Weight Agnostic Neural Networks

Adam Gaier*
Bonn-Rhein-Sieg University of Applied Sciences
Inria / CNRS / Université de Lorraine
adam.gaier@h-brs.de

David Ha
Google Brain
Tokyo, Japan
hadavid@google.com

Abstract

Not all neural network architectures are created equal, some perform much better than others for certain tasks. But how important are the weight parameters of a neural network compared to its architecture? In this work, we question to what extent neural network architectures alone, without learning any weight parameters, can encode solutions for a given task. We propose a search method for neural network architectures that can already perform a task without any explicit weight training. To evaluate these networks, we populate the connections with a single shared weight parameter sampled from a uniform random distribution, and measure the expected performance. We demonstrate that our method can find minimal neural network architectures that can perform several reinforcement learning tasks without weight training. On a supervised learning domain, we find network architectures that achieve much higher than chance accuracy on MNIST using random weights. Interactive version of this paper at https://weightagnostic.github.io/

1 Introduction

In biology, precocial species are those whose young already possess certain abilities from the moment of birth. There is evidence to show that lizard [73] and snake [13, 76] hatchlings already possess behaviors to escape from predators. Shortly after hatching, ducks are able to swim and eat on their own [104], and turkeys can visually recognize predators [28]. In contrast, when we train artificial agents to perform a task, we typically choose a neural network architecture we believe to be suitable for encoding a policy for the task, and find the weight parameters of this policy using a learning algorithm. Inspired by precocial behaviors evolved in nature, in this work, we develop neural networks with architectures that are naturally capable of performing a given task even when its weight parameters are randomly sampled. By using such neural network architectures, our agents can already perform well in their environment without the need to learn weight parameters.

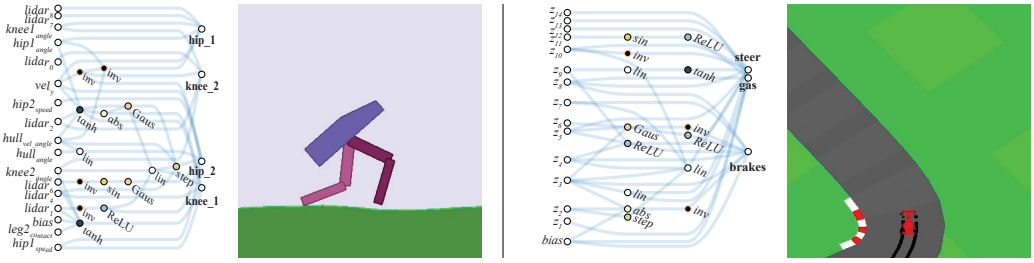

Figure 1: *Examples of Weight Agnostic Neural Networks: Bipedal Walker (left), Car Racing (right)* We search for architectures by deemphasizing weights. In place of training, networks are assigned a single shared weight value at each rollout. Architectures that are optimized for expected performance over a wide range of weight values are still able to perform various tasks without weight training.

*Work done while at Google Brain.

Decades of neural network research have provided building blocks with strong inductive biases for various task domains. Convolutional networks [24, 56] are especially suited for image processing [16]. For example, Ulyanov et al. [109] demonstrated that even a randomly-initialized CNN can be used as a handcrafted prior for image processing tasks such as superresolution and inpainting. Schmidhuber et al. [96] have shown that a randomly-initialized LSTM [45] with a learned linear output layer can predict time series where traditional RNNs fail. More recent developments in self-attention [113] and capsule [93] networks expand the toolkit of building blocks for creating architectures with strong inductive biases for various tasks. Fascinated by the intrinsic capabilities of randomly-initialized CNNs and LSTMs, we aim to search for *weight agnostic neural networks*, architectures with strong inductive biases that can already perform various tasks with random weights.

In order to find neural network architectures with strong inductive biases, we propose to search for architectures by deemphasizing the importance of weights. This is accomplished by **(1)** assigning a single shared weight parameter to every network connection and **(2)** evaluating the network on a wide range of this single weight parameter. In place of optimizing weights of a fixed network, we optimize instead for architectures that perform well over a wide range of weights. We demonstrate our approach can produce networks that can be expected to perform various continuous control tasks with a random weight parameter. As a proof of concept, we also apply our search method on a supervised learning domain, and find it can discover networks that, even without explicit weight training, can achieve a much higher than chance test accuracy of $\sim 92\%$ on MNIST. We hope our demonstration of such weight agnostic neural networks will encourage further research exploring novel neural network building blocks that not only possess useful inductive biases, but can also learn using algorithms that are not necessarily limited to gradient-based methods.[2]

2 Related Work

Our work has connections to existing work not only in deep learning, but also to various other fields:

Architecture Search Search algorithms for neural network topologies originated from the field of evolutionary computing in the 1990s [1, 8, 17, 25, 33, 40, 53, 59, 70, 71, 81, 86, 116, 117]. Our method is based on NEAT [103], an established topology search algorithm notable for its ability to optimize the weights and structure of networks simultaneously. In order to achieve state-of-the-art results, recent methods narrow the search space to architectures composed of basic building blocks with strong domain priors such as CNNs [64, 72, 89, 119], recurrent cells [48, 72, 119] and self-attention [100]. It has been shown that random search can already achieve SOTA results if such priors are used [63, 88, 97]. The inner loop for training the weights of each candidate architecture before evaluation makes the search costly, although efforts have been made to improve efficiency [9, 65, 85]. In our approach, we evaluate architectures without weight training, bypassing the costly inner loop, similar to the random trial approach in [44, 99] that evolved architectures to be more weight tolerant.

Bayesian Neural Networks The weight parameters of a BNN [3, 4, 26, 43, 68, 78] are not fixed values, but sampled from a distribution. While the parameters of this distribution can be learned [29, 54], the number of parameters is often greater than the number of weights. Recently, Neklyudov et al. [79] proposed variance networks, which sample each weight from a distribution with a zero mean and a learned variance parameter, and show that ensemble evaluations can improve performance on image recognition tasks. We employ a similar approach, sampling weights from a fixed uniform distribution with zero mean, as well as evaluating performance on network ensembles.

Algorithmic Information Theory In AIT [101], the Kolmogorov complexity [51] of a computable object is the minimum length of the program that can compute it. The Minimal Description Length (MDL) [34, 91, 92] is a formalization of Occam's razor, in which a good model is one that is best at compressing its data, including the cost of describing of the model itself. Ideas related to MDL for making neural networks "simple" was proposed in the 1990s, such as simplifying networks by soft-weight sharing [80], reducing the amount of information in weights by making them noisy [43], and simplifying the search space of its weights [95]. Recent works offer a modern treatment [6] and application [61, 108] of these principles in the context of larger, deep neural network architectures.

While the aforementioned works focus on the information capacity required to represent the *weights* of a predefined network architecture, in this work we focus on finding minimal *architectures* that can

[2] We release a software toolkit not only to facilitate reproduction, but also to further research in this direction. Refer to the Supplementary Materials for more information about the code repository.

represent solutions to various tasks. As our networks still require weights, we borrow ideas from AIT and BNN, and take them a bit further. Motivated by MDL, in our approach, we apply weight-sharing to the entire network and treat the weight as a random variable sampled from a fixed distribution.

Network Pruning By removing connections with small weight values from a trained neural network, pruning approaches [35, 39, 41, 58, 62, 66, 67, 69, 75] can produce sparse networks that keep only a small fraction of the connections, while maintaining similar performance on image classification tasks compared to the full network. By retaining the original weight initialization values, these sparse networks can even be trained from scratch to achieve a higher test accuracy [22] than the original network. Similar to our work, a concurrent work [118] found pruned networks that can achieve image classification accuracies that are much better than chance even with randomly initialized weights.

Network pruning is a complementary approach to ours; it starts with a full, trained network, and takes away connections, while in our approach, we start with no connections, and add complexity as needed. Compared to our approach, pruning requires prior training of the full network to obtain useful information about each weight in advance. In addition, the architectures produced by pruning are limited by the full network, while in our method there is no upper bound on the network's complexity.

Neuroscience A *connectome* [98] is the "wiring diagram" or mapping of all neural connections of the brain. While it is a challenge to map out the human connectome [102], with our 90 billion neurons and 150 trillion synapses, the connectome of simple organisms such as roundworms [112, 114] has been constructed, and recent works [20, 105] mapped out the entire brain of a small fruit fly. A motivation for examining the connectome, even of an insect, is that it will help guide future research on how the brain learns and represents memories in its connections. For humans it is evident, especially during early childhood [46, 107], that we learn skills and form memories by forming new synaptic connections, and our brain rewires itself based on our new experiences [5, 11, 18, 50].

The connectome can be viewed as a graph [12, 42, 110], and analyzed using rich tools from graph theory, network science and computer simulation. Our work also aims to learn network graphs that can encode skills and knowledge for an artificial agent in a simulation environment. By deemphasizing learning of weight parameters, we encourage the agent instead to develop ever-growing networks that can encode acquired skills based on its interactions with the environment. Like the connectome of simple organisms, the networks discovered by our approach are small enough to be analyzed.

3 Weight Agnostic Neural Network Search

Creating network architectures which encode solutions is a fundamentally different problem than that addressed by neural architecture search (NAS). The goal of NAS techniques is to produce architectures which, once trained, outperform those designed by humans. It is never claimed that the solution is innate to the structure of the network. Networks created by NAS are exceedingly 'trainable' – but no one supposes these networks will solve the task without training the weights. The weights *are* the solution; the found architectures merely a better substrate for the weights to inhabit.

To produce architectures that themselves encode solutions, the importance of weights must be minimized. Rather than judging networks by their performance with optimal weight values, we can instead measure their performance when their weight values are drawn from a random distribution. Replacing weight training with weight sampling ensures that performance is a product of the network topology alone. Unfortunately, due to the high dimensionality, reliable sampling of the weight space is infeasible for all but the simplest of networks.

Though the curse of dimensionality prevents us from efficiently sampling high dimensional weight spaces, by enforcing weight-sharing on *all* weights, the number of weight values is reduced to one. Systematically sampling a single weight value is straight-forward and efficient, enabling us to approximate network performance in only a handful of trials. This approximation can then be used to drive the search for ever better architectures.

The search for these weight agnostic neural networks (WANNs) can be summarized as follows (See Figure 2 for an overview): **(1)** An initial population of minimal neural network topologies is created, **(2)** each network is evaluated over multiple rollouts, with a different shared weight value assigned at each rollout, **(3)** networks are ranked according to their performance *and* complexity, and **(4)** a new population is created by varying the highest ranked network topologies, chosen probabilistically through tournament selection [74]. The algorithm then repeats from **(2)**, yielding weight agnostic topologies of gradually increasing complexity that perform better over successive generations.

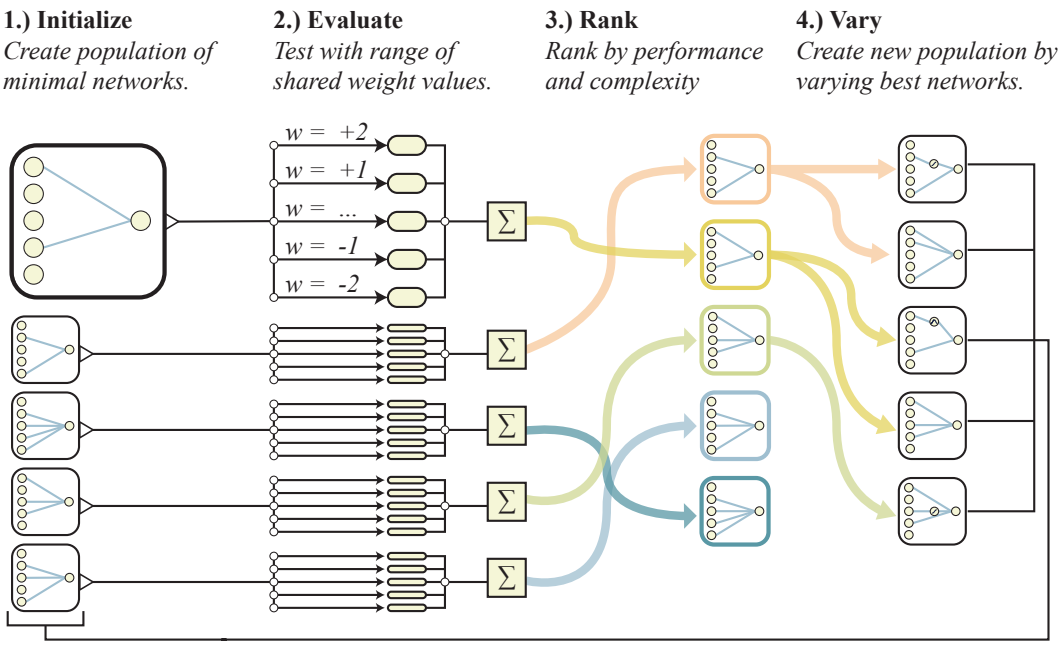

Figure 2: *Overview of Weight Agnostic Neural Network Search*
Weight Agnostic Neural Network Search avoids weight training while exploring the space of neural network topologies by sampling a single shared weight at each rollout. Networks are evaluated over several rollouts. At each rollout a value for the single shared weight is assigned and the cumulative reward over the trial is recorded. The population of networks is then ranked according to their performance and complexity. The highest ranking networks are then chosen probabilistically and varied randomly to form a new population, and the process repeats.

Topology Search The operators used to search for neural network topologies are inspired by the well-established neuroevolution algorithm NEAT [103]. While in NEAT the topology and weight values are optimized simultaneously, we ignore the weights and apply only topological search operators.

The initial population is composed of sparsely connected networks, networks with no hidden nodes and only a fraction of the possible connections between input and output. New networks are created by modifying existing networks using one of three operators: insert node, add connection, or change activation (Figure 3). To insert a node, we split an existing connection into two connections that pass through this new hidden node. The activation function of this new node is randomly assigned. New connections are added between previously unconnected nodes, respecting the feed-forward property of the network. When activation functions of hidden nodes are changed, they are assigned at random. Activation functions include both the common (e.g. linear, sigmoid, ReLU) and more exotic (Gaussian, sinusoid, step), encoding a variety of relationships between inputs and outputs.

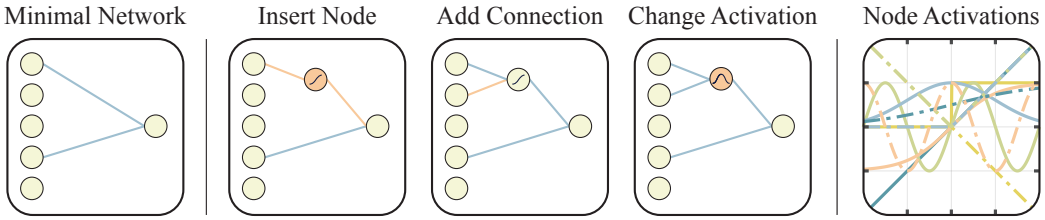

Figure 3: *Operators for searching the space of network topologies*
Left: A minimal network topology, with input and outputs only partially connected.
Middle: Networks are altered in one of three ways. *Insert Node*: a new node is inserted by splitting an existing connection. *Add Connection*: a new connection is added by connecting two previously unconnected nodes. *Change Activation*: the activation function of a hidden node is reassigned.
Right: Possible activation functions (linear, step, sin, cosine, Gaussian, tanh, sigmoid, inverse, absolute value, ReLU) shown over the range $[2, 2]$.

Performance and Complexity Network topologies are evaluated using several shared weight values. At each rollout a new weight value is assigned to *all* connections, and the network is tested on the task. In these experiments we used a fixed series of weight values ($[-2, -1, -0.5, +0.5, +1, +2]$) to decrease the variance between evaluations.[3] We calculate the mean performance of a network topology by averaging its cumulative reward over all rollouts using these different weight values.

Motivated by algorithmic information theory [101], we are not interested in searching merely for *any* weight agnostic neural networks, but networks that can be described with a minimal description length [34, 91, 92]. Given two different networks with similar performance we prefer the simpler network. By formulating the search as a multi-objective optimization problem [52, 77] we take into account the size of the network as well as its performance when ranking it in the population.

We apply the connection cost technique from [15] shown to produce networks that are more simple, modular, and evolvable. Networks topologies are judged based on three criteria: mean performance over all weight values, max performance of the single best weight value, and the number of connections in the network. Rather than attempting to balance these criteria with a hand-crafted reward function for each new task, we rank the solutions based on dominance relations [19].

Ranking networks in this way requires that any increase in complexity is accompanied by an increase in performance. While encouraging minimal and modular networks, this constraint can make larger structural changes – which may require several additions before paying off – difficult to achieve. To relax this constraint we rank by complexity only probabilistically: in 80% of cases networks are ranked according to mean performance and the number of connections, in the other 20% ranking is done by mean performance and max performance.

4 Experimental Results

Continuous Control Weight agnostic neural networks (WANNs) are evaluated on three continuous control tasks. The first, `CartPoleSwingUp`, is a classic control problem where, given a cart-pole system, a pole must be swung from a resting to upright position and then balanced, without the cart going beyond the bounds of the track. The swingup task is more challenging than the simpler `CartPole` [10], where the pole starts upright. Unlike the simpler task, it cannot be solved with a linear controller [87, 106]. The reward at every timestep is based on the distance of the cart from track edge and the angle of the pole. Our environment is closely based on the one described in [27, 120].

The second task, `BipedalWalker-v2` [10], is to guide a two-legged agent across randomly generated terrain. Rewards are awarded for distance traveled, with a cost for motor torque to encourage efficient movement. Each leg is controlled by a hip and knee joint in reaction to 24 inputs, including LIDAR sensors which detect the terrain and proprioceptive information such as the agent's joint speeds. Compared to the low dimensional `CartPoleSwingUp`, `BipedalWalker-v2` has a non-trivial number of possible connections, requiring WANNs to be selective about the wiring of inputs to outputs.

The third, `CarRacing-v0` [10], is a top-down car racing from pixels environment. A car, controlled with three continuous commands (gas, steer, brake) is tasked with visiting as many tiles as possible of a randomly generated track within a time limit. Following the approach described in [38], we delegate the pixel interpretation element of the task to a pre-trained variational autoencoder [49, 90] (VAE) which compresses the pixel representation to 16 latent dimensions. These dimensions are given as input to the network. The use of learned features tests the ability of WANNs to learn abstract associations rather than encoding explicit geometric relationships between inputs.

Hand-designed networks found in the literature [37, 38] are compared to the best weight agnostic networks found for each task. We compare the mean performance over 100 trials under 4 conditions:

1. *Random weights*: individual weights drawn from $\mathcal{U}(-2, 2)$;
2. *Random shared weight*: a single shared weight drawn from $\mathcal{U}(-2, 2)$;
3. *Tuned shared weight*: the highest performing shared weight value in range $(-2, 2)$;
4. *Tuned weights*: individual weights tuned using population-based REINFORCE [115].

[3] Variations on these particular values had little effect, though weight values in the range $[-2, 2]$ showed the most variance in performance. Networks whose weight values were set to greater than 3 tended to perform similarly – presumably saturating many of the activation functions. Weight values near 0 were also omitted to reduce computation, as regardless of the topology little to no signal was sent to the output.

Table 1: *Performance of Randomly Sampled and Trained Weights for Continuous Control Tasks*
We compare the mean performance (over 100 trials) of the best weight agnostic network architectures found with standard feed forward network policies commonly used in previous work (i.e. [37, 38]). The intrinsic bias of a network topology can be observed by measuring its performance using a shared weight sampled from a uniform distribution. By tuning this shared weight parameter we can measure its maximum performance. To facilitate comparison to baseline architectures we also conduct experiments where networks are allowed unique weight parameters and tuned.

Swing Up	Random Weights	Random Shared Weight	Tuned Shared Weight	Tuned Weights
WANN	**57 ± 121**	**515 ± 58**	**723 ± 16**	**932 ± 6**
Fixed Topology	21 ± 43	7 ± 2	8 ± 1	918 ± 7

Biped	Random Weights	Random Shared Weight	Tuned Shared Weight	Tuned Weights
WANN	**-46 ± 54**	**51 ± 108**	**261 ± 58**	332 ± 1
Fixed Topology	-129 ± 28	-107 ± 12	-35 ± 23	**347 ± 1** [37]

CarRacing	Random Weights	Random Shared Weight	Tuned Shared Weight	Tuned Weights
WANN	**-69 ± 31**	**375 ± 177**	**608 ± 161**	893 ± 74
Fixed Topology	-82 ± 13	-85 ± 27	-37 ± 36	**906 ± 21** [38]

The results are summarized in Table 1.[4] In contrast to the conventional fixed topology networks used as baselines, which only produce useful behaviors after extensive tuning, WANNs perform even with random shared weights. Though their architectures encode a strong bias toward solutions, WANNs are not completely independent of the weight values – they do fail when individual weight values are assigned randomly. WANNs function by encoding relationships between inputs and outputs, and so while the importance of the magnitude of the weights is not critical, their consistency, especially consistency of sign, is. An added benefit of a single shared weight is that it becomes trivial to tune this single parameter, without requiring the use of gradient-based methods.

The best performing shared weight value produces satisfactory if not optimal behaviors: a balanced pole after a few swings, effective if inefficient gaits, wild driving behaviour that cuts corners. These basic behaviors are encoded entirely within the architecture of the network. And while WANNs are able to perform without training, this predisposition does not prevent them from reaching similar state-of-the-art performance when the weights *are* trained.

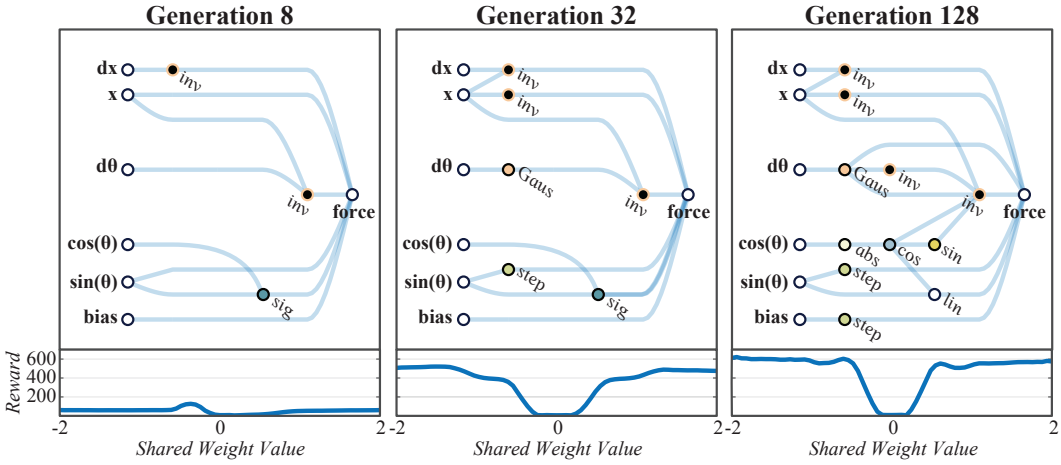

Figure 4: *Development of Weight Agnostic topologies over time*
Generation 8: An early network which performs poorly with nearly all weights.
Generation 32: Relationships between the position of the cart and velocity of the pole are established. The tension between these relationships produces both centering and swing-up behavior.
Generation 128: Complexity is added to refine the balancing behavior of the elevated pole.

As the networks discovered are small enough to interpret, we can derive insights into how they function by looking at network diagrams (See Figure 4). Examining the development of a WANN which solves `CartPoleSwingUp` is also illustrative of how relationships are encoded within an architecture. In the earliest generations the space of networks is explored in an essentially random fashion. By generation 32, preliminary structures arise which allow for consistent performance: the

[4] We conduct several independent search runs to measure variability of results in Supplementary Materials.

three inverters applied to the x position keep the cart from leaving the track. The center of the track is at 0, left is negative, right is positive. By applying positive force when the cart is in a negative position and vice versa a strong attractor towards the center of the track is encoded.

The interaction between the regulation of position and the Gaussian activation on $d\theta$ is responsible for the swing-up behavior, also developed by generation 32. At the start of the trial the pole is stationary: the Gaussian activation of $d\theta$ is 1 and force is applied. As the pole moves toward the edge the nodes connected to the x input, which keep the cart in the center, begin sending an opposing force signal. The cart's progress toward the edge is slowed and the change in acceleration causes the pole to swing, increasing $d\theta$ and so decreasing the signal that is pushing the cart toward the edge. This slow down causes further acceleration of the pole, setting in motion a feedback loop that results in the rapid dissipation of signal from $d\theta$. The resulting snap back of the cart towards the center causes the pole to swing up. As the pole falls and settles the same swing up behavior is repeated, and the controller is rewarded whenever the pole is upright.

As the search process continues, some of these controllers linger in the upright position longer than others, and by generation 128, the lingering duration is long enough for the pole to be kept balanced. Though this more complicated balancing mechanism is less reliable under variable weights than the swing-up and centering behaviors, the more reliable behaviors ensure that the system recovers and tries again until a balanced state is found. Notably, as these networks encode relationships and rely on tension between systems set against each other, their behavior is consistent with a wide range of shared weight values. For video demonstrations of the policies learned at various developmental phases of the weight agnostic topologies, please refer to the supplementary website.

WANN controllers for `BipedalWalker-v2` and `CarRacing-v0` (Figure 1, page 1) are likewise remarkable in their simplicity and modularity. The biped controller uses only 17 of the 25 possible inputs, ignoring many LIDAR sensors and knee speeds. The WANN architecture not only solves the task without training the individual weights, but uses only 210 connections, an order of magnitude fewer than commonly used topologies (2804 connections used in the SOTA baseline [37]).

The architecture which encodes stable driving behavior in the car racer is also striking in its simplicity (Figure 1, right). Only a sparsely connected two layer network and a single weight value is required to encode competent driving behavior. While the SOTA baseline [38] also gave the hidden states of a pre-trained RNN world model, in addition to the VAE's representation to its controller, our controller operates on the VAE's latent space alone. Nonetheless, it was able to develop a feed-forward controller that achieves a comparable score. Future work will explore removing the feed-forward constraint from the search to allow WANNs to develop recurrent connections with memory states.

Classification Promising results on reinforcement learning tasks lead us to consider how widely a WANN approach can be applied. WANNs which encode relationships between inputs are well suited to RL tasks: low-dimensional inputs coupled with internal states and environmental interaction allow discovery of reactive and adaptive controllers. Classification, however, is a far less fuzzy and forgiving problem. A problem where, unlike RL, design of architectures has long been a focus. As a proof of concept, we investigate how WANNs perform on the MNIST dataset [55], an image classification task which has been a focus of human-led architecture design for decades [14, 57, 93].

Even in this high-dimensional classification task WANNs perform remarkably well (Figure 5, Left). Restricted to a single weight value, WANNs are able to classify MNIST digits as well as a single layer neural network with thousands of weights trained by gradient descent. The architectures created still maintain the flexibility to allow weight training, allowing further improvements in accuracy.

It is straight forward to sweep over the range of weights to find the value which performs best on the training set, but the structure of WANNs offers another intriguing possibility. At each weight value the prediction of a WANN is different. On MNIST this can be seen in the varied accuracy on each digit (Figure 5, Right). Each weight value of the network can be thought of as a distinct classifier, creating the possibility of using one WANN with multiple weight values as a self-contained ensemble.

In the simplest ensemble approach, a collection of networks are created by instantiating a WANN with a range of weight values. Each of these networks is given a single vote, and the ensemble classifies samples according to the category which received the most votes. This approach yields predictions far more accurate than randomly selected weight values, and only slightly worse than the best possible weight. That the result of this naive ensemble is successful is encouraging for experimenting with more sophisticated ensemble techniques when making predictions or searching for architectures.

WANN	Test Accuracy
Random Weight	*82.0% ± 18.7%*
Ensemble Weights	*91.6%*
Tuned Weight	*91.9%*
Trained Weights	*94.2%*

ANN	Test Accuracy
Linear Regression	*91.6%* [57]
Two-Layer CNN	*99.3%* [14]

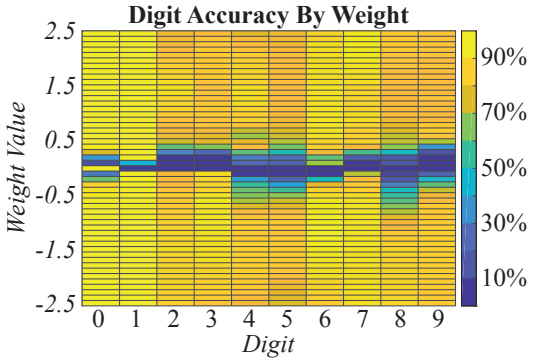

Figure 5: *Classification Accuracy on MNIST.*
Left: WANNs instantiated with multiple weight values acting as an ensemble perform far better than when weights are sampled at random, and as well as a linear classifier with thousands of weights. *Right:* No single weight value has better accuracy on all digits. That WANNs can be instantiated as several *different* networks has intriguing possibilities for the creation of ensembles.

5 Discussion and Future Work

In this work we introduced a method to search for simple neural network architectures with strong inductive biases for performing a given task. Since the networks are optimized to perform well using a single weight parameter over a range of values, this single parameter can easily be tuned to increase performance. Individual weights can be further tuned from a best shared weight. The ability to quickly fine-tune weights is useful in few-shot learning [21] and may find uses in continual lifelong learning where agents continually acquire, fine-tune, and transfer skills throughout their lifespan [83]. Early works [44, 99] connected the evolution of weight tolerant networks to the Baldwin effect [2].

To develop a single WANN capable of encoding many different useful tasks in its environment, one might consider developing a WANN with a strong intrinsic bias for intrinsic motivation [82, 84, 94], and continuously optimize its architecture to perform well at pursuing novelty in an open-ended environment [60]. Such a WANN might encode, through a curiosity reward signal, a multitude of skills that can easily be fine-tuned for a particular downstream task in its environment later on.

While our approach learns network architectures of increasing complexity by adding connections, network pruning approaches find new architectures by their removal. It is also possible to learn a pruned network capable of performing additional tasks without learning weights [69]. A concurrent work [118] to ours learns a *supermask* where the sub-network pruned using this mask performs well at image recognition even with randomly initialized weights – it is interesting that their approach achieves a similar range of performance on MNIST compared to ours. While our search method is based on evolution, future work may extend the approach by incorporating recent ideas that formulate architecture search in a differentiable manner [65] to make the search more efficient.

The success of deep learning is attributed to our ability to train the weights of large neural networks that consist of well-designed building blocks on large datasets, using gradient descent. While much progress has been made, there are also limitations, as we are confined to the space of architectures that gradient descent is able to train. For instance, effectively training models that rely on discrete components [31, 47] or utilize adaptive computation mechanisms [30] with gradient-based methods remain a challenging research area. We hope this work will encourage further research that facilitates the discovery of new architectures that not only possess inductive biases for practical domains, but can also be trained with algorithms that may not require gradient computation.

That the networks found in this work do not match the performance of convolutional neural networks is not surprising. It would be an almost embarrassing achievement if they did. For decades CNN architectures have been refined by human scientists and engineers – but it was not the reshuffling of existing structures which originally unlocked the capabilities of CNNs. Convolutional layers were themselves once novel building blocks, building blocks with strong biases toward vision tasks, whose discovery and application have been instrumental in the incredible progress made in deep learning. The computational resources available to the research community have grown significantly since the time convolutional neural networks were discovered. If we are devoting such resources to automated discovery and hope to achieve more than incremental improvements in network architectures, we believe it is also worth experimenting with new building blocks, not just their arrangements.

Acknowledgments

We would like to thank Douglas Eck, Geoffrey Hinton, Anja Austermann, Jeff Dean, Luke Metz, Ben Poole, Jean-Baptiste Mouret, Michiel Adriaan Unico Bacchiani, Heiga Zen, and Alex Lamb for their thoughtful feedback. Experiments in this work were conducted with the support of Google Cloud.

A Supplementary Materials for Weight Agnostic Neural Networks

A.1 Network Diagrams

The networks shows in the body of the text were selected for both performance and readability. In many cases a great deal of complexity is added for only minimal gains in performance, in these cases we preferred to showcase more elegant networks. Below are the diagrams of the champion networks:

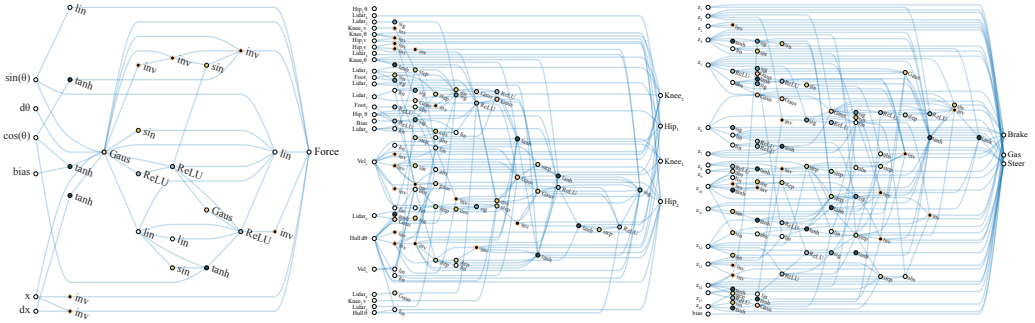

Figure 6: *Best Networks for Continuous Control*
Left to Right (Number of Connections): Swing up (52), Biped (210), Car Racing (245)
Shown in the main text are high performing, but simpler networks, chosen for clarity. This figure illustrates the champion networks whose results are reported in the text.

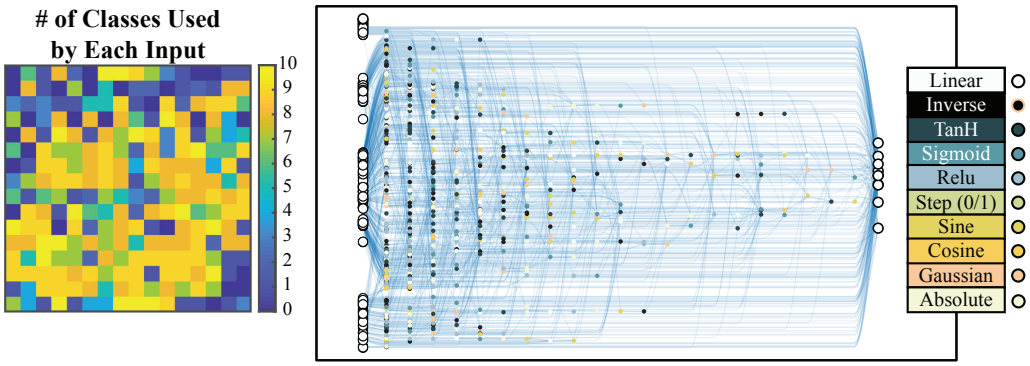

Figure 7: *MNIST classifier network (1849 connections)*
Not all neurons and connections are used to predict each digit. Starting from the output connection for a particular digit, we can trace the sub-network and also identify which part of the input image is used for classifying each digit. Please refer to the supplementary website for more detailed visualizations.

A.2 Code Release

We release a general purpose tool, not only to facilitate reproduction, but also for further research in this direction. Our NumPy [111] implementation of NEAT [103] supports MPI [32] and OpenAI Gym [10] environments. All code used to run these experiments, in addition to the best networks found in each run, is referenced in the interactive article: `https://weightagnostic.github.io/`

A.3 MNIST

The MNIST version used in this paper is a downsampled version, reducing the digits from [28x28] to [16x16], and deskewed using the OpenCV library[7]. The best MNIST network weight was chosen as the network with the highest accuracy on the training set.

To fit into our existing approach MNIST classification is reframed as a reinforcement learning problem. Each sample in MNIST is downsampled to a 16x16 image, deskewed, and pixel intensity normalized between 0 and 1. WANNs are created with input for each of the 256 pixels and one output for each of the 10 digits. At each evaluation networks are fed 1000 samples randomly selected from the training set, and given reward based on the softmax cross entropy. Networks are tested with a variety of shared weight values, maximizing performance over all weights while minimizing the number of connections.

A.4 Hyperparameters and Setup

All experiments but those on Car Racing were performed used 96 core machines on the Google Cloud Platform. As evaluation of the population is embarrassingly parallel, populations were sized as multiples of 96 to make efficient use of all processors. Car Racing was performed on a 64 core machine and the population size used reflects this. The code and setup of the VAE for the Car Racing task is taken from [38], were a VAE with a latent size of 16 was trained following the same procedure as [38]. Tournament sizes were scaled in line with the population size. The number of generations were determined after initial experiments to ensure that a majority of runs would converge.

	SwingUp	Biped	CarRace	MNIST
Population Size	*192*	*480*	*64*	*960*
Generations	*1024*	*2048*	*1024*	*4096*
Change Activation Probability (%)	*50*	*50*	*50*	*50*
Add Node Probability (%)	*25*	*25*	*25*	*25*
Add Connection Probability (%)	*25*	*25*	*25*	*25*
Initial Active Connections (%)	*50*	*25*	*50*	*5*
Tournament Size	*8*	*16*	*8*	*32*

A.5 Results over multiple independent search runs

For each task a WANN search was run 9 times. At regular intervals the network in the population with the best mean performance was compared to that with the previously best found network. If the newer network had a higher mean, the network was evaluated 96 or 64 times (depending on the number of processors on the machine), and if the mean of *those* evaluations was better than the previous best network, it was kept as the new 'best' network. These best networks were kept only for record keeping and did not otherwise interact with the population.

These best networks at the end of each run were reevaluated thirty times on each weight in the series $[-2, -1.5, -1, -0.5, 0.5, 1, 1.5, 2]$ and the network with the best mean chosen as the champion for more intensive analysis and examination. Shown below are the results of these initial tests, both as individual runs and as distributions of performance over weights.

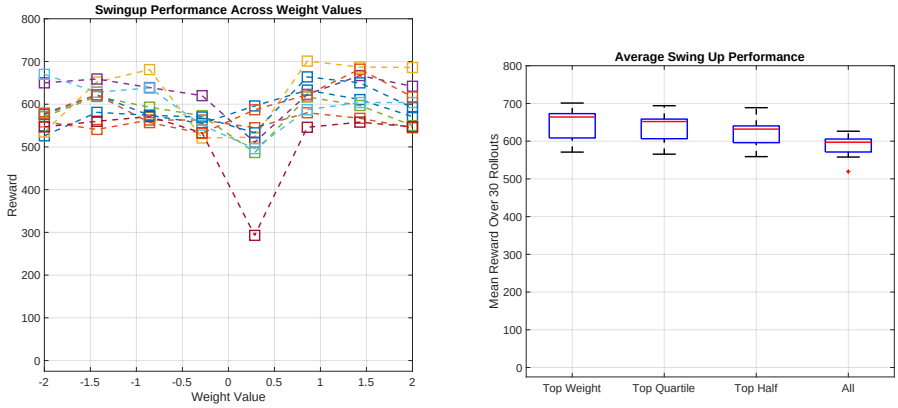

Figure 8: *Swing-up Performance over Multiple Runs.*
Left: Performance per weight value of best network found in each of 9 runs.
Right: Average performance of best networks found at end of each of 9 runs. Performance is shown by top weight, top quartile of weights, top half of weights, and over all weights.

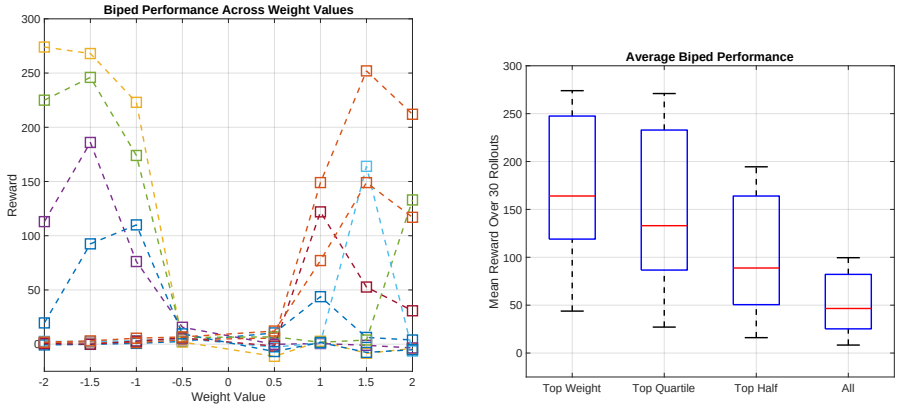

Figure 9: *Biped Performance over Multiple Runs.*
Left: Performance per weight value of best network found in each of 9 runs.
Right: Average performance of best networks found at end of each of 9 runs. Performance is shown by top weight, top quartile of weights, top half of weights, and over all weights.

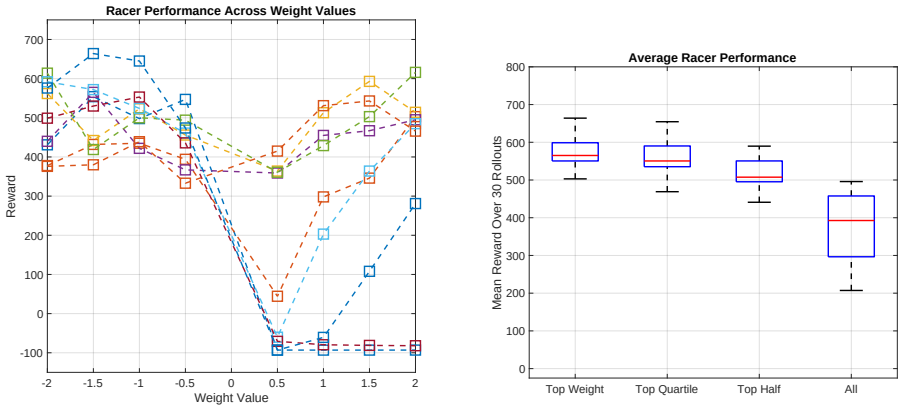

Figure 10: *Car Racing Performance over Multiple Runs.*
Left: Performance per weight value of best network found in each of 9 runs.
Right: Average performance of best networks found at end of each of 9 runs. Performance is shown by top weight, top quartile of weights, top half of weights, and over all weights.

A.6 Optimizing for individual weight parameters

In our experiments, we also fine-tuned individual weight parameters for the champion networks found to measure the performance impact of further training. For this, we used population-based REINFORCE, as in Section 6 of [115]. Our specific approach is based on the open source `estool` [36] implementation of population-based REINFORCE. We use a population size of 384, and each agent performs the task 16 times with different initial random seeds for Swing Up Cartpole and Bipedal Walker. The agent's reward signal used by the policy gradient method is the average reward of the 16 rollouts. For Car Racing, due to the extra computation time required, we instead use a population size of 64 and the average cumulative reward of 4 rollouts to calculate the reward signal. All models trained for 3000 generations. All other parameters are set to the default settings of `estool` [36].

For MNIST, we use the negative of the cross entropy loss as the reward signal, and optimize directly on the training set with population-based REINFORCE. Future work will explore the use of autograd packages such as JAX [23] to fine-tune individual weights of WANNs.

A.7 Fixed Topology Baselines

For Bipedal Walker, we used the model and architecture available from `estool` [36] as our baseline. To our knowledge, this baseline currently, at the time of writing, achieves the state-of-the-art average cumulative score (over 100 random rollouts) on Bipedal Walker as reported in [37].

In the Swing Up Cartpole task, we trained a baseline controller with 1 hidden layer of 10 units (71 weight parameters), using the same training methodology as the one used to produce SOTA results for the Bipedal Walker task mentioned earlier. We experimented with a larger number of nodes in the hidden layer, and an extra hidden layer, but did not see meaningful differences in performance.

For the Car Racing baseline, we used the code and model provided in [38] and treated the 867 parameters of the controller as free weight parameters, while keeping the parameters of the pre-trained VAE and RNN fixed. As of writing, the average cumulative score (over 100 random rollouts) produced by [38] for Car Racing is currently the state-of-the-art. As mentioned in the main text, for simplicity, the WANN controller has access only to the pre-trained VAE, and not to the RNN.

References

[1] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65, 1994.

[2] J. M. Baldwin. A new factor in evolution. *The american naturalist*, 30(354):441–451, 1896.

[3] D. Barber and C. Bishop. Ensemble learning in bayesian neural networks. *NATO ASI series. Series F: computer and system sciences*, pages 215–237, 1998.

[4] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[5] J. E. Black, K. R. Isaacs, B. J. Anderson, A. A. Alcantara, and W. T. Greenough. Learning causes synaptogenesis, whereas motor activity causes angiogenesis, in cerebellar cortex of adult rats. *Proceedings of the National Academy of Sciences*, 87(14):5568–5572, 1990.

[6] L. Blier and Y. Ollivier. The description length of deep learning models. In *Advances in Neural Information Processing Systems*, pages 2216–2226, 2018.

[7] G. Bradski and A. Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[8] H. Braun and J. Weisbrod. Evolving feedforward neural networks. In *Proceedings of ANNGA93, International Conference on Artificial Neural Networks and Genetic Algorithms*, pages 25–32. Springer Berlin, 1993.

[9] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

[10] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[11] J. T. Bruer. Neural connections: Some you use, some you lose. *The Phi Delta Kappan*, 81(4):264–277, 1999.

[12] E. Bullmore and O. Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature reviews neuroscience*, 10(3):186, 2009.

[13] J. Burger. Antipredator behaviour of hatchling snakes: effects of incubation temperature and simulated predators. *Animal Behaviour*, 56(3):547–553, 1998.

[14] F. Chollet et al. Keras, 2015. https://github.com/keras-team/keras/blob/master/examples/.

[15] J. Clune, J.-B. Mouret, and H. Lipson. The evolutionary origins of modularity. *Proceedings of the Royal Society b: Biological sciences*, 280(1755):20122863, 2013.

[16] N. Cohen and A. Shashua. Inductive bias of deep convolutional networks through pooling geometry. *arXiv preprint arXiv:1605.06743*, 2016.

[17] D. Dasgupta and D. R. McGregor. Designing application-specific neural networks using the structured genetic algorithm. In *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 87–96. IEEE, 1992.

[18] E. Dayan and L. G. Cohen. Neuroplasticity subserving motor skill learning. *Neuron*, 72(3):443–454, 2011.

[19] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[20] K. Eichler, F. Li, A. Litwin-Kumar, Y. Park, I. Andrade, C. M. Schneider-Mizell, T. Saumweber, A. Huser, C. Eschbach, B. Gerber, et al. The complete connectome of a learning and memory centre in an insect brain. *Nature*, 548(7666):175, 2017.

[21] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[22] J. Frankle and M. Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[23] R. Frostig, M. J. Johnson, and C. Leary. Compiling machine learning programs via high-level tracing, 2018.

[24] K. Fukushima and S. Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern recognition*, 15(6):455–469, 1982.

[25] B. Fullmer and R. Miikkulainen. Using marker-based genetic encoding of neural networks to evolve finite-state behaviour. In *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 255–262. MIT Press, 1992.

[26] Y. Gal. *Uncertainty in deep learning*. PhD thesis, PhD thesis, University of Cambridge, 2016.

[27] Y. Gal, R. McAllister, and C. E. Rasmussen. Improving pilco with bayesian neural network dynamics models. In *Data-Efficient Machine Learning workshop, ICML*, volume 4, 2016.

[28] A. Goth. Innate predator-recognition in australian brush-turkey (alectura lathami, megapodiidae) hatchlings. *Behaviour*, 138(1):117, 2001.

[29] A. Graves. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pages 2348–2356, 2011.

[30] A. Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[31] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[32] W. D. Gropp, W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: portable parallel programming with the message-passing interface*, volume 1. MIT press, 1999.

[33] F. Gruau, D. Whitley, and L. Pyeatt. A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the 1st annual conference on genetic programming*, pages 81–89. MIT Press, 1996.

[34] P. D. Grünwald. *The minimum description length principle*. MIT press, 2007.

[35] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[36] D. Ha. Evolving stable strategies. *http://blog.otoro.net/*, 2017. http://blog.otoro.net/2017/11/12/evolving-stable-strategies/.

[37] D. Ha. Reinforcement learning for improving agent design. *arXiv:1810.03779*, 2018. https://designrl.github.io.

[38] D. Ha and J. Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pages 2451–2463. Curran Associates, Inc., 2018. https://worldmodels.github.io.

[39] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[40] S. A. Harp, T. Samad, and A. Guha. Designing application-specific neural networks using the genetic algorithm. In *Advances in neural information processing systems*, pages 447–454, 1990.

[41] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[42] Y. He and A. Evans. Graph theoretical modeling of brain connectivity. *Current opinion in neurology*, 23(4):341–350, 2010.

[43] G. Hinton and D. Van Camp. Keeping neural networks simple by minimizing the description length of the weights. In *in Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory*. Citeseer, 1993.

[44] G. E. Hinton and S. J. Nowlan. How learning can guide evolution. *Adaptive individuals in evolving populations: models and algorithms*, 26:447–454, 1996.

[45] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[46] P. R. Huttenlocher. Morphometric study of human cerebral cortex development. *Neuropsychologia*, 28(6):517–527, 1990.

[47] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[48] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.

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

[50] J. A. Kleim, S. Barbay, N. R. Cooper, T. M. Hogg, C. N. Reidel, M. S. Remple, and R. J. Nudo. Motor learning-dependent synaptogenesis is localized to functionally reorganized motor cortex. *Neurobiology of learning and memory*, 77(1):63–77, 2002.

[51] A. N. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of information transmission*, 1(1):1–7, 1965.

[52] A. Konak, D. W. Coit, and A. E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, 2006.

[53] R. Krishnan and V. B. Ciesielski. Delta-gann: A new approach to training neural networks using genetic algorithms. In *University of Queensland*. Citeseer, 1994.

[54] D. Krueger, C.-W. Huang, R. Islam, R. Turner, A. Lacoste, and A. Courville. Bayesian hypernetworks. *arXiv preprint arXiv:1710.04759*, 2017.

[55] Y. LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[56] Y. LeCun, Y. Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[57] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

[58] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[59] C.-H. Lee and J.-H. Kim. Evolutionary ordered neural network with a linked-list encoding scheme. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 665–669. IEEE, 1996.

[60] J. Lehman and K. O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *ALIFE*, pages 329–336, 2008.

[61] C. Li, H. Farkhoor, R. Liu, and J. Yosinski. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*, 2018.

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

[63] L. Li and A. Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

[64] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*, 2017.

[65] H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[66] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[67] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017.

[68] D. J. MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.

[69] A. Mallya, D. Davis, and S. Lazebnik. Piggyback: Adapting a single network to multiple tasks by learning to mask weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 67–82, 2018.

[70] M. Mandischer. Representation and evolution of neural networks. In *Artificial Neural Nets and Genetic Algorithms*, pages 643–649. Springer, 1993.

[71] V. Maniezzo. Genetic evolution of the topology and weight distribution of neural networks. *IEEE Transactions on neural networks*, 5(1):39–53, 1994.

[72] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[73] D. B. Miles, L. A. Fitzgerald, and H. L. Snell. Morphological correlates of locomotor performance in hatchling amblyrhynchus cristatus. *Oecologia*, 103(2):261–264, 1995.

[74] B. L. Miller, D. E. Goldberg, et al. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems*, 9(3):193–212, 1995.

[75] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[76] A. Mori and G. M. Burghardt. Does prey matter? geographic variation in antipredator responses of hatchlings of a japanese natricine snake (rhabdophis tigrinus). *Journal of Comparative Psychology*, 114(4):408, 2000.

[77] J.-B. Mouret. Novelty-based multiobjectivization. In *New horizons in evolutionary robotics*, pages 139–154. Springer, 2011.

[78] R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[79] K. Neklyudov, D. Molchanov, A. Ashukha, and D. Vetrov. Variance networks: When expectation does not meet your expectations. *arXiv preprint arXiv:1803.03764*, 2018.

[80] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural computation*, 4(4):473–493, 1992.

[81] D. W. Opitz and J. W. Shavlik. Connectionist theory refinement: Genetically searching the space of network topologies. *Journal of Artificial Intelligence Research*, 6:177–209, 1997.

[82] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286, 2007.

[83] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter. Continual lifelong learning with neural networks: A review. *arXiv preprint arXiv:1802.07569*, 2018.

[84] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17, 2017.

[85] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*, pages 4092–4101, 2018.

[86] J. C. F. Pujol and R. Poli. Evolving the topology and the weights of neural networks using a dual representation. *Applied Intelligence*, 8(1):73–84, 1998.

[87] T. Raiko and M. Tornio. Variational bayesian learning of nonlinear hidden state-space models for model predictive control. *Neurocomputing*, 2009.

[88] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.

[89] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2902–2911. JMLR. org, 2017.

[90] D. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *Preprint arXiv:1401.4082*, 2014.

[91] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[92] J. Rissanen. *Information and complexity in statistical modeling*. Springer Science & Business Media, 2007.

[93] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[94] J. Schmidhuber. Curious model-building control systems. In *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, pages 1458–1463. IEEE, 1991.

[95] J. Schmidhuber. Discovering neural nets with low kolmogorov complexity and high generalization capability. *Neural Networks*, 10(5):857–873, 1997.

[96] J. Schmidhuber, D. Wierstra, M. Gagliolo, and F. Gomez. Training recurrent networks by evolino. *Neural computation*, 19(3):757–779, 2007.

[97] C. Sciuto, K. Yu, M. Jaggi, C. Musat, and M. Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.

[98] S. Seung. *Connectome: How the brain's wiring makes us who we are*. HMH, 2012.

[99] J. M. Smith. When learning guides evolution. *Nature*, 329(6142):761, 1987.

[100] D. R. So, C. Liang, and Q. V. Le. The evolved transformer. *arXiv preprint arXiv:1901.11117*, 2019.

[101] R. J. Solomonoff. A formal theory of inductive inference. part i. *Information and control*, 7(1):1–22, 1964.

[102] O. Sporns, G. Tononi, and R. Kötter. The human connectome: a structural description of the human brain. *PLoS computational biology*, 1(4):e42, 2005.

[103] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[104] J. M. Starck and R. E. Ricklefs. Patterns of development: the altricial-precocial spectrum. *Oxford Ornithology Series*, 8:3–30, 1998.

[105] S.-y. Takemura, Y. Aso, T. Hige, A. Wong, Z. Lu, C. S. Xu, P. K. Rivlin, H. Hess, T. Zhao, T. Parag, et al. A connectome of a learning and memory center in the adult drosophila brain. *Elife*, 6:e26975, 2017.

[106] R. Tedrake. Underactuated robotics: Learning, planning, and control for efficient and agile machines: Course notes for mit 6.832. *Working draft edition*, 3, 2009.

[107] A. L. Tierney and C. A. Nelson III. Brain development and the role of experience in the early years. *Zero to three*, 30(2):9, 2009.

[108] A. Trask, F. Hill, S. E. Reed, J. Rae, C. Dyer, and P. Blunsom. Neural arithmetic logic units. In *Advances in Neural Information Processing Systems*, pages 8035–8044, 2018.

[109] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9446–9454, 2018.

[110] M. P. Van Den Heuvel and O. Sporns. Rich-club organization of the human connectome. *Journal of Neuroscience*, 31(44):15775–15786, 2011.

[111] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[112] L. R. Varshney, B. L. Chen, E. Paniagua, D. H. Hall, and D. B. Chklovskii. Structural properties of the caenorhabditis elegans neuronal network. *PLoS computational biology*, 7(2):e1001066, 2011.

[113] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[114] J. G. White, E. Southgate, J. N. Thomson, and S. Brenner. The structure of the nervous system of the nematode caenorhabditis elegans. *Philos Trans R Soc Lond B Biol Sci*, 314(1165):1–340, 1986.

[115] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[116] X. Yao and Y. Liu. Towards designing artificial neural networks by evolution. *Applied Mathematics and Computation*, 91(1):83–90, 1998.

[117] B.-T. Zhang and H. Muhlenbein. Evolving optimal neural networks using genetic algorithms with occam's razor. *Complex systems*, 7(3):199–220, 1993.

[118] H. Zhou, J. Lan, R. Liu, and J. Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. *arXiv preprint arXiv:1905.01067*, 2019.

[119] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[120] X. Zuo. PyTorch implementation of Improving PILCO with Bayesian neural network dynamics models, 2018. https://github.com/zuoxingdong/DeepPILCO.