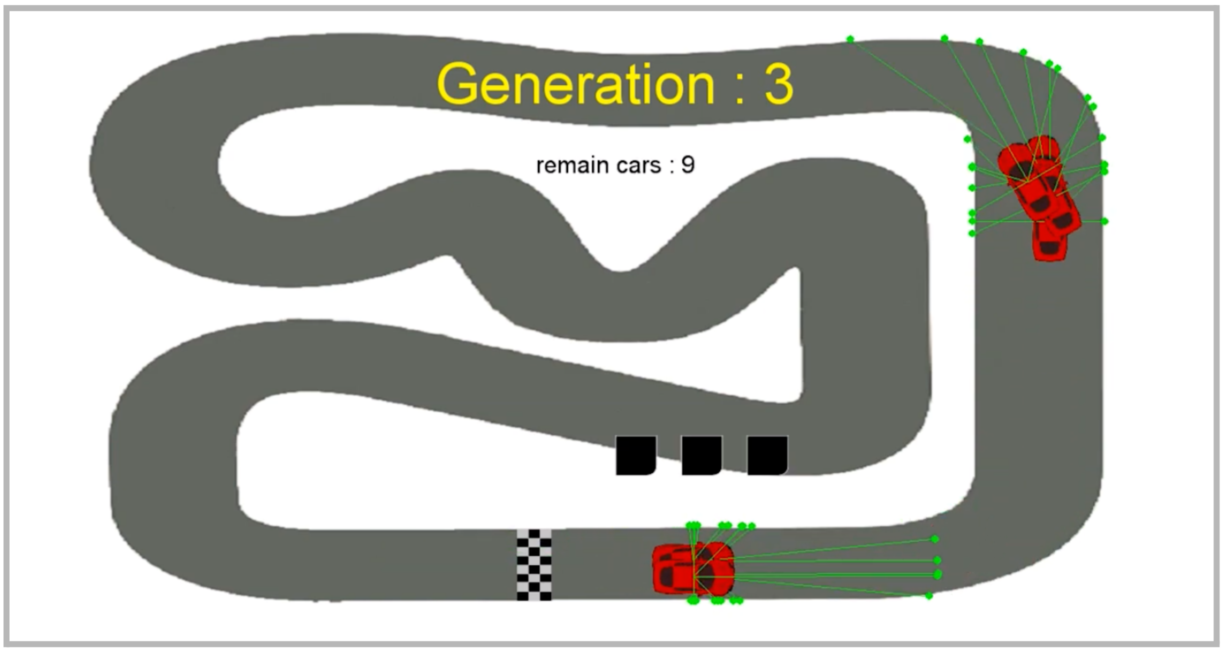
Analysis of change within parameters and its effect on the simulation



Research question: What are the effects of the change to a specific parameter directly related to machine learning concepts such as hyperparameters and activation functions?

By Ava Frodyma-Smith

Code snippet:

```
def check_radar(self, degree, game_map):
length = 0
x = int(
self.center[0]
+ math.cos(math.radians(360 - (self.angle + degree))) * length
)
y = int(
self.center[1]
+ math.sin(math.radians(360 - (self.angle + degree))) * length
)

# While We Don't Hit BORDER_COLOR AND length < 300 (just a max) -> go further and
further
while not game_map.get_at((x, y)) == BORDER_COLOR and length < 300:
length = length + 1
x = int(
self.center[0]
+ math.cos(math.radians(360 - (self.angle + degree))) * length
)
y = int(
self.center[1]
+ math.sin(math.radians(360 - (self.angle + degree))) * length
)

# Calculate Distance To Border And Append To Radars List
dist = int(
math.sqrt(math.pow(x - self.center[0], 2) + math.pow(y - self.center[1], 2))
)
self.radars.append([(x, y), dist])
```

Aim: To explore the difference in code after affecting the parameters and further changing the code to see how the simulation reacts and differs.

Background information:
There were no specific activation functions within this report that were found however, there were a couple of parameters and hyperparameters. For the parameters, the main one found frequently throughout the code was the ‘self’ parameter. It refers back to the general class, which was the agent (the car) in most instances. Another parameter was the degree parameter, which links to the radar and its angle and the degree it was turning. Game_map also stands, this was the environment (game map).
For the hyperparameters, there were three in particular. Length represented the radar scans and how long it could reach. The border_colour was defined earlier but it relates to the colour of the map and how that relates to the self.alive function switching to false, or the car dying. Lastly, the 300 hyperparameter stood to set the maximum length given for the scanners of the sensors, their maximum range. These (hyper)parameters had other links throughout the code that further established their roles.

Observations:
The first change that was carried out was the alterations to the degree parameter. The value was changed to test the alternate angles from the agent's angles. After this, the length attribute was also substituted to see its effects on the code. The range could be decreased and increased and its ability to search for obstacles was increased/decreased by altering this specific value. This changed its way of moving about and searching for difficulties, how long it took to find them and the ability to avoid them. Lastly, the border_colour hyperparameter could be changed to either benefit or detriment the agent. This affects its journey in terms of what it was looking for to be able to navigate around it without dying. With all these parameters being changed throughout it had either a negative or positive exchange to the code and its distance and time reaching certain points, which ends up gaining it more or less rewards.

Hypothesis:
If we change the parameters within a code, then the agent will be affected negatively or positively in terms of progress.

Variables:
The independent variable is the code snippet itself, the changed number or word that can also lead to the results for the change. In the first example of the degree parameter, the number of degrees could be changed. In the length parameters, the initial distance was the affected variable. For border_colour, the pixel colour was changed throughout the code.
In both the degrees and length parameters, the x and y coordinates were the dependent variables, they had the position for the radar and could record the results of the change in the original code to the changed ones. The self.radar list was also impacted as it stored

information on the obstacles experienced by the radars and recorded it for future reference. This specific independent variable was used by all the ones explored above, degrees, length and border_color. Lastly, the distance variable was also changed by all three of them. This independent variable would be changed as the dependent variable was edited to apply correctly to the agent's progress.
Controlled variables are the game_map, If this is kept the same the obstacles don't change, therefore the values that are being recorded and their time to record this aren't affected by the environment changing up. The agent's position and angle throughout this. If this remains constant, the results won't change too much throughout the testing.

Method:

Degree parameter:
Within the degree-defining part of the code, in the check_radar section of the code, the value for 'degree' was edited to suit the likings of the code editor. Using 45 as an example, the value was replaced to affect the progress and agent in the 'pygame' simulator.
Example: `{your_object.check_radar(45, game_map) # Change the degree to 45 degrees}`

Length parameter:
The same type of process was followed for the degree parameter, as in the length one. The value for length was replaced to affect the performance to the editor's wishes.
Example: `your_object.check_radar(90, game_map, length=50) # Change the length to 50`

Border_colour:
The same process was pretty much applied however this was a worded value instead. The check_radar value method was used to affect it and the value for the border_colour would be then changed to suit the editor's desires.
Example: `your_object.check_radar(90, game_map, border_color=NEW_COLOR) # Change the border color`

Results:

The parameters were affected to the editor's desires and this changed the performance of the car in multiple ways. The viewing of obstacles, time, displacement around the game_map and distance were all values that were affected by the changing of these parameters. The program would run smoother and faster at the change of some whilst others were detrimental to this statement and instead negatively changed the progress of the car, whether that be making it crash more or generally slowing its progress through the track. Specifically, when the degree function was changed, the angle for the nose of the car was affected and this changed its performance significantly. In length, when the sensor was given more space to explore, obstacles were easier to predict or foresee so the car would have less trouble finding its way through the track and fewer generations would be spent trying to teach cars to get through. For some cars, however, this addition freaked them out and they either backed down or died quite fast. The opposite effect was had for making the range smaller. Border_colour was quite difficult for the designed maps had already their own colours to stay within, Changing colours made it rather difficult for the cars to navigate the course so their range of space not to hit was either shortened or lengthened. It would either make it more difficult with a smaller space to move within or very easy with lots of space to move about. If the controlled variable, the game map was changed and multiple colours were introduced to the map design itself, there would be a bigger impact on the results.

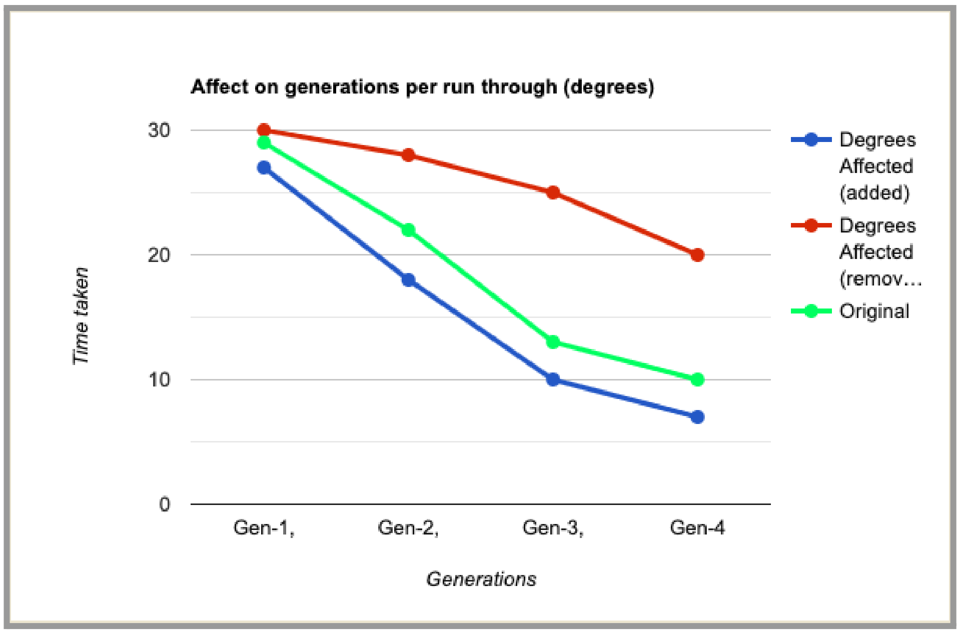
Conclusion:
By changing the parameters, the simulation was heavily affected and could either be made more efficient and faster or slower and more difficult for the agent, hence, the actual learning process of the machine was heavily modified and controlled by the change of a few, simple parameters in the code. This shows the importance of them and their values within the program and how changing them can impact the actual simulation and the machine-learning component of the code. More of this could be explored by finding bigger parts of the code snippet, using arguments and changing them instead, or other bigger components of the code. Exploring the effect on machine learning by altering certain parts of the code, whether bigger or smaller sections, can teach us more about machine learning and how it can be improved or even brought down just by changing a small component to the code. Future research could definitely be conducted on all the aspects of codes, even quite complex big codes and how the smallest change in even just a singular value will change the entire code and its learning process as a machine.

Information in graphs

Graph 1: Degrees

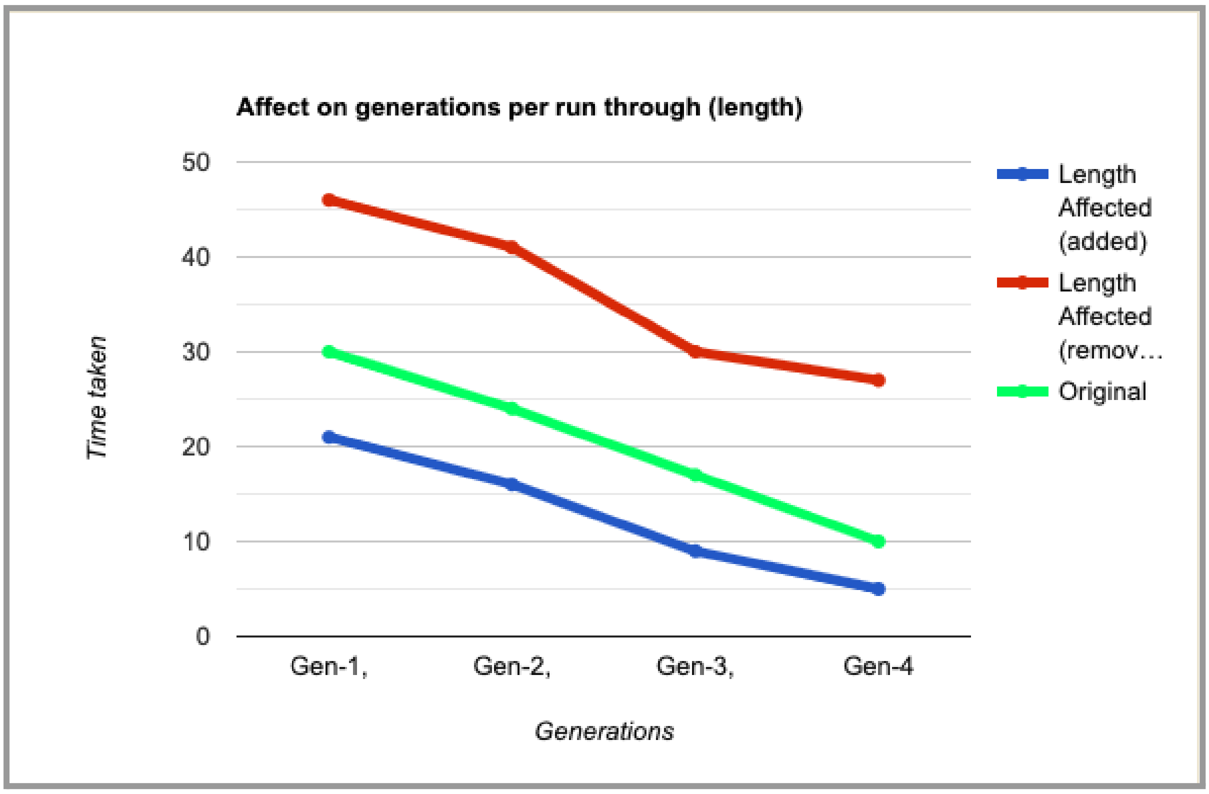
This graph will confirm the result on the effects of the time taken for the car to get successfully around the track depending on the value of the degree being changed. The green line represents the original timing, the blue, the more positive outcome of changing the degree and the red, the less positive outcome, but interesting to explore further the impacts of changing a couple of digits. These results are determined off specific degree numbers and only one of the cars. (The fittest)

|
|
|
|
V



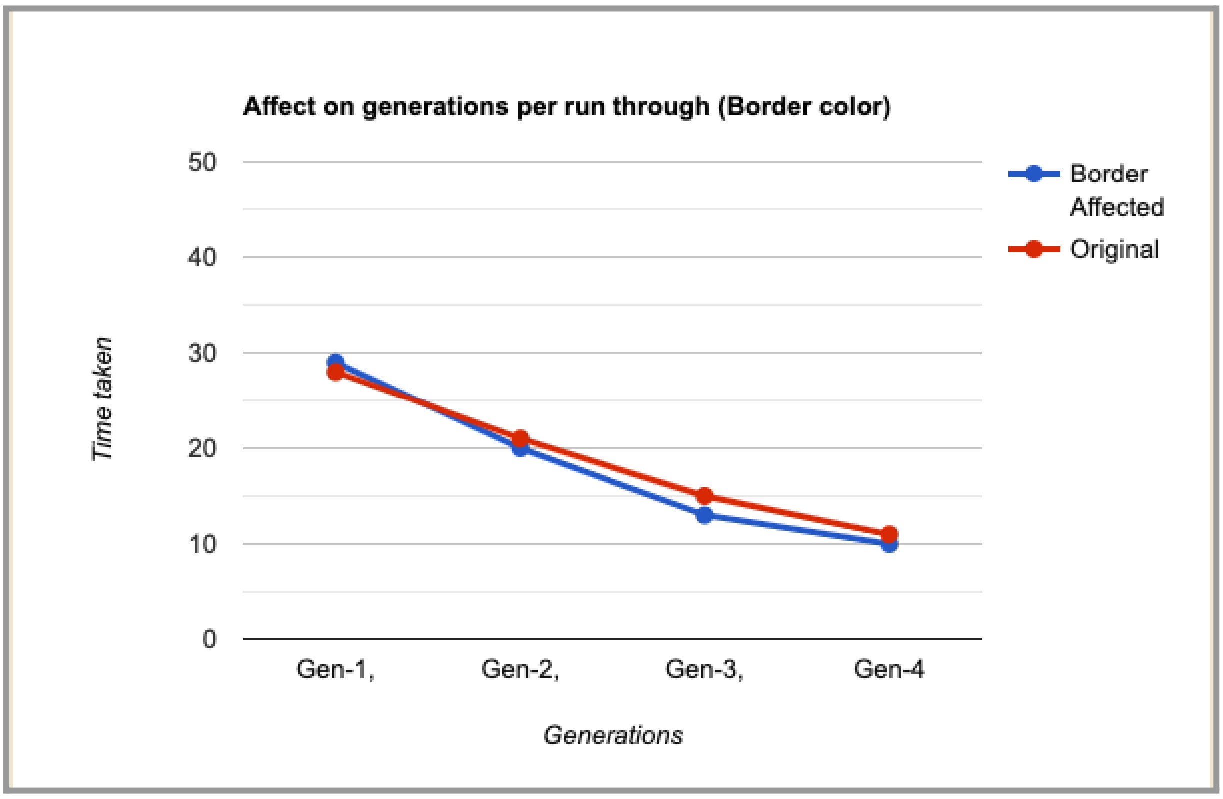
Graph 2: Length

This graph shows the maximum length of the sensors allowed and how it transformed the time it took for the fittest car to memorise the course. The original still stands at a pretty average timing but significantly higher, the red line where the length was a smaller digit took a lot longer to memorise the course and find its way about it. Opposite to this change, the blue line had added length and found it quicker and easier to actually navigate the course effectively and fast. Its results are lower for such reasons.



Graph 3: Border_colour

Lastly, the border colour was more difficult to carry out for there isn't really a range of colours to select from, I added a smaller rim of light grey to the outside of the first map and tried to use that to see how it would change the process. It is clear there isn't a big amount of change, the lines are quite closely connected and I assume it is because the added grey lines were beside the original borders and therefore, not much was changed for the sensors. There was only a slight change in their angle and length throughout.



References:

(No date) *Chatgpt - openai聊天*. Available at: https://chat.openai.com/auth/login?next=%2F%3Fmodel%3Dtext-davinci-002-render-sha (Accessed: 11 September 2023).

Code - interpreter base classes (no date) *Python documentation*. Available at: https://docs.python.org/3/library/code.html (Accessed: 12 September 2023).

Dutta, M. (2022) *Impact of hyperparameters on a deep learning model*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2022/05/impact-of-hyperparameters-on-a-deep-learning-model/ (Accessed: 12 September 2023).

Nielsen, M.A. (1970) *Neural networks and deep learning*. Available at: http://neuralnetworksanddeeplearning.com/ (Accessed: 12 September 2023).

Nyuytiymbiy, K. (2022) *Parameters and hyperparameters in machine learning and Deep Learning*, *Medium*. Available at: https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac#:~:text=Hyperparameters%20are%20parameters%20whose%20values,parameters%20that%20result%20from%20it. (Accessed: 12 September 2023).

Paul, S. (2018) *Hyperparameter Optimization & Tuning for Machine Learning (ML)*, *DataCamp*. Available at: https://www.datacamp.com/tutorial/parameter-optimization-machine-learning-models (Accessed: 12 September 2023).

Unleashing the power of programming (no date) *An Introduction To Coding*. Available at: https://www.arduino.cc/education/an-introduction-to-coding/ (Accessed: 12 September 2023).

Welcome to neat-python's documentation!¶ (no date) *NEAT*. Available at: https://neat-python.readthedocs.io/en/latest/ (Accessed: 12 September 2023).

What are machine learning models? (no date) *Databricks*. Available at: https://www.databricks.com/glossary/machine-learning-models#:~:text=Deep%20learning%20models%20are%20a,features%20from%20the%20data%20provided. (Accessed: 12 September 2023).

What is machine learning? (no date) *IBM*. Available at: https://www.ibm.com/topics/machine-learning (Accessed: 12 September 2023).

MacWha, R. (2021) *Evolving AIS using a NEAT algorithm*, *Medium*. Available at: https://macwha.medium.com/evolving-ais-using-a-neat-algorithm-2d154c623828 (Accessed: 12 September 2023).

Ballard, Z. *et al.* (2021) *Machine learning and computation-enabled intelligent sensor design*, *Nature News*. Available at: https://www.nature.com/articles/s42256-021-00360-9 (Accessed: 12 September 2023).

Create Harvard, APA & MLA Citations (no date) *Cite This For Me, a Chegg service*. Available at: https://www.citethisforme.com/ (Accessed: 12 September 2023).

-End of report-