Machine learning algorithms

130089872

Student Number

**Contents**

[Table of Figures 3](#_Toc126834010)

[Introduction 4](#_Toc126834011)

[Supervised learning 5](#_Toc126834012)

[Classification 5](#_Toc126834013)

[Regression 5](#_Toc126834014)

[Unsupervised learning 6](#_Toc126834015)

[Semi-supervised learning 6](#_Toc126834016)

[Pseudo-labelling 6](#_Toc126834017)

[Reinforcement learning 7](#_Toc126834018)

[Genetic algorithms 8](#_Toc126834019)

[8](#_Toc126834020)

[8](#_Toc126834021)

[Initial population 9](#_Toc126834022)

[Fitness function 10](#_Toc126834023)

[Selection 10](#_Toc126834024)

[Genetic algorithm selection methods 11](#_Toc126834025)

[Tournament 11](#_Toc126834026)

[Elitist 11](#_Toc126834027)

[Crossover 12](#_Toc126834028)

[Mutation 12](#_Toc126834029)

[Neural Networks 13](#_Toc126834030)

[Input layer 13](#_Toc126834031)

[Output layer 14](#_Toc126834032)

[Hidden layer 14](#_Toc126834033)

[Input neuron 14](#_Toc126834034)

[Input weight 15](#_Toc126834035)

[Bias 15](#_Toc126834036)

[Forward propagation 15](#_Toc126834037)

[Preactivation 15](#_Toc126834038)

[Activation 16](#_Toc126834039)

[Step function 16](#_Toc126834040)

[Sigmoid function 16](#_Toc126834041)

[Loss 16](#_Toc126834042)

[Backpropagation 16](#_Toc126834043)

[Dijkstra’s algorithm 17](#_Toc126834044)

[A star algorithm 19](#_Toc126834045)

[Conclusion 20](#_Toc126834046)

[Bibliography 21](#_Toc126834047)

[Appendix/Appendices 22](#_Toc126834048)

[pseudo code Dijkstra 22](#_Toc126834049)

[pseudo code A\* 23](#_Toc126834050)

[Pseudocode neural network 31](#_Toc126834051)

[Test log 36](#_Toc126834052)

# Table of Figures

[Figure 1 Input data example 5](#_Toc126421577)

[Figure 2 Reinforcement Learning example (The MathWorks, Inc, 2023) 7](#_Toc126421578)

[Figure 3 Flow chart of a typical GA (Hassanat, 2019) 8](file:///C:\Users\dales\Dropbox\My%20PC%20(LAPTOP-D0497OJ7)\Desktop\lv%206%20code%20task%20.docx#_Toc126421579)

[Figure 4 Genetic Algorithm overview (Mallawaarachchi, 2017) 9](#_Toc126421580)

[Figure 5 Tournament Selection example (Geeks for Geeks, 2022) 11](#_Toc126421581)

[Figure 6 Crossover point example (Mallawaarachchi, 2017) 12](file:///C:\Users\dales\Dropbox\My%20PC%20(LAPTOP-D0497OJ7)\Desktop\lv%206%20code%20task%20.docx#_Toc126421582)

[Figure 7 Mutation: Before and After (Mallawaarachchi, 2017) 12](#_Toc126421583)

[Figure 8 Neural Network example (Kavlakoglu, 2020) 13](#_Toc126421584)

[Figure 9 Singular input neuron 14](#_Toc126421585)

[Figure 10 Dijkstra example 17](#_Toc126421586)

[Figure 11 Calculation of Dijkstra visualised 17](#_Toc126421587)

[Figure 12 Dijkstra text example 18](#_Toc126421588)

[Figure 13 A star algorithm example 19](file:///C:\Users\dales\Dropbox\My%20PC%20(LAPTOP-D0497OJ7)\Desktop\lv%206%20code%20task%20.docx#_Toc126421589)

## Introduction

Machine learning (ML) assists algorithms to become more independent as well as improve with the experience of the function alongside existing data discovering frequencies thus no longer needing to rely on exact code parameters. This allows the application to function autonomously. Although multiple types of ML techniques are suitable for various applications, inside this document is an overview of ML discussing the following subtopics, the functionality, and types of ML, with a deeper look into genetic algorithms (GA) alongside a demonstration and applying ML into an application.

## Supervised learning

As it states the name of this approach, involves supervision of the data, this is achieved by a training algorithm that uses labelled datasets this consists of input and output parameters, providing a test dataset aids the machine to predict similar outcomes. The input data is normally displayed in a matrix format inside the program.

An example of this would be as shown in (Figure 1) the unit of observation is fruit/objects the program would use the data inside the features to assess what the object is. This could be optimized further for quality checking a product removing ambiguity from the production line.

Table

Description automatically generated

Figure 1 input data example

Continuing with supervised learning there are two main concepts.

## Classification

This concept categorises all data into induvial classes. This is achieved by training from the prerequisite data therefore future data can be stored in the correct field by boundary’s adapting over time. This is commonly used inside K-nearest neighbours.

## Regression

This concept originated from statistical modelling regression outputs are ordered and continuous where an output consists of one or more continuous variables. The most common type of regression is linear regression. Regression uses a trendline to predict with previous data. Unlike classification, this does not require existing boundaries, leading to faster implementation and response time.

## Unsupervised learning

This approach assesses its surroundings by providing its training labels, but before this artificial intelligence (AI) can walk the user will need to set how many clusters of data there should be. First, the AI predicts what data should be clustered together then over time the AI will update its existing labels to a more optimal set of data at first the predictions will be random and gradually become more arcuate and confined over multiple cycles.

## Semi-supervised learning

This approach uses a combination of supervised and unsupervised learning. This is used to assist when there is ladled and unable data. This would be used in large amounts of data where it may not be practical to label all points of data, for this example the user could create some initial labels referring back to the fruit example. This would be colour leaving the rest of the parameters unlabelled. Although this is the base of the input data, this could be improved upon using pseudo-labelling.

## Pseudo-labelling

This method is used to assist the predicted unlabelled data by applying it to the training data. This is achieved by keeping the original labelled data and output predictions.

This process uses the existing model to predict the data on the unlabelled data via the neural network. this is then applied and merged so both label and pseudo-labelled data sets are being trained simultaneously. Through the use of pseudo-labelling, it is possible to train on a much larger scale of data sets this also can assist in saving time and resources needed for numerous training cycles.

## Reinforcement learning

Reinforcement learning essentially is rewarding the AI when executing the correct methods. This does not require a data set this framework is similar to supervised learning as the input frame is run through a neural network that produces an output action this is called the policy network is shown in (Figure2) this is where there are no further decisions that are necessary for the ideal way of teaching the agent in this policy is using the policy gradient method. This is where a prediction respecting the optimized policy is sent back to the application and the next frame is produced continuing the loop, although this loop will need sampling to provide new actions the AI can learn from obtaining better rewards, therefore, optimizing the solution.

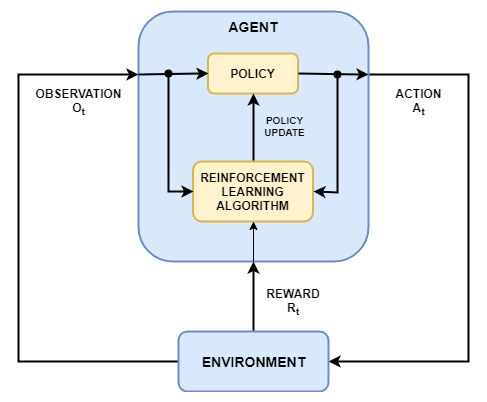
The way the algorithm can understand the correct actions is by rewarding it with data such as a number increasing with each correct cycle, an example of this would be as follows, a program has been created to play air hockey against a player, each time the puck is close enough to the net a frame is sent to the network to either move left or right if the robot scores against the other player it is rewarded a point which is recognized being correct in this instance, therefore, the next outcome are more likely to be similar to that solution until it is the optimal decision, this is where sampling is needed to further the findings.

Figure 2 Reinforcement learning example (The MathWorks, Inc, 2023)

## Genetic algorithms

Genetic algorithms are a part of evolutionary algorithms. The purpose of a genetic algorithm is to provide solutions to optimization and search problems by assessing possible solutions this is commonly a binary encoding of the data known as initial population. For example, with a shopping cart, the user has a set amount of money to buy various fruit, the goal of this is to have under two kilograms of fruit in the kart the algorithm would then test all possible solutions with one stating the fruit is in the kart and a zero if it is not. Once all solutions are created and this is called the generation normally the first generation will be random solutions and over time this will be more refined providing optimal solutions but the way this is achieved is with the process of natural selection and using the fitness function to rank the solution. The process of a genetic algorithm is displayed inside (Figure 3) with further explanation below.

## Diagram Description automatically generated

Figure Flow chart of a typical GA (Hassanat, 2019)

## Initial population

The first step of a genetic algorithm is the initial population this is typically the solution to the problem at hand before being optimized. This is also normally presented in binary which applies normalization to the algorithm, the way optimization is achieved is by evolving the initial population using the selection, crossover, and mutation functions. An overview of a genetic algorithm is shown inside (Figure 4).

Text

Description automatically generated with low confidence

Figure 4 Genetic Algorithm overview (Mallawaarachchi, 2017)

## Fitness function

The fitness function ranks the results of each solution and if the solution does not meet the criteria, it is given a zero and removed from possible future outcomes, as (Cristina Lo´ pez-Pujalte, 2003) states “this score will be used in the parent selection process so that the best-adapted individuals will have the greatest likelihood of being chosen”, without the uses of the fitness function a lot more iterations of the genetic algorithm will be required to be remotely come close to similar findings, but with the function applied, this will maintain an optimal GA that displays the best possible result with fewer iterations.

## Selection

Once all solutions are ranked the parents are selected for the next cycle to guide from usually these are the two highest-ranking solutions known as the selection process. Once this is achieved the parent data is passed on to the crossover.

# Genetic algorithm selection methods

## Tournament

This selection method consists of selecting the highest or lowest-ranking solutions from the current generation and essentially adds them to a bracket this is known as K-way tournament selection. This is where a tournament is run where the highest rankings succeed, and the winner of the tournament is then chosen and passed down to the next generation, to see more successful results and less likely hood of weak-ranked solutions. It is suggested to run a larger scale tournament, using a selection pressure which is measuring the likelihood of a set solution to being in the tournament. This assists in selecting and identifying optimal and near-optimal solutions a visual representation of this process are shown inside (Figure 5).

Chart, box and whisker chart

Description automatically generated with medium confidence

Figure 5 Tournament Selection example (Geeks for Geeks, 2022)

## Elitist

Elitism is similar to the fitness function but always chooses the top two rankings of the generation with the assistance of mutation. This is where a single node of the binary is altered providing new possible solutions that are ranked once again. This loops until the best solution have been found or the limit of generations the user provided has been reached.

## Crossover

Once the two parent solutions are chosen the crossover function takes place (Figure 5) this is where the two binary sets of data switch at a point the user has set this can be randomized, creating new binary solutions for the next generation known as a child this is repeated till a set number of solutions is met and the process is repeated. Eventually, all binary data will be the same although this may not be the best outcome if elitism is not applied.

Shape, rectangle

Description automatically generated

Figure Crossover point example (Mallawaarachchi, 2017)

## Mutation

The mutation function is where one of the parents is selected and values are altered at random to be carried over into the next iteration of the genetic algorithm to reduce duplicate results with the possibility of finding the optimal solution, once this stage is completed the value is assessed with other parents inside the fitness function and cycle continues. A before and after example of the mutation function is shown inside (Figure 7).

Text, table

Description automatically generated

Figure 7 Mutation: Before and After (Mallawaarachchi, 2017)

## Neural Networks

A neural network (NN) is the main body of a deep learning algorithm. This consists of the input layer that is attached to a hidden layer known as a Blackbox due to its complexity. The exit of the Blackbox attached to the output layer is shown in (Figure 8). The purpose of NN’s is to improve the accuracy of training data alongside great efficiency.



Figure 8 neural network example (Kavlakoglu, 2020)

## Input layer

The input layer is the first part of a NN. This is data that will run through the program. This can be pixel values or any given numerical data. It is possible to run the data as raw values however it is suggested to use normalization as this will assist in training a NN allowing the use of larger learning rates.

## Output layer

The output layer is the final part of a NN this is where the returned data is presented, after running through the hidden layer. The number of neurons set in this layer is defined by the number of classes. However, this can be also set as a single neuron using two classifications depending on the required outcome.

## Hidden layer

The hidden layer of a NN is placed between the input layer and output layer, this is where the weights are applied to the input and then directed through the NN. It is possible to have multiple layers inside the NN. This is what defines a simple neural network from a deep learning neural network. Hidden layers in a basic understanding are various types of activation functions that create an output of the probability of the data.

## Input neuron

A neuron is a singular entity inside the neural network that all data passes through to provide an output that varies depending on the position of the neuron, (Figure 9) multiple neurons in the same position would be known as a layer.

Diagram

Description automatically generated

Figure 9 Singular input neuron

## Input weight

The input weight is values that tune the data. This is added to the initial input in the hidden layer wight a calculation alongside the bias this is what transforms the data into a set value. The calculation would typically have three weights for each hidden neuron as well as one bias per layer which would be (inputs \* weights + bias).

## Bias

The bias is used to delay the activation function causing an offset in the result. This is known as the constant achieved by setting a parameter similar to the weight without this the output movements will be limited as this the purpose of a bias is to train the model.

## Forward propagation

Forward propagation is essentially an algorithm that assists in the learning phase. This is where the input data is passed through the NN to provide an output passing through one data point at a time, the process of forward propagation uses features that have been explained prior such as the weights. The multiply and add process is used when calculating the result using forward propagation.

During the process of forward propagation when the input data reaches a neuron inside the hidden layer or output layer, a two-stage calculation is applied the first is preactivation with the following being activation.

## Preactivation

Preactivation is the process of calculating the formula, which is the weighted sum of inputs, this decides if the information gathered should be passed through to the activation function.

## Activation

This function activates after the weight and bias are applied to the input data the purpose is to display the output before transferring the output to the next layer. Each neuron inside the hidden layer and output layer would have this type of function. Although there are many different formats of this calculation, such as the step function.

## Step function

This function displays as either a one or a zero dependent on the value, if it is greater than zero it will be displayed as a one this method does not provide an accurate representation of loss but is similar to the sigmoid function in an inferior way.

## Sigmoid function

The sigmoid function assists with logistical regression as it provides a decimal result between zero and one with this information the loss calculation can compare the accuracy of prior and current iterations whilst using backpropagation techniques.

## 

## Loss

This function is used to compare output values of the NN using predicted outcomes and to evaluate the algorithm's accuracy. If the value returned from the function is greater than the output, then the algorithm will need further improvement.

## Backpropagation

Backpropagation is essentially the process of optimizing the weights of a NN based on the loss/error rate that was achieved in previous iterations, as (Johnson, 2022) explained optimizing the weight and running the cycle again provides a reduced loss/error rate which more or less makes the model reliable by increasing its generalization.

## Dijkstra’s algorithm

This algorithm calculates the shortest distance from one node to every other node in the graph, for example in this graph of eight nodes (Figure 10) it is required to have a starting node and an end node in this example A to H is used.

A picture containing text, clock

Description automatically generated

Figure 10 Dijkstra example

The value between node A to node B is Two in the example above and the distance value between A to C is Five the smallest value is chosen which is B shown in (Figures 11 and 12), so the point is moved to B and the value of the entire journey is updated. Then the process continues till reaching the final node keeping the shortest distance used in mind.

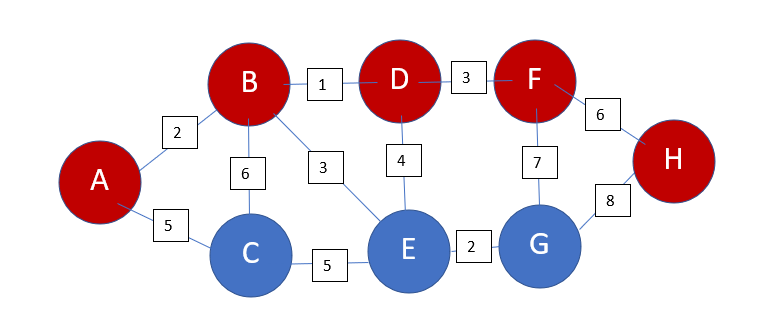


Figure 11 calculation of Dijkstra visualised

Graphical user interface

Description automatically generated with low confidence

Figure 12 Dijkstra text example

The time complexity of this algorithm is O(E+VlogV). This is calculated by E being equal to the total of connected nodes with weighted edges and V is equal to vertices or nodes. This means the greater distance in the calculation such as having hundreds of nodes that the algorithm needs to calculate through will lead to a longer response to the calculation. This is because each time both calculations are required before the node moves to the next one.

## A star algorithm

The a star algorithm (A\*) is a search algorithm used for pathfinding, (Geeks for Geeks, 2022) suggests that A\* is the most popular search algorithm that is widely used inside games as well as web-based maps. This is due to the algorithm being unique using traversal techniques that discover the shortest path by calculating the number of nodes required to travel through. The way the A\* calculates the travel distance is with the following equation F = G + H as (Ravikiran, 2022) explains “At each step, it picks the node with the smallest value of ‘f’ (the sum of ‘g’ and ‘h’) and processes that node/cell.” In a clearer explanation, F is the total estimated cost to pass through the current node, G is the cost of all nodes currently passed, with H being the heuristic and an estimated value to the target a visual example is shown in (Figure 13).

Chart

Description automatically generated

Figure A star algorithm example

Inside the example, an obstacle was created to demonstrate how the algorithm functions. The calculation that was applied cycles until reaching the target displayed the shortest path in yellow. Although this search algorithm is popular there are drawbacks to using A\* for largescale models this is due to the fact the search algorithm individuality searches each possible node before a calculation is made as (Bartakke, 2022) supports this by stating “in the worst case, the A-star algorithm travels all the edges to the reach the destination from the source. So, the worst case time complexity is O(E), where E is the number of edges in the graph.”

# Conclusion

In conclusion, the topics presented inside this document should provide the reader with an understanding of algorithms and functions required to provide outputs, although some have been explained there are plenty of other solutions that may provide an optimal result. Looking back at the neural network using the rectified linear unit activation function (ReLU) instead of the sigmoid function has provided a greater convergence performance, thus optimising the training model when applied correctly, as well as doing this using two different activation functions, in this case, the SoftMax on layer 2 assists with preventing recurrent results. Whilst building the neural network a few challenges were discovered during the building process while testing various techniques whilst attempting to apply the activation function for SoftMax on the second layer an attribute error had arisen. In future studies overcoming this challenge would be beneficial as stated using two different activation functions inside a neural network would provide greater convergence, although this build is simplistic lot of skills and knowledge has been ingested during the studies, giving great confidence and interest inside the field which will be pursued in the future building a more detailed neural network to assist in the health care and medical world.

# Bibliography

Bartakke, P. (2022). *A\* Algorithm (Graph Traversal and Path Search Algorithm)*. Retrieved from enjoy algorithms: https://www.enjoyalgorithms.com/blog/a-star-search-algorithm

Cristina Lo´ pez-Pujalte, V. P.-B. (2003). *Order-Based Fitness Functions for Genetic Algorithms Applied to Relevance Feedback.* Wiley Periodicals, Inc. doi:54(2):152–160, 2003

Geeks for Geeks. (2022, May 30th). *A\* Search Algorithm*. Retrieved from https://www.geeksforgeeks.org/a-search-algorithm/

Geeks for Geeks. (2022, July 11th). *Tournament Selection (GA)*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/tournament-selection-ga/

Hassanat, A. (2019). *Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach.* MDPI. doi:https://doi.org/10.3390/info10120390

Johnson, D. (2022, Dec 31st). *Back Propagation in Neural Network: Machine Learning Algorithm*. Retrieved from Guru99: https://www.guru99.com/backpropogation-neural-network.html

Kavlakoglu, E. (2020). *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What’s the Difference?* IBM. Retrieved Jan 31st, 2023, from https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks

Mallawaarachchi, V. (2017). *Introduction to Genetic Algorithms.* Towards Data Science. Retrieved from https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3

Ravikiran, A. (2022, nov 15th). *A\* Algorithm Concepts and Implementation*. Retrieved from simplilearn: https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/a-star-algorithm

The MathWorks, Inc. (2023). *What Is Reinforcement Learning?* Math works. Retrieved from https://uk.mathworks.com/help/reinforcement-learning/ug/what-is-reinforcement-learning.html

# Appendix/Appendices

# pseudo code Dijkstra

from collections import defaultdict

class Graph:

def \_\_init\_\_(self):

self.nodes = set()

self.edges = defaultdict(list)

self.distances = {}

def addNode(self,value):

self.nodes.add(value)

def addEdge(self, fromNode, toNode, distance):

self.edges[fromNode].append(toNode)

self.distances[(fromNode, toNode)] = distance

def dijkstra(graph, initial):

# visited score increases during movement through the nodes but starts at 0.

visited = {initial : 0}

path = defaultdict(list)

nodes = set(graph.nodes)

while nodes:

minNode = None

for node in nodes:

if node in visited:

if minNode is None:

minNode = node

elif visited[node] < visited[minNode]:

minNode = node

if minNode is None:

break

nodes.remove(minNode)

currentWeight = visited[minNode]

for edge in graph.edges[minNode]:

weight = currentWeight + graph.distances[(minNode, edge)]

if edge not in visited or weight < visited[edge]:

visited[edge] = weight

path[edge].append(minNode)

return visited, path

#Values of the nodes can be alterd to change the optimal path

#nodes can be also added and removed to this

customGraph = Graph()

customGraph.addNode("A")

customGraph.addEdge("A", "B", 2)

customGraph.addEdge("A", "C", 5)

customGraph.addNode("B")

customGraph.addEdge("B", "C", 6)

customGraph.addEdge("B", "D", 1)

customGraph.addEdge("B", "E", 3)

customGraph.addNode("C")

customGraph.addEdge("C", "E", 5)

customGraph.addNode("D")

customGraph.addEdge("D", "E", 4)

customGraph.addEdge("D", "F", 3)

customGraph.addNode("E")

customGraph.addEdge("E", "G", 2)

customGraph.addNode("F")

customGraph.addEdge("F", "G", 7)

customGraph.addEdge("F", "H", 6)

customGraph.addNode("G")

customGraph.addEdge("G", "H", 8)

customGraph.addNode("H")

#This starting point of the nodes which can be changed

print(dijkstra(customGraph, "A"))

#time complexity: O(V^2)

#space complexity: O(E) E = Number of edges

# pseudo code A\*

import pygame

import math

from queue import PriorityQueue

WIDTH = 800

WIN = pygame.display.set\_mode((WIDTH, WIDTH))

pygame.display.set\_caption("A\* Path Finding Algorithm")

RED = (255, 0, 0)

GREEN = (0, 255, 0)

BLUE = (0, 255, 0)

YELLOW = (255, 255, 0)

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

PURPLE = (128, 0, 128)

ORANGE = (255, 165 ,0)

GREY = (128, 128, 128)

TURQUOISE = (64, 224, 208)

class Spot:

def \_\_init\_\_(self, row, col, width, total\_rows):

self.row = row

self.col = col

self.x = row \* width

self.y = col \* width

self.color = WHITE

self.neighbors = []

self.width = width

self.total\_rows = total\_rows

def get\_pos(self):

return self.row, self.col

def is\_closed(self):

return self.color == RED

def is\_open(self):

return self.color == GREEN

def is\_barrier(self):

return self.color == BLACK

def is\_start(self):

return self.color == ORANGE

def is\_end(self):

return self.color == TURQUOISE

def reset(self):

self.color = WHITE

def make\_start(self):

self.color = ORANGE

def make\_closed(self):

self.color = RED

def make\_open(self):

self.color = GREEN

def make\_barrier(self):

self.color = BLACK

def make\_end(self):

self.color = TURQUOISE

def make\_path(self):

self.color = PURPLE

def draw(self, win):

pygame.draw.rect(win, self.color, (self.x, self.y, self.width, self.width))

def update\_neighbors(self, grid):

self.neighbors = []

if self.row < self.total\_rows - 1 and not grid[self.row + 1][self.col].is\_barrier(): # DOWN

self.neighbors.append(grid[self.row + 1][self.col])

if self.row > 0 and not grid[self.row - 1][self.col].is\_barrier(): # UP

self.neighbors.append(grid[self.row - 1][self.col])

if self.col < self.total\_rows - 1 and not grid[self.row][self.col + 1].is\_barrier(): # RIGHT

self.neighbors.append(grid[self.row][self.col + 1])

if self.col > 0 and not grid[self.row][self.col - 1].is\_barrier(): # LEFT

self.neighbors.append(grid[self.row][self.col - 1])

def \_\_lt\_\_(self, other):

return False

def h(p1, p2):

x1, y1 = p1

x2, y2 = p2

return abs(x1 - x2) + abs(y1 - y2)

def reconstruct\_path(came\_from, current, draw):

while current in came\_from:

current = came\_from[current]

current.make\_path()

draw()

def algorithm(draw, grid, start, end):

count = 0

open\_set = PriorityQueue()

open\_set.put((0, count, start))

came\_from = {}

g\_score = {spot: float("inf") for row in grid for spot in row}

g\_score[start] = 0

f\_score = {spot: float("inf") for row in grid for spot in row}

f\_score[start] = h(start.get\_pos(), end.get\_pos())

open\_set\_hash = {start}

while not open\_set.empty():

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

current = open\_set.get()[2]

open\_set\_hash.remove(current)

if current == end:

reconstruct\_path(came\_from, end, draw)

end.make\_end()

return True

for neighbor in current.neighbors:

temp\_g\_score = g\_score[current] + 1

if temp\_g\_score < g\_score[neighbor]:

came\_from[neighbor] = current

g\_score[neighbor] = temp\_g\_score

f\_score[neighbor] = temp\_g\_score + h(neighbor.get\_pos(), end.get\_pos())

if neighbor not in open\_set\_hash:

count += 1

open\_set.put((f\_score[neighbor], count, neighbor))

open\_set\_hash.add(neighbor)

neighbor.make\_open()

draw()

if current != start:

current.make\_closed()

return False

def make\_grid(rows, width):

grid = []

gap = width // rows

for i in range(rows):

grid.append([])

for j in range(rows):

spot = Spot(i, j, gap, rows)

grid[i].append(spot)

return grid

def draw\_grid(win, rows, width):

gap = width // rows

for i in range(rows):

pygame.draw.line(win, GREY, (0, i \* gap), (width, i \* gap))

for j in range(rows):

pygame.draw.line(win, GREY, (j \* gap, 0), (j \* gap, width))

def draw(win, grid, rows, width):

win.fill(WHITE)

for row in grid:

for spot in row:

spot.draw(win)

draw\_grid(win, rows, width)

pygame.display.update()

def get\_clicked\_pos(pos, rows, width):

gap = width // rows

y, x = pos

row = y // gap

col = x // gap

return row, col

def main(win, width):

ROWS = 50

grid = make\_grid(ROWS, width)

start = None

end = None

run = True

while run:

draw(win, grid, ROWS, width)

for event in pygame.event.get():

if event.type == pygame.QUIT:

run = False

if pygame.mouse.get\_pressed()[0]: # LEFT

pos = pygame.mouse.get\_pos()

row, col = get\_clicked\_pos(pos, ROWS, width)

spot = grid[row][col]

if not start and spot != end:

start = spot

start.make\_start()

elif not end and spot != start:

end = spot

end.make\_end()

elif spot != end and spot != start:

spot.make\_barrier()

elif pygame.mouse.get\_pressed()[2]: # RIGHT

pos = pygame.mouse.get\_pos()

row, col = get\_clicked\_pos(pos, ROWS, width)

spot = grid[row][col]

spot.reset()

if spot == start:

start = None

elif spot == end:

end = None

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_SPACE and start and end:

for row in grid:

for spot in row:

spot.update\_neighbors(grid)

algorithm(lambda: draw(win, grid, ROWS, width), grid, start, end)

if event.key == pygame.K\_c:

start = None

end = None

grid = make\_grid(ROWS, width)

pygame.quit()

main(WIN, WIDTH)

## Pseudo code neural network

import sys

import numpy as np

import matplotlib

import random

#neural network input layer

X = random.randrange(2, 20)

y = random.randrange(1,40)

#hidden layer

class Layer\_Dense:

def \_\_init\_\_(self, n\_inputs, n\_neurons):

self.weights = 0.01 \* np.random.randn(n\_inputs, n\_neurons)

self.biases = np.zeros((1, n\_neurons))

def foward(self, inputs):

self.output = np.dot(inputs,self.weights) + self.biases

#activation function ReLU activationb

class Activation\_ReLU:

def foward(self, inputs):

self.output = np.maximum(0, inputs)

#activation function Softmax

class Activation\_Sofatmax:

def foward(self, inputs):

exp\_values = np.exp(inputs - np.max(inputs, axis=1, keepdims=True))

probabilities = exp\_values / np.sum(exp\_values, axis=1, keepdims=True)

self.output = probabilities

#Loss function

class Loss:

def calculate(self, output, y):

sample\_losses = self.foward(output, y)

data\_loss = np.mean(sample\_losses)

return data\_loss

class Loss\_CategoricalCrossentropy(Loss):

def foward(self, y\_pred, y\_true):

samples = len(y\_pred)

y\_pred\_clipped = np.clip(y\_pred, 1e-7, 1-1e-7)

if len(y\_true.shape) == 1:

correct\_confidences = y\_pred\_clipped[range(samples), y\_true]

elif len(y\_true.shape) == 2:

correct\_confidences = np.sum(y\_pred\_clipped\*y\_true, axis=1)

negative\_log\_likelihoods = -np.log(correct\_confidences)

return negative\_log\_likelihoods

#layer 1

dense1 = Layer\_Dense (2,3)

activation1 = Activation\_ReLU()

#layer 2

dense2 = Layer\_Dense(3,3)

activation2 = Activation\_Sofatmax()

dense1.foward(X)

activation1.foward(dense1.output)

dense2.foward(activation2.output)

activation2.foward(dense2.output)

print(activation2.output[:5])

loss\_function = Loss\_CategoricalCrossentropy()

loss = loss\_function.calculate(activation2.output, y)

print("Loss:", loss)

#print(0.10\*np.random.randn(4, 3))

'''

inputs = [1, 2, 3, 2.5]#single sample set

'''

'''

inputs = [[1, 2, 3, 2.5],

[2.0, 5.0, -1.0, 2.0],

[-1.5, 2.7, 3.3, -0.8]]

#batch sample set helping with the fitment line

weights = [[0.2, 0.8, -0.5, 1.0],

[0.5, -0.91, 0.26, -0.5],

[-0.26, 0.27, 0.17, 0.87]]

biases = [2, 3, 0.5]

#layer1

weights2 = [[0.1, -0.14, 0.5],

[-0.5, 0.12, -0.33],

[-0.44, 0.73, -0.13]]

biases2 = [-1,2, -0.5]

#layer2

layer1\_outputs = np.dot(inputs, np.array(weights).T) + biases

layer2\_outputs = np.dot(layer1\_outputs, np.array(weights2).T) + biases2

print(layer2\_outputs)

print(output)

'''

'''

weights = [0.2, 0.8, -0.5, 1.0]

bias = [2]

output = np.dot(weights, inputs)+ bias

print(output)

'''

'''

some\_value = 0.5

weight = 0.7

bias = 0.7

print(some\_value\*weight)

print(some\_value+bias)

'''

'''

layer\_outputs = [] #Ouput of current layer

for neuron\_weights, neruon\_bias in zip(weights, biases):

neuron\_output = 0 # Output of given neuron

for n\_input, weight in zip(inputs, neuron\_weights):

neuron\_output += n\_input\*weight

neuron\_output += neruon\_bias

layer\_outputs.append(neuron\_output)

print(layer\_outputs)

'''

'''

output = [inputs[0]\*weights1[0] + inputs[1]\*weights1[1] + inputs[2]\*weights1[2] + inputs[3]\*weights1[3] + bias1,

inputs[0]\*weights2[0] + inputs[1]\*weights2[1] + inputs[2]\*weights2[2] + inputs[3]\*weights2[3] + bias2,

inputs[0]\*weights3[0] + inputs[1]\*weights3[1] + inputs[2]\*weights3[2] + inputs[3]\*weights3[3] + bias3]

print(output)

'''

# Test log

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test number | Test description | Expected result | Result | Did it work? |
| 1 | Inside the Dijkstra algorithm removing edge of node D will the algorithm discover a new path? | Instead of using node D, Node E will be chosen | Node E was chosen | Yes |
| 2 | Inside the Dijkstra algorithm altering the end location to node G will the path adjust accordingly? | Path is changed and uses Nodes: A B E G | Path was changed | Yes |
| 3 | Inside Dijkstra removing the path between Node B and E will the path adjust again instead of using A B E G | Path is changed to A B D E G | Path was changed | yes |
| 4 | Inside the Dijkstra removing both paths to Node H will the algorithm be able to form a path? | Application is to error as there is no possible path | Errors | No |
| 5 | A\* creating an obstacle inside the program will it acknowledge it or travel straight through? | When running the program, the obstacles are respected | Obstacles are respected | Yes |
| 6 | A\* when putting the obstacle around the start position will the algorithm be able to find the end goal | No the obstacle will be respected and there will be no possible solution | Obstacle is respected however no text box has been created to display that there is no solution | No (no text box) |

## Data Dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| Field name | Data type | Field length | Description |
| Node | String | 9 | Point inside the algorithm |
| Weight | Int | 3-digit int | The cost of movement |
| Edge | String | 9 | The path between nodes. |

## Algorithm Design

# Dijkstra’s

A picture containing text, clock

Description automatically generated

## Program Flow

Diagram

Description automatically generated