

# BIOLOGICALLY INSPIRED NEURAL COMPUTATION

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER  
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY  
IN THE FACULTY OF SCIENCE AND ENGINEERING

2022

Adam William Perrett

Department of Computer Science

# Contents

<b>Acronyms</b>	<b>16</b>
<b>Abstract</b>	<b>19</b>
<b>Declaration</b>	<b>21</b>
<b>Copyright</b>	<b>22</b>
<b>Acknowledgements</b>	<b>23</b>
<b>1 Introduction</b>	<b>24</b>
1.1 Introduction . . . . .	24
1.2 Hypothesis - Research questions . . . . .	25
1.3 Contributions . . . . .	26
1.4 Publications . . . . .	26
1.5 Thesis structure . . . . .	27
1.5.1 Chapter 2 - Background . . . . .	28
1.5.2 Chapter 3 - Visual attention . . . . .	28
1.5.3 Chapter 4 - E-prop on SpiNNaker . . . . .	29
1.5.4 Chapter 5 - Error driven neurogenesis . . . . .	29
1.5.5 Chapter 6 - Conclusions . . . . .	30
<b>2 Background</b>	<b>31</b>
2.1 Understanding biological intelligence . . . . .	31
2.2 Learning how to learn . . . . .	35
2.3 The medium matters . . . . .	40
2.4 Summary . . . . .	45

<b>3</b>	<b>Visual attention</b>	<b>46</b>
3.1	Introduction . . . . .	46
3.2	Background . . . . .	47
3.3	Event-based SNN proto-object model of saliency . . . . .	50
3.4	Experiments and Results . . . . .	56
3.5	Conclusion . . . . .	66
<b>4</b>	<b>E-prop on SpiNNaker</b>	<b>70</b>
4.1	Introduction . . . . .	70
4.2	Background . . . . .	71
4.3	Online and local learning . . . . .	73
4.3.1	E-prop . . . . .	74
4.3.2	SpiNNaker . . . . .	75
4.4	Implementation and experimental design . . . . .	75
4.4.1	An overview of e-prop . . . . .	75
4.4.2	E-prop neuron models . . . . .	76
4.4.3	SNN architecture & mapping to SpiNNaker . . . . .	79
4.4.4	Tasks . . . . .	81
4.5	Results . . . . .	85
4.5.1	Wave-form matching . . . . .	85
4.5.2	Temporal credit assignment . . . . .	87
4.5.3	Firing rate (ir)regularisation . . . . .	88
4.6	Discussion . . . . .	93
<b>5</b>	<b>Error driven neurogenesis</b>	<b>96</b>
5.1	Introduction and background . . . . .	96
5.2	Implementation and experimental design . . . . .	100
5.2.1	Error Driven Neurogenesis algorithm design . . . . .	100
5.2.2	Task specific alterations . . . . .	108
5.2.3	Related algorithms . . . . .	112
5.3	Results . . . . .	114
5.3.1	Visualising the expectation . . . . .	120
5.3.2	Visualising the receptive fields . . . . .	121
5.3.3	Inverted pendulum - reinforcement learning . . . . .	122
5.3.4	Parametric analysis . . . . .	123
5.4	Discussion . . . . .	127

<b>6</b>	<b>Conclusions</b>	<b>132</b>
6.1	Summary and conclusions . . . . .	132
6.2	Future work . . . . .	136
<b>A</b>	<b>Appendix</b>	<b>158</b>
A.1	Error Driven Neurogenesis appendix . . . . .	158

**Word Count: 33671**



# List of Tables

3.1	The percentage firing thresholds for different population connections, input->filter is the only inhibitory connection. Percentage firing threshold is the percentage of the pre-synaptic population that need to fire to produce a spike in the post-synaptic population. Inhibitory connections do not induce a spike but are scaled in the same fashion. This metric is used to standardise weights across varying convolutional kernel sizes.	55
3.2	Qualitative comparison among the PyTevProto and the SNNevProto. From the left column to the right column: the example number, a RGB image representing the scene shown to iCub (the input stimulus), PyTevProto saliency map and SNNevProto saliency map. This table shows only results from clutter experiments of the SalMapIROS dataset. The events are recorded directly from the event-driven cameras mounted on iCub's eyes. The objects and the 2D printed patterns are placed on a desk in front of the robot. . . . .	58
3.3	Metrics summary. This table takes inspiration from [19] . . . . .	59
3.4	Table showing the number of neurons and SpiNNaker boards required given a percentage of overlapping (OL) for the VM filters. The spalloc server was used to run these jobs which allocates boards in multiples of 3. . . . .	59
3.5	Results of latency in milliseconds for different datasets of SalMapIROS. The test is done measuring the latency of two different samples for each dataset. Each row represents a dataset used to measure the latency in two separate samples. Each dataset represents static and dynamic objects placed in front of iCub (such as a paddle, a puck, calibration circles, proto-object patterns, a mouse, a cup and clutter (see Fig. 3.4)	64

# List of Figures

32figure.caption.10

- 2.2 A typical artificial neuron: real-valued inputs,  $x_i$ , (including a bias) are passed along weighted synapses,  $w_i$ , and summed at the neuron. The weighted sum is passed through some nonlinear activation,  $\sigma$ , to produce the neuron output. . . . . 33
  
- 2.3 A typical artificial spiking neuron: inputs are discrete spikes,  $s_i$ , in time (a binary value indicating a spike at time  $t$  or not), spikes pass along weighted synapses,  $w_i$ , and activation collects at the neuron. The contribution of the weighted inputs is added to the neuron's membrane voltage (its internal state,  $v_{mem}$ ); the membrane voltage will decay with time but if enough input is received and it crosses its threshold,  $v_{th}$ , the neuron will release a spike and  $v_{mem}$  will be reset to the resting voltage  $v_{rest}$ ; following a spike there is a refractory period,  $t_{refract}$ , in which the neuron cannot spike again. . . . . 34
  
- 2.4 An example of a parameter being updated via gradient descent. The gradient of the error with respect to a parameter is calculated at a point and the parameter (red cross) is updated in the direction that the gradient (black dotted line) points towards reducing error. The parameter is continuously updated this way, possibly even overshooting the optimal value, until some termination criteria is met. . . . . 37

2.5	The middle plot shows an input data stream, this can be imagined as a single photo sensitive element of a charge-coupled device (CCD). The top plot shows an example of how a frame-based sensor would collect information from the sensor; with regularity in time the current value of the sensor is recorded. The bottom plot shows how an event-based sensor records data; the current value is set following a threshold crossing and when the value next moves a threshold distance away from the stored value an event is triggered. An ON event (green) signifies the value is a threshold above the stored value and an OFF event (red) signifies it went a threshold below the stored value. The red lines connecting dots gives a rough example of how the signal could be reconstructed from the stored values. Both frame-based and event-based sensors recorded 10 data points yet the event-based sensor retains more of the original signal, owing to its asynchronous sensing, and filters the relatively quiescent parts. . . . .	41
2.6	A labelled SpiNNaker chip. Not shown is the DTCM (Data Tightly-Coupled Memory) which is physically mounted on top of the cores and router and stitch-bonded to it. . . . .	43
2.7	Each core is capable of simulating up to 1000 neurons, 18 cores compose a SpiNNaker chip, 48 chips are on a single SpiNN5 board, 24 boards are in a rack with 5 racks per cabinet for a total of 10 cabinets. In total this means the over 1 million core machine could in theory simulate around 1 billion neurons. . . . .	44

3.1	An overview of the model architectures for the PyTevProto (on the left) and the SNNevProto (on the right). The events are split based on the polarity and fed into the two models as input. The event-based model generates different scales by subsampling the "event-frame" and creating a pyramid. The resulting scaled "event-frames" are convolved with VM (Von Mises) filters at 4 different orientations (Border Ownership Pyramid) and grouped at the Grouping Layer which processes the input with the two layers of Border Ownership and Grouping Pyramids. The red lines are inhibitory signals. The spike-based implementation processes the events asynchronously exploiting layers of VM shaped neuron receptive fields at different scales and rotations. The Proto-Object Neurons (Grouping Pyramid Layer) integrate the response connecting VM filters with opposite rotation and pool the response from different scales. The outcome of both models is the saliency map. . . .	51
3.2	Representation of the VM filter described in Eq. 3.1 at $0^\circ$ . . . . .	51
3.3	Representation of a VM layer and its connections. Each VM filter is split into 4 sections all connected to the same Filter neuron. The blue area around the "active" part of the neuron (the moon shaped yellow region) is connected to the Filter neuron with an Inhibitory connection (red lines). This stage of the model represents the Border Ownership pyramids detecting closed contours. Two complementary VM filters with opposite orientation are then connected to the same Proto-Object Neuron (Grouping Pyramid) to identify possible proto-objects. This structure is repeated for each layer with different orientations of the filter: $0^\circ$ , $45^\circ$ , $90^\circ$ and $135^\circ$ . . . . .	52
3.4	Qualitative comparison of the PyTevProto and the SNNevProto. From the left column to the right column: the example number, an RGB image representing the scene shown to iCub (the input stimulus), PyTevProto saliency map and SNNevProto saliency map. These examples are a selection from 13 scenarios of the SalMapIROS dataset. The events are recorded directly from the event-driven cameras mounted on iCub's eyes. The objects and the 2D printed patterns are placed on a desk in front of the robot. The RGB input images are only for a better visualisation of the input stimulus. . . . .	57

3.5	Comparison with different metrics evaluating the similarity between the SNNevProto saliency maps and the PyTevProto saliency maps [63] using the SalMapIROS dataset exploring different OL percentages ( <b>a</b> ) exploring a range of inhibition percentage firing thresholds ( <b>b</b> ) ( $\%/ \mu S$ conductances) with fixed OL percentage at 60%. The metrics used are: the Normalized Scanpath Saliency (NSS), Area under the ROC Curve (AUC-Borji) & (AUC-Judd), Pearson's Correlation Coefficient (CC) and Similarity (SIM) [15, 16, 19, 65], Structural Similarity (SSIM) and Mean Square Error (MSE). A higher score is better for all excluding the MSE where the lower score determines similarity. . . . .	60
3.6	Representation of examples from the NUS3D (robot scenario) dataset. The three columns represent the input RGB image, the outcome from the SNNevProto and the related ground truth from the NUS3D dataset. These examples show how the model performs when the observer fixation maps focus on objects. The response from the model is with 60% OL and 0.013 inhibition. . . . .	62
3.7	Representation of random chosen examples from the NUS3D (random subset) dataset. The three columns represent the input RGB image, the outcome from the SNNevProto and the related ground truth from the NUS3D dataset. These examples show how the model performs when the observer fixation maps are sparse and unclear. The response from the model is with 60% OL and 0.013 inhibition. . . . .	62
3.8	Comparison with different metrics evaluating the similarity of the SNNevProto saliency maps with the NUS3D fixation maps (ground truth) [76] in two different subsets (robot scenario ( <b>a</b> ) and random subset ( <b>b</b> )) for different OL percentages. The metrics used are: the Normalized Scanpath Saliency (NSS), Area under the ROC Curve (AUC-Borji) & (AUC-Judd), Pearson's Correlation Coefficient (CC) and Similarity (SIM) [15, 16, 19, 65], Structural Similarity (SSIM) and Mean Square Error (MSE). A higher score is better for all excluding the MSE where the lower score determines similarity. . . . .	63

4.1	A diagram from the original e-prop paper published in Nature [11]. It shows how the eligibility, $e_{ji}^t$ , at time $t$ is synapse specific, in this case between neuron $i$ and $j$ . Inputs, $x_i$ , are passed into the network at each timestep as well as the neuron specific learning signal, $L_j^t$ , generated by the error module which is a part of the task environment. . . . .	77
4.2	(a) Generic spiking neural network architecture suitable for training with e-prop: input neurons provide network stimulation; hidden neurons perform the computation; and readout neurons capture network output. During training, the readout neuron population additionally computes an error $E$ via supervision, and communicates this to the rest of the network to drive learning. (b) Populations of neurons are partitioned and mapped on individual SpiNNaker cores, with synaptic information stored local to the postsynaptic core. . . . .	80
4.3	Experimental setup of the temporal credit assignment task. The mouse starts at the beginning of a hallway and is presented with left and right cues as it progresses down. Each cue lasts 100ms with a 50ms gap between each. There is a 1 second wait following the final cue before a prompt signal is sent for the network to make a decision which of left or right presented the most cues. . . . .	84
4.4	Wave-form matching task. Top: repeating staggered input spikes. Middle: initial performance before any learning and after one weight update (indicated by dashed line). Bottom: Converged performance after 200 presentations of the inputs. . . . .	85
4.5	Error for each presentation of a target wave-form as seen in the experiment of Fig. 4.4. Each presentation lasts 1024 timesteps with a timestep size of 1ms, error is accumulated over the test to give the iteration error. Performance rapidly improves with convergence at around the 65th presentation. . . . .	86

4.6	Performance of e-prop on the left-right task. The black line shows the running average accuracy over the last 64 tests with the required threshold performance shown in the horizontal dashed green line. The dashed red line displays performance for random action selection. The vertical blue line displays the trial at which the network achieved over 90% accuracy over 64 tests and the number of cues is increased. The first blue line shows completion of the 1 cue task and the last the 7 cue task, network parameters are retained between transitions. . . . .	87
4.7	A comparison of firing rate estimation with a constant rate across 8 neurons (top plot) and 256 neurons (bottom plot) using exponential decay (blue) and a running average (green). The average Poisson firing rate is shown with the red line and the actual spike times for each neuron are represented by the black dots in the bottom half of each plot.	90
4.8	A comparison of firing rate estimation with bursting behaviour across 8 neurons (top plot) and 256 neurons (bottom plot) using exponential decay (blue) and a running average (green). The average Poisson firing rate is shown with the red line. In this instance the 10Hz Poisson firing of 2048ms is condensed into 12.85ms to give an extreme example of bursting behaviour. . . . .	92
5.1	(a) the general structure of the EDN (Error Driven Neurogenesis) algorithm. First an input is presented which produces an associated error. If the magnitude of the error is above the error threshold then neurogenesis is triggered. Input synapses of the newborn neuron are selected and the values of the current inputs set the centre of the kernel function on the synapses. An output synapse connects the neuron to outputs whose error was above threshold with a weight proportional to the error produced. (b) an example topology created with EDN. The different colours connected to the hidden neurons signify different sub-sampled input vectors, which have been stored on the synapses; they all contribute equally to neuron activity. The colour and thickness of connections between the hidden layer and the outputs displays that output connections are weighted and the magnitude can be negative or positive. The weight of these connections is determined during the training process by the error produced when neurogenesis is triggered.	102

- 5.2 A comparison of a standard ANN (Artificial Neural Network) neuron with ReLU activation and an EDN neuron's activation. The black dot is a single data point, the pink shaded area shows the output activation of the neuron in both cases, with a deeper colour representing a higher level of activation. The red and blue shaded areas represent the level of activation of each synapse, which possess the triangle kernel shown in the bottom right, with spread  $s$  and centred at individual values of  $v$ . The EDN neuron's activity is the average of the incoming synapse activity and is also passed along an outgoing synapse using a triangle kernel with the same  $s$  value and  $v = 1$ . This output synapse acts to threshold neuron activity and create the bounded purple area in the input space in which the neuron is active. . . . . 105
- 5.3 A selection of training steps of EDN (top) and an ANN trained with GD (Gradient Descent) (bottom) are shown for a toy classification example. The data set is composed of three classes: blue, green and red. Class boundaries are drawn in a darker shade than the data points showing what output each model would associate to that point in space. The white area around the EDN plots indicates that there are no output values at that point in the input space. . . . . 107
- 5.4 A example of how output values of 0 and 1 are combined to create a position on the regression scale. As described in Equation 5.8 the output value for the real part of the scale (red) is divided by the total of the real and inverted (blue) outputs to create the position on the regression scale, in this case  $\frac{26}{26+14} = 0.65$ . This value is then scaled to the full range of regression values possible in the task to produce the estimated regression value of the input. . . . . 109
- 5.5 A comparison of testing accuracy during training on the wine cultivar classification task of EDN (black) and an ANN trained via gradient descent using Adam (Adaptive moment estimation) optimisation (blue). An inset is shown for the first few iterations of training to display the initial emptiness of the EDN network producing no output and the initialisation bias of the ANN network already achieving testing accuracy equivalent to random choice. The fast acquisition of information in EDN allows it to overtake the testing accuracy of GD before the first batch update is done at the 8th training instance. . . . . 115



5.6	A comparison of an ANN trained via gradient descent using Adam optimisation (red and blue, with $lr$ being the learning rate and $b$ being the batch size) against EDN (black) applied to the auto-mpg non-linear regression task task. A zoomed in inset is given to display the learning curve of EDN. Batch sizes and learning rates were chosen which gave the fastest convergence in gradient descent. . . . .	117
5.7	A comparison of an ANN trained in tensorflow using Adam optimisation (blue) against EDN with surprise driven input selection (black) and random input selection (red) applied to the MNIST (Modified NIST) classification task. A batch size of 64 is used to train the ANN and the moving average of the last 50 batches is used to calculate the running training accuracy. EDN training accuracy is the moving average of the last 3200 ( $50 \times 64$ ) training examples as it does not perform batch updates. Training accuracy is used as since this is the first epoch none of the data has been seen before and therefore it is equivalent to testing accuracy. . . . .	119
5.8	A visualisation of the expectation for the class 3 retrieved from the parameters stored by EDN during training on MNIST. A range of surprise thresholds, $s_{th}$ , are given to show how this effects the stored values and the subsequent effect on testing accuracy after a single epoch through the training set. . . . .	120
5.9	The values of $v$ stored on each neuron's incoming synapses form a record of the inputs which were captured by EDN during training. The weight on the synapse connecting the neurons to the outputs enables the receptive field of each class to be determined by multiplying the stored values by their neuron's associated weights. When taking the positively weighted neurons of each class and multiplying their stored $v$ values by their associated output weight you create the first row of the plot, this forms a weighted expectation of each class. The second row is generated by also including the negatively weighted neurons connected to each output, showing the average receptive field of each class. The bottom row is an unweighted combination of the input synapses which are instances of each class. . . . .	121

5.10	EDN with varied network size limits benchmarked against an ANN trained in tensorflow using an actor-critic model (black) applied to the inverted pendulum task. The lines show the running average of the last 100 trials with the dashed line showing the threshold performance required for the task to be considered solved. Each configuration is repeated 100 times and the average of their performance is shown. . .	123
5.11	Showing the effect of range of parameter values for kernel spread $s$ (top) and surprise threshold $s_{th}$ (bottom) on neuron and synapse count after training on the wine classification task. The left y-axis and the line in blue of each plot corresponds with neuron counts. The right y-axis and the red line correspond with the synapse count. The vertical bars show the standard error over a stratified 10 fold cross validation. .	125
6.1	On the left shows the EDN network output after training on a three class problem. Dots are the training points with the dots with stars representing the saved data points and the larger dots are the weighted average of each class. On the right a tanh neuron is created which draws a hyperplane between all pairs of class centroids with their outputs corresponding to the respective class on each side of the hyperplane. When observing the outputs of the tanh neurons (the coloured background) it can be seen that a good class separation is acquired using nonlinear neurons without any gradient descent. . . . .	138
A.1	How the testing accuracy during training changes with error threshold, $E_{th}$ . . . . .	159
A.2	How the neuron and synapse counts following two epochs changes with error threshold, $E_{th}$ . . . . .	159
A.3	How the running average (moving window over 100 examples) of absolute testing error changes with error threshold, $E_{th}$ . . . . .	160
A.4	How the testing accuracy during training changes with kernel spread size, $s$ . . . . .	160
A.5	How the neuron and synapse counts following two epochs changes with kernel spread size, $s$ . . . . .	161
A.6	How the running average (moving window over 100 examples) of absolute testing error changes with kernel spread size, $s$ . . . . .	161

A.7	How the testing accuracy during training changes with different random sample sizes . . . . .	162
A.8	How the neuron and synapse counts following two epochs changes with different random sample sizes . . . . .	162
A.9	How the running average of absolute testing error changes with different random sample sizes . . . . .	163
A.10	How the testing accuracy during training changes with surprise threshold, $s_{th}$ . . . . .	163
A.11	How the neuron and synapse counts following two epochs changes with surprise threshold, $s_{th}$ . . . . .	164
A.12	How the running average (moving window over 100 examples) of absolute testing error changes with surprise threshold, $s_{th}$ . . . . .	164

# Acronyms

**Adaptive moment estimation (Adam)** Adam optimisation is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. 12, 13, 29, 36, 114–119, 122

**Adaptive-Leaky-Integrate-and-Fire (ALIF)** A variant of the LIF neuron which has its internal threshold increased following the emission of a spike, it will then decay back to resting threshold. 74, 77, 83, 85, 87, 88, 134

**Address Event Representation (AER)** A way in which spike events in an SNN can be represented as addresses for routing, where each address corresponds to the source the spike came from. 43, 49

**Artificial Neural Network (ANN)** A typical computation structure used in machine learning comprised of a network of linearly weighted real valued inputs fed into a node with a nonlinear activation which produces a real valued number. 12, 14, 24, 25, 45, 69–74, 77, 96–98, 100, 101, 103–105, 112, 114–118, 122, 123, 128, 130, 132, 134, 136

**Back-Propagation (BP)** A method which employs the chain rule to calculate the gradients of parameters with respect to an objective function for neural networks. 24, 45, 72, 75, 76, 97

**Back-Propagation-Through-Time (BPTT)** A variant of backpropagation which is applied to recurrent neural networks. It involves storing previous state and then rolling back the gradients and errors in time. 74, 137

**Convolutional Neural Network (CNN)** A neural architecture in which repeating filters with shared weights are tiled over an input, typically visual. 36, 47, 67, 71, 97, 132, 133

**Data Tightly-Coupled Memory (DTCM)** An ARM name for a scratchpad memory holding program data. 7, 43

**Error Driven Neurogenesis (EDN)** An optimisation algorithm which uses neurogenesis and synaptic nonlinearities to immediately store information. 11–14, 28–30, 99–107, 109, 111–124, 128–130, 136, 138, 158

**Evolutionary Algorithm (EA)** An optimisation algorithm which uses processes akin to natural Darwinian evolution to create solutions to problems. 71, 134, 137

**Genetic Algorithm (GA)** An evolutionary algorithm in which the gene defines the agent. Agents are then evaluated, mutated and recombined to produce subsequent generations. 134

**Gradient Descent (GD)** An optimisation algorithm which calculates the gradient of an error with respect to parameters to move towards a position in the solution space with reduced error. 12, 24, 25, 30, 35, 36, 39, 71, 72, 75, 94, 97, 100, 107, 114, 116–118, 122, 128, 130, 132, 134, 135

**Instruction Tightly-Coupled Memory (ITCM)** An ARM name for a scratchpad memory holding instruction data. 43

**Leaky-Integrate-and-Fire (LIF)** A standard type of spiking neuron which integrates inputs over time, with some leak, and when the input crosses a threshold a spike is emitted. 72, 74, 78, 79, 134

**Long Short Term Memory (LSTM)** A neural architecture used in machine learning to imbue networks with a form of memory capable of being trained with gradient descent 36, 77, 98

**Modified NIST (MNIST)** A dataset of images containing handwritten digits 13, 29, 99, 117, 119, 120, 126–129

**Overlap (OL)** The measure of overlap between filters. It is relative to the filter size and is measured as a percentage enabling it to be invariant to the filter size. 55, 56, 58, 63–65, 67

**Radial Basis Function (RBF)** A form of activation function used in RBF networks which has activity relative to a position in space, often a Gaussian function is used. 97, 113

**Spiking Neural Network (SNN)** A version of an ANN in which activity is communicated via discrete spikes rather than numbers. 27, 29, 33, 35, 36, 49, 53, 54, 61, 66, 67, 69, 71–74, 77, 80, 134, 135, 137

**Synchronous Dynamic Random-Access Memory (SDRAM)** Memory in computers which has high density and performance. 43, 81

**Von Mises (VM)** A crescent shaped filter kernel designed to respond to curved edges. 8, 50, 51, 53–56, 58, 61, 66, 67

# Abstract

BIOLOGICALLY INSPIRED  
NEURAL COMPUTATION

Adam William Perrett

A thesis submitted to The University of Manchester  
for the degree of Doctor of Philosophy, 2022

Models of intelligence can come in many forms, from concept driven approaches such as formal mathematical reasoning to data driven approaches such as machine learning. Current state of the art approaches fall into the second category, requiring vast amounts of data to form statistical representations within the architecture of neural networks. This is in stark contrast to biological brains whose neural networks can learn with limited examples and training time. Biology originally inspired the neural network but there is still much more to be learned from nature about how to construct and train neural architectures.

By exploring techniques employed by biology it can be possible to overcome the challenges of modern machine learning algorithms. Biological brains can be trained on tasks sequentially without forgetting previously gathered information and data integration is performed online and in real-time, except for processing done during sleep. The brain also only consumes 12W of energy, which is a far cry from the energy budget of CPU and GPU implementations of neural networks.

The research described in this thesis first investigates the use of biologically inspired models of visual attention, on the SpiNNaker neuromorphic hardware, creating an event-driven low latency model of visual saliency. Following this, biologically plausible training algorithms are examined with the e-prop learning algorithm being instantiated on SpiNNaker to explore the challenges faced when learning using only locally available information. Finally, abstractions of dendritic nonlinearities are co-opted

for use in tandem with neurogenesis to create a learning architecture, which does not rely on gradient descent whilst retaining previously learned information. It is shown to reach similar levels of performance to a network trained using Adam optimisation with less presentations of data samples on a number of benchmark tasks.



# **Declaration**

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

# Copyright

- i. The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the “Copyright”) and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.
- ii. Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made **only** in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.
- iii. The ownership of certain Copyright, patents, designs, trade marks and other intellectual property (the “Intellectual Property”) and any reproductions of copyright works in the thesis, for example graphs and tables (“Reproductions”), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.
- iv. Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see <http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420>), in any relevant Thesis restriction declarations deposited in the University Library, The University Library’s regulations (see <http://www.library.manchester.ac.uk/about/regulations/>) and in The University’s policy on presentation of Theses

# Acknowledgements

I would like to thank my parents for their continual support and encouragement, without them I would not be here, quite literally. I would also like to thank Edward G. Jones for his enticing and distracting conversations both related and not to work. Thanks as well to Miguel Silva for his support and motivation throughout the PhD process and also to Markos Kynigos for always being there to talk things through over a beer.

# Chapter 1

## Introduction

### 1.1 Introduction

*From mathematical reasoning to function approximation*

In essence artificial intelligence is an attempt to capture the most relevant properties of the brain and apply them to a task. This began in earnest with the idea of formal reasoning in which it was thought possible that the processes of human thought could be mechanised and co-opted [12]. It was later with the advent of McCulloch and Pitts's neuron model and the mathematical foundation of neural networks that models of intelligence began to more closely resemble the biological brain [88], and also lend increased computational complexity to models. The universal function approximation capability of neural networks enables them, in theory, to be used to model any computable input-output mapping [25]. The growing processing power of computers has again lifted the capabilities of models enabling large data sets to be processed and summarised within a neural network. The machine learning community often sees gains in two ways, algorithm improvement or increased computing resource.

*Biology is the greatest muse*

What this thesis will argue is that by modelling the biology of the brain we can lend increased computational power to behavioural models. It was noted above that moving from formal mathematical models of intelligence to more biological inspired models lent increased computational complexity to a system. The understanding and abstraction of the biological neuron has enabled complex models of the world to be created. However, the formulation of neural network models has since relied on mathematical methods such as BP (Back-Propagation) in conjunction with GD (Gradient Descent) to imbue ANNs (Artificial Neural Networks) with a model of data. They

have proven themselves as useful tools in the creation of impressive computational constructs, capable of besting humans at tasks thought to have too many possible game state for appropriate calculation, such as chess and go. However, GD is not a panacea and requires added tricks to be able to reach high-levels of performance, such as dropout, which is similar to the inherent stochasticity of biological neuron's firing in which, with some probability, a neuron will not emit a spike after receiving supra-threshold input. An eye must always be kept towards biology for inspiration and clues about how to further improve the complexity of models of intelligence. ANNs are impressive but they are a far cry from the biological brain.

*Evolution is the greatest architect*

Evolutionary search has provided the impressive breadth and depth of the world's biological diversity. It does not succumb to the bias humans do when designing intelligent systems as it does not have specific agency; evolution is a random process in which the probability of a gene persisting is relative to its benefit in its environment. This has resulted in many complex organisms, which are well suited to its surroundings. Arguably the most impressive adaptation is the ability to adapt during one's lifetime through learning. The organ in the seat of learning is the brain and it has proven itself an invaluable tool to a creature's survival. Computational models often abstract the brain to a collection of point neurons, which linearly sum inputs and then perform some non-linear function on them. This is incredibly powerful and can approximate any input-output mapping [25], although it misses many elements of the brain such as dendritic trees, working memory, attention and one of the major unknown states humans spend a third of their life in, sleep. GD is attractive as it offers a way to guide universal approximators towards reduced error but it then becomes restricted to the predefined architecture and requires vast amounts of data to train. Gaze must move towards the solutions stumbled upon by evolution to be able to inform future directions; biology has been discovering solutions for billions of years, it would be remiss to not investigate it as a source of inspiration in any way possible. The question then becomes, what techniques does biology use and what level of abstraction is necessary to achieve similar performance?

## 1.2 Hypothesis - Research questions

This thesis looks towards biology for inspiration and explores applying those ideas to the development of neural computation. The main areas approached in this work are:

- Can visual attention be modelled in a spiking neural network architecture on neuromorphic hardware?
- What are the challenges involved in the application of spiking neural network learning algorithms and their instantiation on the SpiNNaker neuromorphic hardware?
- How can biologically inspired dendritic nonlinearities in conjunction with network growth be used to develop learning architectures?

### 1.3 Contributions

The research questions presented above are broad and possess many potential answers. Each chapter will roughly address each point respectively with the main work of this thesis approaching those challenges with the following contributions:

- Providing and investigating a fully event-based model of a biologically inspired attention model for SpiNNaker simulation running on the iCub humanoid robot, surpassing the latency achieved with previously implemented GPU models
- Creating a SpiNNaker implementation of the e-prop learning algorithm for on-line and real-time spiking neural network learning on neuromorphic hardware using locally available information
- Developing and exploring the use of neurogenesis and synaptic non-linearities for online and one-shot learning in neural networks

### 1.4 Publications

This thesis contains work which has been presented in the following publications:

**Giulia Veronica D’Angelo, Adam Perrett, Massimiliano Iacono, Stephen Furber, and Chiara Bartolozzi. 2022. Event driven bio-inspired attentive system for the iCub humanoid robot on SpiNNaker. *Neuromorphic Computing and Engineering* (April 2022).** This journal article was a joint first authorship between myself and Giulia D’Angelo. This work started from a workgroup collaboration at the Capo Caccia workshop and was continued during a 3 month interruption to visit IIT in Genoa, Italy.

My main contribution was the conversion of the attention model into a spiking neural network architecture for SpiNNaker as well as the testing and gathering of data. Giulia performed the data analysis and comparison with other data sets (as well as integration with the iCub robot, although this work was not completed and is therefore omitted). This forms the core of Chapter 3.

**Adam Perrett, Sara Summerton, Andrew Gait, and Oliver Rhodes. 2022. Online Learning in SNNs with E-Prop and Neuromorphic Hardware. In Neuro-Inspired Computational Elements Conference (NICE 2022), Association for Computing Machinery, Virtual Event, USA, 32–39.** This conference paper was a first authorship by myself with assistance and discussion from Sara Summerton and Andrew Gait on the software and hardware integration of e-prop with SpiNNaker. Oliver Rhodes was instrumental in the development of this work. He instigated the project and helped develop the initial framework that the final e-prop implementation was built upon. It was accepted for a long presentation at the NICE conference in Texas (virtually presented). The experimental work in Chapter 4 comes from this paper.

**Petruț Antoniu Bogdan, Garibaldi Pineda García, Michael Hopkins, Edward Jones, James Courtney Knight, and Adam Perrett. 2020. Learning in neural networks. In SpiNNaker, Steve Furber and Petruț Antoniu Bogdan (eds.). Now Publishers Inc, United States, 209–265.** A contribution was made to the SpiNNaker book [46] regarding learning in neural networks, with a focus on neuro-evolution, and their application to neuromorphic hardware. Some of this work is included in the background section of Chapter 4.

**Adam Perrett, Steve B. Furber and Oliver Rhodes. 2022. Error driven Synapse Augmented Neurogenesis. Frontiers in AI, (2022).** This journal article explores a non-spiking implementation of synaptic non-linearities in conjunction with a network growth algorithm [107]. It is entirely my own work with guidance and proof reading from Oliver Rhodes and Steve Furber. It forms the work of Chapter 5.

## 1.5 Thesis structure

The thesis begins in Chapter 2 with an overview of how natural systems have developed forms of intelligence and how research attempts to capture properties of intelligence. This will set the context for the rest of the thesis with Chapter 3 beginning the discussion with an event-based implementation of an attention model. Learning algorithms and their application to SNNs (Spiking Neural Networks) will be explored in

Chapter 4 with an implementation of the e-prop algorithm on SpiNNaker being investigated. Chapter 5 introduces the EDN (Error Driven Neurogenesis) algorithm which makes use of neurogenesis and synaptic nonlinearities to drive learning in a one-shot fashion without gradient calculation. Finally, Chapter 6 will summarise the work and discuss potential future directions of research.

### 1.5.1 Chapter 2 - Background

To set the context of the wider thesis the background chapter will discuss machine learning techniques from a biological perspective. The main literature review surrounding the separate research contributions will come as a component of each subsequent chapter. First, the progress of artificial intelligence and the advances leading to the development of modern neural networks will be introduced. Next, their qualities and limitations will be touched upon with an eye towards future implementations with more biological insight. This leads into neuromorphic hardware and more biologically inspired technologies.

### 1.5.2 Chapter 3 - Visual attention

Chapter 3 explores the implementation of an event-based attention system. Attention leads the gaze of the observer towards interesting items, allowing a detailed analysis only for selected regions of a scene. A robot can take advantage of the perceptual organisation of the features in the scene to guide its attention to better understand its environment. Current bottom-up attention models work with standard RGB cameras requiring a significant amount of time to detect the most salient item in a frame-based fashion. Event-driven cameras are an innovative bio-inspired technology to asynchronously detect contrast changes in the scene with a high temporal resolution and low latency.

A new neuromorphic pipeline is proposed exploiting the asynchronous output of the event-driven cameras to generate saliency maps of the scene in a more biologically inspired way. In an attempt to further decrease the latency, the neuromorphic attention model is implemented in a spiking neural network on SpiNNaker. SpiNNaker is a dedicated neuromorphic platform whose architecture is closer to the biological brain compared to traditional computers. The proposed implementation has been compared with its bio-inspired GPU counterpart, and benchmarked against ground truth human fixation maps. The system successfully detects items in the scene, producing saliency



maps comparable with the GPU implementation. The asynchronous pipeline achieves an average latency of 16ms to produce a usable saliency map compared to  $\sim 170$ ms for the GPU implementation.

### 1.5.3 Chapter 4 - E-prop on SpiNNaker

Chapter 4 discusses learning in spiking neural networks and explores their challenges. A neuromorphic implementation of the biologically inspired alternative to the back-propagation algorithm, e-prop, is explored. It offers a way to train SNNs online using only locally available information.

Online learning in neural networks has the potential to transform AI research. By enabling new information to be assimilated into existing systems, platforms can be adaptive to unseen data and can personalise performance to an individual. A common approach in providing AI to a user is to send queries to a remote cloud service which processes the information and sends back a response. Neuromorphic hardware offers an alternate solution by providing a dedicated computing platform from which neural networks can be run locally and efficiently.

### 1.5.4 Chapter 5 - Error driven neurogenesis

Capturing the learning capabilities of the brain has the potential to revolutionise artificial intelligence. Humans display an impressive ability to acquire knowledge on the fly and immediately store it in a usable format. Parametric models of learning, such as gradient descent, focus on capturing the statistical properties of a data set. Information is precipitated into a network through repeated updates of connection weights in the direction gradients dictate will lead to less error.

Chapter 5 presents the EDN algorithm which explores how neurogenesis coupled with non-linear synaptic activations enables a biologically inspired mechanism to immediately store data in a one-shot, online fashion and readily apply it to a task without the need for parameter updates. Regression (auto-mpg) test error was reduced more than 135 times faster and converged to an error around 3 times smaller compared to gradient descent using Adam (Adaptive moment estimation) optimisation. EDN also reached the same level of performance in wine cultivar classification 25 times faster than gradient descent and twice as fast when applied to MNIST (Modified NIST) and the inverted pendulum (reinforcement learning).

### 1.5.5 Chapter 6 - Conclusions

Chapter 6 summaries the work and discusses future directions. Each chapter discusses biologically inspired neural computation from different perspectives yet each would likely gain the most benefit from integration with the others. The attention system in Chapter 3 shows the potential for low latency saliency map generation, yet a learning algorithm on top would enable applicability to a broader context than is covered by the current architecture. The implementation of e-prop in Chapter 4 shows the potential for neuromorphic implementations. This algorithm could benefit from a vision based architecture, as is displayed in the attention model, or from stable learning without catastrophic forgetting, as is displayed by EDN. The algorithm discussed in Chapter 5, EDN, would definitely benefit from some fine tuning of parameters, as could be offered by GD. Some form of network conversion or distillation of the trained model may also allow continual learning and more generalised representations. This could be likened to sleep mechanisms in which knowledge acquired during the day is processed and stored in a long term format.

Biology has approached the problem of generating intelligence from multiple perspectives. The evolutionary process has determined the initial architecture learning is performed on. The learning that is performed on the initial architecture is optimised to gather particularly useful information, such as language in humans. Neurons contain many computational elements, such as ion channels, neurotransmitters and dendritic compartments, which create many complex interactions. Connections are made and broken between neurons via structural plasticity whilst synaptic plasticity alters the strength of connections. There are many interwoven elements that make up the biological brain, the deeper the understanding of each part, the better the systems that can be create. But it will not be one element in isolation, which creates high-level intelligence, it will be an amalgamation of many into a cohesive whole.

# Chapter 2

## Background

### 2.1 Understanding biological intelligence

#### *Thinking about the brain*

One of the great mysteries of our time is understanding how the brain functions as a whole. Holding an as yet unsurpassed capability to capture information, manipulate it and apply it to a range of tasks, the human mind presents one of the great frontiers of science. The Ancient Egyptians first thought the heart to be the seat of intelligence but with developments through the ages we have come to understand with greater clarity the role the brain plays in behaviour. Starting with Luigi Galvani's discovery of the role of electricity in nerves in the late 18th century, the importance of the brain in controlling bodily functions was becoming clear. Following the invention of the microscope by Zacharias Janssen and the development of a staining procedure by Camillo Golgi, Santiago Ramón y Cajal would go on discover the neuron as the fundamental unit of the brain which would win him the Nobel Prize in 1908, see Figure 2.1 for a diagram of a typical biological neuron as we know it today.

Alan Hodgkin and Andrew Huxley's experiments on the giant squid axon provided the first use of a mathematical model to represent neural systems [57]. They discovered how ion-channels embedded into the cell membrane govern the progression of the neuron membrane voltage throughout an action potential. Their work culminated with the creation of the Hodgkin-Huxley conductance based circuit model which accurately describes the excitation and inhibition of the cells [56], a model which remains to this day one of the most biologically faithful models of neuron behaviour.

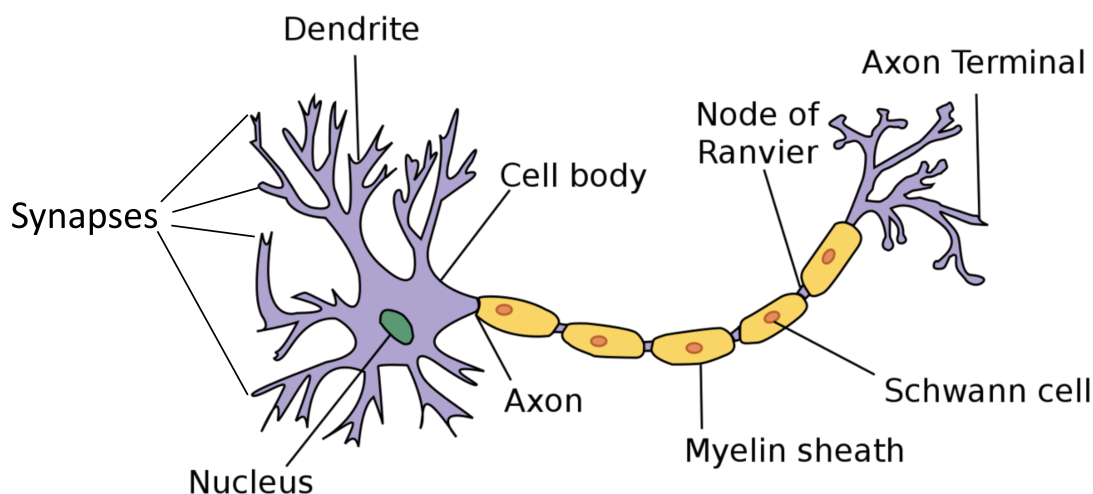


Figure 2.1: A typical biological neuron: synapses at the end of dendrites receive chemical or electrical input from other neurons which is sent towards the soma (cell body), charge collects at the soma until a threshold is crossed at which a spike is emitted down the axon, the myelin sheath is produced by the Schwann cells and acts to insulate the axon improving transmission quality and speed and the nodes of Ranvier act as signal boosters/repeaters along the axon improving transmission quality over distance. This image by Dhp1080 is licensed under CC BY-SA and has been further annotated for this work.

### *The known and unknown unknowns of neurons*

As collective understanding of neurons, and the brain as a whole, progresses it is further grasped that neurons, and brain areas, come in many forms and serve a diverse range of purposes. It is only in recent years that we are beginning to appreciate the role dendrites, the branching arms of neurons, play in brain dynamics. No longer assumed to be passive cables for charge there is mounting evidence surrounding their role in cognition and memory formation [47, 67, 136]. For every piece of information we gain we realise another hole in our understanding. It is this inescapable chasm of knowledge we must cross before we can harness the computational power of the brain with enough clarity to ever seriously attempt creating human-level intelligence. Mathematics is a language to define an understanding but it struggles to facilitate innovation in domains such as artificial intelligence. It allowed the formalisation of neurons as computational units but it took experimentation and investigation to first find out what properties to be modelled. Biology has been guided by one of the greatest innovators we know of, evolution.

Machine learning has taken to using a high level abstraction of biological neurons; they are point neurons with linearly weighted inputs being summed and passed through

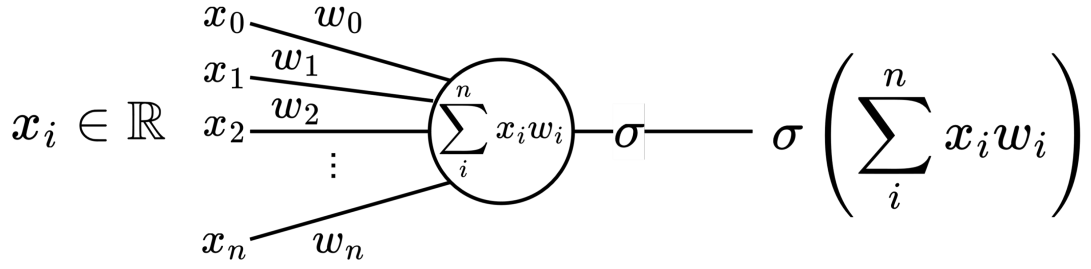


Figure 2.2: A typical artificial neuron: real-valued inputs,  $x_i$ , (including a bias) are passed along weighted synapses,  $w_i$ , and summed at the neuron. The weighted sum is passed through some nonlinear activation,  $\sigma$ , to produce the neuron output.

a nonlinear activation function to produce the output of a neuron (Figure 2.2). The spiking nature of biological neurons is instead compressed into a rate-based model where a real-valued number is the output of a neuron and represents an instantaneous firing rate. This has proved very effective but the abstraction loses important subtleties, such as the temporal element of spiking neurons and the nonlinearities of dendrites. SNNs (Spiking Neural Networks) are a lower level abstraction and, therefore, more faithful to the biological neurons, see Figure 2.3. Their membrane voltage,  $v_{mem}$ , possesses a time constant which imbues the spiking neuron with a temporal dynamic, as past states can influence future states. Different data encoding schemes can also be used in which the timing of spikes can communicate information, such as the relative timing of rank-order coding [138] and precise-time coding which encodes information in the arrival time of spikes [151].

A spiking neuron's potential benefits over their non-spiking counterparts can be summarised as follows:

- sparse activity (neuron spiking is often be zero, reducing communication)
- temporal element to processing (internal states integrate over time)
- energy efficiency (sparse and binary activity saves energy)
- information density (with non rate-based codes single spikes can be information dense)
- event-driven (offers high temporal resolution and energy efficiency, discussed in Section 2.3)
- low latency (event-driven input allows rapid processing)

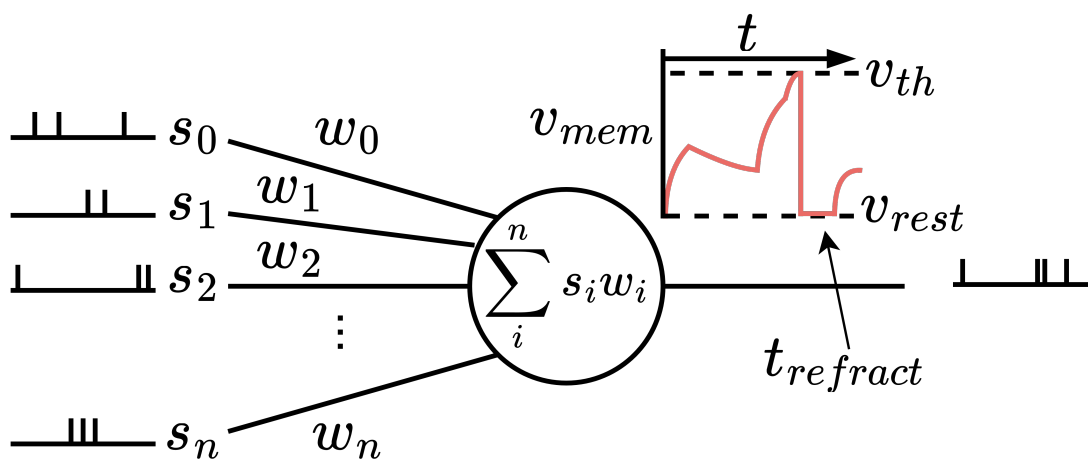


Figure 2.3: A typical artificial spiking neuron: inputs are discrete spikes,  $s_i$ , in time (a binary value indicating a spike at time  $t$  or not), spikes pass along weighted synapses,  $w_i$ , and activation collects at the neuron. The contribution of the weighted inputs is added to the neuron's membrane voltage (its internal state,  $v_{mem}$ ); the membrane voltage will decay with time but if enough input is received and it crosses its threshold,  $v_{th}$ , the neuron will release a spike and  $v_{mem}$  will be reset to the resting voltage  $v_{rest}$ ; following a spike there is a refractory period,  $t_{refract}$ , in which the neuron cannot spike again.

These benefits are attractive for a number of applications, such as mobile devices with limited battery size and self driving cars to reduce reaction time. However, the discontinuity of the SNN neuron's spiking activation proves a challenge to gradient descent approaches because the derivative is not defined at time of spike and zero otherwise. Modern approaches tend to use a derivative of the membrane potential rather than spiking activity, this will be further discussed in Chapter 4.

## 2.2 Learning how to learn

### *Gradient descent ascends to greatness*

At present, the main approach to imbuing a model with intelligence is via GD (Gradient Descent), see Figure 2.4. This offers a mathematically grounded way to update the parameters of a model in a fashion that reduces an objective error. This process of neural network training begins first with defining a computational structure. Following this, the derivatives of the model parameters with respect to the error are calculated and the parameters are updated in the direction the gradient suggests will minimise the error. However, there are a number of challenges with gradient descent techniques:

- catastrophic forgetting (learning can overwrite previously acquired information)
- requires differentiability (no derivatives, no gradients, no learning)
- only smooth transitions (alters parameters, not creation of new ones or architectural changes)
- vanishing gradients (gradients effectively become zero halting learning)
- exploding gradients (gradients accumulate making weight updates large and causing unstable learning or numerical overflow)
- saddle points (points within the solution space without gradients)
- hyperparameter sensitivity (slight alterations to learning rate, batch size etc can significantly effect learning)
- sensitive to initial conditions (the parameters the model begins with shapes the converged representation)

There are proposed solutions to these problems, such as batch learning to mitigate catastrophic forgetting, gradient clipping to limit exploding gradients, skip connections to limit vanishing gradients and learning with momentum to avoid saddle points. The Adam (Adaptive moment estimation) optimizer, used in Chapter 5, incorporates previous gradient calculations to create a per parameter moment for future weight updates that reduces the likelihood of being caught in saddle points and has been shown to speed up learning [68]. However, the general learning structure remains the same with a predefined architecture gradually acquiring information during training, which is incorporated in the model parameters. With representative data this leads to a statistical summary of the data within the model. This is a different paradigm to biological brains, which process information in real time and explore their predictions actively rather than passively. This is termed active sensing and describes how information is acquired through action and is often controlled by attention mechanisms [123]. In real time scenarios this could enable efficient data acquisition [97] and hypothesis testing, as opposed to passive streams of information being captured in a statistical model, although machine learning approaches would need to account for this bias in sampling as the data could no longer be statistically representative.

As progress in the field of machine learning advances, the complexity of the trainable architectures increases. We have transitioned from single layer networks to multi-layer networks to CNNs (Convolutional Neural Networks) and recently to transformers [33, 143], LSTMs (Long Short Term Memorys) have also been an important structural addition to neural networks with regards to temporal processing [52, 110, 121, 135]. Each transition highlights the importance of the topology of the model in controlling the way in which information can be processed, and stored via learning, within the network. GD offers a great way to update the parameters of a model as the structure has already been defined. GD allows movement around a continuous solution space. However, when training relies on the smooth updates of network parameters, this creates a bottleneck of potential updates. There is no way gradient descent could update a feedforward network to create a CNN or a transformer as these are non-smooth architectural transitions. In theory a feedforward network could create the same mapping as a CNN but it would rely on the data being representative enough that the mapping is translation invariant. GD also struggles with an SNN neuron's nonlinearity (as the derivative is undefined around spiking because of its discontinuity), although there are a number of proposed solutions to this. The difficulties in training SNNs will be further expanded in Chapter 4.



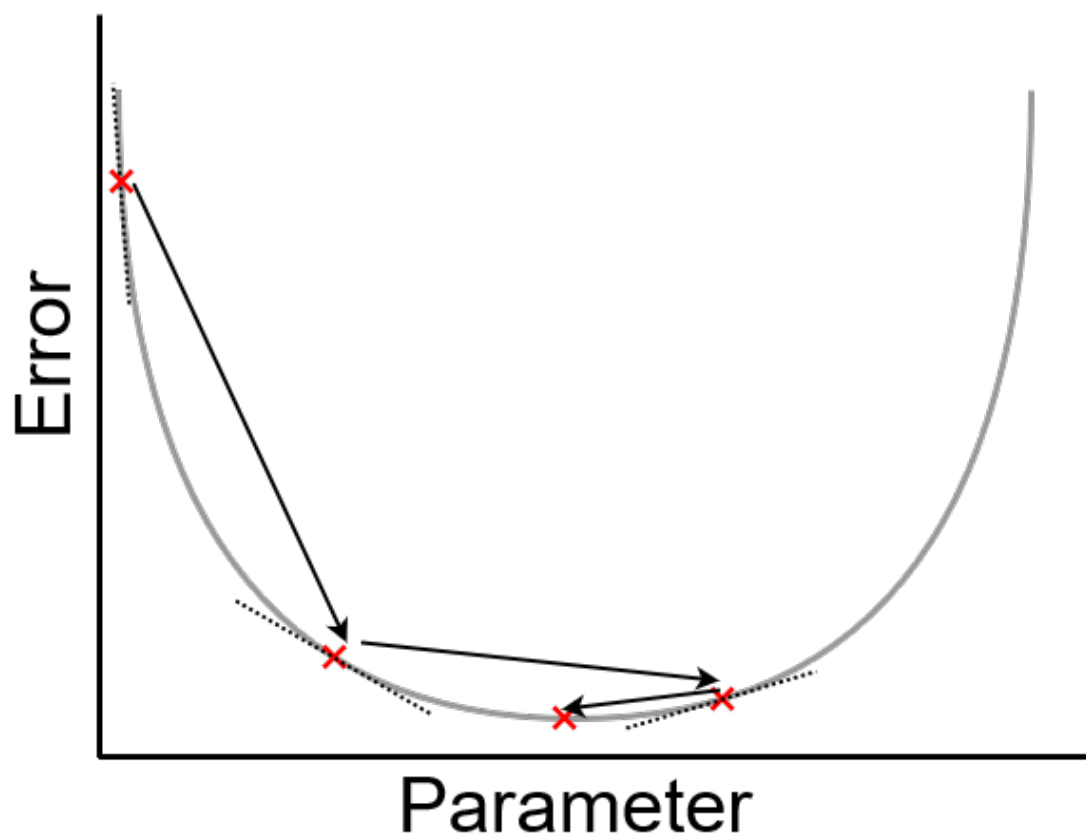


Figure 2.4: An example of a parameter being updated via gradient descent. The gradient of the error with respect to a parameter is calculated at a point and the parameter (red cross) is updated in the direction that the gradient (black dotted line) points towards reducing error. The parameter is continuously updated this way, possibly even overshooting the optimal value, until some termination criteria is met.

*Evolution is the old revolution*

One potential solution to learning without smooth transitions is the application of evolution to network generation. Evolution does not require a gradient to be able to iterate and instead relies on a combination of network evaluation, combination and mutation. Using similar principles as Darwinian evolution, that the fittest have a higher probability of passing on their genes and random mutation is a driver of genetic change, evolutionary algorithms can solve tasks without the need for gradient calculation. There are incremental methods such as NEAT [130] in which the complexity of the agents is increased with successive generations, although direct encoding methods like this often suffer from an inability to scale. Indirect encoding methods, in which a single gene can encode multiple agent parameters, such as hyperNEAT [131], enable larger networks with complex structures to be formed. The genes allow multiple different neuron types with a range of activation functions, such as linear, sigmoid, Gaussian and sinusoidal, to be combined in a network. The network is then queried to determine the connectivity of a secondary network enabling the creation of complex structured connectivity. By developing networks in such a way, architectural design does not require expert knowledge because the learning algorithm is able to adapt. This could create feedforward or convolutional networks as the task demands. However, generally evolutionary algorithms struggle when it comes to the fine tuning of network parameters which is where gradient based methods excel. A combination of gradient based techniques and evolution, often called learning-to-learn [139], which can have evolution controlling the initialisation and hyperparameters of the inner-loop gradient optimisation, has been shown to combine the strengths of both approaches and allow generalised initialisation and improved fine-tuning of parameters [8, 14].

*Online is off-brand*

When creating intelligent systems and installing them in robotic scenarios there are a number of challenges to be faced. The first challenge is the generation of the model. Current popular development techniques require either the presentation of vast amounts of data to the learning model, such as in gradient descent methods, or the evaluation of many candidate models, such as evolutionary algorithms. The commonality between these approaches is the need for offline network training. This puts a limit on robotic applications as it requires the learned model to capture everything the robot could encounter in its environment. If there was to be some environmental change or damage to a component there would have had to have been explicit training about such a scenario,

as well as the ability to detect it, for the robot to be able to adapt its behaviour. Methods such as data augmentation [125] allow small examples to be extrapolated to many training examples but often they are translational augmentations or an added variance to data which cannot capture the full range of possible future situations.

#### *Continuous models continually forget*

A common problem with the application of continuous learning is that of *catastrophic forgetting* [43]. Parametric learning models, such as GD, require that updates are passed into existing network parameters. This allows the network to gradually move towards a position in the solution space with an estimated reduction in error. However, if care is not taken, this can cause previously learned information to be overwritten. In standard training regimes batch updates or low learning rates are used to smooth weight updates over multiple examples which are presented in a random order; this gradually moves the parameters towards a position in the solution space which reduces the error across a number of examples, ideally the whole data set. If you were to train a classification model on each class sequentially then each subsequent class's weight updates will be put into the same parameters which stored the previous class's updates overwriting the previously learned information [69]. This is termed *catastrophic forgetting*, which restricts neural networks to narrow intelligence as they cannot learn tasks sequentially without sacrificing performance on previous tasks [152]. In a robotic scenario this could prove fatal to operation. An agent trained to interact with an environment could forget large amounts of its training if learning was not halted or controlled in some way.

Biology has managed to tackle the problem of continuous learning. Instead of memories being lost when new things are learnt, they follow very reliable forgetting curves [36, 98]. Memories are formed and processed during the day to be later consolidated during sleep [132]. It is processes like this of memory formation and subsequent consolidation that is missed from current machine learning approaches. Generally, gradient descent approaches skip the step of memory formation and jump straight to the creation of a consolidated model. This works well for scenarios in which a large and representative sample of the data has been collected for training but struggles in more dynamic and complex environments, as would be found in real world robotics applications. This suggests a missing step between the maths and the biology. Mathematical approaches want a predefined structure which rigorous operations can be applied to, biological approaches aim for a continuously changing and adapting model with quick

acquisition of experience. If we wish to iterate machine learning more towards human-level intelligence then we need to investigate these biological primitives further.

*Attention reduces what needs to be attended to*

An important primitive that is missed from many machine learning models is that of attention. It offers an agent the ability to reduce the dimensionality of the problem and allows computational resource to be focused on the more salient parts of an environment [93, 97]. At a higher level it can also allow an agent to be attentive to different goals depending on the situation, which is an executive form of attention. Being able to better allocate processing power can mean increased processing of important information and reduced energy expenditure on irrelevant information. In biology this is important to reduce energy expenditure, which would mean increased need for food consumption, and it is especially relevant in machine learning given the energy cost to train models [105].

Current research has found great computational benefits in the use of transformers which are a form of attention mechanisms. First they were applied to natural language processing [143] but have also more recently shown advances in machine vision [33]. They dispense with recurrent connections and convolutions, which are typically used to facilitate processing of related elements, and display state of the art performance whilst also being shown to require significantly less time to train, which means energy and time savings. Transformers are an important step towards instantiating forms of attention in intelligent systems, although higher level executive attention is still yet to be displayed.

## 2.3 The medium matters

Machine learning models are typically run on CPUs, GPUs or TPUs. Their focus is generally on the calculation of matrix multiplications as neural network activations and connections are often expressed as vectors and matrices. Their specialised hardware allows many computations to be done per second but it is far from how a biological brain behaves. There is significantly sparser connectivity in the human brain compared to the all-to-all connections you find in typical machine learning architectures. The spiking nature of biological neurons also means activity is sparse in time too. This construction creates inefficiencies in matrix operations as they do not account for the spatial and temporal sparsity. This leads to more calculations performed per operation than is necessary as many elements are zero and therefore have no effect on the result.

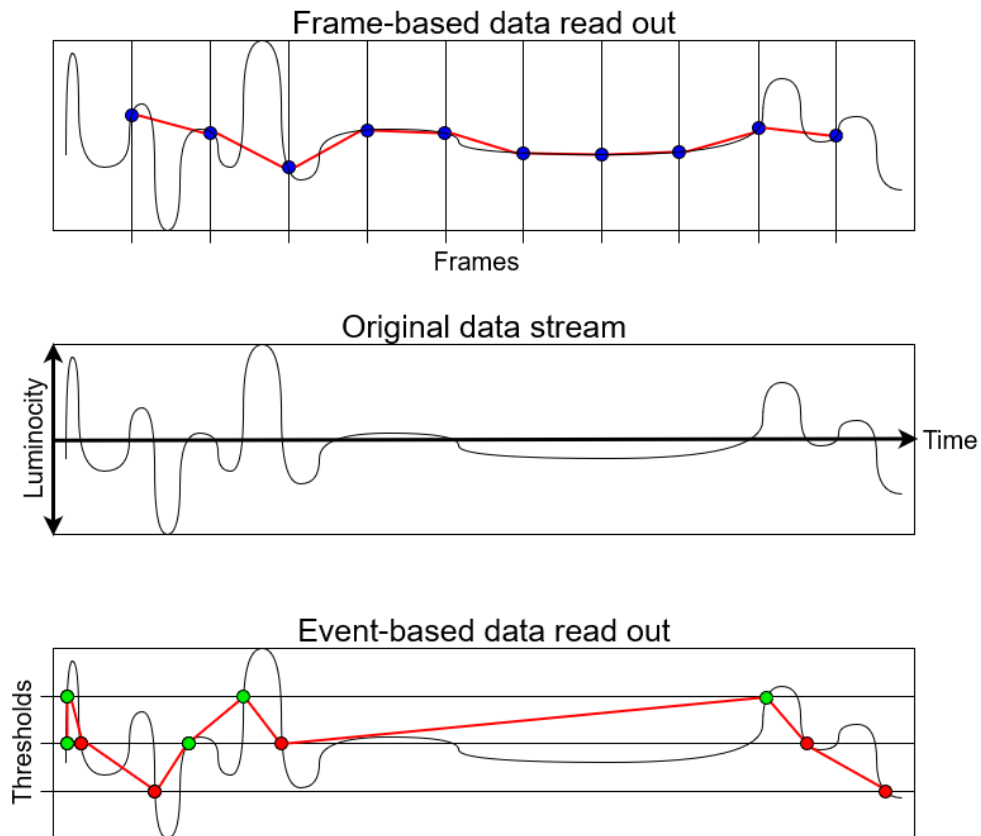


Figure 2.5: The middle plot shows an input data stream, this can be imagined as a single photo sensitive element of a charge-coupled device (CCD). The top plot shows an example of how a frame-based sensor would collect information from the sensor; with regularity in time the current value of the sensor is recorded. The bottom plot shows how an event-based sensor records data; the current value is set following a threshold crossing and when the value next moves a threshold distance away from the stored value an event is triggered. An ON event (green) signifies the value is a threshold above the stored value and an OFF event (red) signifies it went a threshold below the stored value. The red lines connecting dots gives a rough example of how the signal could be reconstructed from the stored values. Both frame-based and event-based sensors recorded 10 data points yet the event-based sensor retains more of the original signal, owing to its asynchronous sensing, and filters the relatively quiescent parts.

*Sensing a change*

There is increased attention given to event-based sensors. These are sensors which represent their inputs as an asynchronous stream of discrete, typically ON and OFF, events rather than the constant stream of data seen in traditional sensors, such as RGB cameras. Figure 2.5 shows a comparative example of frame-based and event-based sensors receiving an input stream. Although event-based sensors only store ON and OFF events they can be interpolated to achieve a representative reconstruction of the original signal. This is in part a result of the asynchronous sensor, which has a significantly lower latency than frame-based cameras enabling fast contrast changes to be captured. Frame-based sensors save the continuous sensor reading for each frame but the relatively high latency means quick fluctuations in the sensor are not captured. During periods when the sensor value does not change the frame-based sensor continues to record values, which is less computationally efficient compared to the event-based sensor, which would not produce events.

By also providing an input stream which is encoded as discrete spikes, as is done with the event-based sensor, an entire data processing pipeline can be made which is biologically inspired, where inputs encoded as events are passed through a sparsely connected and active network to produce an output. The event-based nature of the computation could reduce the number of timesteps which are needed for inference leading to increased response time. A movement needs to be made from constant computation to event-based computation where updates are only performed in the presence of events. Sparse activity and connections could significantly reduce the cost of inference. This is of particular importance when you consider the energy budget of large deep learning models [105]. The amount of computation used to train deep learning models is doubling approximately every 3.4 months [5], if this trend continues the energy required to train these models will become staggering. The estimated energy to train GPT-3 was 1287MWh [106], with the average domestic electricity consumption per year of an English household at 4MWh [4]. This means the energy used to train GPT-3 could have provided a year of electricity to around 320 English households.

*Brain-like computers*

Neuromorphic computers offer a solution to biologically plausible processing by structuring their computational framework to better match biology. The word neuromorphic means ‘brain like’ and neuromorphic systems aim to create computers that better match the operation and power efficiency of the neuronal architectures, rather than treating them as elements in matrices to perform operations on. There are a number of potential

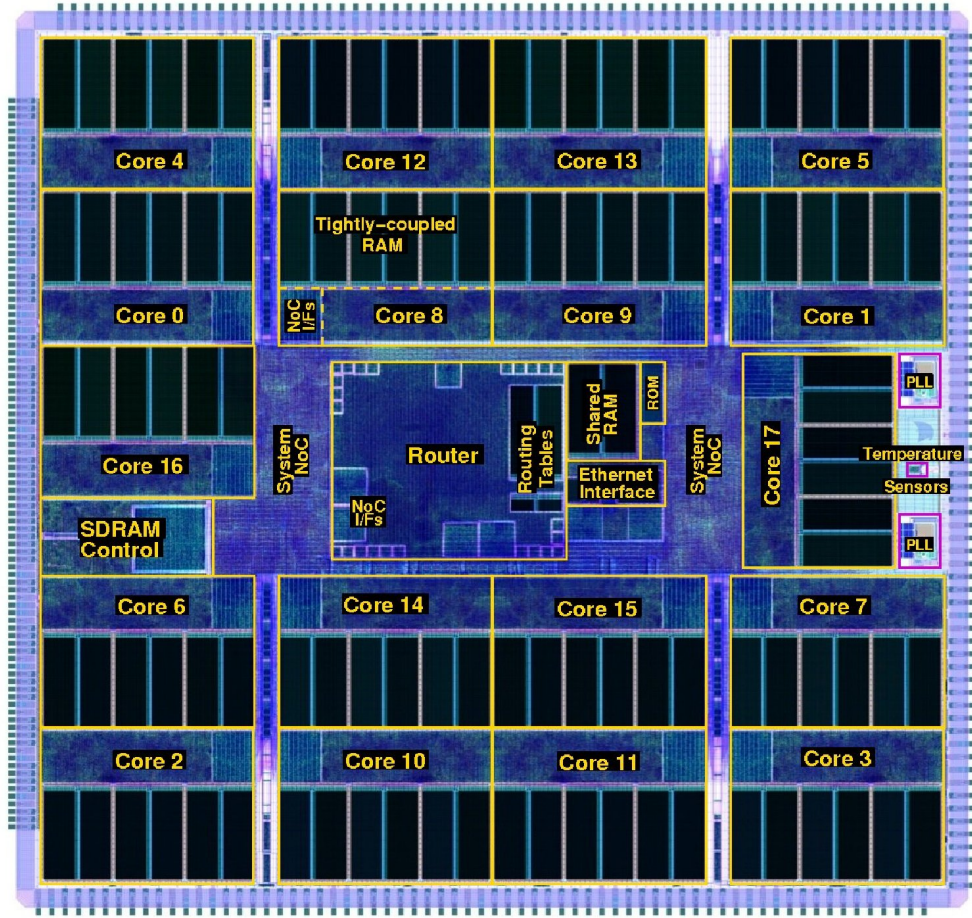


Figure 2.6: A labelled SpiNNaker chip. Not shown is the DTCM (Data Tightly-Coupled Memory) which is physically mounted on top of the cores and router and stitch-bonded to it.

solutions from analogue chips which have passive electrical components whose combined behaviour emulates neurons [22, 122], to digital chips with configurable cores capable of simulating a range of learning rules and neuron types [28, 46].

The neuromorphic hardware used in this work is SpiNNaker [45], a highly parallel digital computing platform. A single chip has 18 interconnected ARM968 processors with 64kBytes of DTCM for data memory and 32kBytes of ITCM (Instruction Tightly-Coupled Memory) for instruction memory locally at each core. There is also 128MBytes of SDRAM (Synchronous Dynamic Random-Access Memory) shared across all cores on a single chip, see Figure 2.6 for a chip schematic. Using a Globally Asynchronous Locally Synchronous approach, each chip is connected to 6 neighbouring chips. Following the AER (Address Event Representation), communication across the SpiNNaker machine is in the form of small packets containing either 32 or 64 bits



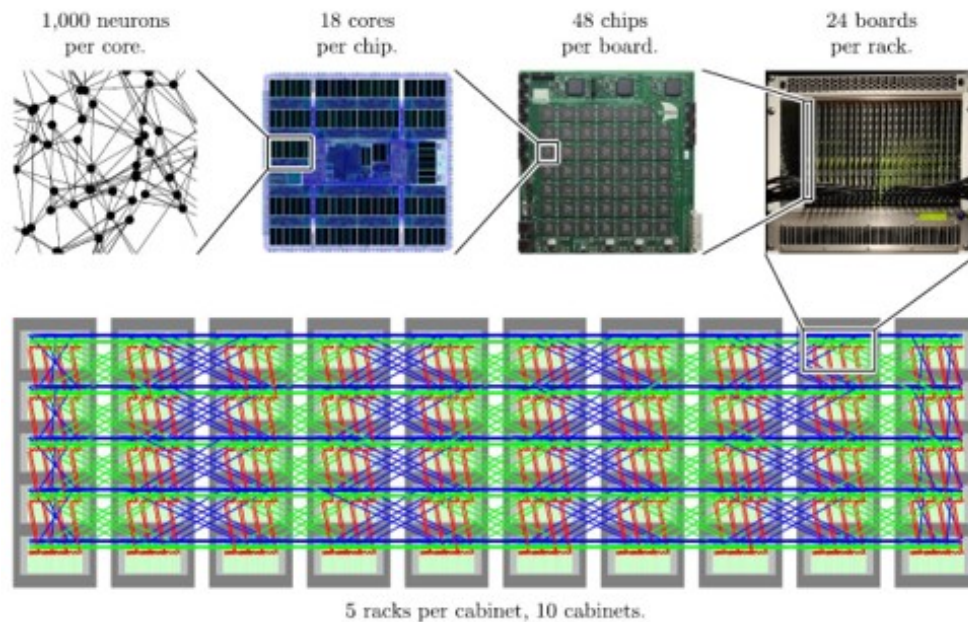


Figure 2.7: Each core is capable of simulating up to 1000 neurons, 18 cores compose a SpiNNaker chip, 48 chips are on a single SpiNN5 board, 24 boards are in a rack with 5 racks per cabinet for a total of 10 cabinets. In total this means the over 1 million core machine could in theory simulate around 1 billion neurons.

of data. Typically, the only information in a packet is the information about the source location which is used for routing and eventually neuron updates. This is similar to the biological brain in which spikes/events do not communicate information apart from knowledge the source node emitted a spike/event which is then broadcast to all connected nodes. 48 interconnected chips compose a SpiNNaker board which can also be interconnected enabling many more cores to operate in parallel, as on the million core SpiNNaker machine, see Figure 2.7 for a broken down version of the scale of the million core machine. In theory this could simulate up to 1 billion neurons which is equivalent to around 10 mouse brains or 1% of a human brain.

SpiNNaker presents an ideal platform for biologically plausible algorithms. Its communication fabric possesses many of the same qualities as the biological brain, namely: information is communicated through packets, data is mainly available locally and operation is in real-time. It is a digital and programmable computational resource, which lends a high level of configurability to simulations. Its real-time operation also makes it ideal for robotics applications, though, when looking at the current state-of-the-art machine learning approaches towards training, which involve many thousands of presentations of inputs to a model, running at real-time can be seen as



a disadvantage. It is for this reason that different paradigms of learning must be explored, the biological brain does not require the many thousands of presentations of inputs to grasp certain concepts and only operates in real-time.

## 2.4 Summary

The brain is one of the most complex structures in the known universe and understanding it will likely be a never ending task. Humans have abstracted away important elements of brain cognition, most notably neural networks, but there is still a large chasm between the ANNs (Artificial Neural Networks) used in machine learning and the biological neurons present in the brain, and likely an even larger difference between the algorithms used to train them. Geoffrey Hinton, one of the pioneers of BP (Back-Propagation), is quoted as saying he is "deeply suspicious" of BP and "My view is throw it all away and start again" [80]. It is the dependence on labelled data that Hinton is justifying the need for abandoning BP for, something that is not required for much learning in the brain. An eye must be kept towards biology and we must be cautious of becoming over reliant on existing mathematical tools to develop neural architectures.

The following chapters will explore a number of biologically inspired mechanisms of neural processing. First, an entirely event based implementation of a visual attention system on the SpiNNaker neuromorphic hardware will be presented. Following this an implementation of the biologically plausible alternative to the BP algorithm, e-prop, will be instantiated on SpiNNaker to explore its effectiveness and limitations of learning online on neuromorphics with only locally available information. Next, a model of dendritic nonlinearities will be integrated with a neurogenesis mechanism to create a gradient free learning architecture in a biologically inspired way. Finally, the work will be concluded and possible future work discussed.

# Chapter 3

## Visual attention

### 3.1 Introduction

#### *Overt and covert*

Attention covers a range of definitions which have different interpretations depending on the context. One of the most fundamental ways to divide classes of attention mechanisms is into overt and covert. Overt is where focus is actively brought towards the most salient stimuli, this could be foveating the eyes towards a position in the visual field. This is of particular importance in biological contexts as the centre of the fovea has the highest density of receptors and therefore higher resolution processing is done on objects centred in the visual field. In robotic contexts this is less important as the visual field generally has a uniform packing of photoreceptors. Covert attention is where focus is brought toward a part of the visual field without the moving of the eyes, such as with saliency map generation (to be discussed later). This is generally how attention works in robotics and machine learning contexts as often the inputs are fixed and it is selectivity to them that is modified rather than alterations to their resolution or a movement of the visual field. Visual transformers [33, 143] always receive all inputs and it is only the weighted vector representation of them that is processed rather than any down or up sampling that is taking place.

#### *Attentive in and out*

A key aspect of attention is to guide processing in an effort to better allocate computational resource. This processes can be exogenous, produced outside, and driven by low level stimuli, such as a bright object moving fast across your visual field; this is sometimes referred to as bottom up attention. The alternative is endogenous, produced inside, and driven by high level features derived from goals and desires; this is

sometimes called top down attention. Both are important to survival, you want to be able to react quickly to stimuli without it needing to be relevant to the current goal and you want to be able to direct attention providing more computational resource towards stimuli related to complex goals. There is also a higher level of attention which is referred to as executive attention, this is a mechanism which allows switching between goals and moving focus between different tasks and subtasks.

All modalities of attention are important for efficient processing and action selection. This chapter explores the application of biologically motivated concepts of attention and their application to a robotic context. It falls into the category of covert attention as the visual field is not moved during the experimentation; the processing outputs a saliency map of the environment. It is driven by high level features, putting it somewhere between exogenous and endogenous attention. Exploring how biological primitives drive attention in the absence of learning mechanisms allows unbiased inferences to be made about how features can drive attention. The design of future learned attention architectures can then be influenced by the conclusions drawn, analogous to how CNNs (Convolutional Neural Networks) are influenced by the structures in the visual system but their shared weights for learning are not biologically plausible [44, 60].

## 3.2 Background

Visual attention guides the perception of the environment [83]. It is a mechanism that selects relevant parts of the scene to sequentially allocate the limited available computational resources to smaller regions of the field of view. In the animal world, this is coupled with eye movements, aimed to sequentially centre the selected region within the highest resolution region of the retina [141]. The detailed analysis only of salient regions of the visual field can dramatically reduce the computational load of processing the full visual field at once. In a similar manner, a robot working in real-time can exploit visual attention advantageously to optimise the use of computational resources.

### *Machine attention*

The motivation of this work is to develop an attention system, which can produce a reduction in computational loads for autonomous systems. Robots, such as the humanoid robot iCub [89], would need to generate a fast and precise response autonomously to interact with the environment when reacting to external stimuli. Recent studies in

computer vision have exploited the concept of attention for different tasks: classifying MNIST handwritten numbers only on regions of interest (ROIs) of the visual field with the 1.07% error [93] (making the visual processing invariant to input dimension), fixation prediction adding audio cues [90], visual search [91], and object recognition, where it has been demonstrated that attentional selection (based on saliency) increases the number of objects that can be identified in a visual scene [119].

Attention has attracted interest since the first psychological experiments where Yarbus et al. [150] were recording the fixation points of subjects examining different pictures. Since then, attention has been modelled in order to understand its underlying neural implementation, and to equip artificial agents with similar capability to obtain a reasonable perception of the scene [64]. Biological attention is a complex mechanism that results from the interplay of a bottom-up process that is driven by the physical characteristic of the stimuli and top-down effects that depends on priors and goals [149]. Diverse studies tried to model the bottom-up components of attention. Some proposed the use of the saliency map formalism [37, 109, 140]. A saliency map is the representation of visual saliency in a scene, where each item appears to be interesting (salient) based on the observer's visual exploration [70]. Specifically, selective attention extracts features from the environment and explains the situation as fast as possible filtering what is not necessary to understand the scene [86].

#### *Model attention models*

The widely used feature-based saliency model [64] extracts in parallel multiple different visual features and finds regions of high contrast within each feature channel. Their contribution defines the saliency of each point in the field of view. The weight of each feature map can be modulated to model the effect of top-down mechanisms competing with each other for the representation of the scene. This model was then augmented [144], by integrating principles of perceptual grouping of individual components that reflect "Gestalt laws" as proximity, common fate, good continuity and closure [72]. These principles give perceptual saliency to regions of the visual field that can be perceived as "proto-objects" [71, 134].

A proto-object describes regions of the visual field that may correspond to real objects in the physical world, referring to the human ability to organise part of the retina stimuli into structures [102]. The work of Russell et al. [118] improved [144] by creating a filter capable of detecting partial contours. Recent studies added other sources of information to the proto-object model such as motion [94], depth [58] and texture [142]. Further, a new line of research has started to develop these types of

models using event-driven cameras as input. In these cameras, the contrast change in the scene is output asynchronously, with high temporal resolution, low latency, and most importantly, a reduced data rate. For a real-time application in a robotics scenario, this leads to a faster response given the low processing required [49, 112].

*An event for the senses*

Adams et. al. [3] exploited the AER (Address Event Representation) and the neuromorphic platform SpiNNaker to allow the humanoid robot iCub [89] to perform real-world tasks fixating attention upon a selected stimulus. Rea et. al. [111] exploited visual attention for a bio-inspired pipeline using event-driven cameras (ATIS cameras) [108] mounted on iCub, the neuromorphic robot [7]. This implementation [111] exploits the low latency of the event-based cameras, further increasing the speed of the response towards online attention, but does not include the proto-object concept, that was later included by modifying a frame-based proto-object model [118] in a way that is suitable for event-based cameras [63].

The implementation proposed by Iacono et al. [63] adapts the proto-object model based on RGB cameras to event-driven input, using the contrast feature maps naturally encoded by event-driven cameras. However that work did not fully exploit the advantages given by the event-based sensor. In fact events were accumulated over time generating frames that were then processed using a GPU. In an attempt to decrease latency and computational cost the model proposed in [63] was implemented on the SpiNNaker neuromorphic computing platform [46] that is able to properly exploit the asynchronous output of the event-based cameras. SpiNNaker is a dedicated neuromorphic computational device which provides a digital platform to model SNNs (Spiking Neural Networks) at large scale in real time. Using an asynchronous and highly parallel architecture, large numbers of small data packets can be processed, which in most applications represent spikes being sent between biological neurons. This provides an ideal computational tool for event based processing.

The neuromorphic platform supports asynchronous spiking models that propagate events from the sensors in the network. Such models yield minimum processing latency, most of which depends on the propagation across layers and on the accumulation of sufficient information [20]. The contribution of this work is the validation of the model implemented in a biologically plausible way on SpiNNaker (SNNevProto) through a direct comparison with the event-driven proto-object (PyTevProto) (i.e. its counterpart implemented on GPU using PyTorch). The two models are compared using the data set from [63] (SalMapIROS) and are both benchmarked against human

fixation maps [76]. Analysis of the trade off between accuracy, number of neurons, computational cost and latency is performed.

### 3.3 Event-based SNN proto-object model of saliency

#### *The model before the event*

This work takes inspiration from the bio-inspired saliency-based proto-object model for frame-based cameras initially proposed by Russell et al. [118] and its event-camera adaptation [63]. The former is composed of three channels: intensity, colour opponency and orientation, competing with each other to represent the scene. Its core is composed of four layers: Center Surround Pyramids (CSP), Edge Pyramids, Border Ownership and the Grouping Pyramid (see Figure 3.1). The term pyramid indicates that it exists at multiple spatial scales.

The CSP layer convolves the input image with a difference of Gaussians kernel to detect regions in the scene with either positive or negative contrast, emulating the Center Surround (or Bipolar) cells present in the retina [18, 61]. The difference of Gaussians kernel has a strong weight in the middle and a weaker inverse sign weight surrounding it to respond to points of high contrast. In parallel, the system convolves the RGB image with Gabor filters, emulating the edge extraction done by the Primary Visual Cortex [74], creating the Edge Pyramids. The Border Ownership and Grouping Pyramid implement the ‘Gestalt laws’ of continuity and figure-ground segmentation, mimicking the neurons in the Secondary Visual Cortex area, which are mostly selective to edges [155]. This groups previous inputs to create a response to contours, the Border Ownership layer, and then closed contours, the Grouping layer. All the computation steps are performed at several scales to obtain object size invariance/tolerance.

In the Border Ownership layer the output of the CSP is convolved with curved VM filters (see Figure 3.2). The convolution with four different orientations of the filter detects partial contours of objects. All filters in the same location are connected via inhibitory connections to each other creating local competition for the dominant orientation. The output is then pooled by the Grouping Pyramid which combines oppositely rotated contours oriented to the same centre forming a partially closed contour. Closed contour activity is captured by the proto-object neurons whose combined activity creates the saliency map. In [63] this model was adapted to run using the output of event-driven cameras. Here it is taken a step further, implementing the model with spiking neurons on neuromorphic hardware.

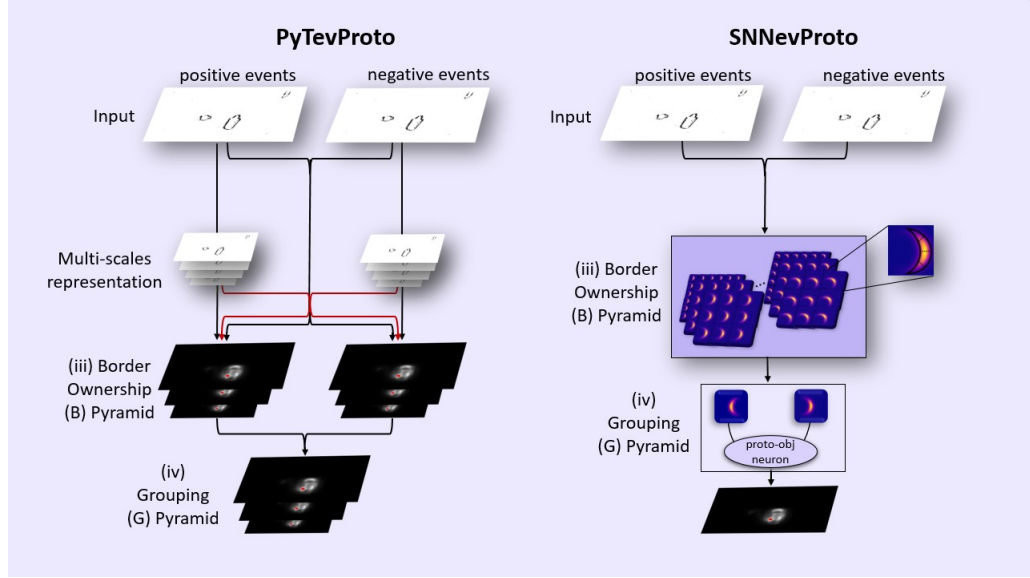


Figure 3.1: An overview of the model architectures for the PyTevProto (on the left) and the SNNevProto (on the right). The events are split based on the polarity and fed into the two models as input. The event-based model generates different scales by subsampling the "event-frame" and creating a pyramid. The resulting scaled "event-frames" are convolved with VM (Von Mises) filters at 4 different orientations (Border Ownership Pyramid) and grouped at the Grouping Layer which processes the input with the two layers of Border Ownership and Grouping Pyramids. The red lines are inhibitory signals. The spike-based implementation processes the events asynchronously exploiting layers of VM shaped neuron receptive fields at different scales and rotations. The Proto-Object Neurons (Grouping Pyramid Layer) integrate the response connecting VM filters with opposite rotation and pool the response from different scales. The outcome of both models is the saliency map.

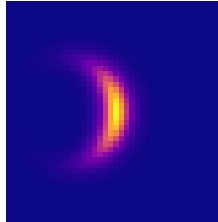


Figure 3.2: Representation of the VM filter described in Eq. 3.1 at  $0^\circ$

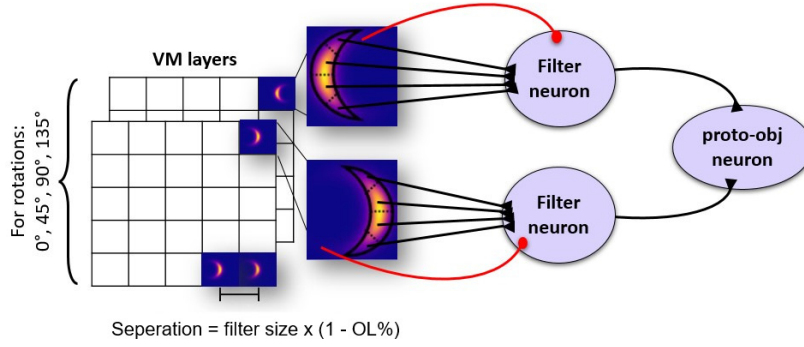


Figure 3.3: Representation of a VM layer and its connections. Each VM filter is split into 4 sections all connected to the same Filter neuron. The blue area around the "active" part of the neuron (the moon shaped yellow region) is connected to the Filter neuron with an Inhibitory connection (red lines). This stage of the model represents the Border Ownership pyramids detecting closed contours. Two complementary VM filters with opposite orientation are then connected to the same Proto-Object Neuron (Grouping Pyramid) to identify possible proto-objects. This structure is repeated for each layer with different orientations of the filter: 0°, 45°, 90° and 135°.

#### *The event that changed it all*

Event-driven cameras' pixels asynchronously produce an event every time a local illumination change occurs larger than a set threshold, providing the information of positive or negative change in contrast. As such, they perform an inherent operation of edge extraction that can functionally be equivalent to the edge extraction performed by *center-surround* (CS) cells in the frame-based model. Similar contrast change information is provided by the CS cells [124]. The event-driven camera does not obtain the local contrast change due to lateral inhibition as in the CS cells, but rather due to the relative motion between the camera and the scene. The two processes are different but the related outcome, the edge extraction and the contrast information, are similar.

#### *PyTorch lights the way to the event*

The inherent capabilities of event-based cameras can be used as substitutes for the first two layers of processing in the event-based version of the saliency-based model [63]:



*center-surround* filtering and edge extraction. Assuming a dynamic scene where a dark object is moving over a white background the leading edge would produce negative events and the trailing edge positive events, therefore providing information about the object contrast with respect to the background.

In the PyTevProto model implementation running on a GPU, the output from the event-based cameras is used to create frames of events divided into positive and negative polarity. The frames of events are fed into the Border Ownership layer following the process explained above. The pipeline following the event-based camera is not spike-based and contains no inhibitory connections, as implemented in the following SNN model.

#### *Event-based through and through*

This work proposes a new fully spike based pipeline, with dedicated neuromorphic hardware, to improve the speed and reduce the latency of the model, dubbed SNNevProto. Code is available at <https://github.com/adamgoodtime/SpiNNiCub>. The SpiNNaker neuromorphic platform [46] acts as a computation medium modelling the SNN in a feedforward architecture (see Figure 3.1). The SpiNNaker *IF\_curr\_exp* neural model mimics the cells as populations of current-based leaky integrate and fire neurons, the default PyNN parameters are used [29]. These neurons process the data coming from the ATIS cameras in the form of events carrying the information of the position in the visual field, polarity (positive or negative contrast change, although they are treated uniformly by the model) and the timestamp of the event.

A full pipeline of SNNevProto can be seen in Figure 3.3. Events are passes into the convolved VM filter segments, on SpiNNaker this is created using the KernelConnector to speed up loading onto the machine. All delays throughout the model are set to 1ms, the timestep of the SpiNNaker simulation. The output of these neurons is combined to create the overall VM filter activity and passed into the filter neuron. The area surrounding the VM is also connected with inhibitory connections to the filter neuron to increase the selectivity of the kernel, in contrast to PyTevProto which did not possess these connections. Complementary filters are then grouped and their activity is passed into the proto-object neuron whose firing creates the saliency map output by the model.

The VM filter, shown in Figure 3.2, is a kernel designed to respond to curved edges that can potentially delimit a closed area. They are formalised as a curve (Equation 3.1) with the largest value at its midpoint providing the ideal shape to respond to closed contours:

$$VM_{\theta}(x,y) = \frac{\exp(\rho \cdot R_0 \cdot \cos(\arctan2(-y,x) - \theta))}{I_0(\sqrt{x^2 + y^2} - R_0)} \quad (3.1)$$

Where  $x$  and  $y$  are the kernel coordinates with origin in the centre of the filter,  $R_0$  is the radius of the filter,  $\rho$  determines the arc length of active pixels in the kernel allowing the convexity of the kernel to change,  $\theta$  its orientation and  $I_0$  is the modified Bessel Function of the first kind [6]. The VM output is then thresholded to reduce sensitivity to localised activity:

$$e(x,y) = \begin{cases} 1 & \text{for } VM_{\theta}(x,y) > 0.75 \\ -1 & \text{else} \end{cases} \quad (3.2)$$

Where  $e(x,y)$  describes whether the pixel at  $(x,y)$  is connected to the filter neuron with excitatory synapses ( $e(x,y) = 1$ ) or inhibitory synapses ( $e(x,y) = -1$ ) (see Figure 3.3). Connection weights,  $w$ , are determined using Equation 3.3 where  $n$  is the size of the pre-synaptic population and  $p$  is the percentage firing threshold for that particular projection between populations. A value of  $5\mu S$  is chosen as it is the minimum weight at which one excitatory input spike produces a spike in the post-synaptic neuron in this implementation of conductance based neurons. As varied kernel sizes are used in the different pyramids, this provides a way to scale weights in a way invariant to filter size and instead a percentage of the population that must fire to produce the same downstream effect. Inhibitory connections are scaled using the same method, but do not produce a post-synaptic spike. Values of the percentage firing thresholds of connection weights can be found in Table 3.1. The filters are used as convolutional kernels which are tiled over the whole image.

$$w = \frac{5}{pn} \quad (3.3)$$

#### *A spiking model*

This implementation of the model is an SNN where the first layer is covered with VM filters spaced with strides relative to their size. Consequently, each VM filter has its own receptive field onto the input layer. Therefore each incoming event triggers a specific pixel belonging to one filter. Each VM filter is composed of four rotationally distributed segments. As the inputs are discrete spikes generated by an event-based camera it is possible for noise and other artefacts to produce a high number of events

input->segment	segment->filter	input->filter	filter->proto-object
0.02%	0.8%	0.0013%	0.75%

Table 3.1: The percentage firing thresholds for different population connections, input->filter is the only inhibitory connection. Percentage firing threshold is the percentage of the pre-synaptic population that need to fire to produce a spike in the post-synaptic population. Inhibitory connections do not induce a spike but are scaled in the same fashion. This metric is used to standardise weights across varying convolutional kernel sizes.

in a small area unrelated to the visual scene. Splitting the VM filter into four sections helps to reduce the sensitivity to localised activity, aiding the filter to respond more selectively to input spikes arranged in the shape of the VM. As the strides of the convolutional kernels are relatively large, appropriate control of VM filter activity is important to reduce undesired spikes and, therefore, inaccurate saliency map generation.

Each filter segment is connected to a neuron representing the entire VM filter. The refractory periods of the segment neurons and input weights to the filter neuron are balanced to require all segments to fire within a narrow temporal window to produce a spike. In addition, all spikes within the filter region that are not part of the VM kernel will have an inhibitory contribution to the combined filter neuron, effectively increasing the selectivity to the VM shape (see Figure 3.2).

The grouping cells, called proto-object neurons, pool the output of VM complementary cells that form a closed contour representing proto-objects (see Figure 3.3). The output of the convolution, and the subsequent output of the proto-objects which form the saliency map, are all represented as spikes emitted by a neuron. The filters exist in 4 rotation pairs with their complementary filters rotated  $180^\circ$ , evenly distributed from  $0-135^\circ$ , and in 5 spatial scales (104, 73, 51, 36, 25 pixels<sup>2</sup>).

Over each layer the VM filters are placed overlapped with each other. Overlap is related to stride used in the convolutional layers of neural networks. Instead of measuring how much the filter has shifted relative to the previous filter it measures how much it is overlapping with the previous filter, which makes it filter size invariant. The overlap among the VM filters is important to define the robustness of the model. In biology, cell receptive fields are often overlapped for robustness, ensuring a response even if a cell no longer functions [40, 128]. Over time, overlapping cells have been used as a way to avoid the aliasing problem in bio-inspired models [21].

The OL (Overlap) percentage increases resolution and accuracy and it is directly

linked to the number of neurons required in the implementation and, hence, its power and computational cost (see Table 3.4). It was therefore decided to use the OL as a parameter of the model to be explored. A percentage is used to ensure a uniform overlap at multiple spatial scales.

Each VM filter is connected with its mirrored counterpart (VM in Figure 3.3) of the opposite side creating a sub-population. All projections between sub-populations share a common weight as described in Equation 3.3. This approach is analogous to tuning the percentage of the pre-synaptic neurons that must fire to produce a spike in the post-synaptic neuron of the next layer. A list of percentage firing thresholds for population projects can be found in Table 3.1. This stage of the SNNevProto mimics the Border Ownership Pyramid in [118].

A similar process to the Border Ownership in [118] pools the activity of mirrored VM filter orientations into a single neuron. The combined filter neuron has maximal activation at the presentation of a closed surface of the same size as the convolution filter size. Following the Gestalt principles [72] this represents detection of a proto-object.

The proto-object spikes are added to a combined saliency map with their energy spread over the surrounding pixels using a 2D Gaussian distribution with standard deviation a third of the filter size in pixels. Therefore, a pooling stage mimicking the Grouping Pyramid is computed making the response size invariant. Values from all scales and the four pairs of rotations are pooled together to produce a combined saliency map.

### 3.4 Experiments and Results

The SNNevProto, the SpiNNaker implementation of the proto-object attention model, was validated by comparing its performance with PyTevProto, the PyTorch GPU implementation [63]. The system is further benchmarked using the Human 2D fixation maps of the NUS-3D dataset [76], obtained recording the eye movements of 14 subjects observing the images of the dataset and averaging their scan paths to create a combined saliency map.

The characterisation compares the responses from the two models qualitatively, showing the strength and the weaknesses of each system. Next, quantitatively, the response was compared between the SNNevProto and the PyTevProto using the latter model as the baseline. The best set of parameters were searched for, exploring different





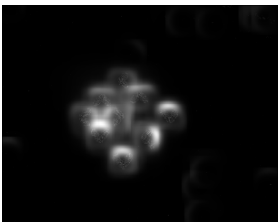

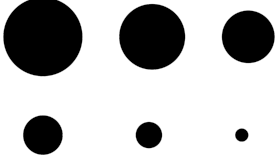
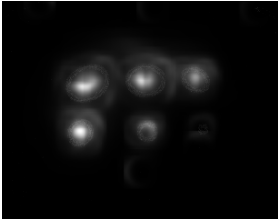

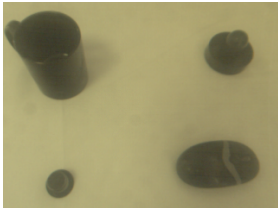
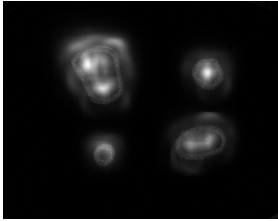
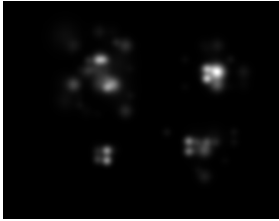
Example #	Input stimulus	PyTevProto SalMap	SNNevProto SalMap
1			
2			
3			
4			

Figure 3.4: Qualitative comparison of the PyTevProto and the SNNevProto. From the left column to the right column: the example number, an RGB image representing the scene shown to iCub (the input stimulus), PyTevProto saliency map and SNNevProto saliency map. These examples are a selection from 13 scenarios of the SalMapIROS dataset. The events are recorded directly from the event-driven cameras mounted on iCub’s eyes. The objects and the 2D printed patterns are placed on a desk in front of the robot. The RGB input images are only for a better visualisation of the input stimulus.

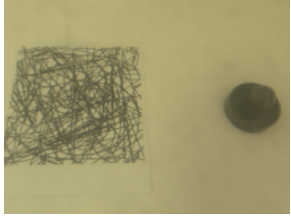
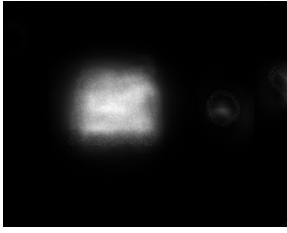
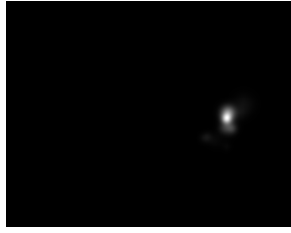
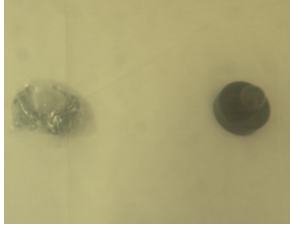
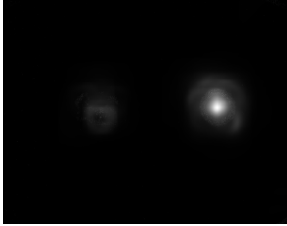

Example #	Input stimulus	PyTevProto SalMap	SNNevProto SalMap
1			
2			

Table 3.2: Qualitative comparison among the PyTevProto and the SNNevProto. From the left column to the right column: the example number, a RGB image representing the scene shown to iCub (the input stimulus), PyTevProto saliency map and SNNevProto saliency map. This table shows only results from clutter experiments of the SalMapIROS dataset. The events are recorded directly from the event-driven cameras mounted on iCub’s eyes. The objects and the 2D printed patterns are placed on a desk in front of the robot.

OL percentages of the VM filters on each layer and the best inhibition value. Increasing the overlap is equivalent increasing the resolution of the model and, therefore, exploring this parameter determines how important resolution is to the final output. Increasing inhibition increases how selective each filter is to the specific VM shape.

#### *The experiments*

To characterise the response, this analysis exploits the SalMapIROS dataset which contains patterns and robotic scenarios with objects and clutter in the scene. The SalMapIROS dataset is obtained recording the events coming from the event-driven cameras mounted on iCub looking at different scenes with real objects or 2D printed patterns. The robot performs small circular periodic stereotyped ocular movements to generate stimulus-dependent activity from event-driven cameras for static scenes.

To estimate the selectivity to a range of sizes a pattern representing circles of different dimensions was used (see Figure 3.4, third row). The other two patterns in Figure 3.4 (first and second row) describe the definition of non proto-object and proto-object exploiting the design used by [118]. The proto-object is represented by the four corners facing each other forming closed contours reminiscent of a square shape. The remaining pictures see objects of different sizes over a desk (fourth row) to study the

Metrics	
<b>Normalized Scanpath Saliency (NSS)</b>	CC approximation, good for saliency evaluation.
<b>Area under ROC Curve (AUC)</b>	Invariant to monotonic transformations, driven by high-valued predictions. Good for detection applications.
<b>Pearson's Correlation Coefficient (CC)</b>	Linear correlation between the prediction and human fixation distributions. Treats false positives and false negatives symmetrically.
<b>Similarity (SIM)</b>	Similarity computation between histograms, more sensitive to false negatives than false positives.
<b>Structural Similarity (SSIM)</b>	Similarity among images, highly sensitive to structural changes.
<b>Mean Square Error (MSE)</b>	Similarity among images, global comparison.

Table 3.3: Metrics summary. This table takes inspiration from [19]

OL%	# of neurons	# of SpiNNaker boards
10%	10428	3
20%	12000	3
30%	15801	3
40%	22266	3
50%	30306	6
60%	48878	6
70%	82084	12
80%	176248	24

Table 3.4: Table showing the number of neurons and SpiNNaker boards required given a percentage of overlapping (OL) for the VM filters. The spalloc server was used to run these jobs which allocates boards in multiples of 3.

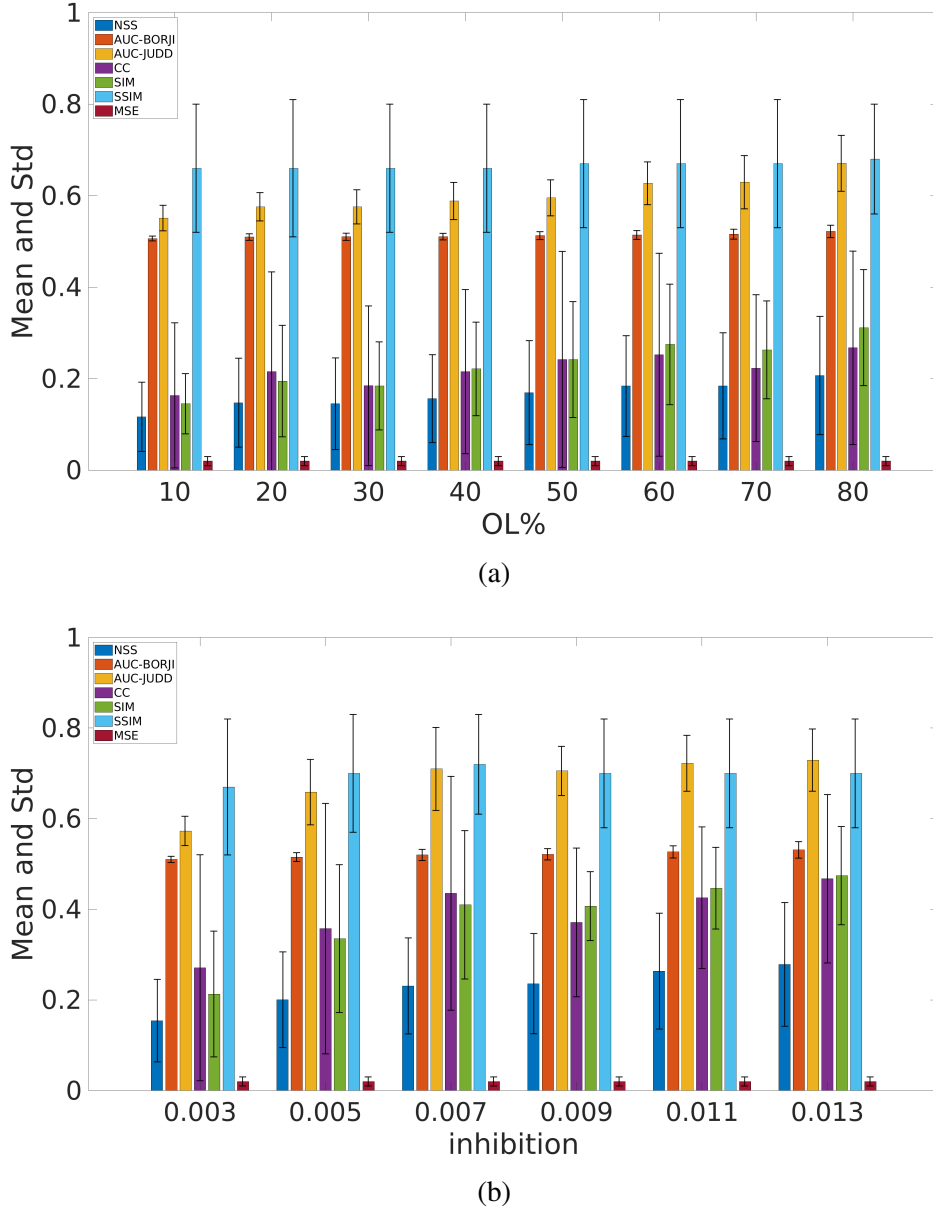


Figure 3.5: Comparison with different metrics evaluating the similarity between the SNNeVProto saliency maps and the PyTevProto saliency maps [63] using the SalMapIROS dataset exploring different OL percentages **(a)** exploring a range of inhibition percentage firing thresholds **(b)** ( $\%/\mu\text{S}$  conductances) with fixed OL percentage at 60%. The metrics used are: the Normalized Scanpath Saliency (NSS), Area under the ROC Curve (AUC-Borji) & (AUC-Judd), Pearson's Correlation Coefficient (CC) and Similarity (SIM) [15, 16, 19, 65], Structural Similarity (SSIM) and Mean Square Error (MSE). A higher score is better for all excluding the MSE where the lower score determines similarity.



applicability of the system in a scenario where the robot is interacting with items in the scene. Table 3.2 shows two cases of simple clutter represented by a pattern and a bag of nails alongside with an object (a puck).

#### *Qualitative response*

Figure 3.4 and Table 3.2 qualitatively show the saliency map from the two models on some samples of the SalMapIROS dataset. Overall, the response from the models is coherent and both implementations detect the objects in the scene. In Figure 3.4 the response from the SNNevProto is more sparse and localised over the targets which is helpful if a robot needs to locate and reach the object. The PyTevProto does not correctly get rid of the clutter in Figure 3.2 (first row) but does in Figure 3.2 (second row). The SNNevProto instead successfully discards clutter in both cases. This result shows robustness to clutter of the SNN model. This behaviour was achieved by tuning the level of inhibition, which PyTevProto does not possess as it does not contain inhibitory connections. By balancing inhibition appropriately the filter can be made selective to the VM kernel shape without silencing the firing of the filter neurons. As the clutter did not contain the specific contours the VM filter is selective to, the inhibition effectively suppresses firing from the filter neurons. PyTevProto does not possess inhibitory connections and therefore is not selective to the VM shape.

#### *Fastest model in the West*

The SalMapIROS dataset has been used also to obtain data related to the latency measurements. As the SpiNNaker simulation is run in real-time, latency is both walk-clock time and simulated time. The results in Table 3.5 show the amount of time needed to obtain spikes from the proto-object neurons, which compose the saliency map, given an input. Each sample is obtained by waiting for the onset of input spikes following a quiescent period and measuring the time taken for activity to flow out of the model. This allows the delay of input spike to consequential output spike to be most clearly extracted. The average latency is 18.5ms (2.40ms standard deviation) and 19.2ms (3.37ms standard deviation) for the second set of samples, compared with the  $\sim 120$ ms needed on average for the PyTevProto model to obtain a saliency map of the scene.

#### *Quantitative response*

Figure 3.5a shows the comparison between the SNNevProto and the PyTevProto saliency maps using the SalMapIROS dataset. The similarity was evaluated among the outcomes using Normalized Scanpath Saliency (NSS), Area under the ROC Curve (AUC-Borji) & (AUC-Judd), Pearson's Correlation Coefficient (CC) and Similarity (SIM) [15,

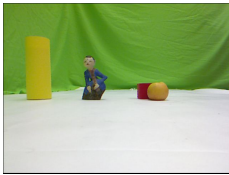



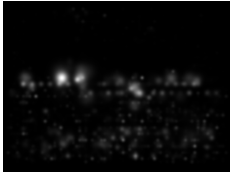
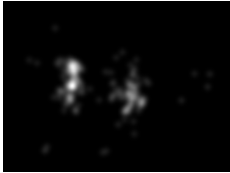
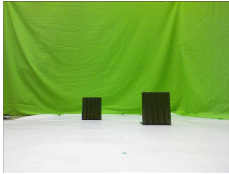


Image #	NUS3D RGB image	SNNeProto SalMap	NUS3D Fixation Map
23			
6			
91			

Figure 3.6: Representation of examples from the NUS3D (robot scenario) dataset. The three columns represent the input RGB image, the outcome from the SNNeProto and the related ground truth from the NUS3D dataset. These examples show how the model performs when the observer fixation maps focus on objects. The response from the model is with 60% OL and 0.013 inhibition.


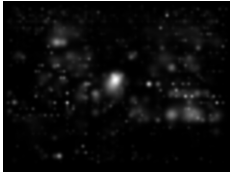


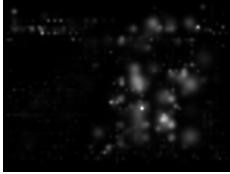




Image #	NUS3D RGB image	SNNeProto SalMap	NUS3D Fixation Map
238			
468			
558			

Figure 3.7: Representation of random chosen examples from the NUS3D (random subset) dataset. The three columns represent the input RGB image, the outcome from the SNNeProto and the related ground truth from the NUS3D dataset. These examples show how the model performs when the observer fixation maps are sparse and unclear. The response from the model is with 60% OL and 0.013 inhibition.

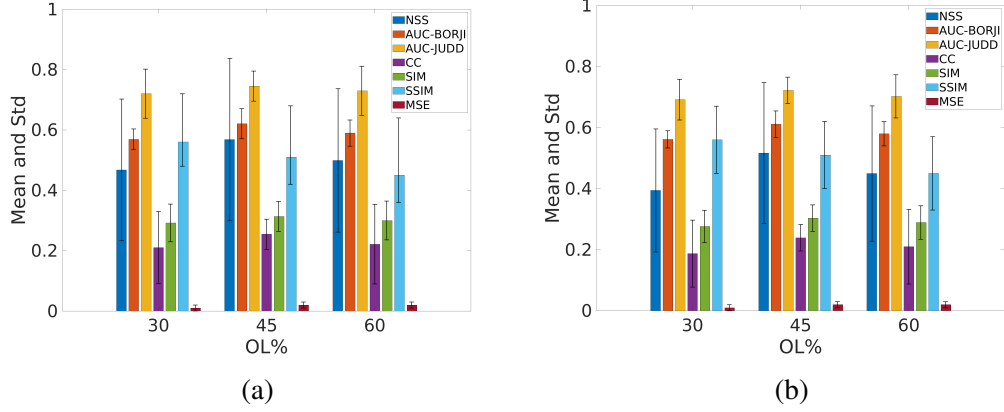


Figure 3.8: Comparison with different metrics evaluating the similarity of the SNNevProto saliency maps with the NUS3D fixation maps (ground truth) [76] in two different subsets (robot scenario (a) and random subset (b)) for different OL percentages. The metrics used are: the Normalized Scanpath Saliency (NSS), Area under the ROC Curve (AUC-Borji) & (AUC-Judd), Pearson's Correlation Coefficient (CC) and Similarity (SIM) [15, 16, 19, 65], Structural Similarity (SSIM) and Mean Square Error (MSE). A higher score is better for all excluding the MSE where the lower score determines similarity.

16, 19, 65], Structural Similarity (SSIM) and Mean Square Error (MSE) (see Table 3.3 for a summary of each metric). These metrics are computed to compare the saliency maps to the human fixation maps, following standard analysis methods in the literature[15, 16, 19, 65]. A single saliency map cannot perform well in all the metrics since they judge different aspects of the similarity between average human saliency and predicted saliency map [75]. These metrics offer a way to determine how well a saliency-based model approximates human eye fixations.

The properties of the chosen images for the benchmark, such as dataset bias (centre biasing, blur and scale), probabilistic input and spatial deviations, affect the result of the metrics [19]. Saliency based models can include such properties. In this work the robot needs to detect objects of different sizes to potentially interact with them. In fact, the SNNevProto only focuses on the scale of the objects rather than other properties. MSE and SSIM are metrics used in classical computer vision to explore the similarity among images. MSE estimates the error between two images and it is a global comparison, and the SSIM estimates the similarity between two images taking into account structural changes in the images.

There is not a significant difference over the OL percentages comparing the saliency maps between the SNNevProto and the baseline (PyTevProto). Only AUC-JUDD and

dataset #	First sample latency [ms]	Second Sample latency [ms]
1	18	19
2	18	18
3	17	18
4	22	29
5	18	15
6	21	19
7	15	17
8	14	16
9	18	19
10	19	20
11	19	20
12	22	19
13	20	21
<b>average</b>	$18.5 \pm 2.40$	$19.2 \pm 3.37$

Table 3.5: Results of latency in milliseconds for different datasets of SalMapiROS. The test is done measuring the latency of two different samples for each dataset. Each row represents a dataset used to measure the latency in two separate samples. Each dataset represents static and dynamic objects placed in front of iCub (such as a paddle, a puck, calibration circles, proto-object patterns, a mouse, a cup and clutter (see Fig. 3.4)

SIM slightly increased with increasing the OL percentage. Although there is not a remarkable increment 60% OL was chosen to explore the inhibition parameter ( $\%/\mu S$  conductances). 60% OL represents a good compromise between the robustness of the model, ensuring enough overlap to cover the whole visual field without losing any area of the visual field, the number of SpiNNaker boards needed (see Table 3.4) and the results obtained. Each significant increment of neurons causes an increase in the number of SpiNNaker boards required. Nevertheless, the number of neurons required does not affect the latency of the model because the pipeline remains unaltered. Figure 3.5b explores a range of different inhibitions showing again no significant incremental or decremental trend. Only SIM and CC show a slight improvement with increasing the inhibition parameter (decreasing the magnitude of inhibition as an increased percentage firing threshold means more spikes are required to produce the same effect). The results exhibit a stable response exploring different parameters showing no need to create a complex network with a large number of neurons to get usable saliency maps. Overall SSIM and AUC-JUDD seem the best metrics to explain the saliency map results.

*Let's fixate on human fixation*

Along with the characterisation where the response of this implementation is compared with the PyTevProto, the response from the model was evaluated by benchmarking the saliency maps with the human fixation maps provided by the NUS-3D dataset [76]. The investigation includes the comparison between the saliency maps generated by the SNNeProto and the fixation maps qualitatively and quantitatively evaluating the similarity between the two maps. The 2D fixation maps of the NUS-3D were collected from subjects looking at images while recording eye movements. The human fixation benchmark obtained recording the response from the subjects includes different mechanisms of bottom-up and top-down processings, increasing the complexity of the observers' fixations. The observer response does not exclusively derive from a data-driven process but also a task-driven mechanism focusing the gaze towards a particular region of the scene. Attention is a complex interplay between these two mechanisms combining bottom-up and top-down mechanisms to perceive the surroundings [149]. The model proposed is a bottom-up system that does not include top-down mechanisms, but 2D fixation maps can be used to evaluate the response of the system as they represent the only benchmark that can be referred to.

To use the NUS-3D dataset within the event-driven proto-object model, the Open Event Camera Simulator [113] is used to shake the images to simulate small periodic circular eye movements. Two subsets of data were chosen from the dataset: one is a selection of 50 images representative of a robotic scenario (robot scenario) and the second one is a collection of 50 random images (random subset). The first subset (see Figure 3.6) represents a simple robotic scenario where objects are placed over a surface. The second subset (see Figure 3.7) is a random selection among all the dataset images adding complexity and variety to the scenarios.

Qualitatively, the saliency maps from the model and the fixation maps are sparse and not easily understandable at a first glance (see Figure 3.6 and Figure 3.7). Figure 3.6 represents a scenario where the SNNeProto Saliency Map and the human fixation target select the same objects as interesting. The highest response (brightest) is located around the objects in the scene. Figure 3.7 shows a slightly sparse response from the model compared to the fixation maps, not allowing a clear understanding of the agent's attention.

Quantitatively, Figure 3.8 shows good results for both datasets exploring different percentages of OL. Furthermore, all the metrics do not show a significant increment changing the OL, validating the response of the model either for simple or for complex

scenarios.

Although the complex bottom-up top-down interplay [149] is not included in this implementation, overall the results yield a good representation of the scene for the work's purposes. Moreover, the metrics used to quantify the similarity do not give equal results among them. All the metrics are used in the literature to explain saliency-based model performances. They compare different aspects depending on the benchmark representation and the definition of the saliency map of the model. These metrics treat differently false negative and positives, viewing biases, spatial deviation and the pre-processing of the saliency maps. Initially there was interest in the location of the responses from the saliency maps rather than the value in that position, for that SSIM could be relied on as the metric. SSIM estimates the structural similarity between two images comparing small sub-samples of the images with each other. This metric well describes this situation where there is more interest in having a response in the same location rather than having the same amount of response in terms of intensity. Other metrics used in the literature were added for completeness [19]. Overall, in this case SSIM seems a good metric to explain the saliency maps. Alongside with the SSIM, AUC-JUDD provides good results too, where each saliency pixel is treated as a classifier splitting them into 'fixation' and 'background'. This metric computes the ratio of true and false positives to the total number of fixations and saliency map pixels using a thresholded mechanism [116].

### 3.5 Conclusion

#### *Inhibiting is exciting*

Overall the response of the spiking implementation of the event-driven attention model on SpiNNaker (SNNevProto) is coherent with PyTevProto, showing a significant improvement in removing the clutter with respect to the baseline GPU-based implementation (PyTevProto). This can be well explained by the nature of the model. The SNN model, as a result of the inhibitory connections, is far more selective to the shape of the VM filter, than in a classic convolution using a kernel with no negative weighting. The convolution will produce activity everywhere the filter overlaps with events, enabling clutter to evoke a response in the saliency map. The advantage of the resulting higher selectivity and localised activity in the saliency map is in the possibility to improve object localisation and segmentation and, hence, the interaction of the robot with the selected object.

For the same structural reason, the response from the SNN is more sparse and focused on the location where the detected objects are placed. Two VM filters of opposite rotation are connected together for every scale and set of rotations. Only when they both respond is there a response from the successive layer of the SNN. Therefore, this helps significantly in generating a more precise saliency map.

*More is not always more*

Given the parallel structure of SpiNNaker, increasing the number of neurons does not affect the latency. For this reason, the model is tested for an increasing OL percentage, and therefore increasing the density of the convolutions. This strategy appears to provide little benefit to model performance and requires the use of additional SpiNNaker boards. Results for low values of OL percentage, equivalent to a large stride in CNNs, produce a similarly reasonable representation of the visual scene compared to high values, with significantly reduced network size. This displays the feasibility of fitting the SNNevProto model on a single SpiNNaker board and having it work in tandem with the iCub humanoid robot which possess a single SpiNNaker board.

*Event-based, more like speed based*

The SNN implementation provides a saliency map of the scene in around 18.9ms. In comparison with the PyTevProto (~120ms), these results are a significant improvement that enables the system to run online in dynamic environments, where the saliency map can be used to drive the gaze and actions of the robot in real-time. To this aim, the SNN implementation on SpiNNaker could easily include Winner-Take-All competition and Inhibition of Return [64] to dynamically select the location of the next saccade within a scene. Additionally, the saliency map allows the system to focus its attention towards a specific target, devoting computational resources to perform other tasks, such as object recognition, only in the area where they are needed.

*Humans - the ultimate benchmark*

Finally, attention and gaze of robots are extremely important in the interaction with humans [147], it is therefore questioned how close the saliency map (used as proxy for the robot's fixational eye movements) was to humans'. The system was validated and characterised, but the quantitative results of the benchmark do not capture the true merit of the model. Quantitatively, the similarity among the benchmark results (robot scenario and random subset datasets) suggests another question; how do we define the complexity of a scenario and which aspects should we take into consideration for attention? These results proved that the random subset does not produce lower results, hence, it may not contain complex scenarios as could be expected.

Each metric captures a specific aspect of the saliency maps, this analysis is instrumental to give a quantitative comparison but mostly to study the effects of the different parameters on the model performance. Moreover, most of the metrics present a high variance due to the mismatch between the SNNevProto and the human fixation saliency maps. This should be investigated in depth creating several subsets from the 600 images of the NUS3D dataset investigating the response's variability. As expected, a pure bottom-up neuromorphic attention system taking into consideration only the input intensity as a feature to determine the saliency map only partially predicts the fixational eye movements of humans. To this aim, the model can be enriched with other channels (such as motion, depth, texture, etc) and with top-down processing to focus the attention towards a specific task.

The model could benefit from the leveraging of learning dynamics in the fine tuning of network parameters. This could allow the model to adapt itself to particular data sets and reach a higher level of performance. This may improve the inference of the model given appropriate training and data as compared to handcrafted parameter selection. It may also provide specific goal driven attention, which could better match the fixation maps of humans, especially if trained using human fixation maps.

Moreover, the spatial integration [26] and the lateral inhibition [31] could enrich the model following a detailed biologically inspired pipeline and further reducing the amount of data to be processed. Finally, further experiments could be done emphasizing the clutter removal capabilities exploring the potential of the model.

#### *The wider context*

This chapter has displayed how biologically inspired mechanisms can be used to drive attention. The translation into a spike based implementation enabled a sparse and efficient implementation to be run on the neuromorphic hardware SpiNNaker. This created a biologically plausible attention system both at the algorithmic and computational level. Compared to its non-spiking counterpart, PyTevProto, it boasted sparser activation and significantly reduced latency. As a result of its hand crafted design it functioned in a limited capacity with differences between its saliency and human fixation maps likely coming from the difference in the primitives driving attention in the two cases.

Although saliency is driven by high level features in SNNevProto, it lacks the endogenous drives and biological primitives, such as being attentive of faces, that humans possess. But how do you instil a network with alternative forms of attention? To remain



within a hand crafted regime new filters could be constructed but this makes every alteration an engineering problem. Learning mechanisms allow function to be driven by data and remove the need for engineered solutions. The problem then becomes, how do you train a network? There are many solutions within ANNs (Artificial Neural Networks) and a large portion of their techniques are transferable to SNNs (see Chapter 4 for further details), although there are still many unanswered questions about how to unlock the full potential of spiking neurons and how biology has managed to reach the computational complexity of the brain. The following chapters will explore potential techniques to train neural networks in a biologically inspired way.

# Chapter 4

## E-prop on SpiNNaker

### 4.1 Introduction

#### *Time for intelligence*

Biological processes have approached the generation of intelligence across two timescales. The first is in the order of millions of years with evolutionary processes guiding random genetic mutations in an organism's DNA towards better adaptation to its environment. This technique proved effective with an incredible diversity of solutions created. However, as the evolutionary process works over a long timescale it can only provide slow alteration to a creature's behaviour, which limits the capacity of intelligent actions. Evolution then stumbled upon a method to increase the capacity for behavioural adaptability through lifetime learning.

Learning during the process of one's life enables acquisition of information about an environment without the need for evolutionary adaptation. This provides a huge benefit to an organism relative to its non-learning counterparts. An animal can learn to avoid dangerous environments and remember places of likely food sources without the need for evolutionary pressures to build in some preference. It can also allow adaptive modelling of prey and competition giving the edge in prediction and action. The majority of current machine learning techniques fall within this second category of fast timescale learning. A neural topology is created (expert knowledge simulating the evolutionary process of brain development) and then a learning algorithm incorporates knowledge into the parameters of the network.

#### *Biology inspires creativity*

The progress of neural network research has highlighted the importance of architecture in the model that can be generated by an ANN (Artificial Neural Network). A

multi-layer feedforward architecture is a universal approximator [25] and yet the advent of CNNs (Convolutional Neural Networks) revolutionised computer vision by altering the architecture learning is performed across. CNNs mimic the connectivity of the early mammalian visual system [60] and the sharing of weights (which is not biologically plausible) allows reduced computational load in training and memory whilst also solving problems of translation invariance. Although ANNs can approximate any function [25] and can be trained precisely with GD (Gradient Descent), inspiration from biology enabled a large step to be made within machine learning.

*What follows what came first*

First, lifetime learning algorithms will be discussed within the context of neural networks, specifically SNNs (Spiking Neural Networks), with their limitations contrasted with EAs (Evolutionary Algorithms). Next an implementation of the e-prop learning algorithm will be presented with its instantiation on the SpiNNaker neuromorphic hardware. Finally, the work performed will be put in the greater context and future directions of research will be suggested.

## 4.2 Background

*Evolution to an update solution*

Learning algorithms come in many forms but in this context they are considered separate from evolutionary algorithms in that they do not use a population of individuals and instead create a single model which gradually incorporates information about the world through experience. Compared to natural Darwinian evolution, they operate on much shorter time scales and in biology this enables an individual to adapt to their surroundings during their lifetime, increasing their individual fitness. There have been many attempts to harness the learning capacity of the biological brain with neural networks forming the foundation of the most successful modern approaches. Training algorithms often rely on the updating of network parameters as governed by some error metric with the most successful forms using a mathematical gradient of the network parameters with respect to the error. With successive data presentations this characterises a learning dynamic in which performance on a task increases with time.

As the field of neural networks matures, the methods available to harness their power have diversified accordingly. The selection of neuron type and learning algorithm allows tailoring of network training to match a range of different applications.

BP (Back-Propagation) is considered a state of the art algorithm with the power to calculate the gradients of weights in deep networks and incrementally incorporate knowledge about errors into the weights of the network. An approximation of the network's gradient with respect to the error is leveraged to move the network weights to a position in the solution space with reduced overall error. This requires a record of the network response to be combined with an error, which is propagated backwards through the same weights the activity was first passed forward, thus *back*-propagation. Despite its effectiveness as an algorithm, researchers struggle to posit how the biological brain could achieve such calculation. The main biological challenge is the weight transfer problem, in which the path the error is propagated back along must have knowledge of the forward propagation's contribution to previous states [81]. In addition, the algorithm requires the halting of network activity whilst updates are calculated, operating offline and restricting instant incorporation of new information.

#### *Continuing past discontinuity*

A major challenge with the application of gradient descent algorithms used in deep-learning to SNNs is the discontinuity of the spiking neuron's activation function. As the derivative is undefined around spiking, a gradient cannot be as easily extracted as with ANNs, making it difficult to construct weight updates for training. A number of possible solutions have been proposed to facilitate the application of gradient descent algorithms to SNNs. Broadly speaking they fall into three categories: pseudo derivatives and surrogate gradients [8, 66, 77, 100, 126], which apply some function to the membrane voltage to approximate the differential of neuron activity; smoothed activation curves [62, 78], where LIF (Leaky-Integrate-and-Fire) neuron dynamics are altered to enable continuous gradients; and spike time based gradients [23, 96, 148, 151], in which the timing of the spikes is used to generate a derivative of the weight with respect to the error. E-prop (discussed later in this chapter) falls into the first category in which pseudo derivatives are used to approximate the gradient of neuron activation.

#### *Biology never switches off*

The above mentioned algorithms attempt to tackle the issue of transferring gradient based learning to SNNs but because of this they fall into the same traps as GD with ANNs, notably in this case the need to compute offline. Biological brains are continuously acquiring information and updating models without access to global information or the state of other synapses. There are neurotransmitters which can have a wider local effect but GD algorithms often rely on the knowledge of the weights across a network, a record of activity over multiple instances and a global error signal which

are not as biologically plausible. E-prop addresses these problems with its online and local formulation and error signal which acts in a way analogous to neurotransmitter release.

### **4.3 Online and local learning**

Artificial intelligence pervades everyday life, from smart fridges to phone assistants, yet often relies on queries sent to a remote server running a pre-trained AI model such as a neural network. Although functionally this provides more computational power than is available on an edge device to a neural network model, it also precludes local tailoring of the system. Local learning has the potential to enable speech recognition to become tailored to your voice and facilitate a personal understanding of queries. There is the added benefit of removing the need to communicate personal data if the network is kept locally, aiding individual privacy. Computing locally also opens up edge applications such as satellites and operating in hazardous environments in which communication with a remote system may prove limiting.

As training becomes more parallel and networks become larger, we must look to novel hardware to meet a wider range of computational requirements. Neuromorphic systems aim to fill this niche by providing novel computing architectures dedicated to SNN simulation. In contrast to standard ANNs, which require information to be passed along a synapse at each timestep, SNNs communicate state intermittently through a discrete spike indicating that accumulated input crossed a threshold. This creates a temporal element to SNN computation, which is missed in standard ANNs. SNNs have been posited as the next generation of neural networks [85], however challenges remain around how to train them to reach and surpass the performance of their non-spiking counterparts.

A recent breakthrough in training SNNs on real-world tasks was the development of the e-prop learning rule [9]. This work developed a learning rule based on synaptic traces, enabling online training in both supervised and reinforcement learning scenarios. The proposed synaptic traces capture an ‘eligibility’ for the weight updates that drive learning, and are computed based only on locally available information, avoiding the weight transfer problem [81]. In contrast, standard ANN training requires information of neuron activation to be passed backwards through the network, requiring global information which is not biologically plausible. This localisation makes e-prop a candidate for implementation in neuromorphic hardware, potentially offering a low-power

online training paradigm for spiking neural networks, with accuracy performance comparable to the commonly employed BPTT (Back-Propagation-Through-Time) training algorithm in recurrent ANNs.

The following work explores the requirements of neuromorphic hardware to run the e-prop learning algorithm online and in real time. SpiNNaker is used as the neuromorphic development platform as its programmable nature enables implementation of the e-prop algorithm. Its architecture also imposes similar restrictions to other neuromorphic systems (e.g. co-location of memory and compute). This setup is applied to two tasks, one explores the application of e-prop to a wave-matching task and the next tests the ability to learn when the error signal is delayed in time from the inputs. First, the algorithm and computing platform will be discussed.

### 4.3.1 E-prop

E-prop [9] is a biologically inspired online learning algorithm for recurrently connected neural networks. It incorporates historic information, obtained and stored locally on a cellular and synaptic level, to accurately update the connectivity of an SNN in response to a global learning signal. Changes to the network's synaptic weights can be made in an online fashion, at each simulation timestep or in batches, incorporating performance information in real time. This is an attractive advantage in that past states of the network, or indeed its performance over the entire duration of a task, need not be precisely recorded to make adjustments to synaptic weights, as is required by BPTT and other learning algorithms for ANNs.

#### *Components*

The e-prop learning rule has been developed using two spiking neuron models, and can be employed in networks comprising one or a combination of both types. LIF neurons linearly sum inputs as contributions to their membrane potential, which decays over time. The neuron will release an action potential, or fire, when a critical membrane potential is reached, after which the potential resets to a baseline and cannot fire again for a short refractory period. In a LIF model, the internal state of the neuron can be wholly described by its membrane potential and time since last action potential. In contrast, ALIF (Adaptive-Leaky-Integrate-and-Fire) neurons allow the neuron's firing threshold to vary relative to its spike frequency. This threshold adaptation cannot be immediately measured from the neuron's current membrane potential, and as such, the firing threshold in ALIF neurons is considered a hidden variable and must be stored internally. E-prop depends on a pseudo-derivative of the membrane voltage and firing

threshold for each neuron that, combined with synaptic activity, allows quantification of its behaviour over time in an *eligibility trace* that is unique to each synapse.

It is important to note that the eligibility trace is derived from purely local information, not shared between neurons or over the entire network. By low-pass filtering the synapse and neuron variables an approximation of the contribution to performance through time can be made. Information about the performance of the network at any moment in time is provided globally in the form of a learning signal and propagated to all neurons in the network via random feedback weights [81]. This can be described as analogous to a neurotransmitter being released and diffusing through the brain. The learning signal only quantifies errors in the network's output at the current point in time, but combined with the eligibility trace it allows synapses to be adjusted according to their impact on the network's behaviour over time. This reliance on local, synapse-specific historic information and only one global error signal is well-suited to implementation on neuromorphic systems.

### 4.3.2 SpiNNaker

A summary of the SpiNNaker architecture and design can be found in Section 2.3, with specific alterations the SpiNNaker architecture described later in this chapter in Section 4.4.3 Code is available at <https://github.com/SpiNNakerManchester/> with the separate repositories being: DataSpecification, PACMAN, PyNN8Examples, spalloc, SpiNNakerGraphFrontEnd, spinnaker\_tools, spinn\_common, SpiNNFrontEndCommon, SpiNNMachine, SpiNNMan, SpiNNStorageHandlers, SpiNNUtils, sPyNNaker, sPyNNaker8 and SupportScripts is also useful. sPyNNaker uses the branch *eprop\_adaptive* for the sinewave matching task and *eprop\_left\_right* for the temporal credit assignment task. All other branches are master if *eprop\_adaptive* is not available.

## 4.4 Implementation and experimental design

### 4.4.1 An overview of e-prop

#### *The pseudo gradient*

Like other GD techniques, e-prop relies on assigning credit to network weight parameters in relation to an error signal. Instead of directly calculating gradients, as in BP, e-prop uses a combination of the incoming spike train,  $z$ , and the pseudo derivative of the post-neuron's membrane potential,  $\psi$ . This essentially assigns credit to the

synapses that had a lot of spikes move across them and are connected to neurons which are close to threshold, as the peak of the triangle-like pseudo derivative is centred at the threshold. It could also be thought of as determining which synapses would affect performance the most if altered. The combined incoming spike train and pseudo derivative creates the synapse specific eligibility,  $e$ . The exact equations describing this will be discussed later.

*The error creates the correction direction*

The environment generates the error,  $E$ , which is broadcast to the network. The error is passed down randomly distributed feedback weights,  $B$ , which creates the neuron specific learning signal,  $L$ . This relies on the principle of random feedback alignment which states that the error distributed to neurons does not need to be directly related to their activity or synapse weights and that the learning undertaken begins to align with  $B$  and improve the performance of the network [81]. Essentially it determines beforehand which weights will be strongly effected by the error and by what sign. The learning signal is then combined with the synapse eligibility to create the weight update. Figure 4.1 shows the general structure of learning.

E-prop offers an alternative to BP by locally storing information about previous activity in an eligibility trace, capturing the effect of making a change in the synaptic weight, akin to a gradient. This record allows weight updates to be calculated online without the need for global knowledge of the connection weights and any propagation of errors through the network. Error is instead incorporated via a global error signal, which is broadcast to the network [8]. This error signal is passed along random feedback weights, which have been shown to enable similar learning to symmetric feedback weights because the network weights and feedback weights align during learning [81]. While there are variants of the e-prop algorithm in which learning takes place on these feedback weights to allow them to align with the feedforward weights, these are not explored in this work.

## 4.4.2 E-prop neuron models

*Working spiking neuron working memory*

For many applications a direct input-output mapping is sufficient for task completion, e.g.  $F(x) = y$  to classify a picture. However, there are many tasks in which inputs have to be accumulated over time, e.g.  $F(x^t, \dots, x^{t-k}) = y$  to generate the spoken text of an audio sample. The later type requires some internal state of the network to store information about previously received input, also known as working memory.



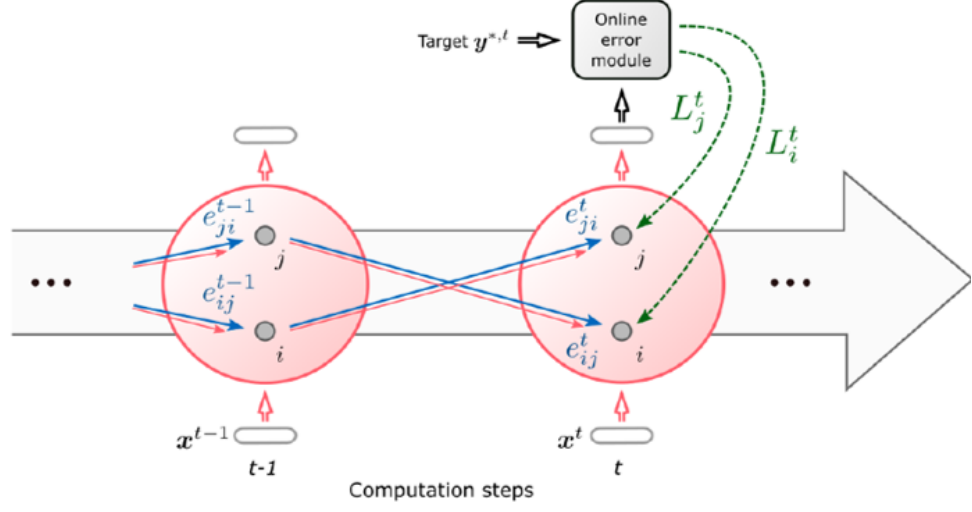


Figure 4.1: A diagram from the original e-prop paper published in Nature [11]. It shows how the eligibility,  $e_{ji}^t$ , at time  $t$  is synapse specific, in this case between neuron  $i$  and  $j$ . Inputs,  $x_i$ , are passed into the network at each timestep as well as the neuron specific learning signal,  $L_j^t$ , generated by the error module which is a part of the task environment.

In standard ANNs this could imply the use of LSTM (Long Short Term Memory) units [55] due to their memory mechanics possessing a differentiable gradient. There are approaches that achieve similar working memory in SNNs. The mechanism implemented in this work employs ALIF neurons. These neurons increase their threshold after firing, creating a memory of previous input, before decaying back down to resting threshold. Equation 4.3 describes the current state of the ALIF neuron's threshold and Equation 4.4 governs how the adaptive threshold increases following a spike and decays back to resting voltage threshold. Training an SNN with ALIF neurons has been shown to produce comparable results to training an ANN with LSTM units [8].

#### *The adaptive neuron's fixed equations*

The membrane voltage of neuron  $j$  at time  $t + 1$ ,  $v_j^{t+1}$ , for an ALIF hidden neuron is governed by Equation 4.1, where  $W_{ji}^{\text{rec}}$  is the weight between neuron  $i$  and  $j$ ,  $z_i^t$  is a binary value indicating whether neuron  $i$  spiked at time  $t$ ,  $W_{ji}^{\text{in}}$  is the weight between neuron  $j$  and input  $i$ ,  $x_i^t$  is a binary value indicating whether input  $i$  spiked at time  $t$  and  $A_j^t$  is the current value of the adaptive threshold,  $A_j^t$  is multiplied by  $z_j^t$  to reset the membrane voltage by subtraction following a spike by neuron  $j$ . The membrane voltage at time step  $t$  of neuron  $j$  is decayed by  $\alpha$  which equals  $e^{-\delta t / \tau_m}$ , where  $\delta t$  is the timestep of the SpiNNaker machine which for all simulations is set to 1ms and

$\tau_m$  governs the rate of decay of the membrane voltage, it is set to 1000ms, making  $\alpha \approx 0.999$ .

$$v_j^{t+1} = \alpha v_j^t + \sum_{i \neq j} W_{ji}^{\text{rec}} z_i^t + \sum_i W_{ji}^{\text{in}} x_i^{t+1} - z_j^t A_j^t \quad (4.1)$$

Equation 4.2 describes how a spike in neuron  $j$  is produced if its membrane voltage goes above its adaptive threshold  $A_j$ , otherwise no spike is produced. The membrane voltage reset is governed by the last term in Equation 4.1 which is a reset by subtraction.

$$z_j^t = \begin{cases} 1, & \text{if } (v_j^t - A_j^t) > 0 \\ 0, & \text{otherwise} \end{cases} \quad (4.2)$$

Equation 4.3 below describes how the adaptive threshold changes with time. It is a combination of the base threshold value,  $v_{th} = 10\text{mv}$ , and adaptive threshold component,  $a_j^t$ , which is scaled by  $\beta = 10$ .

$$A_j^t = v_{th} + \beta a_j^t \quad (4.3)$$

The adaptive threshold component,  $\alpha$ , of the combined adaptive threshold,  $A$ , is governed by Equation 4.4, showing it increases by 1 following a spike from the neuron and is decayed by  $\rho$ , where  $\rho = e^{-\delta t / \tau_a}$  with  $\tau_a = 6500\text{ms}$  and  $\delta t$ , as earlier, being the timestep of the SpiNNaker machine set to 1ms in these experiments.

$$a_j^{t+1} = \rho a_j^t + z_j^t \quad (4.4)$$

#### *Calculating calculation*

As determined in Equation 4.5, this leads to the pseudo derivative,  $\psi$ , for the neurons, where  $v_{th\text{base}}$  is the base voltage threshold at which a spike is emitted. A value of  $\gamma_{pd} = 0.3$  is used to scale the pseudo derivative, as in the original work [8].

$$\psi_j^t = \frac{1}{v_{th\text{base}}} \gamma_{pd} \max \left( 0, 1 - \left| \frac{v_j^t - A_j^t}{v_{th\text{base}}} \right| \right) \quad (4.5)$$

$A_j^t$  is the value of the adaptive threshold for neuron  $j$  at time  $t$  (for the LIF neuron  $A_j^t = v_{th\text{base}}$  when calculating the pseudo derivative).

The eligibility,  $e_{ji}^t$ , of the synapse between neuron  $i$  and  $j$  at time  $t$  is shown in Equation 4.6, where  $\tilde{z}_i^{t-1}$  is the lowpass filtered incoming spike train and  $\epsilon_{ji,a}^t$  is the

adaptive threshold contribution which is zero in the LIF neuron case. It is determined using Equation 4.7 and initialised to zero. The variables used in this equation are the same as presented earlier.

$$e_{ji}^t = \psi_j^t \left( \bar{z}_i^{t-1} - \beta \epsilon_{ji,a}^t \right) \quad (4.6)$$

$$\epsilon_{ji,a}^{t+1} = \psi_j^t \bar{z}_i^t + (\rho - \psi_j^t \beta) \epsilon_{ji,a}^t \quad (4.7)$$

Finally, the eligibility is lowpass filtered to give the eligibility trace,  $\bar{e}_{ji}$ , and multiplied by the learning signal,  $L_j^t$ , and learning rate,  $\eta$ , to give the synaptic weight update  $\Delta W_{ji}$  shown in Equation 4.8. The error,  $E_k^t$ , for output  $k$  at time  $t$  is produced by subtracting the target output,  $y_k^{*,t}$ , from the predicted output,  $y_k^t$ . The neuron specific learning signal,  $L_j^t$ , is created by multiplying the  $E_k^t$  by the feedback weight,  $B_{jk}$ , for neuron  $j$  and output  $k$ . The weighted error for each output is summed to produce the overall learning signal for the neuron.

$$\Delta W_{ji} = -\eta \sum_t \underbrace{\left( \sum_k B_{jk} \overbrace{(y_k^t - y_k^{*,t})}^{=E_k^t} \right)}_{=L_j^t} \bar{e}_{ji} \quad (4.8)$$

The lowpass filtering of the incoming spike train,  $z$ , and eligibility,  $e$ , is used to effectively produce a running average of their values and is controlled by Equation 4.9, where  $\bar{c}$  is the lowpass filtered version of  $c$  (which stands in place of  $z$  and  $e$  here) and  $\alpha$  is the same value as earlier when decaying the membrane voltage, approx. 0.999.

$$\bar{c}^{t+1} = \alpha \bar{c}^t + (1 - \alpha) c^t \quad (4.9)$$

### 4.4.3 SNN architecture & mapping to SpiNNaker

This work defines SNN architectures according to the generic description displayed in Figure 4.2(a). Neural network models are defined in PyNN [30], created from groups of neurons known as *populations*, with models typically containing populations of: input neurons providing network stimuli, hidden neurons providing the core computation, and readout neurons capturing network output. Projections capturing synaptic connections are made between populations, as depicted by the solid blue and green arrows in Figure 4.2(a). All connections are trained using the e-prop learning rule, and are therefore said to contain e-prop synapses (Sec. 4.4.2). The readout neurons

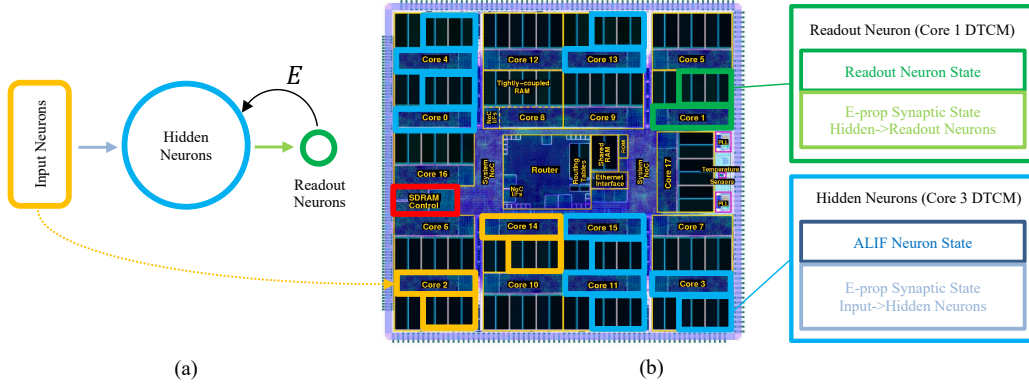


Figure 4.2: (a) Generic spiking neural network architecture suitable for training with e-prop: input neurons provide network stimulation; hidden neurons perform the computation; and readout neurons capture network output. During training, the readout neuron population additionally computes an error  $E$  via supervision, and communicates this to the rest of the network to drive learning. (b) Populations of neurons are partitioned and mapped on individual SpiNNaker cores, with synaptic information stored local to the postsynaptic core.

capture network output, and during training are therefore able to quantify the error between the network prediction and a target. While this target can take different forms in different problems (Sec. 4.4.4), the process of distributing an error is common, and forms the learning signal  $L$ , which drives plasticity (solid black line in Figure 4.2(a)). When modelling neurons and synapses on SpiNNaker, neuron populations are partitioned into sub-groups suitable for execution on a single core. This enables large-scale neural networks to be partitioned and mapped to the many-core SpiNNaker architecture [117]. This process is displayed in Figure 4.2(b), with the input population divided across two SpiNNaker cores and the hidden population across six, while the readout population occupies a single core. This partitioning enables parallelisation and distribution of operations over multiple chips, and hence demonstrates the potential of the system to scale to larger SpiNNaker machines and SNNs.

A number of changes are made to the standard SpiNNaker programming model for SNNs [114], to accommodate the unique features of the e-prop learning rule for specific tasks. While spikes are communicated between cores using the conventional multicast packets with no payload, the learning signal is distributed via multicast packets with 32-bit payload. This payload carries the error, which is evaluated and sent on the core executing the readout population. While this error represents the global learning signal, each individual neuron receives a uniquely-weighted learning signal

by combining the globally-distributed value with an individual feedback weight, represented as an additional parameter within the neuron model stored locally to the receiver. Storage and updating of the synaptic matrix also requires a new approach relative to the conventional SpiNNaker programming model [114]. The SpiNNaker chip was designed to house large data structures such as synaptic matrices in the chip-level 128 MB SDRAM (Synchronous Dynamic Random-Access Memory), as subsections of these structures are typically only required (and therefore fetched) on the arrival of a spike. However, the e-prop learning rule requires direct access to the synaptic state, as updating the eligibility trace requires continuous access to both the incoming spike train and the postsynaptic neuron state. In this work these data structures are therefore located in core-local memory, enabling them to be updated on every timestep along with the neuron state. This is possible on SpiNNaker due to its programmable core and flexible memory allocation. However, the relatively small local memory (64 kB), limits the total number of e-prop synapses per hidden neuron. A total of 6 intermediate state variables are used to capture the operations of Eqs. 4.6 & 4.8, meaning a total of 24 bytes are used per e-prop enabled synapse. In this work cores simulating hidden neurons were therefore assigned 8 neurons, each with a potential 256 e-prop synapses.

#### 4.4.4 Tasks

Two tasks were implemented to display the learning capabilities of e-prop on neuromorphic hardware. The first is tailored to test the ability to match a target output in an online manner. The second is to explore the learning capability when the error signal is delayed in time relative to the inputs, referred to as the temporal credit assignment problem. Together they encompass a regression task and a classification task which requires working memory.

#### Readout neuron design

As SpiNNaker is a real-time neuromorphic system, any environment or test a simulation interacts with must also be processed in real-time. This puts a number of constraints on network simulation and testing. The most pertinent restrictions are memory, with regards to hardware restrictions on shared memory, limiting the scale and complexity of environment simulation on chip. Without simulating the environment on chip a large amount of data would need to be streamed on and off the SpiNNaker system, which has data rate limitations and restricts potential parallelism.

The readout neurons act to encapsulate everything that is needed around the network for learning. This population retains the state of the environment and uses that to control the inputs as necessary. Another key part of its function, and why it is called the readout population, is the incorporation of output neurons. They receive spikes and update membrane potentials as leaky integrator neurons that cannot produce a spike. The membrane voltages of these readout neurons act as the output values that are used for task evaluation. They need to be kept on the same core as the test environment to enable quick access to the current values for error calculation at each timestep.

### Task 1: Wave-form matching

The task explores the training of the readout neurons to produce a 1-D continuous function from a fixed predefined and repeating Poisson spiking input. The inputs are all-to-all connected to the outputs via plastic weights. A target wave-form is composed from the combination of two sine waves of different frequencies, as shown in Equation 4.10. Possessing this capability translates to a number of tasks, such as creating the appropriate motor commands to move a joint in a robotic arm. Following this, from the Fourier transform, all continuous signals can be described as a combination of sinusoids. Therefore, being able to match a target wave-form displays the theoretical ability to match any target signal.

$$y = A\sin(pt) + B\sin(qt) \quad (4.10)$$

Here  $y$  is the output of the target wave-form at time  $t$ ,  $A = 2$ ,  $p = 4\pi/1024$ ,  $B = 2$  and  $q = 8\pi/1024$ .

The network receives input from a population of 100 neurons firing at an average rate of 10Hz. The spikes are generated by a Poisson process and repeated every presentation of the target signal. Replaying the same inputs each trial is crucial to performance. If they are controlled by a random process, even with the same average rate, the network is not able to produce the target output as there is no regularity to sample from. This is a result of the readout neuron requiring a repeating input to create a repeating output. Further improvement in performance was achieved by staggering the inputs in time, although, this is not required to solve the task. By splitting them into 20 groups firing intermittently at 200Hz, maintaining a population average firing rate of 10Hz, the network receives more timing information and can better match the target signal. Inputs are directly all-to-all connected to the readout neurons with weights

initialised from a normal distribution, ( $\mu = 0, \sigma = 1$ ) and divided by the square-root of the input size. A learning rate of  $\eta = 0.04$  is used. The error is described by

$$E = 0.5(y^* - y)^2 \quad (4.11)$$

$$\delta E = y^* - y \quad (4.12)$$

here  $E$  is the error between readout output,  $y$ , and target output,  $y^*$  and  $\delta E$  is the partial derivative of the error which will be broadcast to the network. The network output is the membrane voltage of readout neuron. It cannot spike to allow the membrane voltage to act a readout for a continuous value.

The target value for each timestep is generated beforehand and loaded onto the machine as part of the readout neuron's parameters. The target wave is a repeating signal of length 1024 timesteps, which is looped back through at the end of each presentation. At each timestep the membrane potential of the readout neuron is compared with the current target value and an error is produced via Equation 4.11. The partial derivative of this error, Equation 4.12, is broadcast to all neurons at every timestep via a multi-cast packet with payload. The feedback weights are stored locally at the neuron and are multiplied by the broadcast error to produce the neuron specific learning signal for that timestep,  $L'_j$ . The learning signal is then multiplied by the synaptic eligibility trace,  $\bar{e}_{ji}$ , to produce the weight updates, as described in Equation 4.8. These weight updates are accumulated and used to update the network weights at the end of the presentation (Equation 4.8), acting as a small batch update, but performed online without halting SNN execution.

## Task 2: Temporal credit assignment

This task explores the application of working memory to a classification task. An environment is simulated in which the network acts as a mouse moving down a hallway in search of a reward, as in the original experiment [95], see Figure 4.3 for a diagram of experimental setup. The mouse moves down the hallway and at intervals cues are presented on either the left or right with seven cues presented in total. Following this a prompt signal is given to the network to signify the end of the hallway is reached and it must decide whether it received more cues on the left or the right, which corresponds to the location of the reward. Completing this task requires storing of cues in the working memory of the ALIF neurons and combining this information to evaluate the correct choice. The error signal is evaluated and delivered at the end of the test. This requires

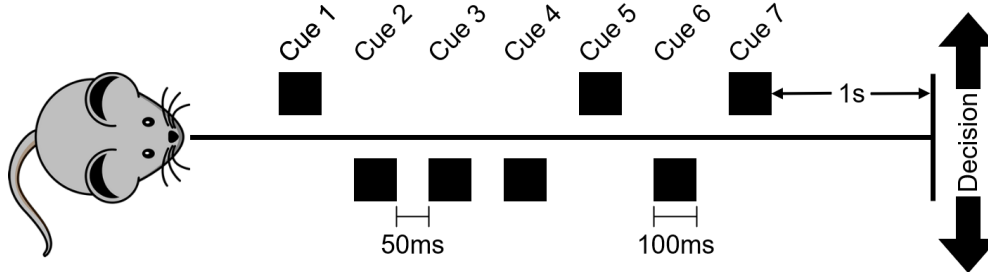


Figure 4.3: Experimental setup of the temporal credit assignment task. The mouse starts at the beginning of a hallway and is presented with left and right cues as it progresses down. Each cue lasts 100ms with a 50ms gap between each. There is a 1 second wait following the final cue before a prompt signal is sent for the network to make a decision which of left or right presented the most cues.

credit to be historically assigned to each neuron's performance through time.

Following the experimental setup used in the original e-prop implementation of the task [11], the network receives from four inputs: left cue, right cue, prompt and random noise. The input is population encoded as a variable rate Poisson spike source of 10 neurons for each input. The rate of the Poisson input is controlled by the readout population which can send rate updates in the payload of a packet to update the rate of the Poisson source depending on the state of the task. The random input fires constantly at 10Hz, which acts as a bias for the neurons to sample from. The cues produce no spikes, 0Hz, unless selected, at that point the rate is increased to 100Hz for 100ms before returning to zero. There is a 50ms wait before a new cue is randomly selected and presented to the network. After the cues have been presented, there is a 1s wait before the prompt signal of 100Hz is given to the network for 150ms. This is the only point in the test at which an error is generated and broadcast to the network. It is because of this that credit must be applied to temporally shifted activity.

The test is set up in an incremental fashion, with the number of cues increasing only when the current difficulty is solved. This is determined by averaging the classification accuracy over the last 64 tests, with the number of cues increased once this average crosses 90%. Starting from one random cue, the number of cues increases by two each time threshold performance is achieved until the final difficulty of seven cues is solved, after which learning is terminated. The readout neuron handles the timings of the experimental inputs and generation of error. The cues are randomly selected and an updated Poisson rate is sent to the appropriate inputs. After the cues have all been sent and following the 1s delay the prompt signal is delivered to the network. The error between the correct output choice and readout membrane potentials is broadcast



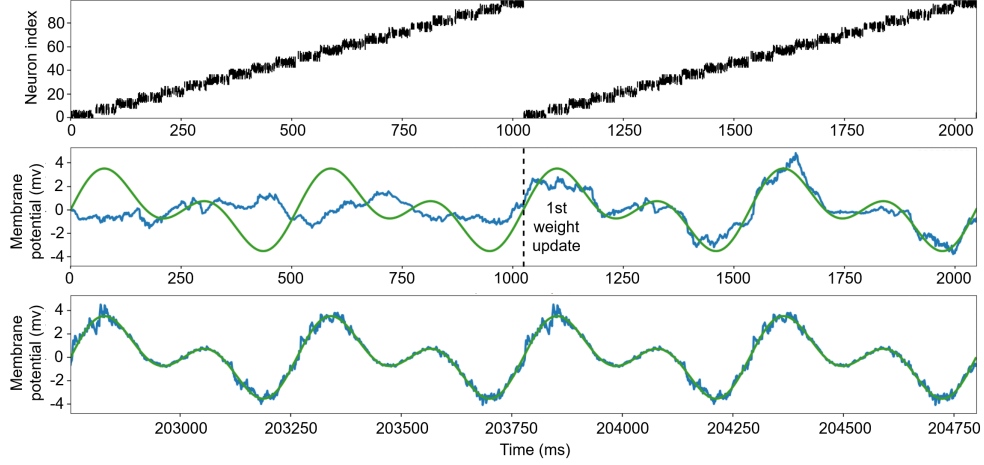


Figure 4.4: Wave-form matching task. Top: repeating staggered input spikes. Middle: initial performance before any learning and after one weight update (indicated by dashed line). Bottom: Converged performance after 200 presentations of the inputs.

to the network only for the 150ms duration of the prompt signal. The softmax of the two readout neuron membrane values is combined with the one-hot-encoded output values to produce the cross-entropy error. Weight updates are accumulated over two trials before being committed into the network weights.

Inputs are all-to-all connected to a single layer of 100 ALIF neurons which are in turn all-to-all connected to the readout neurons. All weights are initialised from a normal distribution and divided by the square-root of the population size. Xavier initialisation [48] is used to ensure the network is started in a state in which spikes can flow through the network. This results in weights being normally distributed with  $\mu = 0.55$  and  $\sigma = 1$ . Without this the pseudo derivative  $\psi$ , remains too small to drive learning without firing rate regularisation. A learning rate of 0.3 is used and an ALIF threshold time constant of  $\tau_a = 6500$  ms and adaptive threshold scale factor  $\beta = 10$  mV.

## 4.5 Results

### 4.5.1 Wave-form matching

This task investigates the power of the readout neuron to sample from its inputs to produce a desired continuous output. Figure 4.4 shows the final performance of the network after training. It can be seen that the membrane voltage of the readout neuron (in blue) matches the target wave with a high level of agreement. Even after the

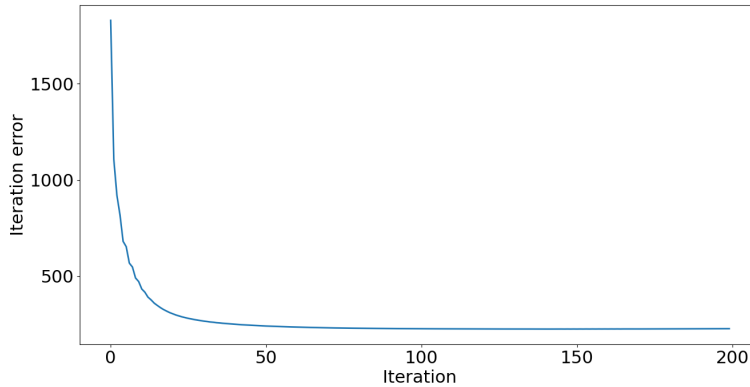


Figure 4.5: Error for each presentation of a target wave-form as seen in the experiment of Fig. 4.4. Each presentation lasts 1024 timesteps with a timestep size of 1ms, error is accumulated over the test to give the iteration error. Performance rapidly improves with convergence at around the 65th presentation.

first weight update the output captures the general form of the target wave with the remainder of the learning process fine tuning the response. The largest disagreement is present at the highest and lowest value of the wave-form. The presence of error at this part of the target signal is likely a result of the high curvature of this section of the wave-form. The quick change from one direction to another proves a challenge for the algorithm to match smoothly with spiking input and leaky-integrator readout dynamics.

The online functionality of this algorithm was able to reach convergence, requiring around 65 presentations of the target-wave form to create a close match, see Figure 4.5. This is approximately 65 seconds of real-world compute time to incorporate enough information about the task to achieve agreement between a target and the actual output. This displays the ability of e-prop to combine local traces of network activity with an instantaneous error to match a global error signal. The online functionality does appear to add some noise to the learning process, as past a certain accuracy the network remains in a state of oscillation with weight updates incrementally moving up and down struggling to match the finer details of the task. It is likely this could be alleviated with the addition of a decay in the learning rate. By slowly decreasing the learning rate, the weight updates would tend towards zero and performance would stabilise. Additional experiments were also run with a hidden layer between the inputs and outputs, however due to the instability of the learning process the hidden layer struggled to converge on a static firing pattern. This created an irregular input for the readout to sample from

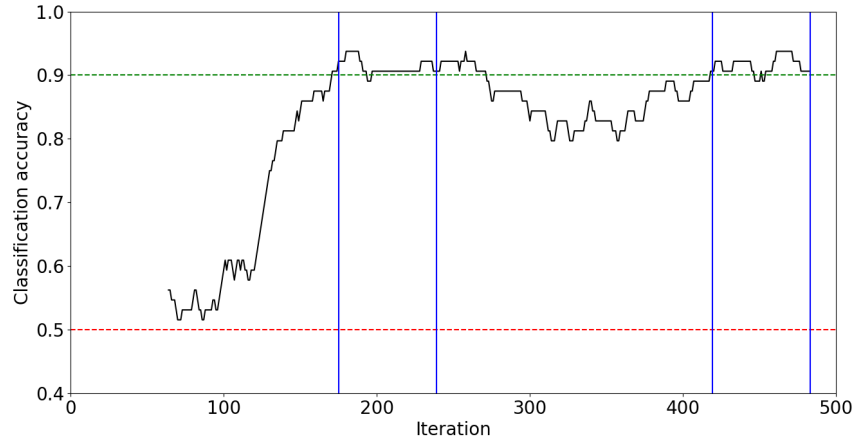


Figure 4.6: Performance of e-prop on the left-right task. The black line shows the running average accuracy over the last 64 tests with the required threshold performance shown in the horizontal dashed green line. The dashed red line displays performance for random action selection. The vertical blue line displays the trial at which the network achieved over 90% accuracy over 64 tests and the number of cues is increased. The first blue line shows completion of the 1 cue task and the last the 7 cue task, network parameters are retained between transitions.

hampering final convergence.

### 4.5.2 Temporal credit assignment

This task highlights the power of e-prop to use instantaneous approximations of previous network activity to solve a task which requires memory of previous states. The main component of memory within the network is the ALIF neuron. As described in Sec. 4.4.2 the ALIF neuron's threshold increases after emitting a spike, decaying back to resting threshold as governed by the time constant  $\tau_a$ . This provides the neuron with a memory of previous input, which can be combined with the trace of previous neuronal activity and a global error signal to update the synaptic weights in response to inputs received previously in time. The memory of previous activity enables the on-line learning to incorporate more than the instantaneous information available locally and solve the temporal credit assignment problem. Although the learning is online, the simulation had to be run at eight times slow-down to ensure the neuromorphic fabric could handle all spikes at each timestep whilst updating all environment and neuron variables.

As in Sec. 4.4.4, the test begins in its simplest form in which only 1 cue is presented. The incremental learning approach breaks down the task enabling quick acquisition of appropriate mapping between inputs and outputs, which can later be fine tuned as the task becomes more difficult. Figure 4.6 shows the network initially at random chance of correctly choosing left or right. Following this the network begins to incorporate enough errors into the weights to classify better than random (50%) chance at approximately the 100th test. This leads to a quick improvement in performance, with the target performance of 90% accuracy over the previous 64 tasks reached at the 175th trial.

Next the task difficulty is increased; the network is now trained using three cues, of which the correct choice is the input which generated the most prompts by the end of trial. This proved a small increment to the network which mostly maintained above target performance reaching the next task increment at the minimum 64 trials. The increase to five cues proved a challenge to learning with performance dropping and taking a further 180 trials to reach target performance. This is likely due to the increased separation in time between input cues and the final prompt to decide. For example, if the first three cues are for the left and the last two signal the right, the network must be able to store this information in the decaying ALIF neuron thresholds without the last two cues becoming the dominant memory retained within the network. This requires a fine balance of the input contribution to each of the ALIF neurons as an imbalance can become amplified given the decay of the memory through time, resulting in an incorrect choice at the end of the trial. The training achieved in the five cue setup was enough to enable the seven cue task to be solved immediately within 64 trials.

### 4.5.3 Firing rate (ir)regularisation

During the learning process care must be taken to keep neurons within a certain firing range. If the rate becomes too high then the entropy of the system drops and little information is carried to the outputs. If the rate becomes too low this can create ‘dead neurons’ which do not receive enough input to learn. As shown in Equation 4.13, the error,  $E_{rate}$ , is generated at the core by comparing the current rate with a target rate of 10Hz. This error is then added to the learning signal to calculate the combined weight update.

$$E_{rate} = f_{rate}(t) - f_{target} \quad (4.13)$$

Two ways were attempted to estimate the firing rate of the population. The first, shown in Equation 4.14, involves taking the number of spikes,  $n$ , and dividing by the elapsed time,  $t$ , to get the frequency via a running average.

$$f_{rate}(t) = (n(t) - n(t_0)) / (t - t_0) \quad (4.14)$$

The second method, shown in Equation 4.15, employs an exponential decay to create a running average of the firing rate. The number of spikes in the current time step is added to the running total,  $f_{rate}$ , which is decayed by  $\tau$ . As this is done every timestep with a  $ms$  timestep  $\tau = e^{-1/1000}$  creating an approximation of the spike rate in Hz.

$$f_{rate}(t) = \tau f_{rate}(t - 1) + (n(t) - n(t - 1)) \quad (4.15)$$

#### *A constant rate is not constant*

Figure 4.7 shows a comparison between the two forms of firing rate calculation when all neurons in the population are firing with a constant 10Hz Poisson rate. The top plot shows how the rate calculation looks across 8 neurons, as in this implementation on a single SpiNNaker core. The bottom plot displays how the rate calculation looks averaged over the whole population of 256 neurons, as in the original TensorFlow implementation.

It can be seen that even with a constant rate across all neurons, 8 neurons is insufficient to get a smooth read out of rate using the exponential decay. After around 2000ms enough time has elapsed for the running average to be relatively accurate for 8 neurons, although this is a long time relative to input stimulus duration.

When looking at the 256 neuron case of Figure 4.7, the running average stabilises after around 300ms. The exponential estimate is significantly more stable with the only issue being the requirement to move from the initialisation value. This shows how even in perfect conditions where neurons are firing regularly the firing rate estimate is far less noisy when averaging over a larger population. A larger population average error would also generate more distributed firing across the population with some subgroups firing more for some stimuli than others, as opposed to many needing to fire as would be encouraged with smaller population averages.

#### *Bursting the estimation*

Figure 4.8 shows a comparison between the two forms of firing rate calculation when all neurons in the population are firing with a bursting Poisson rate of 10Hz Poisson condensed from 2048ms into 12.85ms, as an extreme example of bursting behaviour,

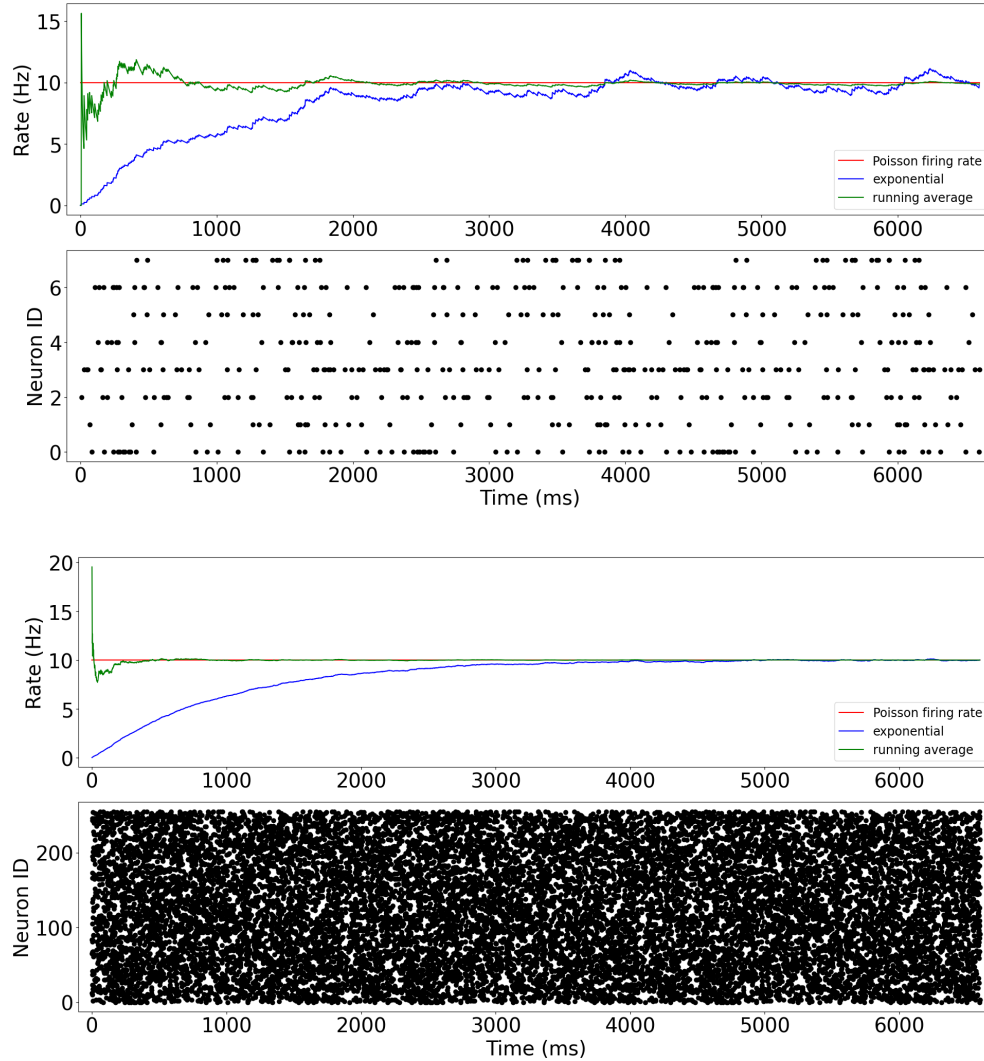


Figure 4.7: A comparison of firing rate estimation with a constant rate across 8 neurons (top plot) and 256 neurons (bottom plot) using exponential decay (blue) and a running average (green). The average Poisson firing rate is shown with the red line and the actual spike times for each neuron are represented by the black dots in the bottom half of each plot.

similar to the bursting nature of the mouse experiment's inputs. The top plot shows how the rate calculation would look on a SpiNNaker core with 8 neurons, as in the implementation. The bottom plot displays how the calculation would look if you were to average over the whole population of 256 neurons, as in the original TensorFlow implementation. A more pronounced difference can be seen between the firing rate estimates of the two population sizes. The population of 256 neurons allows sub-groups of 8 neurons to fire at a much increased rate with the population size creating a smoothing in the rate estimation.

The top plot of Figure 4.8 shows what a single core would see when the whole population fired as in the bottom plot of Figure 4.8. The bursting nature of the input makes exponential decay estimation incredibly inaccurate with only a brief time when the rate is correct. The running average is correct shortly before the subsequent burst with the rate estimate then increasing. As time progresses the increased estimate following a burst diminishes.

The original TensorFlow implementation of e-prop was able to estimate the firing rate over the whole population. As shown in the two contrasting firing examples of Figure 4.7 and Figure 4.8 this leads to a very clean firing rate estimate relative to what is possible with 8 neurons. Appropriate firing regularisation is of particular importance when training recurrent networks. Their looping connections create a lot of potential feedback mechanisms within the network, which need to be carefully balanced to ensure the network is neither firing constantly or quiescent.

#### *Regularity in practice*

When applying mechanisms of firing rate regularisation in the SpiNNaker implementation, the noise in the firing rate calculation would often drive the spiking activity towards low entropy. This may mean no activity as weights were pushed too low and many neurons become 'dead' or, as was more often the case, towards habitual firing of the whole population in synchrony. Although the firing rate may have been close to the target 10Hz, in this last case the small population sizes mean there was little variation across neurons and any individual responses to different inputs was lost, significantly affecting task performance. The error generated from the firing rate regularisation often dominated the weight update reducing variance to inputs. There is also the case when firing rate regularisation would drive the network towards a state when the firing rate would be high enough that packets would be dropped. Eventually, the router would not be able to handle routing that many packets, eventually causing the simulation to crash and be lost.

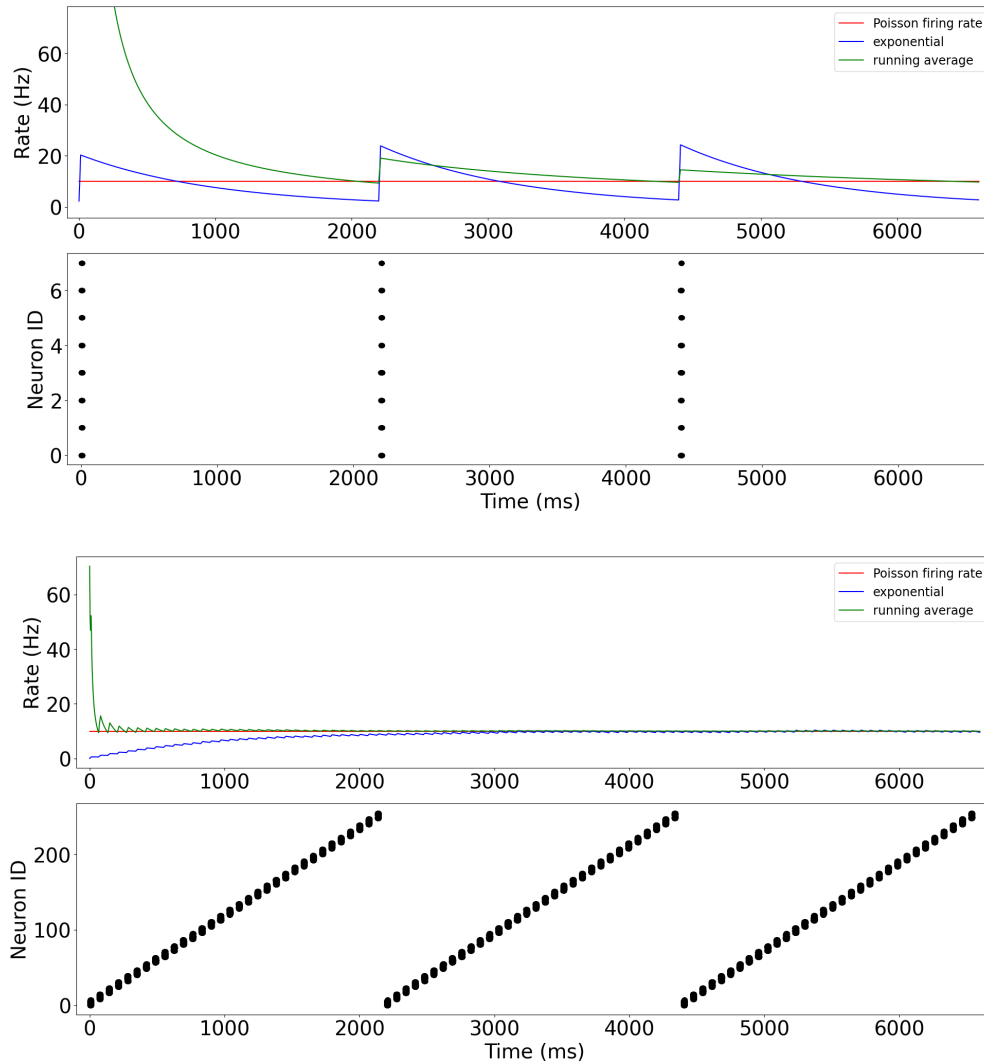


Figure 4.8: A comparison of firing rate estimation with bursting behaviour across 8 neurons (top plot) and 256 neurons (bottom plot) using exponential decay (blue) and a running average (green). The average Poisson firing rate is shown with the red line. In this instance the 10Hz Poisson firing of 2048ms is condensed into 12.85ms to give an extreme example of bursting behaviour.



## 4.6 Discussion

This work demonstrates the applicability of the e-prop learning algorithm [9] to neuromorphic hardware through a functioning implementation on SpiNNaker. Training was carried out in an online fashion with locally stored parameters capturing enough information to enable the matching of a target wave-form and the processing of temporal information. This displays the potential for neuromorphic implementation of learning algorithms in a number of domains. This could allow local tailoring of neural networks on personal devices in an energy efficient manner. Lower power consumption would lend itself well to mobile devices with on device learning increasing the user experience through individualised data processing.

### *Controlling the fire element*

A key challenge in the development of this work was ensuring the network remained in a state in which the pseudo derivative was non-zero. If the weights of the network lead to a neuron not firing this could easily become permanent, resulting in a ‘dead neuron’ that could no longer contribute to network performance. Firing rate regularisation can aid in alleviating this, although it proved difficult to create a stable response using only locally available information. The original e-prop work achieved stable firing rate estimation by averaging over the entire population [8], enabling some neurons to fire a lot and some very little. However, with only local information available in the neuromorphic implementation the neurons can often form a homogeneous firing pattern reducing the information content of the spikes. This was especially true for the temporal credit assignment task, in which the input spike frequency had a large variability in time with high frequency inputs followed by large periods of quiescence. Subsequently this makes the task of sampling from the hidden layer spikes significantly more difficult for the readout neuron.

### *A neuromorphic future*

Implementation of the e-prop algorithm on SpiNNaker demonstrates its suitability for deployment on neuromorphic hardware. Management of individual synaptic traces per synapse was achieved by moving the synaptic matrix into core local memory, and computing updates within the periodic neuron state update. While this led to increased use of core memory and longer state update times, the implementation was able to successfully complete the presented tasks with online learning in real-time for the wave-form matching task, with an effective upper limit of 256 e-prop synapses per neuron. More complex problems such as the temporal credit assignment task challenged these limitations, and may require an alternative mapping to SpiNNaker hardware to improve

performance. This could include parallelisation of neural processing [115], and additional measures such as use of sparse connections to reduce the number of e-prop synapses. The fixed-point arithmetic utilised on SpiNNaker was found to be sufficient for the given tasks, however the lack of an FPU causes the divide operation to be performed in software impacting performance. More complex tasks requiring lower learning rates for training stability may benefit from additional scaling of variables, e.g. re-defining values between 0 and 1 as long fract type, or scaling the weight representation to enable low-end precision. The online batch training mode used here could also be explored further, to enable accumulation of significant weight changes before committing them to synaptic weights. However, overall this work highlights the potential of bio-inspired learning rules implemented on neuromorphic hardware.

#### *The work continues*

There are a number of potential directions for further adaptation of this work. Developing more stable regularisation techniques, which can be applied using only locally available information will allow the network to operate in a higher entropy state, increasing the efficacy of the learning as a whole. In this work the power of e-prop to train recurrently connected networks was not fully explored, in part due to the challenges of firing rate regularisation to create appropriate firing in the hidden layer. If regularisation can be developed to tame the recurrent network using local information then more complex temporal tasks, such as sound and video processing, could be trained online.

Bringing advancements from future implementations of e-prop [11] in which a network is trained to play Pong, the work in this chapter could be extended to reinforcement learning scenarios. Although requiring a large amount of training time for more complex environments such as Atari games, the architecture explored in this work could, in theory, be applied to reinforcement learning tasks. This would enable online real-time learning on neuromorphic systems to be applied to a wider range of applications with sparse and complex error signals.

#### *The wider context*

A biologically inspired learning rule has been explored within the brain-like constraints of the neuromorphic platform SpiNNaker. By operating using only locally available information and constrained with information transmission in similar ways to the biological brain, GD-like learning is shown to be possible. However, it is still only a part of the puzzle of learning. There are many hyper parameters and topological choices

which must be made when applying a learning algorithm to a new task. Expert knowledge and trial and error are often required to determine sufficient settings. Biology has performed this trial and error process through evolution reaching a complex and adaptable architecture capable of learning a plethora of different tasks. Machine learning research can continue to try and identify individual elements, such as useful transfer functions and architectures, although it will only be when we can start putting them together in meaningful ways that complex cognition will emerge.

Matching the creativity and ingenuity of evolutionary search in creating efficient solutions to complex problems remains a difficult challenge. The field of learning-to-learn attempts to bridge this gap, at least in terms of hyperparameter selection and initialisation, by putting an outer-loop optimiser around an inner-loop learning algorithm [8, 14, 145]. This allows qualities of the inner-loop optimiser to be more finely tuned to a particular task enabling faster learning and more efficient operation. The question still remains, what parameters are chosen for the outer-loop optimiser, but learning-to-learn lends more adaptability to the implementation of the fast timescale learning process and potentially paves the way for future algorithmic complexity without requiring expert knowledge. Research has also gone into directly feeding back information from the inner-loop to the outer-loop [39], displaying a potential bridge between the two scales.

There will always be the inherent need for data to drive the learning process but if more efficient learners can be created then individual networks may not always need to start from scratch. If initialisation can move beyond random and begin at starting points with broad applicability then knowledge can be begin to be transferred across tasks and fast learning becomes a simpler task. Slow learning gave us the biological brain but machine learning continues to leave its focus on the fast learning and fine-tuning processes starting from random initialisation.

# Chapter 5

## Error driven neurogenesis

### 5.1 Introduction and background

The brain possesses an impressive ability to acquire information and is able to readily apply it without a disconnect between learning and acting. From a young age humans can be presented novel stimuli with associated labels and is immediately able to recall and manipulate these concepts. This is in part a consequence of the learning that has happened before (both in the life time of the human and on an evolutionary timescale) to extract general features, but it is also a consequence of the state in which information is stored in the brain. With time, through sleep and further training, memories can be consolidated and kept in a more general form for future application [132].

#### *New neurons equal new information*

One possible mechanism by which information can be stored in the brain is via neurogenesis, a process which continues throughout a lifetime [129]. Research has suggested that adult hippocampal neurogenesis plays a roll in learning and memory [32]. Newborn neurons *de novo* grow axons and dendrites forming both efferent (outgoing) and afferent (incoming) synapses enabling topological adaptation. The integration of neurogenesis in the hippocampus suggests an important role in learning and memory throughout a lifetime, which is missed from the majority of machine learning approaches.

#### *Gradient descent is (not so) secretly statistics*

Traditional ANN (Artificial Neural Network) training techniques, such as gradient descent, skip the initial acquisition of memories and their functional storage and jump straight to building a generalised representation. These algorithms are examples of

parametric models that allow an input-output mapping to be compressed into the parameters of a network. Owing to the universal function approximation properties of feedforward artificial neuronal networks with sigmoid activation [25], in theory, a statistical approximation of any continuous mapping between any two Euclidean spaces can be captured in the parameters of an ANN. This has also been extended to show the universal function approximation of neural networks in other cases such as non-Euclidean spaces [73], RBFs (Radial Basis Functions) [104] and CNNs (Convolutional Neural Networks) [154]. Learning usually begins with the weights of a predefined architecture being randomly initialised, the gradient of an error with respect to the weights is estimated and used to gradually move the network towards a position in the parameter space with reduced error. The random initialisation can also have a bearing on the final optimum that will be converged upon due to a sensitivity to initial conditions. This is an incremental approach and requires averaging over many samples before a useful model can be elucidated.

GD (Gradient Descent) uses data to build a statistical summary and incorporates it in the weights of the ANN. This leads to a condensed representation within the network and a generalised solution, assuming suitable architecture and hyperparameters. This condensed statistical representation leads to the black box nature of ANNs; their behaviour cannot easily be investigated without evaluation using a test set. This can be problematic in cases with limited data where situations that must be accounted for are not part of the training data, such as different weather and lighting conditions for an autonomous car. It is also of importance in understanding what particular features are being selected to produce an output, for example in medical diagnosis.

The general function of gradient descent algorithms applied to ANNs, such as BP (Back-Propagation), requires that the network be paused to perform an update. This is in part because of gradient descent often utilising batch updates to smooth weight updates across multiple examples. There is also the computational demand of calculating gradients for all weights within the network and combining them with the produced error, which can prove challenging during operation. Online learning could allow robots to explore an environment and adapt their behaviour continuously, such as changing their gait to adapt to unexpected terrain or select new behaviour after damage. Solutions exist to this such as the multi-compartment model of dendritic microcircuits [120] in which activity is simultaneously passed forwards and backwards to enable online updating of connection weights in a biologically plausible way.

*Neural networks are more than neurons*

Recent research has shown that the branching arms, dendrites, of L2/L3 pyramidal neurons possess non-linear activation functions capable of solving the XOR task [47]. They are not active with low level stimulus before a threshold level at which they begin producing activity. After a point, as input stimulus increases their activity decreases. This suggests some tuning of dendritic activity to a particular level of input activation. If that tuning can be adjusted to the level of inputs then the dendrites can be argued to have stored memory of those activations. This is a contrast to standard ANN connections which perform only linear transforms of inputs. There is evidence of dendritic dynamics and maintenance being involved in long term memory formation and cognition [67, 136], suggesting there is still far more untapped computational power within the biological brain yet to be harnessed by modern artificial intelligence.

#### *Remembering beyond synaptic weights*

Memory within neural networks often takes the form of recurrent circuits like LSTM (Long Short Term Memory) units [8, 52, 110, 135]. They have shown great performance applied to time series data such as video and speech processing [59, 121]. They allow the storing of information in a way that can be trained via gradient descent which is ideal for traditional learning approaches in ANNs. External memory units have also been used in conjunction with neural networks [51, 146]. A powerful example is the neural Turing machine [50] which possesses memory locations which can be used to store and retrieve data. Their design allows training via gradient descent, as with LSTMs. However, due to their reliance on gradient based techniques they require large amounts of compute time to form and manipulate useful memories of the system. Deep neural networks have been augmented with episodic memory units in an attempt to solve the data hungriness of model-based neural network approaches. This is often in the form of a lookup table whose stored value is used in conjunction with a neural network trained with gradient descent [79, 82, 84]. The addition of an episodic memory buffer enables quick acquisition of beneficial behaviours which can be exploited. This is exemplified by Blundell et al. in which a purely table based approach to episodic memory is used to solve Markov Decision Processes [13]. The approach boasted fast learning capabilities as a result of being able to quickly exploit highly rewarding states. However, the episodic approach may be overtaken by parametric function approximators, such as DQN [53], in the later stages of training.

#### *Neurogenesis is continually generative*

As mentioned earlier, neurogenesis has been suggested to play a role in learning and memory formation. However, neurogenesis has been a target of research within the

machine learning community mainly with a focus on continual learning. In [34] this is displayed by first training an auto-encoder on a reduced number of MNIST (Modified NIST) classes using gradient descent. After introducing the missing classes neurons are added to layers with a high reconstruction error and further trained with a reduced learning rate on non-new connections. This allows the network to adapt its architecture in response to the data and incorporate new information with mitigated catastrophic forgetting. Other examples of neurogenesis in literature often rely on starting first with a trained network and adding neurons via some method and further training the networks, such as in Mixter and Akoglu [92] where neurons were added to a seed network using neuron activity as a synapse selection metric. The algorithm grew the network displaying a reduced parameter size compared to standard approaches or methods employing pruning techniques with comparable MNIST performance. Martin and Pilly [87] also start with a pre-trained network applied to MNIST. Following initial training a perturbed version of MNIST is presented and a genetic algorithm used to determine where new neurons should be added before being trained with stochastic variational inference to set the new weights. Parisi et al. [103] apply neurogenesis to self-organising maps with new neurons added to a pre-trained feature extractor layer and later trained with a Hebbian learning rule. Abolfazli Esfahani et al. [2] use a seed network not trained on the desired task and instead took the first feature extraction layer of GoogLeNet trained on Places205 [153], a place recognition data set. By utilising the robust feature detectors of the first convolutional layer, particle swarm optimisation was applied to control neurogenesis to create a corner detector.

There is very limited work on neurogenesis that does not rely on a seed network, the best example comes from [133] in the paper *Lifelong Learning Starting From Zero*. They begin with an empty network and add neurons in response to errors, unlike the similar work of [38] in which neurons are added for unrepresented states. Value nodes connected to neurons use Gaussian transfer functions to enable downstream neurons to be maximally responsive to particular inputs. The weights, biases and Gaussian parameters are updated via backpropagation and gradient descent. Quick learning capabilities are displayed in contrast to networks trained with only gradient descent. Brains could achieve this network growth through neuron recruitment as well as generation.

#### *Error driven neurogenesis*

The algorithm explored in this chapter, EDN (Error Driven Neurogenesis), leverages neurogenesis to enable online one-shot learning in a range of applications. It displays a rapid acquisition of new information, without the need for gradient calculation, in

a biologically inspired way by modelling forms of neurogenesis and dendritic non-linearities. Starting from an empty network imbues it with no initialisation bias and it is therefore only driven by the inputs it is presented and the context they are put in by the current performance of the network and how that effects the error generated. Incrementally a non-spiking connectionist model is grown in response to errors experienced during training. The rest of the chapter is organised as follows: in Section 5.2 an overview will be given of EDN’s design before discussing how it is applied to different domains in Section 5.2.2 and then contrasting it with similar algorithms in Section 5.2.3. Following this results are displayed in Section 5.3 and discussed in Section 5.4.

## 5.2 Implementation and experimental design

The current focus of machine learning research is often on the fine tuning of ANN parameters. An architecture is predetermined using expert knowledge to best suit a specific task and is gradually fed data to create a statistical model. This limits the general applicability of the network as a different topology may be required in a different domain and applying it to a new task can result in catastrophic forgetting, significantly hurting performance on the previously learnt task. It also does not enable the network to fine tune its structure in response to unexpected limitations, such as requiring more layers or more neurons to extract different features. Neurogenesis offers a potential solution to these problems by incorporating new neurons into a network. This can be done in a way that does not effect the previously learned information whilst allowing incorporation of new data. The algorithm explored in this work relies on three key principles: 1) errors drive learning, 2) synapses can be used to store information and 3) neurogenesis facilitates information acquisition. When put in a learning scenario this combination leads to a modular network that grows in response to errors and stores information to mitigate these errors in its synapses. Code is available at <https://github.com/adamgoodtime/neurogenesis> which includes both the code used for EDN and GD comparison.

### 5.2.1 Error Driven Neurogenesis algorithm design

An overview of EDN’s operation can be seen in Figure 5.1a. There are three key elements to the EDN algorithm: error generation, neurogenesis and synaptic storage.



They are used in combination to create a non-spiking neural network which grows as data is presented to it. First an input is presented to the network which in turn produces an output. The output is used to generate an error signal which is compared with a threshold to determine whether neurogenesis is triggered. Following the triggering of neurogenesis input values are selected to be stored, creating a neuron that is maximally active at the presentation of the same input. This newborn neuron is then connected to the corresponding outputs as determined by the generated error, e.g. when classifying digits, guessing a 3 when the class was 8 would connect negatively to the 3 and positively to the 8. This has now added a neuron to the network that upon seeing the same input of an 8 will inhibit the 3 and excite the 8. This process continues incrementally adding more information to the network each time the error is above threshold. The following subsections will first explain how the network processes information and how that determines its representation. Next neurogenesis will be explained and how it makes use of the synapses to store information and alleviate errors.

### Network activation

The EDN network is composed of two elements, the neurons and the synapses. In contrast to traditional ANNs, the synapses contain the non-linearities, not the neurons. The synapse activation is a triangle kernel centred around a value  $v$  which is sampled from the inputs when neurogenesis is triggered and stored as a synapse parameter. The width of this kernel is controlled by the spread  $s$ . Equation 5.1 shows how a synapse input,  $x$ , is transformed with the triangle kernel,  $k$ , to produce the synapse activation. This puts maximum synapse activation when the input  $x$  is the same as  $v$  with activation dropping the larger the separation until a difference of  $s$  at which it becomes zero. This is similar to the non-linear properties of dendrites discussed in [47] shown to be able to solve the XOR problem.

$$k(x) = \max \left( 0, 1 - \frac{|(v-x)|}{s} \right) \quad (5.1)$$

$$a_n = \frac{1}{N} \sum_i^N k(x_i) \quad (5.2)$$

Hidden neuron activation,  $a_n$ , is governed by Equation 5.2. The  $N$  inputs,  $x_i$ , are passed through the triangle kernel,  $k$ , to produce the synapse activations. The synapse activations are averaged to produce the neuron activation without any weight term. This makes neuron activation a measure of similarity between the synapse parameter

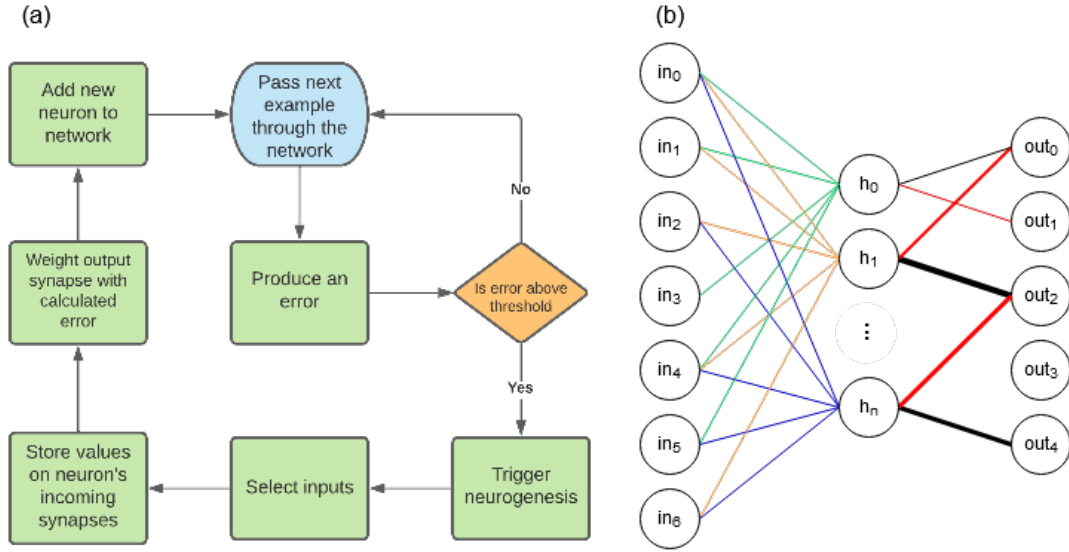


Figure 5.1: (a) the general structure of the EDN algorithm. First an input is presented which produces an associated error. If the magnitude of the error is above the error threshold then neurogenesis is triggered. Input synapses of the newborn neuron are selected and the values of the current inputs set the centre of the kernel function on the synapses. An output synapse connects the neuron to outputs whose error was above threshold with a weight proportional to the error produced. (b) an example topology created with EDN. The different colours connected to the hidden neurons signify different sub-sampled input vectors, which have been stored on the synapses; they all contribute equally to neuron activity. The colour and thickness of connections between the hidden layer and the outputs displays that output connections are weighted and the magnitude can be negative or positive. The weight of these connections is determined during the training process by the error produced when neurogenesis is triggered.

$v$  (taken from previously stored input values) and the current inputs  $x$ ; there is no weighting parameter, just a measure of similarity. The measure of similarity is  $0 - 1$  because of the activation limits imposed by the function  $k$  and the averaging of the inputs. Synapses from the hidden neurons to the outputs also possess the non-linear activation shown in Equation 5.1 with  $v = 1$  and the same value of  $s$  as the rest of the network. This acts to threshold neuron contribution to output values to only the most similar detected features.

$$a_y = \sum_i^N w_i k(a_{ni}) \quad (5.3)$$

Output activation,  $a_y$ , is controlled by Equation 5.3. Output values are calculated in much the same way as neuron activation with the main difference being the addition of a weight term  $w$  which is a function of the error produced during neuron creation (discussed in Section 5.2.2 for each task domain). The weight is set during neurogenesis and modulates the contribution of each neuron, and therefore feature, to the output values. Also, activation is now the sum of the inputs rather than the average. As many hidden neurons can be connected to the outputs with only a fraction of them being active, the sum provides more information about relative difference between output activity.

A graphical 2D example of a single neuronal unit's activation (composed of synapses connecting inputs to the neuron and the neuron to an output) is shown in Figure 5.2. The EDN neuron's synapse activation can be seen to be highest around the values  $v_0$  and  $v_1$  and decreasing further from those stored values. As the neuron activation also passes through a synapse possessing a triangle kernel, the neuron activation is thresholded creating the purple diamond activation at the centre of the two synapses' peak activation,  $(v_0, v_1)$ , which is narrower than the synapse activation.

### Network representation

A key difference in EDN compared to traditional ANN approaches is the way information is stored within a network. Neurons in a standard ANN take a linearly weighted sum of inputs and pass it through a non-linear activation function. This is analogous to drawing a hyperplane through  $n$ -dimensional space and having the neuron's activation relative to the distance from this plane, see Figure 5.2. By combining multiple neurons

in multiple layers a complex high dimensional boundary can be created which determines a certain output for any input. The training of such a network typically requires the gradual acquisition of data to build a statistical summary of input-output mappings into the functionality of the network architecture.

Instead of neuron activity being relative to the distance from a hyperplane, the activity of neurons in EDN are relative to distance from a data point. This removes the need for averaging over multiple instances as a single point can already say with confidence that the area around it is likely to share the same property. This is exemplified by Figure 5.2 where on the left a traditional ANN neuron with ReLU activation attempts to classify a single data point but without other data it becomes hard to say with any confidence where the boundary should be placed. On the right an ANN neuron's activation is shown with the neuron activation being limited to the area around the stored data point. The ReLU neuron will remain active infinitely far from the hyperplane. For this to be beneficial, generalisation over many data points is required to create an appropriate model of the input-output mapping.

EDN neurons allow the instant acquisition of information and an adaptation of the network's model without disruption to previously learned information. The parameters retained within the model are never updated; information is only added, never altered. In parametric versions of ANNs, care must be taken when incorporating new information as error is transferred to the parameters (connection weights) of the network and can effectively overwrite previously learned knowledge. This is termed catastrophic forgetting and can limit the continual learning capabilities of ANNs as result of them being parametric models. Updating the model to include more information requires updating parameters, which is where previous information is stored. The knowledge is being condensed into a limited number of parameters, therefore, changing any can alter the representation as a whole. Training a parametric model requires gradual and repeated tuning of parameters to ensure the model represents the data as a whole.

### **Synapse creation**

When neurogenesis is triggered the current input values are retrieved to become the  $v$  parameter for incoming synapses of the newborn neuron. They form the input synapses of the new neuron creating a neuron which is maximally active when presented with the same inputs as were just stored because each synapses triangle kernel is centred on  $v$ . Subsets of inputs can be selected to be the incoming synapses of the neuron.

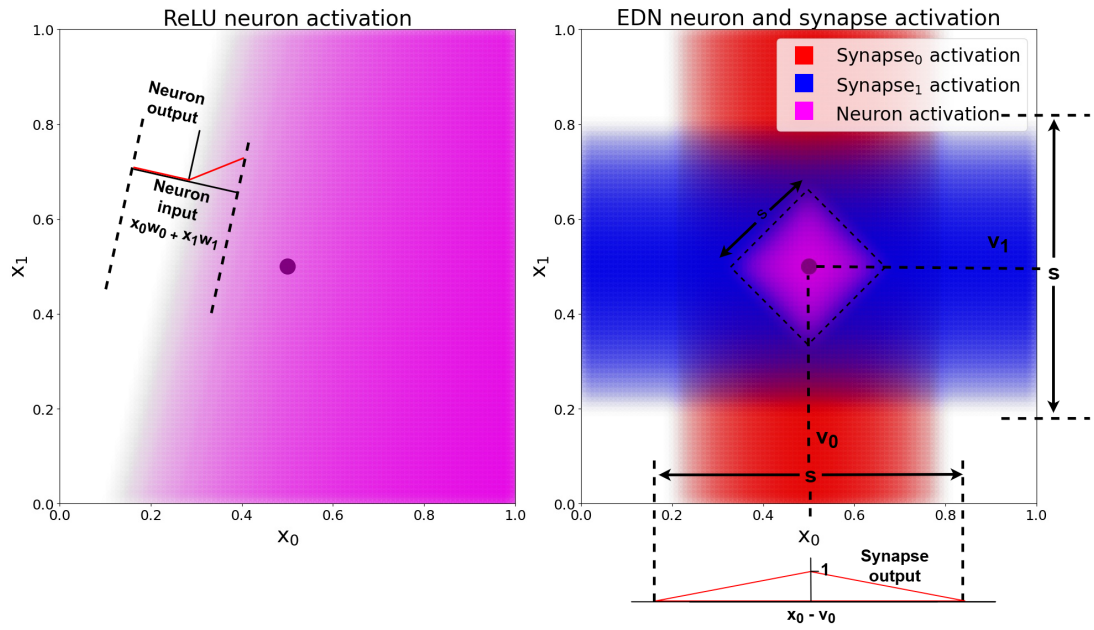


Figure 5.2: A comparison of a standard ANN neuron with ReLU activation and an EDN neuron's activation. The black dot is a single data point, the pink shaded area shows the output activation of the neuron in both cases, with a deeper colour representing a higher level of activation. The red and blue shaded areas represent the level of activation of each synapse, which possess the triangle kernel shown in the bottom right, with spread  $s$  and centred at individual values of  $v$ . The EDN neuron's activity is the average of the incoming synapse activity and is also passed along an outgoing synapse using a triangle kernel with the same  $s$  value and  $v = 1$ . This output synapse acts to threshold neuron activity and create the bounded purple area in the input space in which the neuron is active.

This moves the neuronal representation from an input-output mapping to a feature-output mapping. Output synapses of all hidden neurons have a uniform  $v$  equal to 1, in contrast to the input synapses where  $v$  is set by the values of the inputs. Because the synapses still possess a triangle kernel this acts to threshold and accentuate output contributions of neurons which are receiving the most similar inputs to their stored values. Neuron contributions to outputs are further modulated by error, discussed next.

### Error integration

The algorithm uses neurogenesis to mitigate errors produced by the network. The weight of the synapse connecting the new neuron to the outputs is opposite sign of the error produced. Neurogenesis is only triggered if the magnitude of the error is above the error threshold  $E_{th}$ . The way in which error is produced and integrated into the network is task specific and will be elaborated in Section 5.2.2. A general way to view it is to compare it to how weights are updated in gradient descent. The sign of the gradient indicates which direction the weight should be altered, larger or smaller magnitude, to reduce the error. You take a step in the direction of the gradient to move the weight in the opposite direction of the gradient of the error with respect to the weight. This update can be broken into two components: a magnitude and a direction. In EDN the magnitude is incorporated into the  $w$  parameter of output synapses of hidden neurons to outputs. The direction is governed by the inputs stored in the individual input synapses'  $v$  parameter. This in essence makes each neuron an individual update to overall network performance.

A comparison of learning dynamics between gradient descent and EDN can be seen in Figure 5.3. It is a 2D demonstrative example composed of 3 classes: red, green, and blue, with all points being the training data. Decision boundaries are drawn for each class at each training step. At the start EDN has no output representation and therefore makes no guess about which output belongs to what point in the input space, this can be seen as the white area around the points. The network initialisation of gradient descent produces a bias at the beginning of training creating an output for all points in the input space. This is a consequence of the neuron activation that is active infinitely far from the hyperplane drawn by the incoming synapse weights, although they have no relation to the data at this stage. As training progresses the class boundaries drawn by gradient descent shift and begin to match the data. It is also seen that EDN's storage of a few data points enables the classification of many other data points immediately. This trend continues with gradient descent gradually matching the input data until a

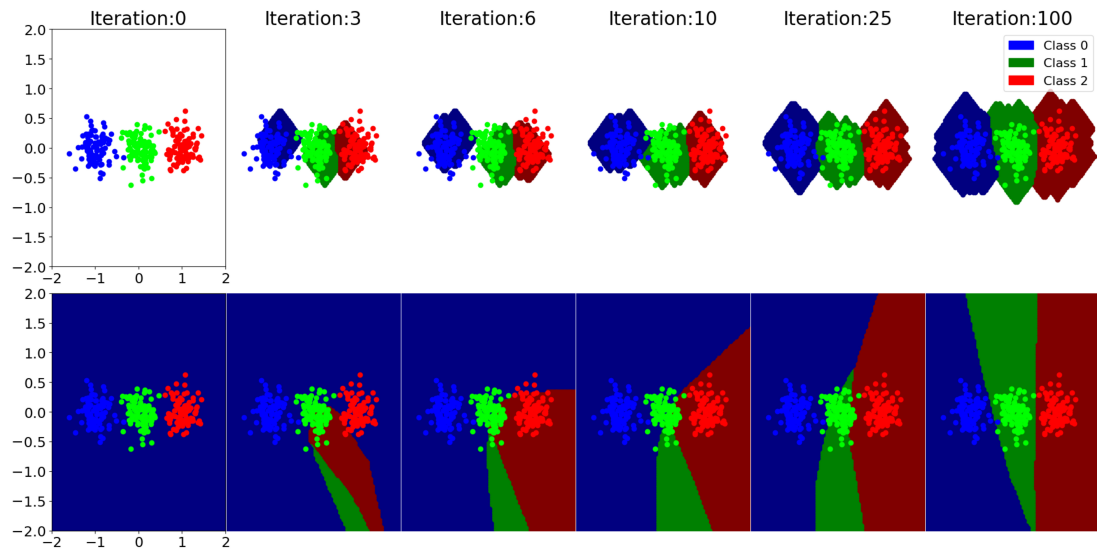


Figure 5.3: A selection of training steps of EDN (top) and an ANN trained with GD (bottom) are shown for a toy classification example. The data set is composed of three classes: blue, green and red. Class boundaries are drawn in a darker shade than the data points showing what output each model would associate to that point in space. The white area around the EDN plots indicates that there are no output values at that point in the input space.

general approximation of data distribution is converged upon. EDN converges to a general boundary surrounding the classes, although, there remains no activity far from the data points. If subsamples of inputs were taken neuron activity could persist along selected dimensions, however, with this example all inputs are selected to be a part of the new neuron.

### The neuronal unit

The network in EDN is a collection of independent and modular neurons. Each one has a set of input synapses that are more active the closer the input activity is to their stored  $v$  value. A neuron's contribution to the outputs is proportional to the output error produced when that neuron was created. Combined, this creates a modular neuronal unit that attributes a specific output value to inputs similar to its stored  $v$  values. The more similar the input the more confident the neuron is in the expected output. En masse this creates a network that, through training, has an expectation of the correct output for all points within a certain distance of the saved features.

The storage of information on the synapses allows neurons to be investigated and input-output mappings to be extracted from the network. The values of  $v$  on the input

synapses store the inputs and the output synapses indicate which outputs those inputs correspond to. When subsets of inputs are selected, this enables feature-output mappings to be extracted. Parametric models do not possess this ability as the knowledge is condensed into a set number of parameters. Evaluating the model requires querying it with input and observing the output. Investigating output response to inputs is important to be able to accurately examine performance, however, being able to dissect a network and determine what constituent elements compose the behaviour as a whole enables fine grained inspection and conclusions to drawn beyond the data presented to the model.

### 5.2.2 Task specific alterations

For all tests input values are normalised between 0 and 1 by subtracting the minimum and dividing by the range before training. An error threshold,  $E_{th}$ , determines the minimum absolute error that is required for neurogenesis to trigger. Only outputs with a magnitude of error greater than  $E_{th}$  will form connections with the new neuron.

#### Classification

For the classification tasks the error is generated by subtracting the estimated classification (softmax of the output values),  $y^*$ , from a one-hot encoding of the correct labels,  $y$ . This gives positive error for outputs which were too small and negative error for outputs which were too high. As shown in Equation 5.4, this error value,  $E$ , becomes the weight of the connection from a new neuron  $i$  to the respective output  $o$  if neurogenesis is triggered.

$$w_{oi} = y_o - y_o^* = E_o \quad (5.4)$$

#### Using surprise to guide input selection in classification

During classification, when neurogenesis is triggered, a record of the input values stored on input synapses is added to an expectation for the associated class,  $e_o$ , where  $e$  is the expectation of the output  $o$ . Equation 5.5 shows how the receptive field for an output is calculated by weighting the vector of values of  $v$  stored on each neuron  $i$  by their weighted contribution to the output  $o$ . The expectation is a uniform weighting,  $w_{oi} = 1$ , which creates a broader representation in the expectation, discussed in the results.





Figure 5.4: A example of how output values of 0 and 1 are combined to create a position on the regression scale. As described in Equation 5.8 the output value for the real part of the scale (red) is divided by the total of the real and inverted (blue) outputs to create the position on the regression scale, in this case  $\frac{26}{26+14} = 0.65$ . This value is then scaled to the full range of regression values possible in the task to produce the estimated regression value of the input.

$$e_o = \frac{\sum_i w_{oi} v_i}{\sum_i w_{oi}} \quad (5.5)$$

When neurogenesis is triggered, the expected output for each class,  $e_y$ , is multiplied by the activation values of their respective outputs,  $a_y$ , to create an expected output,  $e$  (see Equation 5.6). The activity modulated expectation and the input values,  $x$ , are compared for each input,  $i$ , to create the input surprise  $s$ , as shown in Equation 5.7. Inputs with a surprise greater than the surprise threshold,  $s_{th}$ , are selected to form synapses of the new neuron. When there is no expected value of an input, such as at the beginning of training, the input is selected by default.

$$e = \sum_y^{outputs} a_y e_y \quad (5.6)$$

$$s_i = \text{abs}(e_i - x_i) \quad (5.7)$$

### Encoding continuous values

Classification is a mapping from inputs to a discrete label but many possible input output mappings do not map to discrete outputs. Regression tasks involve mapping inputs to continuous values and, therefore, require a different formulation to work with EDN. Neurons within EDN attribute a specific input to a specific output error. In classification this is straightforward as neurons can be directly connected to the outputs associated with the error produced. In regression this is less simple as outputs can take any value within a continuous range. The output connection has to be able to point at a specific position on a scale. It also has to be able to point with a certain magnitude to indicate similarity with stored inputs (neuron activity) and the associated error created during neurogenesis (output weight).

These challenges are solved by splitting each regression value into two components, one for the low value on a scale and one for the high value. The scale is from 0-1 and is rescaled to fit the full range of output values possible for the specific task. A newborn neuron is connected to both the low and the high outputs. The total activity of the network's contribution to both outputs is combined to estimate the regression value. This is exemplified in Figure 5.4 where a value of 0.65 on a scale from 0-1 is encoded as 26 for the low output and 14 for the high. Equation 5.8 is used to retrieve that position on the scale, where  $y$  is the value,  $l$  is the low value magnitude and  $h$  is the high magnitude. This splitting enables the activity level of a neuron to not effect the estimated output value and instead become a measure of confidence in the value. Multiple neurons' activities can also be added together and modified by individual output weights to create a weighted average of regression estimates.

$$y = \frac{l}{l+h} \quad (5.8)$$

During operation all output values are bounded from 0-1 and used to calculate outputs as shown in Figure 5.4. The range of possible regression values is used to calculate what the minimum and maximum values should be when translating back and forth between a scale from 0-1 and actual regression values. They are scaled to the full range of regression values to calculate the mean squared error for a given input and 0-1 when represented in the network. This error,  $E$  then becomes the connection weight from a neuron to the low,  $w_l$ , and high,  $w_h$ , outputs with the ratio between the two encoding the position in the output range associated with that input. This is shown in Equation 5.9 and Equation 5.10 where  $y$  is the current regression value for that input that needs to be stored on the neuron,  $min$  is the minimum possible regression value and  $max$  is the maximum possible value. This attributes a particular input with a particular regression value and weights it by the error produced by the network. The  $min$  and  $max$  value of the data are calculated before hand to allow for an maximum dynamic range of the outputs.

$$w_l = E \frac{y - min}{max - min} \quad (5.9)$$

$$w_h = E \left( 1 - \frac{y - min}{max - min} \right) \quad (5.10)$$

### Reinforcement learning

Classification and regression are learning paradigms where a model is trained with data in which all inputs are given an output label, either discrete or continuous. Reinforcement learning scenarios do not have access to such information and must make connections between actions taken through time and their eventual reward state. In EDN actions are taken greedily with the output with the highest value used to determine the action performed. Random actions are taken when all outputs are equal. There is no correct label due to the reinforcement learning nature of the task, therefore, errors are simulated by inhibiting the last  $m$  timesteps before a negative reward. This is in an effort to reduce the chance of the action associated with a negative behaviour happening in the future. The most recent timestep is given an error of -1, then decreasing by  $\frac{1}{m}$  until the  $t - m^{th}$  timestep after which no more neurons are formed. This generates  $m$  new neurons for each failure with the neurons connected to the output chosen at that particular timestep. Equation 5.11 shows how output weights are generated. Only the  $i^{th}$  output corresponding to action  $i$  being chosen,  $a_i^t$ , at timestep  $t$  forms a connection with the new neuron, where  $t_{final}$  is the timestep when the negative reward was generated.

$$w_i = -a_i^t \left(1 - \frac{t_{final} - t}{m}\right) \quad (5.11)$$

### Neuron reinforcement and deletion

Reinforcement learning presents the challenge of associating actions with rewards. Unlike classification and regression there is not a specific input output mapping to learn via supervision. Negative reinforcement is captured in the error driven neurogenesis. In the inverted pendulum task a positive reward is given at each timestep the pole is balanced. This signal is passed into the network and used to keep track of each neuron's contribution to network performance. A neuron's individual reward value,  $R$ , is first initialised to the average of the network's reward, starting at zero for the first neuron, to allow time for new neurons to be evaluated. The equation for updating the neuron's reward,  $R$ , at each timestep can be seen in Equation 5.12 where  $r = 1$  is multiplied by the activity of the neuron,  $a_n$ , and low-pass filtered with a  $\tau$  value of 0.9999.

To remove unrewarding behaviours, and keep the best input-action mappings, neuron deletion is used in conjunction with neuron reward to prune the network of the least productive neurons. This is done when the number of neurons in the network

goes above a set limit. When a neuron needs to be deleted the neuron with the lowest accumulated reward is selected to be removed. This aids in capping network size and has the added benefit of aiding speed of convergence.

$$R(t+1) = \tau R(t) + [(1 - \tau)ra_n(t+1)] \quad (5.12)$$

### 5.2.3 Related algorithms

While EDN shares attributes with ANNs and gradient descent it also shares commonalities with other learning algorithms. The methodologies discussed below are similar in formulation to EDN and are introduced to highlight the similarities to and differences from related algorithms. They do not have the same breadth of applicability as backpropagation and are therefore not used for comparison in the results.

#### K-nearest neighbours

A similar non-parametric algorithm that leverages the data points to form the function output of the model is K-Nearest Neighbours (KNN)[41]. All the training data forms the model with the K closest training data points performing a majority vote to determine the property of an unseen point. Careful tailoring of K is needed to ensure appropriate classification [24]. EDN does not require the defining of a K with the training process selecting and weighting individual data point's contribution to the combined model output. The thresholding of output synapse activity puts a limit on how many neurons will contribute to the prediction. The synapse thresholding also alters the way in which distance between data points is measured; if two input vectors are similar in all but a small number of inputs which are very different then Euclidean distance between them can become large. In contrast the thresholded synapse activity will put a cap on the distance between variables allowing the vector as a whole to still be considered similar to stored values if a minority are very dissimilar. This puts an emphasis on the feature-output mapping rather than the input-output mapping. Only if an error is produced will this assumption be adjusted and neurogenesis triggered to alter the belief. This mechanism in combination with input subsampling puts a stronger emphasis on the features and their relation to network error rather than the data points as a whole.

### Radial basis functions

RBF networks share a similar topological design to EDN. They both have a single layer of neurons whose activity is distant-dependent from their centres to the input point with the peak at zero distance. However, EDN uses a triangle kernel on each synapse rather than an absolute distance between two points. The neuron then becomes a measure of distance between features rather than points in  $n$ -dimensional space. This emphasis enables exploration of different feature combinations and their contribution to performance and allows neurons to have broader applicability outside their local area.

When designing an RBF network, once a kernel has been decided, often Gaussian, there are 4 main parameters: number of nodes, centre of nodes, radius of the function and the weights of the RBF outputs (see [27] for a survey of RBF networks). Common training methods for determining the centres of nodes involve a clustering of the data points. Following this an iterative process of calculating the radius and weights of each radial basis function reduces some objective error and guides the network towards a local optimum. EDN avoids the need for gradient descent and tuning of neuron parameters by computing error on the fly, and combining this with the current input to adjust the network as a whole. This avoids the need for network parameter optimisation, however, EDN will end up with more nodes within the network compared to a typical RBF network.

### Kernel density estimation

Another algorithm with commonalities to EDN is that of kernel density estimation (KDE) [127]. By applying a kernel to individual samples an approximation of the data distribution can be established. This technique finds its main use within establishing a probability distribution over a geographical area. Applying a kernel to samples within a space allows inference to be made about the area surrounding the samples. This can even be extended to the time domain as in [99] where the temporal element of crime statistics is used to create an estimated crime density in Kyoto in both space and time.

The main difference between KDE and EDN is the uniformity of kernel contribution to the overall output. In KDE each data point is part of a random distribution and the objective is to combine those samples to estimate that distribution. Data points are all considered of equal importance and specific inputs are not subsampled to extract different features. With EDN the data points are not all considered uniformly and are

only added to the network if they are not currently captured by the model, as determined by the error during operation. This pivot towards an error driven distribution removes the need to store all data points in a model and also enables application to a wide variety of machine learning domains. The subsampling of inputs enables useful features to be extracted in contrast to all inputs contributing equally.

### 5.3 Results

Results are shown for the individual tasks comparing EDN with GD using the Adam (Adaptive moment estimation) optimiser. Following this a parametric analysis of EDN is given using the wine data set to explore the sensitivity and influence of different parameters. Parameter values for both EDN and GD were found via grid search for each task individually and given below. The parameters were selected for both EDN and GD based on how quickly they were able to learn to model the data in a stable manner, ideally after a single training epoch. Smaller learning rates could be used for GD, which would likely lead to better performance after hundreds of epochs but the speed of information acquisition was the main criteria for comparison.

#### Wine - classification

The UCI wine data set [35] is a standard classification benchmark comprised of 13 inputs and 3 possible output labels determining the cultivar/type of wine. There are 178 training examples with a class distribution of 59, 71 and 48 for each class respectively. A stratified K-fold cross-validation of  $K=10$  and validation test set size of 10% is used to evaluate the performance of the algorithms. A comparison of performance between EDN and a network trained with GD can be seen in Figure 5.5. After every training update networks are evaluated on the entirety of the test set and the average across all folds is displayed. The EDN parameters used were:  $s = 0.4$ ,  $E_{th} = 0.1$ ,  $s_{th} = 0.05$ . GD used a learning rate of 0.03 with Adam optimisation, a batch size of 8 and a network with a single layer of 200 neurons, a network of similar size to the final number of neurons created in EDN training was selected for comparison.

The bias created by the initialisation of the ANN can be seen in the performance starting at an average testing accuracy of 39.9%. This roughly mirrors the testing accuracy of random output selection given the class distribution of this task. In contrast, EDN begins with an empty network and, therefore, starts with no initial bias. This results in the network not being able to make any initial guesses and beginning with

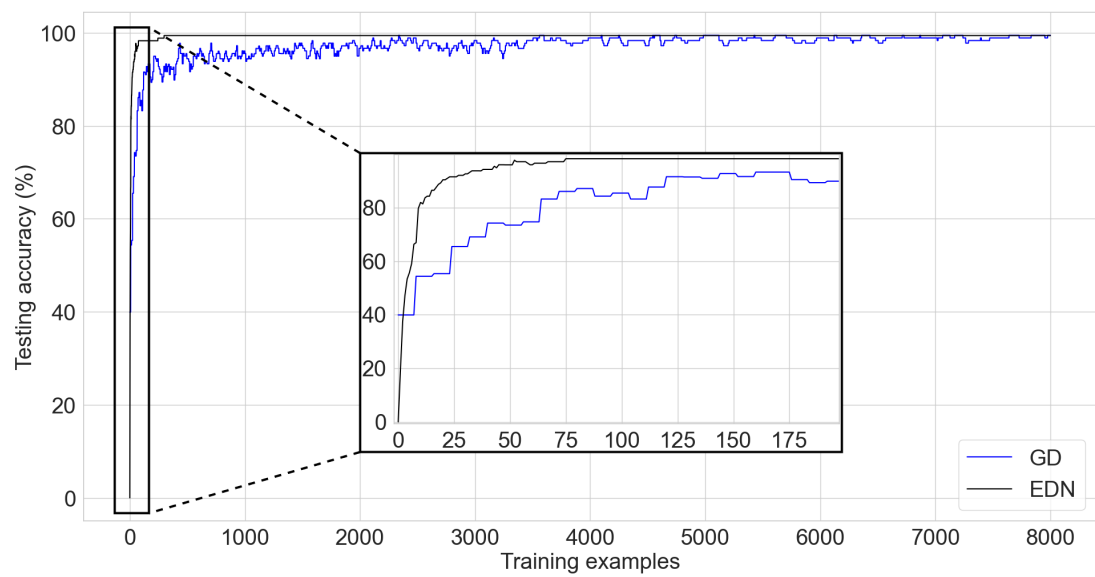


Figure 5.5: A comparison of testing accuracy during training on the wine cultivar classification task of EDN (black) and an ANN trained via gradient descent using Adam optimisation (blue). An inset is shown for the first few iterations of training to display the initial emptiness of the EDN network producing no output and the initialisation bias of the ANN network already achieving testing accuracy equivalent to random choice. The fast acquisition of information in EDN allows it to overtake the testing accuracy of GD before the first batch update is done at the 8th training instance.

a testing accuracy of 0%. The training process adds neurons to the network, rapidly increasing the testing accuracy and surpassing the performance of GD after 4 samples, this is before the first batch update of GD has been calculated. It can also be seen that learning via GD causes fluctuation in testing accuracy which stabilises with time. EDN's performance displays less variance between training examples with a final testing accuracy of 99.4% reached in 298 iterations using 216 neurons. A few thousand training examples is required by GD before testing accuracy converges towards a performance of 99.4%. This is over 25 times slower than EDN with testing accuracy still not completely converged. Neuron count continues to increase in EDN with testing accuracy remaining constant throughout the remainder of the training. Early stopping could have been used here to limit network growth, however, it continued to display the lack of overfitting present after convergence.

### **Auto-mpg - non-linear regression**

The auto-mpg regression data set [1] consists of 398 cars with 9 attributes such as number of engine cylinders and horsepower. The aim is to process the attributes and output the miles-per-gallon of the car. K- fold cross validation of  $K=10$  is used to evaluate performance. The results shown are the average testing accuracy across all folds. The EDN parameters used were  $s = 0.4$ ,  $E_{th} = 0$  and all inputs were selected by each neuron. Two different configurations are shown for GD to display how batch size and learning rate can effect learning. A single layer of 1024 hidden neurons is used as this resulted in the fastest and most stable learning using Adam optimisation.

Mean squared error (MSE) is calculated over the whole testing set after each presentation of an input or batch in the GD cases where the batch size is greater than one. A comparison between EDN and GD on the auto-mpg data set can be seen in Figure 5.6. EDN begins with its MSE above the ANN's, however, it rapidly acquires enough data points to surpass any converged MSE achieved by GD. GD achieved a minimum MSE of  $24.9\text{mpg}^2$  with a batch size of 1 and  $32.9\text{mpg}^2$  with a batch size of 32 during the 20 epochs of training. EDN surpasses the minimum achieved by GD with an MSE of  $23.2\text{mpg}^2$  after 29 training examples (creating 29 neurons) and an MSE of  $12.2\text{mpg}^2$  at the end of the first epoch (creating 358 neurons as  $E_{th} = 0$  so every sample creates a neuron). The error continues to be fine tuned during the proceeding epochs until a converged MSE of  $10.25\text{mpg}^2$ . If it assumed that GD reached its best performance at 4000 training iterations (even though performance has not converged yet, especially for the batch size of 32) this makes EDN over 135 times faster



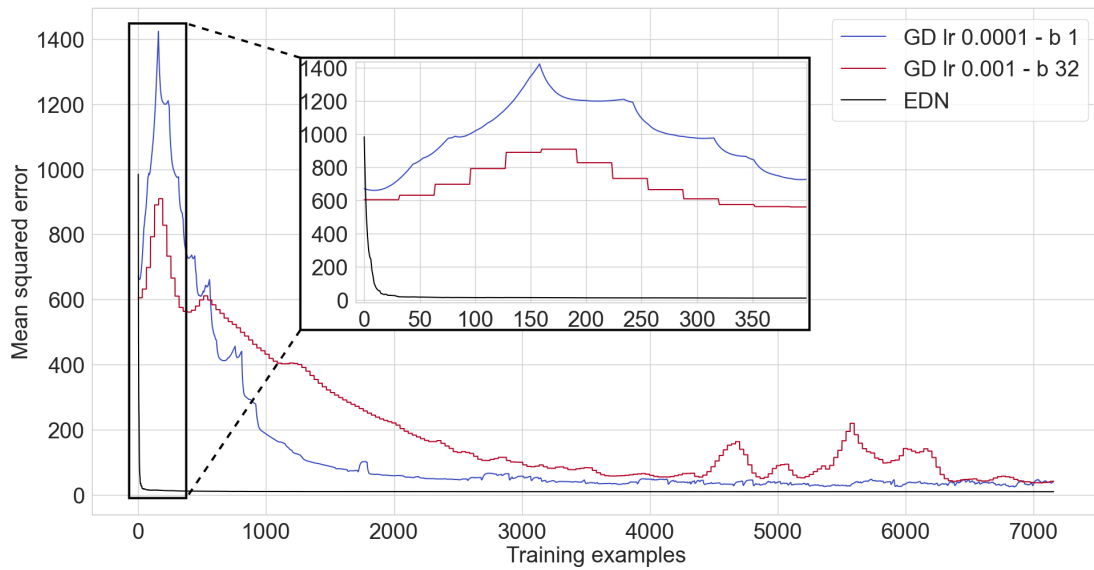


Figure 5.6: A comparison of an ANN trained via gradient descent using Adam optimisation (red and blue, with  $\text{lr}$  being the learning rate and  $b$  being the batch size) against EDN (black) applied to the auto-mpg non-linear regression task. A zoomed in inset is given to display the learning curve of EDN. Batch sizes and learning rates were chosen which gave the fastest convergence in gradient descent.

than GD to achieve comparable accuracy. With further training EDN also converged towards an MSE almost 3 times smaller than GD.

Gradient descent takes considerably longer to produce the same level of performance. EDN can achieve a faster acquisition of appropriate regression values as neurogenesis can instantly store values. This mechanism enables any data points close to this value to be attributed a similar value, which is often close to its true value in regression. EDN is also not effected by the non-linear nature of this regression problem compared to GD. As described in Section 5.2.1 the combination of synapses and neurons in the GD ANN draws a hyperplane through the input dimensions with a neuron's activity being relative to the distance from this plane. This makes producing a smooth output value across the input space difficult for GD to achieve, especially in non-linear regression.

### MNIST - visual classification

MNIST hand-written digit recognition is a common benchmark used in visual classification. It is comprised of the numbers 0-9 discretised into a 28x28 grid with grey-scale pixel intensity from 0-255. The data is split into a training set of 60,000 digits and a

test set of 10,000. The dimensionality of the inputs is far greater for this task than those explored previously and, therefore, an appropriate sampling of the inputs is important to aid performance and reduced the number of parameters in the network. EDN parameters used for experiments unless otherwise specified were  $s_{th} = 0.4$ ,  $E_{th} = 0.1$  and a kernel spread  $s = 0.4$ . For EDN with random input selection the number of input synapses per neuron was limited to 150, which produced the best performance and is inline with the average number of synapses selected per neuron with using  $s_{th} = 0.4$ . The ANN trained with gradient descent with Adam optimisation used 1024 neurons with ReLU activation, a learning rate of 0.001 and a batch size of 64.

The graph in Figure 5.7 compares the performance of EDN against GD. A running average of training classifications is used to enable a fine grained comparison without needing to evaluate over the test set after every training example. As the comparison only shows the first epoch none of the training examples have been seen before and therefore it is equivalent to testing accuracy. As with the previously discussed tasks, a fast acquisition of information allows EDN to reach a higher level of accuracy faster than GD in the initial stages. EDN reaches 90, 92.5 and 95% accuracy after around 4300, 6250, 13000 iterations respectively whereas GD takes around 7200, 12000 and 25000 making EDN almost 2 times faster. When EDN uses random input selection the network's ability to collect the most pertinent input information is hampered resulting in slower learning and a maximum accuracy of around 93% after one epoch. This highlights the importance of surprise driven input selection to guide the network towards the inputs representing information not currently captured by the model.

After one epoch the testing accuracy of EDN is 96.3% with surprise selection and 90.93% with random input selection. GD achieves 96.7% putting surprise selection at comparable levels of performance after seeing the entire training set. EDN's testing accuracy does not increase considerably with further training, gaining only another 0.5% after another epoch and little after that. The parameters for GD were chosen to produce the fastest, stable learning meaning that continued training did not push testing accuracy to the 98%+ seen with state-of-the-art training methods, however, further epochs continued to improve performance up to around 97.5%. These experiments display how EDN is able to acquire information quickly and store it in a functional way that can be applied to the task, however, the ability of GD to perform slight alterations of parameters allows continued refinement of the model, which is not possible with EDN's use of a uniform kernel.

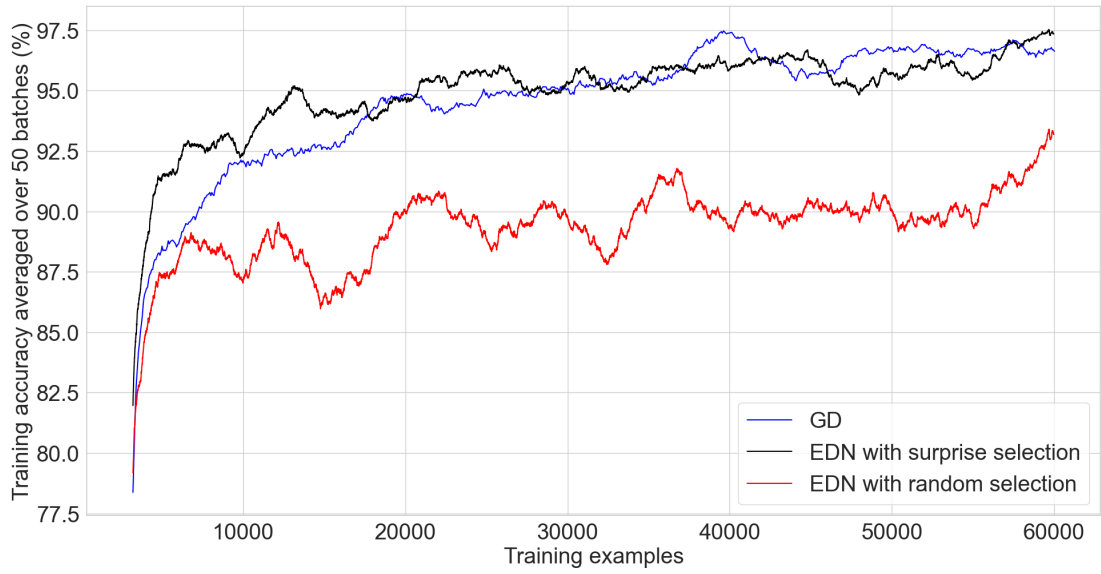


Figure 5.7: A comparison of an ANN trained in tensorflow using Adam optimisation (blue) against EDN with surprise driven input selection (black) and random input selection (red) applied to the MNIST classification task. A batch size of 64 is used to train the ANN and the moving average of the last 50 batches is used to calculate the running training accuracy. EDN training accuracy is the moving average of the last 3200 ( $50 \times 64$ ) training examples as it does not perform batch updates. Training accuracy is used as since this is the first epoch none of the data has been seen before and therefore it is equivalent to testing accuracy.

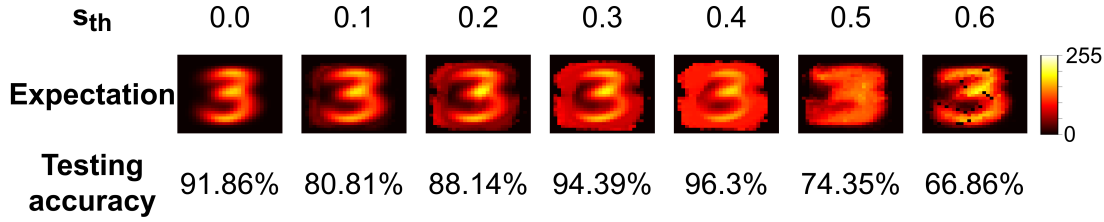


Figure 5.8: A visualisation of the expectation for the class 3 retrieved from the parameters stored by EDN during training on MNIST. A range of surprise thresholds,  $s_{th}$ , are given to show how this effects the stored values and the subsequent effect on testing accuracy after a single epoch through the training set.

### 5.3.1 Visualising the expectation

Using the input surprise method outlined in Section 5.2.2 an expected output can be generated for each class. The expectation is generated by an uniformly weighted combination of the input values stored in the network's  $v$  parameter for each class. In Figure 5.8 the effect of the surprise threshold,  $s_{th}$ , on the expectation and performance after one epoch can be seen. When  $s_{th} = 0$  this corresponds to all inputs being saved when neurogenesis is triggered which leads to a well defined expectation of a 3. As  $s_{th}$  is increased the network now begins to use the expectation of a 3 to guide input selection.

At low values of  $s_{th}$  the performance suffers as many inputs are above threshold, and therefore form synapses with the new neuron, but the most similar inputs do not, resulting in a neuron with an unrepresentative feature-output mapping. When combined they create a good expectation of the class, as can be seen in Figure 5.8, but individually their feature detection suffers. As the threshold increases the neurons begin to select inputs with a greater emphasis on the features of the input that make it different from previous presentations. This leads to a greater diversity and selectivity of feature-output combinations captured by the network. The best performance can be seen when  $s_{th} = 0.4$  and this comes when the expectation is more evenly distributed around the input. This broader expectation allows only the most different inputs to be selected without too much focus on saving what is already captured by the model. When  $s_{th}$  increases beyond this point performance drops considerably as now the threshold is too high for a representative sample of the class to be captured. At  $s_{th} = 0.6$  it can be seen that very few samples are taken as there is pixelation from the lack of samples averaging out the expectation.

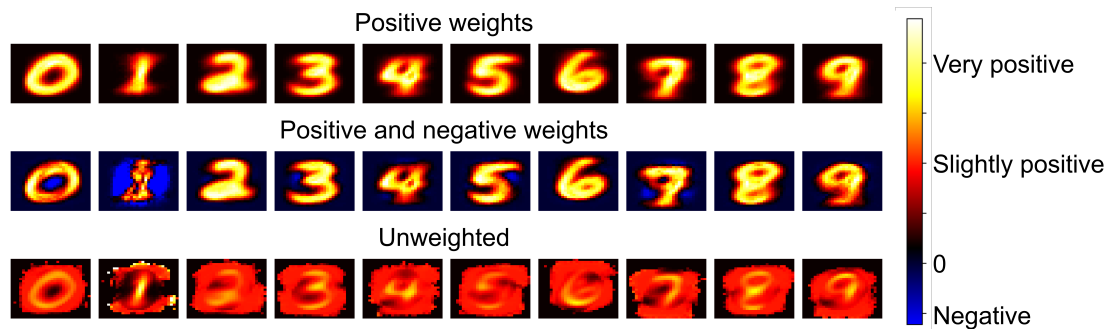


Figure 5.9: The values of  $v$  stored on each neuron's incoming synapses form a record of the inputs which were captured by EDN during training. The weight on the synapse connecting the neurons to the outputs enables the receptive field of each class to be determined by multiplying the stored values by their neuron's associated weights. When taking the positively weighted neurons of each class and multiplying their stored  $v$  values by their associated output weight you create the first row of the plot, this forms a weighted expectation of each class. The second row is generated by also including the negatively weighted neurons connected to each output, showing the average receptive field of each class. The bottom row is an unweighted combination of the input synapses which are instances of each class.

### 5.3.2 Visualising the receptive fields

The previously explored expectation was an unweighted collection of each class's stored values. To examine what each class is sensitive to the weightings of each neuron are included. In Figure 5.9 the receptive fields are extracted in the same way as described in Section 5.2.2 with the stored values of  $v$  on the synapses being multiplied by their associated weight, the weight from the neuron to the output. The top part of Figure 5.9 displays the weighted sum of the neurons which connect positively to the respective outputs, forming an average representation of each class. When negative weights are also included the full receptive field for each class can be seen. Prominent examples of inhibition can be seen for class 0 and 1 where there is strong negative weighting at the centre of the 0 and to the sides of the 1. This technique shows it is straightforward to extract the information present in the network and evaluate what each class is responsive to. It is possible that with some form of clustering that subdivisions of each class could also be extracted, such as sevens with and without a line crossing their middle. Here just the average receptive field is presented.

### 5.3.3 Inverted pendulum - reinforcement learning

Networks are connected to the TensorFlow gym environment [17] `cartpole_v1` to test performance. The task is to keep a pole balanced on a cart without the cart moving too far from the starting position or the angle of the pole moving too far from the vertical. The maximum balance time is 500 timesteps with the task being considered solved if the the average balance time over the last 100 trials is over 475. A slightly broader kernel proved effective in this task resulting in  $s = 0.6$  being chosen. A memory length,  $m$ , of 10 was sufficient which means the last 10 timesteps are classified as a failure and each trigger an individual neurogenesis step. All inputs are selected by each neuron. An ANN actor-critic model is created and trained using GD, a learning rate of 0.003 with Adam optimisation, 128 hidden neurons (more did not improve performance) and a gamma value of 0.99. 100 trials are run for both EDN and the actor-critic with their average performance being shown in figures.

Figure 5.10 compares the performance of EDN against an ANN trained with GD. The quick acquisition of information enables EDN to reach a stable control of the inverted pendulum almost twice as fast as the GD approach case. The actor-critic model takes on average 798 trials before a stable configuration is reached. The best configuration of EDN, with a maximum network size of 350, was able to solve the task in an average of 415 trials. Similar performance is seen for network sizes of 200 and above. The effect of maximum network size on performance is most noticeable below 150 neurons. At this point neuron deletion becomes too frequent to keep a stable set of behaviours in the network. At 100 neurons the performance significantly suffers with the network taking 1091 trials to complete the task. A maximum network size of 50 puts strain on the learning with it failing to solve the task in 2000 trials, only achieving an average balance length of 424 timesteps at the end of training.

Without neuron deletion EDN takes on average 1250 trials to complete the task. Performance is initially better than the actor-critic, displaying the fast learning capability of EDN, although the convergence to stable balancing proves harder leading to more trials needed to average over 475 over the last 100 trials. During training without neuron deletion it was noticed that there were a number of trials in which the learning did not converge to stable behaviour. Instead the performance would quickly drop from balancing for the full duration to struggling to get balance for longer than 100 timesteps. This is likely a result of beneficial behaviours being inhibited via neurogenesis during the learning process and, although those actions are useful to balancing,

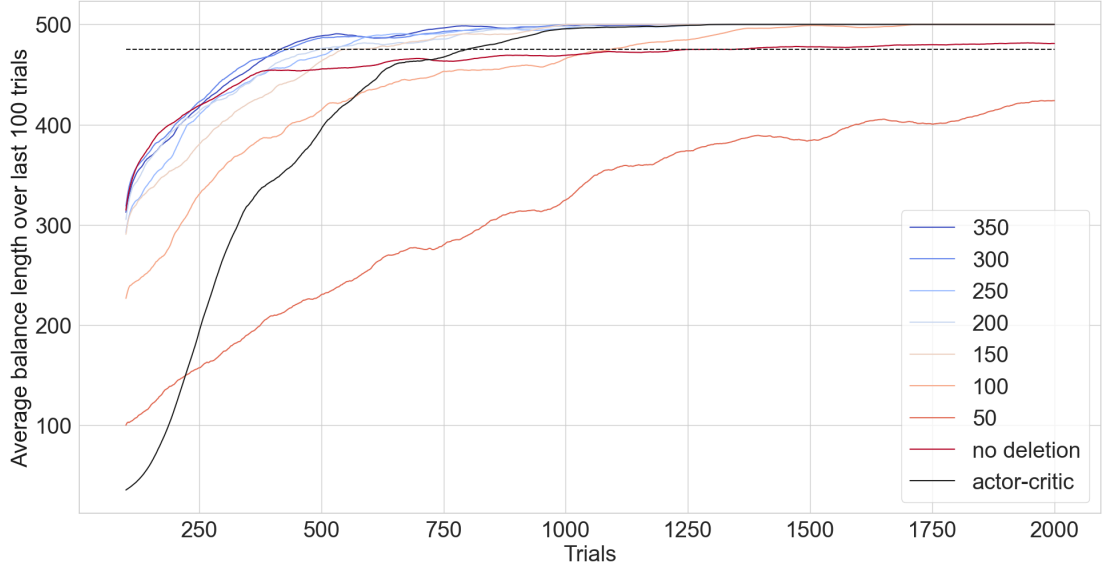


Figure 5.10: EDN with varied network size limits benchmarked against an ANN trained in tensorflow using an actor-critic model (black) applied to the inverted pendulum task. The lines show the running average of the last 100 trials with the dashed line showing the threshold performance required for the task to be considered solved. Each configuration is repeated 100 times and the average of their performance is shown.

causing strain on further learning. Neuron reinforcement and deletion avoids this pitfall by rewarding neurons that contribute to good behaviour and deleting the ones that do not, resulting in a better performing and more condensed network.

The relative speed with which EDN is able to complete this task is a result of the instant labelling of poor behaviour and the adapting of performance. This allows a behavioural space to be built in which actions resulting in negative results are avoided. The rewarding and subsequent deletion of neurons enables this further by injecting positive reward into neurons which are contributing to good behaviour and removing the neurons which do not aid performance. Overall this incorporates reinforcement learning signals into the network behaviour instantly and in a one-shot fashion.

#### 5.3.4 Parametric analysis

Parametric analysis was carried out on the wine classification task. The following section discusses the effect of  $s$ ,  $E_{th}$  and input selection with a focus on the convergence and number of neurons and synapses generated during learning. All experiments used the same random seed and were averaged over a stratified 10 fold cross validation. Unless otherwise specified the parameters used for all tests were  $s = 0.4$ ,  $E_{th} = 0.2$  and

$s_{th} = 0.1$ . Only two of the parametric analysis graphs are included here. Please see the appendix Figure A.1 to Figure A.12 for graphs of testing accuracy, synapse and neuron counts and iteration error for each of the explored parameters.

### The effect of kernel spread

The top plot of Figure 5.11 displays how kernel spread affects the final neuron and synapse count of EDN networks following training. Generally, the more neurons created the worse the performance as the error was more often above  $E_{th}$ . Low values of  $s$  produce many neurons as the hat function is too narrow to allow information transfer across examples. The synaptic response is too specific and therefore the neuron is only active when receiving almost the exact same input causing it to overfit to the training data and perform poorly in testing. When  $s$  is large there is the opposite problem. Note that all inputs are normalised to fall between 0 and 1 and therefore any  $s > 1$  will be active at least by some amount for any input. This means with large values of  $s$  that there are more neurons active at any one time and, therefore, there are over generalisations made about the data. This hurts performance with testing accuracy being far more erratic, especially during the early stage of learning. However, with further training and neuron generation a balance is found in the network for different output activations bringing the final testing accuracy to a comparable level with more appropriately chosen values of  $s$ .

In Figure 5.11 it can be seen that the fewest neurons are created when  $s = 0.6$ , however, this does not correspond to the best testing accuracy. The best performance was found when  $s$  was slightly below this level at 0.4; this is likely because there is less transfer of information between saved data points resulting in more triggering of neurogenesis. Increased neurogenesis in tandem with slightly more specific neuron activation leads to better defined input-output mappings. Overall the choice in  $s$  is a balance between overfitting to the training data at low values and over-generalising from the training data with high values.

### The effect of error threshold

Error threshold,  $E_{th}$ , controls the level at which neurogenesis is triggered. When  $E_{th} = 0$  every training example triggers neurogenesis resulting in as many neurons in the network as examples presented. Increasing the threshold slightly leads to fewer neurons being created with little effect on testing accuracy. It was found that  $E_{th} > 0.2$  was when performance started to be non-negligibly affected, at this point the model



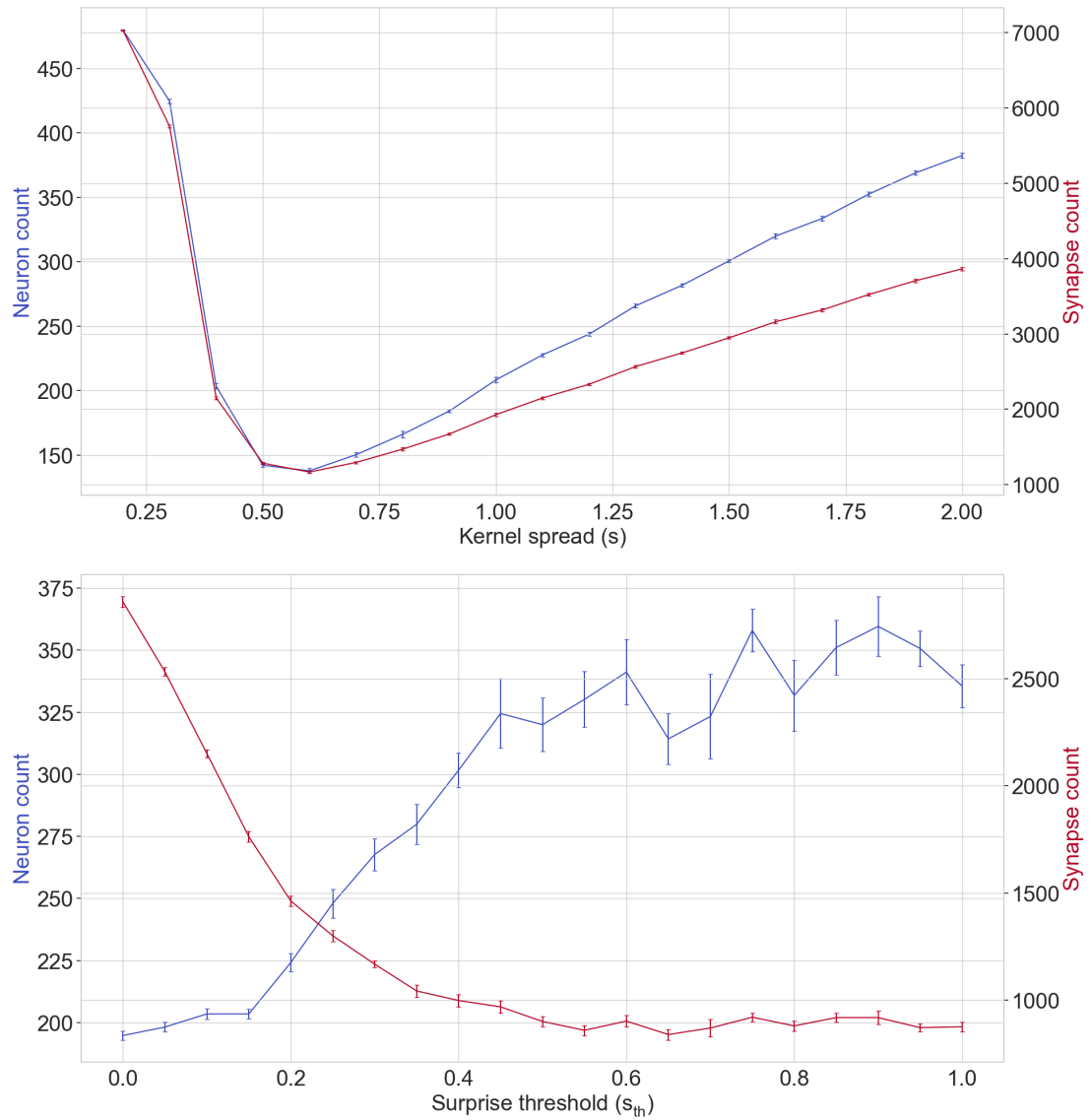


Figure 5.11: Showing the effect of range of parameter values for kernel spread  $s$  (top) and surprise threshold  $s_{th}$  (bottom) on neuron and synapse count after training on the wine classification task. The left y-axis and the line in blue of each plot corresponds with neuron counts. The right y-axis and the red line correspond with the synapse count. The vertical bars show the standard error over a stratified 10 fold cross validation.

becomes a less complete representation of data and testing accuracy reduced. As  $E_{th}$  increases this effect becomes more pronounced with fewer neurons being saved and accuracy dropping until eventually neurogenesis is no longer triggered and the model remains empty. It is possible that some form of annealing of  $E_{th}$  could allow a detailed model to first be created and then add neurons after that if the magnitude of the error is large helping to reduced network size that remains representative of the data.

### The effect of surprise threshold

The method used for selection of synapses in classification tasks is to compare an expected input with the correct input and select the ones with the highest disparity (see Section 5.2.2). The inputs with a surprise value above  $s_{th}$  are chosen to seed the synapses of the newly formed neuron. The bottom plot of Figure 5.11 shows how varying  $s_{th}$  effects the number of synapses and neurons created after training on the wine classification task.

Similar to the results of  $s$ , generally speaking the fewer neurons created the better the performance on the task. From the plot it can be seen that lower values of  $s_{th}$  lead to fewer neurons being created and more synapses being created. For  $s_{th} < 0.2$  the final testing accuracies are equivalent but overall fewer synapses are created the larger the threshold. This shows that the selection process can allow the most important inputs to be selected without hurting overall network performance. This puts focus more on feature-output mappings rather than input-output mappings.

As  $s_{th}$  is increased further the model suffers as the inputs selected by the model become too fragmented and no longer represent the data as a whole. This leads to fewer synapses being created overall whilst the neuron count increases. For high values of  $s_{th}$  eventually a point is reached at which neurogenesis is triggered and no input synapses are selected for the new neuron causing the model to stagnate. Generally the choice of  $s_{th}$  is a balance between saving all information and extracting a reduced form. From MNIST experiments it was found  $s_{th} = 0.4$  allowed the most important features to be extracted resulting in the best performance, again, putting emphasis on the feature-output mapping rather than the input-output mapping.

### How random input selection size effects performance

An alternative method for input selection is to select  $n$  inputs randomly, without replacement, to form the synapses of the new neuron. This avoids the need for constructing an expected input and comparing it with the actual input, reducing the computational overhead for each training example. It was found that when the number of synapses randomly selected was increased the testing accuracy generally increased, however, this also led to increased neurogenesis in these experiments. It would be expected that the better the testing accuracy the less neurogenesis, however, neurogenesis is triggered by the magnitude of the error produced not the classification. The speed at which the error decreased was uniform across the values of  $n$  but the larger the value of  $n$  the larger the error which means more instances in which neurogenesis was triggered. A likely cause for this is that an increased number of synapses per neuron leads to more accurate but less precise representations for each class (accuracy being a measure of correctness and precision a measure of exactness). A more accurate representation of input-output mappings can be stored on the neuron with more synapses as a more complete representation of a data point is stored. However, reducing the number of synapses stored on a neuron makes the input-output mapping more precise as the number of inputs involved in an output prediction are limited. This precise representation comes at the cost of a more fragmented model of each class which hurts testing accuracy. This explains why the MNIST experiments run with random input selection did not get the best performance when all synapses are selected. The best performance does not come from complete input-output mappings but extracting the most important feature-output mappings, creating a more precise representation of the classes.

## 5.4 Discussion

This work has demonstrated how non-linear synaptic activations can be used in conjunction with neurogenesis to tackle a range of tasks in an online fashion. A new neuron is created following an error and the synapses are used to store input values related to that error. This can be done in parallel with network operation and requires no further altering of parameters, which means previously stored information is never forgotten from the model. The instant incorporation of information into the network enables behaviour to be updated immediately. In a robotic scenario this could enable an agent to explore and acquire information without the need for multiple trials and offline

computation. This would allow robots to explore unknown environments whilst updating models of their surroundings. It could also be applied to adjusting their behaviour to account for damage to components without the need for outside intervention, much in the same way humans limp following an injury.

#### *Benchmarking against gradients*

EDN has been applied to a range of tasks (classification, regression and reinforcement learning) and a fast and data efficient learning has been displayed. EDN offered a speed up of 25 times on the wine classification task, 135 times on the auto-mpg regression task, 2 times on MNIST digit recognition and 2 times on the inverted pendulum reinforcement learning task compared to a traditional ANN trained with GD. This improvement in speed is a result of the encoding of input and output values in the neurons enabling immediate application of new information following error driven neurogenesis without the need for gradual parameter updates. Classification accuracy was the same for both EDN and GD, however, mean squared error for the regression task was 3 times smaller with EDN compared to GD. EDN was able to readily attach positions in the input space to regression values. This led to both faster and more accurate performance compared to gradient descent. This is likely a consequence of the non-linear properties of the regression task making capturing the exact input-output mapping more difficult for traditional ANNs. Gradient descent with ANNs requires the building of a statistical summary of the data and encoding it in the weights of a predefined network, paired with the neurons producing an output relative to the distance from a hyperplane (see Section 5.2.1); this makes producing a smooth non-linear output value challenging.

MNIST presented the greatest challenge for EDN with the main gains being shown in comparison to GD in terms of speed. The power of EDN comes from its ability to store a value on the synapses and attribute a particular output to it through the neuron; following this any input that is similar will cause the neuron to activate the associated output. MNIST provides a challenge to this as members of the same class could have exactly the same inputs with only a slight spatial transformation. The result of this transformation is that the input values will now be different, and although the general structure of the input is the same as a previously saved input the input-output mappings will no longer be helpful. If the synapses or neurons could encapsulate possible input transforms, effectively giving the synaptic triangle kernel a spatial spread, then there could be more information transfer between examples. Possibly some form of convolutional filter may achieve a similar affect.

Another possible reason for the difficulty in reaching higher levels of performance on MNIST is because of the increased dimensionality of the input. This makes sampling the correct features and using them to classify inputs increasingly difficult. It was shown that altering the way in which the inputs are sampled can effect the performance of EDN, if sampling or the generation of expected response can be improved it would increase the performance of the model further. A potential route for this could be the implementation of an attention mechanism that adaptively selects different inputs rather than relying on a collective representation of individual classes. It is also possible that a neuron deletion dynamic, as was displayed in reinforcement learning, could enable EDN to remove neurons whose activity does not aid performance in classification.

#### *Future directions*

In the reinforcement learning task the positive reward generated at each timestep was fed into the network and enabled the deletion of the least useful neurons. This acted to put a cap on the network size but came with the added benefit of improving the speed of task completion by removing the least beneficial input-output mappings. In future work it could be useful to find an analogous mechanism for other learning regimes by which network growth can be restricted and potentially, in parallel, improve the performance of the model. As neurogenesis is triggered by errors if they cannot be brought below threshold then network size continues to increase. Some form of early stopping or error threshold annealing may also help curtail the continual growth of the network as performance generally converges quickly with only minor fine tuning happening afterwards even though network growth continues.

The mechanism by which information is stored within the network allows easy access to what specifically is influencing behaviour leading to more informed model debugging. Saved data can also be extracted for post processing and the creation of a more condensed model. This is of particular importance given the size to which EDN can grow a network. Without a cap on network size the number of neurons continues to grow in the presence of error, therefore, a mechanism by which saved data can be elucidated into a reduced model would allow learning over a much longer time period without a growing memory footprint. This would also enable offline training without the need for further examples as previous inputs are saved within the model and can be extracted. The condensed model may also provide beneficial extra dimensions to the inputs, akin to the kernel trick used in support vector machines (SVM). In SVMs, kernels are used to provide additional dimensions to the data thereby allowing a linear

classifier to separate the data. If the produced EDN model could be used to generate ANN neurons this would add extra dimensions to the data which future learning with EDN can take advantage of.

If the branching structure of dendrites could also be made a part of the network's connections, more complex dependencies between features could be constructed. As was shown in the results the best performance came when the most appropriate feature-output mappings were captured by the network and not solely the input-output mappings. Branching dendrites could allow a richer description of inputs to be represented by a single neuron as activity would no longer be a uniform sum of synapse activations. This may allow each neuron to have branching dependencies of inputs with some being more important than others. If neurons could also be allowed to connect to other hidden neurons, instead of only inputs, it would enable synapses to be responsive to higher level features and allow more complex topologies to grow.

#### *The bias of biases*

EDN updates its model by adding new neurons to its current network in response to errors. GD updates its model by moving the weights in the direction gradient calculations dictate will produce less error, eventually creating a statistical summary of the data in its weights. This puts the model bias in EDN on the order in which information is presented as opposed to bias coming from the random initialisation of the network as in GD. This may be utilised with a form of curriculum learning in which simple and representative examples are first presented and the difficulty is increased from there. This is similar to how we teach children (and adults), first starting with simple concepts and characteristic features then incrementally building from there. Modern GD approaches do not require this as they can eventually build a statistical summary of the input-output mappings without concern for first capturing the fundamentals.

#### *The bigger picture*

This work explores how neurogenesis in tandem with synapse non-linearities can be used to store information about inputs in a form that can be used to alleviate errors. It enables fast acquisition of information and updating of a behavioural model. Ultimately, this is not a panacea and is only a part of the puzzle. Traditional ANN transfer functions, whose activity is relative to the distance from a hyperplane, allow neurons to be active beyond a local area of the input space, providing broader statements about input-output mappings. It is likely that a hybrid approach between quick, functional data acquisition and the building of general statistical representations of data is required to create more complete learning systems. This could allow a combination of

learning across timescales, such as the data gathered within a day being condensed down into a reduced model, iteratively increasing the complexity of the representations captured by the network during a lifetime. Something akin to this may happen in biological brains where wakeful hours are used to acquire data through interacting with the world, followed by sleep in which information is consolidated and pushed into a more general model whilst freeing up previously used neural real estate.

# Chapter 6

## Conclusions

### 6.1 Summary and conclusions

*Generally machines are narrowly intelligent*

Humans have taken inspiration from biology in many areas, from aeroplanes to Velcro. Neural networks are another example with their computational structures enabling vast amounts of data to be condensed into their parameters to form a model of a system. ANNs (Artificial Neural Networks) have been trained, using mathematical principles, to surpass human level performance at a range of tasks, such as chess and go. They are games with more possible game states than atoms in the observable universe, however, they fall into a category of environments with perfect information; moving a pawn forwards will always produce the same result, compared to real world environments in which perfect information (especially for training) is not available. Self-driving cars are an example of a domain in which humans are able to reach proficient performance using limited experience but which machine learning models struggle to reach high levels of proficiency due to their lack of a generalised model.

Current neural network training methods are restricted by the methods used to train them, which are heavily reliant on the training data. They are universal function approximators, which when paired with GD (Gradient Descent) allows them to, in theory, model any input-output mapping, however, this then puts a large onus on the data used to train the model [25, 73]. If the data cannot fully encompass all possible input-output mappings then the model will struggle as it does not generalise towards unseen examples well. This can be exemplified in computer vision, a multi-layered feedforward network is a universal function approximator yet it takes a CNN (Convolutional Neural Network) to be able to solve translation invariance (where the position of an object in



the visual field does not effect the detection of the object). If the training data was augmented to include all possible translations of objects then a feedforward architecture could likely compete with a CNN, although it would be more computational expensive to train.

Neural networks need to work towards extracting generalised principles from their training data rather than reliance on building a suitable statistical summary of the input-output mapping. When artificial systems can become more efficient with their data, by making and testing hypotheses, and extract abstract principles, robots will no longer be restricted to training offline and subsequent deployment. This will aid in bridging the gap between training data and real-world environments, which presently is solved by increasing the data presented to the model (through data augmentation or simulation [101, 125]) as training in the real world is too slow. It could also imbue robots with the ability to adapt to unseen environments and situations enabling fast adaptation and fault tolerance.

*Paying attention to attention*

As was discussed in Chapter 3, attention systems allow computational resource to be allocated to the most important elements of an environment. In effect this filters the input stream providing only what is deemed as most relevant to be processed. Having an attention system can move focus around an image, can provide translation invariance as location is only relative to the centre of gaze and in extreme cases can even allow inputs to be shuffled [137].

This thesis showed that an entirely event-based implementation run on SpiNNaker allows the attention processing to be achieved on average in 17.5ms, approximately 7 times faster than a GPU implementation. This displays that when inputs are processed in a biologically plausible way a significant reduction in latency can be achieved. This can allow an attention system to be a preliminary form of processing to better allocate higher level processing without a large cost to processing time.

*To spike or not to spike, is it even a question?*

As explored in the background, spiking neurons offer a number of benefits compared to their non-spiking counterparts:

- sparse activity
- temporal element to processing
- energy efficiency
- robust to noise

- information density (with non rate-based codes)
- low latency
- event-driven

The challenge remains how to train them in a fashion that can fully exploit their benefits and compete with non-spiking ANNs. GD methods have proven incredibly effective with ANNs yet they struggle to transfer to SNNs (Spiking Neural Networks) due to their discontinuity around spike time making the derivative undefined. Attempts have been made to circumvent this problem with the use of pseudo derivatives and surrogate gradients [8, 66, 77, 100, 126], smoothed activation curves [62, 78] and spike time based gradients [23, 96, 148, 151] but they often do not take inspiration from biology and instead rely on approximating ANN neurons with rate-based SNN neurons which is inherently less accurate at transmitting information, therefore, not reaching the same accuracy as ANNs.

E-prop offers a biologically plausible alternative to backpropagation [11], using only locally available information and a global error signal. With this limited information it is able to train recurrently connected LIF (Leaky-Integrate-and-Fire) and ALIF (Adaptive-Leaky-Integrate-and-Fire) neurons online and can be applied to a range of tasks, from classification to Atari games [10]. In Chapter 4 e-prop was instantiated on SpiNNaker to explore its application to real-time, online learning using only local available information. A number of challenges were highlighted in contrast to the TensorFlow implementation, most notably the difficulty in regularising the firing rate. As TensorFlow averaged firing rate over many neurons, it could achieve a more robust approximation of population firing. SpiNNaker could only easily share spike rates across a core, which limited the population average to eight neurons. This showed that, although e-prop uses local information to approximate weight gradients, it relied on global information to regularise the network. Design of learning algorithms must keep in mind the constraints of neuromorphics to better understand the challenges faced by the biological brain. There are also further benefits to be had from EAs (Evolutionary Algorithms), specifically GAs (Genetic Algorithms) due to their increased flexibility of gene expression. Due to SpiNNaker's highly parallel architecture it lends itself to population training as all agents can be run simultaneously without impedance to run-time.

*Gradients are only as good as the hill you are on*

Learning can take many forms, even increasing the number of entries in a lookup table

can be considered a type of learning. Gradient based techniques are a set of powerful training algorithms for updating a parametric model. They calculate the error gradient of some objective function with respect to model parameters and update them in the direction that will mitigate future errors. With careful selection of hyperparameters such as learning rate this can imbue a model with a good representation of the data. However, as discussed in this thesis, there are a number of challenges with gradient descent techniques:

- catastrophic forgetting
- requires differentiability
- only smooth transitions
- vanishing gradients
- exploding gradients
- saddle points
- parameter sensitivity
- sensitive to initial conditions

There are a number of solutions to these problems, such as skip connections that add synapses between distant layers to allow a shortcut for gradients, and pseudo derivatives for the SNN neuron's activation discontinuity. GD is mainly applied to parametric functions where the model topology remains fixed and the function parameters are altered to improve the quality of the model. As all information is placed within the same parameters, care must be taken to ensure learning is controlled, through learning rate and batch size, such that parameter updates can capture qualities of the entire data set without catastrophic forgetting causing previously learned information to be overwritten. This also then limits the longevity of models as further learning means forgetting what was learned. Any time new data is to be trained care must be taken, such as all data being presented to the model again, to update parameters without forgetting what was incorporated before.

Chapter 5 proposed the use of synaptic nonlinearities in tandem with a neurogenesis mechanism to form a non-parametric model that does not overwrite previously learned information. By constructing information without gradients an increased speed

of data acquisition was achieved whilst retaining all learnt information within the network. In EDN (Error Driven Neurogenesis) information remains within the network and can be inspected and retrieved, this allows inspection of model behaviour without inference. Retrieval of stored information could even allow a form of self learning after data acquisition, something akin to sleep. Neural networks have a lot to be gained from more closely matching biology. It was shown with EDN that the nonlinearity of dendrites and neurogenesis can be co-opted to solve classification, regression and reinforcement learning tasks. There are, no doubt, still many more aspects of intelligence to be gleaned from more closely modelling the biological brain.

## 6.2 Future work

The biological brain possesses many facets that are missed from their current computational analogues. Biological neurons alone contain far more complexity than is currently modelled by ANN neurons. When you factor in complex neuromodulatory interactions [42] and the mysterious glial cells that make up the majority of cells in the brain [54] there is still much to unlock. It has been discussed that spiking neurons offer a potentially important step towards increased computational power because of their information density, with certain encoding schemes, and inherent temporal properties. However, the problem remains open as to how to unlock their potential.

### *Attentive towards the future*

As was shown in Chapter 3, a full event-based attention system is possible on neuro-morphic hardware. This was originally designed for real-time operation on the iCub humanoid robot, further work could benefit from increased testing on iCub. Controlling foveation of the robot's gaze was tested to a small extent, and proved possible with saccadic suppression (removal of events caused by moving of the eyes). If this could be paired with a higher level objective, such as looking towards specific objects, this could prove a beneficial research goal.

Controlling the goals of an attention system was shown to be possible with Gestalt principles, however, they are only underlying principles of attention. For more objective driven approaches careful tailoring of filter kernels may prove beneficial, although, in recent years engineered approaches have fallen to the wayside in favour of learned models. The human fixation data set, the ground truth in the SNNevProto work, could be used for supervised learning of a human-like attention model. The architecture could remain the same, which would reduce the number of parameters to learn, leaving

the main kernel to be the learnt element (which can be rotated and up and downsampled as needed), and possibly the percentage firing thresholds between layers to fine tune the response. Appropriate choice of learning algorithm will need to be considered given the sharing of parameters within the model and the spiking neuromorphic implementation. It could be considered an indirect encoding, because the topology is already defined, and then an EA may be well suited as the population can be run in parallel on SpiNNaker.

#### *FuturE-prop*

Having access to a state of the art SNN training algorithm on neuromorphic hardware gives a number of possible directions for future work. Extending to more complex tasks to gauge the limitations of the implementation would be beneficial, such as tackling reinforcement learning tasks with much longer time horizons and less immediate error feedback.

To approach more complex learning domains there are two main challenges that need to be tackled. Firstly, recurrent connections; e-prop boasts being able to train recurrently connected SNNs without the need for BPTT (Back-Propagation-Through-Time), remaining online during the training process [11]. Recurrent connections lend increased computing power, especially with temporal data, to a model. The implementation on SpiNNaker struggled to harness the power of recurrent connections because of the second challenge, firing rate regularisation. Regularising the population firing was difficult because of three main reasons:

- limited to averaging over eight neurons
- sparse/irregular firing of inputs
- cannot easily calculate firing rate over short time scales

A potential solution and direction of future work could be to move to a multi-core model of e-prop neurons. This would split the neurons and synapses to work on different cores giving each one increased memory and processing time. This in turn would allow increased neurons per core and better approximation of firing rates. It would also enable increased time for spike processing, this would reduce problems during the learning process of regularisation pushing firing rates too high and causing spikes to be dropped and simulations to crash.

#### *The end of the beginning of the EDN*

It was shown in Chapter 5 that using nonlinear synapses with uniform parameters was

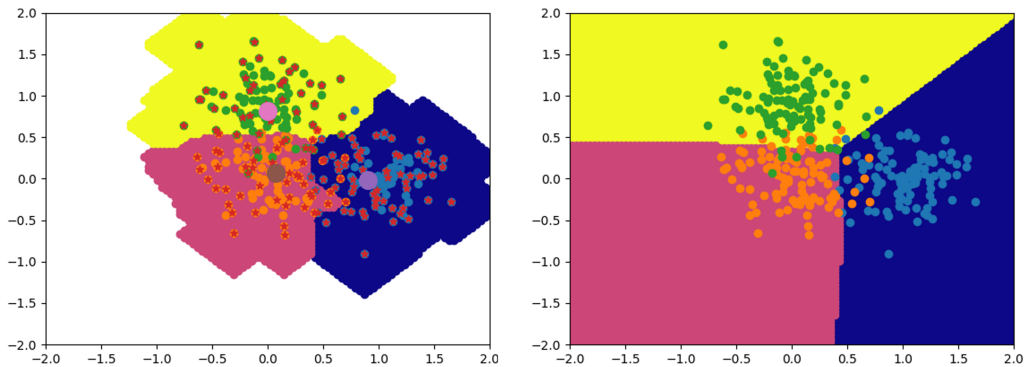


Figure 6.1: On the left shows the EDN network output after training on a three class problem. Dots are the training points with the dots with stars representing the saved data points and the larger dots are the weighted average of each class. On the right a tanh neuron is created which draws a hyperplane between all pairs of class centroids with their outputs corresponding to the respective class on each side of the hyperplane. When observing the outputs of the tanh neurons (the coloured background) it can be seen that a good class separation is acquired using nonlinear neurons without any gradient descent.

able to achieve competitive results at a range of tasks. A possible future iteration of EDN could be used to perform some form of unsupervised learning, after training, in which the network's parameters, output weights and kernel spreads, are trained to form a cohesive whole. As all the learnt information is present within the network it is possible that the network can perform self-supervised learning using the stored information. This may form a sleep-like state in which learnt information is integrated to form a complete picture.

Another possible sleep like mechanism may come in the form of network conversion. In Figure 6.1 a simple mechanism is displayed in which hyperplanes are created between class centres, which when combined create a condensed representation of the original data. These hyperplanes add further dimensions to the data which can be further sampled by EDN providing potentially useful features for learning. This would create a hybrid system capable of both fast learning, via EDN, and broadly applicable feature extraction, via nonlinear neurons. This combined approach could allow continual learning and feature extraction in an entirely self-supervised manner. It would also be entirely offline, as data has been acquired during initial training, making it akin to some sleep-like mechanism.

# Bibliography

- [1] Auto MPG. UCI Machine Learning Repository, 1993. URL <https://archive.ics.uci.edu/ml/datasets/auto+mpg>.
- [2] Mahdi Abolfazli Esfahani, Han Wang, Benyamin Bashari, Keyu Wu, and Shenghai Yuan. Learning to extract robust handcrafted features with a single observation via evolutionary neurogenesis. *Applied Soft Computing*, 106: 107424, 2021. ISSN 1568-4946. doi: <https://doi.org/10.1016/j.asoc.2021.107424>. URL <https://www.sciencedirect.com/science/article/pii/S1568494621003471>.
- [3] Samantha V Adams, Alexander D Rast, Cameron Patterson, Francesco Galluppi, Kevin Brohan, José-Antonio Pérez-Carrasco, Thomas Wennekers, Steve Furber, and Angelo Cangelosi. Towards real-world neurorobotics: integrated neuromorphic visual attention. In *International Conference on Neural Information Processing*, pages 563–570. Springer, 2014.
- [4] Bruna Alves. Average household energy consumption uk, Feb 2022. URL <https://www.statista.com/statistics/517845/average-electricity-consumption-uk/>.
- [5] Dario Amodei. Ai and compute, Jun 2021. URL <https://openai.com/blog/ai-and-compute/>.
- [6] Árpád Baricz. *Generalized Bessel functions of the first kind*. Springer, 2010.
- [7] Chiara Bartolozzi, Francesco Rea, Charles Clercq, Daniel B Fasnacht, Giacomo Indiveri, Michael Hofstätter, and Giorgio Metta. Embedded neuromorphic vision for humanoid robots. In *CVPR 2011 WORKSHOPS*, pages 129–135. IEEE, 2011.

- [8] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *Advances in neural information processing systems*, 31, 2018.
- [9] Guillaume Bellec, Franz Scherr, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets. *CoRR*, abs/1901.09049, 2019. URL <http://arxiv.org/abs/1901.09049>.
- [10] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *bioRxiv*, 2019. doi: 10.1101/738385. URL <https://www.biorxiv.org/content/early/2019/12/09/738385>.
- [11] Guillaume Bellec, Franz Scherr, Anand Subramoney, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature communications*, 11(1):1–15, 2020.
- [12] Evert Willem Beth. *Mathematical Epistemology and Psychology*. New York: Gordon & Breach, 1966.
- [13] Charles Blundell, Benigno Uria, Alexander Pritzel, Yazhe Li, Avraham Ruderman, Joel Z Leibo, Jack Rae, Daan Wierstra, and Demis Hassabis. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.
- [14] Thomas Bohnstingl, Franz Scherr, Christian Pehle, Karlheinz Meier, and Wolfgang Maass. Neuromorphic hardware learns to learn. *Frontiers in Neuroscience*, 13:483, 2019. ISSN 1662-453X. doi: 10.3389/fnins.2019.00483. URL <https://www.frontiersin.org/article/10.3389/fnins.2019.00483>.
- [15] Ali Borji and Laurent Itti. Cat2000: A large scale fixation dataset for boosting saliency research. *CVPR 2015 workshop on "Future of Datasets"*, 2015. arXiv preprint arXiv:1505.03581.
- [16] Ali Borji, Dicky N Sihite, and Laurent Itti. Quantitative analysis of human-model agreement in visual saliency modeling: A comparative study. *Image Processing, IEEE Transactions on*, 22(1):55–69, 2013.



- [17] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. doi: <https://doi.org/10.48550/arXiv.1606.01540>. URL <http://arxiv.org/abs/1606.01540>.
- [18] Dwight A Burkhardt and Patrick K Fahey. Contrast enhancement and distributed encoding by bipolar cells in the retina. *Journal of Neurophysiology*, 80(3):1070–1081, 1998.
- [19] Zoya Bylinskii, Tilke Judd, Aude Oliva, Antonio Torralba, and Frédo Durand. What do different evaluation metrics tell us about saliency models? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(3):740–757, 2019. doi: 10.1109/TPAMI.2018.2815601.
- [20] Luis Camunas-Mesa, Carlos Zamarreño-Ramos, Alejandro Linares-Barranco, Antonio J Acosta-Jimenez, Teresa Serrano-Gotarredona, and Bernabé Linares-Barranco. An event-driven multi-kernel convolution processor module for event-driven vision sensors. *IEEE Journal of Solid-State Circuits*, 47(2):504–517, 2011.
- [21] Manuela Chessa, Guido Maiello, Peter J Bex, and Fabio Solari. A space-variant model for motion interpretation across the visual field. *Journal of vision*, 16(2): 12–12, 2016.
- [22] E. Chicca, G. Indiveri, and R. Douglas. An adaptive silicon synapse. In *Proceedings of the 2003 International Symposium on Circuits and Systems, 2003. ISCAS '03.*, volume 1, pages I–I, 2003. doi: 10.1109/ISCAS.2003.1205505.
- [23] I. M. Comsa, T. Fischbacher, K. Potempa, A. Gesmundo, L. Versari, and J. Alakuijala. Temporal coding in spiking neural networks with alpha synaptic function. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8529–8533, 2020.
- [24] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. doi: 10.1109/TIT.1967.1053964.
- [25] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

- [26] Giulia D’Angelo, Ella Janotte, Thorben Schoepe, James O’Keeffe, Moritz B Milde, Elisabetta Chicca, and Chiara Bartolozzi. Event-based eccentric motion detection exploiting time difference encoding. *Frontiers in neuroscience*, 14: 451, 2020.
- [27] Ch. Sanjeev Kumar Dash, Ajit Kumar Behera, Satchidananda Dehuri, and Sung-Bae Cho. Radial basis function neural networks: a topical state-of-the-art survey. *Open Computer Science*, 6(1):33–63, 2016. doi: doi:10.1515/comp-2016-0005. URL <https://doi.org/10.1515/comp-2016-0005>.
- [28] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, Yuyun Liao, Chit-Kwan Lin, Andrew Lines, Ruokun Liu, Deepak Mathaikutty, Steven McCoy, Arnab Paul, Jonathan Tse, Guruguhanathan Venkataramanan, Yi-Hsin Weng, Andreas Wild, Yoonseok Yang, and Hong Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1): 82–99, 2018. doi: 10.1109/MM.2018.112130359.
- [29] Andrew Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. Pynn: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2, 2009. ISSN 1662-5196. doi: 10.3389/neuro.11.011.2008. URL <https://www.frontiersin.org/articles/10.3389/neuro.11.011.2008>.
- [30] Andrew P Davison, Daniel Brüderle, Jochen Eppler, Jens Kremkow, Eilif Muller, Dejan Pecevski, Laurent Perrinet, and Pierre Yger. PyNN: a common interface for neuronal network simulators. *Frontiers in Neuroinformatics*, 2(January):11, 2008. ISSN 16625196. doi: 10.3389/neuro.11.011.2008. URL <http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2634533&tool=pmcentrez&rendertype=abstract>.
- [31] Tobi Delbruck, Chenghan Li, Rui Graca, and Brian Mcreynolds. Utility and feasibility of a center surround event camera. *arXiv preprint arXiv:2202.13076*, 2022.
- [32] Wei Deng, James B. Aimone, and Fred H. Gage. New neurons and new memories: how does adult hippocampal neurogenesis affect learning and memory? *Nature Reviews Neuroscience*, 11(5):339–350, May 2010. ISSN 1471-0048. doi: 10.1038/nrn2822. URL <https://doi.org/10.1038/nrn2822>.

- [33] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. 2020. doi: 10.48550/ARXIV.2010.11929. URL <https://arxiv.org/abs/2010.11929>.
- [34] Timothy J. Draelos, Nadine E. Miner, Christopher C. Lamb, Jonathan A. Cox, Craig M. Vineyard, Kristofor D. Carlson, William M. Severa, Conrad D. James, and James B. Aimone. Neurogenesis deep learning: Extending deep networks to accommodate new classes. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 526–533, 2017. doi: 10.1109/IJCNN.2017.7965898.
- [35] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL <http://archive.ics.uci.edu/ml>.
- [36] Hermann Ebbinghaus. Memory: A contribution to experimental psychology (no. 3). *University Microfilms*, 1913.
- [37] Charles W Eriksen and James D St James. Visual attention within and around the field of focal attention: A zoom lens model. *Perception & psychophysics*, 40(4):225–240, 1986.
- [38] Pontus Eriksson and Love Westlund Gotby. Dynamic network architectures for deep q-learning: Modelling neurogenesis in artificial intelligence. 2019.
- [39] Chrisantha Fernando, Dylan Banarse, Malcolm Reynolds, Frederic Besse, David Pfau, Max Jaderberg, Marc Lanctot, and Daan Wierstra. Convolution by evolution: Differentiable pattern producing networks. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 109–116. ACM, 2016.
- [40] Burkart Fischer. Overlap of receptive field centers and representation of the visual field in the cat’s optic tract. *Vision research*, 13(11):2113–2120, 1973.
- [41] Evelyn Fix and J. L. Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. *International Statistical Review / Revue Internationale de Statistique*, 57(3):238–247, 1989. ISSN 03067734, 17515823. URL <http://www.jstor.org/stable/1403797>.

- [42] Nicolas Frémaux and Wulfram Gerstner. Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in neural circuits*, 9:85, 2016.
- [43] Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999. ISSN 1364-6613. doi: [https://doi.org/10.1016/S1364-6613\(99\)01294-2](https://doi.org/10.1016/S1364-6613(99)01294-2). URL <https://www.sciencedirect.com/science/article/pii/S1364661399012942>.
- [44] Kunihiro Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In Shun-ichi Amari and Michael A. Arbib, editors, *Competition and Cooperation in Neural Nets*, pages 267–285, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg. ISBN 978-3-642-46466-9.
- [45] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana. The spinnaker project. *Proceedings of the IEEE*, 102(5):652–665, May 2014. ISSN 0018-9219. doi: 10.1109/JPROC.2014.2304638.
- [46] Steve Furber and Petru Bogdan. *SpiNNaker-A Spiking Neural Network Architecture*. Now publishers, 2020.
- [47] Albert Gidon, Timothy Adam Zolnik, Pawel Fidzinski, Felix Bolduan, Athanasia Papoutsis, Panayiota Poirazi, Martin Holtkamp, Imre Vida, and Matthew Evan Larkum. Dendritic action potentials and computation in human layer 2/3 cortical neurons. *Science*, 367(6473):83–87, 2020. doi: 10.1126/science.aax6239. URL <https://science.sciencemag.org/content/367/6473/83>.
- [48] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.
- [49] Arren Glover and Chiara Bartolozzi. Event-driven ball detection and gaze fixation in clutter. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2203–2208. IEEE, 2016.

- [50] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014. doi: <https://doi.org/10.48550/arXiv.1410.5401>.
- [51] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.
- [52] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.
- [53] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2829–2838, New York, New York, USA, 20–22 Jun 2016. PMLR. URL <https://proceedings.mlr.press/v48/gu16.html>.
- [54] Fei He and Yi E. Sun. Glial cells more than support cells? *The International Journal of Biochemistry & Cell Biology*, 39(4):661–665, 2007. ISSN 1357-2725. doi: <https://doi.org/10.1016/j.biocel.2006.10.022>. URL <https://www.sciencedirect.com/science/article/pii/S1357272506003141>.
- [55] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.
- [56] A. L. HODGKIN and A. F. HUXLEY. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, Aug 1952. ISSN 0022-3751. doi: 10.1113/jphysiol.1952.sp004764. URL <https://pubmed.ncbi.nlm.nih.gov/12991237>. 12991237[pmid].
- [57] A. L. HODGKIN, A. F. HUXLEY, and B. KATZ. Measurement of current-voltage relations in the membrane of the giant axon of loligo. *The Journal of physiology*, 116(4):424–448, Apr 1952. ISSN 0022-3751. doi: 10.1113/jphysiol.1952.sp004716. URL <https://pubmed.ncbi.nlm.nih.gov/14946712>. 14946712[pmid].

- [58] Brian Hu, Ralindae Kane-Jackson, and Ernst Niebur. A proto-object based saliency model in three-dimensional space. *Vision research*, 119:42–49, 2016.
- [59] Zhiheng Huang, Wei Xu, and Kai Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015. doi: <https://doi.org/10.48550/arXiv.1508.01991>.
- [60] D. H. Hubel and T. N. Wiesel. Ferrier lecture: Functional architecture of macaque monkey visual cortex. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 198(1130):1–59, 1977. ISSN 00804649. URL <http://www.jstor.org/stable/77245>.
- [61] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.
- [62] Dongsung Huh and Terrence J Sejnowski. Gradient descent for spiking neural networks. pages 1433–1443, 2018.
- [63] Massimiliano Iacono, Giulia D’Angelo, Arren Glover, Vadim Tikhanoﬀ, Ernst Niebur, and Chiara Bartolozzi. Proto-object based saliency for event-driven cameras. In *IROS*, pages 805–812, 2019.
- [64] L Itti and Christof Koch. Computational modelling of visual attention. *Nature reviews. Neuroscience*, 2(3):194–203, mar 2001. ISSN 1471-003X. doi: 10.1038/35058500. URL <http://www.ncbi.nlm.nih.gov/pubmed/11256080>.
- [65] Tilke Judd, Frédo Durand, and Antonio Torralba. A benchmark of computational models of saliency to predict human fixations. In *MIT Technical Report*, 2012.
- [66] Jacques Kaiser, Hesham Mostafa, and Emre Neftci. Synaptic plasticity dynamics for deep continuous local learning. *arXiv preprint arXiv:1811.10766*, 2018.
- [67] Haruo Kasai, Masahiro Fukuda, Satoshi Watanabe, Akiko Hayashi-Takagi, and Jun Noguchi. Structural dynamics of dendritic spines in memory and cognition. *Trends in Neurosciences*, 33(3):121–129, 2010. ISSN 0166-2236. doi: <https://doi.org/10.1016/j.tins.2010.01.001>. URL <https://www.sciencedirect.com/science/article/pii/S0166223610000020>.

- [68] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL <https://arxiv.org/abs/1412.6980>.
- [69] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017. doi: 10.1073/pnas.1611835114. URL <https://www.pnas.org/doi/abs/10.1073/pnas.1611835114>.
- [70] C Koch and S Ullman. Shifts in selective visual attention: towards the underlying neural circuitry. *Human neurobiology*, 4(4):219–227, 1985.
- [71] Kristin Koch, Judith McLean, Michael Berry, Peter Sterling, Vijay Balasubramanian, and Michael A. Freed. Efficiency of Information Transmission by Retinal Ganglion Cells. *Current Biology*, 14(17):1523–1530, sep 2004. ISSN 09609822. doi: 10.1016/j.cub.2004.08.060. URL [https://ac.els-cdn.com/S0960982204006566/1-s2.0-S0960982204006566-main.pdf?{\\_}tid=3dfa152e-d580-4890-998f-01b5291f1cd7{&}acdnat=1549451672{&\\_}da6bc1f8f08d4bee99e6653668b165a0https://linkinghub.elsevier.com/retrieve/pii/S0960982204006566](https://ac.els-cdn.com/S0960982204006566/1-s2.0-S0960982204006566-main.pdf?{_}tid=3dfa152e-d580-4890-998f-01b5291f1cd7{&}acdnat=1549451672{&_}da6bc1f8f08d4bee99e6653668b165a0https://linkinghub.elsevier.com/retrieve/pii/S0960982204006566).
- [72] Wolfgang Köhler. Gestalt psychology. *Psychological research*, 31(1):XVIII–XXX, 1967.
- [73] Anastasis Kratsios and Ievgen Bilokopytov. Non-euclidean universal approximation. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 10635–10646. Curran Associates, Inc., 2020. URL <https://proceedings.neurips.cc/paper/2020/file/786ab8c4d7ee758f80d57e65582e609d-Paper.pdf>.
- [74] Janus J Kulikowski, S Marčelja, and Peter O Bishop. Theory of spatial position and spatial frequency relations in the receptive fields of simple cells in the visual cortex. *Biological cybernetics*, 43(3):187–198, 1982.

- [75] Matthias Kummerer, Thomas SA Wallis, and Matthias Bethge. Saliency benchmarking made easy: Separating models, maps and metrics. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 770–787, 2018.
- [76] Congyan Lang, Tam V Nguyen, Harish Katti, Karthik Yadati, Mohan Kankanhalli, and Shuicheng Yan. Depth matters: Influence of depth cues on visual saliency. In *European conference on computer vision*, pages 101–115. Springer, 2012.
- [77] Chankyu Lee, Syed Shakib Sarwar, and Kaushik Roy. Enabling spike-based backpropagation in state-of-the-art deep neural network architectures. *CoRR*, abs/1903.06379, 2019. URL <http://arxiv.org/abs/1903.06379>.
- [78] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10: 508, 2016. ISSN 1662-453X. doi: 10.3389/fnins.2016.00508. URL <https://www.frontiersin.org/article/10.3389/fnins.2016.00508>.
- [79] Máté Lengyel and Peter Dayan. Hippocampal contributions to control: The third way. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2008. URL <https://proceedings.neurips.cc/paper/2007/file/1f4477bad7af3616c1f933a02bfabe4e-Paper.pdf>.
- [80] Steve LeVine. Artificial intelligence pioneer says we need to start over. *Arlington, VA: Axios*, 2017.
- [81] Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7(1):13276, 2016. ISSN 2041-1723. doi: 10.1038/ncomms13276. URL <https://doi.org/10.1038/ncomms13276>.
- [82] Zichuan Lin, Tianqi Zhao, Guangwen Yang, and Lintao Zhang. Episodic memory deep q-networks. *CoRR*, abs/1805.07603, 2018. URL <http://arxiv.org/abs/1805.07603>.
- [83] Jing Liu, Yang Xiao, Qi Hao, and Kaveh Ghaboosi. Bio-inspired visual attention in agile sensing for target detection. *IJSNet*, 5(2):98–111, 2009.



- [84] David Lopez-Paz and Marc Aurelio Ranzato. Gradient episodic memory for continual learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL <https://proceedings.neurips.cc/paper/2017/file/f87522788a2be2d171666752f97ddeb-Paper.pdf>.
- [85] Wolfgang Maass. Networks of spiking neurons: The third generation of neural network models. *Neural Networks*, 10(9):1659 – 1671, 1997. ISSN 0893-6080. doi: [https://doi.org/10.1016/S0893-6080\(97\)00011-7](https://doi.org/10.1016/S0893-6080(97)00011-7). URL <http://www.sciencedirect.com/science/article/pii/S0893608097000117>.
- [86] George R Mangun. Neural mechanisms of visual selective attention. *Psychophysiology*, 32(1):4–18, 1995.
- [87] Charles E. Martin and Praveen K. Pilly. Probabilistic Program Neurogenesis. ALIFE 2019: The 2019 Conference on Artificial Life:440–447, 07 2019. doi: 10.1162/isal\_a\_00199. URL [https://doi.org/10.1162/isal\\_a\\_00199](https://doi.org/10.1162/isal_a_00199).
- [88] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. 5(4):115–133. ISSN 1522-9602. doi: 10.1007/BF02478259. URL <https://doi.org/10.1007/BF02478259>.
- [89] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, and Francesco Nori. The icub humanoid robot: an open platform for research in embodied cognition. In *Proceedings of the 8th workshop on performance metrics for intelligent systems*, pages 50–56, 2008.
- [90] Xiongkuo Min, Guangtao Zhai, Ke Gu, and Xiaokang Yang. Fixation prediction through multimodal analysis. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 13(1):1–23, 2016.
- [91] Silviu Minut and Sridhar Mahadevan. A reinforcement learning model of selective visual attention. In *Proceedings of the fifth international conference on Autonomous agents*, pages 457–464, 2001.
- [92] John Mixter and Ali Akoglu. Growing artificial neural networks. In Hamid R. Arabnia, Ken Ferens, David de la Fuente, Elena B. Kozerenko, José Angel Olivás Varela, and Fernando G. Tinetti, editors, *Advances in Artificial Intelligence*

- and Applied Cognitive Computing*, pages 409–423, Cham, 2021. Springer International Publishing.
- [93] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. pages 2204–2212, 2014.
  - [94] Jamal Lottier Molin, Alexander F Russell, Stefan Mihalas, Ernst Niebur, and Ralph Etienne-Cummings. Proto-object based visual saliency model with a motion-sensitive channel. In *2013 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 25–28. IEEE, 2013.
  - [95] Ari S Morcos and Christopher D Harvey. History-dependent variability in population dynamics during evidence accumulation in cortex. *Nature neuroscience*, 19(12):1672–1681, 2016.
  - [96] H. Mostafa. Supervised learning based on temporal coding in spiking neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(7): 3227–3235, July 2018. ISSN 2162-2388. doi: 10.1109/TNNLS.2017.2726060.
  - [97] Scott O. Murray and Ewa Wojciulik. Attention increases neural selectivity in the human lateral occipital complex. *Nature Neuroscience*, 7(1):70–74, Jan 2004. ISSN 1546-1726. doi: 10.1038/nn1161. URL <https://doi.org/10.1038/nn1161>.
  - [98] Jaap M. J. Murre and Joeri Dros. Replication and analysis of ebbinghaus’ forgetting curve. *PLOS ONE*, 10(7):1–23, 07 2015. doi: 10.1371/journal.pone.0120644. URL <https://doi.org/10.1371/journal.pone.0120644>.
  - [99] Tomoki Nakaya and Keiji Yano. Visualising crime clusters in a space-time cube: An exploratory data-analysis approach using space-time kernel density estimation and scan statistics. *Transactions in GIS*, 14(3):223–239, 2010.
  - [100] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks. *arXiv preprint arXiv:1901.09948*, 2019.
  - [101] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder,

- Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand. 2019. doi: 10.48550/ARXIV.1910.07113. URL <https://arxiv.org/abs/1910.07113>.
- [102] Lucas Paletta. *Attention in Cognitive Systems. Theories and Systems from an Interdisciplinary Viewpoint: 4th International Workshop on Attention in Cognitive Systems, WAPCV 2007 Hyderabad, India, January 8, 2007 Revised Selected Papers*, volume 4840. Springer Science & Business Media, 2007.
- [103] German Ignacio Parisi, Xu Ji, and Stefan Wermter. On the role of neurogenesis in overcoming catastrophic forgetting. *CoRR*, abs/1811.02113, 2018. doi: <https://doi.org/10.48550/arXiv.1811.02113>. URL <http://arxiv.org/abs/1811.02113>.
- [104] J. Park and I. W. Sandberg. Universal Approximation Using Radial-Basis-Function Networks. *Neural Computation*, 3(2):246–257, 06 1991. ISSN 0899-7667. doi: 10.1162/neco.1991.3.2.246. URL <https://doi.org/10.1162/neco.1991.3.2.246>.
- [105] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluís-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. Carbon emissions and large neural network training. 2021. doi: 10.48550/ARXIV.2104.10350. URL <https://arxiv.org/abs/2104.10350>.
- [106] David Patterson, Joseph Gonzalez, Urs Hölzle, Quoc Le, Chen Liang, Lluís-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. The carbon footprint of machine learning training will plateau, then shrink, 2022. URL <https://arxiv.org/abs/2204.05149>.
- [107] Adam Perrett, Steve B. Furber, and Oliver Rhodes. Error driven synapse augmented neurogenesis. *Frontiers in Artificial Intelligence*, 5, 2022. ISSN 2624-8212. doi: 10.3389/frai.2022.949707. URL <https://www.frontiersin.org/articles/10.3389/frai.2022.949707>.
- [108] Christoph Posch, Daniel Matolin, and Rainer Wohlgenannt. A QVGA 143 dB dynamic range frame-free PWM image sensor with lossless pixel-level video compression and time-domain CDS. In *IEEE Journal of Solid-State Circuits*, volume 46, pages 259–275, jan 2011. ISBN 9781424460342. doi: 10.

- 1109/JSSC.2010.2085952. URL <http://ieeexplore.ieee.org/document/5648367/>.
- [109] Michael I Posner. Orienting of attention. *Quarterly journal of experimental psychology*, 32(1):3–25, 1980.
- [110] Arjun Rao, Philipp Plank, Andreas Wild, and Wolfgang Maass. A long short-term memory for ai applications in spike-based neuromorphic hardware. *Nature Machine Intelligence*, 4(5):467–479, May 2022. ISSN 2522-5839. doi: 10.1038/s42256-022-00480-w. URL <https://doi.org/10.1038/s42256-022-00480-w>.
- [111] Francesco Rea, Giorgio Metta, and Chiara Bartolozzi. Event-driven visual attention for the humanoid robot icub. *Frontiers in Neuroscience*, 7:234, 2013. ISSN 1662-453X. doi: 10.3389/fnins.2013.00234. URL <https://www.frontiersin.org/article/10.3389/fnins.2013.00234>.
- [112] Henri Rebecq, Timo Horstschäfer, Guillermo Gallego, and Davide Scaramuzza. Evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real time. *IEEE Robotics and Automation Letters*, 2(2):593–600, 2016.
- [113] Henri Rebecq, Daniel Gehrig, and Davide Scaramuzza. ESIM: an open event camera simulator. *Conf. on Robotics Learning (CoRL)*, October 2018.
- [114] Oliver Rhodes, Petruț A. Bogdan, Christian Brenninkmeijer, Simon Davidson, Donal Fellows, Andrew Gait, David R. Lester, Mantas Mikaitis, Luis A. Plana, Andrew G. D. Rowley, Alan B. Stokes, and Steve B. Furber. sPyN-Naker: A Software Package for Running PyNN Simulations on SpiNNaker. *Frontiers in Neuroscience*, 12(November), 2018. ISSN 1662-453X. doi: 10.3389/fnins.2018.00816. URL <https://www.frontiersin.org/article/10.3389/fnins.2018.00816/full>.
- [115] Oliver Rhodes, Luca Peres, Andrew G.D. D. Rowley, Andrew Gait, Luis A. Plana, Christian Brenninkmeijer, and Steve B. Furber. Real-time cortical simulation on neuromorphic hardware. *Phil. Trans. R. Soc. A*, 378:20190160, 2020. doi: 10.1098/rsta.2019.0160. URL <https://doi.org/10.1098/rsta.2019.0160>.

- [116] Nicolas Riche, Matthieu Duvinage, Matei Mancias, Bernard Gosselin, and Thierry Dutoit. Saliency and human fixations: State-of-the-art and study of comparison metrics. In *Proceedings of the IEEE international conference on computer vision*, pages 1153–1160, 2013.
- [117] Andrew G. D. Rowley, Christian Brenninkmeijer, Simon Davidson, Donal Fellows, Andrew Gait, David R. Lester, Luis A. Plana, Oliver Rhodes, Alan B. Stokes, and Steve B. Furber. Spinntools: The execution engine for the spinaker platform. *Frontiers in Neuroscience*, 13:231, 2019. ISSN 1662-453X. doi: 10.3389/fnins.2019.00231. URL <https://www.frontiersin.org/article/10.3389/fnins.2019.00231>.
- [118] Alexander F. Russell, Stefan Mihalas, Rudiger von der Heydt, Ernst Niebur, and Ralph Etienne-Cummings. A model of proto-object based saliency. *Vision Research*, 94:1–15, 2014. ISSN 00426989. doi: 10.1016/j.visres.2013.10.005.
- [119] Ueli Rutishauser, Dirk Walther, Christof Koch, and Pietro Perona. Is bottom-up attention useful for object recognition? In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–II. IEEE, 2004.
- [120] João Sacramento, Rui Ponte Costa, Yoshua Bengio, and Walter Senn. Dendritic cortical microcircuits approximate the backpropagation algorithm. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31, pages 8721–8732. Curran Associates, Inc., 2018. URL <https://proceedings.neurips.cc/paper/2018/file/1dc3a89d0d440ba31729b0ba74b93a33-Paper.pdf>.
- [121] Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *CoRR*, abs/1402.1128, 2014. URL <http://arxiv.org/abs/1402.1128>.
- [122] Johannes Schemmel, Daniel Brüderle, Andreas Grübl, Matthias Hock, Karlheinz Meier, and Sebastian Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950, 2010. doi: 10.1109/ISCAS.2010.5536970.

- [123] Charles E Schroeder, Donald A Wilson, Thomas Radman, Helen Scharfman, and Peter Lakatos. Dynamics of active sensing and perceptual selection. *Current Opinion in Neurobiology*, 20(2):172–176, 2010. ISSN 0959-4388. doi: <https://doi.org/10.1016/j.conb.2010.02.010>. URL <https://www.sciencedirect.com/science/article/pii/S0959438810000322>. Cognitive neuroscience.
- [124] Robert Shapley and V Hugh Perry. Cat and monkey retinal ganglion cells and their visual functional roles. *Trends in Neurosciences*, 9:229–235, 1986.
- [125] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):60, Jul 2019. ISSN 2196-1115. doi: 10.1186/s40537-019-0197-0. URL <https://doi.org/10.1186/s40537-019-0197-0>.
- [126] Sumit Bam Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. pages 1412–1421, 2018. URL <http://papers.nips.cc/paper/7415-slayer-spike-layer-error-reassignment-in-time.pdf>.
- [127] BW Silverman. Density estimation for statistics and data analysis. 1986. doi: <https://doi.org/10.1201/9781315140919>.
- [128] Takuma Sonoda, Yudai Okabe, and Tiffany M Schmidt. Overlapping morphological and functional properties between m4 and m5 intrinsically photosensitive retinal ganglion cells. *Journal of Comparative Neurology*, 528(6):1028–1040, 2020.
- [129] Kirsty L. Spalding, Olaf Bergmann, Kanar Alkass, Samuel Bernard, Mehran Salehpour, Hagen B. Huttner, Emil Boström, Isabelle Westerlund, Céline Vial, Bruce A. Buchholz, Göran Possnert, Deborah C. Mash, Henrik Druid, and Jonas Frisén. Dynamics of hippocampal neurogenesis in adult humans. *Cell*, 153(6):1219–1227, 2013. ISSN 0092-8674. doi: <https://doi.org/10.1016/j.cell.2013.05.002>. URL <https://www.sciencedirect.com/science/article/pii/S0092867413005333>.
- [130] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, June 2002. ISSN 1063-6560. doi: 10.1162/106365602320169811.

- [131] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, April 2009. ISSN 1064-5462. doi: 10.1162/artl.2009.15.2.15202.
- [132] Robert Stickgold. Sleep-dependent memory consolidation. *Nature*, 437(7063):1272–1278, 2005.
- [133] Claes Strannegård, Herman Carlström, Niklas Engsner, Fredrik Mäkeläinen, Filip Slottnér Scholm, and Morteza Haghir Chehreghani. Lifelong learning starting from zero. In Patrick Hammer, Pulin Agrawal, Ben Goertzel, and Matthew Iklé, editors, *Artificial General Intelligence*, pages 188–197, Cham, 2019. Springer International Publishing. ISBN 978-3-030-27005-6.
- [134] S. P. Strong, Roland Koberle, Rob R. de Ruyter van Steveninck, and William Bialek. Entropy and Information in Neural Spike Trains. *Physical Review Letters*, 80(1):197–200, jan 1998. ISSN 0031-9007. doi: 10.1103/PhysRevLett.80.197. URL <https://journals.aps.org/prl/pdf/10.1103/PhysRevLett.80.197><https://link.aps.org/doi/10.1103/PhysRevE.69.056111><https://link.aps.org/doi/10.1103/PhysRevLett.80.197>.
- [135] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.
- [136] Michael A. Sutton and Erin M. Schuman. Dendritic protein synthesis, synaptic plasticity, and memory. *Cell*, 127(1):49–58, 2006. ISSN 0092-8674. doi: <https://doi.org/10.1016/j.cell.2006.09.014>. URL <https://www.sciencedirect.com/science/article/pii/S0092867406012062>.
- [137] Yujin Tang and David Ha. The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning, 2021. URL <https://arxiv.org/abs/2109.02869>.
- [138] Simon Thorpe and Jacques Gautrais. *Rank Order Coding*, pages 113–118. Springer US, Boston, MA, 1998. doi: 10.1007/978-1-4615-4831-7\_19. URL [https://doi.org/10.1007/978-1-4615-4831-7\\_19](https://doi.org/10.1007/978-1-4615-4831-7_19).
- [139] Sebastian Thrun and Lorien Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

- [140] Anne M Treisman and Garry Gelade. A feature-integration theory of attention. *Cognitive psychology*, 12(1):97–136, 1980.
- [141] John K Tsotsos and Albert Rothenstein. Computational models of visual attention. *Scholarpedia*, 6(1):6201, 2011.
- [142] Takeshi Uejima, Ernst Niebur, and Ralph Etienne-Cummings. Proto-object based saliency model with second-order texture feature. In *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4. IEEE, 2018.
- [143] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. 30, 2017. URL <https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf>.
- [144] Dirk Walther and Christof Koch. Modeling attention to salient proto-objects. *Neural networks*, 19(9):1395–1407, 2006.
- [145] Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dhharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. 2016.
- [146] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014. doi: <https://doi.org/10.48550/arXiv.1410.3916>.
- [147] Cesco Willemse and Agnieszka Wykowska. In natural interaction with embodied robots, we prefer it when they follow our gaze: A gaze-contingent mobile eyetracking study. *Philosophical Transactions of the Royal Society B*, 374(1771):20180036, 2019.
- [148] Timo C. Wunderlich and Christian Pehle. Eventprop: Backpropagation for exact gradients in spiking neural networks. 2020.
- [149] Agnieszka Wykowska and Anna Schubö. On the temporal relation of top–down and bottom–up mechanisms during guidance of attention. *Journal of Cognitive Neuroscience*, 22(4):640–654, 2010.
- [150] Alfred L Yarbus. *Eye movements and vision*. Springer, 2013.



- [151] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural Computation*, 30(6):1514–1541, 2018. doi: 10.1162/neco\_a\_01086. URL [https://doi.org/10.1162/neco\\_a\\_01086](https://doi.org/10.1162/neco_a_01086). PMID: 29652587.
- [152] Friedemann Zenke, Ben Poole, and Surya Ganguli. Improved multitask learning through synaptic intelligence. *CoRR*, abs/1703.04200, 2017. URL <http://arxiv.org/abs/1703.04200>.
- [153] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Antonio Torralba, and Aude Oliva. Places: An image database for deep scene understanding. *CoRR*, abs/1610.02055, 2016. doi: <https://doi.org/10.48550/arXiv.1610.02055>. URL <http://arxiv.org/abs/1610.02055>.
- [154] Ding-Xuan Zhou. Universality of deep convolutional neural networks. *Applied and Computational Harmonic Analysis*, 48(2):787–794, 2020. ISSN 1063-5203. doi: <https://doi.org/10.1016/j.acha.2019.06.004>. URL <https://www.sciencedirect.com/science/article/pii/S1063520318302045>.
- [155] Hong Zhou, Howard S Friedman, and Rüdiger von der Heydt. Coding of border ownership in monkey visual cortex. *The Journal of Neuroscience*, 20(17):6594–6611, 2000.

# Appendix A

## Appendix

### A.1 Error Driven Neurogenesis appendix

The following figures are additional plots of the EDN (Error Driven Neurogenesis) parametric analysis. They explore the effect error threshold  $E_{th}$ , kernel spread  $s$ , random sampling and sampling with a varied surprise threshold  $s_{th}$  have on testing accuracy, absolute error, neuron count and synapse count.

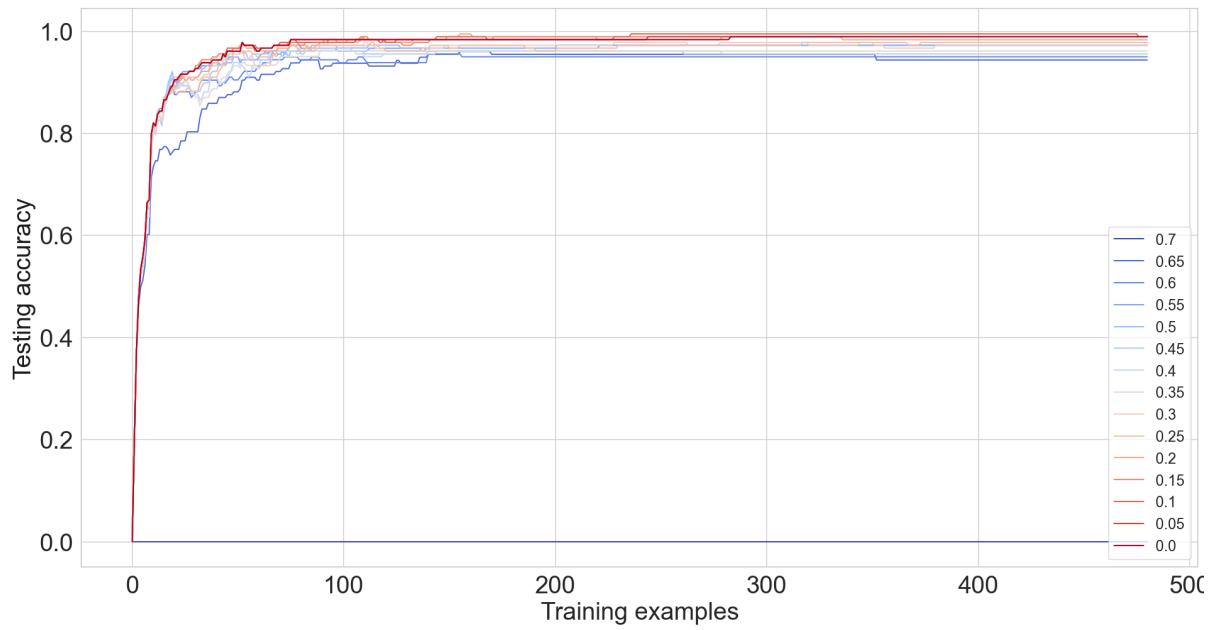


Figure A.1: How the testing accuracy during training changes with error threshold,  $E_{th}$

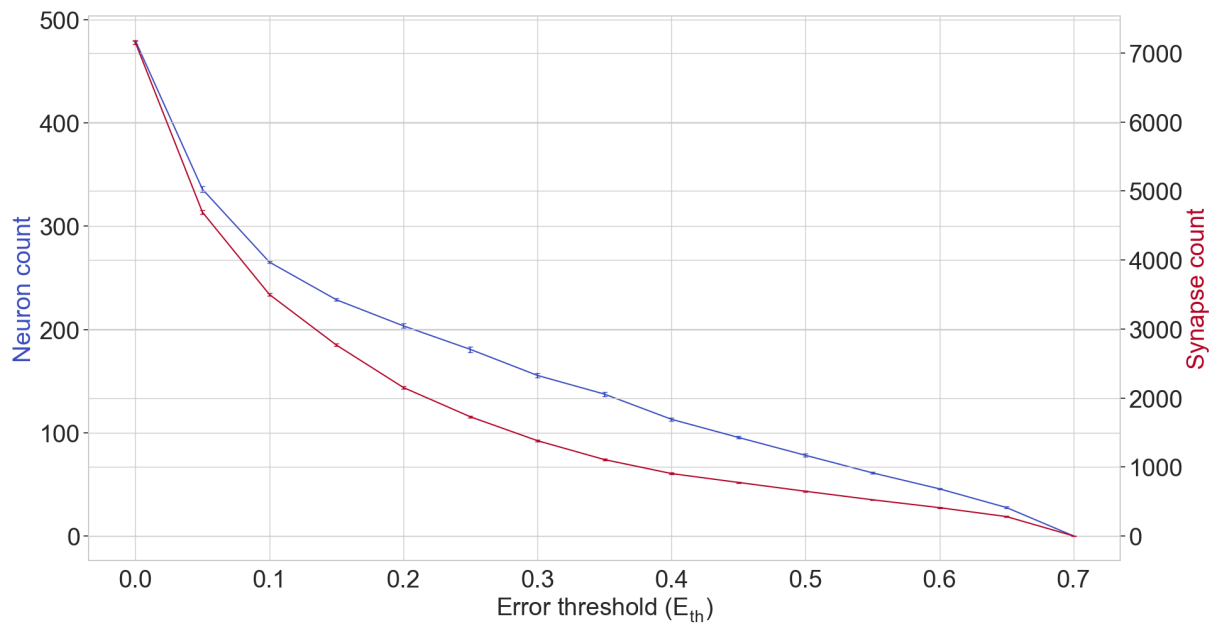


Figure A.2: How the neuron and synapse counts following two epochs changes with error threshold,  $E_{th}$

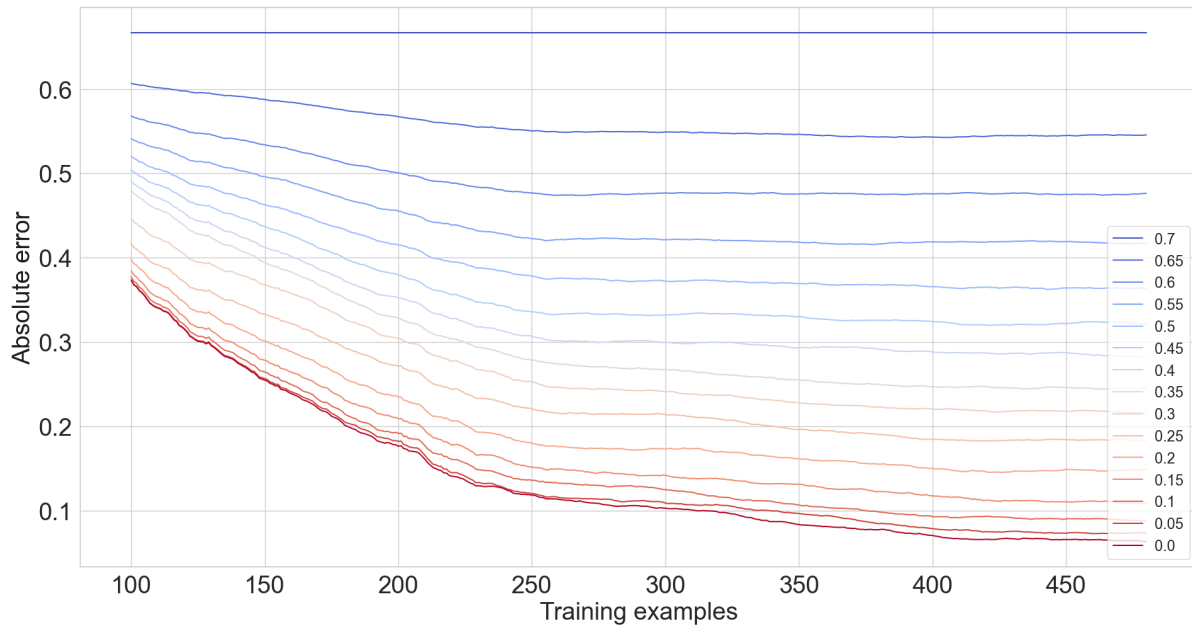


Figure A.3: How the running average (moving window over 100 examples) of absolute testing error changes with error threshold,  $E_{th}$

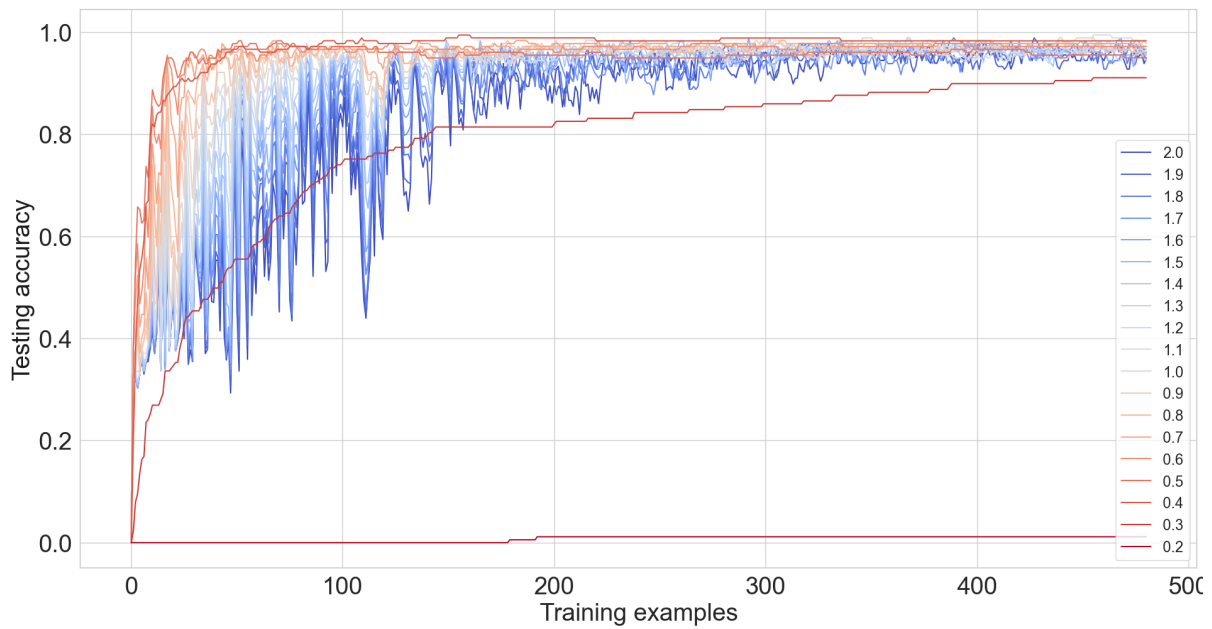


Figure A.4: How the testing accuracy during training changes with kernel spread size,  $s$

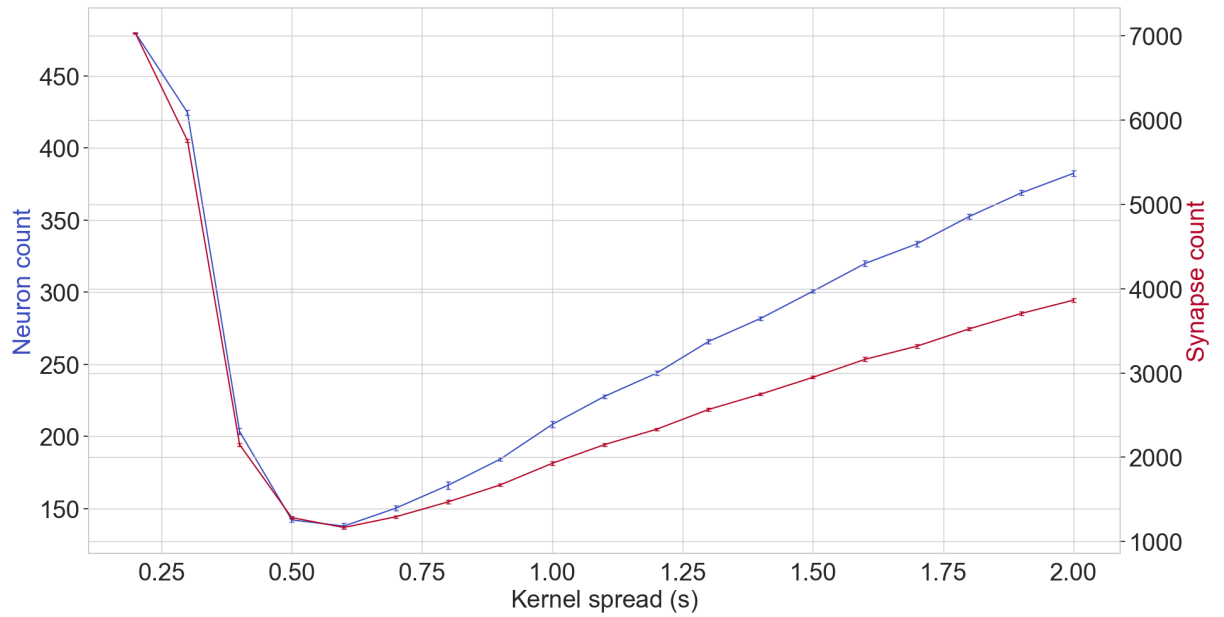


Figure A.5: How the neuron and synapse counts following two epochs changes with kernel spread size,  $s$

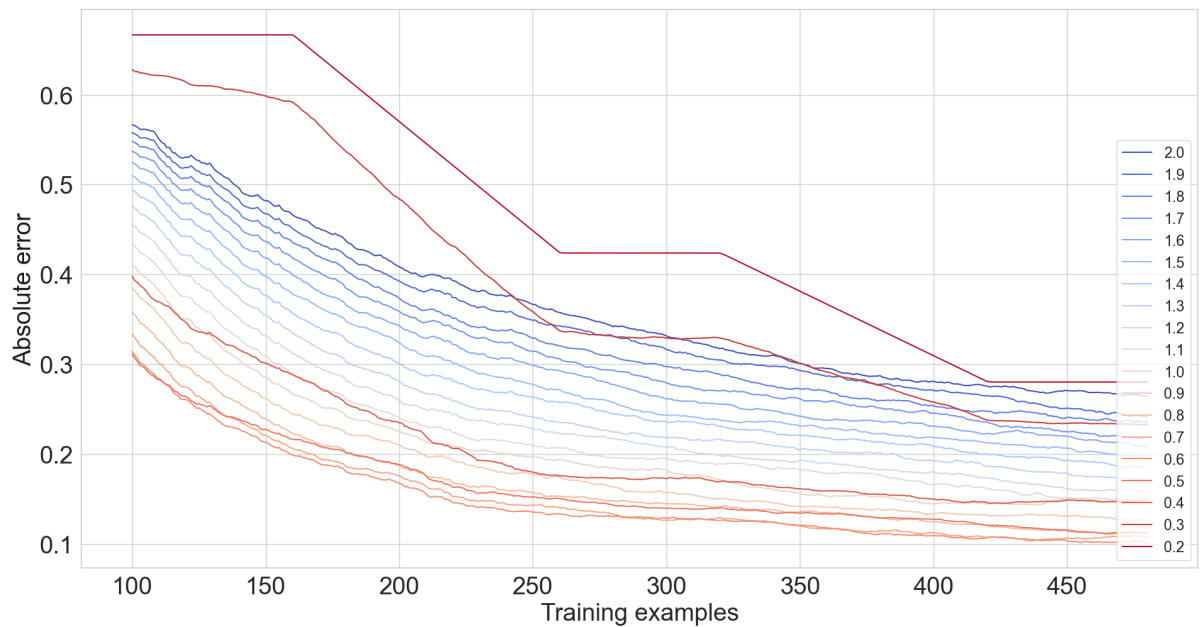


Figure A.6: How the running average (moving window over 100 examples) of absolute testing error changes with kernel spread size,  $s$

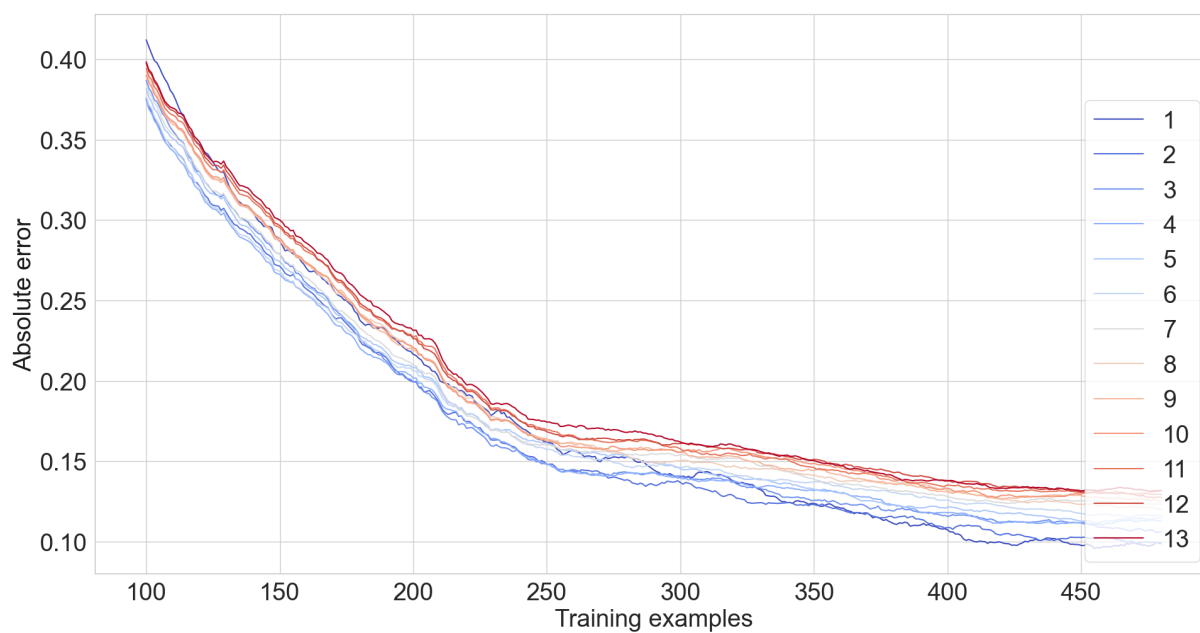


Figure A.7: How the testing accuracy during training changes with different random sample sizes

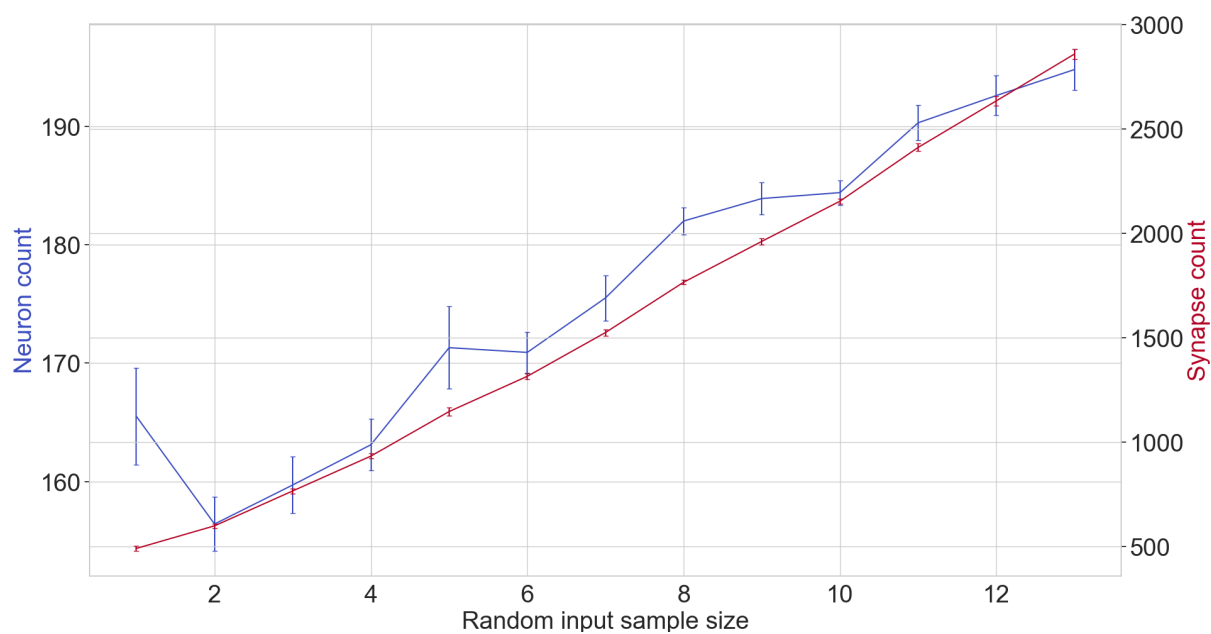


Figure A.8: How the neuron and synapse counts following two epochs changes with different random sample sizes

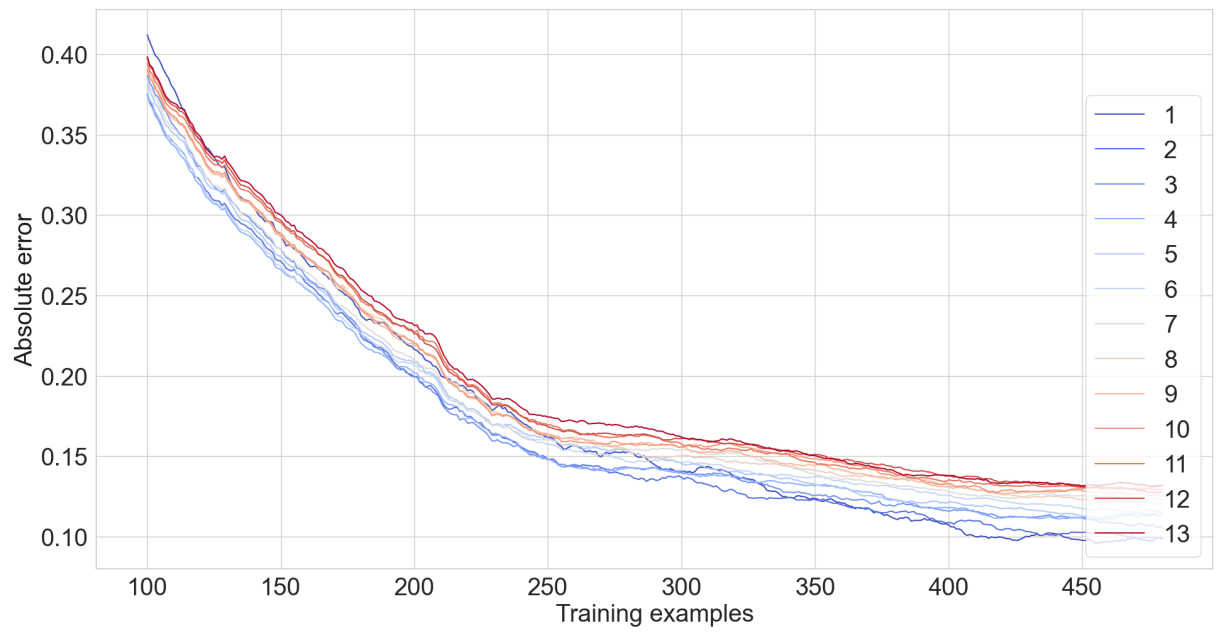


Figure A.9: How the running average of absolute testing error changes with different random sample sizes

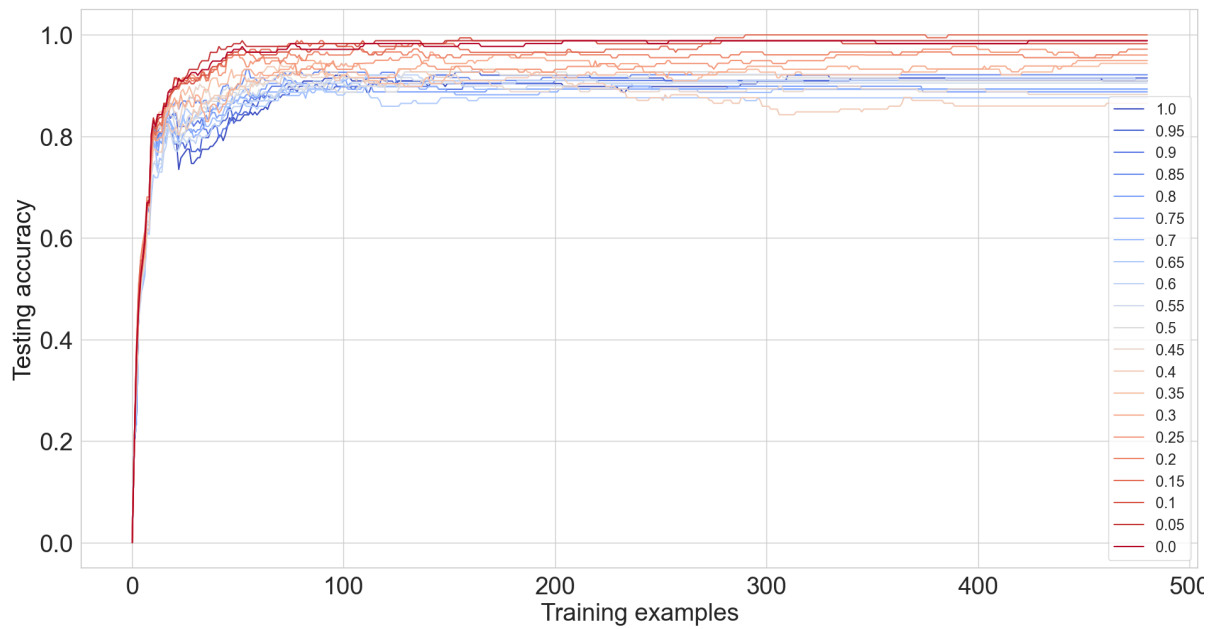


Figure A.10: How the testing accuracy during training changes with surprise threshold,  $s_{th}$

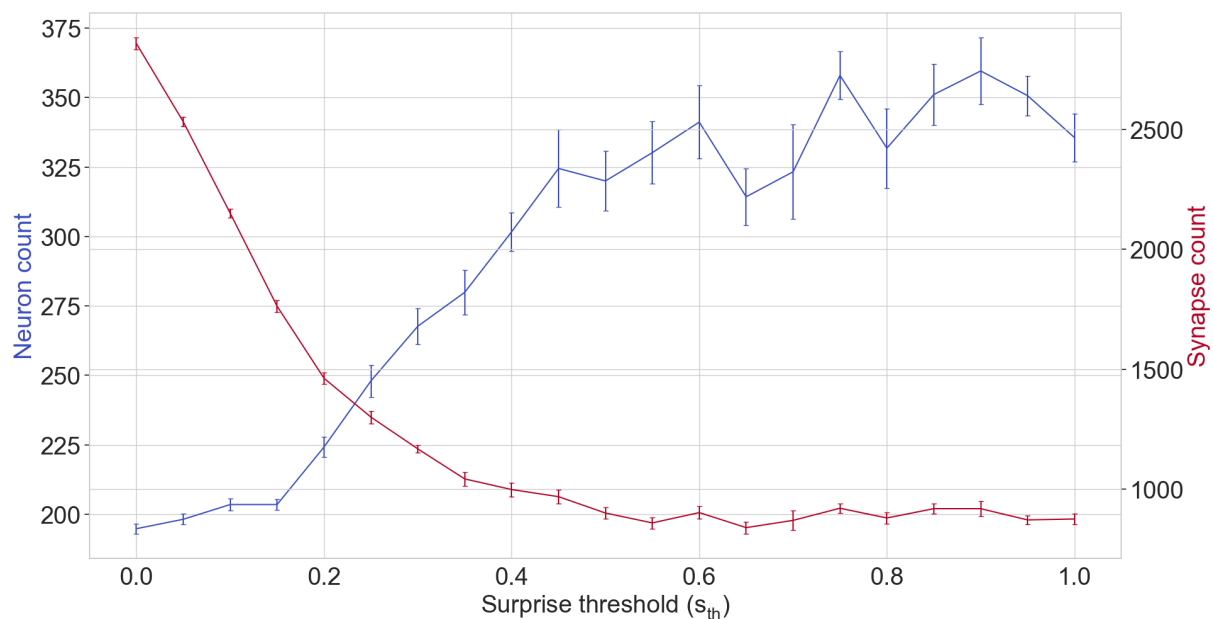


Figure A.11: How the neuron and synapse counts following two epochs changes with surprise threshold,  $s_{th}$

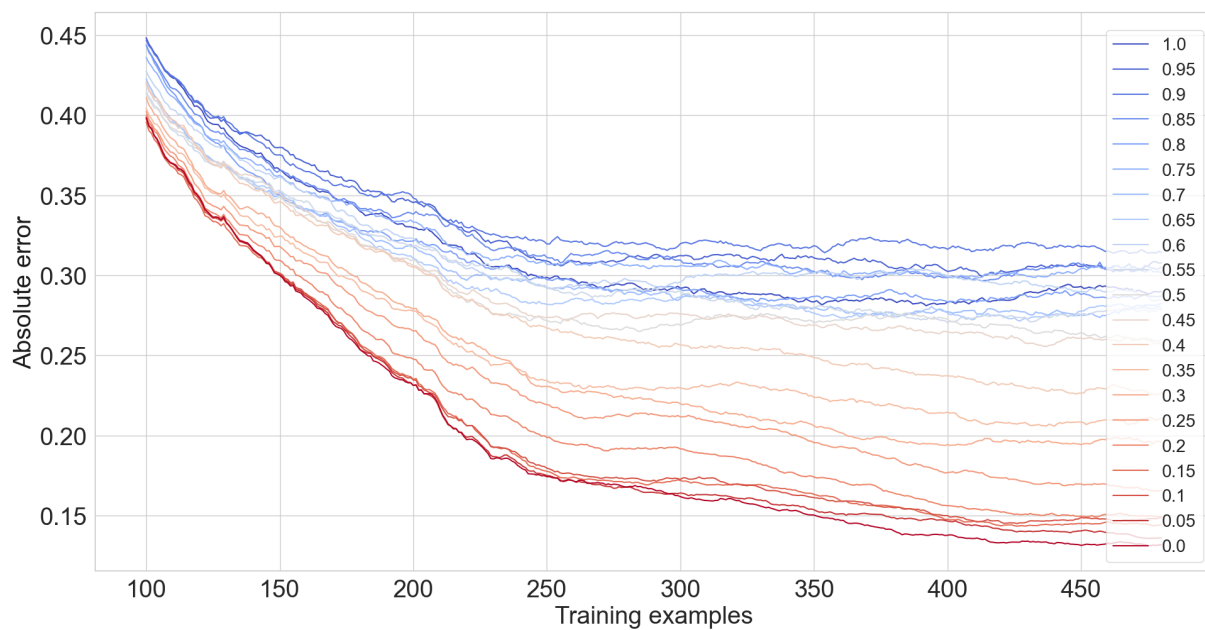


Figure A.12: How the running average (moving window over 100 examples) of absolute testing error changes with surprise threshold,  $s_{th}$