**Background on Machine learning**

Gaining popularity in research and being the centre of technological advancements in all domains, machine learning is a field that focuses on providing data-driven, intelligent answers to research questions. We generally divide it into Supervised Learning, Unsupervised Learning, and Reinforcement Learning.

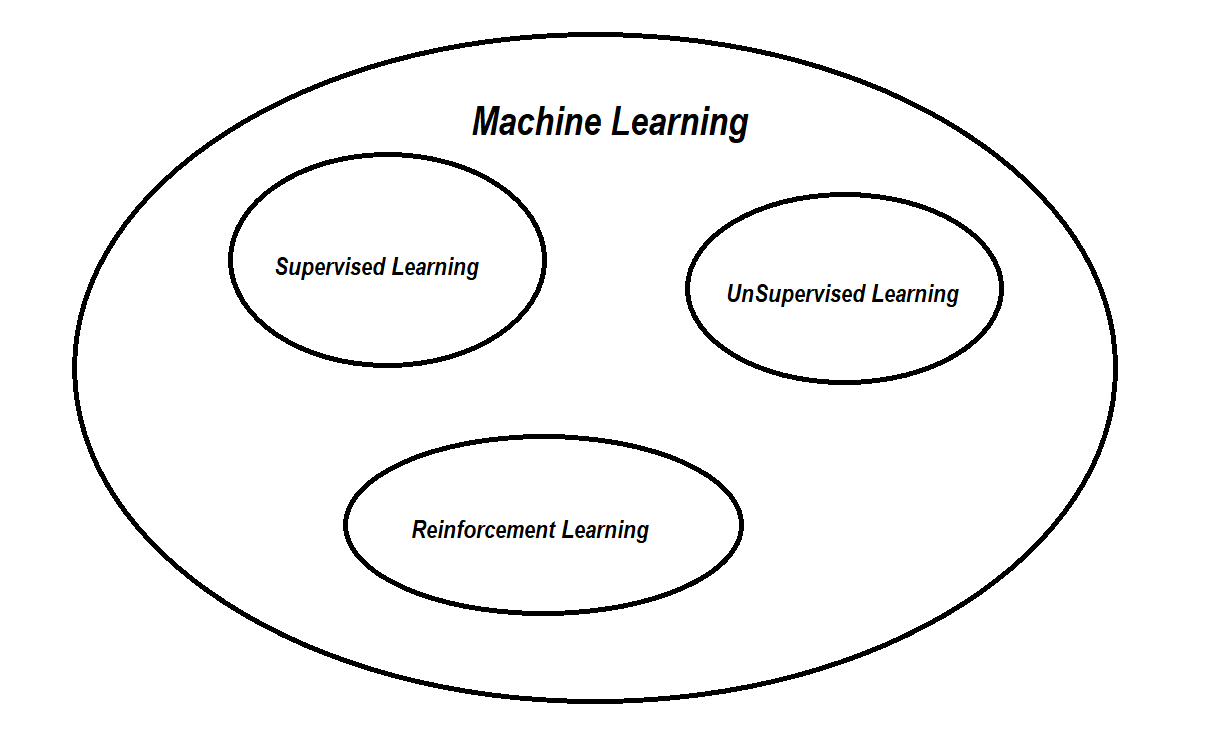


Fig 2.1

Supervised Learning (SL) is the domain where we try to map inputs to outputs using labelled data. The SL task is either a classification task or a regression task. In classification, we use features of a particular object in order to predict its class (i.e. classifying a malignant or benign tumour). In regression where we use features of an object to try to predict another feature of that object, i.e. predicting the price of a house using features of size, location.

Unsupervised Learning (UL) uses unlabeled data in order to divide it into an unknown set of classes, extract specific structures from the data …etc. We divide UL into Parametric UL where we assume probabilistic distribution of the data based on specific parameters, and the mission is to try and get those parameters so that we can predict the future. The other field of UL is Non-Parametric UL. Here, we make no assumptions about the data, and we only group the data into clusters with resembling features.

Although a large number of problems can be solved using these two methods of learning, the real power and research start when we start talking about the opportunities Artificial Neural Networks ANNs provide. These are trials of modelling the process of thinking that lies inside a human brain. A neural net is a collection of small processing units called neurons that receive input, use its predefined function to calculate a result then output this result to another neuron or the user. Collecting a group of neurons, we create a neural net that can take multiple inputs and use complex functions to get a single output. We call each level of neurons a layer; we have an input layer, output, hidden layers.

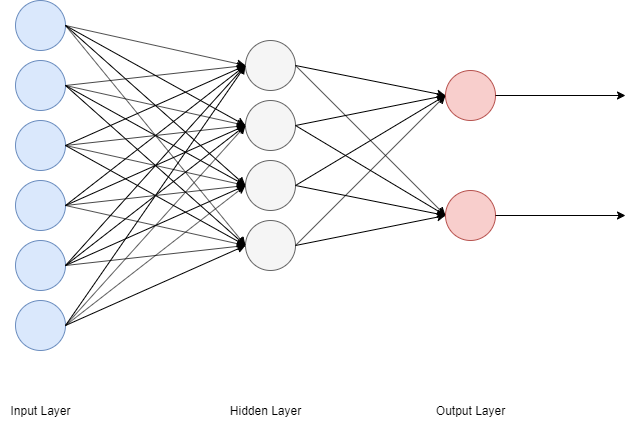


Fig 2.2

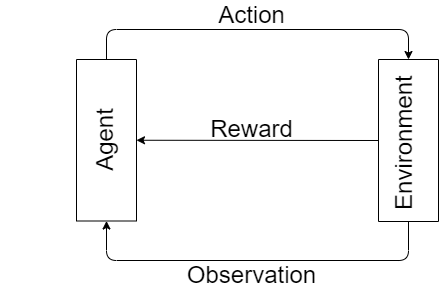
Deep Learning DL is an application on ANNs where we have many levels of hidden layers typically more than three. We use this architecture to solve problems with increasing levels of complexity and individual requirements. The most famous divisions of deep learning are Convolutional Neural Networks CNNs and Recurrent Neural Networks RNNs. CNNs use filters to tackle problems in relation to pictures and images. These problems include object detections, image classification and much more in-depth domain problems. The focus of RNNs is time-varying data, i.e. data that require time-based context to solve. The most important application of this is Natural Language Processing NLP which needs text context data to get general speech understanding.

**Reinforcement Learning Introduction**

Reinforcement Learning RL is the third type of machine learning domain. It is learning what to do so to maximize a digital reward signal [1]. The problem is as one where an agent (i.e. a player in a game) is traversing an environment, and he takes actions and collects rewards as he goes. We can describe the whole problem as an environment env that can be described as a state-space that consists of states s that describe fully the world that can affect or be affected by the decisions made by the agent.

The agent can take an action from an action space where *a* that will change its state and receive a reward *r* where *r*.

This Mathematical representation means that an RL problem can be described as a Markov Decision Process MDP. In an MDP we have the concepts of a reward function which can map states to rewards achieved when reaching that state. In an episodic process (one with a clear starting state and a terminal state) the total reward is the accumulation of each reward received through the journey traversing the environment until reaching the terminal state. This total reward is the value we are trying to maximize in our RL problem.



Another concept apparent in MDPs is the concept of a policy which defines the path (also known as trajectory *τ*) that the agent will take during the episode. What the policy maps are state action pairs i.e. what action to take when an agent is in state s. There are two types of policies, either a deterministic policy; one that the agent is told exactly what action to make when arriving at state s where a. The other type of policy is a stochastic policy; here the agent is told in probabilistic values the probability of taking action a when in state s where

a|sa|s.

Achieving the maximum reward possible implies that we will take the best possible policy that will give the maximum reward at each step. This policy is called the optimal policy and accordingly, we can say that the goal of RL is to find the optimal policy for the environment that we work in. Finding this optimal policy requires a way to measure the optimality or goodness of a certain policy and this can be done using a Value Function s which is the total amount of reward an agent is expected to accumulate over the future starting at current state s [1]. This can be expressed as

ss

where s is the set of rewards expected to be accumulated starting at state s and following policy ח until the end of the episode .

Using the definition of return which is given by:

We could re-write s as

Representing the environment is the model, which is a mapping of what the environment behaviour in response to the agent’s action with describing the probability of arriving at state s’ when taking action a at state s and being the expected reward for arriving at state s taking action a described as follows

a

and

a.

Following, the optimal value function of a state s following a policy in a number of steps H is given by

ss

and the process of finding the best possible value function of a given policy and then updating the policy in order to find the best policy is called the policy iteration and it can be given by the algorithm (Value Update or Bellman Update Backup):

Start with ;

For k = 1, 2, …, H:

For all states s in :

One alternative to using these dynamic programming methods of value iteration and policy iteration is to introduce a way to learn as the episode is going, using an entire set of visits over states to come up with the value function of each state. These methods are called Monte-Carlo methods and their basic concept is that the value of a state is calculated using a number of visits to the state in the episode and total return on that state by the following algorithm:

To evaluate s

Each timestep t when s is visited in an episode:

Increment counter

Increment total return

Value is given by

With

Using this method and generalizing overall episodes:

For all states with return :

For non-stationary environments:

2.x

**Cross-Entropy Methods**

One of the most basic solution methods when talking RL problems is cross-entropy methods. Their idea is pretty simple and builds on the intuition gained from looking at the RL main problem. The idea of gaining as much reward as possible used as we replace all the agent complications with a non-linear trainable NN function, with the input being the observations from the environment and the output being the policy . In practice, we represent the policy as a probability distribution over the actions making the problem a classification problem. The algorithm describing the method is:

1. Play N number of episodes with initial NN model and environment.
2. Calculate the total reward for each episode and set a reward boundary usually at the 70th percentile of rewards.
3. Discard all rewards below the boundary.
4. Train NN model on elite episodes using state as input and used actions as a target.
5. Repeat 1 until target mean reward is reached.

**Q-Learning**

Another value that’s important do describe an RL environment is the Q-value which describes the quality of taking a certain action a when in state s following policy ח and is given by

Finding a solution to this value is inherently a temporal difference problem, which can be defined as a combination between Monte Carlo and Dynamic Programming in which we can learn directly from experience without needing a dynamics model (MC) and we update estimates based in part by other learned without waiting for final estimates. Working from equation 2.x, TD doesn’t wait for a whole episode to update , it only waits for the next time step t+1 to update the value function using both updating on the transition to using the equation

This is called one-step or , a special version of the complete or n-step . The algorithm for implementing it is

Loop for each episode:

Initialize S:

Loop for each step of the episode:

Until is terminal

Q-learning is an off-policy algorithm [2] which made a major breakthrough in RL in which we use the update 2.xx in the normal algorithm.

2.xx

**Deep Reinforcement Learning**

These techniques work on using neural networks to approximate either policy parameters or to get V, Q and A. In value function Deep-RL methods, more specifically DQNs we can look at the problem as one of regression, using a deep NN to approximate the value of Q function, updating our Q value using the tabular Q learning update:

When using this update in practice, we face some problems that limit the usability of the method. The problem of acting randomly or using the Q approximation is solved using an epsilon-greedy method as we start with a completely random action and decaying the randomness with probability ε till a small value of 2% randomness. Another problem is the requirement of SGD on training data, as it requires independent and identically distributed data i.i.d, this is not satisfied. We solve this problem by using a large replay buffer, adding new experiments, and pushing old ones out, allowing for independent data that are fresh. The last problem is the similarity between states after each other, not allowing our NN to distinguish between the states. We will solve this by using a target network, keeping a copy of the NN parameters using it for , this network is updated periodically making the network stable.

Our algorithm for DQN is:

Initialize , with random weights, , and an empty replay buffer

Choose random action with probability ;otherwise,

Execute action , observe next state and reward

Store transition in replay buffer

Sample a random mini-batch of transitions from the replay buffer

For every transition in the buffer, calculate target if the episode has ended at this step and otherwise

Calculate loss

Update using SGD

Every N steps copy parameters from

Repeat 2 till convergence

**Policy Gradient Methods**

These can learn a parametrized policy that can select an action without returning to a value function, it can be used to learn policy parameters but not to select exact actions. Updating the parameters of the policy can be achieved either using gradient-based methods or gradient-free methods with the aim to maximize expected return. Policy gradients update the policy parameter on each step in the direction of an estimate of the gradient of performance with respect to the policy parameter. Given the trajectory which is the set of state action reward sequence, we can define a policy parameter performance as

2.xx1

It’s obvious that we need to maximize in order to find the optimal . Using gradient descent on this problem (the most basic machine learning idea) we find that

The far right of the equation can be approximated to

Using an expansion for we can arrive at

This tells us that in order to successfully update the parameters we don’t need a model; we can use the state-action rewards only to make a useful update. We can solve the expectation in the equation by sampling a large number of trajectories while replacing removes the variance that appears in the normal equation. The baseline b is used to reduce the bias of the estimate, b must be independent of the parameters and a good baseline makes use of the state-value current state. Equation 2.xxx without adding baseline is called REINFORCE which is the classic policy gradient algorithm.

Actor-critic methods combine policy gradients with model fitting, we use an actor to model the policy and a critic to model the value function V. By introduce a critic, we reduce the number of samples to collect for each policy update, we don’t collect all samples until the end of an episode.

We will talk about these specific solution methods and more state-of-the-art solutions in the next chapter where we explain the methodology of our work.

1. Sutton S. Richard, Barto G. Andrew Reinforcement Learning an Introduction 2nd edition 2018
2. Watkins, 1989
3. W. Saad, Z. Han, H. V. Poor, and T. Baar, “A Noncooperative Game  
   for Double Auction-Based Energy Trading Between PHEVs and Distribution Grids,” 2011 IEEE International Conference on Smart Grid  
   Communications, pp. 267–272, 2011.
4. B. Ramachandran, S. K. Srivastava, C. S. Edrington, and D. A. Cartes,  
   “An Intelligent Auction Scheme for Smart Grid Market Using a Hybrid Immune Algorithm,” IEEE Transactions on Industrial Electronics,  
   vol. 58, no. 10, pp. 4603–4612, 2011.
5. J. Matamoros, D. Gregoratti, and M. Dohler, “Microgrids Energy  
   Trading in Islanding Mode,” 2012 IEEE 3rd International Conference  
   on Smart Grid Communications, pp. 49–54, 2012.
6. S. Chen, N. B. Shroff, and P. Sinha, “Energy Trading in the Smart Grid:  
   From End-user’s Perspective,” Asilomar Conference on Signals, Systems  
   and Computers, pp. 1–9, 2013
7. S. Park, J. Lee, G. Hwang, and J.K. Choi,” Event-Driven Energy Trading System in Microgrids: Aperiodic Market Model Analysis with a Game Theoretic Approach” IEEE Access, vol 5, pp. 26291-26302, Nov. 2017
8. J. Lee, J. Guo, J.K. Choi, and M. Zukerman “Distributed Energy Trading in Microgrids: A Game-Theoretic Model and Its Equilibrium Analysis” IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, Nov. 2014
9. W. Saad, Z. Han, and H. V. Poor, “Coalitional game theory for cooperative micro-grid distribution networks,” in Proc. IEEE ICC, Kyoto,  
   Japan, Jun. 2011, pp. 1–5.
10. J. Matamoros, D. Gregoratti, and M. Dohler, “Microgrids energy trading  
    in islanding mode,” in Proc. IEEE SmartGridComm, Tainan, Taiwan,  
    Nov. 2012, pp. 49–54.
11. G. E. Asimakopoulou, A. L. Dimeas, and N. D. Hatziargyriou, “Leaderfollower strategies for energy management of multi-microgrids,” IEEE  
    Trans. Smart Grid, vol. 4, no. 4, pp. 1909–1916, Dec. 2013.
12. H. S. V. S. K. Nunna and S. Doolla, “Multi agent-based distributed energy-resource management for intelligent microgrids,” IEEE Trans.  
    Ind. Electron., vol. 60, no. 4, pp. 1678–1687, Apr. 2013
13. S. M. Ali, “Electricity trading among microgrids,” Master’s thesis, The  
    University of Strathclyde, Glasgow, UK, 2009. [Online]. Available:  
    [http://www.esru.strath.ac.uk/Documents/MSc 2009/Ali.pdf](http://www.esru.strath.ac.uk/Documents/MSc%202009/Ali.pdf)
14. W. Saad, Z. Han, and H. V. Poor, “Coalitional game theory for cooperative micro-grid distribution networks,” in Proc. IEEE ICC, Kyoto,  
    Japan, Jun. 2011, pp. 1–5.
15. J. Matamoros, D. Gregoratti, and M. Dohler, “Microgrids energy trading  
    in islanding mode,” in Proc. IEEE SmartGridComm, Tainan, Taiwan,  
    Nov. 2012, pp. 49–54.
16. G. E. Asimakopoulou, A. L. Dimeas, and N. D. Hatziargyriou, “Leaderfollower strategies for energy management of multi-microgrids,” IEEE  
    Trans. Smart Grid, vol. 4, no. 4, pp. 1909–1916, Dec. 2013.
17. H. S. V. S. K. Nunna and S. Doolla, “Multi agent-based distributed energy-resource management for intelligent microgrids,” IEEE Trans.  
    Ind. Electron., vol. 60, no. 4, pp. 1678–1687, Apr. 2013
18. [16] H. M. Soliman and A. Leon-Garcia, “Game-Theoretic Demand-Side  
    Management with Storage Devices for the Future Smart Grid,” IEEE  
    Transactions on Smart Grid, vol. 5, no. 3, pp. 1475–1485, 2014.
19. W. Tushar, J. A. Zhang, D. B. Smith, H. V. Poor, and S. Thiebaux, ´  
    “Prioritizing Consumers in Smart Grid: A Game-Theoretic Approach,”  
    IEEE Transactions on Smart Grid, vol. 5, no. 3, pp. 1429–1438, 2014.
20. J. Lee, J. Guo, J. K. Choi, and M. Zukerman, “Distributed Energy  
    Trading in Microgrids: A Game-Theoretic Model and Its Equilibrium  
    Analysis,” IEEE Transactions on Industrial Electronics, vol. 62, no. 6,  
    pp. 3524–3533, 2015.
21. H. Wang and J. Huang, “Incentivizing energy trading for interconnected  
    microgrids,” IEEE Trans. Smart Grid, 2016, in press.
22. N. Yaagoubi and H. T. Mouftah, “A Distributed Game-Theoretic  
    Approach to Energy Trading in the Smart Grid,” 2015 IEEE Electrical  
    Power and Energy Conference (EPEC), pp. 203–208, 2015
23. L. Zhang, Z. Li, and C. Wu, “Randomized auction design for electricity  
    markets between grids and microgrids,” in Proc. ACM SIGMETRICS,  
    Austin, TX, USA, 2014, pp. 99–110
24. S. Maharjan, Q. Zhu, and Y. e. a. Zhang, “Dependable demand response  
    management in the smart grid: A Stackelberg game approach,” IEEE  
    Trans. Smart Grid, vol. 4, no. 1, pp. 120–132, Mar. 2013
25. B.-G. Kim, Y. Zhang, and M. e. a. van der Schaar, “Dynamic pricing  
    and energy consumption scheduling with reinforcement learning,” IEEE  
    Trans. Smart Grid, vol. 7, no. 5, pp. 2187–2198, Nov. 2016
26. Y. Zhang and M. van der Schaar, “Structure-aware stochastic storage  
    management in smart grids,” IEEE Journal of Selected Topics in Signal  
    Processing, vol. 8, no. 6, pp. 1098–1110, Dec. 2014.
27. Z. Wen, D. O’Neill, and H. R. Maei, “Optimal demand response using  
    device-based reinforcement learning,” abs/1401.1549, 2014. [Online].  
    Available: I://arxiv.org/abs/1401.1549
28. E. Kara, M. Berges, B. Krogh, and S. Kar, “Using smart devices for  
    system-level management and control in the smart grid: A reinforcement  
    learning framework,” in IEEE SmartGridComm, 2012
29. X.Wang, M. Zhang, F. Ren, and T. Ito,” GongBroker: A Broker Model for Power Trading in Smart Grids Markets “, International Conference on Web Intelligence and Intelligent Agent Technology, IEEE,2015