Pruning Latent Neurons in Autoencoders using Early-Bird Tickets

by

Rembrandt Klazinga

to obtain the degree of Master of Science in Computer Science
at the EEMCS faculty of the Delft University of Technology,
to be defended publicly on Friday June 10, 2022 at 10:00 AM.

Student number:	4674642	
Project duration:	November 8, 2021 – June 10, 2022	
Thesis committee:	Dr. M. Loog,	TU Delft, supervisor
	Dr. J. van Gemert,	TU Delft, chair
	Dr. C. Liem,	TU Delft

Preface

You are reading the thesis “Pruning Latent Neurons in Autoencoders using Early-Bird Tickets”, written during my master’s at the Delft University of Technology. It describes a technique to remove redundant latent neurons from an autoencoder using a recent pruning method, and is written in a conference format. Due to the size constraints of the conference format, there is not enough space to discuss all of the relevant background information. Hence, the report is followed by a Background Knowledge section, which you are encouraged to read: it explains a number of key concepts such as network pruning, batch normalisation, and autoencoders, as well as discussing the foundational papers of this research.

During the thesis, my main supervisor was Marco Loog, whom I would like to thank for the excellent guidance and engrossing conversations. I would often feel uncertain about my next steps in the thesis, but that would invariably be resolved after a single meeting. My thesis committee further consisted of Jan van Gemert, who was also the chair, and Cynthia Liem, both of whom I would like to thank for their interest in my thesis, as well as their availability, openness, and feedback.
I also know that I could not have done this without the support and help of my family, friends, and colleagues: Iris, Andor, Tomas, Nick, Terry, and Stefan. Thank you for keeping me sane during this project, and supporting me whenever I encountered issues.

I hope you enjoy reading the thesis.

Rembrandt Klazinga
Delft, June 2022

Pruning Latent Neurons in Autoencoders using Early-Bird Tickets

Rembrandt Klazinga **Marco Loog**

Abstract

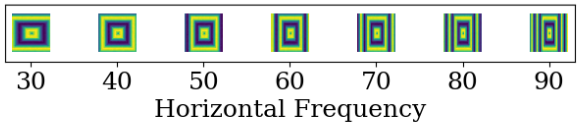
Autoencoders seek to encode their input into a bottleneck of latent neurons, and then decode it to reconstruct the input. However, if the input data has an intrinsic dimension (ID) smaller than the number of latent neurons in the bottleneck, this encoding becomes redundant. In this paper, we study using the Early-Bird (EB) technique, a structural pruning method, to regularise and prune the redundant latent neurons. We do this for both linear-layer and convolutional autoencoders, on 1D and 2D data. We find that increasing the strength of EB regularisation specifically on the latent layer can lead to all redundant latent neurons (and no more) being removed in one training run. We also compare using EB in this manner to existing ID estimation methods: we find it performs comparable to older methods like local-PCA, also being relatively robust to noise, but that it does not match the best existing ID estimation methods.

1. Introduction

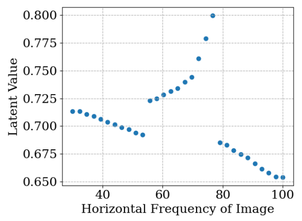
Autoencoder networks have historically been used as feature extraction methods (Hinton & Zemel, 1993), and more recently in generative models (Goodfellow et al., 2014). An autoencoder is trained to compress the input into a latent layer, such that these latent values can be used to reconstruct the original input. Although the latent layer is the bottleneck of the network, it may still be the case that there are *too many* latent neurons, if the dataset on which the network is trained has an underlying distribution that can be expressed with fewer variables than there are latent neurons. This number of variables needed to describe the dataset is know as the *intrinsic dimension* (ID) (Camastra & Staiano, 2016). In such a case, the network nonetheless uses all of the available latent neurons, resulting in an encoding that is redundant (Laakom et al., 2022).

Conventionally, the redundant parts of a network can be removed using structured network pruning[1]. Structured pruning aims to cause as little loss increase as possible per removed component, and therefore tends to prune layers that are large relative to their importance in the network.

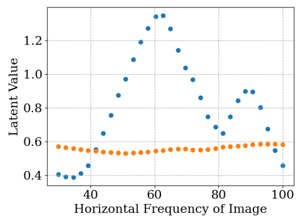
[1]Structured pruning is covered in the Background Knowledge



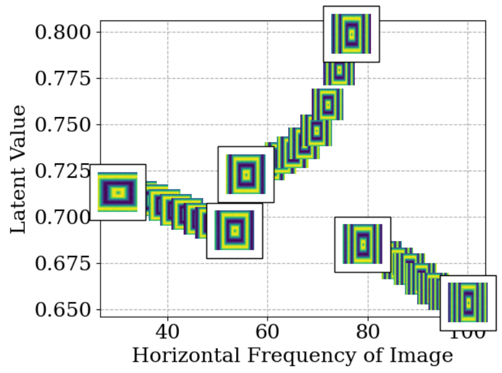
(a) Images with an intrinsic dimension of 1: each image is defined only by the horizontal frequency.



(b) Autoencoder with 1 latent neuron. The encoding is a bijective mapping to the horizontal frequency.



(c) Autoencoder with 2 latent neurons. Note how neither neuron is a bijective mapping to the horizontal frequency.



(d) Images from a overlaid on b. When the latent neurons match the ID, the latent space can become meaningful: in this case, it is segmented into three sections, corresponding to the number of vertical lines in the image.

Figure 1. The impact of a (mis)matching number of latent neurons.

However, the latent layer is already the smallest layer in the network, and pruning it will have the largest immediate impact on the loss, even if the encoding is redundant. This raises the question if naively applied structured pruning can recognise and remove redundant latent neurons, which we investigate in this paper.

There are two motivating factors for pruning redundant latent neurons. Firstly, pruning redundant neurons has a regularising effect on the latent space. As the example in Figure 1 shows, an autoencoder with too many latent neurons may have a redundant and less meaningful latent space.

Secondly, being able to find the correct number of latent neurons can be used in other fields: this can be seen as equivalent to finding the intrinsic dimension, which is an active field of study (Camastra & Staiano, 2016). A caveat for this equivalence is that the network must have sufficient complexity in the encoder and decoder to make the required transformation to and from the latent space.

To structurally prune latent neurons, we use the Early-Bird (EB) technique (You et al., 2019). EB regularises channels and neurons using the weight parameter of Batch Normalisation layers (Ioffe & Szegedy, 2015), and prunes those with the smallest weights. EB is notable because it needs only a fraction of a training run before it prunes the network. This is an improvement in efficiency over other structured pruning techniques, which often require a network to be fully trained, possibly multiple times, before pruning can occur.

The main research question of this paper is as follows:

What is the relation between the intrinsic dimension of a dataset and the weights of latent neurons in networks trained with the Early-Bird technique?

The following sub-questions are considered:

- Is this relation dependent on hyperparameters which have to be optimised separately, and if so: why?
- How effective is this application of the EB technique compared to existing methods of intrinsic dimension estimation?

The first sub-question is relevant because the EB technique has default values for its hyperparameters, but the novel application of EB in this paper might increase the sensitivity to hyperparameters, hence requiring a hyperparameter search each time the technique is applied in a different setting. A hyperparameter search would negate the key property of the EB technique that it can otherwise be applied in a single training run.
The second sub-question considers the use of the EB technique in the existing field of intrinsic dimension estimation, which has other requirements than the regularisation of autoencoders (these requirements are described in Section 2 and Section 5.4).

In researching these questions, this paper produces the following contributions:

- We demonstrate that a much higher regularisation strength applied specifically on the latent layer allows the Early-Bird technique to find the appropriate number of latent neurons for a given dataset. We do this on both 1D and image datasets.
- We propose an explanation for how this effect works, which also gives a possible cause for an observed difference in performance on convolutional autoencoders.
- We propose a method to prevent overpruning, by scaling the regularisation strength based on the loss.
- We compare removing redundant latent neurons with EB to contemporary intrinsic dimension estimation techniques.

The paper has the following structure. In Section 2, related and background work is discussed. In Section 3, we describe how synthetic data is constructed to have a specific ID, and how we train autoencoders on this data. In Section 4, the results of training are shown, as well as an experiment comparing to other ID estimation methods. In Section 5, the experimental results, potential weaknesses in the methodology, and future research directions are discussed. In Section 6 the paper is summarised and concluded.

Appendix A describes the pilot experiment in detail, Appendix B provides details for reproducing the results in this paper, and Appendix C contains additional figures.

2. Preliminaries and Related Work

This section covers fundamental work on which this paper builds, as well as comparable methods for regularising autoencoders and intrinsic dimension estimation. Due to the length of this section, it is divided into primary and secondary topics, where the primary topics relate most directly to the work in this paper, and the secondary topics are more focused on alternative views on the work in this paper.

2.1. Primary topics

Lottery Tickets. Lottery Tickets are a form of network pruning. Their introduction challenged previous assumptions about how network pruning can be applied and laid the foundation for Early-Bird tickets, which we use.
Frankle & Carbin (2018) first proposes the Lottery Ticket (LT) hypothesis:
"A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations." (Frankle & Carbin, 2018)

The key novelty in the LT hypothesis is that the trainable subnetwork exists as soon as the parent network is initialised; no pre-training of the parent network is required before pruning. Conversely, before this point, it was assumed that a network can only be pruned late in training, or after training is finished; an untrained, pruned network was said to be more difficult to train (Li et al., 2016).

The aforementioned subnetwork is defined by a binary mask M over the network. The mask and the initial (untrained) weights together form the Lottery Ticket. Frankle & Carbin demonstrate the LT hypothesis with unstructured pruning. Though they show the existence of LTs, they do not provide a method for extracting these tickets early in training.

Liu et al. (2018) contests the results of Frankle & Carbin by demonstrating that the initialisation of LTs does not always matter in the case of unstructured pruning (which Frankle & Carbin use), and *never* matters in the case of structured pruning. Since we use the Early-Bird technique, which is a structured method, we apply a similar experiment to Liu et al. (2018), and reach the same conclusion in Appendix A: the initialisation of the ticket does not matter.

Early-Bird. You et al. (2019) introduces Early-Bird (EB) Tickets. The EB technique is different from LT in a number of ways. Firstly, the pruning is changed to be structured, meaning that entire components are removed from the network, instead of sparsifying the weight matrices. Secondly, EB improves upon LT by requiring less training time before the subnetwork mask can be extracted: a stopping criterion is added, based on the Hamming distance between recent binary pruning masks, which triggers when this Hamming distance stays below a given threshold.

Chen et al. (2020) applies EB to BERT, and You et al. (2021) translates EB to the field of Graph CNNs. In this paper, we also use the Early-Bird technique in a new domain (autoencoders), but do not need to alter the implementation due to the similarity to a regular CNN.

Intrinsic dimension. We previously introduced the idea of a theoretical "correct" number of latent neurons. This is related to the intrinsic dimension (ID), a statistical and signal processing term for the number of variables required to describe a dataset or signal. Estimating the intrinsic dimension is a long-studied problem. Methods can generally be divided into two categories (Camastra & Staiano, 2016): global and local.

Global methods consider the entire dataset as lying on a single manifold, and try to estimate the ID of this manifold. For this, simple PCA can be used (by counting nonzero eigenvalues), but this tends to overestimate the ID. Variations of PCA have been formulated to resolve this (Bishop (1998), Zou et al. (2006), Li & Tao (2010)). Other categories of global methods are Multidimensional Scaling (Cox & Cox, 2008) and Fractal-based methods (Grassberger & Procaccia, 2004).

Conversely, local methods for ID estimation consider only the neighbourhood of a single datapoint at a time, and estimate the ID of the local topology. This allows local methods to produce different ID estimates for different parts of the dataset, though they can still be used to determine an average ID over the whole dataset. Categories of local methods include local-PCA (Fukunaga & Olsen, 1971), nearest-neighbour (Farahmand et al., 2007; Facco et al., 2017) and maximum-likelihood (Levina & Bickel, 2004; Amsaleg et al., 2019).

A complicating factor in ID estimation is that the correct ID is often ambiguous, as Campadelli et al. (2015) points out: depending on the scale at which the dataset is viewed, the apparent ID can have different values. One reason for this is noise: any dimension of the data that contains independent noise can arguably be seen as an intrinsic dimension needed to fully represent the signal. However, this is usually not the desired answer, and ID estimation techniques attempt to be robust to such noise. This is also called *multiscale robustness*.

Pruning redundant latent neurons can be seen as a form of ID estimation: if all redundant latent neurons are removed, and the remaining number allows any point in the dataset to be encoded, this number of latent neurons is the ID. By the above categorisation, ours would be a global method, since the autoencoder trains on the entire dataset. We compare our method to both local and global methods in Section 4.3.

2.2. Secondary topics

Neural Architecture Search. When structured network pruning is used to find a better topology for a network, it can be thought of as a type of Neural Architecture Search (NAS). NAS seeks to construct a fitting network topology for a given problem. A NAS method generally consists of three parts (Elsken et al., 2019): a search **space** of possible topologies, a **strategy** to navigate the search space, and a method to estimate the **performance** of a single candidate. Categories of search strategies include evolutionary approaches (Real et al., 2019; Miller et al., 1989), reinforcement learning (Zoph et al., 2018) and Bayesian optimisation (Domhan et al., 2015). **Model compression** is also closely related to NAS, since it can produce a smaller topology that is better suited to the problem. Since Early-Bird is based on on the model compression method Network Slimming (Liu et al., 2017), we also perform a type of NAS: the search space is the set of possible autoencoder topologies smaller than the starting topology, while the strategy for navigation is regularising neurons and channels during training.

Dimensionality reduction. Autoencoders with bottlenecked latent spaces implicitly perform dimensionality reduction when encoding the input data into the latent layer. Besides autoencoders, many other forms of classical and learned dimensionality reduction exist. These can generally be categorised as either feature selection, or feature extraction. Feature selection does not transform features, but only seeks to find a subset that best explains the dataset.

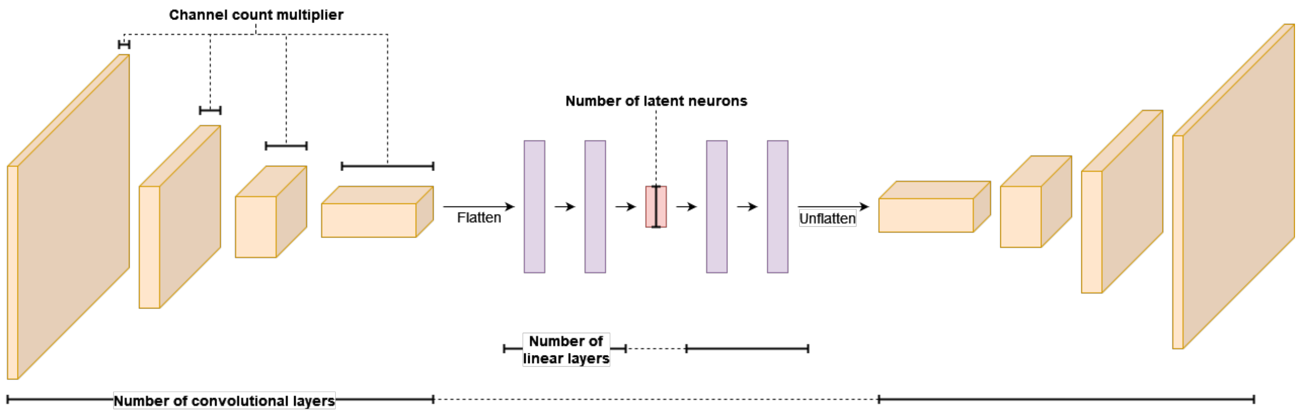


Figure 2. Parameterised Autoencoder. The hyperparameters governing the topology (number of latent neurons, etc.) are shown in bold.

Feature selection methods are typically not learned techniques, but instead a genetic algorithm (Shah & Kusiak, 2004), or a probabilistic technique like simulated annealing (Meiri & Zahavi, 2006). By contrast, feature extraction combines features to create novel ones, for example Kernel PCA (Schölkopf et al., 1997). Autoencoder networks are also widely applied as a form of feature extraction (Hinton & Zemel, 1993; Xing et al., 2016; Zabalza et al., 2016), with the advantages that they can be nonlinear, and scale well on image data using convolutional layers. Because of these advantages, we also focus on autoencoders.

Regularising the latent space of autoencoders. Without regularisation of some kind, autoencoders tend to have a latent space that is not usable for other applications, such as generative models or human-interpretable features. Extensions of autoencoders exist for this reason. Variational autoencoders (Kingma & Welling, 2013) make the latent space more continuous by sampling each latent variable from a learned parameterised distribution, and regularising those parameters with the KL-divergence. This also makes the latent space more suited for generative purposes. Sparse autoencoders (Goodfellow et al., 2016) take a different approach. Instead of containing a physical bottleneck in the number of latent neurons, they force a fixed proportion of the neurons to be disabled in each forward pass, based on the activation strengths. This allows the network to remain dense in terms of parameters, but keeps the information flow sparse, which helps with overfitting.
Similarly to this work, Laakom et al. (2022) attempts to reduce the redundancy of latent neurons: they minimise the pair-wise covariance between latent features, instead of removing redundant neurons, as in this work.
The regularisation of the autoencoder in this paper does not aim to enforce any kind of distribution of the latent variables, such as in VAEs. Also, where sparse autoencoders disable a fixed proportion of neurons, in this paper the number of removed latent neurons depends on the L1 regularisation.

3. Methodology

In this section, we describe the implementation and purpose of the paper's experiments in detail. Broadly speaking, an experiment consists of first constructing a dataset with a known intrinsic dimension, and then training an autoencoder with a number of latent neurons larger than the ID, while we regularise the network with EB. We investigate if this results in a visible relation between the weights of latent neurons and the ID. Specific details for the purposes of reproduction are given in Appendix B.

3.1. Model

The model used in the experiments is a standard autoencoder. For 2D input data, the linear layers are partially replaced by convolutional layers, as shown in Figure 2. This figure also shows the four hyperparameters that determine the topology: the number of latent neurons, the number of linear layers around the bottleneck, the number of convolutional layers, and a multiplier of the channel count.

3.2. Training and Early-Bird regularisation

We follow a typical training regime (App. B), with the notable addition of Early-Bird's L1 regularisation. This regularisation operates on Batch Normalisation layers, which we will now explain in detail. A BatchNorm layer applies two steps, first normalising an activation x:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}, \tag{1}$$

where μ is the sample mean, σ is the sample standard deviation, and ϵ is some small value. The BatchNorm layer then applies a scaling and shifting to the normalised activation:

$$y = \gamma \hat{x} + \beta, \tag{2}$$

where γ is called the weight parameter and β the bias, which are defined per channel or neuron. Note that for the rest of the paper, we use the phrase "the weight of a neuron" to refer to the weight parameter γ in the associated BatchNorm layer. The EB technique performs sub-gradient L1 regularisation on only the weight parameters: during training, after each backward pass, the gradient of all weight parameters is modified as follows:

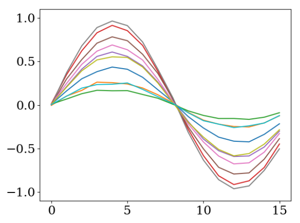
$$grad_\gamma = grad_\gamma + \lambda \cdot \text{sign}(\gamma), \tag{3}$$

where $grad_\gamma$ is the gradient of the weight, and λ is the regularisation strength. Note that because the gradient is modified after the backward pass, this L1 regularisation does not directly affect the rest of the network during the subsequent optimiser step. The indirect effects on the network are discussed in Section 5.2: it is hypothesised that over multiple training iterations, the sub-gradient regularisation effect propagates through the network through the influence on the loss.

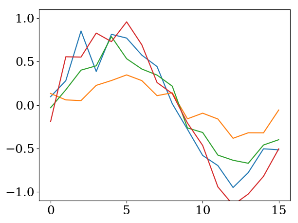
If the absolute value of γ for a latent neuron goes below 10^{-4} due to the regularisation, it is considered disabled, and the weight and gradient are permanently set to 0. This removes any information the latent neuron could pass to the decoder, though the bias is still a non-zero, learnable parameter. We make this addition (which is not present in EB) to ensure that no information can flow through disabled neurons: without this cutoff, a latent neuron with a weight of e.g. 10^{-5} might still be able to pass some information through, despite ostensibly not being used. The impact of this decision is discussed in Section 5.6.

3.3. Synthetic 1D Data

The goal of this experiment is to demonstrate whether the intrinsic dimension of a dataset can be extracted using the strength of the latent neurons during the training phase of the Early-Bird technique. To control the ID, we construct a synthetic dataset. A sample from this synthetic dataset is the output of a function, which contains a number of random variables: the number of random variables is equal to the intrinsic dimension of the dataset.



(a) Synthetic data with an ID of 1: each line is a datapoint, and only the amplitude varies; noise strength $\sigma = 0.01$.



(b) Increasing noise to $\sigma = 0.1$. Despite an ostensibly more complex signal, the ID is still 1.

Figure 3. Examples of synthetic data with an ID of 1.

For one-dimensional (vector) data, we use a sine function:

$$Sin_{1D}(x) = A \cdot \sin((x - P) \cdot F) + B, \tag{4}$$

where A is the amplitude, P the phase, F the frequency and B the bias. The intrinsic dimension can be set by leaving some variables at a constant value, while defining others as independent random variables. For example, if the amplitude and phase are random, and the bias and frequency constant, the resulting datapoints can be completely explained by only two numbers (amplitude and phase).
The sine wave is sampled at 16 points, which produces a 16-dimensional dataset sample. We then apply Gaussian noise to each value. As was discussed in the ID section of the Related Work, adding noise makes the true ID ambiguous, but ID estimation techniques should still be robust to such noise. An example of synthetic data with an ID of 1 is given in Figure 3.

3.4. Synthetic Image Data

For 2D image data, on which a convolutional autoencoder can be evaluated, we use a 2D sine, similarly to Equation (4):

$$Sin_{2D}(x, y) = A \cdot \sin(\max(F_x \cdot |x - P|, F_y \cdot |y - P|)). \tag{5}$$

This produces a concentric, rectangular sine wave with random frequency (F_x, F_y). The brightness of the rectangles is controlled by A, and the phase of the wave is determined by P. Examples of the produced synthetic images are shown in Figure 4. A rectangular sine wave is chosen (instead of e.g. an elliptical shape) so that the horizontal and vertical frequencies are axis-aligned, meaning that each can theoretically be estimated with only a single kernel. This makes the task well-suited for convolutional layers, meaning that the model can be kept relatively small. A small model implies faster training times and a lower risk of overfitting. This simplifies the experiments.

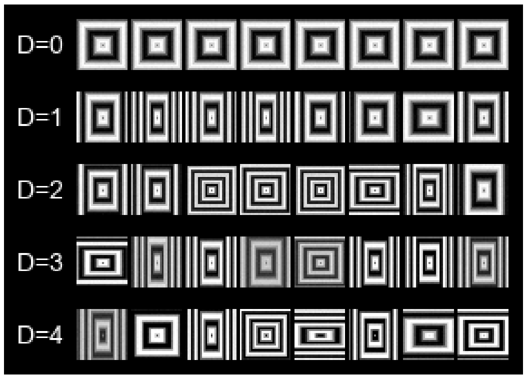


Figure 4. Example of synthetic images for increasing D. The variables are, in order of addition: X frequency, Y frequency, amplitude and phase.

3.5. Hyperparameters

There are three key hyperparameters that have to be set in each experimental run.

- D, the dimensionality of the dataset, which is determined by the number of random variables in the generator function.
- L, the initial number of latent neurons in the autoencoder network, which we pick to be larger than D.
- λ, the strength of the BatchNorm regularisation. For some experiments, this is defined per layer type, i.e. $\lambda_{linear} \neq \lambda_{latent} \neq \lambda_{conv}$, referring to the linear, latent, and (transpose) convolutional layers respectively.

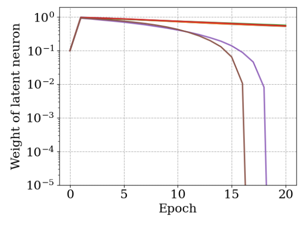
4. Experiments

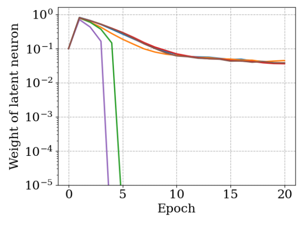
This section documents the results of the various experiments. Experiment 1 applies the methodology in the 1D case. Experiment 2 naively extends this to 2D, but finds that the regularisation is applied too soon in training, while the network is not yet fitted. Hence, a scaling of the regularisation is added, based on the current training loss. Experiment 3 compares the methodology to other intrinsic dimension estimation techniques.

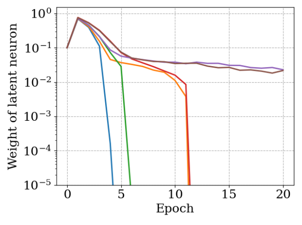
4.1. Experiment 1: latent neuron weights for high λ

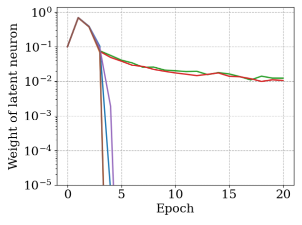
We start by evaluating the weights of the latent neurons over time in a linear-layer autoencoder, with $D = 2$ and $L = 6$, meaning that there are theoretically 4 redundant latent neurons. We vary λ from 10^{-4} to 10^{0} (note that Early-Bird uses $\lambda = 10^{-4}$ by default). In Figure 5, we plot the weight (γ) of all 6 latent neurons during training. It can be observed that for $10^{-2.5} \leq \lambda \leq 10^{-1}$, the weights of four latent neurons drop to zero, while the two remaining weights are stable. Ostensibly, the four disabled neurons were the redundant ones, and the remaining two latent neurons indicate that $D = 2$.

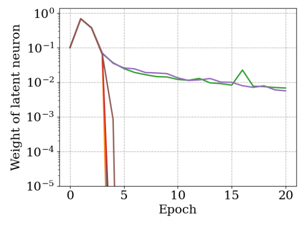
We examine when this disabling effect occurs by evaluating a wider range of hyperparameters in Table 1. D, λ_{linear}, and λ_{latent} are now varied separately. In each cell of the table, the number of active ($|\gamma| > 10^{-4}$) latent neurons is counted after training is complete.

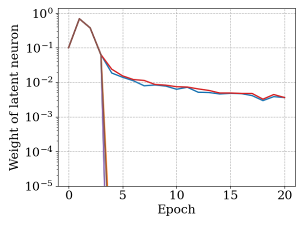

(a) $\lambda = 10^{-4}$

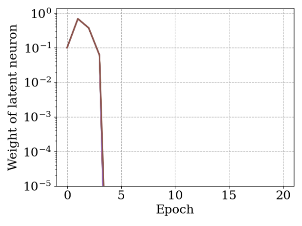

(b) $\lambda = 10^{-3}$


(c) $\lambda = 10^{-2.5}$


(d) $\lambda = 10^{-2}$


(e) $\lambda = 10^{-1.5}$


(f) $\lambda = 10^{-1}$


(g) $\lambda = 10^{-0.5}$

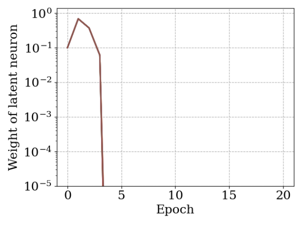

(h) $\lambda = 10^{0}$

Figure 5. Weight (γ) of each latent neuron over training epochs. $L = 6$ and $D = 2$.

Table 1. Hyperparameter search for experiment 1. Each value is the active latent neuron count after 20 epochs, median of 3 runs; $L = 8$; boldface indicates that the median latent count matches D; the median-absolute-deviation is given if it is not 0.

	$D = 1$			$D = 2$			$D = 3$			$D = 4$		
λ_{lin} / λ_{lat}	10^{-3}	10^{-2}	10^{-1}	10^{-3}	10^{-2}	10^{-1}	10^{-3}	10^{-2}	10^{-1}	10^{-3}	10^{-2}	10^{-1}
10^{-3}	**1**	**1**	0	4	**2**	**2**	5	**3**	**3**	7 (1)	**4**	3
10^{-2}	**1**	**1**	0	4	**2**	**2**	**3**	**3**	**3**	5	**4**	3
10^{-1}	**1**	**1**	0	**2**	**2**	**2**	**3**	**3**	**3**	**4** (1)	3	3
10^{0}	0	**1**	0	**2**	**2**	**2**	**3**	**3**	**3**	**4**	3	3

It can be observed that disabling all redundant neurons occurs under a relatively wide range of hyperparameters. There are two exceptions to this: under a high λ_{linear}, and for $D = 4$. However, there are still values for λ that disable the correct number of latent neurons for all D:
($\lambda_{lat} = 10^{-3}, \lambda_{lin} = 10^{-2}$), ($\lambda_{lat} = 10^{-2}, \lambda_{lin} = 10^{-3}$), and ($\lambda_{lat} = 10^{-1}, \lambda_{lin} = 10^{-3}$). Figure 9 (App. C) shows the weights of latent neurons using these values for λ.

4.2. Experiment 2: convolutional autoencoders

The setup of experiment 1 was replicated, but with a convolutional autoencoder and image dataset instead of their 1D equivalents. The results of this experiment differed greatly from what was seen in experiment 1, as can be seen in Table 2, which shows the number of remaining latent neurons: it can be observed that for higher D, there is an increasing proportion of runs where *all* latent neurons are removed. There is also a sizeable proportion of runs that only leave 1 neuron active when $D \geq 2$.

Table 2. Distribution of remaining active latent neurons when naively extending experiment 1 to image data.

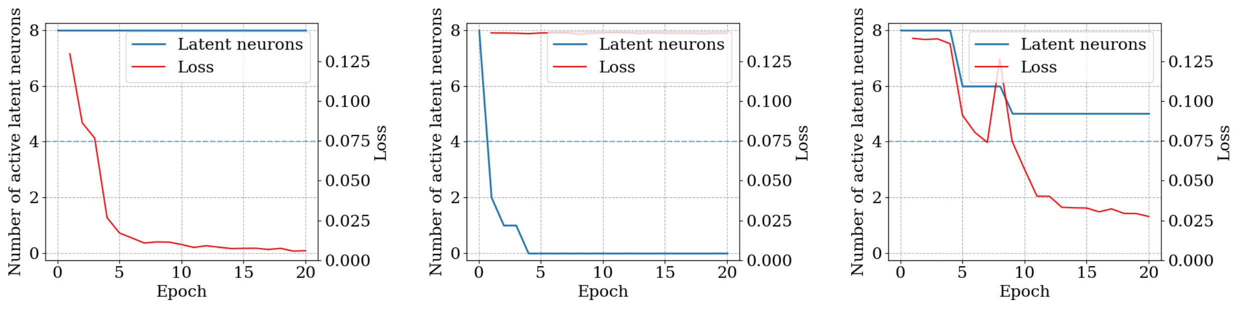
NEURONS	$D = 1$	$D = 2$	$D = 3$	$D = 4$
0	11%	19%	22%	31%
1	42%	53%	47%	33%
2+	47%	28%	31%	36%

We investigate this result in more detail in Figure 6. Figure 6a shows the normal training behaviour without any regularisation: most of the network convergence takes place in the first 5 epochs. In Figure 6b, we see the difference made by introducing a relatively small, constant regularisation of $\lambda = 10^{-3}$. Notably, the number of active latent neurons drops to 0 after only 4 epochs, giving the network no opportunity to converge to a state that actually uses the latent neurons before they are disabled.

We hypothesise that the effect of the regularisation is in some way proportional to the current loss: at a very high loss, the neurons are removed easily, and vice versa. This hypothesis is elaborated upon in Section 5.2. Accordingly, we make a simple alteration to how the L1 regularisation is applied: we use the training loss of the last batch to scale the strength of the regularisation. This is done via the following formula:

$$\lambda_{scaled} = \begin{cases} \lambda_{unscaled} \cdot \frac{t}{\mathcal{L}} & \text{if } \mathcal{L} < t \\ 0 & \text{otherwise} \end{cases}, \qquad (6)$$

where t is the loss threshold and $\mathcal{L}$ is the training loss. This means we do not apply the regularisation when the loss is above the threshold, and that a loss below the threshold causes proportionally stronger regularisation. t has to be determined manually, and corresponds to what the practitioner sees as a reasonable loss, indicating that the network



(a) Without regularisation: the network converges in 5 epochs, no latent neurons are removed.

(b) With constant regularisation: all latent neurons are removed before the network can converge.

(c) With scaling regularisation: neurons are pruned proportionally to the loss.

Figure 6. Impact of how regularisation is applied to convolutional autoencoders. $L = 8$, $D = 4$, $\lambda_{lat} = \lambda_{lin} = 10^{-3}$.

Table 3. Hyperparameter search for experiment 2: multiplicative scaling of regularisation based on loss. Note that the resulting neuron count is generally monotonous with increasing regularisation. Each value is the active latent neuron count after 20 epochs, median of 3 runs; $L = 8$; boldface indicates that the median latent count matches D; the median-absolute-deviation is given if it is not 0.

	$D = 1$			$D = 2$			$D = 3$			$D = 4$		
λ_{lat} \ λ_{lin}	10^{-3}	10^{-2}	10^{-1}	10^{-3}	10^{-2}	10^{-1}	10^{-3}	10^{-2}	10^{-1}	10^{-3}	10^{-2}	10^{-1}
10^{-3}	3	2 (1)	**1**	4 (1)	4	4 (1)	6	4	4 (1)	6 (1)	**4**	6
10^{-2}	2	2	**1**	3	3 (1)	3 (1)	4	4	**3**	**4** (1)	5	3
10^{-1}	**1**	**1**	**1**	**2**	**2**	**2**	**3**	**3**	2	**4**	3	3
10^{0}	**1**	**1**	**1**	1	**2**	1	2	2	2	3 (1)	2	3 (1)

is fitting to some degree. Based on the loss achieved thus far, we set $t = 0.1$. The positive results of this change can be observed in Figure 6c: neurons are not removed until the loss converges somewhat, and overpruning does not take place.

We once again perform a hyperparameter search, now with this addition. The results are shown in Table 3. It can be observed that the number of latent neurons found is largely monotonic with the regularisation strength. The sensitivity to hyperparameters is higher than in experiment 1, and again highest for $D = 4$. A positive observation is that a set of hyperparameters which generally worked in experiment 1 ($\lambda_{latent} = 10^{-1}$, $\lambda_{linear} = 10^{-3}$) also works for all D in this experiment. Figure 10 (App. C) shows the number of active latent neurons over time using these values for λ.

4.3. Experiment 3: comparison to other intrinsic dimension estimation methods

In this experiment, we answer the second sub-question by evaluating how usable Early-Bird is as a form of intrinsic dimension estimation. To reach a fair comparison, we first have to set our hyperparameters to fixed values, since we will also leave the hyperparameters of the other methods at their default values. Additionally, a key benefit of the Early-Bird technique is that it can be applied in a single-shot fashion; a hyperparameter optimisation phase would negate this. We choose values that worked well in both prior experiments, namely $\lambda_{latent} = 10^{-1}$ and $\lambda_{linear} = 10^{-3}$. Note that since this experiment uses the linear-layer autoencoder, we do not apply the loss scaling from Equation (6).

We compare our method against global and local intrinsic dimension estimation methods. The global methods are Correlation Dimension (Grassberger & Procaccia, 2004), and Fisher Separability (Albergante et al., 2019). The local methods are Fukunaga-Olsen lPCA (Fukunaga & Olsen, 1971), Manifold-Adaptive Dimension Estimation (Farahmand et al., 2007), Maximum Likelihood (Levina & Bickel, 2004), Method of Moments (Amsaleg et al., 2018), Tight Local Estimator (Amsaleg et al., 2019), and TwoNN (Facco et al., 2017). We do not re-implement these methods; instead we use an existing Python implementation by Bac et al. (2021).

We evaluate these methods on multiple datasets: the synthetic Sine wave described in Section 3.3 for various D and noise strengths, as well as a number of ID estimation benchmark datasets defined by Campadelli et al. (2015). The estimated ID produced by each method is listed in Table 4. We interpret these results in Section 5.4.

5. Discussion

In this section, we consider the implications and limitations of the experimental results.

5.1. Relation between ID and weights

We start by answering the main research question, which reads: “What is the relation between the intrinsic dimension of a dataset and the weights of latent neurons in networks trained with the Early-Bird technique?”.
This relation appears in the form of redundant neurons' BatchNorm weights dropping to zero, while other weights reach a steady, non-zero state. The intrinsic dimension can be inferred from the number of non-zero latent weights.
It should be noted that weights dropping to 0 is not necessarily the expected behaviour of the EB technique: according to Liu et al., the L1 regularisation can induce some sparsity, but the weights are generally only meant to have an ordering, such that a fraction of the smallest weights can be pruned in a separate step. Conversely, we do not see such an ordering of the weights in latent neurons (see Figure 9): they are either on or off. This is likely a result of λ being in an entirely different regime (1000x higher), and not a result of the change in architecture.

5.2. Explanation of L1 effects and results on image data

Before we can answer the first sub-question about the influence of hyperparameters, we should understand how EB can determine which neurons are redundant: the sub-gradient L1 regularisation is not applied through backpropagation, but it nonetheless appears to have an effect on other parts of the network, since regularising the latent layer can force the entire network to converge to a non-redundant encoding. We now give a possible explanation of how redundant latent neurons can be selected and removed by this indirect process:

- At some iteration I, the network is partially converged and attains a training loss $\mathcal{L}$.
- After the backward pass of I, but before the optimiser takes a step along the gradient, the gradient is modified by the EB regularisation: λ is added to the gradient, based on the current sign of the weight.
- The regularisation causes the weight to have a smaller absolute value in iteration $I + 1$. In this iteration, the narrowed spread of the latent weights causes a higher loss $\mathcal{L} + \epsilon$ to be attained. If a latent neuron was “important” i.e. the spread of its possible values was used to keep the loss at $\mathcal{L}$, it will have contributed a relatively large amount to this ϵ.

- Through backpropagation, the gradient of the latent neuron weights (before it is regularised) comes to reflect the neurons' importance to the network, based on the induced additional loss: a gradient is generated that has an opposite effect to the regularisation.
- Over many iterations, latent neurons that do not affect the loss enough will fail to generate a sufficient counter-gradient, and they will be removed from the network. Conversely, important latent neurons will reach a state of equilibrium, where the counter-gradient has a similar magnitude to the regularisation. This state of equilibrium can be observed in Figure 5, where the equilibrium point also depends on the value of λ.
- Notably, the other network parameters are not frozen during this process, and they also respond to the decreasing latent neuron weights through backpropagation of the loss: it is possible that a latent neuron is initially important, but after the rest of the network trains further, it is ultimately found to be redundant, and removed.

A key requirement for the process outlined above, is that the network must be in such a state that decreasing the weights results in a larger loss: $\mathcal{L} + \epsilon$. This is not the case if the network attains a high loss to begin with, either because it is fitting poorly, or just training slowly. To use informal phrasing: a latent neuron may *eventually* be relevant, but if the network is not fitted well enough to notice this relevance, the neuron will be removed regardless.

This can also explain why the initial, naive, application in Experiment 2 saw a large amount of overpruning compared to Experiment 1: while the linear-layer model fits extremely quickly, reaching minimal test and train loss within 1 epoch, the convolutional autoencoder takes much longer to fit. Hence, the linear model quickly has the ability to generate a counter-gradient that resists the regularisation for non-redundant latent neurons, but the convolutional model does not.

5.3. Regularisation strength

We now answer the first sub-question, which reads:
"Is [the relation between the ID and the BN weights of latent neurons] dependent on hyperparameters which have to be optimised separately, and if so: why?".

The results suggest that this relation is heavily dependent on using the correct value for λ: by setting λ lower or higher, an arbitrary number of remaining latent neurons can be reached. Hence, without prior knowledge, it may be difficult to identify the correct setting. However, there do appear to be sensible default values ($\lambda_{latent} = 10^{-1}$ and $\lambda_{linear} = 10^{-3}$) that work in both experiment 1 and 2.

The reason λ is so impactful, is because it dictates the trade-off between the loss and the number of remaining latent neurons. That also means λ is directly related to the scale of the loss: if every value in the dataset would be multiplied by a large factor, the scale of the loss would correspondingly increase, meaning that the gradients of weights would increase, and so the relative influence of a fixed λ would decrease. This implies that some kind of normalisation of the data is required in all cases, if a default value for λ is to be used.

Regarding the difference between λ_{latent} and λ_{linear}, we were surprised by Table 1: it suggests that increasing either λ_{latent} or λ_{linear} will cause more latent neurons to be removed. That is counter to the reasonable assumption that only λ_{latent} affects how strongly latent neurons are regularised.

Table 4. ID estimation of various methods on different datasets. If an estimate is correct, it is show in **boldface**. Since all correct ID's are integers, any fractal dimension is rounded to the nearest integer for the purpose of determining correctness.

Dataset	D	Ours	CorrInt	FisherS	lPCA	MADA	MLE	MOM	TLE	TwoNN
Sine (σ=0.01)	1	**1**	8.28	**0.92**	**1**	10.21	10.22	6.69	10.28	13.16
Sine (σ=0.1)	1	**1**	10.42	**0.92**	**1**	13.05	12.60	11.15	12.82	14.20
Sine (σ=0.01)	2	**2**	**2.24**	**1.97**	**2**	2.51	2.65	**2.12**	2.99	5.47
Sine (σ=0.1)	2	**2**	8.16	**1.96**	**2**	9.77	9.72	6.85	10.04	12.55
Sine (σ=0.01)	3	**3**	**2.93**	**2.96**	**3**	**3.21**	**3.03**	**2.95**	**3.33**	**3.43**
Sine (σ=0.1)	3	**3**	6.38	**2.97**	**3**	7.11	7.20	5.03	7.77	10.73
Sine (σ=0.01)	4	**4**	3.37	**3.61**	**4**	**3.99**	**3.73**	**3.73**	**4.10**	**4.06**
Sine (σ=0.1)	4	**4**	5.82	**3.67**	**4**	7.04	6.92	5.48	7.60	9.82
Swiss Roll	2	3	**1.94**	2.88	3	**2.09**	**1.96**	**1.99**	**2.12**	**1.93**
Paraboloid	3	**3**	2.44	**2.80**	**3**	**3.09**	**2.91**	**2.90**	**3.17**	**2.96**
Concentrated	4	3	3.45	**3.76**	6	**4.17**	**3.80**	**3.86**	**4.26**	**3.93**
Manifold	4	**4**	**3.77**	5.79	8	**4.24**	**3.90**	**4.26**	**4.36**	**3.95**
Hypercube	10	8	8.83	**10.31**	11	9.41	8.85	8.59	**9.79**	9.22
Affine20	20	18	14.15	19.16	**20**	15.52	14.85	14.13	15.83	16.18

For some of these cases, the cause is trivial: if $\lambda_{latent} = 10^{-3}$, increasing λ_{linear} to 10^{-1} will cause the latent regularisation to be stronger simply by making the layer before or after the latent neurons the new bottleneck. After this happens, it takes little regularisation to also disable the latent neurons. However, we also see the effect of λ_{linear} when $\lambda_{latent} >> \lambda_{linear}$ (for example under D=4 in Table 1). Here, we would expect there to be no bottleneck outside the latent layers, and so the results are less easy to explain. It may be the case that having a larger number of active neurons around the latent layer makes it more difficult to prune a latent neuron; perhaps because there is a higher chance that some of the following neurons depend on it, or because it is more complex for the network to transition to a state where the latent neuron is not used. Some amount of linear layer regularisation may help with this. It may also be the case that regularising linear layers, and correspondingly making the spread of these features narrower after each BatchNorm layer, decreases the variability of the data, thereby making it easier to fit said data in a smaller latent space.

5.4. Use as an ID estimation method

Here we interpret the results of Experiment 3 (shown in Table 4), where our method was compared against a range of ID estimation techniques.

We can broadly identify two groups of methods: those that work well on our Sine data (Ours, FisherS, lPCA), and those that work well on the ID benchmark datasets (MADA, MLE, MOM, TLE, TwoNN). The exception is the older CorrInt method, which Camastra & Staiano (2016) also discard in their comparison for being uncompetitive.

Although our technique may appear relatively competitive, the results on the Sine datasets should be taken with a grain of salt: the hyperparameters we selected were specifically chosen based on the performance on this same Sine dataset in experiment 1. Hence, these rows are not a fair direct comparison between our technique and the others, but instead serve to validate whether the Sine dataset is trivial as an ID estimation problem. Given the results of the other techniques, it seems estimating the ID of the Sine dataset is not trivial.

Our technique appears to perform most similarly to Fukunaga-Olsen local-PCA (Fukunaga & Olsen, 1971). A difference can be seen on the “Manifold” dataset, which is specifically described as as being nonlinear by Campadelli et al. (2015). This may explain why the (linear) PCA overestimates the ID in this case, which is also a flaw pointed out by Camastra & Staiano (2016).

The results on the “Affine” dataset may appear surprising: every technique gives the wrong answer in this case, apart from lPCA, which is otherwise not strong on the benchmark datasets. The Affine dataset consists of 20-dimensionsional datapoints, while the ID is also 20. Hence, there is no redundancy in the data: the datapoints are only subject to an affine transformation. It appears that most techniques mistakenly identify some of the less impactful dimensions as redundant, underestimating the ID. Conversely, lPCA consistently over-estimates the ID on all datasets. This seems to result in it accidentally finding the correct ID here, since the highest value that could be given was 20.

A key vulnerability of our technique is illustrated in the result on the “Concentrated” dataset. This dataset is described by Campadelli et al. as a “concentrated figure, mistakable with a 3-dimensional one”: it is a figure that has 3 clear intrinsic dimensions, and a fourth dimension which is less evident. Because this fourth intrinsic dimension has a low impact on the loss, the fourth remaining latent neuron seemingly did not generate a sufficient counter-gradient to stay active. This caused the ID to be underestimated.

Another observation of interest is robustness to noise (multiscaling robustness), which is a desirable property of ID estimation techniques. On the Sine dataset, changing the standard deviation of the noise from 0.01 to 0.1 (the impact of which is visualised in Figure 3) causes the local estimators, with the exception of lPCA, to overestimate the ID somewhat, possibly because this noise makes the local manifold topology seem more complex than it actually is. Our technique seems to be robust to this noise (note that this is the first time that we use a noise strength of 0.1 instead of 0.01). This may be because a deep, fully connected neural network has sufficient complexity to filter out such Gaussian noise and extract the most likely underlying signal.

5.5. Early-Bird as Model Compression

We believe there is a misconception in the original interpretation of Early-Bird tickets: the EB technique is based on Lottery Tickets, meaning that it considers the initialisation of the network as relevant, and also differentiates between *which* channels in a layer are removed. Due to the results from Liu et al. (2018), which we validated in autoencoders in Appendix A, we do not follow this viewpoint: the initialisation and choice of channels appears to have no impact on performance under structured pruning. Although You et al., the authors of EB, were aware of Liu et al. (2018), they did not consider the implication of that result: if the initialisation can be disregarded, EB is reduced to simple Model Compression, where the only product of the technique is a fractional value for each layer, indicating how much of the layer should be pruned. This is how we use EB in this paper.

5.6. Limitations

There are some methodological deficiencies and open questions not addressed in this paper, which we discuss here.

Hyperparameters. A crucial requirement for the usefulness of pruning redundant latent neurons is that it must be possible in a single run. If multiple runs are required due to hyperparameter optimisation of λ, the practitioner can just as easily train multiple autoencoders with a different number of latent neurons, and choose the network with the smallest number that still achieves an acceptable loss. Though we find a set of hyperparameters that works for all of our experiments, we still believe it was not sufficiently shown that a single choice of hyperparameters will work in a variety of cases.

Weight cutoff. During training, we disable any neuron if the absolute weight of the corresponding BatchNorm channel is below 10^{-4}. We do this to guarantee that information is not still passed through the "inactive" latent neurons. The value of this threshold was determined based on preliminary experiments, where it was observed that the majority of removed weights dropped far below 10^{-5}, and that the weights which did not drop dramatically generally stayed above 10^{-3}. The addition of this cutoff, as well as the chosen threshold value, may have further effects on the results, which are not studied in this paper. For instance, this cutoff may remove neurons too aggressively, since a neuron is removed if its weight drops below the threshold for even a single iteration, and this removal is permanent. Additionally, it may be the case that without the cutoff, neurons could re-enable if they continue to have a strong enough effect on the loss (and therefore a large enough gradient) to overcome the regularisation.

Distribution of results. The empirical results in this paper consist of the median of 3 identical, independent runs, due to hardware limitations. While this removes some variance from the data, it is not sufficient to study the distribution of the results. An improvement would be to perform more runs so that this distribution, and therefore the reliability of the technique, can be estimated.

Importance of dimensions. In defining the synthetic datasets, the random variables were purposefully distributed such that they would all have an approximately equal effect on the signal. For instance, the amplitude and bias were chosen from the same range. In a real-world scenario, some dimensions will certainly be less relevant than others. What if, for instance, the bias is constant (non-random) in 80% of the datapoints? This still makes it a relevant dimension, but one that has less impact on the loss.

Starting number of latent neurons. The value of L was set to be larger than D, but not extremely so ($L = 8$, $D = 1...4$). If we want to estimate D from a larger range, L would also have to be larger. However, it was not studied if a large number of latent neurons, say $L = 30$, would still work with e.g. $D = 1$.

Overfitting. On the synthetic datasets, no overfitting was observed. This was likely due to multiple factors: the data was relatively simple, the network was small, the test and training sets were drawn from the exact same distribution, and some gaussian noise was added. On real, messier data, it can be expected that we see some overfitting, which may impact the results.

5.7. Directions of Future Study

Stopping Criterion. Our work looks at the number of latent neurons that remains after the autoencoder is completely trained, i.e. when the loss has converged. However, the Early-Bird technique contains a stopping criterion based on the Hamming distance of network masks, which can be triggered after only a fraction of a training run. A direction of future research could be whether this stopping criterion can also identify the right moment to stop pruning latent neurons: when the binary mask stops changing throughout the whole network, it may be the case that the network has found a steady state where the regularisation of latent neurons is completely counterbalanced, implying that the correct number of latent neurons has been found.

Effect of removing all redundant neurons. The possibility that an autoencoder architecture can be given the exact number of required latent neurons raises questions of the impact of such non-redundancy on the latent space. The possible regularising effects were briefly alluded to in Section 1, but otherwise not studied in this paper. For instance, is the latent representation forced to be orthogonal in the absence of redundant neurons? Would this orthogonal representation contain features that are more human-interpretable?

6. Conclusion

The use of the Early-Bird technique was studied as a method to remove redundant latent neurons from autoencoders. It was found that setting the EB sub-gradient regularisation strength higher in the latent layer causes redundant latent neurons to have their weights reduced to near-zero, such that the remaining number of latent neurons matches the intrinsic dimension of the synthetic dataset. This effect was shown on both 1D and 2D synthetic data, provided an appropriate value for the regularisation strength is used. On 2D data, an additional step was needed, where the regularisation strength was scaled based on the current training loss, to avoid excessive removal of neurons early in training.

The ability to remove all redundant latent neurons presents a potential intrinsic dimension estimation technique, by counting the remaining latent neurons after training. Using EB in this manner was compared to existing ID estimation techniques: it was found that EB performs similarly to the local-PCA method, and better than some older global methods. However, it was not as accurate as state-of-the-art local methods on independent benchmark datasets. Notably, EB appears to be relatively robust to noise in the data.

References

Albergante, L., Bac, J., and Zinovyev, A. Estimating the effective dimension of large biological datasets using fisher separability analysis. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2019.

Amsaleg, L., Chelly, O., Furon, T., Girard, S., Houle, M. E., Kawarabayashi, K.-i., and Nett, M. Extreme-value-theoretic estimation of local intrinsic dimensionality. *Data Mining and Knowledge Discovery*, 32(6): 1768–1805, 2018.

Amsaleg, L., Chelly, O., Houle, M. E., Kawarabayashi, K.-I., Radovanović, M., and Treeratanajaru, W. Intrinsic dimensionality estimation within tight localities. In *Proceedings of the 2019 SIAM International Conference on Data Mining*, pp. 181–189. SIAM, 2019.

Bac, J., Mirkes, E. M., Gorban, A. N., Tyukin, I., and Zinovyev, A. Scikit-dimension: a python package for intrinsic dimension estimation. *Entropy*, 23(10):1368, 2021.

Bishop, C. Bayesian pca. *Advances in neural information processing systems*, 11, 1998.

Bjorck, N., Gomes, C. P., Selman, B., and Weinberger, K. Q. Understanding batch normalization. *Advances in neural information processing systems*, 31, 2018.

Camastra, F. and Staiano, A. Intrinsic dimension estimation: Advances and open problems. *Information Sciences*, 328: 26–41, 2016.

Campadelli, P., Casiraghi, E., Ceruti, C., and Rozza, A. Intrinsic dimension estimation: Relevant techniques and a benchmark framework. *Mathematical Problems in Engineering*, 2015, 2015.

Chen, X., Cheng, Y., Wang, S., Gan, Z., Wang, Z., and Liu, J. Earlybert: Efficient bert training via early-bird lottery tickets. *arXiv preprint arXiv:2101.00063*, 2020.

Cox, M. A. and Cox, T. F. Multidimensional scaling. In *Handbook of data visualization*, pp. 315–347. Springer, 2008.

Domhan, T., Springenberg, J. T., and Hutter, F. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

Elsken, T., Metzen, J. H., and Hutter, F. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

Facco, E., d'Errico, M., Rodriguez, A., and Laio, A. Estimating the intrinsic dimension of datasets by a minimal neighborhood information. *Scientific reports*, 7(1):1–8, 2017.

Farahmand, A. M., Szepesvári, C., and Audibert, J.-Y. Manifold-adaptive dimension estimation. In *Proceedings of the 24th international conference on Machine learning*, pp. 265–272, 2007.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 3 2018. URL http://arxiv.org/abs/1803.03635.

Fukunaga, K. and Olsen, D. R. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, 100(2):176–183, 1971.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*, chapter 14.2.1. MIT press, 2016.

Grassberger, P. and Procaccia, I. Measuring the strangeness of strange attractors. In *The theory of chaotic attractors*, pp. 170–189. Springer, 2004.

Hinton, G. E. and Zemel, R. Autoencoders, minimum description length and helmholtz free energy. *Advances in neural information processing systems*, 6, 1993.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Laakom, F., Raitoharju, J., Iosifidis, A., and Gabbouj, M. Reducing redundancy in the bottleneck representation of the autoencoders. *arXiv preprint arXiv:2202.04629*, 2022.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. In Touretzky, D. (ed.), *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1990. URL https://proceedings.neurips.cc/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

Levina, E. and Bickel, P. Maximum likelihood estimation of intrinsic dimension. *Advances in neural information processing systems*, 17, 2004.

Li, H., Kadav, A., Durdanovic, I., Samet, H., and Graf, H. P. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Li, J. and Tao, D. Simple exponential family pca. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 453–460. JMLR Workshop and Conference Proceedings, 2010.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE international conference on computer vision*, pp. 2736–2744, 2017. URL http://arxiv.org/abs/1708.06519.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 10 2018. URL http://arxiv.org/abs/1810.05270.

Luo, P., Wang, X., Shao, W., and Peng, Z. Towards understanding regularization in batch normalization. *arXiv preprint arXiv:1809.00846*, 2018.

Meiri, R. and Zahavi, J. Using simulated annealing to optimize the feature selection problem in marketing applications. *European journal of operational research*, 171 (3):842–858, 2006.

Miller, G. F., Todd, P. M., and Hegde, S. U. Designing neural networks using genetic algorithms. In *ICGA*, volume 89, pp. 379–384, 1989.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Aging evolution for image classifier architecture search. In *AAAI conference on artificial intelligence*, volume 3, 2019.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. How does batch normalization help optimization? *Advances in neural information processing systems*, 31, 2018.

Schölkopf, B., Smola, A., and Müller, K.-R. Kernel principal component analysis. In *International conference on artificial neural networks*, pp. 583–588. Springer, 1997.

Shah, S. C. and Kusiak, A. Data mining and genetic algorithm based gene/snp selection. *Artificial intelligence in medicine*, 31(3):183–196, 2004.

Xing, C., Ma, L., and Yang, X. Stacked denoise autoencoder based feature extraction and classification for hyperspectral images. *Journal of Sensors*, 2016, 2016.

You, H., Li, C., Xu, P., Fu, Y., Wang, Y., Chen, X., Baraniuk, R. G., Wang, Z., and Lin, Y. Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*, 9 2019. URL http://arxiv.org/abs/1909.11957.

You, H., Lu, Z., Zhou, Z., and Lin, Y. Gebt: drawing early-bird tickets in graph convolutional network training. *arXiv preprint arXiv:2103.00794*, 2021.

Zabalza, J., Ren, J., Zheng, J., Zhao, H., Qing, C., Yang, Z., Du, P., and Marshall, S. Novel segmented stacked autoencoder for effective dimensionality reduction and feature extraction in hyperspectral imaging. *Neurocomputing*, 185:1–10, 2016.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8697–8710, 2018.

Zou, H., Hastie, T., and Tibshirani, R. Sparse principal component analysis. *Journal of computational and graphical statistics*, 15(2):265–286, 2006.

A. Pilot Experiment

This appendix describes the initial experiment, which intended to entirely reproduce the EB-technique, and demonstrate that it functions as described by You et al. (2019) when used in autoencoders.

A.1. Experimental Setup

For the experiment, the Early-Bird technique was replicated in PyTorch (see Appendix B). The EB source code[2] was examined to verify that details were not missing in their paper, but no code was copied over. The model used is identical to the one described in Section 3.1.

The autoencoder is trained on the Fashion-MNIST dataset according to Section 3.2, with a notable addition: multiple times during training, a *mask drawing* phase occurs. Each "mask" is a binary mask over the network, that defines what channels should be pruned. The mask is calculated based on the *pruning ratio* and the BatchNorm weights: if the pruning ratio is 0.3, the lowest 30% of BatchNorm weights are marked for pruning in the network mask. In every mask drawing phase, we store the mask corresponding to each of the pruning ratios under study (0.1, 0.3, 0.5, and 0.7, in this experiment).

After all the masks have been drawn, we can determine the quality of a mask by retraining the network with that pruning mask applied. This happens according to the following procedure:

- Restore the convolutional autoencoder to its initial parameters before the first training.
- Structurally prune the network such that each convolutional, transpose convolutional and linear layer produces the number of channels expected by the binary mask. This also means that the input channel count of the following layer has to be pruned correspondingly.
- Train the pruned network for 20 epochs in normal fashion (without L1 regularisation).
- Track the training and test accuracy of the training pruned network per epoch.

The goal of the first experiment is to see if a sizeable fraction of the network can be pruned using EB without a significant increase in loss, as was previously demonstrated by You et al. (2019).

We also conduct a validation experiment, where we determine if the initialisation and chosen channels of a ticket matter. Frankle & Carbin (2018) states that they do have impact on the loss, but Liu et al. (2018) contests this if the pruning is structured, as it is here. For the experiment, we retrain two networks in order to compare them: one is the lottery ticket, including it's original initialisation, as the LT method requires. The other is a "random ticket", which has the same topology, but the initial parameters are randomised.

A.2. Pilot Results

Firstly, the network masks that were drawn could be compared using the mask distance, as is shown in Figure 7. Our heatmap appears to show a similar effect to that of You et al.: the mask distance gradually decreases, indicating that tickets "cool off" and settle into a stable state. Based on this heatmap we conclude that 8 epochs seems to be sufficient for the network mask to cool down. Hence, we use the network masks drawn in epoch 8 for the retraining phase.

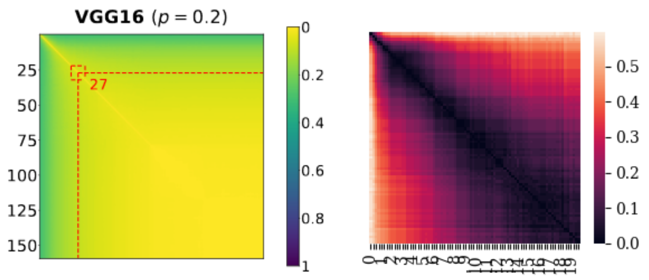


Figure 7. Heatmap of pairwise Hamming distance between network masks. The left figure is from You et al. (2019), the right figure is our result, when training over the course of 20 epochs.

The results of the retraining phase broadly indicate that EB is applicable to the autoencoder architecture. We find that up to 30% of the network can be structurally pruned without significant loss of accuracy, even if the tickets are drawn early in training. We also find that latent neurons are never pruned by the EB technique. This is sensible, since pruning latent neurons would have by far the greatest impact on the loss.

[2]The Early-Bird source code is available at `https://github.com/RICE-EIC/Early-Bird-Tickets`

Notably, the results of the validation experiments show that the initialisation of the network make no difference, as can be seen in Figure 8: for any particular pruning ratio, the randomly initialised network attains the same loss as the original ticket. This corresponds with the results of Liu et al. (2018), and goes against the phrasing of the original Lottery Ticket hypothesis. There is an important implication in this result: without the initialisation, the only product of the EB technique is a binary pruning mask, which defines a smaller topology. Hence, due to this result, EB is reduced to a form of neural architecture search, or model selection.

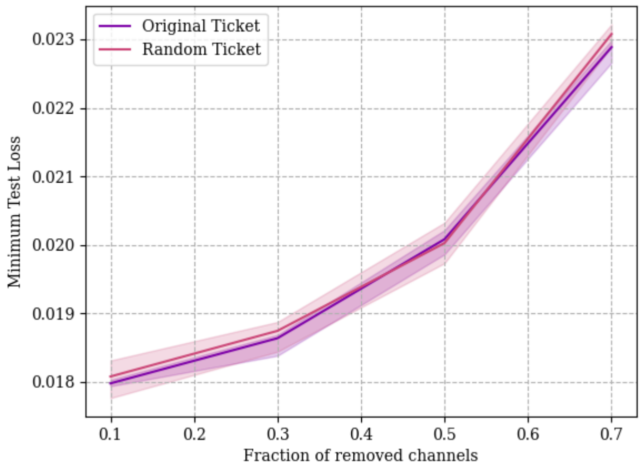


Figure 8. Attained loss for increasing pruning ratio. Note how the lottery ticket, with the specific initialisation, is equivalent to a randomly initialised ticket with equivalent pruning.

B. Reproducibility

B.1. Code

All code is available on GitHub: `https://github.com/RKlazinga/thesis-subnetworks-autoencoders`. The file `experiments/progressive_mask_drawing.py` is the usual starting point for an experiment: this performs a training run of a network according to the EB technique. All available settings and hyperparameters can be found in `settings.py`. The implementation of the subgradient descent L1 regularisation from EB can be found in `utils/subgrad_l1.py`.

B.2. Model

As shown in Figure 2, the autoencoder topology is defined by a number of hyperparameters. For experiments with the linear-layer autoencoder, these were as follows: 8 latent neurons, 4 hidden layers (both before and after the bottleneck latent layer), and a size multiplier of 1. For the convolutional autoencoder, these were as follows: 8 latent neurons, 2 linear layers, 5 convolutional layers, and a channel count multiplier of 12. These values were chosen by incrementing the size until the network could attain a reasonably low loss on the most complex dataset ($D = 4$). For experiment 3 (using a linear-layer autoencoder), the same shape parameters were used. However, the number of latent neurons was set equal to the number of input neurons, so that the ID estimate would not be limited by the initial number of latent neurons (e.g. if a dataset contained 12-dimensional datapoints and had an ID of 10, using 8 latent neurons would mean the ID estimate could never be larger than 8, so we start with 12 latent neurons).

B.3. Training

During training, we use Adam as the optimiser with a learning rate of 0.01, using MSE as the loss function. Networks are trained for 20 epochs on a training set of 20000 samples. If the training set is smaller than this, such as in the comparative experiment (5000 samples), the number of epochs is increased such that the total number of training iterations remains the same. The test set consists of 2000 samples.

B.4. Synthetic Data

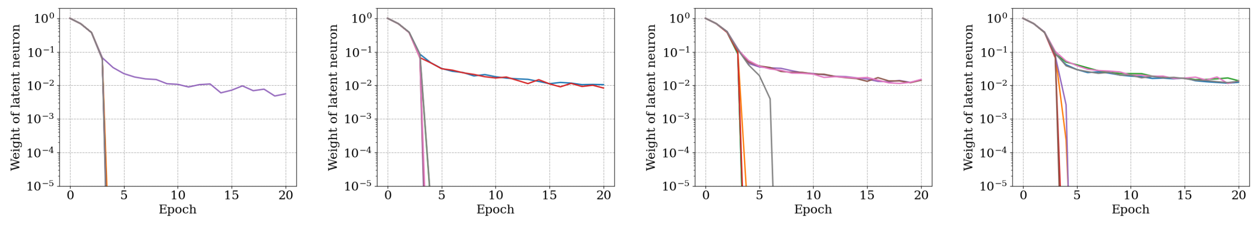
For synthetic 1D data, there are four random variables as described in Equation (4): amplitude, phase, bias, and frequency. When D was increased, these were added in that order. If the variable was set to be random instead of constant, the random value was uniformly chosen from a range. This range, as well as the default constant value for each variable, is given here. Note that all values assume a sine function that takes its input in degrees.

- $A \rightarrow$ default 0.5, random range $[0.1, 1]$
- $P \rightarrow$ default 0, random range $[0, 16]$
- $B \rightarrow$ default 0, random range $[0.1, 1]$
- $F \rightarrow$ default 22, random range $[11, 33]$

For synthetic 2D data, the following default values and random ranges were used:

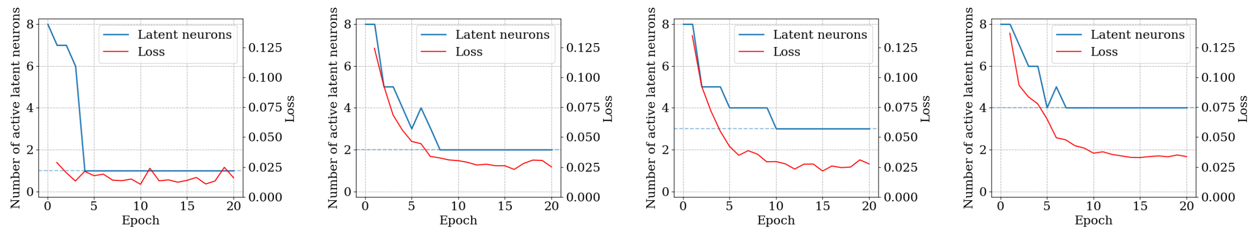
- $F_x \rightarrow$ default 40, random range $[30, 100]$
- $F_y \rightarrow$ default 40, random range $[30, 100]$
- $A \rightarrow$ default 0.5, random range $[0.25, 1]$
- $P \rightarrow$ default 0, random range $[0, 360]$

C. Additional Figures



(a) $D = 1$ (b) $D = 2$ (c) $D = 3$ (d) $D = 4$

Figure 9. Channel strength of each latent neuron over training epochs, from experiment 1 (linear-layer autoencoder with 1D data), after hyperparameter search; $\lambda_{latent} = 10^{-1}$, $\lambda_{linear} = 10^{-3}$



(a) $D = 1$ (b) $D = 2$ (c) $D = 3$ (d) $D = 4$

Figure 10. Number of active latent neurons and test loss, over training epochs, from experiment 2 (convolutional autoencoder with 2D data, including scaling regularisation per Equation (6)), after hyperparameter search; $\lambda_{latent} = 10^{-1}$, $\lambda_{linear} = 10^{-3}$

Background Knowledge

The following chapters provide background information for the reader, in support of the research paper above. A level of pre-existing knowledge on machine learning and deep learning is assumed, including the following concepts: tensors, neural networks, loss, gradient descent, and convolutions.

The following topics are discussed:

1. Batch-Normalisation

Batch Normalisation (Ioffe & Szegedy, 2015), abbreviated as BatchNorm or BN, is a normalisation technique that has been shown to greatly improve the stability of network training, especially in deep networks. Although the positive impact of batch normalisation is evident, the reason for *why* it works is poorly understood and the subject of ongoing debate ((Bjorck et al., 2018), (Luo et al., 2018), (Santurkar et al., 2018)). However, this discussion falls outside the scope of this paper.

A Batch Normalisation layer works by normalising the data flowing through the network along the **batch** dimension, using the sample mean and variance of each feature:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}, \tag{7}$$

where x is an activation, μ is the sample mean, σ is the sample standard deviation, and ϵ is some small value. The BatchNorm layer then applies a scaling and shifting to the normalised activation:

$$y = \gamma\hat{x} + \beta, \tag{8}$$

where γ is called the weight parameter and β the bias, which are defined per channel or neuron. These are learned parameters, so they have a gradient which is calculated during backpropagation, and they are updated by the optimiser like any other parameter. Note that the learned weight parameter γ will play an important role in the techniques discussed later.

2. Autoencoders

2.1. Topology

Autoencoders (AE) are a class of network structure falling under the category of unsupervised (or self-supervised) learning. This means they do not operate on labeled data: instead, given any input, their task is simply to provide an identical output.

This task is non-trivial due to the topology of autoencoders (see Figure 11). The network has a "bottleneck", splitting the topology into two portions: the **encoder** compresses the input to a **latent** representation. The second half of the network, called the **decoder**, then attempts to convert this latent representation back into something resembling the input.

Autoencoders may work on vectors of input data, on images using convolutional layers, or any other input format for a neural network.

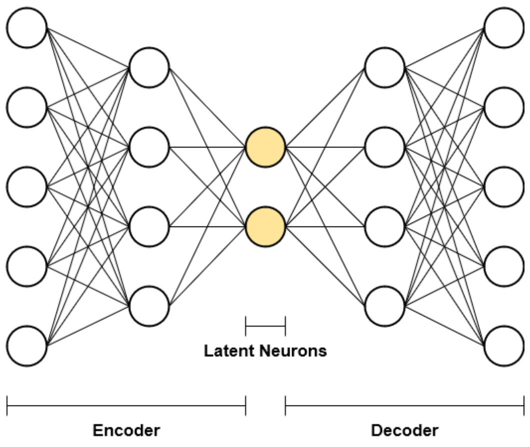


Figure 11. Autoencoder topology. Its task is to reproduce the input exactly in the output layer.

2.2. Usage

An assumption when using autoencoders is that the dataset it is trained on has some meaningful structure. If you would try to train a convolutional AE on images consisting only of white noise, it would not get very far, since such images do not contain a coherent signal, but instead a great number of individual, uncorrelated signals. There is no way to represent such an image without describing each pixel separately.

Conversely, an example of a dataset that *is* suitable for autoencoders is MNIST[5], a dataset of handwritten digits. The digits are encoded in a relatively large amount of data (28x28 brightness values), but fundamentally represent a very small range of possible images. Hence, an autoencoder can make a meaningful representation of these images using as little as 2 latent neurons.

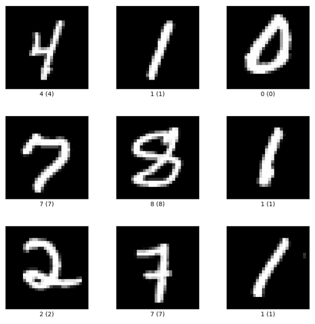


Figure 12. Sample of MNIST dataset[4]

Once an autoencoder is trained, it can be used in multiple ways. Using only the encoder, you can "compress" a new image from the same distribution, and decompress it using the decoder. You can also pick a random point in the latent space and feed it into the decoder to generate novel data in the same distribution.

In practice, autoencoders have been applied in a variety of ways, for example as a method of feature extraction, image denoising, image generation and translation of text.

[4]Image taken from https://www.tensorflow.org/datasets/catalog/mnist; accessed on 14-03-2022
[5]MNIST can be downloaded here: http://yann.lecun.com/exdb/mnist/

3. Pruning

Network pruning is a well-studied and broadly applied technique in neural networks. The core idea is that during training, some parts of the network may turn out to be less relevant than others. You can remove these less relevant components without (strongly) impacting the loss. In fact, the test loss may actually improve after pruning, because the remaining network is smaller and hence inherently less capable of overfitting. On top of this, pruning can improve the model size on disk, as well as inference and training speed. As early as 1990, LeCun et al. applied this idea in a technique called "Optimal Brain Damage".

Typically, pruning consists of three steps (Liu et al., 2018):

1. The network is trained for a number of epochs
2. The least relevant components of the network are identified and removed
3. The pruned network is fine-tuned further. In the case of **iterative** pruning, go back to step 2

The amount of pruning is determined by the **pruning ratio**, which is the fraction of the network that should be removed after pruning.

Pruning methods can be broadly divided into two types: **structured** and **unstructured** techniques. Unstructured pruning operates on individual connections between elements, such as a single weight between two neurons. This results in a sparse weight matrix. Conversely, structured pruning changes the network topology by completely removing a component, such as a neuron in a feed-forward layer, or a channel from a convolutional layer. Instead of producing a sparse weight matrix, this results in a weight matrix with smaller dimensions, because entire rows and columns are removed from the weight matrix. There are even structured pruning techniques that remove larger portions of the network, such as an entire layer, but these are not considered in this work.

The advantage of unstructured pruning is that it applies finer alterations than structured pruning. Hence, it allows more aggressive pruning, with pruning ratios up to 0.9 (90% of weights removed) without losing accuracy (Li et al., 2016). However, modern machine learning hardware cannot take advantage of this level of sparsity, so the network will be equally expensive to train and to perform inference with after training.

Structured pruning, by comparison, has the benefit that it produces an actually smaller network topology, which will be faster at both training and inference, and use less memory. However, because structured techniques are more "coarse", they cannot achieve the same pruning ratios as unstructured pruning without incurring a significant loss increase.

4. Lottery Tickets

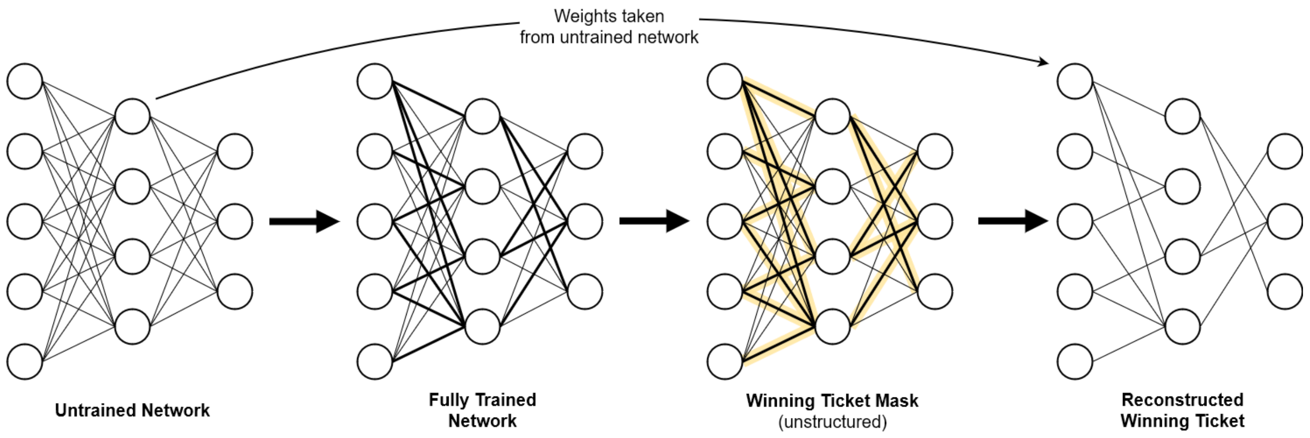
The Lottery Ticket hypothesis (Frankle & Carbin, 2018) brought a new view on pruning. Previously, it was assumed that a network can only be pruned late in training, or after training is finished; an untrained, pruned network is more difficult to train (Li et al., 2016). Hence, it was implied that the training process is necessary to transform the network from its initial state to a "prune-able" one.

Frankle & Carbin show that these assumptions do not always hold by stating and demonstrating the **Lottery Ticket Hypothesis**:

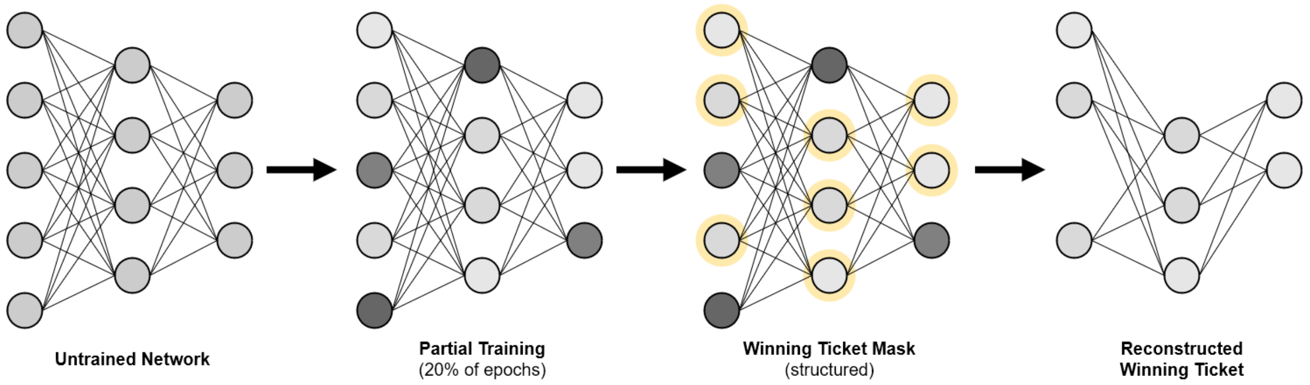
"A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations." (Frankle & Carbin, 2018)
This subnetwork is defined by a binary mask M over the network. The mask and the initial (untrained) weights together form the Lottery Ticket.

An important consequence of the existence of Lottery Tickets, is that these theoretically exist as soon as the network is instantiated. If the binary mask could be discovered, the pruned network would train well, even without any training before it is pruned (counter to the assumption in earlier literature).

Frankle & Carbin demonstrate the existence of Lottery Tickets by extracting them using **unstructured** weight pruning: a large network is trained on a task, and when this training is completed, a mask is defined based on the largest weights in the trained network. This mask is then applied to the network with its initial weights, and trained. They demonstrate that this process produces a network that achieves an equal or better test loss, despite having fewer parameters and not being trained before pruning. A visualisation of the technique is provided in Figure 13a. It should be noted that the approach by Frankle and Carbin is really only a theoretical demonstration, because it requires fully training the network before the mask can be extracted. They then show that *if* those masks were found earlier, the actual network weights would not require pre-training for the lottery ticket to train effectively.



(a) Lottery Ticket method by Frankle & Carbin. The network is first fully trained, after which the most important weights are identified, which defines the binary mask. This mask is applied to the *untrained* network, producing the winning ticket



(b) Early-Bird method by You et al.. After a fraction of the total training epochs, the Batch Normalisation channel weights are used to identify the most important channels, defining the binary mask. This mask is then applied without resetting the weights

Figure 13. Difference between Lottery Ticket extraction methods

5. Early-Bird Tickets

The Early-Bird (EB) technique (You et al., 2019) addresses a weakness in the original Lottery Tickets paper: the network had to be trained completely to find the subnetwork mask. You et al. change the pruning method to a **structured** technique, and demonstrate that this allows them to draw tickets (subnetwork masks) much earlier, after only 20% of training epochs. The differences between Lottery Tickets and Early-Bird are shown in Figure 13. The pruning technique used in Early-Bird is called Network Slimming.

5.1. Network Slimming

Network Slimming (Liu et al., 2017) is a structured pruning method that uses the Batch Normalisation (Ioffe & Szegedy, 2015) layers in a network to estimate the importance of channels. As discussed in Section 1, each BatchNorm layer contains learned weight parameters, one for every input feature. Liu et al. add L1 regularisation to the weight parameters. This provides a regularising pressure to the weights, which is countered by each weight's importance to the network. This naturally produces some sparsity, and forces the network to use as few channels as possible. The result is that the channel weights indicate the importance of channels, since some channels will retain a higher weight despite the regularisation, because of their importance to the network.

5.2. Usage

In the Early-Bird technique, a binary mask is defined over the channels in the network, such that it excludes the channels with the smallest BatchNorm weight. A stopping criterion is defined based on this mask, using the pairwise Hamming distance. When the distance between the last two masks stays below a threshold ϵ for a number of epochs, it implies that the ticket is not changing much anymore, and that it has "cooled down" enough to be usable. At this point, the training process stops and the ticket is extracted. An example of the pairwise Hamming distance is given in Figure 14.

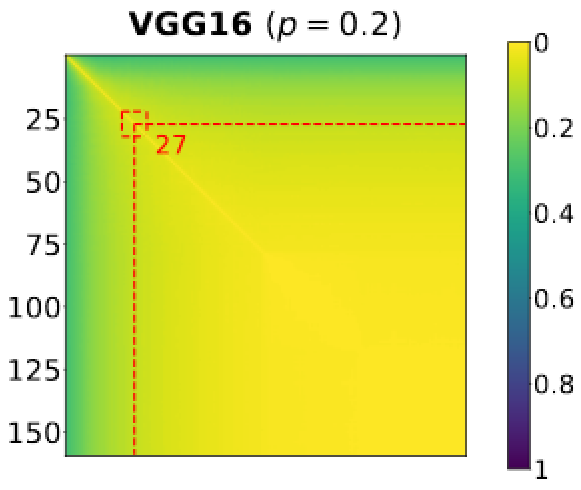


Figure 14. Pairwise Hamming distance over epochs ((You et al., 2019), Figure 3). A brighter color indicates a smaller Hamming distance, meaning the binary mask is changing relative little. The moment the stopping criterion is triggered is indicated by the dashed red line