Enhancing Learning in Sparse Neural Networks: A Hebbian Learning Approach

Alexander de Ranitz
alexanderderanitz@gmail.com
University of Twente

***Abstract*—Artificial neural networks have proven to be capable of mastering many complex tasks. However, training such networks can be extremely resource intensive. In this research, the learning rule of a neural network trained using sparse evolutionary training (SET) is extended based on Hebbian theory. A mathematical formulation of Hebbian theory, encompassing inhibitory neurons and tailored for artificial neural networks, is proposed. The resulting novel algorithm, referred to as HebbSET, exhibits enhanced performance in terms of learning speed and accuracy on two datasets, with particularly notable improvements observed on the MNIST digit recognition dataset. These findings underscore the potential of incorporating neuroscientific theories to enhance the capabilities of artificial neural networks and bridge the gap between neuroscience and AI.**

***Keywords*—Artificial Neural Networks, Dynamic Sparse Training, Hebbian Learning, Sparse Neural Networks**

I. Introduction

Over the last few years, artificial neural networks (ANNs) have shown to be able to solve many interesting and complex problems, ranging from image recognition to playing Go at a super-human level [1][2]. Most state-of-the-art ANNs learn from a set of data using backpropagation and gradient descent [3]. Furthermore, the vast majority of these networks use a fully-connected architecture, in which each neuron is connected to every neuron in the next layer, leading to the number of parameters increasing quadratically with network size. Between 2012 and 2018, larger and larger ANN models were created, leading to the number of computations needed to train the largest AI models doubling every 3.4 months on average, displaying a 300,000-fold increase in computational load in less than six years [4]. Due to these high computational demands, training such large ANNs leads to high resource and energy consumption. For example, the training of GPT-3, a large language model with 175 billion parameters, consumed approximately 1,287 MWh of energy [5] and cost approximately 4.6 million USD [6].

Research has shown that trained ANNs tend to have a weight distribution centred around 0, indicating that a significant number of connections is not meaningfully contributing to the output of the network [7]. In order to reduce the number of (possibly unnecessary) parameters and thus the computational load of ANNs, sparse neural networks can be used. Contrary to fully-connected neural networks, in sparse neural networks, each neuron is connected to only a subset of the neurons in the following layer. Sparse neural networks are able to reach performance similar to their dense counterparts while having significantly fewer parameters [8]. There are two main approaches to creating sparse neural networks: dense-to-sparse training methods [7][9][10], in which an ANN is initialised with a dense architecture and unimportant weights are pruned during or after training, and sparse-to-sparse training [11][12], in which the ANN is initialised with an already sparse topology. Within sparse-to-sparse training methods, a further subdivision can be made between static methods [13], in which the topology of the network remains unchanged, and dynamic methods, in which the connections of the ANN can change during training. See [14] for an overview of sparse training methods.

Artificial neural networks were originally inspired by the human brain. However, there are significant differences between ANNs and biological neural networks (BNNs). For example, in the structure of how neurons are connected. Some BNNs have been shown to display small-worldness and a scale-free topology, yet these characteristics are absent in fully-connected ANNs [15][16]. Another aspect in which ANNs significantly differ from BNNs is the way they learn. A famous concept in neuroscience is Hebb's postulate: *"When an axon of cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased"* [17]. This concept, often

referred to as Hebbian learning, is fundamental to our understanding of how the human brain learns. However, modern ANNs that learn using gradient descent methods do not (explicitly) incorporate this idea.

Nonetheless, there are several reasons why incorporating theories and findings from neuroscience in ANNs could be beneficial [18]. In the first place, implementing mechanisms and processes theorised to take place in the human brain into ANNs could help create more powerful AI models. Considering that the human brain is currently the most generally intelligent system known, creating AI systems that function in a similar fashion is perhaps the most plausible way to reach such levels of intelligence [19]. Secondly, if ANNs are created that somewhat accurately model the behaviour of the human brain, these can be used to gain a deeper understanding of how the human brain functions and how intelligence emerges from relatively simple neural substrates.

In this paper, it will be investigated whether Hebbian learning can be used to enhance learning in sparse neural networks. To this end, a first-of-its-kind Hebbian learning rule for training dynamic sparse neural networks is introduced. Specifically, the following three questions are posed:

1) How does the use of Hebbian learning in dynamic sparse training affect the manner in which weights are updated during training?
2) What effect does the use of Hebbian learning in dynamic sparse training have on the topology of the resulting neural networks?
3) Can Hebbian learning be used to enhance the performance of sparse neural networks in terms of classification accuracy and learning speed?

II. Background

A. Artificial Neural Networks and the Brain

Artificial neural networks were originally inspired by the brain. Both consist of a number of neurons that are connected to each other in some manner. ANNs consist of several layers of neurons: an input layer, an output layer, and zero or more hidden layers in between. Neurons in consecutive layers of an ANN are connected through weights. In fully-connected ANNs, each neuron is connected to every neuron in the following layer. A schematic overview of a simple ANN is depicted in Figure 1.

Each neuron produces a certain output depending on the inputs it receives from other neurons. When considering two neurons in consecutive layers that are connected, the neuron that receives the input is called the postsynaptic neuron, whereas the neuron that sends the signal is called the presynaptic neuron. This terminology originates from biological neural networks, in which the point where a signal is passed from one neuron to the next is called a synapse [20]. In this paper, the terms pre- and postsynaptic neuron will be used for both biological and artificial neural networks.

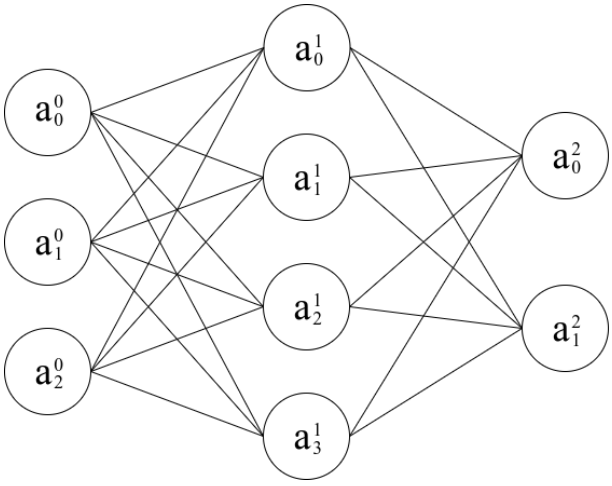

Fig. 1: Example of a 3-layered fully-connected artificial neural network. Superscripts are used to denote the layer of the neuron, while subscripts are used to index neurons within a layer.

In biological neurons, the output of a neuron is binary: the neuron either fires or not. However, depending on the intensity of the incoming signal, the neuron might fire multiple times in a given window of time. As such, the activity of a biological neuron is best represented by its firing frequency, rather than the strength of firing. In artificial neurons, the output of a neuron is usually a single, continuous number, analogous to the firing rate of a biological neuron.

In ANNs, the output, or activation, of the j^{th} neuron in the l^{th} layer, a_j^l, is computed according to equation 1.

$$a_j^l = f\Big(\sum_i w_{ji} \cdot a_i^{l-1} + b_j^l\Big) \tag{1}$$

Here, w_{ji} is the weight of the connection between the i^{th} neuron in layer $l-1$ and the j^{th} neuron in layer l, and b_j^l is the bias of the neuron. The activation function, f, is a function applied to the total weighted input of the neuron, often a Sigmoid function or a Rectified Linear Unit (ReLU) [21] (see [22] for a deeper discussion and comparison of several common activation functions).

B. Learning in Neural Networks

Training ANNs on a set of data is typically done using gradient descent methods. Although several variations exist, they all aim to minimize the loss or cost of an ANN by iteratively adjusting its parameters based on the gradient of the loss function [23]. Given network parameters θ (i.e. the weights and biases), loss function $J(\theta)$, and learning rate α, the network parameters are updated as follows:

$$\theta \leftarrow \theta - \alpha \cdot \nabla J(\theta) \tag{2}$$

Here, the gradient of the loss function, $\nabla J(\theta)$, yields the direction in which the loss increases the fastest. Taking a step of a fixed small size in the opposite direction consequently reduces the loss optimally. This gradient consists of the partial derivative of each network parameter with respect to the loss function. Therefore, a single weight is updated according to Equation 3.

$$w_{ji}^{l} \leftarrow w_{ji}^{l} - \alpha \cdot \frac{\partial J}{\partial w_{ji}^{l}} \tag{3}$$

Note, however, that Equations 2 and 3 apply to a single training example. Therefore, the parameters might be updated in a manner that reduces the cost for one input, but increases the cost of many other inputs. In order to train the network on the whole dataset, the gradient can be computed for each training example and then averaged. This average gradient can then be used to update the network parameters. However, this method of gradient descent is computationally expensive [24]. Instead, mini-batch gradient descent is often used, in which the network parameters are updated based on a small batch of data, which provides a good balance between speed and effective learning [25].

C. Sparse Evolutionary Training

Sparse Evolutionary Training (SET) is a dynamic sparse training algorithm that aims to find a suitable sparse network topology for a given problem [12]. In SET, the neural network is initialised with a small number of randomly generated connections instead of a fully-connected topology. During training, a fixed percentage of the weights closest to 0 is periodically removed, and an equal number of connections are randomly added, keeping the total sparsity level constant. When viewing the weights of a network as a population of individuals, this process resembles evolution, where "weak" connections are removed (selection), and new connections are randomly generated (mutation). As the weights that are removed are close to 0, their removal has relatively little effect on the performance of the network. This process allows the network to evolve an effective topology and reach high performance with just a fraction of the parameters of a fully connected network.

Initially, the layers of the network are connected using Erdős Rényi random graphs [26]. In this random graph, the probability of a connection existing between the j^{th} neuron in layer l and the i^{th} neuron in layer $l-1$ is given by Equation 4.

$$p(w_{ji}^{l}) = \frac{\varepsilon(n^{l} + n^{l-1})}{n^{l} \cdot n^{l-1}} \tag{4}$$

Here, n^{l} gives the number of neurons in layer l, ε is a hyperparameter controlling the sparsity level of the network. For layers with a sufficient number of neurons, the number of connections scales linearly with the total number of neurons in the connected layers, as opposed to quadratically as seen in fully-connected neural networks.

Pseudocode for the implementation of SET used in this research can be seen in Algorithm 1[1].

Algorithm 1 SET Pseudocode

```
set ζ, max_epochs
initialise layers using Erdős Rényi graphs
epoch ← 0
while epoch < max_epochs do
    for batch in data do
        run network for each sample in batch
        compute average gradient
        update neural network parameters
    end for
    remove fraction ζ of weights closest to 0
    if epoch < max_epochs - 1 then
        add an equal number of random weights
    end if
    epoch ← epoch + 1
end while
```

III. Hebbian Learning

Although Hebb's postulate has proven to be invaluable, its original formulation is rather incomplete and imprecise. Hebb's original proposal only stated under what conditions synaptic weights should increase [17]. Clearly, a learning rule that allows only for increasing weights is not particularly useful. As such, Hebb's postulate must be extended to incorporate weight decreases

[1]The implementation of SET used in this paper was adapted from https://github.com/dcmocanu/sparse-evolutionary-artificial-neural-networks/tree/master [12]

and be appropriately quantified before it can be used to train ANNs.

In both biological and artificial neural networks, neurons can send two types of signals: excitatory and inhibitory. Excitatory signals cause the postsynaptic neuron to be more likely to fire, while inhibitory signals reduce the activity of the postsynaptic neurons. In biological neural networks, the type of neurotransmitter(s) involved determines whether a signal is excitatory or inhibitory [20]. In ANNs, when using activation functions that output only positive values, the type of signal depends on the magnitude of the connection weight: positive weights transfer excitatory signals, while negative weights transfer inhibitory signals. Hebb's postulate only states the conditions under which an excitatory weight should increase. However, it can easily be extended to inhibitory neurons by replacing *"firing"* with *"suppressing"* in Hebb's postulate. Together, these two statements can be summarised as follows:

1) If presynaptic neuron A is active and postsynaptic neuron B is active, A's efficiency, as one of the neurons firing B, should be increased (i.e. the weight between A and B should increase in ANNs).
2) If presynaptic neuron A is active and postsynaptic neuron B is inactive (suppressed), A's efficiency, as one of the neurons suppressing B, should be increased (i.e. the weight between A and B should decrease in ANNs).

It is important to note that the two conditions above not only dictate when excitatory or inhibitory connections should be strengthened, but also provide the conditions for when connections should weaken. If presynaptic neuron A has an excitatory connection to postsynaptic neuron B (i.e. a positive weight in ANNs) but neuron A is active whilst neuron B is inactive, the weight between A and B should decrease per the second condition described above. Conversely, if a connection between two neurons is inhibitory (i.e. a negative weight in ANNs) but both neurons are simultaneously active, the first condition dictates that the weight should increase, weakening the inhibitory connection. These two conditions, although a relatively simple extension of Hebb's original statement, thus provide sufficient details to apply Hebbian learning to ANNs.

A. Mathematical Formulation of Hebbian Learning

Having qualitatively described how Hebb's postulate can be used to strengthen and weaken connections in ANNs, these conditions need to be quantified before they can be used to train ANNs.

As neural activity is usually a purely positive, continuous quantity in ANNs, it can be hard to determine whether a neuron is active or inactive based solely on its current activation. Therefore, the average activation of a postsynaptic neuron can be used to determine whether a given activation value should be considered active or inactive. If the current activation is above average, that neuron is said to be active, whereas it is inactive when the current activation is below average. Thus, when subtracting the average activation from the current activation, a positive result indicates that the neuron is active, while a negative value indicates that the neuron is inactive. The sign of this measure of activity corresponds with the expected sign of the change in synaptic strength:

$$sign(\Delta w_{ji}^{l}) = sign(a_j^l - \overline{a_j^l}) \tag{5}$$

with $\overline{a_j^l}$ representing the average activation of the postsynaptic neuron over a given time period.

Furthermore, the activity of the presynaptic neuron should be included in the weight update. In Hebb's postulate, a strength increase takes place when the presynaptic neuron *takes part* in firing the postsynaptic neuron, indicating that the presynaptic neuron itself must be active before the synaptic weight can increase- there must be cooperation. This also implies causality: the presynaptic neuron should not simply fire together with the postsynaptic neuron, but it should fire before it. This causality is naturally present in artificial neural networks, as signals flow only in one direction, from the input layer toward the output layer. This requirement of cooperation and causality yields:

$$a_i^{l-1} = 0 \Rightarrow \Delta w_{ji}^{l} = 0 \tag{6}$$

Furthermore, it naturally follows that as the activity of the presynaptic neuron increases, so should the magnitude of the weight update. Combining these insights results in the following equation:

$$\Delta w_{ji}^{l} = a_i^{l-1} \cdot (a_j^l - \overline{a_j^l}) \tag{7}$$

This equation fulfils all requirements previously set. However, it should be noted that many other formulations of Hebbian learning are possible (see [27]). This specific formulation was chosen for its simplicity and flexibility.

The learning rule described above, and Hebbian learning in general, is not sufficient to train a neural network to recognize images or predict values, as this

learning rule has no knowledge of the expected output value associated with each input. Therefore, it must be combined with traditional learning rules, such as mini-batch gradient descent, to be used to train neural networks. However, in this paper, it is hypothesised that incorporating this Hebbian learning term can improve the performance of neural networks by aiding in strengthening important weights and reducing unimportant weights, which in turn improves the topology of the network when using SET.

IV. Hebbian-Inspired Sparse Evolutionary Training

In this paper, based on our best knowledge, a first-of-its-kind Hebbian-inspired algorithm is introduced for dynamic sparse training in deep neural networks. This algorithm, named HebbSET, extends the standard mini-batch gradient descent method used in SET based on Hebbian learning.

In HebbSET, each weight is updated by combining regular gradient descent with a Hebbian learning term (as described in Equation 3 and Equation 7, respectively). This yields the following equation:

$$\Delta w_{ji}^{l} = -\alpha \cdot \frac{\partial J}{\partial w_{ji}^{l}} + a_i^{l-1} \cdot (a_j^l - \overline{a_j^l}) \cdot \lambda(t) \quad (8)$$

Here, the left part of the equation, $-\alpha \cdot \frac{\partial J}{\partial w_{ji}^{l}}$, corresponds with standard mini-batch gradient descent, whereas the right part of the equation, $a_i^{l-1} \cdot (a_j^l - \overline{a_j^l}) \cdot \lambda(t)$, consists of the Hebbian learning term multiplied by a time-dependent hyperparameter λ that controls the strength of this Hebbian learning term. Note that, since mini-batch gradient descent is used, Equation 8 is computed for each sample in one mini-batch of data and averaged before being applied to the weights.

In order to compute the weight update described in Equation 8, the average activation $\overline{a_j^l}$ needs to be known for each neuron. This average is taken using the activations computed on the current mini-batch of data. Furthermore, in order to ensure numerical stability and allow Equation 8 to be used for a range of activation functions, the activations of each neuron are scaled to a range of $[0, 1]$ prior to computing Equation 8. For a batch of data of size n, some neuron a_i^l produces a vector of n activations named $\mathbf{a}_i^l$.

$$\mathbf{a}_i^l = [a_i^l(0), a_i^l(1), ..., a_i^l(n)] \quad (9)$$

This vector is then normalised to $[0, 1]$ using Equation 10. Thus, for a given batch of data, each neuron has a minimum activation of 0 and a maximum activation of 1.

$$\mathbf{a}_i^l = \frac{\mathbf{a}_i^l - min(\mathbf{a}_i^l)}{max(\mathbf{a}_i^l) - min(\mathbf{a}_i^l)} \quad (10)$$

The pseudocode for HebbSET is presented in Algorithm 2. Additions to regular SET as presented in Algorithm 1 are marked in blue.

HebbSET is somewhat more computationally intensive than SET, due to the calculations required to scale the activations and compute the Hebbian learning term. The number of additional calculations scales linearly with the number of weights in the network. However, since the required calculations are rather simple, HebbSET is still able to scale to extremely large networks, similar to SET.

Algorithm 2 Proposed Method- HebbSET

```
set hyperparameters
initialise layers as Erdős Rényi random graphs
epoch ← 0
while epoch < max_epochs do
    for batch in data do
        run network for each sample in batch
        compute average gradient
        compute and scale average activations
        compute Hebbian learning term
        update neural network parameters
    end for
    remove fraction ζ of weights closest to 0
    if epoch < max_epochs - 1 then
        add an equal number of random weights
    end if
    epoch ← epoch + 1
end while
```

V. Results

In this section, the performance of HebbSET is evaluated and compared to the performance of SET. Performance is assessed based on accuracy, loss, and speed of learning. Furthermore, the topology and connectivity patterns of the ANNs created using HebbSET and SET are analysed in order to better understand the differences between these two algorithms.

A. Dataset Description

To test the behaviour and performance of the proposed method, two datasets are used.

1) *MNIST dataset* [28]. The MNIST dataset is a very popular dataset for testing machine learning

Dataset	Data Type	Features	Classes	Used Samples [% of total]	Network Architecture	Sparsity [%]	Activation functions
Lung	Real Values	3312	5	203 (100%)	3312-2000-2000-2000-5	98.1	ReLu-ReLu-Relu-Sigmoid
MNIST	Grayscale Pixels	784	10	2400 (4%)	784-1000-1000-1000-10	95.6	ReLu-ReLu-Relu-Sigmoid

TABLE I: Description of datasets and neural networks used. For the MNIST dataset, a random subset of all data was used in order to speed up experiments. The sparsity level indicates what percentage of weights are missing compared to a fully-connected ANN with an identical architecture.

algorithms. It consists of grayscale images of handwritten digits with a size of 28 by 28 pixels. Each image contains one digit (0-9) portrayed on a black background. Thus, images are classified into one of ten classes.

2) *Lung dataset* [29]. This dataset consists of 203 samples from lung tumours (n=186) and normal lung tissue (n=17). Samples are divided into five categories: four different types of tumours and normal lung specimens [30]. Each sample consists of 3312 real values.

The data in both datasets are scaled to a range of $[0, 1]$. Furthermore, both datasets are split into a training and test set consisting of 75% and 25% of the samples respectively. Further information regarding the used datasets and employed networks can be found in Table I.

B. Implementation Details

The strength of the Hebbian learning term in the weight update procedure of HebbSET, as presented in Equation 8, is controlled by the parameter $\lambda(t)$. To balance exploration and exploitation, the hyperparameter λ is reduced over time. By starting training with a large value of λ, the Hebbian learning term can have a strong influence on which connections are strengthened and, therefore, kept. However, as the network trains, fine-tuning existing weights, which can be seen as exploitation, becomes more important, which is done through gradient descent. Therefore, the value of λ is reduced over time as follows:

$$\lambda(t) = \lambda_0 \cdot 0.98^t \tag{11}$$

with λ_0 being the initial value for λ and t being the current epoch. Preliminary experiments indicated that a value of 0.98 yielded the best results.

For each dataset, several values of λ_0 are used to gain a deeper understanding of the effect of the Hebbian learning term on the neural network. All other hyperparameters were kept equal throughout all trials, except for the network size and batch size, which differed depending on the dataset used. The network sizes can be seen in Table I and batch sizes were set to 30 and 200 for the Lung and MNIST dataset, respectively. The learning rate α was set to 0.02, the fraction of weights removed ζ per epoch was 0.30, the hyperparameter controlling the sparsity level ε was set to 20, and the networks were trained for 250 epochs using the Mean Square Error loss function.

C. Experimental Results

The acquired accuracy and test loss of the neural networks are presented in Table II and visualised in Figure 2. For the MNIST dataset, HebbSET outperforms SET for all values of λ_0. HebbSET not only shows faster initial learning, but the final accuracy reached is also significantly increased. The learning speed in very early epochs seems to increase as λ_0 increases. However, for the largest λ_0 value of 0.025, the network performance becomes less stable and after approximately 30 epochs, learning slows down significantly and accuracy even temporarily drops.

For the Lung dataset, similar results are obtained, but the differences between SET and HebbSET are much less pronounced. HebbSET still displays increased learning speed, with HebbSET with a λ_0 value of 0.025 yielding the highest accuracy out of all models at epoch 50. However, just like on the MNIST dataset, learning slows down after approximately 30 epochs for this value of λ_0. For the vast majority of training, HebbSET with a λ_0 value of 0.01 has a slight edge over SET in terms of accuracy and loss, although this advantage seems to disappear in the very final epochs.

Based on these results, a λ_0 of 0.01 is used for further experiments, as it performs well in terms of both accuracy and test loss and is less noisy compared to higher values of λ_0. However, further tuning of this parameter and how it is changed over time will likely yield improved results.

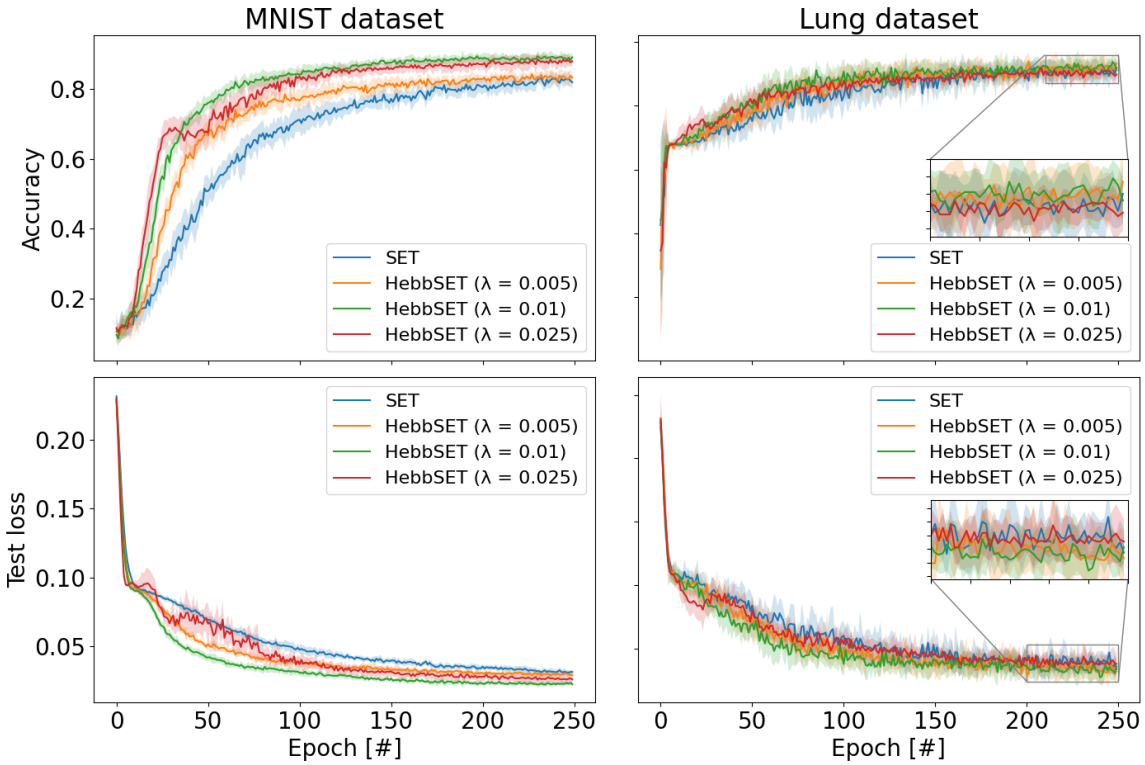

Fig. 2: Accuracy and loss over time of SET and HebbSET on the test data of the MNIST and Lung dataset for various values of λ_0. Each neural network was trained for ten iterations. The solid lines show the mean values obtained, whereas the shaded regions indicate the standard deviation.

Model	λ_0	Epoch [#]	MNIST		Lung	
			Accuracy [%]	Loss	Accuracy [%]	Loss
SET	-	50	51 (±4.2)	0.069 (±0.0025)	75 (±4.0)	0.084 (±0.008)
		249	82 (±1.5)	0.031 (±0.0016)	92 (±3.5)	0.033 (±0.009)
HebbSET	0.005	50	66 (±3.9)	0.052 (±0.0041)	78 (±6.0)	0.074 (±0.019)
		249	83 (±1.3)	0.029 (±0.0022)	93 (±4.0)	0.031 (±0.010)
	0.01	50	76 (±2.0)	0.042 (±0.0024)	79 (±3.8)	0.070 (±0.011)
		249	89 (±1.5)	0.022 (±0.0015)	91 (±2.7)	0.036 (±0.004)
	0.025	50	67 (±3.8)	0.066 (±0.0145)	81 (±3.2)	0.080 (±0.010)
		249	88 (±1.5)	0.025 (±0.0024)	89 (±2.1)	0.038 (±0.005)

TABLE II: Summary of test results on the MNIST and Lung datasets. Each neural network was trained for ten iterations. This table shows the mean accuracy and loss obtained, with the standard deviation given between brackets. Results are displayed for epoch 50 and for the final epoch of training.

D. Effect of Hebbian Term on Weight Update

In section IV, a weight update rule was proposed in order to modify weights based on Hebb's postulate. In Figure 3, the weight update for a single training example, as computed by Equation 8, is plotted against the pre- and postsynaptic activation of the two neurons connected by that weight. The data plotted are for a single, randomly chosen weight between a neuron in the second and a neuron in the third layer of the ANN.

Figure 3 shows that when the presynaptic activation is zero, the weight update is zero too. This is expected based on Equation 8, as both the Hebbian term and the gradient descent term are by definition zero when the presynaptic activation is zero. Furthermore, it can be seen that the weight update is largest when both the pre- and postsynaptic activations are large, which is exactly as expected based on Hebb's postulate. Similarly, the

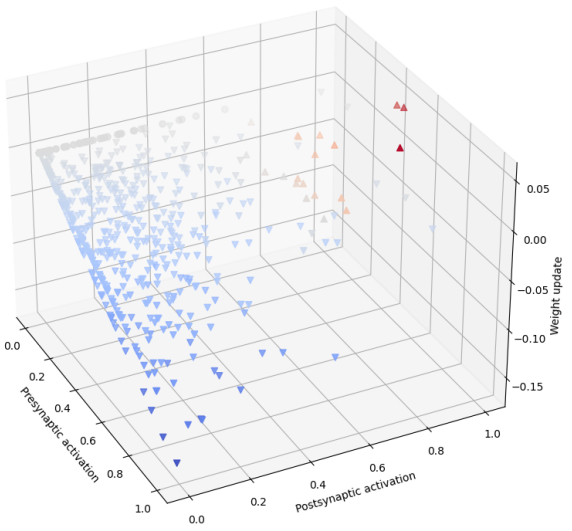

Fig. 3: Weight update plotted against the pre- and postsynaptic activations of two randomly picked neurons for the first epoch. Each marker represents a single training example. To aid with interpretation, a negative weight update is indicated by a blue, downward-facing triangle, whereas weight increases are indicated by red, upward-facing triangles- with larger weight updates having a darker colour. Weight updates of zero are indicated by grey circles.

weight update is most strongly negative when the presynaptic activation is large but the postsynaptic activation is low (i.e. the postsynaptic neuron is suppressed). In conclusion, this figure shows that HebbSET successfully implements Hebbian learning in dynamic sparse training.

Furthermore, In Figure 4, the size of the gradient descent term and Hebbian learning term are depicted for the weight updates applied based on one batch of data. These two terms combined make up the total weight update applied to each weight, as described in Equation 8.

E. Network Topology

Although HebbSET only alters the weight update procedure of the neural network compared to regular SET, this also affects the way the network's topology evolves, due to connections being removed based on their magnitude in SET. Therefore, one way in which HebbSET might improve the network's performance is by aiding in evolving a topology that can effectively solve the given problem. In order to investigate this, the connectivity patterns of two networks trained using SET and HebbSET, respectively, are shown in Figure 5. These networks were trained on the MNIST digit dataset using identical hyperparameters (except for the λ_0 parameter in HebbSET, which was set to 0.01). It appears that HebbSET is better able to evolve a suitable network topology for this problem compared to regular SET. One possible reason for this is that input neurons whose activation tends to be large (i.e. neurons around the centre of the image in this case) have a higher likelihood to fire together with their postsynaptic neurons, simply due to their high activation being forwarded to the postsynaptic neuron. This increased likelihood to fire together leads to the connection weight being increased when using HebbSET. This, in turn, reduces the probability that this connection will be removed in the weight-pruning phase of SET. Consequently, input neurons that often have large activations end up having a larger number of connections.

Furthermore, Figure 6 shows how rapidly the topology of the network changes over time for each layer of weights. It should be noted that due to the small number of neurons in the output layer, the final layer has a sparsity level of just 10%, meaning that this layer has only slightly fewer connections than a fully-connected layer would. As such, the topology of this layer is relatively stable. In general, both networks change very rapidly in early epochs, with less than 20% of the initial connections remaining after 25 epochs. As training progresses, however, the network topology changes more slowly, and approximately 60% of connections persist over 25 epochs. In the final epoch, no random weights are added after the weight pruning step, which means that the final network has 30% fewer connections than it had during training (see Algorithm 2). In the last layer, this causes the large drop that is visible at the final epoch. No such drop is visible for the other layers, indicating that the weights that are removed in the final epoch were recently added, as out of the 70% of remaining weights, approximately 60% were already part of the network 25 epochs prior.

It appears that the topology of the ANN trained using HebbSET tends to stabilise faster than that of the network trained using SET. This finding is consistent with Figure 5, in which HebbSET is shown to quickly find a topology that fits the input data, after which the connectivity pattern stays relatively stable. Thus, these results suggest that the addition of the Hebbian learning term allows for an effective topology to be found more quickly, accelerating the learning process.

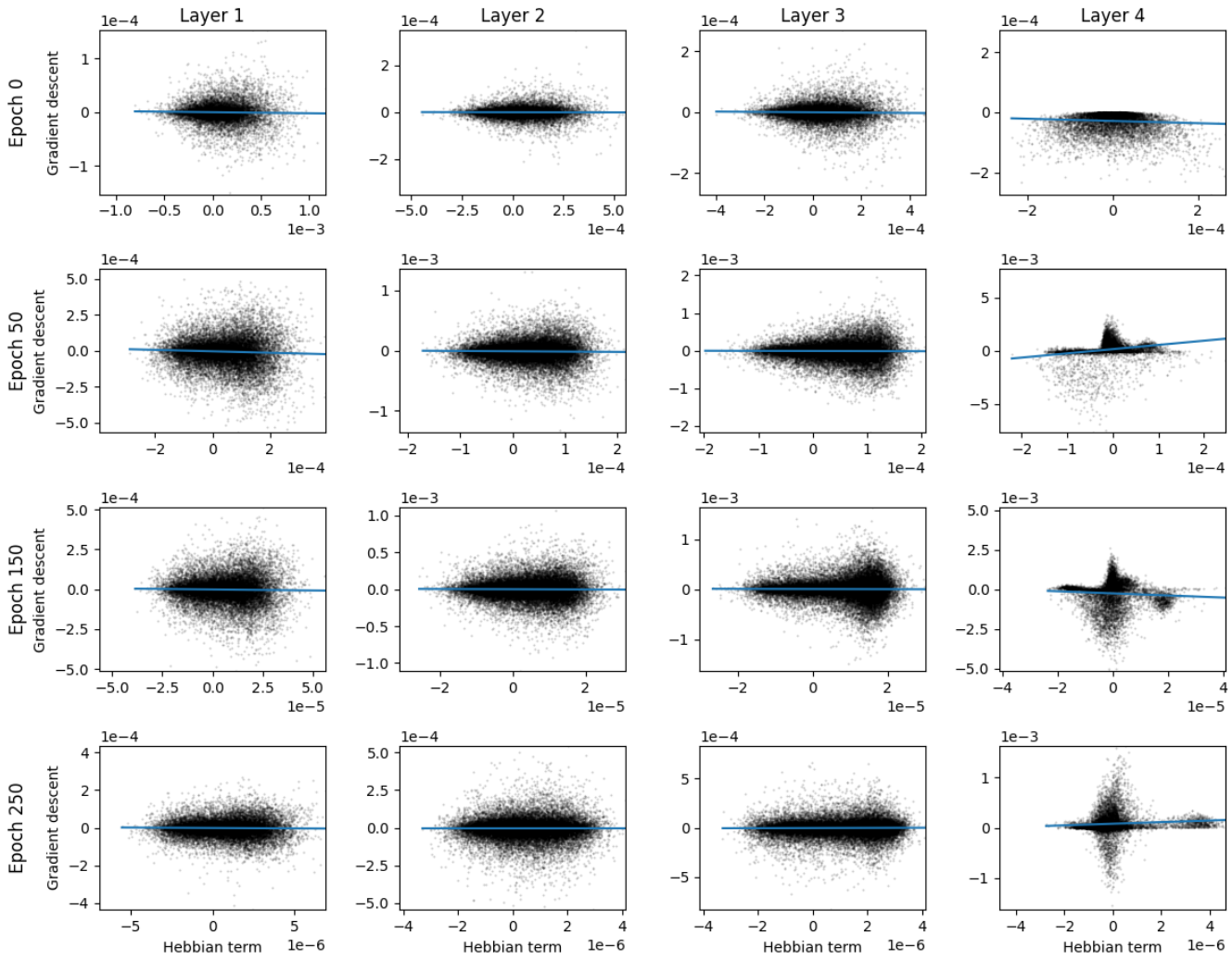

Fig. 4: Hebbian term versus gradient descent term plotted for each layer at several epochs. The blue line shows the line of best fit through the data. Each dot represents the weight update applied to a single weight based on one batch of data from the MNIST dataset. The size of the Hebbian learning term tends to decrease over time, as expected based on Equation 11.

VI. Conclusion and Future Work

In this paper, a novel learning algorithm called HebbSET was designed, taking inspiration from Hebb's postulate. This algorithm aimed to allow artificial neural networks to learn in a manner that more closely resembles human learning. Indeed, results indicated that weight updates in HebbSET adhered to Hebb's postulate and its extended formulation as presented in this paper.

Considering network topology, HebbSET appeared to be better able to find a topology that suits the data compared to SET. Furthermore, HebbSET showed to stabilise its topology slightly faster than SET, with relatively more connections remaining in the network during training.

Compared to SET, HebbSET showed improved performance in terms of learning speed and accuracy, especially on the MNIST dataset, indicating that adapting weights based on the activity level of connected neurons can indeed be used to improve the performance of ANNs trained using SET. However, these increases in performance come at the price of additional computations needed to calculate the Hebbian learning term. Further research could explore how to best balance this trade-off between improved performance and increased computational load. For example, instead of exponentially decaying λ, the Hebbian learning term could only be applied during the first part of training, after which SET

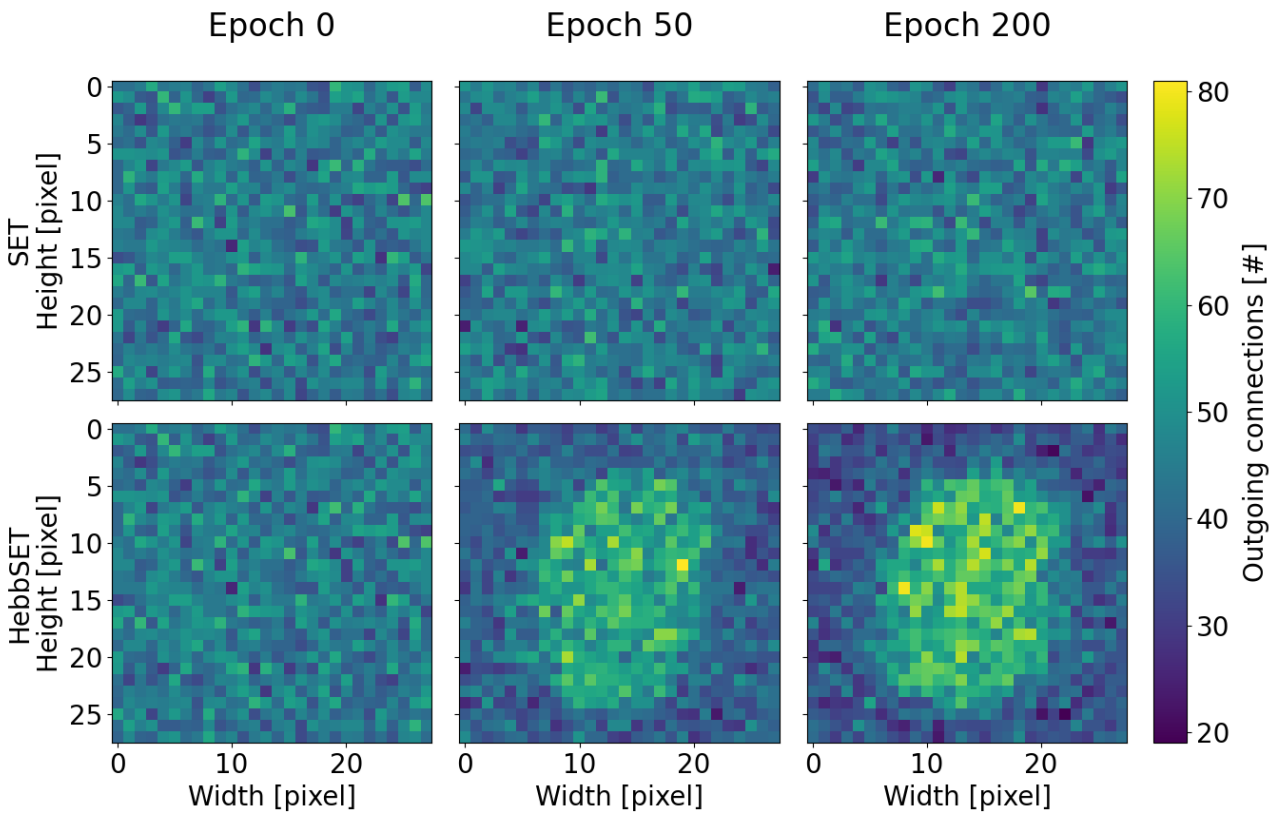

Fig. 5: Connectivity patterns for the input layer of two neural networks trained with SET and HebbSET ($\lambda_0 = 0.01$) after 0, 50, and 200 epochs. Each pixel represents a single input neuron of the network and is coloured depending on the number of outgoing connections from that neuron to the next layer. It can be seen that both networks start with a random topology. After just 50 epochs, however, the network trained using HebbSET shows a connectivity pattern that fits the data that are trained on. This pattern further evolves in subsequent epochs, while the network trained using SET displays no visible pattern in its connectivity even after 200 epochs.

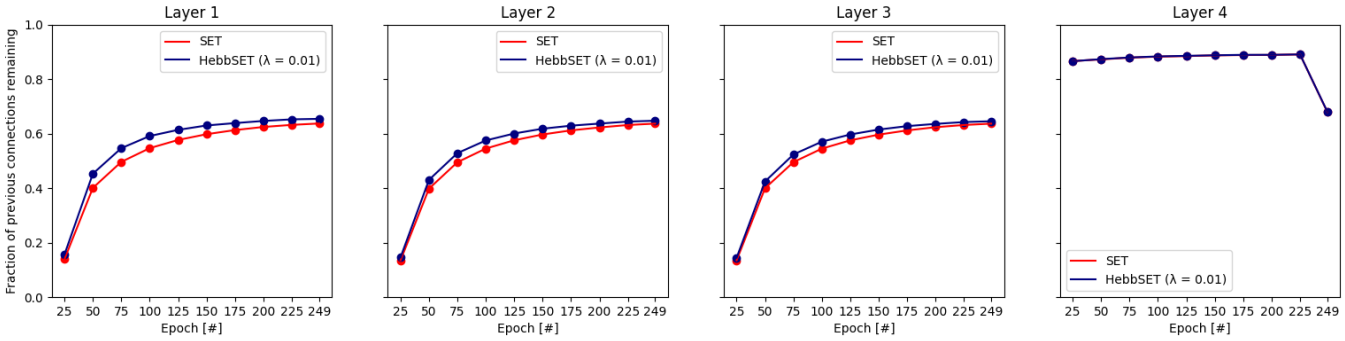

Fig. 6: Fraction of connections that were present in the network at the previous data point that still exist. Every 25 epochs, it is computed how many connections currently exist that were also part of the network 25 epochs previously. Measurements were taken over ten trials, of which the means are shown by the solid lines and the standard deviation by the shaded areas (which are nearly invisible due to the low standard deviation). Initially, the network topology changes rapidly and very few connections remain for 25 or more epochs, but the network topology appears to stabilize in later epochs. HebbSET tends to have more connections remain in the network over time compared to SET in the first three layers. In the final layer, both algorithms show the same result, likely because there is less room for variance due to the lower sparsity level of the final layer (see Equation 4).

could be used to further train the neural network.

Before Hebb's postulate could be included in HebbSET, it needed to be extended to include conditions for weight decreases and formalised mathematically. However, the formulation used in this paper (see Equation 7) is just one of many possible formulations of Hebbian learning. Future work could explore different learning rules, by, for example, including the average activation of the presynaptic neuron, adding weight decay, or introducing non-linearities (see [27] for an overview of several formulations of Hebbian learning).

To balance the effects of gradient descent and the proposed Hebbian term, a hyperparameter $\lambda(t)$ was used to scale the influence of the Hebbian term. Based on the results of this research, an initial value for λ of 0.01, decreased by 2% per epoch, appears to be effective in most cases. However, results suggest that during early epochs, a higher value of λ might result in faster learning. More research could be done to explore how to set this parameter and how to most effectively scale it over time.

Overall, HebbSET shows promising results, allowing ANNs to learn in a manner that is closer to what is believed to be happening in the human brain whilst simultaneously improving network performance. However, more, larger experiments on diverse datasets are necessary to further investigate the effectiveness of HebbSET. Furthermore, more research using different network architectures and methods, such as optimizers, activation functions, loss functions, dropout, regularisation, etc. should be conducted in order to better understand the potential of using Hebbian learning in artificial neural networks.

VII. Contextual Exploration

In this section, the broader impact of this research, and research on artificial intelligence and (sparse) neural networks in general, will be discussed.

A. Neural Networks and Energy Consumption

Artificial intelligence (AI) and machine learning (ML) have become ubiquitous in modern society. Many organizations already employ ML systems and recently, ChatGPT has allowed anyone with an internet connection to leverage the power of AI. However, training and running such models is expensive in terms of computational resources, power consumption, and money. For instance, the training of a large generative language model called GPT-3, resulted in the release of approximately 552 tons of CO_2 equivalent, highlighting the environmental cost of current AI practices [5]. Furthermore, such large models are not able to run, let alone be trained, on devices with limited computational power and energy capacity, such as smartphones. Sparse neural networks offer a promising opportunity to greatly reduce the computational load and environmental impact of machine learning models by significantly reducing the number of parameters of the models. In fact, researchers have recently shown that up to 50% of model weights can be removed from large language models with minimal loss in accuracy [31]. Further development of sparse neural networks can therefore help in the transition to environmentally friendly AI and enable ML models to be run and trained on lightweight devices. The algorithm proposed in this paper could help in these efforts, having shown that HebbSET is able to learn significantly faster than regular SET whilst still requiring much less computational power than fully-connected neural networks.

Although sparse-to-sparse neural network training can be used to train neural networks that require relatively little computations during both training and inference, this advantage is currently mainly a theoretical one. This is due to the fact that essentially all popular software and hardware for deep learning is highly optimised for dense matrix operations [32]. Because of this, many sparse ANN implementations make use of dense matrices combined with binary masks to simulate sparse neural networks, which negates most of the advantages in efficiency sparse neural networks have. However, recently, more efforts have been made to implement truly sparse ANNs which do not use binary masks. For example, in [33], a truly sparse ANN was introduced that was able to be trained on a typical laptop in spite of having over one million neurons. Furthermore, a Python library implementing a truly sparse version of SET was created based on the popular machine learning framework PyTorch [34]. HebbSET, as introduced in this paper, was also implemented in a truly sparse manner, based on a truly sparse implementation of SET [12]. Further development of truly sparse algorithms, software packages, and hardware optimised for sparse operations could further increase the efficiency of sparse neural networks, paving the way for a more sustainable generation of AI.

B. The Intersection of AI and Neuroscience

The fields of AI and neuroscience have a long history of influencing each other. Many models and theories from neuroscience have inspired AI techniques, such as perceptrons, reinforcement learning, etc., while AI has

helped mathematically formalize neuroscience concepts and offered interesting insights into the mechanisms of intelligence. Although neuroscience and AI have somewhat grown apart in recent years, there are good reasons to believe that even today these two fields can provide valuable insights into each other [18].

Perhaps the ultimate goal of AI research is to create artificial general intelligence (AGI), a system capable of performing a wide range of tasks at or beyond the human level [35]. While AI systems have shown superhuman performance on specific tasks such as playing Chess and Go [36], these models struggle to generalize to other tasks. Modelling AI systems after the human brain is perhaps the most plausible route to achieving AGI, given that humans are currently the only known generally intelligent entity [19][18][35]. However, our understanding of the human brain and how it functions is still very limited, and creating an accurate model of the human brain is far beyond our current capabilities. Nevertheless, there are many possibilities to already start bridging the gaps between ANNs and the human brain. The results of this research highlight that findings in neuroscience can indeed be successfully translated to AI systems in order to create more powerful models.

One area where AI research could benefit from studying the brain is the connectivity of neural networks. In nature, many animals exhibit innate, instinctive behaviour, that is encoded in their genetics rather than learned during their lifetime [19]. This suggests that genetic mechanisms for developing the structure of the brain can create effective neural networks without the need for learning. Similar observations have been made in ANNs, with researchers showing that ANNs can solve complex problems without learning any weights by exploiting suitable network architectures [37]. Therefore, although it has been shown that an ANN with just a single hidden layer can approximate any function [38], more sophisticated architectures, such as those found by (Hebb)SET, could provide significant benefits in terms of both performance and computational load.

Neuroscience and AI can also benefit other scientific fields and society at large. Most notably, knowledge and techniques from these fields can be used to better understand psychiatric and neurological illnesses and disorders such as Parkinson's disease and ADHD. Advancements in reinforcement learning have already proven to be useful for understanding the reward mechanisms in the human brain and how these mechanisms might relate to abnormal brain function [39]. Additionally, neuroimaging techniques combined with machine learning methods can allow researchers and doctors to diagnose and analyze mental disorders [40]. Thus, further advancements in neuroscience and AI could help mitigate, prevent, and possibly cure mental disorders- arguably one of the largest health issues of modern society [41].

VIII. Acknowledgements

I am very grateful to my supervisor, dr. Elena Mocanu, for all the valuable feedback and insights she offered over the course of this research, and for helping me gain a deeper understanding of this topic. I would also like to thank Ardion Beldad for joining me on this journey and always being ready to help when needed.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, pp. 84–90, 2017. DOI: 10.1145/3065386.

[2] D. Silver, J. Schrittwieser, K. Simonyan, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[3] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. DOI: 10.1038/nature14539.

[4] D. Amodei and D. Hernandez. "Ai and compute." Accessed: 2023/06/11. (May 2018), [Online]. Available: https://openai.com/research/ai-and-compute.

[5] D. Patterson, J. Gonzalez, Q. Le, *et al.*, *Carbon emissions and large neural network training*, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2104.10350.

[6] C. Li. "Openai's gpt-3 language model: A technical overview." Accessed: 2023/06/11. (Jun. 2020), [Online]. Available: https://lambdalabs.com/blog/demystifying-gpt-3.

[7] S. Han, J. Pool, J. Tran, and W. J. Dally, *Learning both weights and connections for efficient neural networks*, 2015. [Online]. Available: https://doi.org/10.48550/arXiv.1506.02626.

[8] J. Frankle and M. Carbin, *The lottery ticket hypothesis: Finding sparse, trainable neural networks*, 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1803.03635.

[9] Y. Lecun, J. Denker, and S. Solla, "Optimal brain damage," *Advances in Neural Information Processing Systems*, vol. 2, pp. 598–605, Jan. 1989.

[10] J. Liu, Z. Xu, R. Shi, R. C. C. Cheung, and H. K. H. So, "Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers," *International Conference on Learning Representation*, 2020.

[11] G. Bellec, D. Kappel, W. Maass, and R. Legenstein, "Deep rewiring: Training very sparse deep networks," Nov. 2017.

[12] D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta, "Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science," *Nature Communications*, vol. 9, 2018. DOI: 10.1038/s41467-018-04316-3.

[13] S. Dey, K.-W. Huang, P. Beerel, and K. Chugg, "Pre-defined sparse neural networks with hardware acceleration," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, pp. 332–345, Jun. 2019. DOI: 10.1109/JETCAS.2019.2910864.

[14] S. Liu and Z. Wang, *Ten lessons we have learned in the new "sparseland": A short handbook for sparse neural network researchers*, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2302.02596.

[15] D. S. Bassett and E. Bullmore, "Small-world brain networks," *The Neuroscientist*, vol. 12, no. 6, pp. 512–523, 2006. DOI: 10.1177/1073858406293182.

[16] J. C. Reijneveld, S. C. Ponten, H. W. Berendse, and C. J. Stam, "The application of graph theoretical analysis to complex networks in the brain," *Clinical Neurophysiology*, vol. 118, no. 11, pp. 2317–2331, 2007. DOI: 10.1016/j.clinph.2007.08.010.

[17] D. O. Hebb, *The organisation of behaviour: A neuropsychological theory*. John Wiley and Sons, 1949.

[18] D. Hassabis, D. Kumaran, C. Summerfield, and M. Botvinick, "Neuroscience-inspired artificial intelligence," *Neuron*, vol. 95, no. 2, pp. 245–258, 2017. DOI: https://doi.org/10.1016/j.neuron.2017.06.011.

[19] A. M. Zador, "A critique of pure learning: What artificial neural networks can learn from animal brains," *Nature Communications*, 2019. DOI: 10.1101/582643.

[20] D. Purves, G. J. Augustine, D. Fitzpatrick, *et al.*, *Neuroscience*. Sinauer Associates, Publishers, 2004, ISBN: 0-87893-725-0.

[21] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines," *Proceedings of ICML*, vol. 27, pp. 807–814, Jun. 2010.

[22] D. Pedamonti, *Comparison of non-linear activation functions for deep neural networks on mnist classification task*, 2018. eprint: 1804.02763.

[23] S. Ruder, *An overview of gradient descent optimization algorithms*, 2017. eprint: 1609.04747.

[24] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into deep learning*.

[25] L. Bottou, F. E. Curtis, and J. Nocedal, *Optimization methods for large-scale machine learning*, 2018. arXiv: 1606.04838.

[26] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[27] W. Gerstner and W. M. Kistler, "Mathematical formulations of hebbian learning," *Biological Cybernetics*, vol. 87, no. 5-6, pp. 404–415, 2002. DOI: 10.1007/s00422-002-0353-y.

[28] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2010.

[29] J. Li, K. Cheng, S. Wang, *et al.*, "Feature selection: A data perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, p. 94, 2018.

[30] A. Bhattacharjee, W. G. Richards, J. Staunton, *et al.*, "Classification of human lung carcinomas by mrna expression profiling reveals distinct adenocarcinoma subclasses," *Proceedings of the National Academy of Sciences*, vol. 98, no. 24, pp. 13 790–13 795, 2001. DOI: 10.1073/pnas.191502998.

[31] E. Frantar and D. Alistarh, *Sparsegpt: Massive language models can be accurately pruned in one-shot*, 2023. eprint: 2301.00774.

[32] S. Curci, D. C. Mocanu, and M. Pechenizkiyi, *Truly sparse neural networks at scale*, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2102.01732.

[33] S. Liu, D. C. Mocanu, A. R. R. Matavalam, Y. Pei, and M. Pechenizkiy, *Sparse evolutionary deep learning with over one million artificial neurons on commodity hardware*, 2021. [Online]. Available: https://doi.org/10.48550/arXiv.2102.01732.

[34] M. Klear. "The sparse future of deep learning." Accessed: 2023/06/12. (2018), [Online]. Available: https://towardsdatascience.com/the-sparse-future-of-deep-learning-bce05e8e094a.

[35] B. Goertzel, "Artificial general intelligence: Concept, state of the art, and future prospects," *Journal of Artificial General Intelligence*, vol. 5, no. 1, pp. 1–48, 2014. DOI: doi:10.2478/jagi-2014-0001.

[36] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[37] A. Gaier and D. Ha, "Weight agnostic neural networks," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, 2019.

[38] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991. DOI: 10.1016/0893-6080(91)90009-T.

[39] T. Maia and M. Frank, "From reinforcement learning models to psychiatric and neurological disorders," *Nature Neuroscience*, vol. 14, pp. 154–62, 2011. DOI: 10.1038/nn.2723.

[40] N. Yahata, K. Kasai, and M. Kawato, "Computational neuroscience approach to biomarkers and treatments for mental disorders," *Psychiatry and Clinical Neurosciences*, vol. 71, 2016. DOI: 10.1111/pcn.12502.

[41] H. Wittchen, F. Jacobi, J. Rehm, *et al.*, "The size and burden of mental disorders and other disorders of the brain in europe 2010," *European Neuropsychopharmacology*, vol. 21, no. 9, pp. 655–679, 2011. DOI: 10.1016/j.euroneuro.2011.07.018.

[42] V. Capano, H. J. Herrmann, and L. De Arcangelis, "Optimal percentage of inhibitory synapses in multi-task learning," *Scientific reports*, vol. 5, no. 1, p. 9895, 2015.

APPENDIX
ADDITIONAL VISUALISATIONS

A. *Network Topology*

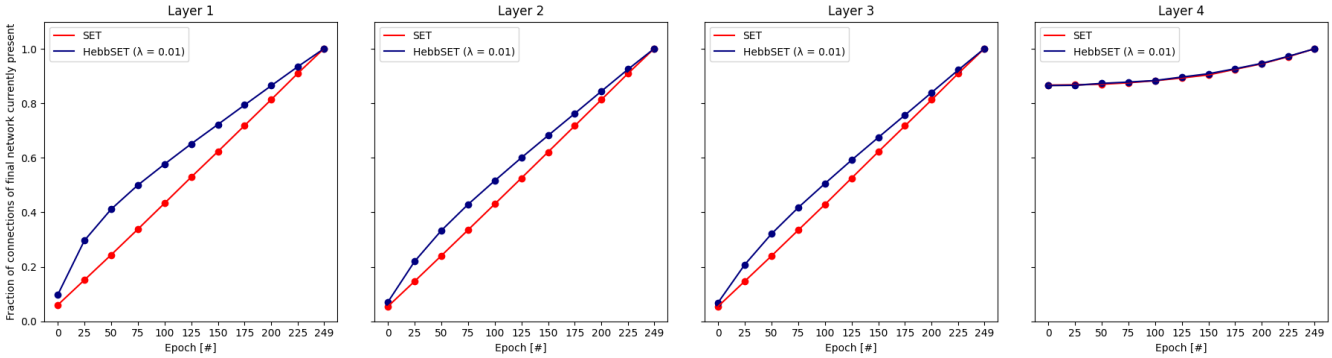

Fig. 7: Fraction of connections that are currently present in the network that also exist in the final network. Measurements were taken over ten trials, of which the means are shown by the solid lines and the standard deviation by the shaded areas (which are nearly invisible due to the low standard deviation). In the final layer, both algorithms show the same result, likely because there is less room for variance due to the lower sparsity level of the final layer (see Equation 4). Note that measurements are made after the weight pruning and adding step of SET.

Similar to Figure 6, Figure 7 shows how the topology of the ANNs trained using HebbSET and SET evolves over time. However, instead of comparing the current topology with that of the previous data point, as done in Figure 6, it is compared with the final topology instead. Interestingly, at epoch zero, HebbSET and SET have a nearly equal fraction of weights that end up in the final network. This suggests that it is not the case that HebbSET simply evolves a topology that is relatively similar to its starting topology. In fact, after training, both HebbSET and SET only have around 5-10% of their original connections remaining. However, HebbSET is able to more quickly stabilise into a suitable topology than SET. This might indicate that the Hebbian learning term is able to effectively modify weights in such a manner that important connections are more likely to be kept, and vice versa.

B. Weight Evolution and Distribution

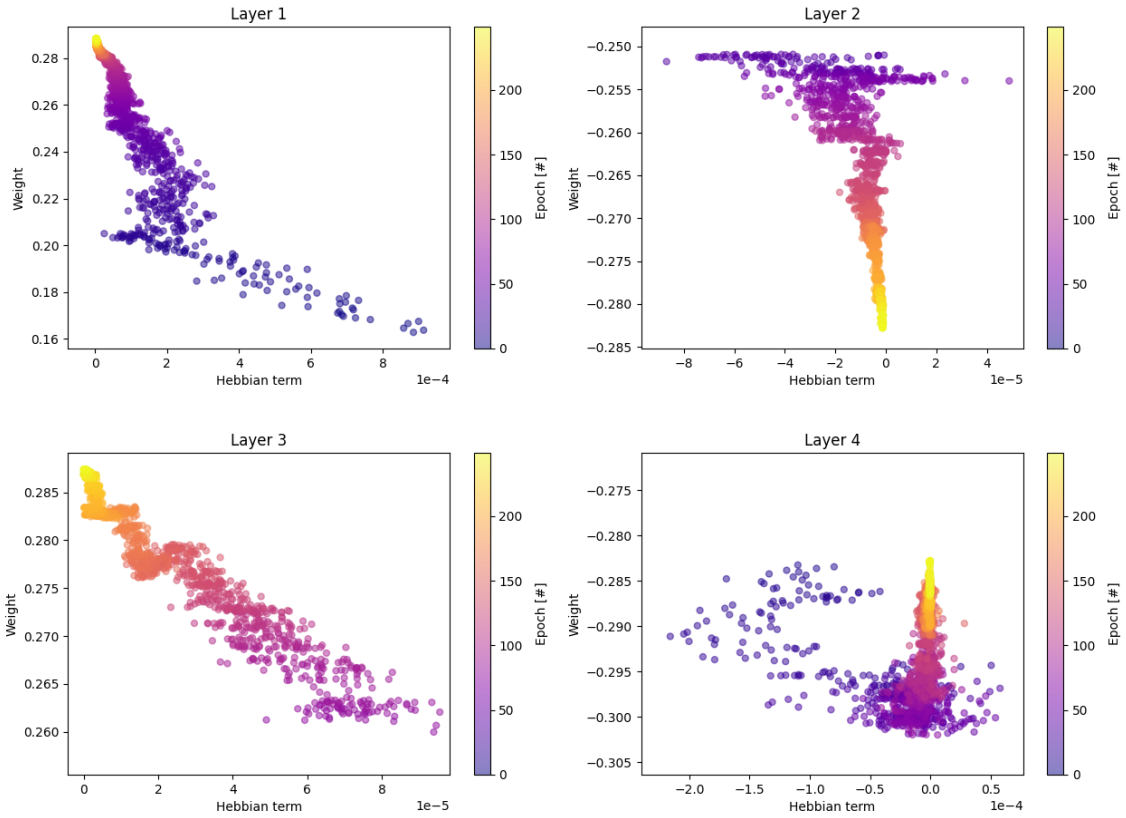

Fig. 8: Hebbian term versus the weight magnitude over time.

Figure 8 shows how an individual weight, as well as its Hebbian term, evolves over time. From each layer, one random weight was picked for this figure. Each dot represents a single mini-batch training step for HebbSET on the MNIST dataset. The dot is positioned based on the magnitude of the weight at that time and the magnitude of the Hebbian term as computed for that mini-batch of data. Furthermore, each dot is coloured based on the current epoch, with earlier epochs having darker colours and the final epochs having lighter colours. Note that, since most weights are not present during the whole lifetime of the ANN, the plots are only drawn starting at the epoch that the connection in question gets added to the neural network.

Figure 8 clearly shows how λ reduces over time. As the epoch increases, the Hebbian term tends to zero for all connections depicted. Furthermore, it can be seen that the sign of the Hebbian term is quite consistent. Even over a wide range of epochs and a large number of mini-batches, the Hebbian term tends to be either positive or negative for a given weight. Furthermore, the direction in which the weight moves tends to correspond with the sign of the Hebbian update. In layers one and three, the Hebbian term is consistently positive, and the connection weight can be seen to steadily increase as training progresses. Conversely, in layers one and four, the Hebbian term tends to be negative, and the associated weight consequently decreases, although the weight in layer four ends up increasing again during later epochs.

In Figure 9, the weight distributions after training are presented for SET and HebbSET. These ANNs were trained for 250 epochs using the MNIST dataset. Table III gives a further statistical description of the obtained weight distributions. In HebbSET, weight magnitude tends to be slightly larger and more varying than in SET. Furthermore, in HebbSET, all layers except the last have significantly more positive than negative weights, whereas in SET these are approximately equally split. Interestingly, the increased fraction of positive weights found in ANNs trained using HebbSET is consistent with evidence from neuroscience. In BNNs, excitatory synapses tend to make up the vast majority of connections, with research showing that only 30% of synapses are inhibitory in mammalian brains

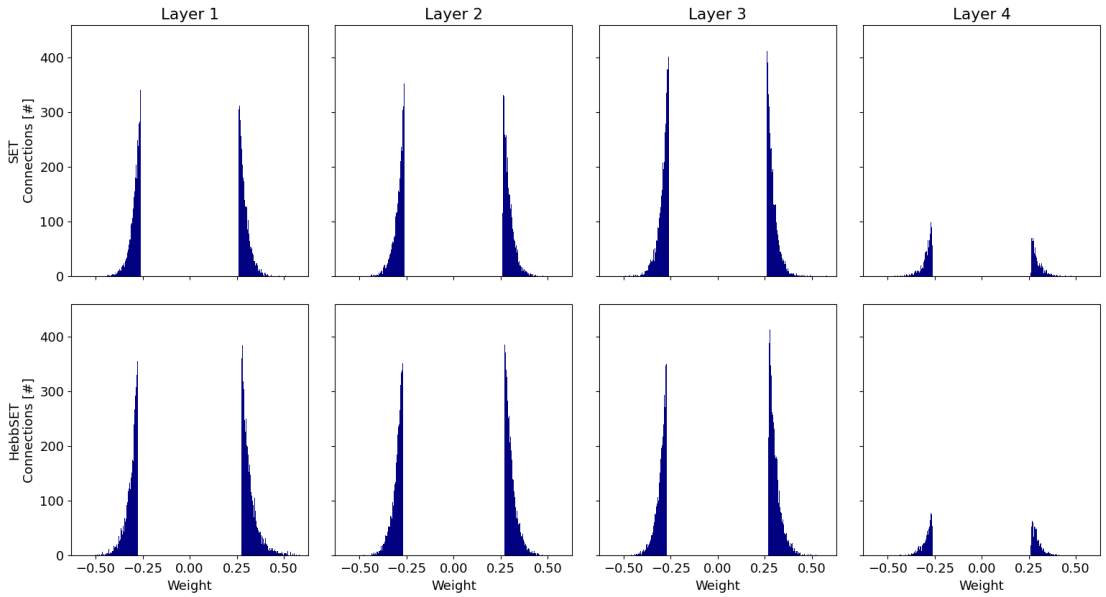

Fig. 9: Distribution of weights in SET and HebbSET after training for 250 epochs.

[42]. The final layer of the ANNs trained with both SET and HebbSET contains relatively many negative weights. This is likely due to the fact that for a given input, all outputs are expected to be zero except for one. Therefore, in order to minimize loss, the ANN needs to suppress nine out of ten output neurons, which requires a relatively large number of negative weights.

Model	Layer	Positive weights [%]	Mean of positive weights	Mean of negative weights
SET	Layer 1	50.3	0.292 ($\pm$0.029)	-0.291 ($\pm$0.028)
	Layer 2	50.5	0.292 ($\pm$0.029)	-0.292 ($\pm$0.029)
	Layer 3	50.0	0.292 ($\pm$0.029)	-0.292 ($\pm$0.029)
	Layer 4	46.8	0.294 ($\pm$0.029)	-0.293 ($\pm$0.030)
HebbSET	Layer 1	54.3	0.322 ($\pm$0.050)	-0.312 ($\pm$0.036)
	Layer 2	52.8	0.306 ($\pm$0.033)	-0.304 ($\pm$0.031)
	Layer 3	56.6	0.306 ($\pm$0.034)	-0.303 ($\pm$0.031)
	Layer 4	46.7	0.295 ($\pm$0.030)	-0.294 ($\pm$0.030)

TABLE III: Summary statistics for the weight distribution per layer for SET and HebbSET (λ_0 = 0.01)

.