# Intelligent Decision making systems

Abhishek Manoharan
*Computer Technology*
*MIT, Anna University*
2019503502

Mohammed Junaid N
*Computer Technology*
*MIT, Anna University*
2019503025

Gurbani Bedi
*Computer Technology*
*MIT, Anna University*
2019503518

***Abstract*—This documentation is about the project of developing an intelligent system(agent) capable of thriving in an environment with dynamically changing obstacles and thereby figuring out the way to reach the target location autonomously by means of continous exploration and learning. The decision making process is going to be handled by a single-layered neural network based on the input parameters recieved from the frontend part of the project. There is a separate class which handles the memory of the neural network. The agent will be having a separate memory object to store it's observations in the environment in the form of states ( at times t and t+1) and the corresponding reward recieved. The memory has a fixed size and a FIFO (First In First Out) structure. The functionalities of the neural network and it's memory are used in a single huge class that bridges the connection between the frontend and the backend by taking up the state parameters as the input and returning the decision to be taken. Separate class methods for saving and loading the snapshot of the neural network have also been implemented. The frontend is going to be built using kivy module which breaks the GUI into discreet objects called widgets which are handled by a single parent class that has the main eventloop. The aim of the frontend is to provide an elegant way of visualising the movement of the agent in the environment from the top view.**

## I. Introduction

Developing an intelligent agent capable of weaving through it's way, avoiding obstacles in a confined environment.

## II. The Language and the Modules used

### A. The Programming Language

We have used Python as the core programming language which is known for it's exceptional popularity in the world of Artificial Intelligence [1]. The Python programming language provides us with a plethora of packages which come handy for building learning models and their GUI.

**About Python:**

Python is an *interpreted high-level general-purpose programming language*. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is *dynamically-typed and garbage-collected*. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. Python consistently ranks as one of the most popular programming languages.

### B. The Packages

We have used the Pytorch library developed and maintained by Facebook inc. to build the neural network. The library is known for it's dynamic gradient calculation (autograd) and the flexibility it provides for designing our own models.

**About the Pytorch library:**

PyTorch [2] is an *open source machine learning library* based on the *Torch* library, used for applications such as *computer vision* [3] and *natural language processing* [4], primarily developed by Facebook's AI Research lab (FAIR). It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a C++ interface.

The Kivy module has been chosen to create a GUI to visualize the motion of the agent in the environment under observation.

**About Kivy module:**

Kivy [5] is a free and open source Python framework for developing *mobile apps* and other *multitouch application software with a natural user interface (NUI)*. It is distributed under the terms of the MIT License, and can run on Android, iOS, Linux, macOS, and Windows.

Kivy is the main framework developed by the *Kivy* organization, alongside Python for Android, Kivy iOS, and several other libraries meant to be used on all platforms. In 2012, Kivy got a grant from the Python Software Foundation for porting it to Python 3.3. Kivy also supports the *Raspberry Pi* which was funded through *Bountysource*.

## III. The structure of the backend file

The prominent classes:

- The 'Network' Class
- The 'Memory' Class
- The 'Brain' Class

## IV. The 'Network' Class

The network class contains the neural network [6] that is responsible for the decision making ability of the agent. This class contains a method for the feed-forward process, to obtain the outputs from the neural network.

Department of Computer Technology, MIT, Anna University

**What is an Artificial Neural Network(ANN)?**

Artificial neural networks (ANNs) [7], usually simply called neural networks (NNs), are computing systems vaguely inspired by the biological neural networks that constitute animal brains.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron that receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

### *A. The structure of the network*

The network consists of three layers: The input, a hidden and an output layer.

- The number of nodes in the input layer is equal to the number of input parameters (yet to be determined).
- The hidden layer has 30 nodes.
- The output layer has 3 nodes representing the probability for the three available decisions.

### *B. The Activation functions*

The activation function decides whether or not to accept the output (to activate) of a particular neuron. For the input and the hidden layers, we are going to use the **ReLu (Rectified linear unit)** [8] activation function to remove the linearity in the output of neurons. The relu activation function is going to be picked up from the *torch.nn.Functional* module. The relu function is represented as:

$$f(x) = max(0, x) \tag{1}$$

We are going to use the softmax activation function [9] to scale the outputs of the output layer into probabilities that sum up to 1. The softmax activation function looks like:

$$\sigma\left(\vec{z_i}\right) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2}$$

In order to pick up the best action by means of sampling (multinomial probability distribution) [10] using the *torch.multinomial()* function which takes the population lot and the sample size as arguments.

### *C. Backpropagation Algorithm*

The backpropagation algorithm [11] facilitates the updation of the weights and other parameters of neural network using the chain rule, applied to the loss function used in the neural network. By harnessing the power of partial derivatives, the backpropagation algorithm finds out the amount of error associated with every weight involved In the prediction process and updates them (by using complex mathematical techniques under the hood) so that the output gets more accurate with every feed-forward process in the neural network.

Amongst many back propagation algorithms, the **'Adam' (Adaptive moment estimation)** algorithm [12] is known for it's quicker convergence and better optimization [13] and so we are going to use it to obtain more accurate results in a short time period.

The adam optimizer is gonna be taken from the *torch.optim.Adam()* class which takes the model parameters, learning rate as the prominent arguments.

**Adam Configuration Parameters:**

- *alpha*($\alpha$) Also referred to as the learning rate or step size. The proportion that weights are updated (e.g. 0.001). Larger values (e.g. 0.3) results in faster initial learning before the rate is updated. Smaller values (e.g. 1.0E-5) slow learning right down during training
- *beta1*($\beta_1$): The exponential decay rate for the first moment estimates (e.g. 0.9)
- *beta2*($\beta_2$): The exponential decay rate for the second-moment estimates (e.g. 0.999). This value should be set close to 1.0 on problems with a sparse gradient (e.g. NLP and computer vision problems)
- *epsilon*($\epsilon$): Is a very small number to prevent any division by zero in the implementation (e.g. 10E-8)

### *D. The Loss Function*

The **'Smooth L1' loss function** (also known as the *Huber loss* [14]) is used as the evaluation metric for this decision making problem.The formula for the loss function is as follows:

$$loss = \begin{cases} \frac{0.5*(x_i - y_i)^2}{\beta}, & \text{if } \mid x_i - y_i \mid < 1 \\ \mid x_i - y_i \mid - 0.5 * \beta, & \text{otherwise} \end{cases}$$

The $\beta$ is the *scaling factor* and it is an optional parameter that defaults to 1, $x_i$ is the predicted value and $y_i$ is the actual or ideal value

This loss function is chosen because it is less sensitive to outliers and also prevents the *exploding gradient problem* [15] ( The weights getting larger due to the accumulation of gradients having a larger magnitude).

This loss function is imported from the *torch.nn.functional* module.

## V. The 'Memory' Class

### A. The need for a memory

The neural network needs to undergo the training phase in order to get fit for it's environment. But the catch here is, that the environment NEED NOT BE STATIC!
The environment will be provided with new obstacles on the go in a dynamic manner and the agent has to adapt to the changes to survive. Hence a specific **memory bag(queue type)** will be allotted to the agent so that it stores the most recent data obtained by exploring the environment and gets trained on it thereby being able to tackle the recent changes made in the environment

### B. The structure of the Memory Class

The 'Memory' data structure has a *queue type architecture*. The memory will have a fixed capacity. For every update, the most recent 100 records will be pulled out of the memory and the model will be trained on that data.
By this, the model keeps on updating itself about the latest changes in the environment

## VI. The 'Brain' Class

### A. The use of the Brain class

So far , we have got two classes namely, the 'Network' class and the 'memory' class. We have to build another class so as to **combine the functionalities of both the classes** and to create an overall decision making class which takes in the inputs and returns the decision by means of class methods.
The brain class facilitates the decision making by utilizing the memory class to fuel the training of the neural network and harnessing its power to make decisions given the parameters of the current state of the agent.

### B. The structure of the Brain class

The brain class has various methods at it's disposal to perform a lot of essential functions:

- The **'learn' function** that takes in the previous state, previously selected action, the corresponding reward and the current state to calculate the appropriate action that must have been taken in the previous step and to calculate the error associated with the prediction of the network in the previous step. The error is later *backpropagated* using the optimizer.
- The **'update' function** takes the previous reward and the current state to predict the step to be taken in the current state. The data is stored and the recent records are pulled out from the memory for learning in each and every update so that the network is up to date with the recent changes in the environment.
- A **'select_action' function** to select the action by means of multinomial sampling from the different probabilities obtained as a result from the neural network.
- A **'save_model' function** has been implemented to save the the model at a particular time of execution in a *'dictionary' (Python datatype)* format. The dictionary will have two key value pairs , one will have the parameters of the of the neural network and the other will have the parameters of it's optimizer algorithm. It be saved in *'.pth' format* in the home directory.
- The **'load_model' function** will search for a *'.pth' file* in the home directory and will load it into the present simulation.
- A **'score' function** has been made to return the mean of the recent rewards garnered by the agent.

## VII. Making the GUI(Frontend)

The entire front-end part rests on the basis of *widgets* and a *base app class*.

## VIII. The structure of the widgets

The structure of the widgets are specified in a separate file with a *'.kv'* extension. The file has different sections where a widget is referred by it's name (The same name used in the main python file as classes), followed by the specification of it's dimensions and it's position relative to the root widget of the GUI window. The structure of this '.kv' file will resemble a css file which is used to specify the design format of a webpage.

## IX. The widgets

The prominent widgets used in the GUI are mentioned below:

- The Agent widget
- The Ball widget
- The Game widget
- The Paint Widget

## X. The Agent Widget

The Agent widget has the attributes corresponding to the location of the sensors, the input obtained from them, details about the agent such as the current velocity and it's relative angle.
This widget class has a user defined class method to facilitate movement of the agent by taking the rotation as the input parameter and updating the position vectors of the visible parts (The body and the sensors) as well as the overall direction of the agent. This function also calculates the signals of the sensors which detect the sand (obstacle) density in their vicinity (400 sq.units).
The presence of sand is recorded in a numpy array with a dimension equal to that of the entire canvas with elements having values of either 0 or 1 indicating the absence or presence of sand respectively in a particular point of the environment.

## XI. The Ball Widget

Every sensor of the agent is represented in the form of a circle. This widget doesn't have any specialised functions and is just for visual representation. We have three instances of this widget to represent the three sensors which are placed at an angle of $30^0$ relative to each other.

## XII. The Game Widget

The Game widget controls the overall widget of the agent. It has an instance of both the Agent widget and the ball widgets(to represent the sensors). It has the facility to pass the parameters to the backed file and get the decision to be taken.

The decision is converted into an appropriate change in the angle of current direction vector using an action-to-direction array. On each updation, it checks if the agent has stepped on the sand and tweaks it's speed accordingly. This class also rewards or punishes the agent based on the consequences of it's actions.

Rules of reward and punishment(for a discreet action taken):

- If the robot steps on the sand, it gets punished by a magnitude of 1 unit. (reward = -1)
- If the robot gets away from the target as a result of it's action, it gets punished. (reward = -0.2)
- If the robot gets nearer to the target as a result of it's action, it gets rewarded. (reward = 0.1)
- If the robot gets near to any of the boundaries of the environment, it gets punished. (reward = -1)

The target gets shifted to it's mirrored opposite location so that the agent keeps on moving in a to and fro manner.

## XIII. The Paint Widget

This widget is to handle the dynamic placement of sand in the environment with a mouse click. This widget class has methods that act as event triggers for the mouse button actions such that the sand can be added to the environment dynamically by clicking on the desired area of the environment.

## XIV. The Main GUI window class

This class has the App class from the kivy module as it's base class. This class is responsible for maintaining the entire event loop of the main GUI window.

It has the instance of the Game widget to deploy the car in the environment. The scheduling of the main Clock to update the state of the car is configured in this class. Separate buttons for the 'Clear', 'Save' and 'Load' functionalities are declared, positioned and binded with their call functions.

The main app GUI class along with the widgets constitute the frontend of the project which aims provide better visualisation of the movement of the robot in the environment and also to test it's capabilities by adding obstacles (sand) in the runtime itself. There are options to save and load the recently saved snapshot of the brain of the agent at any timestamp.

## XV. Conclusion

Hereby, the complete details about the backend as well as the frontend of the project have been elaborated which sums up about 90% of the total project work. The integration of this project with the **ROS (Robot Operating System)** [16] has just began and we are looking forward to a smooth and successful completion of the proposed project.

## References


[1] A. M. Turing, "Computing machinery and intelligence," in *Parsing the turing test*. Springer, 2009, pp. 23–65.

[2] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[3] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[4] G. G. Chowdhury, "Natural language processing," *Annual review of information science and technology*, vol. 37, no. 1, pp. 51–89, 2003.

[5] M. Virbel, T. Hansen, and O. Lobunets, "Kivy–a framework for rapid creation of innovative user interfaces," in *Workshop-Proceedings der Tagung Mensch & Computer 2011. uberMEDIEN— UBERmorgen*. Universitätsverlag Chemnitz, 2011.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[8] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[9] R. A. Dunne and N. A. Campbell, "On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function," in *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, vol. 181. Citeseer, 1997, p. 185.

[10] S. Blumenthal, "Multinomial sampling with partially categorized data," *Journal of the American Statistical Association*, vol. 63, no. 322, pp. 542–551, 1968.

[11] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*. Elsevier, 1992, pp. 65–93.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[13] Z. Zhang, "Improved adam optimizer for deep neural networks," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 2018, pp. 1–2.

[14] J. Niu, J. Chen, and Y. Xu, "Twin support vector regression with huber loss," *Journal of Intelligent & Fuzzy Systems*, vol. 32, no. 6, pp. 4247–4258, 2017.

[15] G. Philipp, D. Song, and J. G. Carbonell, "The exploding gradient problem demystified-definition, prevalence, impact, origin, tradeoffs, and solutions," *arXiv preprint arXiv:1712.05577*, 2017.

[16] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.