NEUROEVOLUTION TICKET SEARCH — FINDING SPARSE, TRAINABLE DNN INITIALISATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

The *Lottery Ticket Hypothesis* (LTH) asserts that a randomly initialised overparameterised Deep Neural Network (DNN) contains a sparse subnetwork that, when trained (up to) the same amount as the original network, performs just as well. So-called *winning tickets* are vastly more efficient to train than dense networks; however, finding such tickets currently relies on pre-training an overparameterised network via *Iterative Magnitude Pruning* (IMP). Due to the increasing demand for computing resources, there are major incentives to find good search procedures for winning tickets (which can drastically reduce the training time required to solve a range of problems). In this paper, we propose a new method for the evolution of sparse DNN initialisations that generates results commensurate with winning ticket search procedures: we refer to the method as *Neuroevolution Ticket Search* (NeTS). By training an overparameterised network using gradient descent for use as a baseline only, we show that NeTS quickly converges to near state-of-the-art performance in terms of network size and trainability. Additionally, we find that NeTS appears to be a competitive alternative to IMP in terms of wall clock time.

1 INTRODUCTION

In recent years, Deep Neural Networks (DNNs) have been shown as an incredibly powerful and newly practical tool for addressing a range of problems (Samek et al., 2021). Whilst the expressive power of deep neural networks has been known since the early 1990s (e.g. Bishop, 1994), it is only in the last decade that we have developed the requisite technology to efficiently train very large models (e.g. Ahranjany et al., 2010). This had led to an explosion of interest in the field of *deep learning*, and the key to this success has been the design, implementation, and strategic deployment of hardware that can efficiently perform tensor arithmetic (Cireşan et al., 2010): the core computation involved in optimising neural network parameters. However, large models are not particularly wieldly: not only do they need an expensive training process, they also require greater resources for storage and inference (Hinton et al., 2015; Molchanov et al., 2017; Luo et al., 2017; Yoo et al., 2022). In this paper, we propose a new method for the evolution of *sparse* networks in the shape of a given architecture that does not require iterative gradient descent. Here, sparsity refers to the connectedness of the layers of neurons composing the network — more connections means a lower sparsity (and higher density).

The current state-of-the-art in deep learning first involves training an overparameterised dense network, before subsequently removing extraneous connections. This has typically resulted in methods for neural network *pruning*: the systematic removal of units, weights, or other structures, without compromising inference performance (e.g. Han et al., 2015; 2016; Luo et al., 2017; Molchanov et al., 2017; Liang et al., 2021; Agarwal et al., 2023). Through the application of pruning, large networks can often be transformed into a much smaller size, before possibly retraining, or *tuning*, the network to restore performance. Training the original overparameterised network is computationally expensive, but empirical evidence suggests that the initial overparameterisation is necessary for the learning dynamics of the network (Han et al., 2015; Li et al., 2017; Frankle & Carbin, 2019) — in other words, training a randomly initialised sparse network from the outset does not appear to be as successful as training and pruning a dense one.

Challenging the foundation of this strategy, Frankle & Carbin (2019) posited the *Lottery Ticket Hypothesis* (LTH). The hypothesis asserts that a randomly-initialised DNN contains a subnetwork initialisation that, when trained in isolation, achieves the test accuracy of the original network after training for at most the same number of iterations. The LTH has been empirically and analytically validated for, among other models, fully-connected *Feed-forward Neural Networks* (FNNs) (Frankle & Carbin, 2019; Mehta, 2019; Frankle et al., 2020b; Malach et al., 2020; Zhang et al., 2021). In simpler terms, the LTH says that the key to a particular network's success is contained within a subset of its starting conditions — that key (or subnetwork) is referred to as a *winning ticket*.

Inspired by the LTH and its implications, we introduce a new method for finding sparse network initialisations (winning tickets) that uses a genetic algorithm to optimise both the initial weights and a binary pruning mask for an FNN architecture. We refer to the algorithm as *Neuroevolution Ticket Search* (NeTS) to emphasise the goal of finding tickets (or initialisations) as opposed to outright successful networks. Such *neuroevolution* in deep learning typically uses a genetic algorithm to evolve a neural network through the iterative mutation and/or multi-parental crossover of a *genome* that encodes, either directly or indirectly, a DNN (Galván & Mooney, 2021). The DNN expression of a genome is referred to as a *phenotype*. Like traditional approaches, we mutate and combine the weights to optimise a fitness function: in our case, the validation loss after a small amount of training via gradient descent. In addition, to find sparse initialisations, we also apply and evolve a binary pruning mask alongside the weights. However, unlike existing approaches, only the network initialisations are communicated across generations: meaning the weights of the network after the period of training that occurs in the fitness function are discarded.

In this paper, we compare the phenotypes of the best genomes (encoding network initialisations) found by the evolutionary process to winning tickets found using a random strategy, and *Iterative Magnitude Pruning* (IMP). While this work presents only preliminary results, we find that winning tickets evolved using NeTS are comparable in performance and sparsity with those found via IMP (and significantly improving on the random approach). Whilst the efficiency of our approach does not appear to be superior to state-of-the-art gradient descent methods, we present encouraging results that suggest NeTS is a competitive alternative with substantially different learning dynamics.

2 Background

Before exploring our research questions and experimental results in detail, we briefly give some technical background on deep learning and the LTH. The formal notation and properties introduced in this section will be useful for defining our method formally in Section 3 and evaluating its performance in Section 4.

2.1 Feed-forward Networks

A *Feed-forward Neural Network* (FNN) is represented as a function $f(\boldsymbol{x};\theta)$. As standard, $\boldsymbol{x}$ denotes the input to the network f (parameterised by the contents of θ). In the simplest case, the parameters of the network are sampled from a distribution ($\theta \sim \mathcal{D}_\theta$) and are represented by an r-rank tensor of connection weights W^r (we will omit the rank in future). A *pruning mask* is a binary tensor $\mathsf{M}^r = [m] \in \{0,1\}^r$ where each element corresponds to a single connection of the network. The element-wise multiplication of a pruning mask M to weights W, otherwise referred to as the Hadamard product, is denoted $\mathsf{W} \odot \mathsf{M}$. For brevity, we introduce a shorthand for applying a mask to the weights of a fully-connected network f as $f(\boldsymbol{x};\theta \odot \mathsf{M})$. The *density* (and, by reciprocal, the *sparsity*) of a network is defined with respect to the mask tensor M. In the following, we use $|\mathsf{M}|_i$ to denote the number of i-valued elements in the mask such that $|\mathsf{M}|_1 = \sum_{j=1}^{|\mathsf{M}|} m_j$ and $|\mathsf{M}|_0 = |\mathsf{M}| - |\mathsf{M}|_1$.

Definition 1. The *density* of a neural network $f(\boldsymbol{x};\theta_0 \odot \mathsf{M})$ is given as $d = \frac{|\mathsf{M}|_1}{|\mathsf{M}|}$. The *sparsity* of a network is $s = \frac{|\mathsf{M}|_0}{|\mathsf{M}|}$ or $1-d$.

In general form, a FNN is a DNN consisting of an input layer, output layer, and n hidden layers. The first layer encodes the input data to the network, and each subsequent layer is dependent only on the nodes in the layer directly preceding it. More formally, we model a non-input layer (with optional bias vector $\boldsymbol{b}$) as a linear function $f_i(\boldsymbol{x}_{i-1};\theta_i) = a(\boldsymbol{W}_i\boldsymbol{x}_{i-1} + \boldsymbol{b}_i)$ for $i \in \{1,2,\dots,n\}$ (where a is a differentiable activation function). Here, $\boldsymbol{x}_{i-1}$ is the output of the previous layer ($\boldsymbol{x}_0 = \boldsymbol{x}$),

$\boldsymbol{W}$ is a matrix of the incoming connection weights for the layer, and n is the number of non-input layers. Accordingly, the full network f is simply the composition of each of layer to the input data: $f(\boldsymbol{x};\theta) = (f_{n+1} \circ f_n \circ \cdots \circ f_1)(\boldsymbol{x})$.

The *de facto* standard for training FNNs is via backpropagation and, specifically, *gradient descent* — this applies to most types of DNN. The combination enables efficient optimisation of a network's weights $\mathbf{W}$ by following the gradient of a differentiable loss function L and making updates on the basis of a learning rate η such that $\mathbf{W}_{i+1} \leftarrow \mathbf{W}_i - \eta \cdot \nabla_{\mathbf{W}_i} L$. We say a network *converges* when the loss of the network against a set of unseen validation inputs reaches a predetermined minimum (after this point, we say the network has been *over-fit*). Stochastic Gradient Descent (SGD) is an iterative method that approximates gradient descent and reduces the time taken to update the weights after a single iteration of training data (it does, however, increase the time to convergence).

2.2 The Lottery Ticket Hypothesis

As we outlined in the introduction, Frankle & Carbin (2019) introduced the LTH as an assertion of the presence of a sparse and trainable subnetwork within the starting conditions of a dense, overparameterised network. Numerous authors have demonstrated the LTH's applicability to various DNN architectures and training configurations (Morcos et al., 2019; Brix et al., 2020; Chen et al., 2020; Frankle et al., 2020a; Girish et al., 2021; Bai et al., 2022). In the following, we define a *ticket* more concretely than the original work, but to the same practical effect. We find that our definition is useful for positioning our contribution and specifying a search procedure more formally, but do not mean to restrict or change the notion of a (winning) ticket in any meaningful way.

Definition 2. For an architecture named f, a *ticket* is a pair $(\theta_0, \mathbf{M})$ where θ_0 is a valid parametrisation of f such that $f(\boldsymbol{x}; \theta_0 \odot \mathbf{M})$ is an FNN.

We remain agnostic to the particular implementation details of a given architecture (e.g. the number of layers, activation functions, etc.) in our definition of a ticket, but we do require a fixed number of (initial) parameters for the initial initialisation. The set of all tickets for an FNN with n parameters is defined as $\Theta = \{([w_1 w_2 \dots w_n], [m_1 m_2 \dots m_n]) \mid w_i \in \mathbb{R} \text{ and } m_i \in \{0, 1\}\}$. Naturally, this set is extremely large rendering exhaustive methods intractable for practical problems. However, the LTH is concerned with two non-trivially related properties of tickets for which clear (if not necessarily efficient) heuristics exist. That is, *trainability:* the LTH asserts there exists tickets that can be trained efficiently to the same (or superior) minimum validation loss as the overparameterised network given the same training dataset $\mathbf{D}_{\text{train}}$. And sparsity: the LTH asserts that a much sparser network is sufficient to reconstruct the performance after training of a denser network.

Again, similarly but using our more concrete (c.f. Frankle & Carbin, 2019) understanding of a ticket, we formally define a *winning ticket* as one that is both sparse and trainable.

Definition 3 (Winning Ticket)**.** Let $f(\boldsymbol{x}; \theta_0)$ be an FNN optimised using SGD on a training dataset such that it reaches minimum validation loss l at iteration j with test accuracy a. Similarly, let $f'(\boldsymbol{x}; \theta_0 \odot \mathbf{M})$ be an FNN optimised identically to f to reach minimum validation loss l' at iteration j', with test accuracy a'. The ticket $(\theta_0, \mathbf{M})$ is a *winning ticket* iff (1) $j' \leq j$, the training time of f' is commensurate with f; (2) $a' \geq a$, the accuracy of f' is commensurate with f; and (3) $|\mathbf{M}|_1 \ll |\mathbf{W}_0|$, f' has (far) fewer (non-zero) weights than f.

According to these definitions, we frame our contribution — the evolutionary algorithm NeTS — as a ticket search procedure that seeks to find winning tickets from the set of all possible tickets Θ. We believe the evolutionary approach offers an advantage insofar as we are able to explicitly optimise for both criteria (trainability and sparsity) in a common fitness function, using the outcome of limited SGD as a signal for genetic selection and combination only.

2.3 Iterative Magnitude Pruning

Alongside the LTH, Frankle & Carbin (2019) introduce Iterative Magnitude Pruning (IMP) as a method for finding winning tickets — we consider IMP the canonical example of a ticket search procedure. A later formulation of IMP, with k-rewinding, was later found to improve the performance of IMP for large networks (Frankle et al., 2020b). The original variation (where weights are reset to starting values) is the special case when $k = 0$. The procedure works by first randomly initialising and storing parameters for a densely connected neural architecture. The initialised network is then trained for T iterations of a training dataset, before the smallest $p \in [0, 1]$ weights are pruned (by layer). Critically, after pruning, the weights are reset to their initial (or close to initial values). The network is then retrained for T iterations (and re-pruned iteratively from then on for $c \in \mathbb{N}$ cycles).

A crucial observation of IMP's experimental performance articulated by Frankle & Carbin (2019) in their original work was that resetting, or rewinding, weights is necessary for finding winning tickets. From this, we can generate two further untested hypotheses. First, the learning dynamics of SGD are liable to damage the winning ticket during training. Second, the learning dynamics of SGD are useful for finding winning tickets in the original initialisation. In Section 3, we introduce NeTS and illustrate how we capitalise on the signal that validation loss after SGD emits without destroying the ticket through premature convergence. In Section 4 we confirm that NeTS performs better when weights learned via SGD are discarded but we do not test the above hypotheses any further in this work.

3 Neuroevolution Ticket Search

In this section, we will formally define our algorithm for finding sparse networks using neuroevolution. In essence, we want to explore the extent to which the concurrent optimisation of weights and masks is able to reproduce the behaviour of IMP without the need to first train an overparameterised dense network to conversion with SGD. Ultimately we find that our algorithm NeTS, by incorporating features of the iterative approach in a multi-objective fashion, is able to generate similar performance with substantially different learning dynamics. Indeed, whilst we only report on preliminary results, we are encouraged by contemporary results in machine learning that imply neuroevolution can be a competitive alternative to gradient descent, particularly for large networks in reinforcement learning domains (e.g. Such et al., 2018); and we intend to explore NeTS's scalability and performance in more detail in future work.

3.1 Genetic Encoding

We use a direct genetic encoding for the problem of training a subnetwork: encoding a particular architecture as a fixed-length genome.

Definition 4. A *genome* g is the concatenation of a real valued vector $\boldsymbol{w} = [w_0 w_1 \dots w_n]$ encoding the weights of a network $f(\boldsymbol{x}, \theta)$, and a binary vector $\boldsymbol{m} = [m_0 m_1 \dots m_n]$ encoding a mask to be applied to the network. Here, $n = |\theta|$ is the number of parameters in the network, and we say that $g_i = (w_i, m_i)$ is a single *gene* when $n \in \mathbb{N}$ and $n \leq |\theta|$.

Weight vectors are initialised from an arbitrary distribution $w_i \sim \mathcal{D}_\theta$ (we use a normal distribution with mean 0 and standard deviation 0.1); and binary vectors are sampled from a Bernoulli distribution according to a hyperparameter p that determines the initial probability that a parameter is enabled. As with standard representations of DNNs, the problem size grows non-linearly as new nodes are added to the network. The phenotypic expression of a genome is computed by trivially reshaping $\boldsymbol{w}$ to $\mathbf{W}_0$ (a component of θ_0) and $\boldsymbol{m}$ to $\mathbf{M}$ such that the phenotypic network is $f(\boldsymbol{x}; \theta_0 \odot \mathbf{M})$.

3.2 Fitness Evaluation

One particular benefit of evolutionary algorithms is the ability to introduce arbitrary customisations to the fitness function. In NeTS, we are able to combine information about the training loss after a period of optimisation via SGD (e.g. one epoch of a training dataset) with information about the density of the network as a whole. The fitness function in Appendix A defines the minimisation objective of the genetic optimisation which uses the sum of the validation loss of a trained network and a penalty term P defined relative to an actual density d and target density τ as follows.

$$P = \left(\frac{d - \tau}{1 - \tau}\right)^2 \tag{1}$$

In Section 1, we stated that weights optimised through SGD in the fitness function are discarded. This approach may appear counter-intuitive given the success found by alternating evolution and gradient descent methods; however, the function was specifically formulated in this manner to ensure that the NeTS searches the space of initialisations only. In particular, we seek to mitigate any destructive influence SGD has on the search for winning tickets (whether or not it exists). We believe that this is necessary for the success of the algorithm, and this has been supported by our empirical results — we will discuss this observation in more detail in the final section of the paper.

We evaluate the fitness of genomes according to their density and the average loss computed on a training dataset $\mathbf{D}_{train}$. We assume that a reasonably-sized validation dataset $\mathbf{D}_{valid}$ is sampled from the training dataset prior to execution. In our experiments we use a validation dataset that is 10% the size of the original training set.

3.3 Selection, Crossover, and Mutation

As we are seeking to observe the learning dynamics of a new approach to finding winning tickets, we intend to keep the neuroevolution algorithm as simple as possible. The following are the minimal set of genetic operations we conduct in our algorithm, in the order they are applied.

1. *Uniform Crossover:* two parent genomes are combined via the stochastic transference of genes with probability $p \in [0, 1]$ from parent 1 (and probability $1 - p$ from parent 2) to a new offspring.
2. *Random Noise Mutation:* an individual gene is mutated through the addition of Gaussian noise to the current weight value with probability $q \in [0, 1]$. We use a mean of 0 and a standard deviation of 0.1 to generate noise from the standard normal distribution.
3. *Random Weight Mutation:* an individual gene is mutated through the resampling of its weight value $w_i \sim \mathcal{D}_{\mathbf{W}_0}$ with probability $r \in [0, 1]$. Weights are sampled identically to their original initialisation.
4. *Dormancy Mutation:* an individual gene is mutated by setting the mask value to 0 with probability $s \in [0, 1]$; the weight value remains in the genome but is effectively dormant or *unexpressed*.

In addition to the above operations, we carry out *elitism* to (optionally) carry through the $e \in \mathbb{N}$ most fit genomes unchanged into the next generation. This has been observed to encourage conversion.

3.4 Using NeTS to Find Winning Tickets

In order to observe the dynamics of NeTS, we run a series of experiments that seek to evolve winning tickets for the LeNet-300-100 architecture when trained on the popular MNIST handwriting classification dataset (Deng, 2012). Figure 1 describes the general behaviour of NeTS for 25 generations. We run the algorithm with a population size of 5 and the following probabilistic parameters.

Algorithm 1: Neuroevolution Ticket Search (NeTS).

```
Data: $k \in \mathbb{N}^{+} \backslash 1$ (population size), $n \in \mathbb{N}$ (number of generations).
Result: $f^{\star}(\boldsymbol{x}; \theta_0)$ (most successful network)
$\mathbb{G} \leftarrow$ init(k); // initialise population
$\sigma^{\star} = -\infty, f^{\star} = \emptyset$; // find best fitness
for $i \in \{0, 1, \ldots, n\}$ do
    for $g \in \mathbb{G}$ do
        $g.\sigma \leftarrow$ fitness($g$); // compute fitness
    $\mathbb{G}' \leftarrow$ crossover($\mathbb{G}$); // crossover w.r.t fitness
    $\mathbb{G}' \leftarrow$ mutate($\mathbb{G}$); // mutate genomes
    $\sigma \leftarrow \arg\max\{g.\sigma \mid g \in \mathbb{G}\}$;
    if $\sigma > \sigma^{\star}$ then
        $\sigma^{\star} \leftarrow \sigma$; // update best fitness
        $f^{\star}(\boldsymbol{x}; \theta_0) \leftarrow$ pheno($g$) where $g.\sigma = \sigma^{\star}$; // update best network
return $f^{\star}(\boldsymbol{x}; \theta_0)$
```

Probability of uniform crossover, $p = 0.5$; probability of random noise mutation, $q = 0.1$; probability of random weight mutation, $r = 0.1$; and probability of dormancy mutation, $s = 0.2$. Initially we elected to run experiments with $s = q = r = 0.1$; however, we found that increasing s to 0.2 decreased the time to converge with no discernible trade-off. We set a target density $\tau = 0.2$ in line with the minimum (approx. 21% density) successful network reliably found by IMP (Frankle & Carbin, 2019).

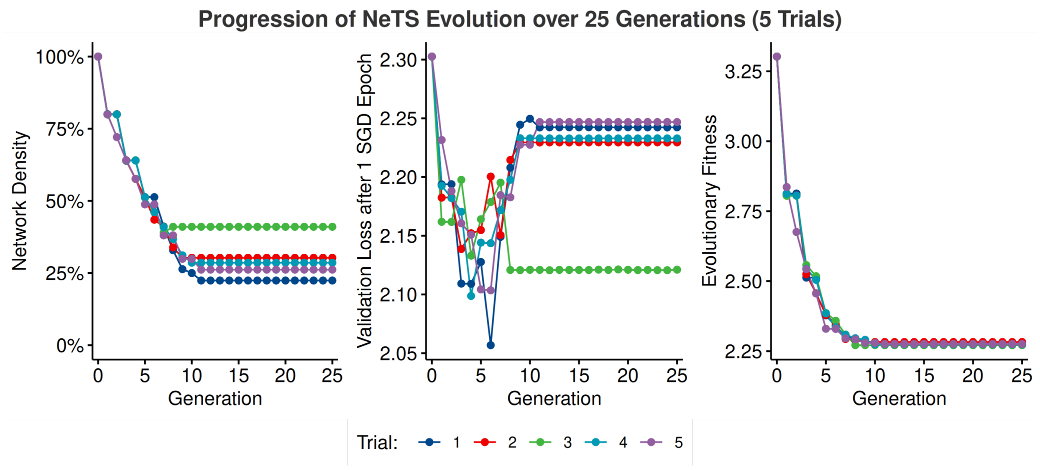

Figure 1: This plot shows the progression of the evolutionary algorithm across three measures: network density, accuracy against a previously unseen test dataset, and the minimisation objective, or fitness function, computed after one epoch of SGD training before discarding weights.

From the data presented in Figure 1, we can observe that NeTS quickly (within a single generation) finds a handle on the two loss gradients relating to network density and validation loss respectively. Interestingly, we see that even after finding tickets that can achieve up to 40% unseen validation accuracy with only one epoch of SGD by the sixth generation, NeTS begins to converge on solutions with close to target density but a lower validation accuracy. By generation 20, NeTS finds a local minimum from which it cannot escape; longer running experiments have shown that this remains the case for (at least) 50 generations. As alluded to in Section 2, we hypothesise that this is due to SGD damaging the winning ticket — in other words, we suspect that SGD destroys the generality of the initialisation (in order to find an optimal solution). Future work will seek to explore the impact removing elitism has on the dynamics of NeTS, as well as increasing the population size and number of experimental trials to much higher values.

4 Evaluating the Performance of NeTS

We have now formally introduced NeTS, an algorithm for finding sparse, trainable initialisations of FNNs and shown it is capable of minimising a fitness signal to find sparse and (potentially) trainable networks. We now seek to evaluate two questions: (1) how fast does NeTS converge on a ticket when compared to IMP, and (2) how good are NeTS winning tickets when compared to tickets found via IMP (or a random method)? In addressing the first question, we note that comparing the runtime of an evolutionary search procedure with a solely gradient descent procedure is challenging. First, the tensor computation involved in the backpropagation required for SGD has been highly optimised by over a decade of specific research in this area. Second, evolutionary algorithms are highly parallelisable both in terms of the computation of the fitness function and, often, the distribution of the population across multiple processes. As a consistent measure across both IMP and NeTS, we record the wall clock time at every iteration of training and generation of the genetic algorithm.

In this section, we present the results of limited experiments on two standard machine learning problems: the XOR problem (computing a non-linear decision boundary for classifying bits according to the XOR function), and the character recognition problem encoded in the MNIST training and testing datasets.

4.1 Algorithm Runtime, Density, and Loss

To give NeTS and IMP a consistent and fixed amount of resources, we perform our experiments on one CPU only. In Figure 2, we concatenate the period of evolution (up to approx. 1000s) with the period of SGD that trains the winning ticket once it has been found. During NeTS, we target a density of 20% as this was close to the smallest successful initialisation found in Frankle & Carbin (2019) original work. We stop the algorithm when NeTS remains at a minimum fitness value for more than 2 generations, in practice this entailed running NeTS for a mean of 11 generations. In each iteration of IMP we used 50 epochs of training time, which was the closest round number to the number of epochs used in the original paper introducing the LTH (Frankle & Carbin, 2019). In order to reach our target density of 20%, we need 7 cycles of IMP (culminating with a network of 21% density).

4.2 Quality of Winning Tickets

Once winning tickets have been identified, in our case via NeTS or IMP, we train them using SGD on the MNIST character recognition dataset Deng (2012) to compare their performance with respect to the trainability and sparsity measures introduced in Section 2. To do so, we train tickets generated from both procedures, as well as randomly generated tickets for comparison, for 50 epochs of the training data (37,500 iterations). We chose this value on the basis of the experimental configuration given in the original paper introducing the LTH (Frankle & Carbin, 2019). We generate five tickets for five independent stochastic trials. The test losses and accuracies determined during the training execution are shown in Figure 3.

On the basis of these results, we can observe IMP converges quicker and to a lower loss and higher test accuracy than both NeTS and the random method. However, encouragingly NeTS clearly performs much better than randomly initialising sparse networks of comparable density. In contrast to Figure 1 that shows the evolution of candidate initialisations whose weights have not yet been permanently modified by SGD, the results in Table 1 in Appendix B describe the initialisations after

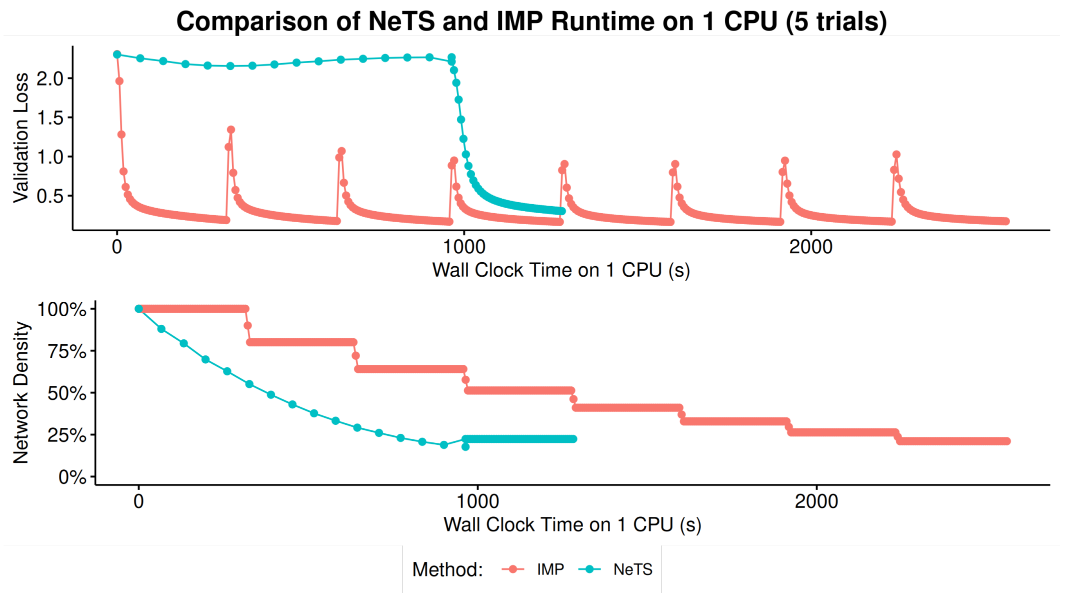

Figure 2: The upper plot shows the validation loss of both NeTS and IMP as they progress; the plot below describes the network densities. The first segment of the blue graphs (prior to approx. 1000s) show the period of evolution of the best chromosome in the population. The second segment of the blue graphs depict the SGD phase of NeTS. The red graphs show the consecutive cycles of IMP.

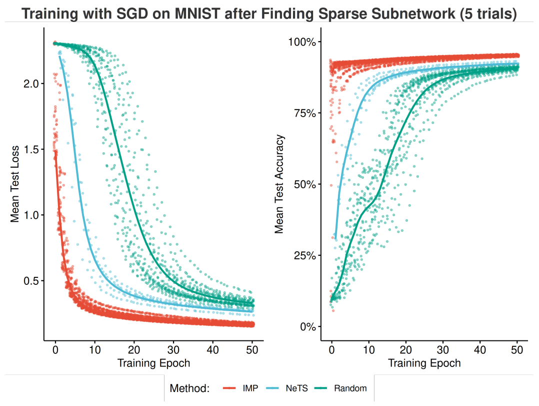

Figure 3: We plot the mean test loss and accuracy for winning ticket initialisations found using IMP (red), NeTS (blue), and sparse initialisations generated randomly (green). On the x-axis we plot the training epoch.

training via SGD. Whilst NeTS is dominated by IMP in terms of loss and accuracy; we do find that the range of tickets found by NeTS is much broader than would be expected by random chance. In a similar vein to other approaches in evolutionary algorithms, it may be the case that NeTS finds candidate solutions which tradeoff different aspects of the multi-objective optimisation. Accordingly, we see future work as exploring whether, and if so what, the set of tickets found during a run of NeTS display different properties to one another.

5 Conclusions

In this paper we have sought to introduce a new method for finding winning tickets, or sparse and trainable DNN initialisations, without first pruning. Instead our method NeTS (as introduced in Section 3) has a fitness function that trades off network density with accuracy (or more precisely with validation loss of a trained network). This is preliminary work and it is clear that further study of NeTS is needed in order to more clearly understand its behaviour with respect to other train-and-prune approaches. Future work will seek to examine this behaviour through hyperparameter variation experiments and the introduction of new evolutionary techniques (like speciation) aiming to avoid premature convergence on a local minima. That being said, NeTS does show some potential, and we believe it is a new result to show that genetic algorithms can be used to learn not only optimal weights of a network, but also a pruning mask to limit the size of the resulting architecture. The key benefit of this approach (as opposed to Iterative Magnitude Pruning) is that the fitness function can be customised in ways that may not be possible in gradient descent (for example, non-differentiable functions can be used).

5.1 Future Work

As we have pointed out throughout the main body of this paper, the information conveyed represents only a preliminary investigation into NeTS, its application to finding winning tickets, and its performance relative to state-of-the-art methods. The main areas of future work we see as being particularly important are as follows. First, we want to investigate the hyperparameter space of NeTS in much greater detail. In order to retain a fair hypothesis test, we have avoided changing the hyperparameters of the search to improve the outcome — instead opting for reasonable values. In the future, we expect to test NeTS under a much wider variety of configurations. Second, we want to explore the impact of using dropout in the fitness function SGD to determine whether this enables the weights learned during SGD to be retained. And, finally, we would like to examine how NeTS performs with variable dataset lengths. One of the more surprising findings of our work has been that NeTS can perform well with only the validation loss of one epoch of training data as a fitness signal. Exploring whether this can be reduced even further is, we believe, an interesting line of future inquiry.

References

Mohit Agarwal, Suneet Kr. Gupta, and K. K. Biswas. Genetic algorithm based approach to compress and accelerate the trained Convolution Neural Network model. *International Journal of Machine Learning and Cybernetics*, January 2023. ISSN 1868-8071, 1868-808X. doi: 10.1007/s13042-022-01768-4.

Sajjad S. Ahranjany, Farbod Razzazi, and Mohammad H. Ghassemian. A very high accuracy handwritten character recognition system for Farsi/Arabic digits using Convolutional Neural Networks. In *2010 IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA)*, pp. 1585–1592, September 2010. doi: 10.1109/BICTA.2010.5645265.

Yue Bai, Huan Wang, Zhiqiang Tao, Kunpeng Li, and Yun Fu. Dual Lottery Ticket Hypothesis, March 2022.

Chris M. Bishop. Neural networks and their applications. *Review of Scientific Instruments*, 65(6): 1803–1832, June 1994. ISSN 0034-6748. doi: 10.1063/1.1144830.

Christopher Brix, Parnia Bahar, and Hermann Ney. Successfully Applying the Stabilized Lottery Ticket Hypothesis to the Transformer Architecture, July 2020.

Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. The Lottery Ticket Hypothesis for Pre-trained BERT Networks. In *Advances in Neural Information Processing Systems*, volume 33, pp. 15834–15846. Curran Associates, Inc., 2020.

Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, big, simple neural nets for handwriting digit recognition. *Neural Computation*, 22(12):3207–3220, December 2010. ISSN 0899-7667. doi: 10.1162/NECO_a_00052.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks, March 2019.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, and Michael Carbin. Linear Mode Connectivity and the Lottery Ticket Hypothesis. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 3259–3269. PMLR, November 2020a.

Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, and Michael Carbin. Stabilizing the lottery ticket hypothesis, July 2020b.

Edgar Galván and Peter Mooney. Neuroevolution in Deep Neural Networks: Current Trends and Future Challenges. *IEEE Transactions on Artificial Intelligence*, 2(6):476–493, December 2021. ISSN 2691-4581. doi: 10.1109/TAI.2021.3067574.

Sharath Girish, Shishira R. Maiya, Kamal Gupta, Hao Chen, Larry S. Davis, and Abhinav Shrivastava. The Lottery Ticket Hypothesis for Object Recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 762–771, 2021.

Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks, October 2015.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding, February 2016.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, March 2015.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient ConvNets, March 2017.

Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, October 2021. ISSN 0925-2312. doi: 10.1016/j.neucom.2021.07.045.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. ThiNet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5058–5066, 2017.

Eran Malach, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir. Proving the Lottery Ticket Hypothesis: Pruning is All You Need. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 6682–6691. PMLR, November 2020.

Rahul Mehta. Sparse Transfer Learning via Winning Lottery Tickets, December 2019.

Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference, June 2017.

Ari S. Morcos, Haonan Yu, Michela Paganini, and Yuandong Tian. One ticket to win them all: Generalizing lottery ticket initializations across datasets and optimizers, October 2019.

Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J. Anders, and Klaus-Robert Müller. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE*, 109(3):247–278, March 2021. ISSN 1558-2256. doi: 10.1109/JPROC.2021.3060483.

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning, April 2018.

Joanna Yoo, Kuba Perlin, Siddhartha Rao Kamalakara, and João G. M. Araújo. Scalable training of language models using JAX pjit and TPUv4, April 2022.

Zeru Zhang, Jiayin Jin, Zijie Zhang, Yang Zhou, Xin Zhao, Jiaxiang Ren, Ji Liu, Lingfei Wu, Ruoming Jin, and Dejing Dou. Validating the Lottery Ticket Hypothesis with Inertial Manifold Theory. In *Advances in Neural Information Processing Systems*, volume 34, pp. 30196–30210. Curran Associates, Inc., 2021.

A Fitness Function for NeTS

Algorithm 2: Fitness function for NeTS (to be minimised).

```
Data: $g \in \mathbb{G}$ (genome), $\mathbf{D}_x$ (datasets), $L$ (loss function), $T$ (iterations of SGD), $\tau$ (target
      density).
Result: $fit \in \mathbb{R}$
$f(\boldsymbol{x}; \theta_0 \odot \mathbf{M}) \leftarrow \texttt{pheno}(g)$; // construct network
$f'(\boldsymbol{x}; \theta_T) \leftarrow \texttt{train}(f(\boldsymbol{x}; \theta_0), \mathbf{D}_{train}, T)$; // train network
$l \leftarrow L(f'(\boldsymbol{x}; \theta_0 \odot \mathbf{M}), \mathbf{D}_{valid})$; // compute loss
$d \leftarrow \frac{|\mathbf{M}|_1}{|\mathbf{M}|}$; // compute density
$P \leftarrow \left(\frac{d-\tau}{1-\tau}\right)^2$; // penalty term (1)
return $l + P$
```

B Itemisation of XOR and MNIST Results

Dataset	XOR			MNIST		
After 50 epochs of SGD:	Random	IMP	NeTS	Random	IMP	NeTS
Mean Density	20%	20%	20%	22.8%	21.1%	29.7%
Density Range	=	=	=	[15%, 39.9%]	=	[22.4%, 41%]
Mean Validation Loss	0.032	≈ 0.0	≈ 0.0	0.350	0.173	0.302
Mean Test Loss	0.024	0.012	0.053	0.329	0.163	0.263
Mean Validation Accuracy	82%	100%	100%	90.6%	95.1%	91%
Mean Test Accuracy	84%	100%	100%	90%	95.2%	92.3%

Table 1: The data presented in this table describes the performance of winning ticket initialisations found via three ticket search methods on two common datasets (XOR and MNIST with target densities 20%) after training via SGD. For the random method, tickets are generated from a given density d sampled from a normal distribution with mean 0.2, standard deviation 0.05; for IMP, 7 iterative cycles of training and pruning are carried out for 50,000 iterations; and for NeTS, 25 generations of 5 genomes are evolved. Our results report on the sparse networks after 50 epochs of SGD.