85-419

Modeling Project

SIMULATING RESPONSE PROPERTIES OF NEURONS IN AREA 7A OF POSTERIOR PARIETAL CORTEX

By Payton Chi, May 2022

Abstract

Neurons in area 7a of the posterior parietal cortex are responsible for encoding spatial information which is later transformed into a motor plan. Specifically, these neurons respond to the retinal location of a visual stimulus as well as the position of the eye and combine these signals to represent the location of an object in real space. The aim of this paper is to replicate the findings of David Zipser's and Richard A. Andersen's article "A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons". This paper uses Keras, a popular software library in Python for designing artificial neural networks, to implement the model laid out in the paper. Additionally, the accuracy and loss (average error over training data) are measured as the network trains. Lastly, there is a discussion on the findings and practical applications of the network as well as future experiments to be done.

Introduction

The posterior parietal cortex encodes spatial information from the visual system which is translated into motor planning and movement (Snyder et al. 1997). Specifically, area 7a of the posterior parietal cortex is responsible for integrating eye position and retinotopic visual information (retinal location) for spatial analysis (Andersen 1989). A majority of the cells examined in this area have visual receptive fields (Hyvarinen, 1981). Many of these visual cells also carry eye position signals, for many neurons visual excitability varies as a function of the position of the eyes in the orbit, that is, the angle of gaze. Three major classes of neurons in area 7a of the posterior parietal cortex are the visual cells responding to visual stimulation only, the eye-position cells responding to eye-position only, and the spatially tined cells responding to both eye position and visual stimulation (Zipser & Andersen, 1988). The first two classes of neurons can be thought of as the inputs while the third class of neuron integrates the first two producing the head-centered spatial coordinates. While the brain isn't literally doing any kind of back-propagation, it is still interesting to see how this model is able to accurate simulate an important biological process.

The advantage of an artificial neural network over a computer program is that the network will not need to know in advance the algorithm being used to computes such transformations and instead discover an appropriate algorithm as it learns. This is done via backpropagation which can program neural networks to compute arbitrary functions (Kienker et al., 1986). Back propagation networks have internal or hidden units that are free to take on the response properties to best accomplish the computational task being learned. It is the properties of theses hidden units that resemble those of cortical neurons (Zipser & Andersen, 1988).

Similar to David Zipser's and Richard A. Andersen's article "A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons", the question to be asked is whether a feedforward artificial neural network using backpropagation can accurately integrate retinotopic visual information (retinal location) and eye position to determine where an object is in real space. Additionally, can the network be replicated such that the finding in the article, which is that it can be integrated to produce a real space output, be substantiated. This paper hypothesizes that the artificial neural network from the article can be recreated to show that retinal location and eye position can be integrated to determine the location of an object in real space.

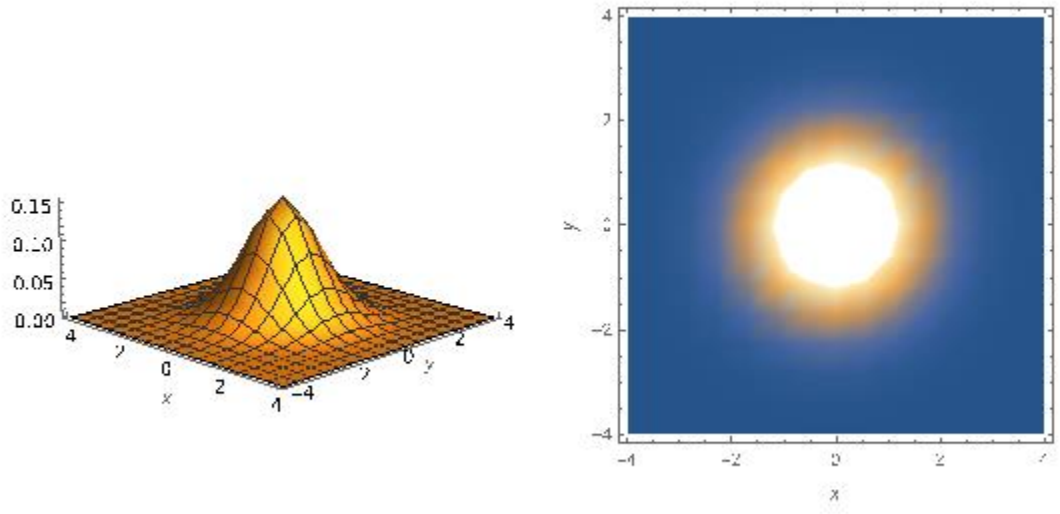
Methods

As previously mentioned, the network included two inputs: retinal position and eye position and a single output: spatial representation. The retinal position, eye position, and spatial representation was coded as a 1D array of values which was meant to represent a 9x9 unit graph (Fig. 1). The retinal position was implemented as multiple gaussian (normal) distributions representing receptive fields over a plane centered at the origin and each unit in the 1D array being the peak of that unit's receptive field and can be calculated as the joint pdf of the bivariate normal distribution given the x and y coordinates, means, and standard deviations for the x and y axis (Fig. 2). The eye position was implemented as a plane in real space (again as a 1D array) where a "1" was the spot in real space that the eye was looking at in head-center coordinates (Fig. 3). Furthermore, the important calculation regarding the eye position was how far from origin the eye was with respect to the center of a fixed spatial position, in head-center coordinates, in other words the difference in the x and y position from the origin was the spot is space that the eye was looking at. One thing to note is that the spot in real space that an eye was looking at does not necessary mean that was the spot in space of the "real" object because the object could still be in the periphery, which is where retinal location comes into play. Lastly, the spatial output is also represented as a bivariate normal distribution of receptive fields and is linear function of eye position in head-center coordinates.

Since the network will be learning a linear transformation of the pdf of a bivariate normal distribution (Fig. 2) over a 1D array, it will need a lot of data to train properly. Creating thousands of training examples by hand would have been very impractical, so pairs of retinal locations and eye positions were created by randomly setting a μ_x and μ_y for the array of values from a bivariate normal distribution (for retinal location) and randomly selecting a dx and dy (difference in x and y from origin) for the eye position array such that $\mu_x + dx$ and $\mu_y + dy$ would still lie withing the 9x9 unit plane. Lastly, the true output (teaching signal) was generated by creating an array of values from the bivariate normal distribution with mean in the x direction being $\mu_x + dx$ and mean in the y direction being $\mu_y + dy$ (a linear function of means from retinal location and differential from eye position). Again, these 1D arrays are meant to represent a two-dimensional plane (Fig. 1). This process was repeated 5000 times to create a training data set and 500 times to create a testing (validation) data set.

```
empty = np.array([
    0, 0, 0, 0, |, 0, 0, 0, 0,
    0, 0, 0, 0, |, 0, 0, 0, 0,
    0, 0, 0, 0, |, 0, 0, 0, 0,
    0, 0, 0, 0, |, 0, 0, 0, 0,
    -, -, -, -, x, -, -, -, -,
    0, 0, 0, 0, |, 0, 0, 0, 0,
    0, 0, 0, 0, |, 0, 0, 0, 0,
    0, 0, 0, 0, |, 0, 0, 0, 0,
    0, 0, 0, 0, |, 0, 0, 0, 0
])
```

Fig. 1 Conceptual representation of 1D array as a coordinate matrix with "x" representing the origin. Representation is only for demonstrative purposes and not how inputs/outputs were physically coded.



$$f(x,y) = \frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} \mathrm{e}^{-\frac{1}{2(1-\rho^2)}\left[\left(\frac{x-\mu_X}{\sigma_X}\right)^2 - 2\rho\left(\frac{x-\mu_X}{\sigma_X}\right)\left(\frac{y-\mu_Y}{\sigma_Y}\right) + \left(\frac{y-\mu_Y}{\sigma_Y}\right)^2\right]}$$

Fig. 2 Conceptual representation of retinal position in two dimensions (as 1D array) with the bivariate normal distribution acting as the distribution of the receptive fields. This example had $x = 0, y = 0, \mu_x = 0, \mu_y = 0, \sigma_x = 0, \sigma_y = 0$. This representation is for demonstrative purposes only and not how retinal position was physically coded. The function is the joint pdf of the bivariate normal distribution.

Fig. 3 Conceptual representation of eye position in two dimensions (as 1D array) where "x" represents origin and "1" would be where the eye is looking. In this case, the eye would be at a +2 differential in the x direction (dx=2) and a +1 in the y direction (dy=1). This input is almost identical to how eye position was actually coded, with the absence of the "x" in place of a 0 being the only difference, with the "x" in place for demonstrative purposes.

```
empty = np.array([
    0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 1, 0, 0,
    0, 0, 0, 0, x, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0
])
```

The neural network was created on a Jupyter Notebook through the Anaconda software distribution. Specifically, the network was coded in Python using a TensorFlow environment which, primarily, included Keras, an open-source software library that provides an interface for artificial neural networks (Keras Team, 2022); other libraries were used for data visualization and data manipulation, however Keras is the most significant of theses libraries. Additionally, the network was made using Keras' functional API which is designed to allow networks to handle multiple inputs, in comparison to Keras' Sequential model class which allows only a single input. The model is 3-layer feed-forward network with backpropagation trained to map visual targets to head centered coordinates given any arbitrary pair of eye and retinal positions. Specifications were laid out by the paper being simulated/recreated (Zipser & Andersen, 1988):

- The input layer has 2 sections: a set of units representing visual stimulus (aka retinal position) and a set of units representing eye position. These two input layers are then concatenated into a single layer whose size was the sum of the two inputs layers (each input is a 9x9 unit graph which corresponds to a 1D array of size 81 for a total concatenated layer of size 162).
- The second layer consists of 30 hidden units which map the input to the output.
- The output layer consists of a 9x9 unit graph (implemented as 1D array) mapping spatial location in head-centered coordinates.
- The output of each unit of the hidden and output layers is computed as an S-shaped logistic function (sigmoid function) of the synaptic strength weighted sum of inputs, plus a bias term
- The training paradigm uses back propagation learning which consists of choosing an input and desired outputs and applying the inputs to the first layer of the network and propagation the activity it generates through the network to the output units. The actual output or teacher signal is then subtracted from the desired output to generate an error. This error is used to adjust the weights of synapses on the hidden and output layer units. This is done via the .fit() method on Keras.

Networks on Keras also required the loss function and optimizer be specified, in this case the built-in loss function binary crossentropy was used and built-in optimizer rmsprop was used. The loss function, binary crossentropy is a probabilistic loss function which was chosen because the output is represented as multiple probabilities. Additionally, among all the probabilistic loss functions, this one worked the best in terms of accuracy. The optimizer, rmsprop is a standard SDG (stochastic gradient descent) optimizer. The summary of the model (Fig. 4) as well as the directed graph (Fig. 5) show the outline and layering of the network.

```
Model: "model"
_________________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================================
retinal position (InputLayer)   [(None, 81)]         0
_________________________________________________________________________________________________
eye position (InputLayer)       [(None, 81)]         0
_________________________________________________________________________________________________
concatenate (Concatenate)       (None, 162)          0           retinal position[0][0]
                                                                 eye position[0][0]
_________________________________________________________________________________________________
hidden (Dense)                  (None, 30)           4890        concatenate[0][0]
_________________________________________________________________________________________________
output (Dense)                  (None, 81)           2511        hidden[0][0]
=================================================================================================
Total params: 7,401
Trainable params: 7,401
Non-trainable params: 0
_________________________________________________________________________________________________
```

Fig. 4 The summary of the model which takes in 2 inputs (retinal position and eye position) of size 81 and concatenates them into a single layer of size 162 (sum of inputs) then passes through a hidden layer with 30 units (size 30) which connects to the output layer, outputting an array representing a 9x9 grid.

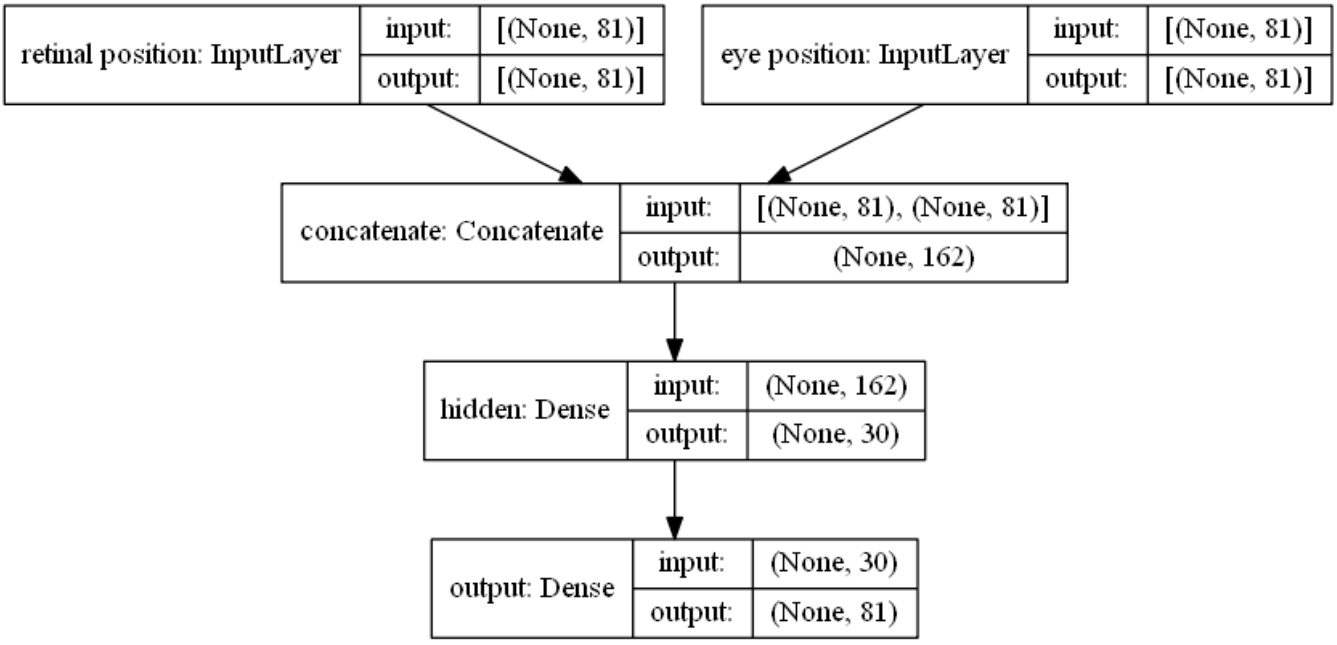


Fig. 5 Model represented as a directed graph of layers starting with inputs (top) and ending with the output (bottom) with a concatenated and hidden layer in between.

Results

The network ran for 500 epochs in batches of 100 samples and simultaneously tested the accuracy and loss (average error over training data) of the training and validation (testing) data set. The accuracy of the training and validation data set seems to steadily increase until about 200 epochs then tappers off, increasing in accuracy very slowly (Fig. 6). The loss of the training and validation data set decreases very fast over just a few epochs and stays very small over most of the training. The graphs are further substantiated by the predicted output for the test data which seems to be very close to the actual target (Fig. 8). While the predicted output it a bit different form the target, it is a small difference and, for most values, a rounding error.

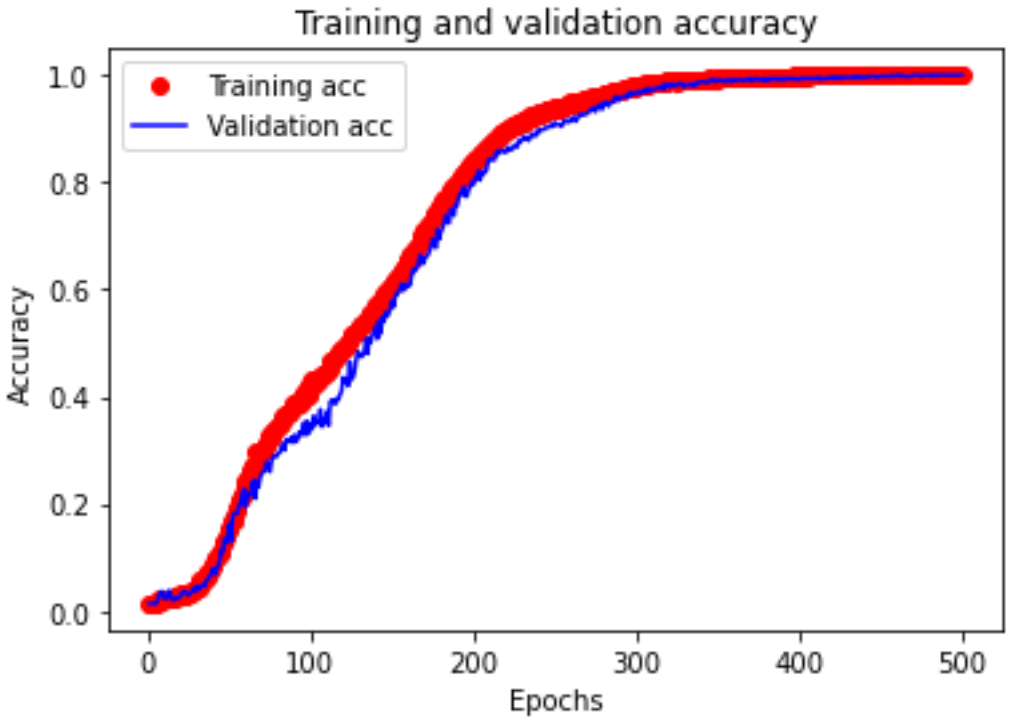


Fig. 6 The accuracy of the model when compared to the target/teaching signal (separate data set). The blue line being the validation (testing) data set and the red dots being the training data set. Most of the epochs are spend increasing the accuracy from 0.9 to 1.

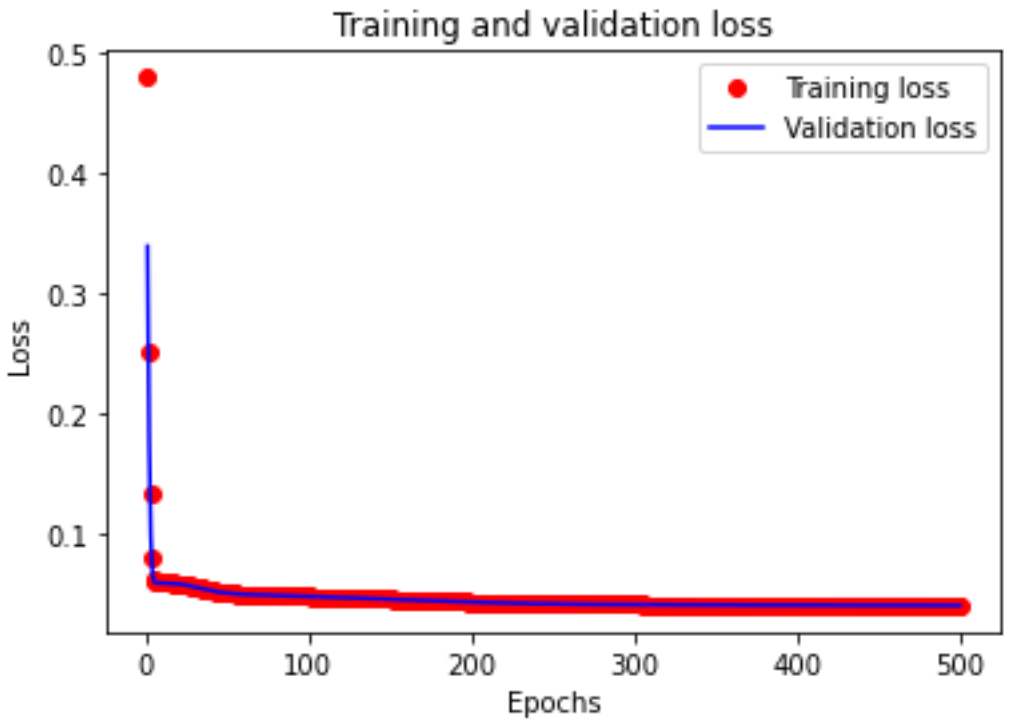


Fig. 7 The loss, or average error, of the model showing the training loss (red dots) and validation loss (blue line). The loss drops very quickly over a few epochs and stays very low for most of the epochs.

```
#target
[
 1.22250169e-15 2.21033492e-12 1.47018578e-09 3.59742598e-07 3.23829967e-05 1.07237757e-03 1.30642333e-02 5.85498315e-02 9.65323526e-02
 2.01556455e-15 3.64422619e-12 2.42392656e-09 5.93115274e-07 5.33905355e-05 1.76805171e-03 2.15392793e-02 9.65323526e-02 #1.59154943e-01
 1.22250169e-15 2.21033492e-12 1.47018578e-09 3.59742598e-07 3.23829967e-05 1.07237757e-03 1.30642333e-02 5.85498315e-02 9.65323526e-02
 2.72776999e-16 4.93192384e-13 3.28042787e-10 8.02694235e-08 7.22562324e-06 2.39279779e-04 2.91502447e-03 1.30642333e-02 2.15392793e-02
 2.23908996e-17 4.04836961e-14 2.69273918e-11 6.58891552e-09 5.93115274e-07 1.96412803e-05 2.39279779e-04 1.07237757e-03 1.76805171e-03
 6.76146580e-19 1.22250169e-15 8.13136774e-13 1.98968008e-10 1.79105293e-08 5.93115274e-07 7.22562324e-06 3.23829967e-05 5.33905355e-05
 7.51131001e-21 1.35807671e-17 9.03313360e-15 2.21033492e-12 1.98968008e-10 6.58891552e-09 8.02694235e-08 3.59742598e-07 5.93115274e-07
 3.06970072e-23 5.55014911e-20 3.69163524e-17 9.03313360e-15 8.13136774e-13 2.69273918e-11 3.28042787e-10 1.47018578e-09 2.42392656e-09
 4.61510838e-26 8.34431169e-23 5.55014911e-20 1.35807671e-17 1.22250169e-15 4.04836961e-14 4.93192384e-13 2.21033492e-12 3.64422619e-12
 ]

#prediction
[
 1.68449421e-11 7.85523924e-10 5.58514088e-08 2.93843982e-06 6.76410564e-05 8.63999128e-04 8.43584538e-03 4.43303287e-02 8.27622116e-02
 2.06104543e-11 6.62640498e-10 6.85578243e-08 5.43630631e-06 1.36554241e-04 1.59433484e-03 1.39144957e-02 6.75511360e-02 #1.20233715e-01
 1.55090853e-11 4.16598617e-10 5.58592923e-08 4.50913876e-06 1.04583036e-04 1.24219060e-03 9.83637571e-03 4.26546931e-02 7.26875663e-02
 8.40212188e-12 2.66564132e-10 2.48254999e-08 1.65139613e-06 3.76639327e-05 4.24325466e-04 3.25033069e-03 1.21262074e-02 1.83120072e-02
 1.16709051e-12 5.83320753e-11 5.86515458e-09 3.97358036e-07 7.61275396e-06 6.57161363e-05 4.74691391e-04 1.60032511e-03 2.04640627e-03
 8.19361651e-15 1.01413669e-12 2.45365311e-10 2.24957457e-08 5.14593808e-07 3.59205160e-06 2.42133647e-05 7.78011599e-05 9.82377678e-05
 2.17575782e-17 1.30146263e-15 5.72299640e-13 1.40792641e-10 5.50513857e-09 4.42933086e-08 3.06861864e-07 1.10599353e-06 1.96854671e-06
 1.05821675e-19 6.11053952e-18 2.38924955e-15 1.08295393e-12 5.60709916e-11 5.70169412e-10 4.52139570e-09 2.33437838e-08 7.41250119e-08
 2.95029498e-23 5.36513182e-21 1.76420188e-17 4.45578836e-14 5.65590413e-12 6.47690235e-11 5.57866253e-10 2.95083913e-09 5.84478510e-09
]
```

Fig. 8 The target output for a single pair of inputs (retinal location and eye position) and predicted output for those same inputs. The outputs displayed here might differ from other iterations of experiment because data is randomly generated each time the Jupyter kernel runs (explained in Methods). The "#" next to the value at distance of +4 from the origin in the x-direction and +3 in the y-direction is the peak value of the bivariate normal distribution over the plane and is very similar in value and, most importantly, in position; that is to say the peak is in the same spot in each plane and is of similar value.

Discussion

The original question was whether the network in David Zipser's and Richard A. Andersen's article "A back-propagation programmed network that simulates response properties of a subset of posterior parietal neurons" could be replicated to show that retinotopic visual information (retinal location) and eye position can be integrated to determine where an object is in real space. This paper hypothesized that the feedforward network with back propagation from the article could, in fact, be replicated to show that retinal location and eye position could, in fact, integrate to determine the location of an object in real space. This hypothesis is supported from the results of the experiment. Specifically, the network is able to accurate predict the target output of data it has not seen before. In other words, the model has high validation accuracy (Fig. 6) with the difference in target output and predicted output being nominally different (Fig. 8). While the predicted outputs from the model was very similar to the target output, there was still a bit of error, though very tiny. However, this would probably be remedied by additional training examples or additional epochs. Additionally, the similarity in pattern for both the training and validation data sets for both the accuracy and loss graphs shows that the model is not being overfit for the training data. Instead, the model is correctly identifying patterns in how the inputs are related and not just memorizing how to correlate the training inputs to the training outputs.

Since the artificial neural network is able to accurately output the location of an object in real space, it is in agreement with the function of neurons in area 7a of the posterior parietal cortex. Therefore, this network works as an accurate computation model for spatial analysis, an important neurological process. Additionally, this model could be used to help describe other, important, and related biological processes, such as motor planning and movement. A practical application of a computational model such as this is the use of BCIs.

Brain-computer interfaces (BCIs) link the brain to the external world by computer processing the recorded neural signal to extract the subject's command to control an external device, like a prosthetic arm (Schwartz 2004). This is done by decoding neural spike data, typically from motor cortical cells in primary motor cortex (Schwartz 2004). Specifically, electrodes record the spike data from motor neurons and decode them into various outputs relating to motor command, like reach angle (Santhanam et al., 2006), depending on the type of neuron being recorded from. BCIs themselves are applicable to many different fields for many different purposes. In this case, the neural network could be used in conjunction with a prosthetic arm to create a BCI that executes motor commands. Specifically, the network would determine where an object is in real space while the arm interfaces with the motor system to create a BCI that is able move a prosthetic arm, for example, to a place in real space. In other words, the artificial neural network would aid the BCI in motor planning.

Additional experiments could be done to improve the upon the model or even understand the relationship between retinal location and eye position better. For example, the network could be trained with different number of hidden units. In this experiment, the network was trained on 30 hidden units, however additional hidden units could improve performance, or fewer hidden units could be used to achieve the same performance, which makes the network more efficient. The size of the receptive fields for the retinal location could be adjusted by changing the standard deviations of the bivariate normal distribution, this could be an interesting experiment to investigate the affect of retinotopic receptive field size on spatial output and efficiency. Lastly, the effect of lesioning in area 7a of the posterior parietal cortex could be simulated by adding in randomness to the receptive fields of the retina. Furthermore, the affect of lesioning on spatial analysis could be investigated.

References

Andersen, R A. “Visual and Eye Movement Functions of the Posterior Parietal Cortex.” *Annual Review of Neuroscience*, vol. 12, no. 1, 1989, pp. 377–403., https://doi.org/10.1146/annurev.ne.12.030189.002113.

Hyvärinen, Lea, et al. “Contrast Sensitivity Function in Evaluation of Visual Impairment Due to Retinitis Pigmentosa.” *Acta Ophthalmologica*, vol. 59, no. 5, 1981, pp. 763–773., https://doi.org/10.1111/j.1755-3768.1981.tb08744.x.

Kienker, Paul K, et al. “Separating Figure from Ground with a Parallel Network.” *Perception*, vol. 15, no. 2, 1986, pp. 197–216., https://doi.org/10.1068/p150197.

Santhanam, Gopal, et al. “A High-Performance Brain–Computer Interface.” *Nature*, vol. 442, no. 7099, 2006, pp. 195–198., https://doi.org/10.1038/nature04968.

Schwartz, Andrew B. “Cortical Neural Prosthetics.” *Annual Review of Neuroscience*, vol. 27, no. 1, 2004, pp. 487–507., https://doi.org/10.1146/annurev.neuro.27.070203.144233.

Snyder, L. H., et al. “Coding of Intention in the Posterior Parietal Cortex.” *Nature*, vol. 386, no. 6621, 1997, pp. 167–170., https://doi.org/10.1038/386167a0.

Team, Keras. “Simple. Flexible. Powerful.” *Keras*, https://keras.io/#why-this-name-keras.

Zipser, David, and Richard A. Andersen. “A Back-Propagation Programmed Network That Simulates Response Properties of a Subset of Posterior Parietal Neurons.” *Nature*, vol. 331, no. 6158, 1988, pp. 679–684., https://doi.org/10.1038/331679a0.