Citation for published version (APA):
Finding Sparse Initialisations using Neuroevolutionary Ticket Search (NeTS)

Alex Jackson¹,², Nandi Schoots¹,², Amin Ahantab³, Michael Luck¹,², and Elizabeth Black¹,²

¹King’s College London, London, United Kingdom, WC2B 4BG
²UKRI CDT in Safe and Trusted Artificial Intelligence
³Independent Researcher, Munich, Germany
alex.jackson@kcl.ac.uk

Abstract

The explosion of interest in deep learning over the last decade has been driven by scaling mathematical models of brains referred to as Deep Neural Networks (DNNs). Common deep learning architectures have millions of parameters that require optimisation (training) for the network to learn tasks; and empirical evidence suggests training an overparameterised network (with more parameters than data points in the training data) is necessary for the network to learn. Despite this, it has been shown that overparameterised DNNs can typically be compressed to a fraction of their original size once trained (frequently by over 90%). Further, the Lottery Ticket Hypothesis (LTH) asserts that there exists a much sparser but equally trainable subnetwork within any sufficiently overparameterised DNN initialisation. In other words, the number of parameters could be significantly reduced from the outset if we could find these smaller initialisations (referred to as winning tickets). In this paper, we introduce a new evolutionary algorithm for finding winning tickets given a feed-forward or convolutional DNN architecture, and compare our approach to the current state-of-the-art. We refer to our algorithm as Neuroevolutionary Ticket Search (NeTS) and find it discovers competitive winning tickets for a variety of architectures and two common training datasets (MNIST and CIFAR-10). We show that NeTS can be applied to pruning DNNs before substantial training with gradient descent by genetically optimising a genome consisting of a set of initial weights and a binary pruning mask; this appears to offer a significant performance benefit.

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 (Alzubaidi 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. Ahranjan et al., 2010). This has 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 network initialisations that does not require iterated gradient descent (a key feature of the best performing method). Here, sparsity refers to the connectedness of the layers of neurons composing the network — fewer connections means a higher sparsity (and lower density).

The current state-of-the-art in deep learning first involves training an overparameterised dense network before subsequently removing extraneous connections. Typically, this has 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 over-parameterisation is necessary for the learning dynamics of the network (Han et al., 2015; Li et al., 2017; Frankle and 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 and 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. 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. Since its publication, the LTH has been proved analytically (Malach et al., 2020; Zhang et al., 2021) as well as empirically validated for a variety of unsupervised learning algorithms.

Finding Sparse Initialisations using Neuroevolutionary Ticket Search (NeTS)
architectures (Frankle and Carbin, 2019; Mehta, 2019; Frankle et al., 2020b; Malach et al., 2020; Zhang et al., 2021; Yu et al., 2020; Yin et al., 2023; Iqbal and Mishra, 2023; Mishra and Gupta, 2023). The LTH has also been applied successfully to natural language processing, reinforcement learning, tabular datasets, and even pre-trained language models and transformers (Yu et al., 2020; Bluteau and Gras, 2023; Singh and Bhatele, 2020).

Inspired by the LTH and its implications, we introduce a new method for finding winning tickets that uses a genetic algorithm to search sparse initialisations for a particular architecture. We refer to the algorithm as Neuroevolutionary Ticket Search (NeTS) to emphasise the goal of finding tickets (or initialisations) as opposed to outright successful networks. To find sparse networks: a mutation operation disables weights resulting in a pruned chromosome. During fitness evaluation: a copy of the chromosome is trained (using SGD), evaluated, and discarded. This evaluation is used to decide how much the (pruned but untrained) chromosome is allowed to reproduce in a multi-parental crossover.

The LTH and by extension methods for finding winning tickets have applications for, among other aspects of the deep learning pipeline, neural network pruning. To assess the performance of NeTS, we compare the density and trainability of the phenotypes of the best genomes (encoding network initialisations) found by the evolutionary process to those found using Iterative Magnitude Pruning (IMP) and a random method. Frankle and Carbin (2019) introduced IMP alongside the LTH in their original paper: unlike NeTS, IMP relies on the iterative training and pruning of an overparameterised network. We seek to answer the following research questions:

1. **Can NeTS find winning tickets?** To evaluate NeTS in this regard, we compare the trainability of sparse initialisations found by NeTS with randomly sampled sparse initialisations for a variety of architectures. We find that NeTS consistently outperforms the random method: finding sparse networks that are quicker to converge and close to the overparameterised performance.

2. **How do tickets found by NeTS compare with those found via IMP?** To compare NeTS’s performance w.r.t. state-of-the-art methods, we compare the trainability of tickets found using IMP. We find that NeTS does not find tickets as trainable as IMP but does perform significantly better than random.

3. **How efficient is NeTS at finding winning tickets compared to IMP?** To assess NeTS’s competitiveness in terms of computational efficiency we compare the runtime of NeTS and IMP on one CPU. We find that NeTS provides a highly competitive alternative to IMP with low pruning rates.

2 Related Work

Within the larger field of evolutionary algorithms a variety of methods for hybrid applications of neuroevolution have been proposed. We will briefly highlight the approaches most relevant to this paper.

Evolutionary algorithms, in particular genetic algorithms, have been used for both structured and unstructured pruning of convolutional deep networks. For example, motivated by network science, Mocanu et al. (2018) develop a method of neural network pruning that uses a genetic algorithm to find a sparse initialisation (as opposed to a traditional fully-connected network). Evolutionary algorithms have also found particular adoption in transfer learning, for example Samala et al. (2018) apply evolutionary pruning for breast cancer diagnoses; and, more recently Agarwal et al. (2023) use a genetic algorithm for filter pruning focusing on developing methods for finer control of the tradeoff between accuracy and sparsity in a transfer learning context. In addition to convolutional neural networks, Li et al. (2022) introduce a method for pruning vision transformers that significantly reduces the computational overhead of many architectures with minimal reduction in test accuracy. Like Mocanu et al. (2018), we focus on finding a sparse initialisation using a genetic algorithm; however, we only use gradient descent as a signal in the fitness function as opposed to mutating the individuals’ weights.

3 Deep Learning 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 4 and evaluating its performance in Section 5.3.

3.1 Feed-forward Networks

A Feed-forward Neural Network (FNN) is represented as a function \( f(x; \theta) \). As standard, \( x \) denotes the input to the network \( f \) (parameterised by the contents of \( \theta \)). In the simplest case, the parameters of the network are sampled from a distribution \( (\theta \sim D_\theta) \) 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 \( M^r = \{m\} \in \{0, 1\}^r \) 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 \( W \odot M \). For brevity, we introduce a shorthand for applying a mask to the weights of a fully-connected network \( f \) as \( f(x; \theta \odot M) \). The density (and, by reciprocal, the sparsity) of a network is defined with respect to the mask tensor \( M \). In the following, we use \( |M|_1 \) to denote the number of \( i \)-valued elements in the mask such that \( |M|_1 = \sum_{j=1}^{|M|} m_j \) and \( |M|_0 = |M| - |M|_1 \).
Definition 1 (Density and Sparsity). The density of a neural network \( f(x; \theta_0 \odot M) \) is given as \( d = \frac{|M|}{M} \). The sparsity of a network is \( s = \frac{|M|}{M} \) 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 \( b \)) as a linear function \( f_i(x_{i-1}; \theta_i) = a(W_i x_{i-1} + b_i) \) for \( i \in \{1, 2, \ldots, n\} \) (where \( a \) is a differentiable activation function). Here, \( x_{i-1} \) is the output of the previous layer \( (x_0 = x) \), \( W \) 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: \( f(x; \theta) = (f_{n+1} \circ f_n \circ \cdots \circ f_1)(x) \).

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 \( W \) by following the gradient of a differentiable loss function \( L \) and making updates on the basis of a learning rate \( \eta \) such that \( W_{i+1} \leftarrow W_i - \eta \cdot \nabla_W L \). 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).

In the next section we will use the following definition.

Definition 2 (Trainability). The trainability of a neural network \( f(x; \theta_0 \odot M) \) after \( t \) iterations of SGD on a training dataset \( D_{\text{train}} \) is defined to be the accuracy of \( f \) on a validation dataset \( D_{\text{validation}} \).

3.2 The Lottery Ticket Hypothesis

As we outlined in the introduction, Frankle and 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 3 (Ticket). For an architecture named \( f \), a ticket is a pair \((\theta_0, M)\) where \( \theta_0 \) is a valid parameterisation of \( f \) such that \( f(x; \theta_0 \odot M) \) 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 \( \Theta = \{([w_1, w_2, \ldots, w_n], [m_1, m_2, \ldots, m_n]) \mid w_i \in \mathbb{R} \text{ and } m_i \in \{0, 1\}\} \). 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 \( D_{\text{train}} \). 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 and Carbin, 2019) understanding of a ticket, we formally define a winning ticket as one that is both sparse and trainable.

Definition 4 (Winning Ticket). Let \( f(x; \theta_0) \) 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 \( f'(x; \theta_0 \odot M) \) be an FNN optimised identically to \( f \) to reach minimum validation loss \( l' \) at iteration \( j' \), with test accuracy \( a' \). The ticket \((\theta_0, M)\) is a winning ticket if (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) \(|M| \ll |W_0|\), \( f' \) 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 \( \Theta \). 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.

3.3 Iterative Magnitude Pruning

Alongside the LTH, Frankle and 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 Franke and 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 5.3, 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.

4 Neuroevolutionary 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 over-parameterised 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.

4.1 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 5. A genome $g$ is the concatenation of a real valued vector $w = [w_0 w_1 \ldots w_n]$ encoding the weights of a network $f(x, \theta)$, and a binary vector $m = [m_0 m_1 \ldots m_n]$ 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 $w_i \sim D_0$ (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.

<table>
<thead>
<tr>
<th>Algorithm 1: Fitness function for NeTS (to be minimised).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data: $g \in \mathcal{G}$ (genomes), $D_i$ (datasets), $L$ (loss function), $T$ (iterations of SGD), $\tau$ (target density).</td>
</tr>
<tr>
<td>Result: $fit \in \mathbb{R}$</td>
</tr>
<tr>
<td>$f(x; \theta_0 \odot M) \leftarrow \text{pheno}(g)$; // construct network</td>
</tr>
<tr>
<td>$f'(x; \theta_\tau) \leftarrow \text{train}(f(x; \theta_0), D_{\text{train}}, T)$; // train network</td>
</tr>
<tr>
<td>$l \leftarrow L(f'(x; \theta_0 \odot M), D_{\text{valid}})$; // compute loss</td>
</tr>
<tr>
<td>$d \leftarrow \frac{</td>
</tr>
<tr>
<td>$P \leftarrow \left(\frac{d - \tau}{1 - \tau}\right)^2$; // penalty term</td>
</tr>
<tr>
<td>return $f + P$</td>
</tr>
</tbody>
</table>

The phenotypic expression of a genome is computed by trivially reshaping $w$ to $W_0$ (a component of $\theta_0$) and $m$ to $M$ such that the phenotypic network is $f(x; \theta_0 \odot M)$.

4.2 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 Algorithm 1 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 $\tau$ as follows.

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

In Section 4.1, 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 $D_{\text{train}}$. We assume that a reasonably-sized validation dataset $D_{\text{valid}}$ 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. **Elitism**
In addition to the above operations, we carry out elitism in our algorithm, in the order they are applied.

4. **Dormancy Mutation**: an individual gene is mutated by setting the mask value to 0 with probability \( s \in [0, 1] \). We use a mean of 0 and a standard deviation of 0.1 to generate noise from the standard normal distribution.

5. **Random Weight Mutation**: an individual gene is mutated through the resampling of its weight value \( w_i \sim D_{\sigma_0} \) with probability \( r \in [0, 1] \). Weights are sampled identically to their original initialisation.

6. **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.

### 4.3 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.

Figure 1: An illustration of the genetic operations: selection, crossover, and mutation; followed by an illustration of how masks are applied to extract a phenotype network.

### 4.4 Applying NeTS to MNIST

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 2 describes the general behaviour of NeTS for 25 generations. We run the algorithm with a population size of 5 and the following probabilistic parameters. 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 and Carbin 2019).

From the data presented in Figure 2, 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 3 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.

### Algorithm 2: Neuroevolutionary Ticket Search (NeTS)

**Data:** \( k \in \mathbb{N} \setminus \{1\} \) (population size), \( n \in \mathbb{N} \) (number of generations).

**Result:** \( f \times (x; \theta_0) \) (most successful network)

\[G \leftarrow \text{init}(k); \quad \text{// initialise population}\]

\[\sigma^* = -\infty; f^* = \emptyset; \quad \text{// find best fitness}\]

for \( i \in \{0, 1, \ldots, n\} \) do

for \( g \in G \) do

\[g.\sigma \leftarrow \text{fitness}(g); \quad \text{// compute fitness}\]

\[G' \leftarrow \text{crossover}(G); \quad \text{// crossover w.r.t fitness}\]

\[G' \leftarrow \text{mutate}(G); \quad \text{// mutate genomes}\]

\[\sigma \leftarrow \arg \max \{g.\sigma \mid g \in G\};\]

if \( \sigma > \sigma^* \) then

\[\sigma^* \leftarrow \sigma; \quad \text{// update best fitness}\]

\[f^*(x; \theta_0) \leftarrow \text{pheno}(g) \quad \text{where} \quad g.\sigma = \sigma^*;\]

// update best network

return \( f^*(x; \theta_0) \)
5 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 proceed by evaluating the questions posed in the introduction:

1. Can NeTS find winning tickets? (See Definition 4.) Which we answer in Section 5.1.
2. How do winning tickets found by NeTS compare with those found via IMP? Which we answer in Section 5.2.
3. How efficient is NeTS at finding winning tickets compared to IMP? Which we answer in Section 5.3.

In this section, we present the results of limited experiments on a standard machine learning problem of character recognition problem encoded in the MNIST training and testing datasets. We used the CREATE high performance computing cluster (King’s College London, 2022) to run our experiments.

5.1 Finding Winning Tickets

In Section 4.4, we have shown how tickets with high fitness value are obtained using NeTS. Recall that winning tickets are those tickets that after training (with SGD) can obtain the same or higher test accuracy to a randomly initialised dense network after training (with SGD).

The experimental results shown in Figure 3 and Table 1 report on three types of initialisation trained on MNIST with a LeNet-300-100 architecture; and on CIFAR-10 with a Conv-6 architecture. For NeTS, we carry out 15 generations of evolution during which 5 individual chromosomes (genomes) are evolved towards a target density of 20%. For the sparse method, tickets are generated with identical density \( d \) to those found via NeTS. We take the average of five trials for all experiments. In Figure 3, we show the loss curves for initialisations obtained via NeTS, randomly pruning a network and taking an unpruned dense network. We see that for the degree of pruning that we chose (a target density of 20%), an overparameterised dense network reduced in loss faster than networks that were pruned via either of the two pruning methods (NeTS and randomly pruning).

In Table 1 we show that using NeTS we can find much sparser (up to 90.7% pruning) initialisations that can be trained close to the test accuracy of a dense overparameterised network using both a LeNet and convolutional architecture for MNIST and CIFAR-10 respectively. Whilst we do not find winning tickets (that match the test accuracy of a dense network) we are able to consistently find more immediately successful initialisations (compared to random...
Table 1: This table shows the performance of an overparameterised dense and sparse randomly pruned network with winning ticket initialisations found via NeTS on the MNIST and CIFAR-10 datasets. For NeTS, 15 generations of 5 genomes are evolved towards a target density of 20%. For the sparse method, tickets are generated with identical density \( d \) to those found via NeTS.

<table>
<thead>
<tr>
<th>Dataset:</th>
<th>MNIST (after 50K iterations)</th>
<th>CIFAR-10 (after 30K iterations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>After SGD:</td>
<td>Dense</td>
<td>Sparse</td>
</tr>
<tr>
<td>Mean Density</td>
<td>100%</td>
<td>26.3%</td>
</tr>
<tr>
<td>Density Range</td>
<td>100%</td>
<td>12.4%–63.3%</td>
</tr>
<tr>
<td>Mean Val. Loss</td>
<td>0.174</td>
<td>0.243</td>
</tr>
<tr>
<td>Mean Test Loss</td>
<td>0.163</td>
<td>0.230</td>
</tr>
<tr>
<td>Mean Val. Acc.</td>
<td>94.9%</td>
<td>93.0%</td>
</tr>
<tr>
<td>Mean Test Acc.</td>
<td>95.3%</td>
<td>93.4%</td>
</tr>
</tbody>
</table>

networks of the same density). This has been achieved with minimal to no exploration of the evolutionary hyperparameter space.

5.2 Quality of Winning Tickets

Once successful initialisations 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 their trainability, i.e. test accuracy after SGD (see Definition 2) and sparsity (see Definition 1). 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. We chose this value on the basis of the experimental configuration given in the original paper introducing the LTH [Frankle and 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 4.

![Image of Figure 4](image)

Figure 4: 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.

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 2, which shows the evolution of candidate initialisations whose weights have not yet been permanently modified by SGD, the results in Table 1 describe the initialisations after 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.

5.3 Algorithm Runtime, Density, and Loss

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.

To give NeTS and IMP a consistent and fixed amount of resources, we perform our experiments on one CPU only. In Figure 5, we concatenate the period of evolution (up to approx. 1000s) with the period of SGD that trains the winning ticket. During NeTS, we target a density of 20% as this was close to the smallest successful initialisation found in [Frankle and Carbin, 2019] original work. We stop the algorithm when NeTS remains at a minimum fitness value for more than 2 generations (mean 11 generations). In each iter-
Figure 5: 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.

6 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 4) 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).

6.1 Implications for Control over Model Safety

A fitness function can check for any properties, not just ones that are a differentiable function of the weights. The flexibility of fitness functions may give an engineer the opportunity to nudge their model towards desired robustness, interpretability and verification properties [Xiao et al., 2019; Lipton, 2018].

Hypotheses (about what might result in good behaviour) can be more easily encoded in a fitness function than a differentiable loss function. In evolutionary algorithms we can for example incorporate training signals that are a function of the topology of the network (as opposed to all training signals needing to be differentiable functions of the weights in gradient descent). Another property of evolutionary deep learning approaches is that they tend to leave large parts of the model unchanged in between two updates. For example, if there is a circuit in a network with a property; the chance that this circuit is carried over to future generations unchanged may be higher in evolution than SGD. By contrast, in gradient descent we expect small change in weights to have big impact on property.

Especially in cases of large population, and high incidence and variance of a property, a fitness function will allow nudging in the direction of the property. However, if the property does not occur in the population at all, then the fitness function is not able to encourage convergence in this direction. Although gradient descent is only able to use differentiable training signals, we do of course (in both gradient descent and evolutionary programming) have the option to train a network and post-hoc decide to discard it because it lacks a certain (non-differentiable) property.

6.2 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 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.
Acknowledgments This work was supported by UK Research and Innovation [grant number EP/S023356/1], in the UKRI Centre for Doctoral Training in Safe and Trusted Artificial Intelligence (https://safeandtrustedai.org).
References


Han, S., Mao, H., and Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding.

Han, S., Pool, J., Tran, J., and Dally, W. J. (2015). Learning both weights and connections for efficient neural networks.


