Brain AI Coursework

Thomas Vernon 1907240

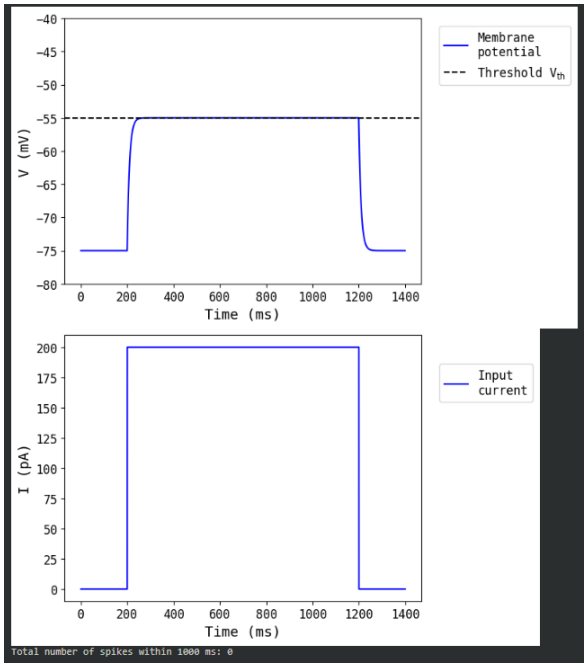
Part 1

Task 1

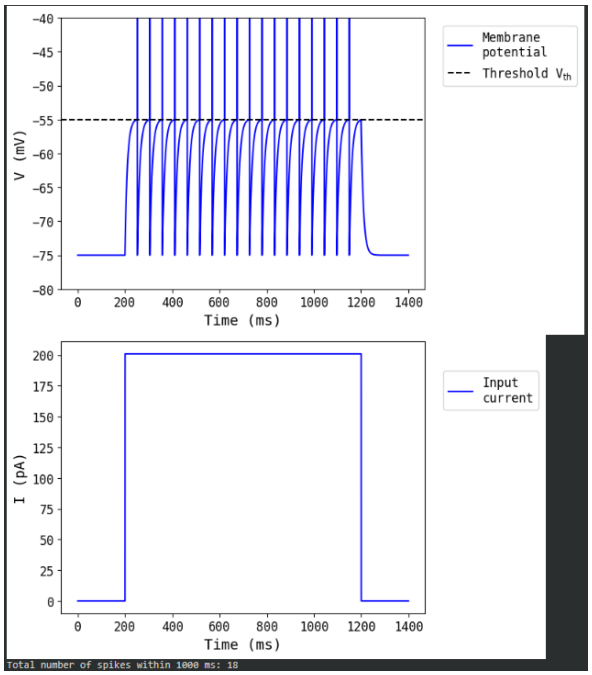
Q1

I_Inj = 200 is the threshold to not produce any spike. Any higher will produce a spike.

I_Inj = 200:



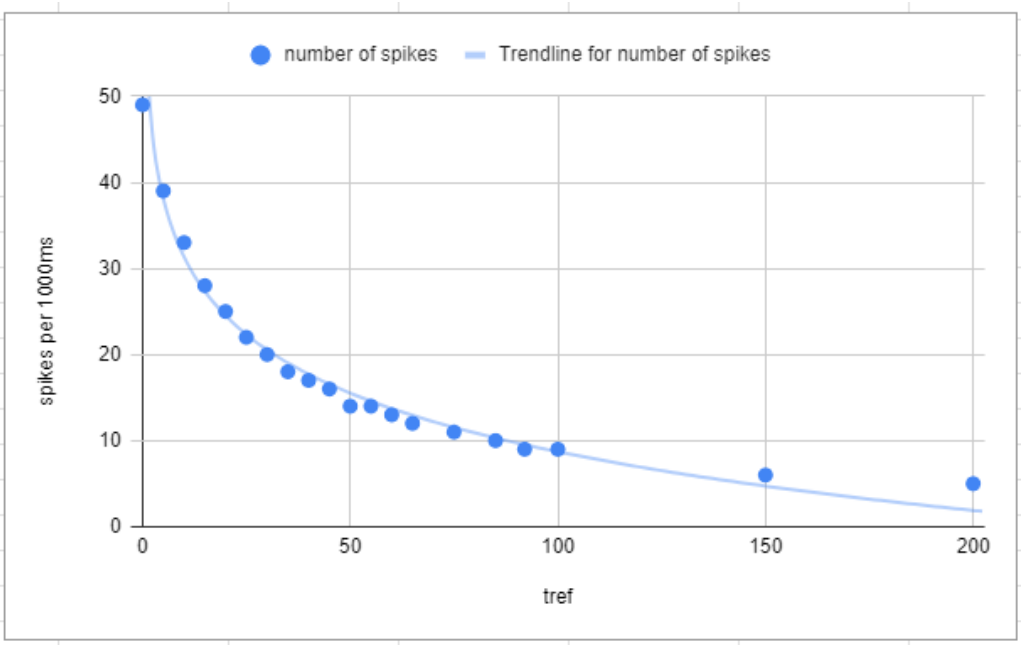
I_Inj = 201



Q2

The default settings where I_Inj = 230 and tref = 0 produce 49 spikes within 1000ms.

When tref = 1, this decreases slightly to 46. Raising tref each time by 1 shows similar decreases in spikes, as at tref = 5, the number of spikes is 39. There seems to be a correlation here, so I plotted the data on a scatter chart to see if there is a relationship of some sort.



Using a logarithmic trend line, we can see that as tref increases, the number of spikes decreases.

Task 2

Q1

The test image:

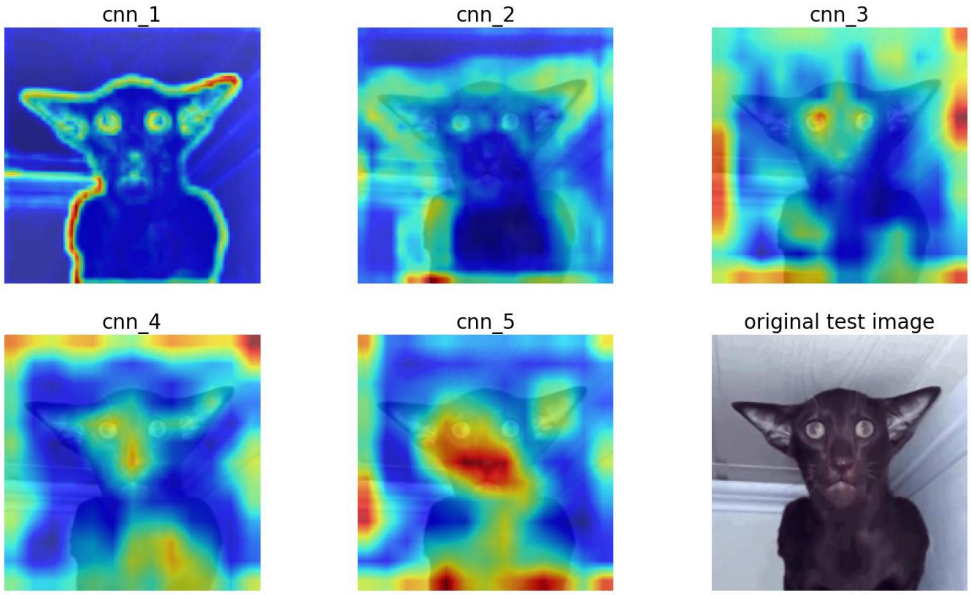
Output:

```
The first 5 classes of the testing image and their probabilities are:
-------------------------------------------------
Mexican hairless: 36.27%
miniature pinscher: 12.06%
Egyptian cat: 7.60%
Chihuahua: 6.93%
Italian greyhound: 5.73%
```

The image is of an Egyptian cat, so while the higher probable predictions are incorrect, the model did at least guess it at 7.6%.

Q2

Activation maps:



We can see that cnn_1 tries to look at the outline/shape of the animal. Cnn_3 seems to be looking at some of the background of the image. Cnn_4 seems to be looking at the ceiling. Cnn_5 seems to be focussing on the facial features/head of the cat.

Q3

Input image:

Classification results:

```
The first 5 classes of the testing image and their probabilities are:
--------------------------------------------
sombrero: 8.19%
cowboy hat, ten-gallon hat: 5.35%
fur coat: 4.41%
French bulldog: 3.17%
Boston bull, Boston terrier: 2.83%
```

The results do not line up with my own categorisation, as the image is of a person, and not of any of the classification predictions. This could be to the distance of the person to the camera being too close, which may be unusual or not present in the training data for the model, and therefore difficult for the model to classify.

Task 3

Error in code, cannot run even though jax is installed:

```
Running sample on GPU
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-23-e829b9f58ef9> in <cell line: 3>()
      6 else:
      7   print('Running sample on GPU')
----> 8   simple_ddm.sample(sampler="nuts_numpyro", cores=1,chains=1,draws=1500,tune=500 )

                                 8 frames
/usr/local/lib/python3.10/dist-packages/numpyro/ops/provenance.py in <module>
      8 from jax.interpreters.partial_eval import trace_to_jaxpr_dynamic
      9 from jax.interpreters.pxla import xla_pmap_p
---> 10 import jax.linear_util as lu
     11 import jax.numpy as jnp
     12

ModuleNotFoundError: No module named 'jax.linear_util'
```

```
!pip freeze | grep jax

blackjax==1.1.1
jax==0.4.26
jaxlib @ https://storage.googleapis.com/jax-releases/cuda12/jaxlib-0.
jaxopt==0.8.3
```

Part 2

B1

No username/password?

B2

Default settings after 20 generations:

Most of the population reached a shape with a large body and small wheels, with a front heavy bias. They reached far due to their stable nature and slow speed.

Mutation rate set to 100%, mutation size set to 100% after 20 generations:

Only one or two cars seemed to go far at a time. It has a very large front wheel almost the same size as its body, and a small rear wheel. The body seems to be front heavy to prevent flipping, although it seems to prioritise speed over stability.

Mutation rate set to 5%, mutation size set to 100% after 20 generations:

The dominant shape of the body here was very small and seemed to serve simply as a connector to both wheels. The vehicles would frequently get stuck by wheeling repeatedly. The overall shape of the vehicles looked this way at around 10 generations and didn't change up to and after 20 generations.

It seemed like the "fitness" of the algorithm seemed to be how far the vehicle travels. Certain traits becoming dominant and successful seem to be down to randomness. There were several different ways that seemed to allow the vehicles to go far at regular "earth gravity": a slow, evenly weighted "off roader", a fast, rear heavy "speedster" or an all rounded, front heavy "motorbike".

Changing the gravity level to much higher (Jupiter) seemed to allow the vehicles to go farther in a lower amount of time, presumably since the vehicles were less prone to flipping. Setting the gravity to something much lower (Moon) led the algorithm to try and make more stable shapes that would flip less due to wheelieing and in mid-air, most with a very large front wheel or a front-heavy body shape. These eventually reached an evenly weighted shape that allowed the vehicles to "jump" very far over the terrain and reaching the end of the map with ease.

B3

1. The code:

```
def calculate_weights(pattern):
    pattern = np.array(pattern)
    weights = np.outer(pattern, pattern)
    np.fill_diagonal(weights, 0)
    return weights
```

2. Input vector values:

```
pattern = [1, -1, 1, 1, -1, 1, 1, -1, -1]
```

3. Resulting matrix:

```
[[ 0 -1  1  1 -1  1  1 -1 -1]
 [-1  0 -1 -1  1 -1 -1  1  1]
 [ 1 -1  0  1 -1  1  1 -1 -1]
 [ 1 -1  1  0 -1  1  1 -1 -1]
 [-1  1 -1 -1  0 -1 -1  1  1]
 [ 1 -1  1  1 -1  0  1 -1 -1]
 [ 1 -1  1  1 -1  1  0 -1 -1]
 [-1  1 -1 -1  1 -1 -1  0  1]
 [-1  1 -1 -1  1 -1 -1  1  0]]
```

4. The diagonal is zeroed because a neuron should not have a connection with itself in a Hopfield network.