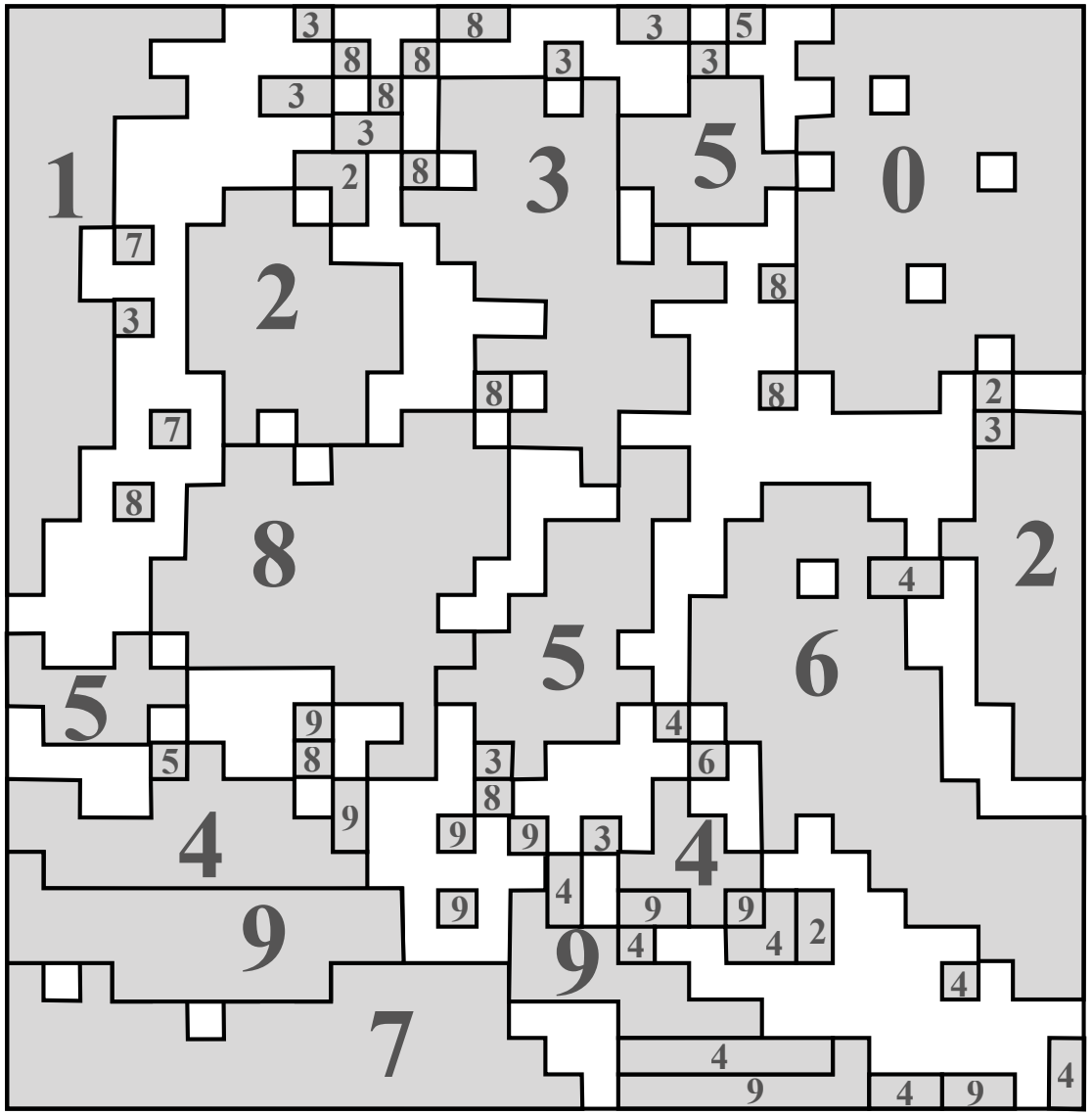
Non-linear dimensionality reduction.

Inputs: MNIST digits.

Iterate learning rule to reach steady state.

Then, for each input $\boldsymbol{x}^{(\mu)}$, find location of winning neuron in output array (the one with the largest output).

Winning neurons for same digits form clusters. Topographic map.



Human Genome Diversity Project

Home

Stanford HGDP SNP Genotyping Data

https://hagsc.org/hgdp

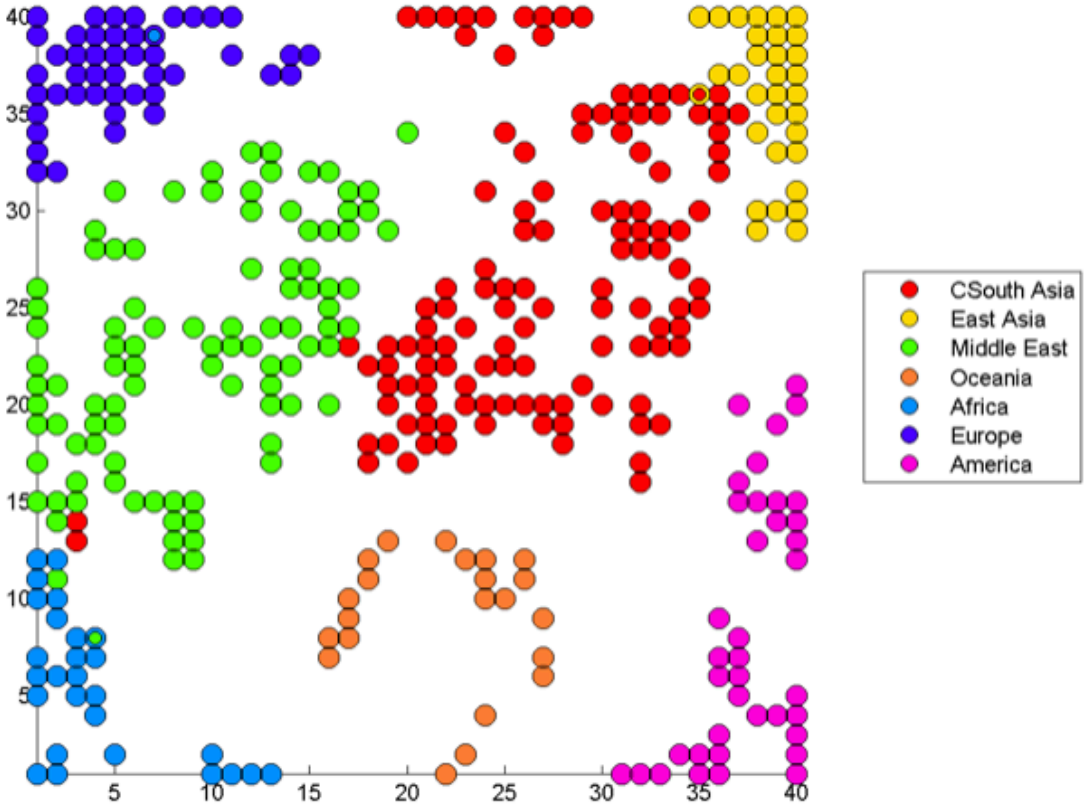
Genetic data (10^5 SNPs) from 1043 individuals from 51 different populations around the world.

1043 individuals

10^5 SNPs

	HGDP00448	HGDP00479	HGDP00985	HGDP01094	HGDP00982
rs10000543	CC	CC	TC	CC	CC
rs10000918	GG	AG	GG	GG	GG
rs10000929	AG	AA	AA	AG	AG
rs10001378	TT	CC	TT	CC	TT
rs10001548	TC	TT	TC	CC	TC
rs10002472	GG	AG	GG	GG	AA

Self-organising map creates a topo-praphic map, placing individuals who are closely related (few SNPs) in the same cluster.

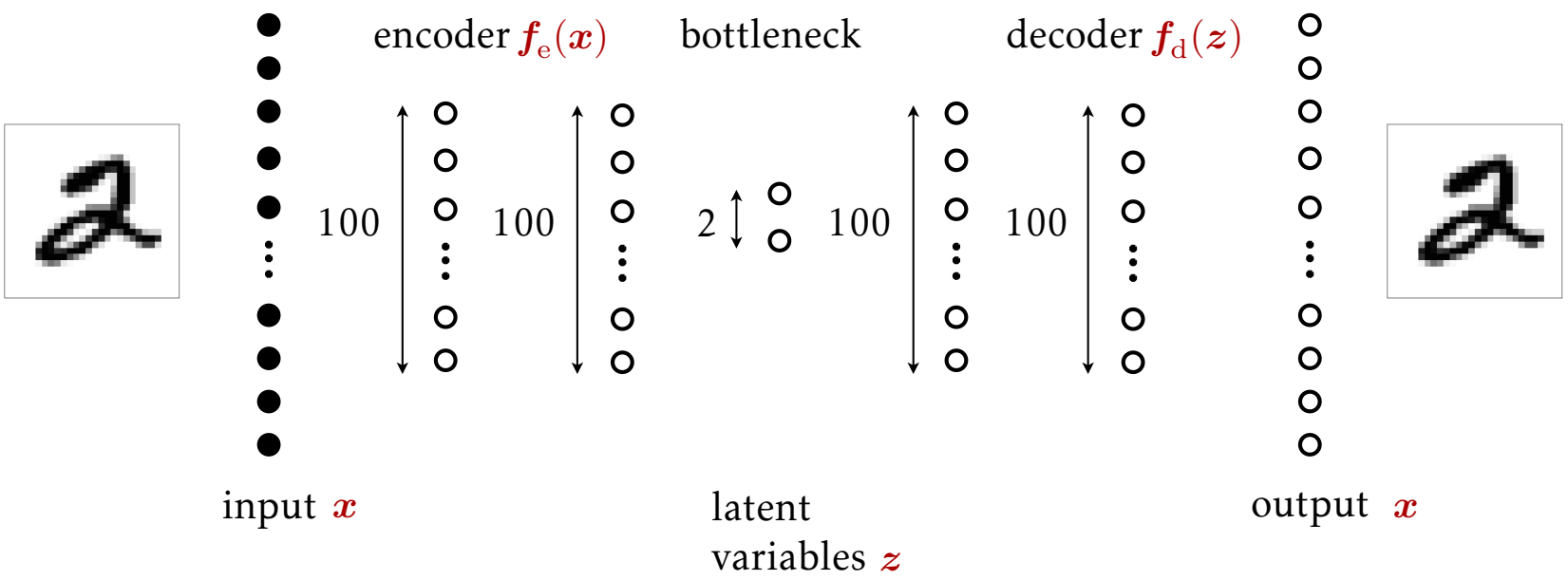


Eriksson, Loviken, Luketina & Mehlig (2014)

Genetic variation reflects Human demographic history.

Autoencoders

Idea: use inputs as targets, $\boldsymbol{t}^{(\mu)} = \boldsymbol{x}^{(\mu)}$



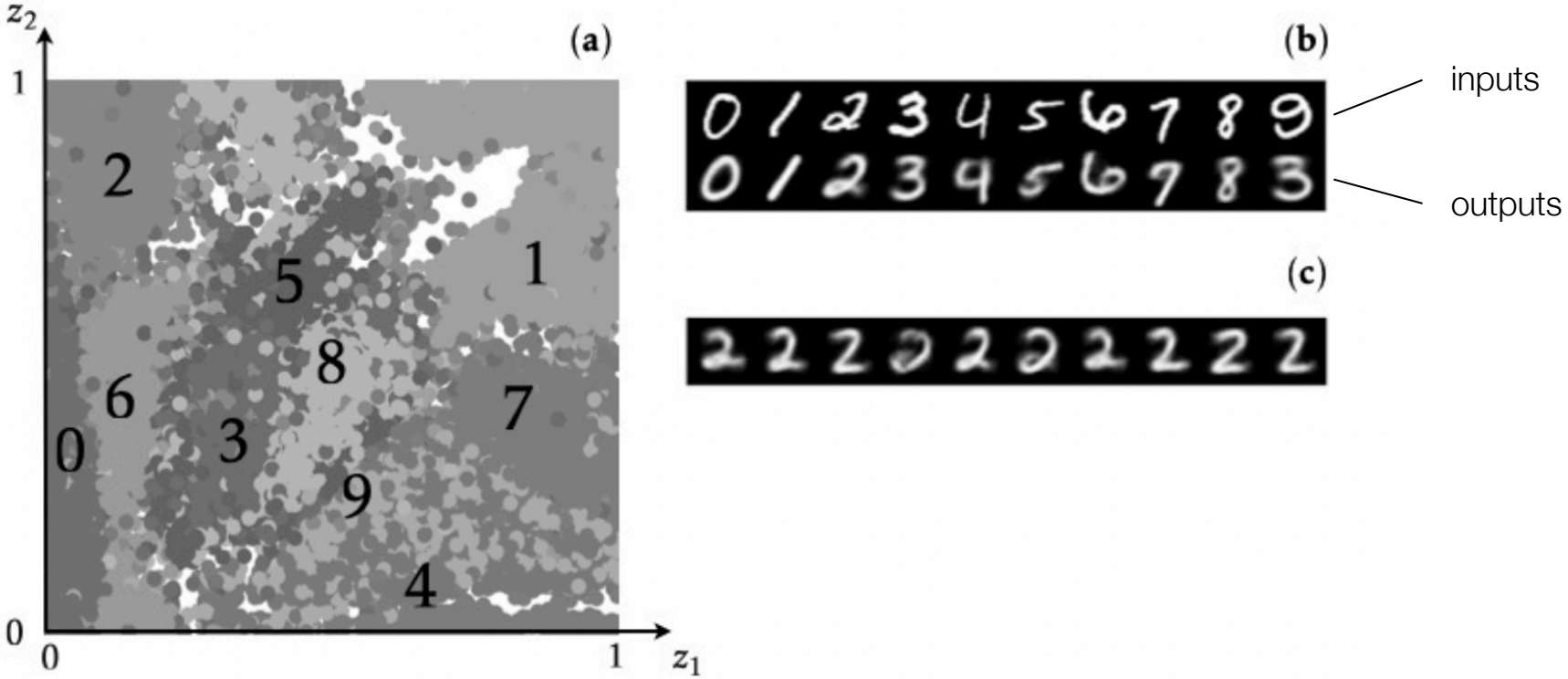
Encoder: $\boldsymbol{z} = \boldsymbol{f}_{\mathrm{e}}(\boldsymbol{x})$. Decoder: $\boldsymbol{x} = \boldsymbol{f}_{\mathrm{d}}(\boldsymbol{z})$. Energy function

$$H = \tfrac{1}{2} \sum_{\mu} \left| \boldsymbol{x}^{(\mu)} - \boldsymbol{f}_{\mathrm{d}}[\boldsymbol{f}_{\mathrm{e}}(\boldsymbol{x}^{(\mu)})] \right|^2 .$$

Minimise H . Expensive way of learning the identity function $\boldsymbol{x} - \boldsymbol{f}_{\mathrm{d}}[\boldsymbol{f}_{\mathrm{e}}(\boldsymbol{x})]$?

Autoencoders

Clustering of inputs in latent plane, panel (**a**).



Autoencoders are generative models, panel (**c**).

Reinforcement learning

Reinforcement learning

Sutton & Barto, *Reinforcement Learning: An Introduction*, MIT Press (2018)
Mehlig, *Machine learning with neural networks*, CUP (2021)

Supervised learning requires labelled data (targets $\boldsymbol{t}^{(\mu)}$)
Unsupervised learning does not need labels.

Reinforcement learning: only partial feedback in terms of a reward function, e.g.

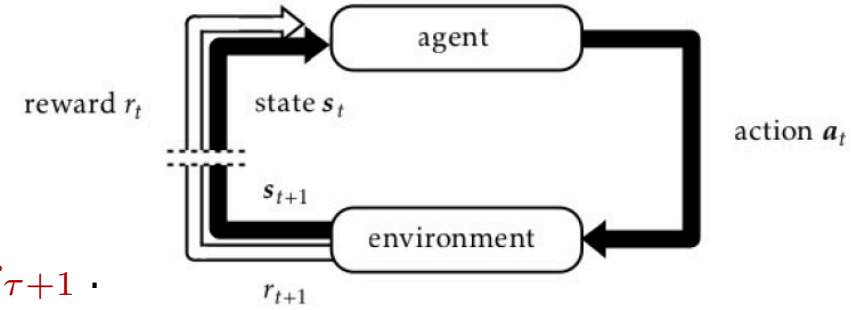
$$r = \begin{cases} +1 & \text{reward if all outputs correct} \\ -1 & \text{penalty otherwise} \end{cases}$$

(*immediate* reward)

Learning by trial and error.

Sequential decision process. Estimate expected *future* reward.

Agent explores a sequence of states $\boldsymbol{s}_0, \boldsymbol{s}_1, \boldsymbol{s}_2, \ldots$ through a sequence of actions $\boldsymbol{a}_0, \boldsymbol{a}_1, \boldsymbol{a}_2, \ldots$ and receives rewards $r_1, r_2, r_3, \ldots$.
Goal: estimate expected *future* reward $R_t = \sum_{\tau=t}^{T-1} r_{\tau+1}$.

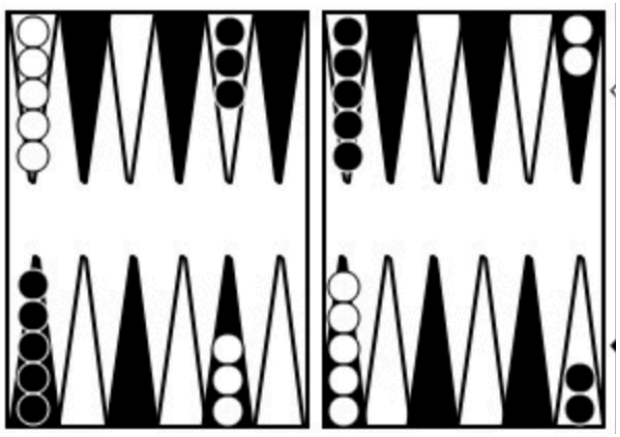


Method: iteratively improve estimate of expected future reward, given state $\boldsymbol{s}_t$ and action $\boldsymbol{a}_t$.

Backgammon

Tesauro, NIPS (1991)

Reinforcement learning allows computers to learn to play board games.



Program	Training Games	Opponents	Results
TDG 1.0	300,000	Robertie, Davis, Magriel	-13 pts/51 games (-0.25 ppg)
TDG 2.0	800,000	Goulding, Woolsey, Snellings, Russell, Sylvester	-7 pts/38 games (-0.18 ppg)
TDG 2.1	1,500,000	Robertie	-1 pt/40 games (-0.02 ppg)

Table 1. Results of testing TD-Gammon in play against world-class human opponents. Version 1.0 used 1-play search for move selection; versions 2.0 and 2.1 used 2-ply search. Version 2.0 had 40 hidden units; versions 1.0 and 2.1 had 80 hidden units.

Tesauro, Communications of the ACM (1995)

Practical Issues in Temporal Difference Learning

Gerald Tesauro
IBM Thomas J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598
tesauro@watson.ibm.com

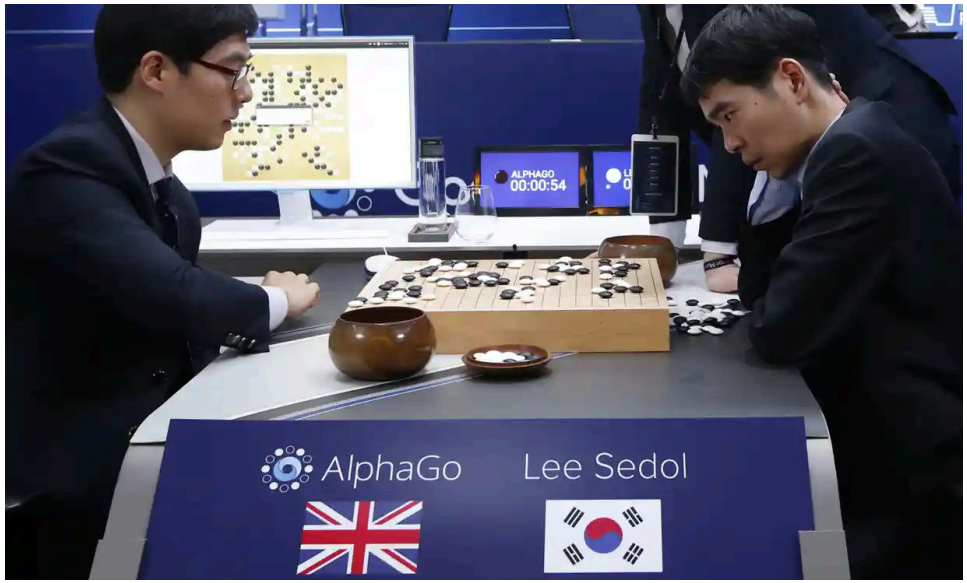
Abstract

This paper examines whether temporal difference methods for training connectionist networks, such as Suttons's TD(λ) algorithm, can be successfully applied to complex real-world problems. A number of important practical issues are identified and discussed from a general theoretical perspective. These practical issues are then examined in the context of a case study in which TD(λ) is applied to learning the game of backgammon from the outcome of self-play. This is apparently the first application of this algorithm to a complex nontrivial task. It is found that, with zero

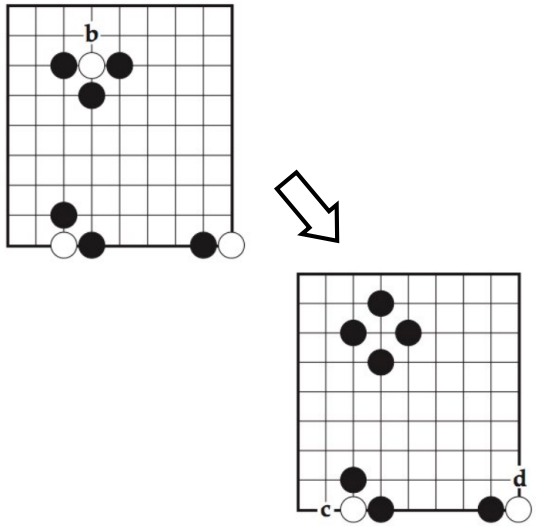
AlphaGo

Silver *et al.* Nature (2016,2017)

Reinforcement learning allows computers to learn to play board games.



Agents: two players,
Environment: the opponent,
States: board configurations,
Actions: moves,
Future reward: $r = +1$ (win),
$r = -1$ (lose).



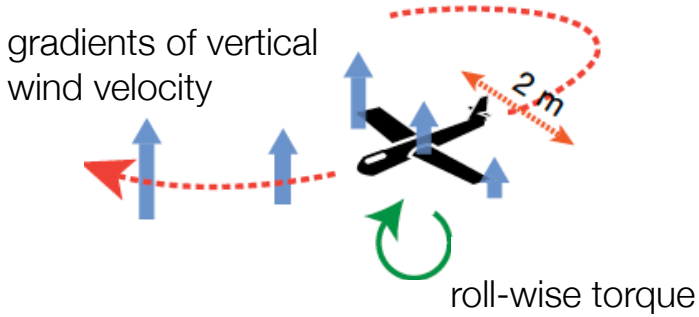
British GO association

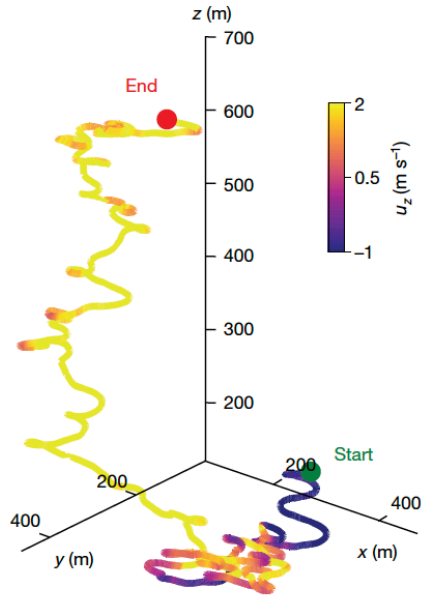
> In AlphaGo's case, that involved splitting itself in half and playing millions of matches against itself, learning from each victory and loss. In one day alone, AlphaGo was able to play itself more than a million times, gaining more practical experience than a human player could hope to gain in a lifetime. In essence, AlphaGo got better at Go simply by thinking extremely hard about the problem.
>
> Alex Hern, in: The Guardian (2016)

Control

Reddy, Wong-Ng, Celani, Sejnowski & Vergassola, Naturę (2018)

Control of soaring glider with reinforcement learning.





Different representations of actual glider trajectories. Left: soaring with learned strategy.

Associative reward-penalty algorithm

Behavioural sciences: learn to associate expected behaviour given certain stimuli.

Single stochastic neuron subject to stimuli $\boldsymbol{x}$

$$y = \begin{cases} +1 & \text{with probability } p(b)\,, \\ -1 & \text{with probability } 1-p(b)\,, \end{cases}$$

with $b = \boldsymbol{w}\cdot\boldsymbol{x}$ and $p(b) = (1+e^{-2b})^{-1}$.

Reward

$$r(\boldsymbol{x},y) = \begin{cases} +1 & \text{with probability } p_{\text{reward}}(\boldsymbol{x},y), \\ -1 & \text{with probability } 1-p_{\text{reward}}(\boldsymbol{x},y). \end{cases}$$

Learn to maximise reward by adjusting the weights.

Learning rule $\delta w_n = \eta r[y-\langle y\rangle]x_n$, related to Hebb's rule $\delta w_{mn} = \eta y_m x_n$

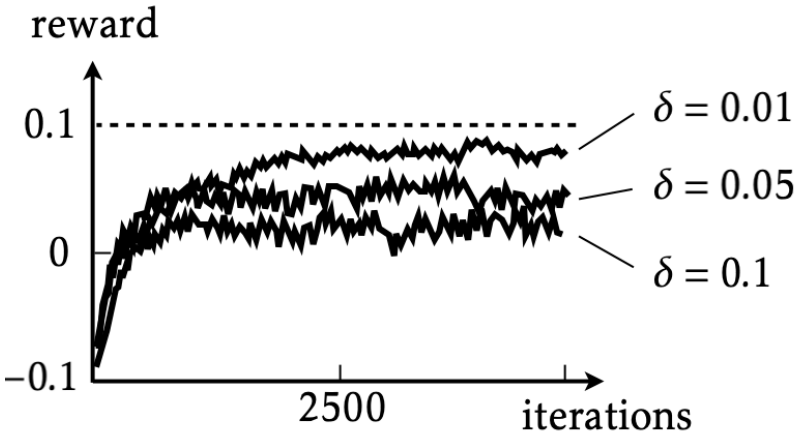
Better convergence with asymmetry, reduce weight update by factor $0<\delta<1$ when $r=-1$

Reward probabilities

p_{reward}	$y=-1$	$y=+1$
$\boldsymbol{x}^{(1)} = [1,0]^{\mathsf{T}}$	0.6	0.8
$\boldsymbol{x}^{(2)} = [1,1]^{\mathsf{T}}$	0.3	0.1

Maximal expected reward

$$r_{\max} = \tfrac{1}{2}\left[\langle r(\boldsymbol{x}^{(1)},+1)\rangle_{\text{reward}} + \langle r(\boldsymbol{x}^{(2)},-1)\rangle_{\text{reward}}\right]$$
$$= \tfrac{1}{2}[0.8-0.2+0.3-0.7] = 0.1$$

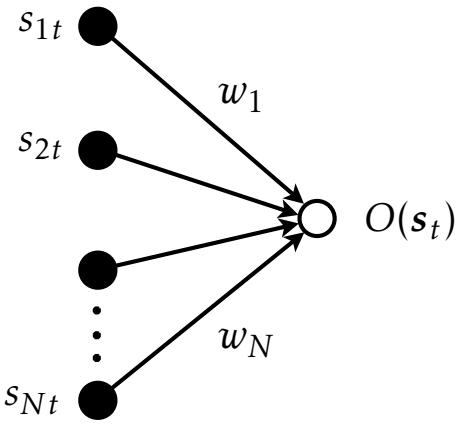


Temporal difference learning

Future reward $R_t = \sum_{\tau=t}^{T-1} r_{\tau+1}$.

Use neural network with input $\boldsymbol{s}_t$ (state) to estimate R_t ,

$$O(\boldsymbol{s}_t) = \boldsymbol{w} \cdot \boldsymbol{s}_t$$ (linear unit, weight vector $\boldsymbol{w}$)



Minimise energy function $H = \frac{1}{2} \sum_{t=0}^{T-1} [R_t - O(\boldsymbol{s}_t)]^2$ using gradient descent:

$$\delta w_m = \eta \sum_{t=0}^{T-1} [R_t - O(\boldsymbol{s}_t)] \frac{\partial O}{\partial w_m}$$

Trick: express error $R_t - O(\boldsymbol{s}_t)$ as sum over temporal differences

$$R_t - O(\boldsymbol{s}_t) = \sum_{\tau=t}^{T-1} [r_{\tau+1} + O(\boldsymbol{s}_{\tau+1}) - O(\boldsymbol{s}_\tau)] \quad \text{with} \quad O(\boldsymbol{s}_T) \equiv 0$$

.

Temporal difference learning

Insert this expression for $R_t - O(\boldsymbol{s}_t)$ into the gradient-descent rule:

$$\delta \boldsymbol{w} = \eta \sum_{t=0}^{T-1} \sum_{\tau=t}^{T-1} [r_{\tau+1} + O(\boldsymbol{s}_{\tau+1}) - O(\boldsymbol{s}_\tau)] \boldsymbol{s}_t$$

Terms in this double sum can be summed in a different way

$$\delta \boldsymbol{w} = \eta \sum_{\tau=0}^{T-1} \sum_{t=0}^{\tau} [r_{\tau+1} + O(\boldsymbol{s}_{\tau+1}) - O(\boldsymbol{s}_\tau)] \boldsymbol{s}_t$$

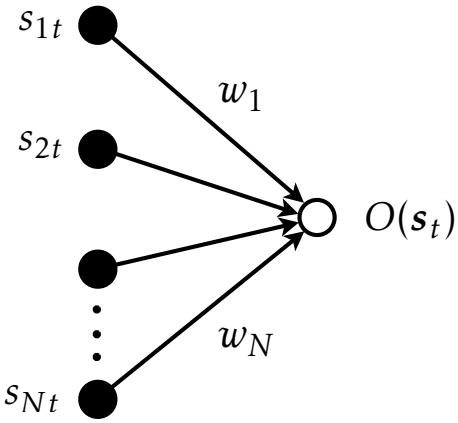
Exchange summation variables and add weights λ:

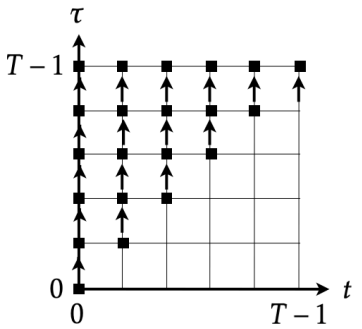
$$\delta \boldsymbol{w} = \eta \sum_{t=0}^{T-1} [r_{t+1} + O(\boldsymbol{s}_{t+1}) - O(\boldsymbol{s}_t)] \sum_{\tau=0}^{t} \lambda^{t-\tau} \boldsymbol{s}_\tau$$

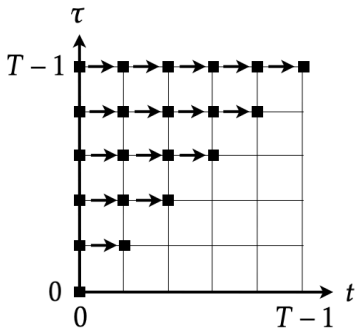
Alternative: update $\boldsymbol{w}$ (and hence O) at every time step:

$$\delta \boldsymbol{w}_t = \eta [r_{t+1} + O(\boldsymbol{w}_t, \boldsymbol{s}_{t+1}) - O(\boldsymbol{w}_t, \boldsymbol{s}_t)] \sum_{\tau=0}^{t} \lambda^{t-\tau} \boldsymbol{s}_\tau$$

This is the temporal difference learning rule TD(λ).







SARSA

Temporal difference learning TD(λ)

$$\delta \boldsymbol{w}_t = \eta[r_{t+1} + O(\boldsymbol{w}_t, \boldsymbol{s}_{t+1}) - O(\boldsymbol{w}_t, \boldsymbol{s}_t)] \sum_{\tau=0}^{t} \lambda^{t-\tau} \boldsymbol{s}_\tau$$

The TD(0)-rule corresponds to the following learning rule for the network output

$$O_{t+1}(\boldsymbol{s}_t) = O_t(\boldsymbol{s}_t) + \eta[r_{t+1} + O_t(\boldsymbol{s}_{t+1}) - O_t(\boldsymbol{s}_t)]$$

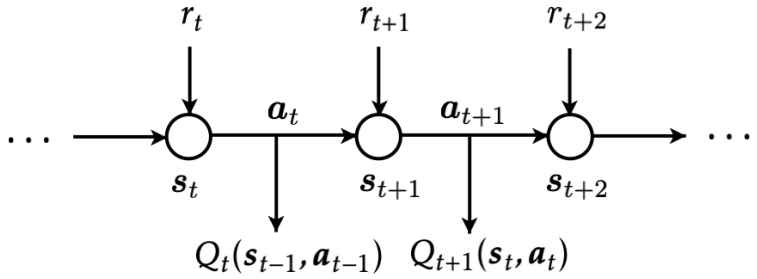
For a sequential decision process, estimate the expected future reward given $\boldsymbol{s}_t$ and $\boldsymbol{a}_t$

$$Q_{t+1}(\boldsymbol{s}_t, \boldsymbol{a}_t) = Q_t(\boldsymbol{s}_t, \boldsymbol{a}_t) + \eta\left[r_{t+1} + Q_t(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) - Q_t(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]$$

To update one needs $\boldsymbol{s}_t$, $\boldsymbol{a}_t$, r_{t+1} , $\boldsymbol{s}_{t+1}$, $\boldsymbol{a}_{t+1}$ (SARSA).

Problem: iteration depends upon policy for how to choose the next action, $\boldsymbol{a}_{t+1}$.
Greedy policy: choose the action one with largest $Q_t(\boldsymbol{s}_t, \boldsymbol{a}_t)$.
Stochastic policy: mainly greedy, but sometimes do something else.

Explore-versus-exploit dilemma.



Q-learning

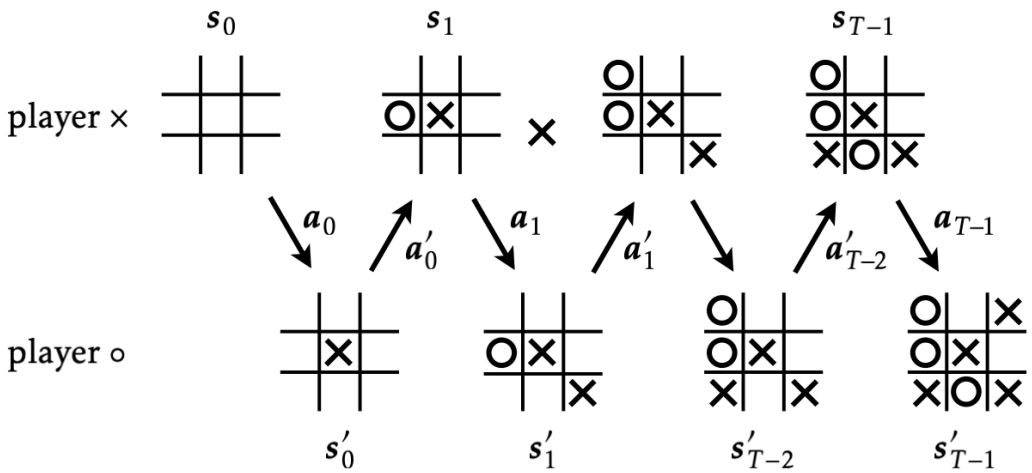
The Q-learning rule is an approximation to SARSA that does not depend on $\boldsymbol{a}_{t+1}$.

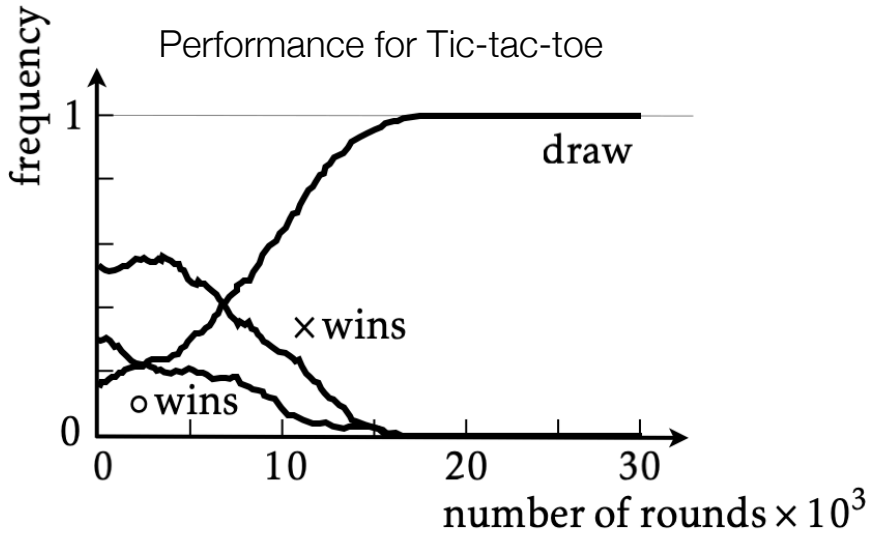
Instead one assumes that the next action, $\boldsymbol{a}_{t+1}$, is the optimal one, regardless of the policy that is currently followed:

$$Q_{t+1}(\boldsymbol{s}_t, \boldsymbol{a}_t) = Q_t(\boldsymbol{s}_t, \boldsymbol{a}_t) + \eta\left[r_{t+1} + \max_{\boldsymbol{a}} Q_t(\boldsymbol{s}_{t+1}, \boldsymbol{a}) - Q_t(\boldsymbol{s}_t, \boldsymbol{a}_t)\right]$$

Q-learning is simpler than SARSA, but approximate.

Learn to play tic-tac-toe: $r = +1$ (win), $r = 0$ (draw), $r = -1$ (lose).





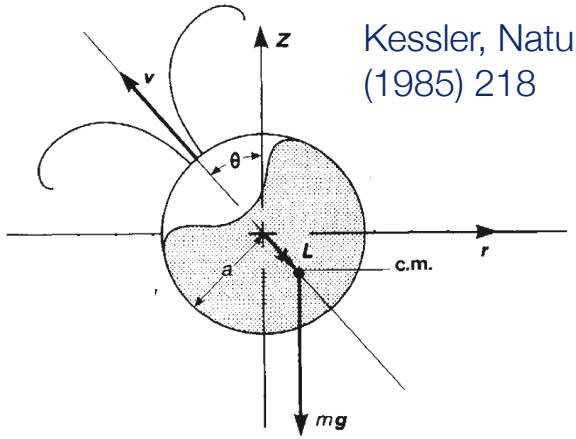
Motile micro-organisms

Motile micro-organisms in the ocean react to sensory input.

Copepods jump to escape predator

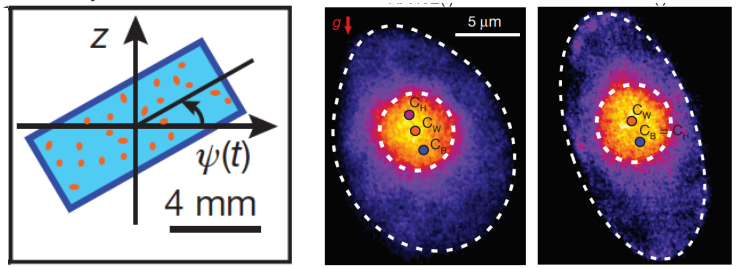
Stefan Di Criscio (2012) youtube.com

Gyrotaxis for vertical migration



Kessler, Nature **313** (1985) 218

Shape change: bet-hedging in a turbulent eddy



Sengupta, Carrara & Stocker, Nature **543** (2017) 555

Chain formation for vertical migration

Lovecchio, Climent, Stocker & Durham, Science Advances **5** (2019) 2375

Mechanism:

Gustavsson, Berglund, Jönsson & Mehlig, Phys. Rev. Lett. **116** (2016) 108104

Reinforcement learning for microswimmers

Optimal strategy not obvious. Depends on the *signals* the organism picks up as it moves (strain, vorticity, fluid acceleration), and upon how it may *respond*.

Use machine-learning to find optimal strategies for selected targets. Goals:
- -understand which selective pressures matter for evolution
- -improve models for motile microswimmers
- -optimise smart swimmers to perform certain tasks

Phytoplankton in the ocean: maximise upward swimming velocity to reach water surface.
Colabrese, Gustavsson, Celani & Biferale, Phys. Rev. Lett. **118** (2017) 158004

Zooplankton (copepods): jump to minimise risk of predation.
Ardeshiri, Schmitt, Souissi, Toschi & Calzavarini, J. Plankton. Res. **39** (2017) 878

Control problem: minimise time for active particle to move from A to B in turbulent flow. Control variable: steering angle.
Buzzicotti, Biferale, Bonaccorse, Clark Di Leoni & Gustavsson (2019)

Responses: change buoyancy, shape, internal mass distribution, swimming speed & stroke...

Symmetries

Qiu, Mousavi. Gustavsson, Xu, Mehlig & Zhao, arxiv:2104.11303 (2021)

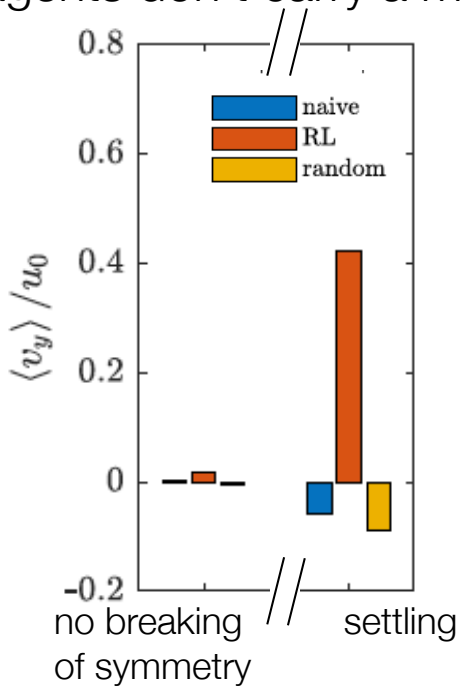
Proof-principle-studies show that reinforcement learning finds superior strategies.

Colabrese, Gustavsson, Celani & Biferale, Phys. Rev. Lett. **118** (2017) 158004
Buzzicotti, Biferale, Bonaccorse, Clark Di Leoni & Gustavsson, Chaos **29** (2019) 103138
Alegashan, Verma, Bec & Pandit, Phys. Rev. E **101** (2020) 043110
Schneider & Stark, Europhys. Lett. **127** (2019) 34003
Gunnarson, Mandralis, Novati & Koumoutsakos, arxiv:2102.10536 (2021)
Muinos-Landin, Fischer, Holubec & Cichos, Science Robotics **6** (2021)

Agents had access to global information in lab frame. But usually agents don't carry a map: signals & actions are local.

To learn good strategies for vertical migration in isotropic flow requires symmetry breaking in signals and/or actions.

Example: settling allows upward navigation in steady isotropic flow using only local signals (strain) and actions (local steering).



Black box?

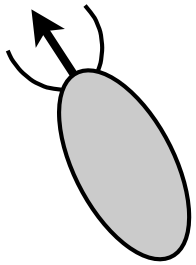
Qiu, Mousavi. Gustavsson, Xu, Mehlig & Zhao, arxiv:2104.11303 (2021)

Reinforcement learning gives interpretable results.

Q-matrix allows to infer mechanism that underlies optimal strategy.

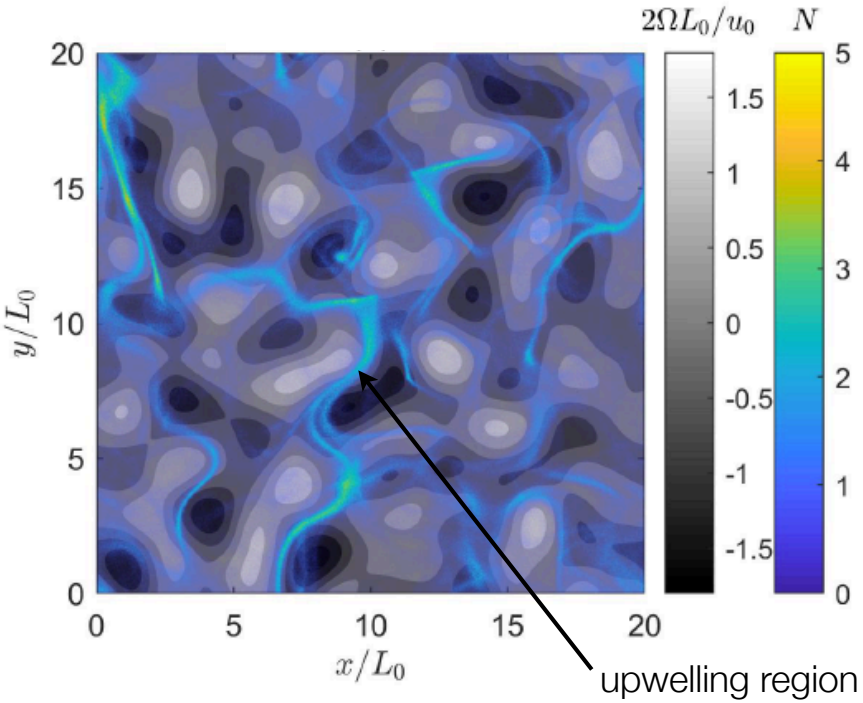
Previous example: swimmer navigating two-dimensional steady isotropic flow with local signals and actions. Symmetry breaking due to gravity: settling & gyrotaxis.

Swimmer measures local strain.



It learns to behave like a slender passive, bottom-heavy particle that tends to sample upwelling regions of the flow that transport the particle upwards.

Gustavsson, Berglund, Jönsson & Mehlig, Phys. Rev. Lett. **116** (2016) 108104



Challenges

Curse of dimensionality (Q learning inefficient if number of state-action pairs too large)
-resolved by deep reinforcement learning (old idea)?
-use symmetries to reduce size of Q matrix?

Interpretation (how to extract most important and robust features of best strategy)
-general method?

Signals
-how does organism perceive the flow?
-which signals are most important for given reward function?
-which signals can micro-robot measure reliably?

Redaelli, Candelier, Mehaddi & Mehlig, arxiv:2105.01408 (2021)

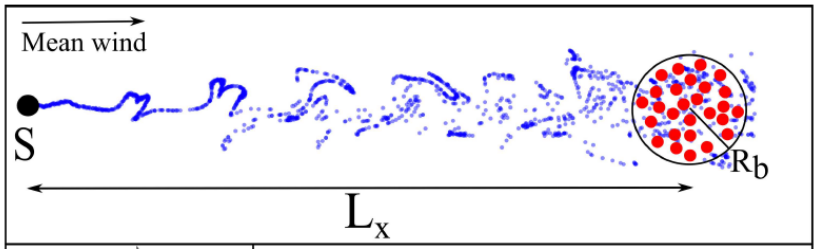
Reward function
-reward based on mean values doesn't allow to learn to respond to rare events
-competing rewards (minimise energy consumption, avoid predation)

Convergence
-convergence proofs of Q-learning rely on Markovian dynamics. Usually not Markovian.

Beyond single swimmers

Collective search strategies

-collective olfactory search in turbulence (heuristic model)



Durve, Piro, Cencini, Biferale, Celani, Phys. Rev. E **102** (2020) 012402

Optimal strategies for individuals of different species to avoid **competition for resources**?

-evolutionary branching Sagitov, Mehlig, Jagers & Vatutin, Theor. Pop. Biol. 83 (2013) 145

Population genetics

Ravinet, Faria, Butlin, Galindo, Bierne, Rafajlovic, Noor, Mehlig & Westram, J. Evol. Biol. **30** (2017) 1450

-competition/speciation/game theory

-remains to be seen whether reinforcement learning can inform genetics/evolution

Multi-agent reinforcement learning

-efficient strategies for distributed search in fluctuating environments

Busonio, Babuska & Schutter, IEEE Transactions on systems, Man & Cybernetics C: Applications and Reviews **38** (2008)

Conclusions

Supervised learning with artificial neural networks was studied in the 40ies, 60ies, 80ies.
In the last few years: *deep-learning revolution* due to

- better training sets
- better hardware (GPUs, dedicated chips)

Unsupervised learning: learning without labels. Requires redundancy.

- clustering of high-dimensional data
- topographic maps
- non-linear projections
- self-organising maps in the visual cortex?

Reinforcement learning was developed in the 80ies and 90ies.
In the last few years: success due to better hardware and deep Q-learning.

- AlphaGo
- Multi-agent reinforcement learning