2019

Training Neural Networks Through the Integration of Evolution and Gradient Descent

Gregory Morse
University of Central Florida

TRAINING NEURAL NETWORKS THROUGH THE INTEGRATION OF EVOLUTION AND GRADIENT DESCENT

by

GREGORY MORSE
B.S. Wichita State University, 2007
M.S. University of Central Florida, 2016

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2019

Major Professor: Kenneth O. Stanley

ABSTRACT

Neural networks have achieved widespread adoption due to both their applicability to a wide range of problems and their success relative to other machine learning algorithms. The training of neural networks is achieved through any of several paradigms, most prominently gradient-based approaches (including deep learning), but also through up-and-coming approaches like neuroevolution. However, while both of these neural network training paradigms have seen major improvements over the past decade, little work has been invested in developing algorithms that incorporate the advances from both deep learning and neuroevolution. This dissertation introduces two new algorithms that are steps towards the integration of gradient descent and neuroevolution for training neural networks. The first is (1) the Limited Evaluation Evolutionary Algorithm (LEEA), which implements a novel form of evolution where individuals are partially evaluated, allowing rapid learning and enabling the evolutionary algorithm to behave more like gradient descent. This conception provides a critical stepping stone to future algorithms that more tightly couple evolutionary and gradient descent components. The second major algorithm (2) is Divergent Discriminative Feature Accumulation (DDFA), which combines a neuroevolution phase, where features are collected in an unsupervised manner, with a gradient descent phase for fine tuning of the neural network weights. The neuroevolution phase of DDFA utilizes an indirect encoding and novelty search, which are sophisticated neuroevolution components rarely incorporated into gradient descent-based systems. Further contributions of this work that build on DDFA include (3) an empirical analysis to identify an effective distance function for novelty search in high dimensions and (4) the extension of DDFA for the purpose of discovering convolutional features. The results of these DDFA experiments together show that DDFA discovers features that are effective as a starting point for gradient descent, with significant improvement over gradient descent alone. Additionally, the method of collecting features in an unsupervised manner allows DDFA to be ap-

plied to domains with abundant unlabeled data and relatively sparse labeled data. This ability is highlighted in the STL-10 domain, where DDFA is shown to make effective use of unlabeled data.

This work is dedicated to Evelyn. May the future hold an endless bounty of wonder as you explore the mysteries of the universe.

ACKNOWLEDGMENTS

I would first like to offer my heartfelt appreciation to my advisor, Dr. Kenneth O. Stanley, who has provided a model of what a scientific researcher should be. From the generation and development of ideas to the application thereof, and the effective explanation both through writing and public presentation, Dr. Stanley has offered guidance at every step. Along with his enthusiasm for discovery, his neverending drive to improve an idea and promulgate it effectively played a critical role in the creation of this dissertation.

I would also like to give special thanks to my mother and father, who encouraged my interest in both computers and science from a young age. I would also like to share my appreciation for my wife, Jennifer, whose love and support were crucial to the completion of this work. And thank you to Bill and Marlene Hyde, whose encouragement led me to pursuing my graduate degree at UCF.

I am also grateful to my other committee members, Dr. Mubarak Shah, Dr. R. Paul Wiegand, and Dr. Annie S. Wu, whose feedback has helped improve the quality of this dissertation.

I would like to extend my gratitude to all of the members of the EPlex lab that I had the pleasure of working with throughout the years. From the many discussions and brainstorming sessions we had to the collaborative projects we worked on, my peers were instrumental in my personal and professional development.

Finally, I would like to thank all those who support and maintain the Stokes cluster at the UCF Advanced Research Computing Center. Without the computational resources of Stokes, this dissertation would not have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

The past decade has seen an explosion in the application of neural networks to a range of artificial intelligence domains [19, 57, 75, 101, 104]. This widespread adoption can be credited primarily to the rise of deep learning, which allows neural networks of many layers to be trained effectively. This deep architecture allows simple features discovered in the lower layers to be composed into more complex features, leading to a hierarchy of increasingly complex features. This hierarchy is thought to be at the core of what makes deep neural networks so powerful. Because of its success across a wide range of domains, deep learning has come to dominate neural network research in recent years. However, despite its success, deep learning depends upon some form of gradient descent, which makes it vulnerable to theoretical pitfalls such as local optima and empirically observed anomalies such as fooling [83]. Furthermore, the application of deep learning is greatly constrained by its requirement of abundant labeled training data. For example, ImageNet [26], which is the most commonly used image processing domain in deep learning, contains over a million labeled images for training. However, few real world problems contain such an abundance of labeled data, and generating labeled data may be cost prohibitive.

Other neural network training paradigms have seen advances in recent years. Among these is neuroevolution, in which neural networks are trained with an algorithm that borrows from the process of biological evolution. Neuroevolution has seen several important advances in the past decade [63, 90, 111], including competitive performance to deep learning in some reinforcement learning (RL) domains [17, 87]. More importantly, because it does not require an abundance of labeled training data, or even any labeled data at all, neuroevolution is broadly applicable to almost any problem, making it popular on domains where it is difficult to apply an algorithm with the limitations that come with deep learning [17, 19, 65, 95].

While deep learning and neuroevolution are most often viewed as competing neural network training paradigms that are perhaps each suited to only certain problems, there is a small but growing array of hybrid algorithms that combine aspects of both paradigms to exploit the strengths of each [5, 12, 53, 67, 72, 92, 94]. However, these hybrid algorithms typically adopt a very simple evolutionary algorithm, ignoring the major advances in neuroevolution over the past decade. These hybrids also tend to treat evolution and gradient descent as distinctly separate processes, with little interaction between the two paradigms. The primary goal of this dissertation is to provide an example of how the more sophisticated developments in neuroevolution can be combined with gradient descent to form an algorithm that exploits the strengths of both paradigms. Secondarily, this work provides a new algorithm that allows neuroevolution to behave more like gradient descent, permitting the two paradigms to be combined in a more interactive manner.

In its first contribution, this dissertation presents a new variation of the traditional evolutionary algorithm (EA), named Limited Evaluation Evolutionary Algorithm (LEEA), that allows EAs to behave more like their gradient descent cousins. The key insight introduced by LEEA is that individuals in the population do not need to be evaluated against the entire training set to determine their fitness. Instead, they can be evaluated against a small subset of the training set (as few as two examples) and that information, along with fitness information from their ancestors, can provide a useful estimate of their fitness. This reduction greatly speeds up training, particularly on domains with a large amount of training data. More importantly, the process of limited evaluation mirrors an aspect of stochastic gradient descent called mini-batch training, where a network is trained against a small subset of the training set at a time. By providing an analogous method of training in EAs, LEEA provides the ability to combine EAs with gradient descent in a more tightly coupled relationship.

The second and most extensive contribution of this dissertation is the development of a new kind of hybrid algorithm, named Divergent Discriminative Feature Accumulation (DDFA), that combines

the strengths of gradient descent and two of the more powerful tools available in neuroevolution: indirect encoding [111] and novelty search [63]. DDFA contains an unsupervised feature collection phase, in which candidate features are encoded by Compositional Pattern Producing Networks (CPPNs) [38], which are a type of indirect neural network encoding. CPPNs tend to produce patterns with regularities such as symmetry and repetition with variation, which can be useful when looking for image features. The CPPN-generated candidate features are evolved with novelty search providing the evolutionary pressure, which allows DDFA to discover a set of features that discriminate across the training set in diverse ways. Once enough novel features are discovered, they are composed into a fully-connected neural network and trained in a more traditional way with stochastic gradient descent (SGD).

The third major contribution is an analysis of the distance function in DDFA, which is a critical component of the novelty search algorithm that drives the evolution and discovery of new features. While the traditional choice for comparing individuals in novelty search is Euclidean distance, the analysis performed here reveals other distance functions to be more suited to this task, due in large part to the high dimensionality of the behavior vectors that are produced by the feature evaluation process in DDFA. The implications of this investigation extend beyond DDFA back to novelty search in general, where these insights can potentially improve the application of novelty search across a diverse range of applications.

The fourth and final contribution of this dissertation is the application of DDFA to the discovery of convolutional features, unlocking the potential to apply DDFA to some of the most advanced architectures in use today. This work applies DDFA first to the discovery of convolutional features on the MNIST handwritten digit recognition domain. Although MNIST is not a cutting-edge visual recognition domain, it acts as a launching pad for the initial development of DDFA, which requires significant computational resources to tune hyperparameters of the system. To then demonstrate its potential, DDFA is applied to the STL-10 domain [16], which is a subset of ImageNet with a

limited amount of labeled training data and a larger sample of unlabeled data. This domain was designed for testing semi-supervised algorithms such as DDFA, and this contribution finally shows that DDFA can successfully exploit the unlabeled data to generate convolutional features that provide a helpful starting point for gradient descent. This discovery implies that not only can DDFA provide a potential benefit to gradient descent on domains where gradient descent already does well, but DDFA can also allow gradient descent to be applied to new domains where it currently struggles due to a limitation on available labeled training data. Furthermore, an experiment on this domain reveals that DDFA can also provide better exploitation of a wider network than gradient descent alone can efficiently train, providing another avenue of opportunity for the training of wide networks.

DDFA is an important contribution for several reasons. First, the feature discovery technique provides a performance improvement over random initialization even when all of the training data is labeled, as is shown by experiments on the MNIST domain. Furthermore, the non-convergent nature of novelty search may assist gradient descent in avoiding local optima. Finally, the unsupervised nature of feature discovery allows it take advantage of unlabeled data, which is often far more abundant than labeled data. This capability, as is shown in multiple experiments on the STL-10 domain, potentially can assist in applying gradient descent to to many problems where an abundant amount of labeled data is unavailable.

Taken as a whole, the contributions in this dissertation are focused on expanding the possibilities for combining neuroevolution and gradient descent for training neural networks. While LEEA provides an important stepping stone towards the development of such hybrid algorithms, DDFA provides an example of an algorithm that combines some of the more sophisticated neuroevolution tools with gradient descent, exploiting the strengths of both paradigms. This example could open the door to many new hybrid algorithms that combine the best aspects of these two popular neural network training paradigms.

The dissertation continues in Chapter 2 with background into gradient descent and neuroevolution and a review of previous attempts to combine the two training paradigms. Chapter 3 introduces the LEEA algorithm and Chapter 4 introduces the DDFA algorithm, both including experimental results. From there, Chapter 5 contains a discussion and experiments related to the distance function used by the novelty search component of DDFA and Chapter 6 details the approach and experiments related to learning convolutional features within DDFA. The implications of the work as a whole are discussed in Chapter 7, followed by the conclusion in Chapter 8.

CHAPTER 2: BACKGROUND

This chapter reviews foundational material to the LEEA and DDFA algorithms introduced in later chapters. First is a brief introduction to neural networks, followed by an introduction of convolutional neural networks. Next is a review of the neural network training paradigms contained within DDFA: gradient descent and neuroevolution. Finally, the chapter ends with a review of the literature on hybrid algorithms that combine gradient descent with neuroevolution.

Artificial Neural Networks

Artificial neural networks (ANNs) are computational models inspired by the biological neural networks found in animals, including humans. ANNs are composed of a layer of input neurons, one or more layers of hidden neurons, and a layer of output neurons, where each layer is typically connected to adjacent layers. Each neuron is loosely modeled upon the activity of biological neurons, and information in feedforward networks flows from the input layer to the output layer. Due to the ability of any ANN with at least one hidden layer to model nonlinear processes, ANNs have been successfully applied to a wide array of problems, including visual recognition [57], speech recognition [101], robotic control [19], board games [104], and video games [75].

Many algorithms have been developed for training ANNs; early work focused on biologically plausible learning methods such as Hebbian learning [49], which is based on neural plasticity. However, Hebbian learning had limited success and applicability and was quickly supplanted by less biologically plausible methods such as perceptron learning [96]. Since then several other learning algorithms have been applied to ANNs, such as evolutionary algorithms [4, 24], expectation-maximization [82], particle swarm optimization [29], and simulated annealing [14], but the basic

method of gradient descent [98], which the perceptron learning rule is based upon, has dominated the field of ANNs for decades. In that time, gradient decent has grown from a simple algorithm for training perceptrons to the algorithm of choice for training deep neural networks in the field of deep learning.

Gradient Descent

Gradient descent (GD) is an optimization algorithm for finding the minimum of a function by moving in the negative direction of the gradient of the error at each step. The perceptron learning rule was the first implementation of this approach for optimizing simple ANNs and is defined by the equation

$$w_i(t+1) = w_i(t) + (d_j - y_j(t))x_{j,i}, \tag{2.1}$$

for all weights, where w_i is the i-th weight, d_j is the desired output of the j-th training example, $y_j(t)$ is the actual output of the perceptron, and $x_{j,i}$ is the i-th input value for the j-th training example. While the perceptron learning rule was an important step forward for ANN learning algorithms, it came with significant limitations. While Minsky and Papert [74] showed that perceptrons are guaranteed to converge when the classes are linearly separable, the absence of a hidden layer hinders perceptrons from converging when the classes are not linearly separable, such as with the XOR problem. Without a better alternative for training ANNs, research into ANN training stagnated for over a decade.

The revival of ANN research in the 1980s was due in large part to a new GD-based technique called backpropagation (BP) [59, 86, 97, 118]. Put simply, BP is the method of propagating the error backwards through a neural network, permitting the use of gradient descent across any arbitrary

number of layers. With at least one hidden layer, neural networks are able to compute functions that are not linearly separable, such as XOR. In fact, it has been shown that a neural network with just a single hidden layer is sufficient to approximate any continuous function [20]. BP was described generally and theoretically in many forms throughout the 1970s, but the first ANN-specific application of BP was described by Werbos [118]. BP was extended and popularized by further work in the mid 1980s by Parker [86], LeCun [59], and Rumelhart et al. [97].

Over the decades since its inception, BP saw increasing success that largely followed the trajectory of increasing computational power, which is particularly helpful for deep ANNs, and increased access to labeled training data. Various algorithmic improvements have also expanded the power of BP's ability to train ANNs. The most commonly used form of gradient descent through backpropagation is called stochastic gradient descent (SGD), in which the ANN is evaluated and the weights updated after only a small number of training examples. SGD enables even large networks to be trained relatively quickly, and with increases in computational power, very large networks can now be trained. Because of this, the training of ANNs with many hidden layers, known as *deep learning*, has seen an enormous rise in popularity and success [11, 50, 58, 71]. A key unifying principle in deep learning is that an ANN with multiple hidden layers can encode increasingly complex features across its layers. Other substantial improvements in deep learning include convolution, which is explained in more detail in the next section, and long short-term memory units (LSTMs), which are recurrent network structures that give ANNs a type of memory [51]. These advances in ANN training, which still rely on GD to learn the network weights, have led to substantive records being broken in many areas of machine learning [15, 47, 58, 104], including unsupervised feature learning [8].

The success of a principle as simple as SGD in achieving record-breaking performance is perhaps surprising. After all, in a space of many dimensions, SGD should be susceptible to local optima, and unlike an evolutionary algorithm, all its eggs are essentially in a single basket because it

works in effect with a population of one. Yet it turns out empirically that SGD is penetrating farther towards optimality in networks of thousands or millions of weights than any other approach. Attempting in part to explain this phenomenon, Dauphin et al. [23] make the intriguing argument that in fact very high-dimensional ANN weight spaces provide so many possible escape routes from any given point that local optima are actually highly unlikely. Instead, they suggest that the real stumbling blocks for SGD are *saddle points*, or areas of long, gradual error plateaus. This insight has both helped to explain why SGD might not get stuck, and also to improve it to move faster along such saddle points in the search space. There are also variants of SGD such as RMSProp [116] that help to extricate it from similar situations.

In fact, the smooth application of SGD in deep learning remains a work in progress. Many tricks have been developed to improve its performance, such as the introduction of rectified linear units (ReLUs) for activation functions, which improves the passing of the gradient from layer to layer over sigmoidal units [81]. Yet even then, researchers continue to observe challenges with finding the right parameters to make such structures learn smoothly, leading to complicated work-arounds like interpolating between different architectures over the course of learning [2] and the *highway networks* architecture of Srivastava et al. [107] that in effect turns some neurons on and off over the course of learning, which is reminiscent of evolutionary algorithms that learn structure like NEAT [108]. Thus there remains ample room for new approaches.

Convolutional Neural Networks

Convolutional neural networks (CNNs) are a special type of neural network that enforce a type of local connectivity to achieve translation invariance. Translation invariance allows CNNs to detect the presence of a feature regardless of where it shows up in the input space and without duplication of network structure. CNNs draw inspiration from biology, where it was observed by Hubel and

Wiesel [55] that the visual cortex contains what are described as *simple cells*, which respond to edges of particular orientations in a specific location of the field of vision, and *complex cells*, which respond to edges of a particular orientation regardless of where they show up in the field of vision. Inspired by this work, Fukushima [37] developed a model named *Neocognition*, which is regarded as the first implementation of the principles of translation invariance within neural networks. It was over a decade after this first implementation before the increasing computational power of computers and the development of backpropagation allowed CNNs to develop into their current form [60, 62].

Like a more traditional fully connected neural network, CNNs consist of an input layer, one or more hidden layers, and an output layer. However, the first several hidden layers of a CNN will typically be a special type of layer called a convolutional layer where rather than being connected to all neurons of the previous layer, the neurons are connected in a special, locality-specific manner to the previous layer. A convolutional layer consists of a set of filters or kernels that have a relatively small receptive field. Each feature is evaluated against the input and a shift in location produces multiple outputs that are composed into what is often called a *feature map*. This feature map, which provides location information for each feature, then acts as the input for the next layer in the neural network. The process of shifting, or convolving, the filters across the input data can take place in multiple dimensions, which allows this process to be applied to domains such as image recognition, where data is typically two dimensional. This process of convolving filters across the input data is what allows convolutional layers to detect features regardless of where they occur in the input. In addition, the process of convolution, with its creation of feature maps, exploits geometric information within the data, which is typically lost in a fully-connected network.

CNNs have seen a rise in popularity, with nearly ubiquitous use on visual recognition tasks and widespread use in other areas such as natural language processing [126], time series prediction [30], and board games [103]. However, while convolution has become a critical component of

deep learning, convolution is only an architectural change and not tied to the learning algorithm behind deep learning, gradient descent. The door thus remains open for other neural network training paradigms to exploit the power of convolution.

Evolutionary Algorithms and Neuroevolution

Evolutionary algorithms (EAs) are a broad class of nature-inspired algorithms based upon Darwin's theory of evolution. While in nature populations of organisms adapt to their environment through survival of the fittest and genetic inheritance with recombination and mutation, EAs apply operations inspired by those in nature to populations of candidate solutions to a given problem. For example, the concept of survival of the fittest in biological evolution is implemented by a fitness function in an EA. Each candidate solution is evaluated on how well it solves the problem, yielding its fitness, and its likelihood of being propagated into the future through reproduction is linked to this fitness. In this way, less desirable solutions are discarded while the EA is free to explore areas nearby the most desirable previously discovered solutions. This form of search space exploration differs significantly from gradient descent. While gradient descent starts in a single location in the search space and climbs to the nearest peak, EAs simultaneously explore multiple parts of the search space, making them potentially less susceptible to deception of local optima.

The way in which candidates encode their solution to the problem is known as the problem representation. While simple representations such as integer or binary strings are available, EAs may also be applied to more complex representations such as ANNs. The application of EAs to optimizing ANNs is called *neuroevolution* (NE) by its practitioners [33, 108]. While EAs predate the advent of backpropagation by well over a decade [35, 52], the field of neuroevolution only began in the 1980s (e.g. Montana and Davis [76]), at a time when backpropagation was already on the rise [98]. Interest in neuroevolution picked up in the 1990s, during which a wide variety of approaches

were introduced [124]. In these early years, many researchers focused on the intriguing idea that evolution might in fact exceed the capabilities of backpropagation.

In fact, a number of early studies showcase neuroevolution through a variety of EAs matching [120] or even outperforming backpropagation in classification problems [40, 76, 89]. In these early years enthusiasm was high in part because the future potential of evolving topology along with connection weights seemed to provide a potential advantage for neuroevolution. As Mühlenbein [80] put it, "We conjecture that the next generation of neural networks will be genetic neural networks which evolve their structure." Many researchers echoed this enthusiasm [13, 21]. Others touted the potential for combining topology evolution with backpropagation [6, 119].

However, as computational capabilities increased over the ensuing decade, researchers began to recognize that the apparent advantages of neuroevolution on relatively simple, low-dimensional problems (i.e. requiring relatively small networks) with small amounts of data might be eclipsed as the amount of data and size the ANNs increases. For example, even before deep learning really began to showcase the power of gradient descent with big data, Mandischer [70] begins to articulate the more negative outlook (focusing on Evolution Strategies, or ESs, which are a kind of EA): "We will see that ESs can only compete with gradient-based search in the case of small problems and that ESs are good for training neural networks with a non-differentiable activation function."

As gradient descent techniques increasingly dominated the world of classification, especially after the advent of deep learning [10, 50], a significant shift in attention away from classification took hold in the neuroevolution literature. Researchers began to focus on types of problems where gradient descent is more challenging to apply, such as reinforcement learning problems requiring recurrent connections and specialized architectures [3, 34, 43, 77]. As the focus in neuroevolution largely shifted towards reinforcement learning and away from classification, a new generation of neuroevolution algorithms such as NEAT [108, 110], HyperNEAT [39, 111], and a modified

CMA-ES [56] gained popularity in part by focusing on the daunting challenge of finding the right architecture for complicated control and decision-making problems, for which gradient-based approaches provide less convenient options. Thus the aspiration of EAs to compete directly with SGD in training ANNs for state-of-the-art classification and supervised learning gradually became only a memory.

Nevertheless, neuroevolution enjoys several advantages over gradient descent that make it both applicable to a broader range of problems and less susceptible to the classic gradient descent pitfalls of local optima and saddle points [23]. For instance, the fitness function contained within EAs is inherently more expressive than the error function in gradient descent. This difference allows neuroevolution to be applied easily to problems where a gradient is difficult or impossible to compute, such as in unsupervised and reinforcement learning domains. Recent work from OpenAI has shown that evolution can perform just as well as the more standard gradient-based approaches at training networks of high dimensionality on several reinforcement learning benchmarks [100]. Furthermore, the flexibility provided by the fitness function allows EAs to be applied to problems with multiple objectives [25].

Another seemingly small but revolutionary modification to the fitness function was the discovery of novelty search [63], which rewards individuals for novel behavior rather than raw ability at a given task. This subtle change to the selection process transforms an EA from a convergent process into a divergent one. Put simply, novelty search encourages behavioral diversity within the population with little regard for the final goal, which frequently leads to better solutions than through the search for raw performance with a traditional fitness function in a variety of domains [42, 64, 78]. In addition to improving performance on traditional objective goals, novelty search has also been instrumental in the development of the new field of *quality diversity* [90], which focuses on producing a behaviorally diverse set of high-quality solutions in a single run.

Other advantages that neuroevolution enjoys over gradient descent include topological evolution such as in NEAT, indirect encodings such as HyperNEAT, which encourage the discovery of repetitive neural structures, and genetic diversity techniques, which promote exploration of the search space and prevent premature convergence. These approaches provide capabilities that are unavailable to GD-based approaches.

Hybrid Algorithms

Many neural network researchers, having recognized that both gradient descent and neuroevolution provide their own benefits, have attempted to create algorithms that exploit both paradigms. While neuroevolution can provide capabilities that are necessary to overcome obstacles within a particular domain, the primary drive behind attempts to hybridize neuroevolution and gradient descent comes from the hope of improving performance at a task that gradient descent already performs well on.

One of the more common approaches to hybridization involves exploiting evolution for topological discovery while gradient descent adjusts the weights [5, 6, 27, 72, 73, 91, 102, 119, 122]. While this approach to hybridization has seen some impressive results [93], this approach comes with a high computational cost involved in performing gradient descent across all members of a population. Because of this computational bottleneck, few of these approaches extend beyond a few dozen generations, which is too few for evolution to fully explore the topology space. However, all of these approaches may yet benefit from the increased parallelism in modern compute clusters.

Another common approach to hybridization is to execute gradient descent and neuroevolution in separate phases such that the two have very little interaction. In the more common variant of this approach, neuroevolution is performed first and the champion is then fine-tuned with gradient

descent [85, 94, 115]. Less commonly, multiple runs of gradient descent are performed to produce a population of individuals that act as a starting point for neuroevolution [12, 69].

Other novel hybrid algorithms include recent work from DeepMind that performs learning with gradient descent and employs an EA to determine the best parts of the network to reuse for transfer learning [32] and a differentiable type of indirect encoding where the topology is evolved while the weights are discovered through gradient descent [31].

While many researchers have recognized that neuroevolution can provide important tools to augment gradient descent, the full set of tools developed by the neuroevolution community over the past two decades has not been fully exploited. In short, the vast majority of hybridization work has been focused on only two relatively simple methods of combining these potentially complex algorithms. More importantly, there has been little to no research in exploiting recent advances in neuroevolution such as indirect encodings and novelty search for augmenting gradient descent. The work in this dissertation aims to fill some of these gaps by combining these recent advances in neuroevolution with gradient descent to more fully exploit the power of both paradigms.

CHAPTER 3: NEW APPROACH: LEEA

Hybridizations between evolution and SGD naturally raise the question of whether evolution can mimic some aspects of SGD under the right conditions as well. If the algorithmic gap between evolution and SGD could be bridged, it could open new possibilities for hybridizing these two neural network training paradigms. This chapter explores this question with an algorithm that obtains parity with SGD on lower-dimensional networks.

Limited Evaluation Evolutionary Algorithm

Of all gradient descent algorithms, stochastic gradient descent (SGD) is the most successful, powering the last decade of advances in deep learning. A key property of SGD is that the gradient does not have to be calculated for the network over the entire training set. Instead, the gradient can be calculated for a single or very few training examples at a time, which greatly reduces computational cost compared to training on all training examples at once and reduces the chance of becoming stuck in local optima. Interestingly, this ability to adjust weights after very few examples is not necessarily exclusive to gradient descent. Rather, it can in principle apply to any algorithm that traditionally evaluates its solutions against an entire training set, such as EAs. With this in mind, Ken Stanley and I created the limited evaluation evolutionary algorithm, or LEEA, which runs evolution against a very few number of training examples in each generation. Through a few simple modifications to a generic EA, LEEA is able to compete with stochastic gradient descent (SGD) on multiple benchmark problems on networks with over $1,000$ weights. Moreover, because LEEA transforms the EA into an algorithm that more closely resembles SGD, it opens the possibility of integrating evolution and SGD in ways that were not possible previously.

Approach

LEEA implements the core idea from SGD of limited evaluation for the first time in a very simple traditional generational EA. Instead of evaluating the population against the entire training set each generation, the population is evaluated against only a limited number of training examples each generation. This lean approach to evaluation greatly relieves the computational load, particularly on large training sets, allowing much more rapid network training.

However, one potential weakness of LEEA is that performance on a single example may not correspond to performance across all examples. In SGD, this problem of deceptive examples is tempered by the learning rate and the fact that the population of in effect one individual is never "replaced" by a defective mutant, which prevents the network from shifting too far towards a globally poor configuration during any given evaluation. In contrast, an EA does not inherently contain such safeguards against such deceptive training examples. For example, in the EA, a species well suited to only the present example could displace one that had mastered all the examples before. Two strategies in LEEA mitigate this problem. The first is simple: the population is evaluated against more than one example per generation, though still very few. This strategy reduces the potential damage caused to the population by a single deceptive training example.

It should be noted that while the technique of evaluating the population against a small set of examples each generation may appear to be analogous to a technique employed by SGD called "mini-batching" [18], its motivation in LEEA is different. SGD employs mini-batches primarily to better utilize parallel computational resources, while LEEA employs these mini-batches for more stable population dynamics.

The way the examples are selected for each mini-batch naturally can influence the effectiveness of the algorithm. For instance, if all of the examples selected for a given mini-batch happen to have a

similar expected output, then even degenerate networks that output a constant value may achieve a high fitness. Instead, if the examples are selected so that they have a diversity of expected outputs, then networks will only be highly rewarded when they can also produce a diversity of outputs that match the expected values. As an example, consider the case of applying this approach to the MNIST handwritten digit domain. If a single mini-batch contains only images in the class "1", then a network that always classifies an image as "1" without regard to the input would achieve a high fitness on this mini-batch. However, if the mini-batch contains one image of each class, then networks will only achieve a high fitness through a more complex behavior that can discriminate between the different inputs. This mini-batching approach thereby prevents degenerate networks from ever achieving a high fitness and rewards networks that exhibit heterogeneous behavior.

Even with carefully selected mini-batches, LEEA might still lose individuals who are relatively fit in a global sense, but weak on the examples encountered during any single mini-batch. This danger is particularly acute during early evolution, when the best individuals may only succeed on a small percentage of all examples. To further combat this complication, the second key strategy in LEEA is that fitness is calculated based on both the performance on the current mini-batch and the performance of the individual's *ancestors* against their mini-batches. As long as each step of evolution is not changing the behavior of each network in a radical way, this *fitness inheritance* builds up for those individuals who are more likely to be more globally fit than their peers, regardless of how they performed on the current mini-batch. It is important to note that this form of fitness inheritance is inspired by, though differs from, previous approaches to fitness inheritance that aimed to avoid directly evaluating a portion of the population [36, 106]. To implement fitness inheritance in LEEA, the fitness for individuals produced by sexual reproduction and asexual reproduction, respectively, is given by

$$f' = \frac{f_{p_1} + f_{p_2}}{2}(1 - d) + f, \text{and} \tag{3.1}$$

$$f' = f_{p_1}(1 - d) + f, \tag{3.2}$$

where f' is the individual's modified fitness, f is the fitness of the individual against the current mini-batch, f_{p_n} is the fitness of parent n, and d is a constant decay value.

The introduction of output-diverse mini-batching and fitness inheritance enables LEEA to take advantage of the computational benefits of SGD within the framework of evolutionary computation. Other than these two strategies, LEEA is just a simple generational EA with a mutation power decay (which is analogous to learning rate decay in SGD):

Algorithm 1 LEEA

while $gen < maxGenerations$ **do**
    select mini-batch of training examples
    evaluate population
    modify fitnesses based on fitness inheritance
    select parent(s) from a truncated list with roulette wheel selection
    create offspring – sexual reproduction with uniform crossover (without mutation) or asexual reproduction with uniformly distributed mutation
    reduce maximum mutation power by multiplying by decay constant
    $gen = gen + 1$
end while

This algorithm contains only the bare essentials required for a more traditional EA to work, along with the modifications necessary to combat the complications caused by limited evaluations per generation.

Experiment

To assess the effectiveness of LEEA as an alternative to more traditional neural network optimization methods, performance is tested in three domains against a traditional generational EA (TGEA), SGD, and RMSProp, which is a popular variant of gradient descent based on the idea

that following shallow gradients can often be as useful as following steep ones [116]. While many evolutionary-based approaches exist, the traditional generational EA was chosen as it provides the closest analog to LEEA, allowing this experiment to more easily disentangle the question of whether the limited evaluation process in LEEA provides a benefit. The TGEA works like LEEA, but with *all* training examples evaluated in each generation (instead of a mini-batch), and no fitness inheritance mechanism. In RMSProp, the enhancement to SGD, the current gradient information is divided by a running average of recent gradients. This technique allows the algorithm to escape from plateaus with tiny gradients more quickly. All three domains are chosen because they can benefit from a neural network of moderate dimensionality, i.e. over 1,000 connection weights that must be optimized. In general it is not a common view that a simple EA is suited to optimizing so many parameters in a neural network as effectively as SGD. Therefore these experiments demand an unusual ability for EAs that is rarely contemplated in deep learning literature.

Sample data from each domain is divided into a training set, a validation set, and a test set. All algorithms evaluate performance against the validation set to test for over-fitting. The evolution-based algorithms additionally require this validation performance data to determine which individual from the population is selected for evaluation against the test set. All reported results are thus based on performance against the test set and for evolution the individual tested is the one that performed best against the validation set.

The first domain, with 800 training examples, is a function approximation task, where the function is given by the equation

$$f(x, y) = \frac{sin(5x(3y + 1)) + 1}{2}. \tag{3.3}$$

Preliminary tests indicate that the surface generated by this function is sufficiently difficult to approximate that networks with over 1,000 connections have an advantage over smaller networks, thus making a good test for the effective use of high dimensions.

The second domain is a time series prediction task with 1,200 training examples where the time series is generated using the Mackey-Glass chaotic time series equation

$$\frac{dx}{dt} = \beta \frac{x_r}{1 + x_r^n} - \gamma x, \quad \gamma, \beta, n > 0 \tag{3.4}$$

where $\gamma = 1$, $\beta = 2$, $r = 2$, $n = 9.65$, $x_0 = 1.1$, and $x_1 = 1.2$. The prediction for time t is based on values at $t-6$, $t-12$, $t-18$, and $t-24$. This task has been employed to test neural networks in the past [54], and is also sufficiently complex to potentially benefit from over 1,000 connections.

The third domain is the California housing dataset, which is a housing value prediction task with 10,000 training examples also previously given to neural networks [54, 68]. This task contains observations of housing data where each observation consists of eight input attributes for a particular block of housing (median income, median house age, etc.) and one output for the median house value.

All learning algorithms train with the same network topology, which contains two hidden layers with 50 nodes in the first layer and 20 nodes in the second layer, as well as a single output. Because of differences in the number of inputs, the resultant networks have 1,170, 1,270, and 1,470 connections for the first, second, and third domains, respectively (networks include also a bias node). LEEA and TGEA are both evolved using a direct encoding (i.e. one floating-point gene per connection in the network) with no ability to change the network topology.

To evaluate the effectiveness of each algorithm, they are exposed to the same total number of examples during training. That way, the number of generations given to TGEA equals the number

of epochs given to SGD and RMSProp, while LEEA is given a proportionately higher number of generations (which of course still means the same number of evaluations) than TGEA based on the ratio of total training examples to the number of examples evaluated per generation. Because the EAs must evaluate all members of their population, on a single processor their execution time becomes greater proportionally to the population size. However, parallelization can potentially benefit EAs to a greater degree because each such evaluation is independent. As the capacity for parallelization increases, the instrumental issue thus shifts from population size to the ability to effectively learn from each training example, which is why performance is measured here based on the number of examples evaluated.

Parameters for all algorithms were selected through a parameter sweep on the first domain. There were two key differences found between the ideal parameters of LEEA compared to TGEA. For LEEA, the ideal starting mutation power (the maximum size of a weight mutation) is 0.03, compared to 0.1 for TGEA. This difference reflects that it is better to take smaller steps when evaluating individuals on a small set of training examples. In addition, while TGEA performs best with fitness sharing-based speciation [41] in the spirit of NEAT [108] based on genetic similarity (which maintains diversity), LEEA performs best without any speciation. This observation makes sense because the changing mini-batch in each generation is a force for diversity on its own.

Parameters in common between LEEA and TGEA are population size (1,000), mutation power decay (0.99), mutation rate (0.04), mutation type (uniform), selection proportion (0.4), and sexual reproduction proportion (0.5). Selection proportion refers to the ratio of individuals (sorted by fitness) that are eligible for reproduction.

Ideal mini-batch size in LEEA was determined experimentally to be 2. In all three domains, there is no notion of output classes, but rather a single real number output in the range $\{0, 1\}$. To apply the idea of diversity in mini-batches, the output range is divided into two equally sized

sections, and examples selected so that each mini-batch contains one example for both sections of the range. A new mini-batch is randomly generated at the beginning of each generation (while ensuring the diversity of the expected outputs). The ideal fitness inheritance decay rate for LEEA was determined experimentally to be $d = 0.2$.

SGD and RMSProp were implemented as online learning (i.e. one example at a time). Mini-batching with these methods is not included in the comparison because there is not a consensus on an empirical advantage for mini-batching on a per-epoch basis, and some have even suggested perhaps a slight disadvantage [121]. Therefore, the online case serves as a more transparent baseline that avoids introducing any confounding factors that might come with mini-batching. The training examples are reshuffled at the start of each epoch. The initial learning rate for SGD and RMSProp is 0.3 and 0.001, respectively. The learning rate is decayed exponentially by 0.0001 per epoch for both SGD and RMSProp. RMSProp maintains a queue of 250 recent magnitudes for each parameter for calculating the per-parameter learning rate.

All neurons in every setup are sigmoidal based on the function $f(x) = \frac{1}{1+e^{-x}}$. That way, all methods are compared in equivalent conditions. Of course it is known that other functions like rectified linear units [81] can sometimes help deep networks under SGD, but the aim here is to establish how these methods behave under equivalent controlled conditions to get a sense of their relative capabilities in general.

Results

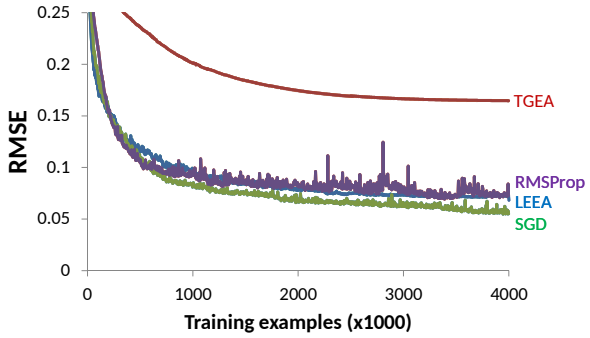
Table 3.1 shows testing results for each algorithm on each domain. Overall test performance in the table is measured as the root mean square error (RMSE) of the individual who performs best against the validation set across the run, where individuals are tested against the validation set at

regular intervals based on the number of training examples in the domain (every 4,000 samples for function approximation, every 6,000 samples for the time series, and every 30,000 samples for California Housing). The individual tested in the evolutionary runs is the one who performs best on validation from the population at the current generation. It is important to note that overall, each algorithm experiences precisely the same number of examples before each validation check, so the amount of training data experienced is always equivalent. Test performance is averaged over 10 runs for each algorithm/domain combination. Some results did not exhibit a normal distribution, so significance between results is calculated with the Mann-Whitney U test. On the function approximation domain, LEEA's performance is not statistically significantly different from RMSProp, but it performs slightly though significantly worse than SGD ($p < 0.01$) and significantly better than TGEA ($p < 0.01$). On the time series domain, though LEEA performs best by a small margin, there is no statistical difference between LEEA, SGD, and RMSProp, and all three algorithms perform significantly better than TGEA ($p < 0.01$). On the California housing domain, LEEA performs slightly though significantly worse than SGD and RMSProp ($p < 0.01$) and significantly better than TGEA ($p < 0.01$). As a whole, comparing results between LEEA and TGEA in all the domains confirms that EA performance can be significantly enhanced by evaluating a limited number of examples per generation. They also show that differences between LEEA, SGD, and RMSProp range from very small to insignificant, with LEEA even performing best (though insignificantly) in one domain, supporting the conclusion that LEEA is a viable option for training neural networks.

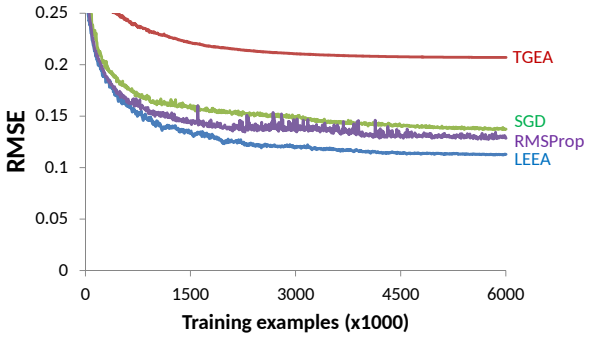
Table 3.1: **LEEA Testing Results.** This table shows the root mean squared error with standard deviation for each algorithm on each domain.

	Function Approximation	**Time Series**	**California Housing**
TGEA	0.1643 ± 0.0121	0.2068 ± 0.0107	0.1216 ± 0.0013
LEEA	0.0711 ± 0.0116	$\mathbf{0.1080 \pm 0.0340}$	0.1147 ± 0.0020
SGD	$\mathbf{0.0539 \pm 0.0087}$	0.1336 ± 0.0380	0.1097 ± 0.0016
RMSProp	0.0636 ± 0.0138	0.1227 ± 0.0410	$\mathbf{0.1092 \pm 0.0007}$

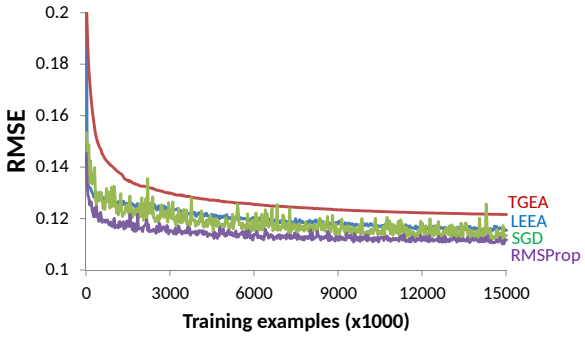
To provide further insight into these results, Figure 3.1 shows average testing accuracy across whole runs, with the x-axis normalized such that all algorithms are evaluated against an equal number of training examples on the same problem at the same point along x. For the evolution-based algorithms, the whole population is evaluated against the validation set at each measurement interval and the individual with the best validation performance is selected for testing. While TGEA learns more slowly and converges to a less optimal solution than SGD and RMSProp, LEEA exhibits a qualitatively similar training curve to these conventional deep learning training algorithms. This observation provides further evidence that LEEA offers an evolution-based alternative for training complex neural networks that is potentially competitive with the state of the art.



(a) Function approximation



(b) Time series



(c) California housing

Figure 3.1: **Average root mean square error on the test set over time.** The average performance over total training examples seen so far as measured against the test set is displayed for each algorithm on the (a) function approximation, (b) time series, and (c) California housing domains. The main result is that the performance of LEEA closely matches that of SGD and RMSProp.

Synthesis

The results suggest that a simple EA with limited evaluations in each generation (LEEA) can optimize neural networks of over 1,000 dimensions about as effectively and in about the same number of iterations as gradient descent algorithms that currently dominate the field of deep learning.

While these results do not prove that such EAs will continue to work well on the neural networks of millions or more weights that now feature in the most cutting-edge results in deep learning [46], they are an intriguing hint that the potential for EAs in this area may be greater than previously believed. At the least it suggests that investigation of such algorithms on larger networks is warranted.

The core question on much larger networks is whether somehow the gradient over the high number of dimensions becomes too hard to approximate for the EA. For example, if such high-dimensional optimization requires effectively sampling changes along nearly every dimension, the EA population might fail to sample densely enough. Yet if it is true that there are many paths to near-optimality as has been argued even in deep learning literature [23], it may not ultimately be essential to sample even a large proportion of the possible paths. In fact, sparse sampling could be a winning trade-off as the price to gain diversity, which conventional SGD lacks. In any case, only further empirical investigation on more complex domains such as MNIST can settle this question.

The potential for EAs to offer an alternative to SGD for training neural networks is compelling because EAs are a genuinely different paradigm for specifying a search problem. That is, the real payoff of such a novel option is not that it might perform better in some scenario, but that it allows researchers to approach problems in entirely different ways and capitalize on different forms of regularization, thereby greatly expanding the toolbox available to neural network researchers. Furthermore, EAs offer options unavailable to SGD like diversity maintenance [79], the evolution of topology [108], and indirect encoding [39, 109]. These can all act as powerful regularizers different from those in SGD. Problems facing SGD in backpropagation like vanishing gradients through multiple layers or through recurrence also would not even exist for neuroevolution through EAs, perhaps enabling radically different architectures to be learned than the ones designed for SGD like LSTMs [51], or even autoencoders [8].

Another interesting possibility is that LEEA may not simply be a potential competitor or replacement to SGD, but that it may enable a bridging between the two different learning paradigms. LEEA's algorithmic similarities to SGD allow entirely new ways of integrating the algorithms in order to exploit the strengths that each has to offer. In this way, LEEA may in the future offer complementary paths to hybridization in addition to DDFA, which is the central focus of this dissertation.

CHAPTER 4: NEW APPROACH: DDFA

This chapter introduces the core idea that forms the foundation for the rest of this dissertation through its subsequent extensions.

Divergent Discriminative Feature Accumulation

Divergent Discriminative Feature Accumulation (DDFA) is a hybrid algorithm developed by Paul Szerlip, Justin Pugh, Ken Stanley, and myself that leverages the strengths of both gradient descent and neuroevolution [114]. DDFA takes advantage of the ability of Compositional Pattern Producing Networks (CPPNs) to produce regular patterns to generate simple features. These simple features are evaluated by novelty search in order to discover features that discriminate in a novel way across the training set. These unique neuroevolution tools permit DDFA to perform this feature accumulation in an unsupervised manner and add new features indefinitely, which means it does not depend on a fixed hidden-layer size. Because novelty search is based on how each feature discriminates across the training set, DDFA features are inherently discriminative from the start even though they are trained without knowledge of the ultimate classification problem. In this way, DDFA provides the first unsupervised and discriminative technique for generating features. These accumulated features are then composed into a neural network that is optimized using stochastic gradient descent. Results confirm that indeed DDFA is a powerful technique for learning useful features.

Approach

Unsupervised pretraining in deep learning has historically focused on generative approaches such as autoencoders and RBMs [11, 50] that attempt to *reconstruct* training examples by first translating them into a basis of features different from the inputs, and then from those features regenerating the inputs. Of course, one potential problem with this approach is that there is no assurance that the learned features actually align with any particular classification or discrimination problem for which they might be used in the future. Yet this conventional approach to learning features also raises some interesting deeper questions. For example, is there any other way to extract meaningful features and thereby learn representations from a set of examples without explicit supervision?

There are some well-known simple alternatives, though they are not usually characterized as feature-learning algorithms. For example clustering algorithms such as k-means or Gaussian mixture models in effect extract structure from data that can then assist in classification; in fact at least one study has shown that such clustering algorithms can yield features as effective or more so for classification than autoencoders or RBMs [16]. This result highlights that reconstruction is not the only effective incentive for extracting useful structure from the world.

More recently, a new algorithm called generative adversarial nets (GANs) was introduced that generates realistic unlabeled training data to augment the data set for semi-supervised training of neural networks [44]. GANs work by generating data through a generative neural network whose goal is to fool the classification neural network into believing that the generated data comes from the actual data set. The data generated by GANs look remarkably similar to the data from the training set, requiring the classification network to learn a richer set of features during the adversarial process and has led to new records on reduced-data versions of multiple data sets [99].

The approach introduced here also goes beyond simple clustering by emphasizing the general ability to learn diverse *distinctions*. That is, while one can learn how to *describe* the world, one can also learn how different aspects of the world *relate* to each other. Importantly, there is no single correct view of such relations. Rather, a rich set of learned relationships can support drawing important distinctions later. For example, in one view palm trees and regular trees share properties that distinguish them from other plants. However, in another view, palm trees are in fact distinct from regular trees. *Both* such views can be useful in understanding nature, and one can hold both simultaneously with no contradiction. When an appropriate question comes up, such as which plants are tall and decorative, the feature *tall* becomes available because it was learned to help make such general distinctions about the world in the past.

The idea in DDFA is to continually accumulate such distinctions systematically through novelty search, thereby building an increasingly rich repertoire of features that help divide and relate observations of the world. Why should accumulating a diverse set of features provide an advantage over optimization-based approaches? One reason is that searching for diversity is not susceptible in the same way to local optima: while e.g. an autoencoder or direct optimization on a classifier can converge to a local optimum, in contrast DDFA features are independently rewarded for diverging from other features, preventing such premature convergence. In addition, in principle, to discriminate one class from another a classifier must possess knowledge of every feature that might distinguish the two classes; DDFA is *explicitly* seeking all such possible distinctions while the objective of learning features that *reconstruct* the input in e.g. an autoencoder is not. Intuitively, DDFA can be interpreted as a formalization of *divergent thinking*, wherein the thinker achieves a novel insight by exploring many perspectives simultaneously and without prejudice.

More formally, suppose there are n training examples $\{x^{(1)}, ..., x^{(n)}\}$; whether or not they are labeled will not matter because feature learning will be unsupervised. Suppose also that any single *feature detector* h_i (i.e. a single hidden node that detects a particular feature) outputs a real number

whose intensity represents the degree to which that feature is present in the input. It follows that h_i will assign a real number $h_i^{(t)}$ to every example $x^{(t)}$ depending on the degree to which $x^{(t)}$ contains the feature of interest for h_i. The output of h_i for all features $x^{(t)}$ where $t = 1, \ldots, n$ thereby forms a vector $\{h_i^{(1)}, \ldots, h_i^{(n)}\}$ that can be interpreted as the *signature* of feature detector h_i across the entire training set. In effect the aim is to continually discover new such signatures.

This problem of continually discovering novel signatures is naturally captured by novelty search, which can be set up explicitly to evolve feature detectors h_i, each of which takes a training example as input and returns a single output. The signature $\{h_i^{(1)}, ..., h_i^{(n)}\}$ of h_i over all training examples is then its *behavior characterization* (BC) for novelty search. The novelty of the signature is then measured by comparing it to the k-nearest signatures in the novelty archive and the current population. The sparseness ρ at point b is given by

$$\rho(x) = \frac{1}{k} \sum_{i=0}^{k} \text{dist}(b, \mu_i), \tag{4.1}$$

where μ_i is the ith-nearest neighbor of b with respect to the distance metric *dist*, which is a domain-dependent measure of behavioral difference between two individuals in the search space. Candidates from more sparse regions of this behavioral search space then receive higher novelty scores, which lead to a higher chance of reproduction. Those features whose sparseness ρ (i.e. novelty) exceeds a minimum threshold ρ_{min} are stored in the growing novel feature collection for later classifier training. In summary, DDFA continually collects features that classify the examples in the training set differently from features that came before, thereby accumulating increasingly diverse means for distinguishing training examples from each other.

A likely source of confusion is the question of whether DDFA is a kind of exhaustive search over signatures, which would not tend to discover *useful* features in a reasonable runtime. After all, the number of theoretically possible distinctions is exponential in the number of training examples.

However, a critical facet of HyperNEAT-based neuroevolution, which DDFA is based upon is that the complexity of features (and hence distinctions) tends to increase over the run [63]. As a result, the initial features discovered encompass simple principles (e.g. is the left side of the image dark?) that gradually increase in complexity. For this reason, the most arbitrary and incoherent features (e.g. are there 17 particular dots at specific non-contiguous coordinates in the image?) are possible to discover only late in the search. Furthermore, because the novelty signature is measured over the training set, features that make broad separations *relevant* to the training set itself are more likely to be discovered early. In effect, over the course of a DDFA run, the feature discoveries increasingly shift from simple principles to intricate minutia. Somewhere along this road are likely diminishing returns, well before all possible signatures are discovered. Empirical results reported here support this point.

Interestingly, because DDFA does not depend on the minimization of error, in principle it can continue to collect features virtually indefinitely, but in practice at some point its features are fed into a classifier that is trained from the collected discriminative features.

Experiment

The key question addressed in this chapter is whether a divergent discriminative feature accumulator can learn *useful* features, which means they should aid in effective generalization on the test set. If that is possible, the implication is that DDFA is a viable alternative to other kinds of unsupervised pretraining. To investigate this question DDFA is trained and tested on the MNIST handwritten digit recognition dataset [61], which consists of 60,000 training images and 10,000 test images. Therefore, the signature of each candidate feature discovered by DDFA during training is a vector of 60,000 real values. While MNIST is not considered a challenging image recognition domain, it

serves as a useful tool in the creation of DDFA, as the relatively low resolution and smaller training set simplify the process of initial development and hyperparameter tuning.

Because the structure of networks produced by HyperNEAT can include as many hidden layers as the user chooses, the question arises how many hidden layers should be allowed in *individual features* h_i learned by HyperNEAT. This consideration is substantive because in principle DDFA can learn arbitrarily-deep individual features all at once, which is unlike e.g. the layer-by-layer training of a deep stack of autoencoders. However, the explicit choice was made in this introductory experiment to limit DDFA to single-layer features (i.e. without hidden nodes) to disentangle the key question of whether the DDFA process represents a useful principle from other questions of representation such as the implications of greater depth. Therefore, feature quality is addressed straightforwardly in this study by observing the quality of classifier produced based only on single-layer DDFA features. As a result, the final classifier ANN has just two layers: the layer of collected features and the ten-unit output layer for classifying MNIST digits.

The single-layer DDFA approach with novelty search and HyperNEAT is difficult to align directly with common deep learning approaches in part because of its lack of permutation invariance even though it is not convolutional in any sense (thereby lacking the representational power of such networks), and its lack of depth in this initial test. Thus to get a fair sense of whether DDFA learns useful features it is most illuminating to contrast it with the leading result on an equivalently shallow two-layer architecture (which are rare in recent years) that similarly avoided special preprocessing like elastic distortions or deskewing. In particular, Simard et al. [105] obtained one of the best such results of 1.6% test error performance. Thus a significant improvement on that result would suggest that DDFA generates useful features that help to stretch the capacity of such a shallow network to generalize. DDFA's further ability to approach the performance of conventional vanilla deep networks, such as the original 1.2% result from Hinton et al. [50] on a four-layer

network pretrained by a RBM, would hint at DDFA's potential utility in the future for pretraining deeper networks.

During the course of evolution, features are selected for reproduction based on their signature's novelty score (sparseness ρ) calculated as the sum of the distances to the k-nearest neighbors ($k = 20$), where neighbors include other members of the population as well as the historical novelty archive. At the end of each generation, each individual in the population (size $= 100$) has a 1% chance of being added to the novelty archive, resulting in an average of 1 individual added to the novelty archive on each generation. Separately, a list of individuals called the *feature list* is maintained. At the end of each generation, each member of the population is scored against the current feature list by finding the distance to the nearest neighbor ($k = 1$), where neighbors are members of the feature list. Those individuals that score above a threshold $\rho_{\text{min}} = 2{,}000$ are added to the feature list. In effect, the feature list is constructed in such a way that all collected features have signatures that differ by at least ρ_{min} from all others in the collection. This threshold-based collection process protects against collecting redundant features. A simple variant of HyperNEAT called HyperNEAT-LEO [117] was the main neuroevolution engine. HyperNEAT-LEO, which reduces connectivity in the substrate by removing connections based on the value of an additional CPPN output, was selected for its ability to produce feature detectors that focus on subsections of the input space. The HyperNEAT setup and parameters can be easily reproduced in full because they are simply the default parameters of the SharpNEAT 2.0 publicly-available package (Green 2003–2014).

To observe the effect of collecting different numbers of features, DDFA was run separately until both 1,500 and 3,000 features were collected. After collection concludes, a set of ten classification nodes is added on top of the collected features, and simple backpropagation training commences. The training and validation procedure mirrors that followed by Hinton et al. [50]: first training is run on 50,000 examples for 1,000 epochs to find the network that performs best on a 10,000-

example validation set. Then training shifts to the full 60,000-example set, which is trained until it reaches the same training error as in the best validation epoch. The resulting network is finally tested on the full 10,000-example test set. The entire process of collecting feature and training the network was performed five times for each feature collection size.

Results

The main results are shown in Table 4.1. DDFA was able to achieve average test errors of 1.36% and 1.20% from collections of 1,500 and 3,000 features, respectively, which are both well below the 1.6% error of the similar shallow network trained without preprocessing from Simard et al. [105]. In fact, the average for the 3,000-feature network matches the 1.2% error of the significantly deeper network of Hinton et al. [50], and the best run from DDFA beat the best network from Hinton et al. [50] with an error of 1.16%. This result shows that shallow networks can generalize surprisingly well by finding sufficiently high-quality feature sets, even despite a lack of exposure to distortions during training. It also appears that more collected features lead to better generalization, at least at these sizes. It took an average of 338 and 676 generations of feature collection to obtain the 1,500 and 3,000 features, respectively. Collecting 3,000 features took about 36 hours of computation on 12 3.0 GHz cores.

Table 4.1: **MNIST Testing Results.** This table shows the average error and standard deviation for DDFA and the controls over five runs.

Features	**DDFA Test Error**	**Random CPPNs**	**Random Weights**	**Random Weights - Sparse**
1,500	0.0136 ± 0.0008	0.0156 ± 0.0006	0.0208 ± 0.0006	0.0178 ± 0.0012
3,000	$\mathbf{0.0120 \pm 0.0003}$	0.0153 ± 0.0005	0.0180 ± 0.0009	0.0176 ± 0.0013

Figure 4.1 shows a typical set of features collected by DDFA. Interestingly, unlike the bottom layer of deep learning networks that typically exhibit various line-orientation detectors, DDFA

also collects more complex features because newer features of increasing complexity evolve from older features and these features are full-field and non-convolutional.

Figure 4.1: **Example of collected features.** Each square is a weight pattern for an actual feature discovered by DDFA in which white is negative and black is positive. As these examples show, features range from simple line orientation detectors reminiscent of those found in the lower levels of conventional deep networks (towards left) to more complex shapes (towards right).

To rule out the possibility that the reason for the testing performance is simply the HyperNEAT-based encoding of features, a **random CPPNs control** was also run. It follows an identical procedure for training and testing, except that novelty scores and adding to the feature list during the feature accumulation phase are decided *randomly*, which means the final collection in effect contained random features with a range of CPPN complexity similar to the normal run. To further investigate the value of the HyperNEAT representation, an additional **random weights control** was tested whose weights were assigned from a uniform random distribution, bypassing HyperNEAT entirely. Finally, a control that initializes weights based on a technique known as *sparse initialization* was tested. This final control is an example of one of the more successful initialization heuristics, and acts as a good benchmark to compare DDFA against. As the results in Table 4.1 show, the CPPN encoding in HyperNEAT provides a surprisingly good basis for training even when the features are entirely randomly-generated. However, they are still inferior ($p < 0.001$) to the features collected by DDFA. As shown in the last column, without HyperNEAT, testing performance with a collection of random features is unsurprisingly poor. In sum these controls show that the pretraining in DDFA is essential to priming the later classifier for the best possible performance.

Also of interest is the seeming ability for DDFA to continue improving performance as the number of features increases. While SGD was unable to improve performance from $1,500$ to $3,000$ features when initialized with sparse initialization, and the same is true for random CPPNs, the DDFA runs show a significant improvement ($p < 0.01$ by Student's t-test). This observation raises the intriguing possibility that DDFA could provide a path forward for training very wide networks.

Synthesis

This chapter introduced a new hybrid algorithm named DDFA that combined an unsupervised feature discovery process through neuroevolution with gradient descent for training neural networks. DDFA was successfully applied to the MNIST domain, providing gradient descent with a set of starting features in the first layer that discriminate across the training set in diverse ways. These features provide a starting point that allows gradient descent to generalize better across the test set compared to randomly initialized features. While MNIST is not considered a challenging domain since the rise of deep learning, it nevertheless provided a useful test bed for laying the foundation of DDFA. The relatively simple domain allowed focus to remain on disentangling the issues that arise from developing a new type of feature discovery algorithm.

Furthermore, the unsupervised nature of the feature discovery process in DDFA deserves special attention. This capacity allows DDFA to discover features from unlabeled data, which is often in greater supply than labeled data. One of the challenges in applying deep learning is the need for abundant labeled training data. While other approaches to unsupervised pretraining have typically focused on generative models, such as RBMs and GANs, DDFA provides a discriminative model for unsupervised pretraining. Therefore, if DDFA can learn features from unlabeled data that deep learning can then exploit, it could assist in the application of deep learning to a wider range of domains.

The implications of this work extend beyond the development of a new algorithm. While DDFA has shown that neuroevolution techniques can provide a powerful set of tools for discovering neural network features, and the unsupervised nature of DDFA is potentially very useful on its own, the more interesting aspect of DDFA is that it combines the strengths of neuroevolution with those of gradient descent. In this way, DDFA treats these two traditionally rival approaches to neural network training as complementary. This hybridization paves the way for the possibility of a new class of algorithms that exploits both neuroevolution and gradient descent.

CHAPTER 5: DDFA DISTANCE FUNCTION

A critical component of the feature discovery process in DDFA is the novelty search algorithm, which was introduced in Chapter 2 and discussed in more detail in Chapter 4. For novelty search to function, the distance of an individual to its k-nearest neighbors must be determined. To this end, DDFA requires a function to compute how different an individual is compared to other individuals in the population and an archive of individuals discovered in previous generations. Because behaviors are typically represented as vectors, this comparison is traditionally made with a distance function that operates on vectors.

The introductory conception of DDFA in the previous chapter chooses Euclidean distance for the distance function. While this distance function is the intuitive choice due to our everyday experience with distances in two and three dimensions, there are challenges with applying Euclidean distance to machine learning problems. First, Euclidean distance tends to focus on dimensions where there are large differences while ignoring those where it is more difficult to draw a distinction. For instance, if an image contains relatively little information, such as a number "1" drawn as a simple straight line, it will be difficult for features to produce much variability on this image. Therefore, this dimension in the behavior vector may be largely irrelevant to the Euclidean distance function, where larger contributions are given more attention. Second, Euclidean distance tends to draw distinctions on pairs of points that are aligned in the same direction, but differ by magnitude. To understand why this is a problem, consider a natural language problem where a document is vectorized by counting the frequency of each word in the dictionary. The phrase "Bob likes ice cream." would be represented by the vector $[..., 1, ..., 1, ..., 1, ..., 1, ...]$ and the phrase "Bob like ice cream. Bob likes ice cream." would be represented by the vector $[..., 2, ..., 2, ..., 2, ..., 2, ...]$. While the second phrase doesn't present any new information, the Euclidean distance between the vectors is non-zero. In fact, the distance between these vectors is just as large as the distance of the first

phrase to the origin. To address these two specific problems with Euclidean distance, two distance functions, which are simple variants of Euclidean distance, are examined in this chapter: Weighted Euclidean distance and cosine distance.

A broader problem also exists with Euclidean distance due to the large size of behavior vectors in DDFA, which have one dimension for each image in the training set. One of the more challenging aspects to selecting an appropriate distance function is the difficulty in visualizing spaces with more than the three dimensions we deal with in our daily lives. Many of our intuitive notions break down in high dimensions, which makes it more difficult to understand what a distance function is actually measuring in such a space. When explaining these strange phenomena, Domingos [28] notes:

> This is only one instance of a more general problem with high dimensions: our intuitions, which come from a three-dimensional world, often do not apply in high-dimensional ones. In high dimensions, most of the mass of a multivariate Gaussian distribution is not near the mean, but in an increasingly distant "shell around it; and most of the volume of a high-dimensional orange is in the skin, not the pulp. If a constant number of examples is distributed uniformly in a high-dimensional hypercube, beyond some dimensionality most examples are closer to a face of the hypercube than to their nearest neighbor. And if we approximate a hypersphere by inscribing it in a hypercube, in high dimensions almost all the volume of the hypercube is outside the hypersphere. This is bad news for machine learning, where shapes of one type are often approximated by shapes of another.

These strange phenomena are collectively referred to as the *curse of dimensionality*, a term coined by Richard E. Bellman [7]. This problem is particularly acute for DDFA, where the behavior vector may be tens or hundreds of thousands of dimensions. To address the curse of dimensionality in

DDFA, two distance functions that are targeted at this problem are examined: Manhattan distance and fractional norm distance.

Beyond the ability to meaningfully determine distances between individual vectors in such high-dimensional space, there are also practical considerations when selecting a distance function. The distance functions discussed thus far, including Euclidean, are bivariate, meaning they operate on pairs of vectors. Because novelty search compares each individual to the rest of the population as as well as an archive of previously discovered individuals, the novelty score calculation has a computational complexity of $\mathcal{O}(n^2 + nm)$ where n is the population size and m is the archive size. As the archive size grows over the course of evolution, the demands of the novelty calculation can come to dominate and in fact the novelty calculation comes to dominate the feature discovery process as a whole. Because of this effect, a non-bivariate distance function, named Component Neighbor Distance, and designed for efficient novelty calculation is also examined.

While algorithms exist that approximate the nearest neighbor search, such as locality sensitive hashing [22], the goal of this chapter is an accurate evaluation of each distance function. To that end, only exact methods are applied to the bivariate distance functions.

This chapter explores three hypotheses: (1) Selecting a distance function that addresses specific shortcomings of Euclidean distance will lead to a better set of features for the task of classification. (2) Selecting a distance function that more accurately represents distances in high dimensions will lead to a better set of features for the task of classification. (3) A less accurate, but computationally simpler distance function, can be devised to aid in the application of DDFA to larger problems and larger networks.

To test these hypotheses, several possible distance functions are analyzed from a theoretical perspective and then tested empirically to determine their suitability for DDFA.

Euclidean Distance

One of the most widely used distance functions by novelty search practitioners is Euclidean distance, also known as the L_2 norm, which is given by the equation

$$d(p, q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}, \tag{5.1}$$

where p and q are vectors. This distance function seems like a natural choice due to its traditional application to 2-D and 3-D spatial distance problems, and in fact it was the distance function of choice for the initial work on DDFA in the previous chapter. However, while Euclidean may provide a good metric in lower dimensions, it has specific problems when applied to this domain, as mentioned in the introduction to this chapter, and it struggles to discriminate between the nearest and furthest neighbor in very high dimensions and therefore fails to provide a helpful measure of distance [1].

Alternative Distance Functions

In response to the challenges with Euclidean distance, this section explores several possible alternative distance functions from a theoretical perspective.

Weighted Euclidean Distance

Weighted Euclidean distance is meant to address the issue discussed previously where Euclidean distance will tend to focus on dimensions where it is easier to draw large distinctions, while ignoring dimensions where the differences are more subtle. This distance metric is simply Euclidean distance where each component's contribution to the distance is weighted in some way, and is

given by the equation

$$d(p,q) = \sqrt{\sum_{i=1}^{n} w_i (p_i - q_i)^2}, \tag{5.2}$$

where w_i is a weight vector. Because the goal is to produce novel behaviors, the weight contribution for each component is calculated as the inverse of the range of values that have been observed along that dimension, as given by the equation

$$w_i = \frac{1}{(p_{i_{max}} - p_{i_{min}})}, \tag{5.3}$$

where $p_{i_{max}}$ and $p_{i_{min}}$ are the maximum and minimum values observed for this component, respectively. In simpler terms, images where little variance in feature behavior has been observed are given more importance when calculating feature distance. While this distance function still relies upon Euclidean distance and will encounter the curse of dimensionality, the weight component may still provide pressure towards the discovery of features that discriminate well on images where previous features have failed to provide a meaningful distinction. One possible concern regarding this equation is that once a single high and low value are discovered for a given component, that component will no longer be given any additional weight, even if the remaining values for that component are the same. To address this concern, a simple heuristic is added where the top n and bottom n values for that component are averaged to compute the minimum and maximum. Preliminary experiments set the value of n to 5.

Cosine Distance

Cosine similarity is a popular measure of similarity in multiple areas of machine learning [66, 84, 125], and is meant to address the problem where vectors that lie along the same axis, but are

different lengths, produce a non-zero distance in Euclidean space. Cosine similarity is calculated as the cosine of the angle between two vectors, as given by the equation

$$s(p, q) = \frac{p \cdot q}{\|p\|\|q\|}. \tag{5.4}$$

Because novelty search requires a measure of distance between two vectors and cosine similarity provides a measure of similarity, which is in essence the inverse of distance, a conversion function is applied, given by the equation

$$d(p, q) = \frac{1 - s(p, q)}{2}. \tag{5.5}$$

It should be noted that cosine distance is actually equivalent to Euclidean distance when the vectors are normalized. This distance function has an advantage over Euclidean distance in that vectors that align along the same axis, but that are different lengths, will have a cosine distance of 0, while their Euclidean distance may be high. Because the features discovered by DDFA are composed into a neural network with a successive layer, and the weights in that successive layer are adjusted through learning, the magnitude of the behavior vector is a less important consideration than the direction of the vector. Cosine distance therefore potentially focuses on the more important aspect of the behavior vector. However, cosine distance is not a cure for the curse of dimensionality, and in theory suffers nearly as much as Euclidean distance.

Manhattan Distance

Manhattan distance is the first distance function in this work that aims to address the curse of dimensionality. It is defined as the distance between two points as measured along the axes, and is

given by the equation

$$d(p, q) = \sum_{i=1}^{n} |p_i - q_i|. \tag{5.6}$$

Manhattan distance derives its name from the distance it takes to travel between two points in Manhattan, where the streets are laid out in a grid pattern. While it isn't intuitively clear that this distance metric is better than Euclidean distance in high dimensions, analysis of the behavior of randomly selected points in high dimensional space shows this to be the case.

Aggarwal et al. [1] provide both a theoretical and empirical analysis of L_k norms in high dimensional space. Recall that the L_2 norm is Euclidean distance, since the component differences are squared. Manhattan distance is therefore the L_1 norm, and in fact an infinite number of L_k norms exist. In their work, Aggarwal et al. [1] investigate the behavior of L_k norms on a set of randomly placed points in high dimensional space. If $Dmax_d^k$ represents the distance from the origin of the furthest point and $Dmin_d^k$ represents the distance from the origin of the nearest point, where k is the norm and d is the number of dimensions in the space, then then as d increases, $Dmax_d^k - Dmin_d^k$ increases at a rate of $d^{1/k-1/2}$. Therefore, for values of $k > 2$, this expression converges to 0, meaning the distance metric no longer provides any meaningful contrast between the furthest point and nearest point. For Euclidean distance, where $k = 2$, this expression is bounded by constants and for Manhattan distance, where $k = 1$, this expression diverges to ∞, which means it provides much better contrast between points than Euclidean distance. This counterintuitive idea that Manhattan distance provides a better notion of distance in high dimensions is then shown empirically by several classifications tasks with k-means clustering. In every domain that it was tested, Manhattan was superior to Euclidean distance at representing distances in high dimensions.

Fractional Norm Distance

Based on the observation that for the L_k norm, lower numbers provide more contrast, Aggarwal et al. [1] also explored the possibility of going beyond the L_1 norm (Manhattan distance) with *fractional norms*. Fractional norms, which are defined as an L_k norm where $0 < k < 1$, were shown empirically to be even more effective at representing distances than the L_1 norm, with a range of $0.1 < k < 0.5$ being ideal. For this work, k was selected in the middle of this range at 0.25, giving the equation

$$d(p, q) = (\sum_{i=1}^{n} |p_i - q_i|^{1/4})^4. \tag{5.7}$$

Aggarwal et al. [1] also showed empirically that fractional norm distance is more robust to noisy data, which makes intuitive sense when considering that taking a fractional power of the distance minimizes the effect of outliers in the data. While the domains on which DDFA is tested in this work are not noisy, noise is still an important consideration if DDFA is to be successfully applied to a broader range of domains.

Component Neighbor Distance

The distance functions discussed to this point are all examples of bivariate distance functions, where distance is calculated between pairs of individuals. As mentioned earlier in this chapter, the high-dimensional nature of the behavior vectors in DDFA leads to high demands on computation for bivariate distance functions. Not only do these demands increase as the behavior vector expands based on the size of the training set, but if a larger set of features is desired, evolution may need to be extended to collect enough interesting features. To aid in the application of DDFA to larger networks or larger training sets, a non-bivariate distance function, called Component Neighbor

Distance (CND), has been developed. Based on the principle that the desired outcome of DDFA is a set of features that provide a wide range of output values for all given inputs, the components within the behavior vector are decoupled from each other and considered separate values. The equation describing this decoupling is

$$d(p, Q) = \sum_{i=1}^{n} |p_i - Q_{i_{nearest}}|, \tag{5.8}$$

where p is the target vector, Q is a collection of vectors, and $Q_{i_{nearest}}$ represents the closest neighbor to p on the $i - th$ component. In this way, CND ignores the global direction of the vector and instead considers only the distance of each component to its nearest neighbor. While it intuitively seems that this loss in direction information may hinder the discovery of effective features, it also reduces the computational burden of novelty search. As mentioned earlier in this chapter, bivariate distance metrics all have a computational complexity of $\mathcal{O}(n^2 + nm)$. When component values are stored as sorted lists, CND has a computational complexity of $\mathcal{O}(nlog(m))$. Furthermore, the behavior space of each component can be discretized into bins, further reducing the computational complexity to $\mathcal{O}(n)$. This discretization requires a slight modification to the distance function, which is described by the equation

$$d(p, Q) = \sum_{i=1}^{n} \frac{1}{Q_{bin}(p_i)}, \tag{5.9}$$

where $Q_{bin}(p_i)$ corresponds to the bin that p_i maps to. Preliminary experiments revealed no loss in performance when the behavior space is discretized in this way. In its entirety, CND offers a far faster method for calculating novelty, which allows DDFA to be applied to larger training sets or when a larger set of features is required.

QMNIST Domain

To assess the empirical performance of the candidate distance functions, DDFA is applied to a variant of the MNIST domain, called QMNIST [123]. One of the criticisms of the MNIST dataset is that it contains too few images in the test set. Through the process of model testing and selection by researchers, it becomes possible to overfit the test set. A larger test set helps alleviate this issue and to this end, QMNIST was created by reconstructing digits from the original NIST dataset that MNIST was derived from. QMNIST contains $60,000$ test images instead of $10,000$, and while Yadav and Bottou [123] observe that previously published models that they tested did not appear to overfit the test set, QMNIST is chosen here to assist in preventing any possible test set overfitting.

While MNIST and QMNIST are no longer considered challenging image recognition domains, they serve as a useful tool in this work as the intricacies of the DDFA feature discovery process are disentangled. With relatively low resolution (28 x 28) grayscale images and a smaller training set, the process of hyperparameter tuning is less computationally cumbersome, which is an important consideration with a complex neuroevolution system like DDFA. Once the fundamentals of how of the feature discovery process are understood, DDFA can be more readily applied to more complex domains. This approach mirrors the development of deep learning, which started on simpler domains like MNIST [50].

Experiment

Novelty search drives the selection of features within the evolutionary process in DDFA, and the distance function plays a crucial role in that process. The experiments here are designed to give a better understanding of the effect of the distance function on the quality of the discovered features,

as determined by their ability to aid gradient descent in training a network to effectively generalize over the test set.

While CPPNs can encode features of any arbitrary depth or complexity, the choice was made here to restrict the feature substrate produced by the CPPN to only a single layer that is fully connected to the input. This simple setup mirrors the initial work on DDFA in the previous chapter to give focus to the primary question of what effect the distance function has on feature discovery in DDFA. The final network architecture from the previous chapter is also re-used, with $3,000$ features selected from the DDFA archive and composed into a fully-connected network with a single hidden layer.

During evolution, features are selected for reproduction based on their novelty score, which for bivariate distance functions is calculated as the sum of the distance to the k-nearest neighbors $(k = 20)$, where the neighbors include both the population and the historical novelty archive. With bivariate distance functions, individuals have a small chance (1.5%) of being added to the archive at the end of each generation. For Component Neighbor Distance (CND), only the target bin for each component is considered in the distance calculation and a cutoff, c, is added so that once a bin is sufficiently explored, it will no longer contribute to the novelty score. Preliminary experiments showed a value of 10 for c is beneficial compared to no cutoff. Because the size of the archive has no effect on the computational complexity for CND, all individuals are added to the archive at the end of each generation. The population size was increased to 144 to make better use of parallel computational resources and all runs were executed for $1,000$ generations.

At the end of evolution, every individual that was discovered has a chance to be selected by an iterative tournament selection process given by Algorithm 2. This algorithm iteratively evaluates a random set of features from the full collection of all features produced during evolution. In each iteration, it selects the feature from this set that is maximally distant from its nearest neighbor in

the set of features that have already been selected. In effect, this algorithm is a simple heuristic that produces a final set of selected features that are relatively distant, covering a significant portion of the discovered feature behavior space.

Algorithm 2 DDFA Selection Algorithm

```
S = select 1 feature at random
while count(selectedFeatures) < 3000 do
    T = select tsize features from the collection
    for all t ∈ 𝒯 do
        t_min ← find nearest neighbor in S
    end for
    S.add(select t where t_min = max(T_min))
end while
```

In the experiments in this chapter, $tsize$ is set to 16. Note that for Component Neighbor Distance, there is no concept of nearest neighbor and instead each tournament member is evaluated in the same way as they were during evolution, with the list of selected features acting as the novelty archive. This selection algorithm differs from the previous chapter as it was observed that the threshold-based selection algorithm that was previously implemented has a bias towards features discovered early in the run and the threshold requires frequent adjustment to exploit the entire feature collection. Additionally, the tournament selection approach implemented here does a better job at maximizing the average distance between the features, which more directly addresses the primary question of this chapter: Which distance function(s) provide the best measure of feature quality, as measured by their ability to serve as an effective starting point for gradient descent? Finally, with threshold-based feature selection, the first several features are typically unhelpful as they are only added due to showing up early in a vast feature space. While this property does

not appear to hinder performance to a great degree when collecting $3,000$ features, the problem becomes more acute in the work in the next chapter, when fewer features are selected overall.

Once DDFA has generated the required set of features, they are composed into a fully connected network that is then trained with SGD in PyTorch. PyTorch parameters are set to a mini-batch size of 64, learning rate of 0.1 and momentum of 0.5. The training and validation procedure mirrors that from the previous chapter, where a validation run is first conducted with a subset of training examples set aside for testing. The training error that corresponds to the best test error during this validation run then serves as a stopping point during the training run over the full training set. Once this stopping point is reached during the full run, the network is tested against the standard test set. This process of collecting features, performing a validation run of SGD, and performing the final run of SGD is executed ten times for each distance metric. In addition, a set of ten runs is conducted on a network with identical topology and randomly initialized weights and another set of ten runs is conducted on a network with identical topology and initialized with random CPPNs, where the process for generating the random CPPNs mirrors that of the previous chapter. This control is necessary to rule out the possibility that regularities generated by CPPNs are the only component that contributes to the improvement in performance of DDFA over the random control. In other words, an improvement in performance over the random CPPN control indicates that novelty search is contributing to the feature discovery process.

Results

Results of the distance function experiment are shown in Figure 5.1. Taken as a whole, the results largely mirror the results from the previous chapter. The random CPPN control is significantly better ($p < 0.001$ by Student's t-test) than the random initialization control and all DDFA distance functions are significantly better ($p < 0.001$) than the random CPPN control. Another observation

of the results as a whole is the general rise in error rate for all DDFA variants, and to a lesser degree, the randomly initialized and random CPPN controls, as compared to the results from the previous chapter. This rise is due in large part to the change in domain to QMNIST, which is slightly more challenging than its predecessor. However, the increased error in the results here align well with previously observed increased error rates on QMNIST by a wide range of models [123].

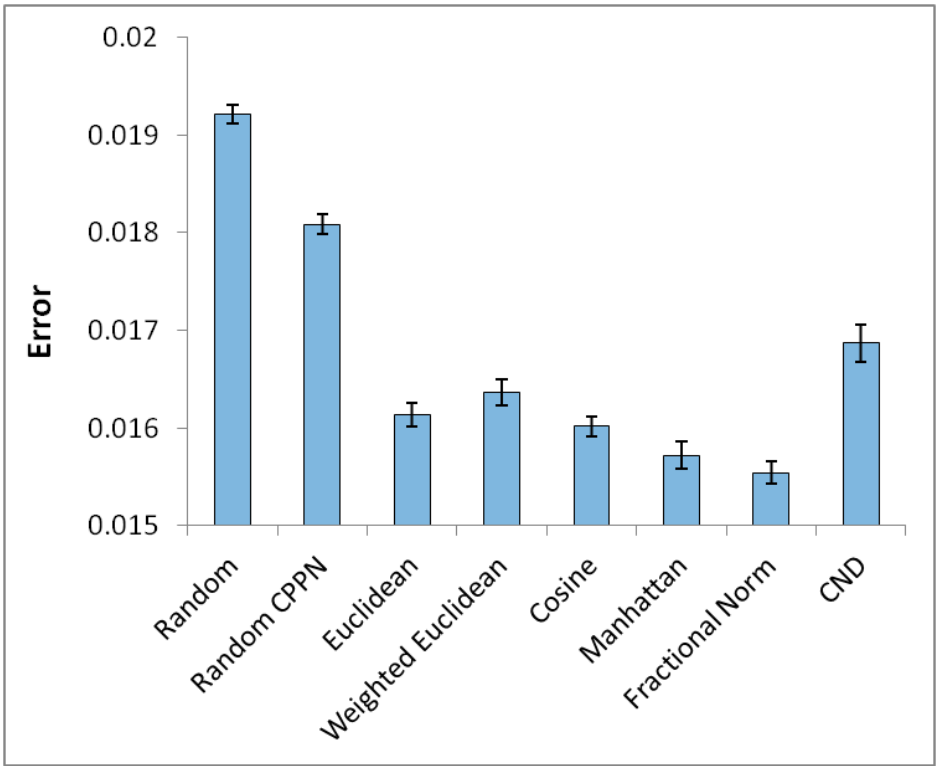


Figure 5.1: **Average testing error on the QMNIST test set by distance function.** The average testing error (over 10 runs) on the QMNIST test set is shown for each distance function, and error bars represent standard error. Two controls are included where the weights are initialized randomly and where the weights are initialized by random CPPNs. The primary result shown here is that the distance functions that were chosen for their ability to handle high dimensions, Manhattan distance and fractional norm distance, perform significantly better than Euclidean distance.

The results of the individual distance functions are compared against Euclidean distance and can be divided into three groups based on whether they perform better, worse, or equal to Euclidean distance. Fractional norm and Manhattan distance perform significantly better ($p < 0.05$) than Euclidean distance, weighted Euclidean and cosine distance show no significant difference from Euclidean distance, and Component Neighbor Distance is significantly worse ($p < 0.05$) than Euclidean distance.

These results make clear that the primary issue impacting the performance of the distance function is the curse of dimensionality. The distance functions that were designed to improve upon specific limitations of Euclidean distance without addressing the dimensionality issue, weighted Euclidean and cosine, failed to deliver any improvement over Euclidean distance. Meanwhile, the distance functions that were selected for their ability to address the curse of dimensionality, fractional norm and Manhattan, do provide improved performance over Euclidean distance.

In addition, while Component Neighbor Distance fell behind the rest of the distance metrics, it is important to note that it was still significantly better ($p < 0.001$) than the random CPPN control, which makes it suitable for problems where bivariate distance functions may be unusable due to computational constraints. For example, when compared against Euclidean distance, Component Neighbor Distance takes about half the time to calculate novelty in the early generations, when the novelty archive is empty. Even more striking are the results as the novelty archive fills throughout evolution. While the novelty calculation for Component Neighbor Distance remains constant due to the bin-based approach, the computation increases linearly with the size of the novelty archive for bivariate distance functions. In the last few generations of experiments performed on the QMNIST domain, Euclidean distance takes around 50 *times longer* to calculate than Component Neighbor Distance. Therefore, while Component Neighbor Distance may not be the ideal choice in distance function from a feature quality standpoint, it does provide an avenue to discovering moderately useful features on domains where computation is the primary consideration.

Synthesis

The results of this chapter provide an important insight into the factors that affect the performance of DDFA. One of the core principles behind DDFA is that a set of features that discriminates across the training set in a diverse way provides a beneficial starting point for gradient descent. This chapter helps address the question of how that diversity is measured. The distance function within novelty search provides a mathematical description of diversity, and the results from this chapter show that the distance function is an important consideration when applying DDFA. Work on the initial development of DDFA in the previous chapter applied Euclidean distance, as it is the most common and intuitive distance function, to novelty search within DDFA. However, Euclidean distance suffers from several problems that led to the investigation of alternative distance functions in this chapter.

The first hypothesis of this chapter was that two specific issues with Euclidean distance could be remedied directly through modifications to the distance function, leading to improved test set performance. Weighted Euclidean distance was introduced as a solution to the problem where Euclidean distance would tend to favor components of the behavior vector that have a lot of variability. In terms of application to a visual domain, Euclidean distance gives a larger contribution to those images where a large range of values could be produced easily. Cosine distance was introduced as a solution to the problem where Euclidean distance would results in a non-zero, and sometimes large, distance between two behavior vectors that lie on the same axis, but have different magnitudes. Empirical results showed no improvement with these two distance metrics, suggesting that these perceived shortcomings are not sufficiently acute to prevent Euclidean distance from discovering useful features. Therefore, the first hypothesis was not confirmed.

The second hypothesis of this chapter was that selecting a distance function that addresses the problem of the curse of dimensionality would lead to improved test set performance. Manhattan

distance and fractional norm distance were selected to test this hypothesis, as Aggarwal et al. [1] had found these distance functions to best represent distances in high dimensions. Empirical results here confirmed that this is indeed the case, with both distance functions significantly outperforming Euclidean distance. This result further supports the idea that an accurate measure of distance is an important factor in the performance of DDFA, and confirms the second hypothesis.

The third hypothesis of this chapter was that there exists a more computationally efficient means to measure distance with enough efficacy to provide useful starting features for gradient descent. To this end, Component Neighbor Distance was introduced, which reduces the computational complexity of the novelty calculation from $\mathcal{O}(n^2 + nm)$ to $\mathcal{O}(n)$, while still providing better empirical results than either randomly initialized weights or randomly generated CPPNs. The ability for Component Neighbor Distance to provide features that performed significantly better than random CPPNs confirms the third hypothesis, and opens the door to application of DDFA to larger domains or larger networks.

Taken as a whole, these results highlight the importance of the distance function on the quality of features produced by the novelty search algorithm that is at the heart of DDFA. Empirical results show that the most important factor in the discovery of high-quality features is the accurate measure of distance in high dimensions.

Furthermore, while understanding the impact of the distance function on high dimensional behavior vectors is important in the context of this work, it also provides an important lesson for the proper application of novelty search within the larger neuroevolution community. While behavior vectors in this dissertation are larger than typically encountered due to the domains that it is applied to, the curse of dimensionality could potentially cause similar complications on domains with smaller behavior vectors where neuroevolution researchers often apply novelty search. Additionally, neuroevolution researchers may artificially limit the size of their behavior vectors due

to problems encountered as dimensionality increases. Therefore, the results from this chapter can serve as a guide for selecting an appropriate distance function for these researchers, and provide a pathway towards more descriptive behavior vectors.

CHAPTER 6: CONVOLUTIONAL DDFA

As discussed in Chapter 2, *convolution* is an important component to the application of deep learning across a wide range of machine learning domains. The concept of a convolutional feature, more commonly referred to as a *convolutional filter* in the literature, is fundamentally different from the fully-connected features that DDFA has been applied to discovering in previous chapters. A convolutional feature is typically much smaller in size than the input data, often in the range of 3×3 to 11×11 in image recognition domains [48, 57, 93, 113]. These features are convolved across the image to produce a *feature map* that in essence indicates the relative presence of a feature at each point in the image. These unique properties of convolutional features present special challenges to the application of DDFA, which are explored in this chapter. The hypothesis here is that DDFA can discover convolutional features that provide a better starting point for gradient descent, leading to better test performance. This chapter gives a discussion of the theoretical considerations in applying DDFA to convolutional features and includes experiments that provide empirical support to the hypothesis, suggesting that DDFA may be applied to convolutional neural networks.

Approach

The basic application of DDFA to the discovery of convolutional features is similar to the application of DDFA to fully-connected features, with two small differences. First, the size of the input layer in the HyperNEAT substrate is reduced. Rather than an input layer that equals the size of each image in the domain, only a small section, 5×5 or 9×9, acts as the input layer for the substrate. The output of the substrate remains the same, with a single output neuron. The convolutional feature is then convolved across the image to produce a feature map, which contains a behavior vector component for each location in the image. This feature map is converted to a single vector, whose

size is given by the equation

$$V_{size} = (n - m + 1)^2, \tag{6.1}$$

where the domain images are of size $n \times n$ and the convolutional features are of size $m \times m$. Therefore, while the full behavior vector in non-convolutional DDFA contains a single component for each input image, the behavior vector in convolutional DDFA contains *many* components for each image.

The increase in the size of the behavior vector introduces the first complication in applying DDFA to convolutional features. For instance, the behavior vector for convolutional DDFA on the QMNIST domain is 576 times larger than the behavior vector in the previous chapter. This major expansion presents an obvious computational challenge, while also exacerbating the problem of the curse of dimensionality. This complication, introduced by the nature of convolutional features, is not unique to the DDFA process. Deep learning faces similar computational hurdles due to the nature of convolution, and two of the techniques developed to handle this problem in deep learning can be borrowed to simplify the evaluation of the behavior vector in DDFA.

The first technique commonly employed for reducing the dimensionality of the output of a convolutional layer is called *striding*, which refers to the process of only applying the convolutional features at regular intervals in both the x and y dimensions of the input space. For instance, a stride of 3 will skip every second and third location in both the x and y dimensions, therefore reducing the number of elements in the feature map by a factor of 9. While striding introduces the theoretical concern that a feature in the image will be missed by a convolutional filter that would normally pick up on that feature, the widespread application of striding indicates that this concern does not present a practical problem.

The second technique for reducing the dimensionality of the output of a convolutional layer is called *pooling*, which refers to the process of down sampling feature map components to reduce dimensionality. The most common methods of pooling are max-pooling and mean-pooling, where the maximum or mean of a set of feature map components is passed along, therefore reducing the dimensionality of the convolutional layer output by a factor of $n \times n$ for a pool size of n. Pooling in deep learning is typically implemented as a special network layer, which is necessary in order to propagate error back to the convolutional layer for updating the weights. Preliminary experiments with DDFA showed no improvement when basic types of pooling were implemented, and therefore pooling was not included in any experiments in this dissertation. However, pooling offers a potential bounty of future extensions. Because the purpose of pooling in DDFA is for reducing the size of the behavior vector, there is no reason why it must be restricted to the basic forms of pooling available in deep learning, where the requirement of error propagation imposes its own limitations. For example, pooling could be performed across much larger portions of the image, with the behavior vector components corresponding to various statistics about the feature location(s), intensity, or frequency. In fact, there are very few restrictions on the way this pooling could occur, so long as there is a way to meaningfully compare the behavior vectors.

A second complication related to the convolutional process is the relative dullness of the majority of image sections where convolutional features are applied. For instance, on the QMNIST domain, almost 20% of image sections contain nearly zero variance. Not only will DDFA have difficulty discovering features that discriminate well across these uninteresting image sections, but intuitively it would seem that any features that can discriminate on such image sections would be arbitrary in nature.

To address the problem of uninteresting image sections, a simple heuristic is applied where image sections in the data set are discarded if their variance is below an arbitrary threshold, thereby focusing search on only those portions of the image where a significant amount of variance is

observed. In this way, novelty search can focus its efforts on the more interesting parts of the image. This heuristic, while primarily designed to address the issue of dull image sections, also helps address the previously discussed computational problem by reducing the effective number of behavior vector components generated for each image.

Finally, due to the increase in complexity that convolutional features present, the total number of features selected for composition into a neural network are reduced in this chapter as compared to the fully-connected features from the previous chapters.

STL-10 Domain

To assess the ability of DDFA to produce convolutional features that serve as an effective starting point, DDFA is applied first to the QMNIST domain that was introduced in the previous chapter. In addition, DDFA is applied to a second domain that is more challenging than QMNIST and was designed for algorithms that perform unsupervised pre-training such as DDFA.

STL-10 is a subset of the ImageNet domain, containing 10 classes of color images in the training and test sets. Each class contains only 500 images in the training set and 800 images in the test set. Additionally, $100{,}000$ unlabeled images that are drawn from a wider distribution of related classes are included. For example, the training and test set contain classes for airplanes, ships, and trucks, while the unlabeled set also contains images of trains and buses. Each image is 96×96 pixels with three color channels (red, green, blue). This relatively high-dimensional input space was designed to make the benchmark more challenging to encourage the development of scalable unsupervised feature learning methods.

This domain was chosen to test DDFA as the relative sparsity of training data and abundance of unlabeled data provides a good benchmark to test whether the unsupervised feature discovery pro-

cess of DDFA is generating features that are effective at describing the domain. In experiments in this chapter, the images are converted to grayscale prior to both feature discovery and training through gradient descent. This conversion disentagles the question of whether DDFA can effectively discover features in a semisupervised domain from the issues that arise from applying this new algorithm to a multi-channel input space. Examples of the images found in the STL-10 data set are shown in figure 6.1.

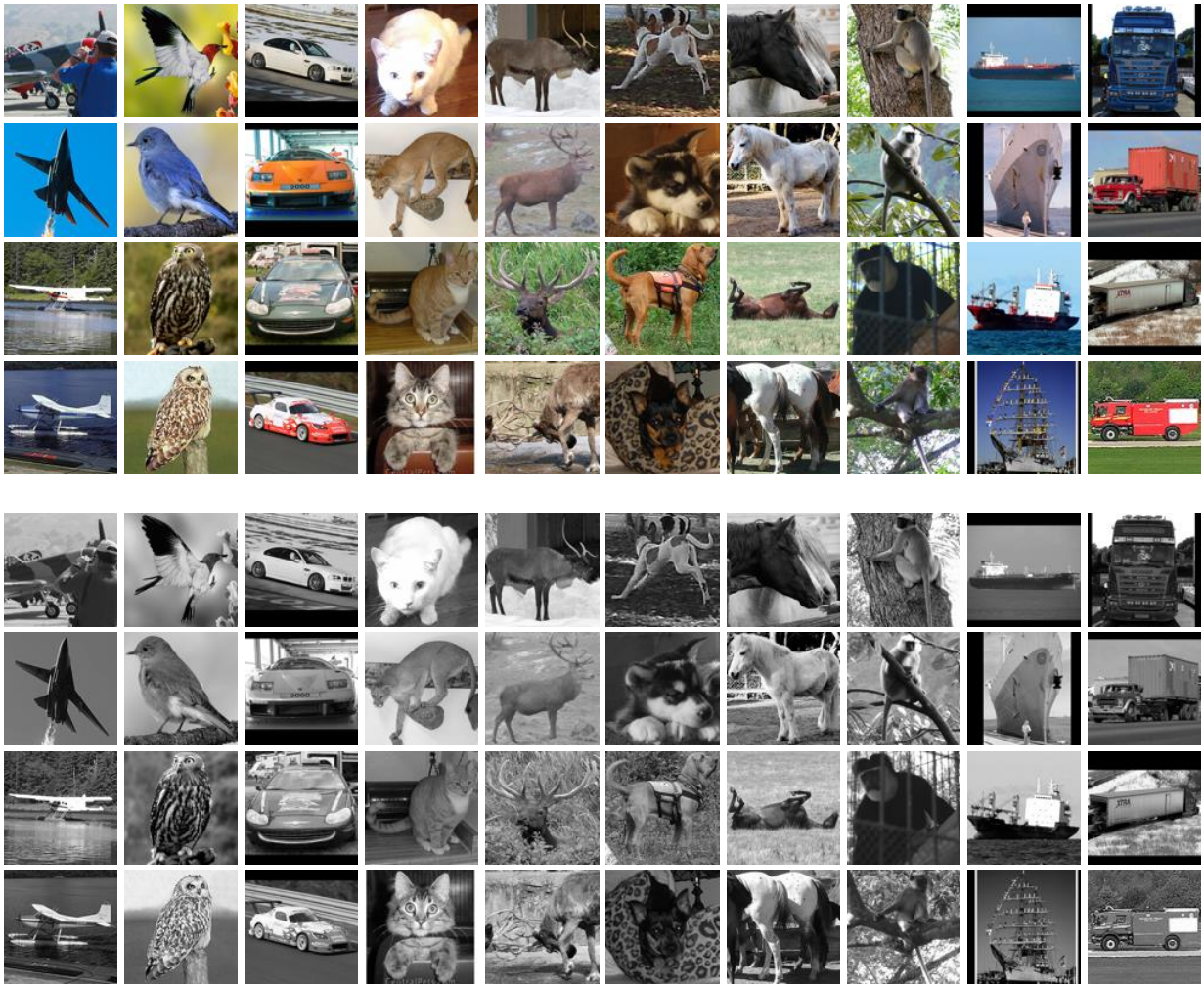


Figure 6.1: **Examples of training images from the STL-10 data set.** STL-10 is taken from ImageNet and contains 10 classes of vehicles and animals in the training and testing sets. The images are 96×96 pixels with red, green, and blue color channels. Color images are shown above, while their grayscale conversion, which is used by DDFA, are shown below.

Experiment

Convolution has been a critical component of cutting-edge neural networks across a variety of domains. The primary goal of this chapter is to evaluate the ability of DDFA to generate convolutional features that serve as a good starting point for gradient descent, allowing DDFA to contribute to the further improvement of these cutting-edge systems.

As with work in previous chapters, the choice was made to apply DDFA first to a relatively simple network architecture so as to avoid complications that can arise from attempting to apply a new technique immediately to a deeper architecture. This choice focuses attention on the ability of DDFA to produce quality convolutional features in the first layer. Therefore, the final classifier neural network has four layers: a convolutional layer with its associated pooling layer, a layer that is fully connected to the pooling layer, and the final output layer with ten units, which is fully connected to the previous fully-connected layer.

The convolutional feature size was chosen based on both the most commonly observed filter size from other works as well as preliminary tests [48, 57, 93, 113]. The convolutional feature size is 5×5 for QMNIST and 9×9 for STL-10, with a stride of 1 for QMNIST and 3 for STL-10. The images were padded, resulting in feature maps of size 28×28 for QMNIST and 32×32 for STL-10. The number of convolutional features in the convolutional layer and features in the fully-connected layer are chosen based on the point where expanding the network provides no significant improvement for gradient descent alone. For the QMNIST domain, the convolutional layer consists of 128 features and the fully connected layer consists of 256 neurons. For STL-10, the convolutional layer has 256 features and the fully-connected layer has 512 neurons. The pooling layer during gradient descent is 2×2 for QMNIST and 4×4 for STL-10. All six distance functions tested in the previous chapter are included here. The population size remains at 144 and, due to computational limitations, all runs were executed for only 400 generations. The variance

cutoff, which removes image sections with relatively little variance, was set for both domain so that only the top 10% of image sections, as measured by variance, is included in the data set.

Because the number of features selected to form the final network is greatly reduced from previous experiments due to the increase in complexity introduced by convolutional features, both in the size of the behavior vector and the size of the final network, more care must be taken in this selection process. First, the tournament size is increased to 128 to improve the odds of finding features that are suitably distant from each other. One complication in the feature selection process arises due to the fact that the first feature is selected entirely at random, which causes the selection of the subsequent features to be somewhat arbitrary, though this effect should diminish as the set of selected features grows. This complication could potentially lead to a significant fraction of the network containing unhelpful features. To address this issue, the selection algorithm was modified to include a pruning process where less novel features are pruned and new features are inserted. This process is described by Algorithm 3. In experiments in this work, $pruneCount$ is set to 32.

Algorithm 3 Convolutional DDFA Selection Algorithm

S = select 1 feature at random
while $count(S) < fCount$ **do**
    T = select $tsize$ features from the collection
    for all $t \in \mathcal{T}$ **do**
        $t_{min} \leftarrow$ find nearest neighbor in S
    end for
    S.add(select t where t_{min} = max(T_{min}))
    if $count(selectedFeatures) == fCount \& pruned < pruneCount$ **then**
        calculate novelty of all features in S
        S.remove(lowestNovelty)
        $pruned \leftarrow pruned + 1$
    end if
end while

Once the required number of features has been selected, they are composed into a network that is again trained with PyTorch. The PyTorch hyperparameters for experiments in this chapter are identical to the hyperparameters used in all experiments in the previous chapter. The testing protocol for QMNIST is identical to the previous chapter, while the protocol for STL-10 requires ten-fold cross validation. For the experiments on the applicability of DDFA to convolutional features, ten runs were executed for each distance metric, with an additional set of ten runs each on randomly initialized networks and networks initialized by randomly generated CPPNs.

An additional experiment is performed on the STL-10 domain to answer the question of whether DDFA has the ability to take advantage of the abundance of available unlabeled training data. To address this question, an experiment is performed where DDFA is applied to only the labeled train-

ing data. The parameters of this experiment match the initial experiment on the STL-10 domain, with fractional norm selected as the distance function. By comparing results of this experiment to results from the initial experiment on STL-10, where DDFA is applied to the larger set of unlabeled data, we can determine whether DDFA is effectively utilizing the availability of a large set of unlabeled data. This is an important experiment because one of the primary goals of an unsupervised training algorithm such as DDFA is the opportunity to apply it to domains where labeled data is limited. A successful exploitation of unlabeled data would therefore allow deep learning to be applied to domains that are currently out of reach due to the unavailability of sufficient labeled data.

Recall that from the results in Chapter 4, it was hinted that DDFA's ability to produce a diverse set of features may allow it to continue improving the performance of the final classifier even as the number of features grows beyond the point where gradient descent alone no longer sees an improvement from additional features. To test this hypothesis, a final experiment is performed on the STL-10 domain with a variable number of convolutional features. In each test, the size of the fully-connected hidden layer is set to $2n$ where n is the number of convolutional features. All parameters except for the network topology match the initial experiment on STL-10. Due to computational considerations in selecting a large number of features, cosine distance was selected as the distance function for this experiment. While fractional norm has been shown to perform the best in this domain, computers are less efficient at calculating roots than other operations, leading to a difference of about $3x$ in selection time between cosine and fractional norm distance. This difference becomes an important consideration when selecting a large number of features. For example, selecting $1,024$ convolutional features with cosine distance takes about 48 hours with 8 cores. Because fractional norm provides a better measure of distance in high dimensions than cosine distance, if cosine distance can effectively be applied to wider networks, it would be expected that fractional norm distance could be as well. If the hypothesis that DDFA can effectively

utilize a larger number of features than gradient descent alone is true, then this could add a new tool in the search for increasingly complex neural networks. While much of deep learning community has focused its efforts on increasing network complexity through an increase in network depth, DDFA may provide an additional tool for increasing network complexity through the expansion of network width.

Results

This section includes an analysis of the results from the four experiments detailed in the previous section. The first two experiments are designed to analyze DDFA's ability to discover effective convolutional features, while the third and fourth experiments address specific questions related to the applicability of DDFA to unsupervised data and wide networks.

QMNIST

Results of the experiment to apply DDFA to convolutional features in the QMNIST domain are shown in Figure 6.2. Overall, the results with a convolutional network follow the same general pattern as with a fully-connected network. The random CPPN control is significantly better ($p < 0.001$) than the randomly initialized control and all DDFA distance functions are significantly better ($p < 0.01$) than the random CPPN control. The grouping of distance functions has changed, with only fractional norm distance showing a significant improvement ($p < 0.05$) over Euclidean distance, and no distance function is significantly worse than Euclidean.

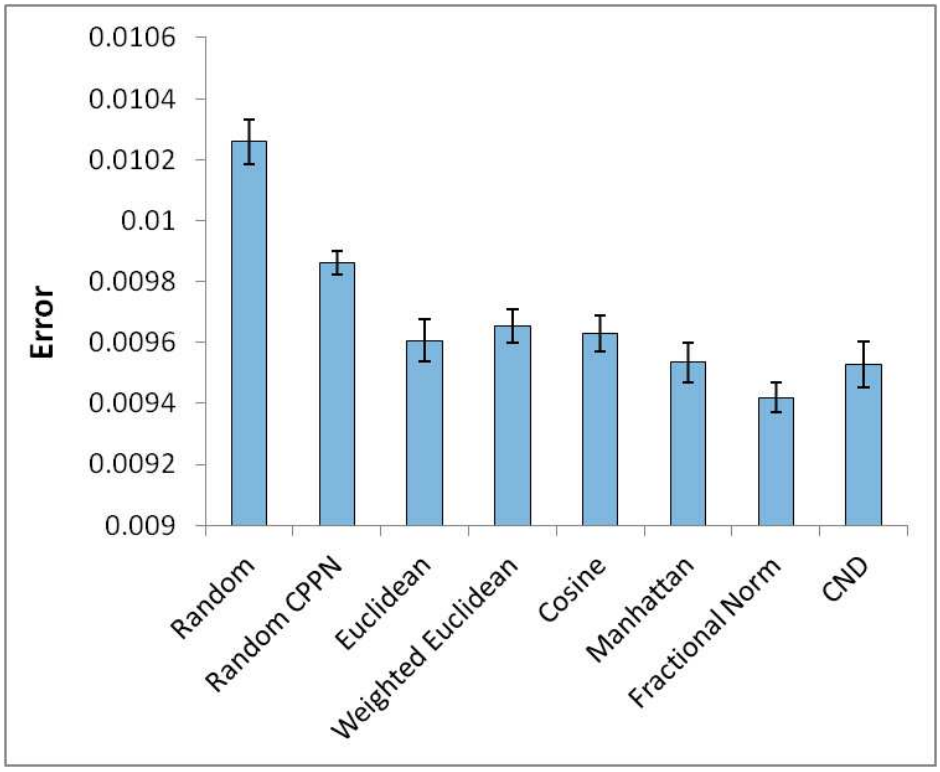


Figure 6.2: **Average testing error on the QMNIST test set by distance function with convolutional features.** The average testing error (over 10 runs) on the QMNIST test set is shown for each distance function, and error bars represent standard error. Two controls are included where the weights are initialized randomly and where the weights are initialized by random CPPNs. The primary result is that DDFA can discover a set of effective convolutional features that when used as a starting point for gradient descent, lead to improved performance as measured by performance on the QMNIST test set. In addition, the fractional norm distance function performed significantly better than Euclidean distance, adding further evidence that it is a better choice in distance function.

These results add empirical evidence to support the conclusions from the previous chapter. The most effective distance function, fractional norm distance, was originally selected for its ability

to give a useful measure of distance in high dimensions. More importantly, the results here confirm that DDFA can effectively discover convolutional features that form an effective basis for a convolutional neural network, as measured by test performance.

STL-10

Figure 6.3 displays the results of the experiment to apply DDFA to convolutional features in the STL-10 domain. These results mirror the results on the QMNIST domain, with the random CPPN control significantly ($p < 0.01$) outperforming the randomly initialized control and all DDFA distance functions significantly ($p < 0.01$) outperforming the random CPPN control. Furthermore, the grouping of distance functions remain the same, with only fractional norm distance showing a significant improvement ($p < 0.05$) over Euclidean distance, and no distance function function performing significantly worse than Euclidean.

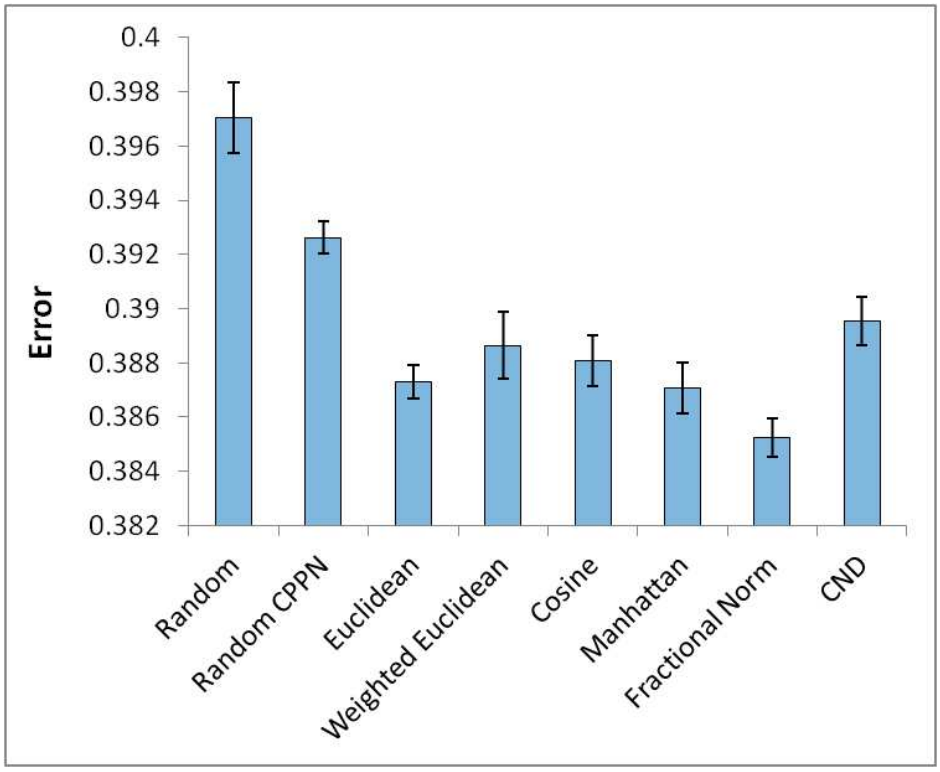


Figure 6.3: **Average testing error on the STL-10 test set by distance function with convolutional features.** The average testing error (over 10 runs) on the STL-10 test set is shown for each distance function, and error bars represent standard error. Two controls are included where the weights are initialized randomly and where the weights are initialized by random CPPNs. The primary result is confirmation that DDFA can produce an effective set of convolutional features. In addition, the fractional norm distance function performed significantly better than Euclidean distance, adding further evidence that it is a better choice in distance function.

To qualitatively evaluate the convolutional features produced by DDFA, Figures 6.4 and 6.5 provide a visualization for the entire set of a features selected by a single run of DDFA on the STL-10 domain with the Euclidean and fractional norm distance functions. Each small image section repre-

sents a single convolutional feature, with green and red representing connections with positive and negative weights, respectively, where higher brightness representing higher magnitude of weight. The gray areas of the features indicate a weight of zero, meaning the connection was removed by HyperNEAT-LEO, which was introduced in Chapter 4. As a whole, these feature sets show that DDFA discovers a diverse set of edge detectors of various orientations, widths, and intensities. A comparison of the two sets of features provides further insight into the performance of the two distance metrics. The features produced with the fractional norm distance metric appear smoother, with fewer extreme values, and contain fewer zero-valued weights. While the runs with Euclidean distance produced an average of approximately 34% non-zero weights, the runs with fractional norm produced an average of approximately 42% non-zero weights. Furthermore, if an arbitrary measure is defined where a feature is considered interesting if it contains at least 3 positive weights and 3 negative weights or at least 9 non-zero weights overall, then the set of features produced by Euclidean distance contains about 78% interesting features while fractional norm contains 90% interesting features. These results imply that one reason Euclidean distance is less effective is due to its tendency to produce features that focus on smaller sections of the input space rather than larger structures.

The results from this section add additional empirical evidence to support the conclusions from the previous chapter, as well as supporting the hypothesis that DDFA can effectively discover convolutional features for application to convolutional neural networks. Due to the widespread success of convolutional neural networks across a wide range of domains, the successful application of DDFA to the discovery of convolutional features represents a critical step in the future application of DDFA to state-of-the-art deep neural networks. Furthermore, this result shows that DDFA has some flexibility in the ways that it can be applied, and is not restricted to a single type of feature or architecture.

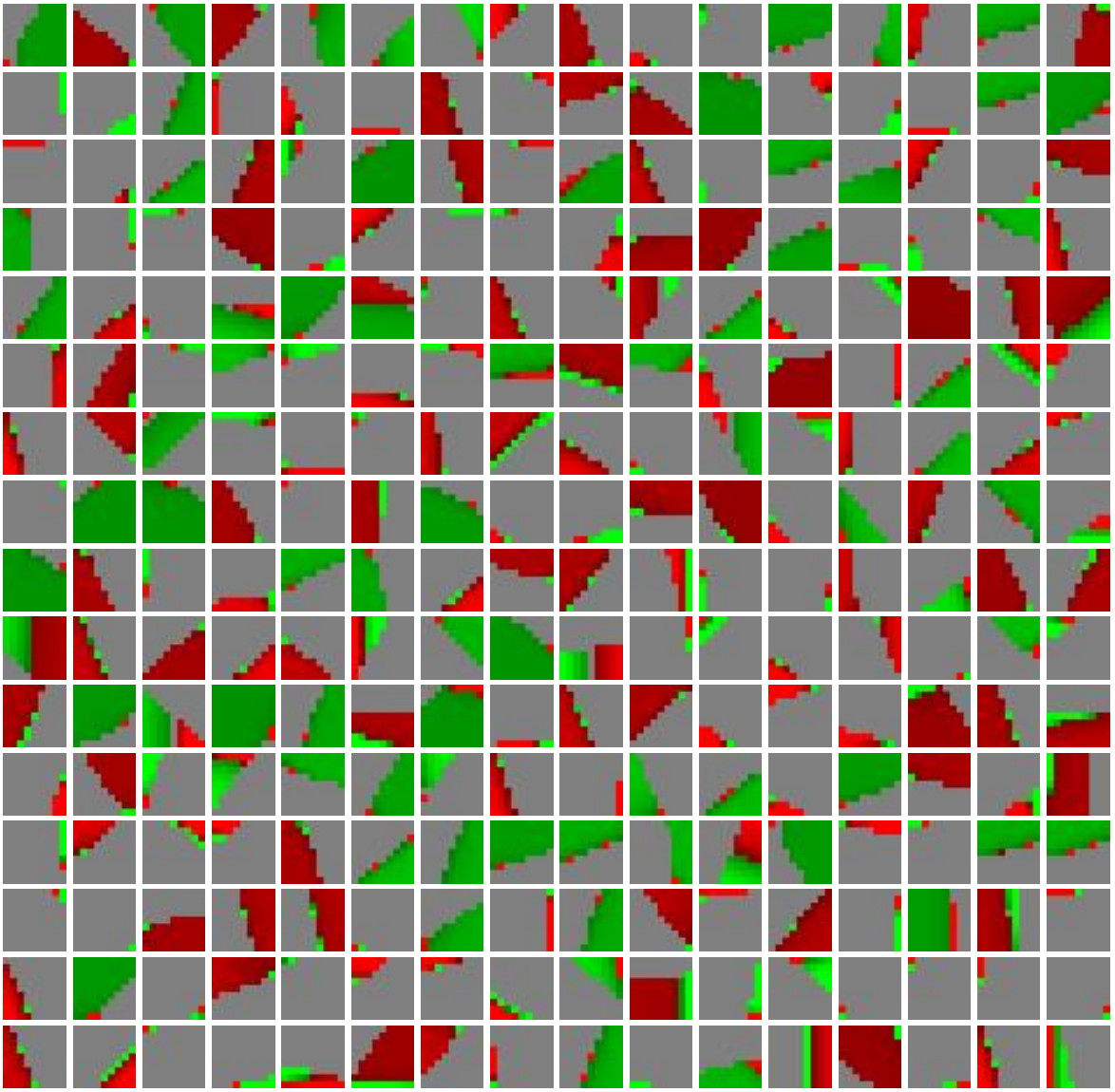


Figure 6.4: **Convolutional features discovered by DDFA on the STL-10 domain with Euclidean distance.** The full feature set of a single run with the Euclidean distance function on the STL-10 domain is depicted. Each small image represents a single convolutional feature, with green and red representing connections with positive and negative weights, respectively, with higher brightness representing higher magnitude of weight. Gray represents areas where weights are zero. The feature set contains edge detectors with a variety of orientations and intensities.

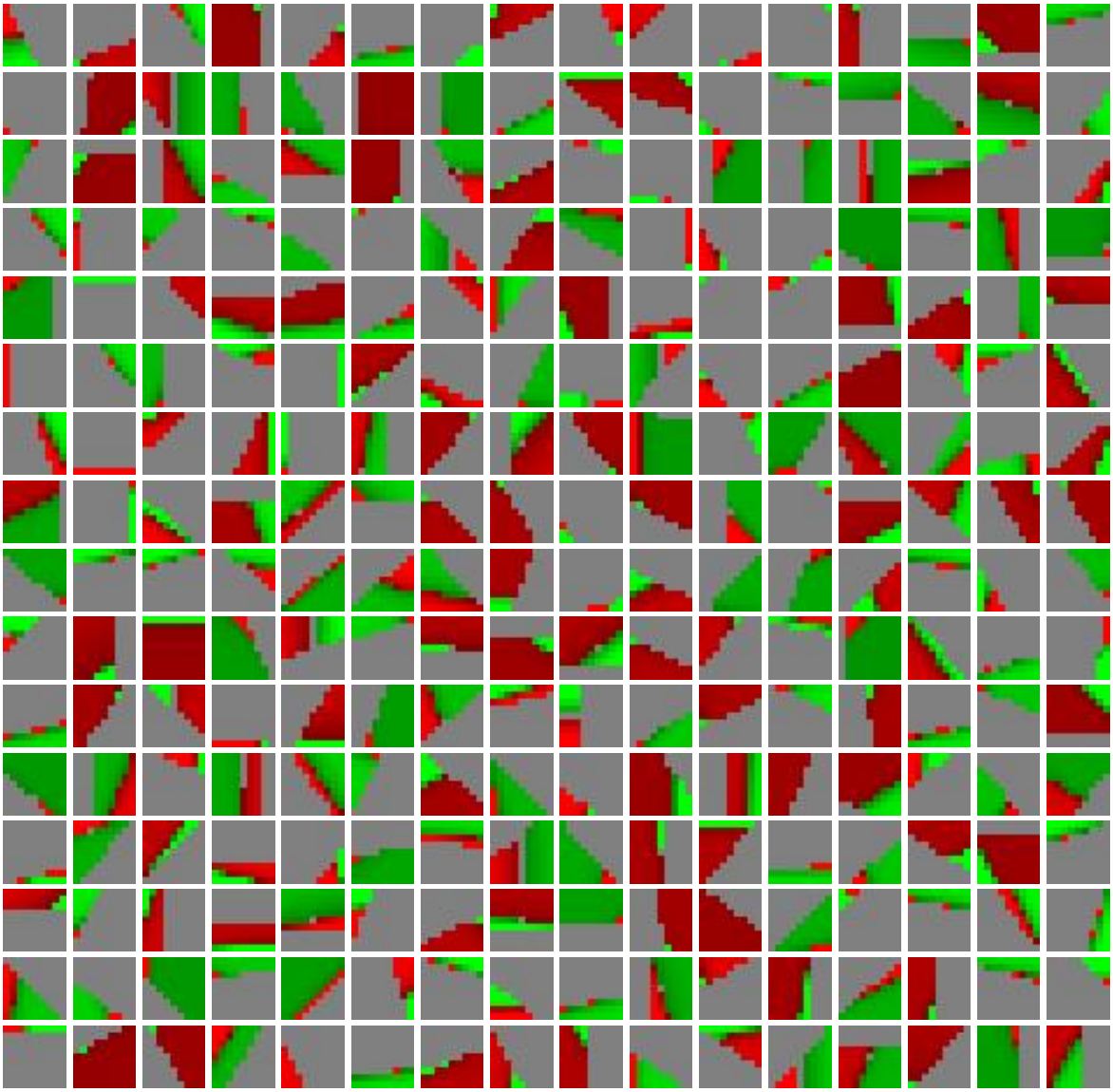


Figure 6.5: **Convolutional features discovered by DDFA on the STL-10 domain with fractional norm distance.** The full feature set of a single run with the fractional norm distance function is depicted. Each small image represents a single convolutional feature, with green, red, and gray representing connections with positive, negative, and zero weights, respectively, with higher brightness representing higher magnitude of weight. The features are similar to those discovered with the Euclidean distance metric, but they appear smoother and contain less area with zero weight.

Effective Utilization of Unlabeled Data

Results of the experiment to determine whether DDFA effectively utilizes large amounts of unlabeled training data are shown in Figure 6.6. These results show a significant improvement ($p < 0.05$) when unlabeled data is included in the feature discovery process, indicating that DDFA utilizes this additional data. Another interesting observation is that even without the unlabeled data, DDFA significantly ($p < 0.001$) outperforms gradient descent alone. This result raises the possibility that DDFA could serve as a tool in the application of deep learning to a wider range of domains where abundant labeled data is unavailable, whether unlabeled data is present or not.

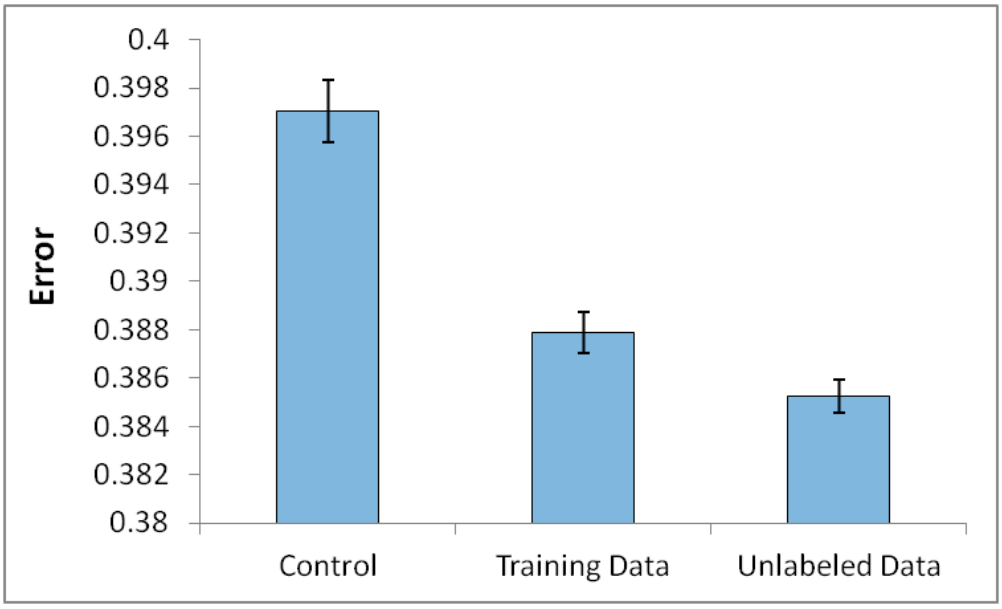


Figure 6.6: **Average testing error on the STL-10 test set with unlabeled data and training data.** The average testing error (over 10 runs) on the STL-10 test set is shown where DDFA is supplied with only the training set and where it is supplied with the larger unlabeled data set, as well as the control where weights are randomly initialized. Error bars represent standard error. This result shows that DDFA effectively utilizes the unlabeled data to form a better set of features as measured by the performance of gradient descent when it is applied to these features.

Network Architecture and Wide Networks

Figure 6.7 shows results from the experiment to determine the utility of DDFA across a range of network size. While gradient descent fails to realize any significant improvement after a first layer size of 256 convolutional features, DDFA continues to see significant improvement ($p < 0.05$) up

to 512 convolutional features. This result implies that DDFA may permit wider architectures than are typically employed in deep learning.

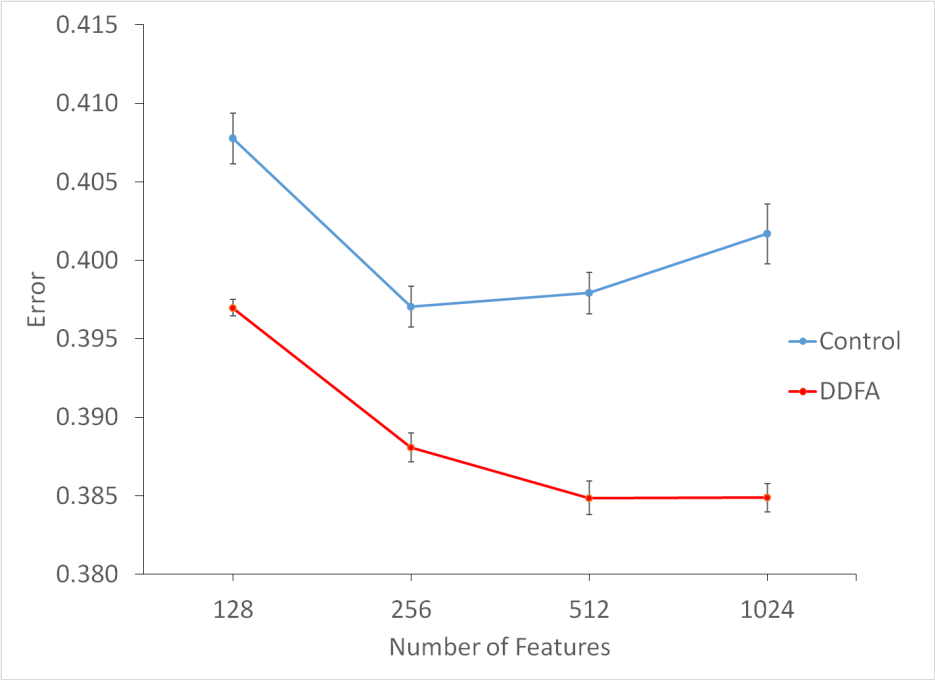


Figure 6.7: **Average testing error on the STL-10 test set by number of convolutional features.** The average testing error (over 10 runs) on the STL-10 test set is shown across a range of convolutional layer sizes for randomly initialized features and features initialized by DDFA, and error bars represent standard error. As the number of convolutional features increases beyond 256, gradient descent alone fails to realize any improvement, while DDFA achieves its best performance at 512 features.

Synthesis

The results of this chapter provide evidence to support the hypothesis that DDFA can be applied to the discovery of convolutional features that can form the basis for a convolutional neural network. DDFA was able to successfully improve performance of convolutional neural networks on both the QMNIST and STL-10 domains, suggesting that DDFA can be applied to a wider range of image recognition domains.

Furthermore, an experiment on the STL-10 domain provides evidence that DDFA can exploit unlabeled data to further improve the set of features that it produces. This result is important because it implies that not only can DDFA be applied to aid gradient descent on domains where gradient descent alone already performs well, but that DDFA can further the application of gradient descent and deep learning to domains where training data is too limited for gradient descent to be successfully applied on its own. Many real-world problems are of this type, where labeled training data is expensive to obtain or otherwise unavailable. While approaches to unsupervised pretraining have often focused on generative models, such as RBMs [11, 50] and GANs [44], DDFA provides a discriminative model for unsupervised pretraining. DDFA therefore provides an example of how neuroevolution can serve a role in expanding the reach of deep learning into a wide array of real-world applications.

Finally, an additional experiment on the STL-10 domain provides evidence that DDFA may allow wider networks to be trained than gradient descent alone. The diversity of the feature set seems to permit gradient descent to more effectively generalize with a wider network when the features are initialized by DDFA. This result has important implications for deep learning, where the focus has been primarily on the increase in depth, which has its own associated challenges and pitfalls. Tools to increase network width, such as DDFA, may therefore provide a new avenue to increasing network complexity that is complimentary with the increase in depth.

CHAPTER 7: DISCUSSION

This dissertation as a whole has focused on the development of algorithms that combine neuroevolution and gradient descent for the purpose of training neural networks.

While LEEA shows that evolutionary algorithms can effectively train neural networks of moderate complexity, the more important distinction is in the way that it achieves that goal. By reducing the number of training examples required to evaluate an individual, LEEA provides an analog to stochastic gradient descent (SGD) within neuroevolution that allows the two to be combined in ways that were previously impossible due to computational considerations. For example, suppose a researcher wants to create an algorithm that includes neuroevolution for topological search with gradient descent for learning the weights of each individual in the population, and they want these phases to alternate so that weights are learned at the same time as topology. Without LEEA, the neuroevolution phase would include evaluating individuals against the entire training set. Meanwhile, the gradient descent phase could evaluate a single training example, such as with SGD, or a small number of training examples with mini-batching. Either way, the limiting factor in the ability of the algorithm to execute many generations is the neuroevolution phase that must evaluate against the entire training set. With LEEA, the time taken by the neuroevolution phase can be greatly reduced, enabling the algorithm to execute many more generations than it could previously. This capability allows for a much tighter coupling of the neuroevolution and gradient descent components than was previously possible.

The most extensive work in this dissertation is the development of a novel hybrid algorithm (DDFA) that combines some of the more sophisticated tools of neuroevolution with the power of a convolutional neural network trained through stochastic gradient descent. DDFA was first applied to a relatively simple benchmark visual problem, MNIST, where gradient descent already performs

well, and with a relatively simple network architecture. This simplified first application of DDFA provided a useful tool as the difficulties in applying a new and relatively complex neuroevolution approach were resolved. This approach then allowed the initial development of DDFA to focus on the questions that arise from the application of neuroevolution to the discovery of features without the complications that arise from more extensive data sets and complex neural network topologies. This initial work showed that even on a domain where gradient descent performs well on its own, DDFA is effective at providing a set of starting features that improve performance as measured by the ability of the network to generalize to the test set.

One of the more complicated issues that arose from DDFA was the distance function that provides novelty search with an explicit measure of the novelty of an individual feature. The traditional choice for distance function in novelty search, Euclidean distance, was discovered to be faulty for application to the large behavior vectors produced by DDFA. A study of alternate distance functions produced two new distance measures, fractional norm distance and component neighbor distance, that each provide significant improvements over Euclidean distance. Component behavior distance provides a much less computationally burdensome distance metric, which aids in the application of DDFA to larger domains and larger networks. Fractional norm distance was found to provide a better sense of distance in high dimensions, which leads to the discovery of a better set of features. This discovery has implications beyond future applications of DDFA. The neuroevolution community as a whole is often working with behavior vectors with high dimensions, and the discovery that fractional norm distance can provide a better measure of distance in this space may aid researchers as they apply novelty search.

With the fundamentals of DDFA established, DDFA was applied to the discovery of convolutional features. As convolution has become a critical tool for applying neural networks to a wide array of more challenging domains, this was an important step in the development of DDFA. The basic approach to applying DDFA to convolutional features is fairly simple. Rather than a HyperNEAT

substrate that connects the entire input space to a single output, as with fully-connected features, the HyperNEAT substrate connects a small patch (5×5 or 9×9) to a single output. This substrate is then convolved across the image, producing a feature map rather than a single output. While this process introduced new computational challenges, this work provided solutions to these challenges, resulting in convolution being successfully applied to the MNIST domain. In addition to showing that DDFA was effective at discovering convolutional features in the MNIST domain, this work showed how DDFA could be applied to the more challenging STL-10 domain, which contains relatively high resolution images of real-world objects. The successful application of DDFA to STL-10 shows that it could be applied to a potentially broader range of problems.

One of the main challenges to applying deep learning is that it requires a large set of labeled data to learn from. One of the most important aspects of DDFA is that feature discovery is done in an unsupervised way, which means training data need not be labeled for DDFA to extract features from it. This allows DDFA to be applied to any set of data, regardless of whether it is labeled. An experiment on the STL-10 domain, which provides a large set of unlabeled data along with a small set of labeled data, showed that not only does DDFA provide an improvement over gradient descent when applied to only the labeled training data, but that it further improves performance when applied to the unlabeled data. These results show that DDFA provides an avenue to expand the reach of deep learning to domains where sufficient labeled training data is unavailable.

Because DDFA produces a set of features that discriminate across the feature set in novel ways, they could have other interesting properties when applied to neural networks. One of these properties that was identified in this work is the ability for gradient descent to effectively exploit a larger set of DDFA features than it can with randomly generated features. While deep learning has primarily focused on increasing network complexity through the increase in depth, this discovery provides a tool for increasing network complexity through width as well.

While the experiments in this dissertation applied DDFA to discovering features in the first layer only, there is no obvious reason that the principle of features that discriminate in novel ways could not be applied beyond the first layer. While such an extension of DDFA to deeper layers could provide an important new tool for deep learning, it does come with its own set of challenges. First, because features beyond the first layer are compositions of features in the previous layer, those conglomerations of features do not have a geometric relationship with each other in the same way that the input space does (though each individual feature map found through convolution of course does have its own geometry). This difference makes the application of CPPNs, which exploit the problem geometry, more complicated. One potential solution to this problem is to eschew geometry in these deeper layers, at least between different features, and instead encode individuals with a direct encoding like NEAT. NEAT could even decide which of the prior layer features to include, based perhaps on the concept of co-occurrence. For example, if a round feature is observed at a location frequently with a smaller round feature inside it, that observation may indicate the presence of a higher-level feature such as an eye. Note that the loss of geometry does not undermine the search for features that discriminate in novel ways driven by novelty search. Therefore, DDFA can still retain much of its power even as the search for features is applied to successive layers. Additionally, because the feature maps produced by convolutional layers do contain geometric information, HyperNEAT could still be applied to the convolutional layers of a deep network. The application of DDFA across all layers of a deep network presents challenges, but if successful, it could have a tremendous impact on the application of deep learning on both tasks where deep learning already performs well, and on tasks where deep learning cannot be applied currently due to limitations of labeled data.

It is important to recognize that significantly more than performance is at stake in the dissemination of alternative unsupervised training techniques based on new principles. Deep learning faces several fundamental challenges that are not only about testing performance. For example, results from

Szegedy et al. [112] show that very small yet anomalous perturbations of training images that are imperceptible to the human eye can fool several different kinds of deep networks that nevertheless ominously score well on the test set. The implications of these anomalies are not yet understood. At the same time, as Bengio [9] points out, local descent on its own will not ultimately be enough to tackle the most challenging problems, suggesting the need for radical new kinds of optimization that are more global. These kinds of considerations suggest that simply scoring well on a test set in the short run may not necessarily foreshadow continuing success for the field in the long run, highlighting the need for new approaches.

By showing that unsupervised discriminative learning can be effective, DDFA brings several intriguing corollaries. Among those, it is possible to conceive training methods that act as continual feature accumulators that do not require a fixed "hidden layer size." Furthermore, it is possible to learn useful features without any kind of error minimization (which is even used in conventional unsupervised techniques). Relatedly, an interesting question is whether anomalous results are sometimes a side effect of the very idea that all useful knowledge ultimately must come from minimizing error. The divergent dynamics of novelty search also mean that the search is inherently more global than local descent for the very reason that it is continually diverging, thereby offering a hint of how more expansive feature sets can be collected.

The importance of the capacity to learn from unlabeled data cannot be understated. Unsupervised learning is widespread in the process of human learning, as vast quantities of labeled data are rarely available. As infants, we observe relationships in our sensory input, such as shapes and colors, and automatically classify objects based on their similarity, developing a kind of mental language referred to as *Mentalese* by Pinker [88]. Only later do we then learn the spoken language labels for these mental categories that were developed without any explicitly defined labels. This capacity is why children do not need to be told the correct label for thousands of pictures of cats. Once they are shown a small number of cats, they automatically understand the concept of a cat

due to the unsupervised learning that has taken place since the moment they opened their eyes. They can then quickly attach a label based on the language they are taught. In this way, DDDFA introduces a process that is much more aligned with how children learn to classify objects.

Finally, this work should be viewed at a higher level as more than only a new algorithm that provides a better solution for unsupervised pretraining. It also aims to serve as an example and help open the door to other novel hybrid algorithms containing neuroevolution and gradient descent. The majority of work done on such hybrid algorithms has utilized only the simplest tools from the neuroevolution toolkit, often implementing simple neuroevolution algorithms that have been around for decades. Major neuroevolution advances of the past decade, such as indirect encodings and novelty search, have not yet been fully exploited in hybrid algorithms, leading these algorithms to be less effective than they possibly could be. Therefore this dissertation shows that these tools can serve a critical role, encouraging neural network researchers to fully exploit the rich set of tools provided by neuroevolution.

CHAPTER 8: CONCLUSION

This dissertation introduced two new neural network training algorithms, the Limited Evaluation Evolutionary Algorithm (LEEA) and Divergent Discriminative Feature Accumulation (DDFA). These two new algorithms were conceived to promote the integration of gradient descent and neuroevolution in hybrid neural network training algorithms, and they each achieve this goal in a different way. While LEEA introduces a variation on the standard evolutionary algorithm that permits the integration with stochastic gradient descent, DDFA gives an example of an effective integration of the two neural network training paradigms.

Four primary contributions summarize the overall achievement of this work:

(1) LEEA provides a modification to the standard evolutionary algorithm that allows individuals to be evaluated against only a small batch of training examples in each generation rather than the entire training set. This limited evaluation, which is made possible by a process called fitness inheritance, permits EAs to behave in a similar manner to mini-batch learning in stochastic gradient descent, which is a critical component of deep learning. While LEEA shows that evolution alone can effectively train moderately complex neural networks, it is more important to observe that by introducing an analog to mini-batch learning within neuroevolution, LEEA has created new opportunities for the development of new types of hybrid algorithms where neuroevolution and gradient descent are more tightly coupled than before.

(2) DDFA is an example of a hybrid gradient-neuroevolution algorithm that incorporates two of the more sophisticated tools available to neuroevolution, indirect encoding and novelty search. This contribution was further augmented by (3) investigating the impact of the distance function on the quality of the features generated by DDFA and (4) applying DDFA to the discovery of convolutional features. Not only was DDFA shown experimentally to provide starting features

that achieve better results on supervised tasks where gradient descent already does well, but it was also shown experimentally that the unsupervised nature of the feature discovery process allows the algorithm to be applied on domains where labeled data is less abundant. This capability could potentially expand the reach of deep learning to a wider range of domains in the future

Taken as a whole, these contributions serve as an example of how recent advances in neuroevolution can be leveraged to augment deep learning and paves the way for expanding the reach of algorithms that exploit the power of both neuroevolution and deep learning.

LIST OF REFERENCES

[1] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.

[2] D. Almeida and N. Sauder. GradNets: Dynamic interpolation between neural architectures. *ArXiv e-prints*, abs/1511.06827, 2015.

[3] 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.

[4] T. Bäck, D. B. Fogel, and Z. Michalewicz. *Handbook of evolutionary computation*. CRC Press, 1997.

[5] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *ArXiv e-prints*, abs/1611.02167, 2016. URL `http://arxiv.org/abs/1611.02167`.

[6] R. K. Belew, J. McInerney, and N. N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In *Artificial Life II: Proceedings of the Workshop on Artificial Life*, 1990.

[7] R. Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[8] Y. Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[9] Y. Bengio. Deep learning of representations: Looking forward. In *Statistical Language and Speech Processing*, pages 1–37. Springer, 2013.

[10] Y. Bengio and Y. LeCun. Scaling learning algorithms towards ai. *Large-Scale Kernel Machines*, 34, 2007.

[11] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19 (NIPS)*, Cambridge, MA, 2007. MIT Press.

[12] I. Bertini, M. De Felice, and S. Pizzuti. Combining back-propagation and genetic algorithms to train neural networks for start-up time modeling in combined cycle power plants. In *Proceedings of the European Symposium on Artificial Neural Networks (ESANN)*, 2010.

[13] H. Braun and J. Weisbrod. Evolving neural feedforward networks. In *Artificial Neural Nets and Genetic Algorithms*, pages 25–32. Springer, 1993.

[14] L. Chen and K. Aihara. Chaotic simulated annealing by a neural network model with transient chaos. *Neural networks*, 8(6):915–930, 1995.

[15] D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333–338, 2012.

[16] A. Coates, A. Y. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011.

[17] E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. O. Stanley, and J. Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *arXiv preprint arXiv:1712.06560*, 2017.

[18] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan. Better mini-batch algorithms via accelerated gradient methods. In *Advances in neural information processing systems*, pages 1647–1655, 2011.

[19] A. Cully, J. Clune, D. Tarapore, and J.-B. Mouret. Robots that can adapt like animals. *Nature*, 521:503–507, 2015. doi: 10.1038/nature14422.

[20] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

[21] D. Dasgupta and D. McGregor. Designing application-specific neural networks using the structured genetic algorithm. In *Proceedings of the International Conference on Combinations of Genetic Algorithms and Neural Networks*, pages 87–96, 1992.

[22] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[23] Y. Dauphin, R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *ArXiv e-prints*, abs/1406.2572, 2014.

[24] K. A. De Jong. *Evolutionary Computation: A Unified Perspective*. MIT Press, Cambridge, MA, 2002.

[25] 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.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[27] T. Desell. Large scale evolution of convolutional neural networks using volunteer computing. *ArXiv e-prints*, abs/1703.05422, 2017. URL `http://arxiv.org/abs/1703.05422`.

[28] P. M. Domingos. A few useful things to know about machine learning. *Commun. acm*, 55 (10):78–87, 2012.

[29] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE, 1995.

[30] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[31] C. Fernando, D. Banarse, M. Reynolds, F. Besse, D. Pfau, M. Jaderberg, M. Lanctot, and D. Wierstra. Convolution by evolution. *ArXiv e-prints*, abs/1606.02580, 2016. URL `http://arxiv.org/abs/1606.02580`.

[32] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *ArXiv e-prints*, abs/1701.08734, 2017. URL `http://arxiv.org/abs/1701.08734`.

[33] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1:47–62, 2008.

[34] D. B. Fogel. *Blondie24: Playing at the Edge of AI*. 2001.

[35] L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial intelligence through simulated evolution*. John Wiley and Sons, Hoboken, NJ, 1966.

[36] L. G. Fonseca, A. C. Lemonge, and H. J. Barbosa. A study on fitness inheritance for enhanced efficiency in real-coded genetic algorithms. In *Proceedings of the 2012 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2012.

[37] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[38] J. Gauci and K. O. Stanley. Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007)*, New York, NY, 2007. ACM Press.

[39] J. Gauci and K. O. Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural Computation*, 22(7):1860–1898, 2010.

[40] C. Goerick and T. Rodemann. Evolution strategies: An alternative to gradient-based learning. In *Proceedings of the International Conference on Engineering Applications of Neural Networks*, volume 1, pages 479–482. Citeseer, 1996.

[41] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Hillsdale, NJ: Lawrence Erlbaum, 1987.

[42] J. Gomes, P. Urbano, and A. L. Christensen. Evolution of swarm robotics systems with novelty search. *Swarm Intelligence*, pages 1–30, 2013.

[43] F. Gomez and R. Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[44] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[45] C. Green. SharpNEAT homepage. `http://sharpneat.sourceforge.net/`, 2003–2015.

[46] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[47] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[48] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[49] D. O. Hebb. *The organization of behavior: A neuropsychological theory*. New York: Wiley, 1949.

[50] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

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

[52] J. H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[53] R. Houthooft, R. Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, and P. Abbeel. Evolved policy gradients. *arXiv preprint arXiv:1802.04821*, 2018.

[54] G.-B. Huang, P. Saratchandran, and N. Sundararajan. A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. *Neural Networks, IEEE Transactions on*, 16(1):57–67, 2005.

[55] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[56] C. Igel. Neuroevolution for reinforcement learning using evolution strategies. In R. Sarker, R. Reynolds, H. Abbass, K. C. Tan, B. McKay, D. Essam, and T. Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 2588–2595, Piscataway, NJ, 2003. IEEE Press.

[57] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[58] Q. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. Corrado, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning (ICML-2012)*, 2012.

[59] Y. LeCun. Une procédure d'apprentissage pour réseau à seuil assymétrique. In *Cognitiva 85: A la Frontière de l'Intelligence Artificielle des Sciences de la Connaissance des Neurosciences*, pages 599–604, Paris, 1985. CESTA.

[60] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

[61] Y. LeCun and C. Cortes. The MNIST database of handwritten digits, 1998.

[62] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[63] J. Lehman and K. O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223.

[64] J. Lehman, K. O. Stanley, and R. Mikkulainen. Effective diversity maintenance in deceptive domains. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2013)*, New York, NY, 2013. ACM Press.

[65] J. Lehman, J. Chen, J. Clune, and K. O. Stanley. Safe mutations for deep and recurrent neural networks through output gradients. *arXiv preprint arXiv:1712.06563*, 2017.

[66] B. Li and L. Han. Distance weighted cosine similarity measure for text classification. In *International Conference on Intelligent Data Engineering and Automated Learning*, pages 611–618. Springer, 2013.

[67] J. Liang, E. Meyerson, and R. Miikkulainen. Evolutionary architecture search for deep multitask networks. *arXiv preprint arXiv:1803.03745*, 2018.

[68] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. *Neural Networks, IEEE Transactions on*, 17(6):1411–1423, 2006.

[69] G. D. Magoulas, V. P. Plagianakos, and M. N. Vrahatis. Neural network-based colonoscopic diagnosis using on-line learning and differential evolution. *Applied Soft Computing*, 4(4): 369–379, 2004.

[70] M. Mandischer. A comparison of evolution strategies and backpropagation for neural network training. *Neurocomputing*, 42(1):87–117, 2002.

[71] R. Marc'Aurelio, L. Boureau, and Y. LeCun. Sparse feature learning for deep belief networks. In *Advances in Neural Information Processing Systems 20 (NIPS)*, pages 1185–1192, Cambridge, MA, 2007. MIT Press.

[72] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, and B. Hodjat. Evolving deep neural networks. *ArXiv e-prints*, abs/1703.00548, 2017. URL `http://arxiv.org/abs/1703.00548`.

[73] 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.

[74] M. Minsky and S. Papert. *Perceptrons*. MIT Press, 1969.

[75] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[76] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. pages 762–767, 1989.

[77] D. E. Moriarty and R. Miikkulainen. Forming neural networks through efficient and adaptive co-evolution. *Evolutionary Computation*, 5:373–399, 1997.

[78] G. Morse, S. Risi, C. R. Snyder, and K. O. Stanley. Single-unit pattern generators for quadruped locomotion. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2013)*, New York, NY, USA, 2013. ACM.

[79] J.-B. Mouret and S. Doncieux. Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary computation*, 20(1):91–133, 2012.

[80] H. Mühlenbein. Limitations of multi-layer perceptron networks – steps towards genetic neural networks. *Parallel Computing*, 14(3):249–260, 1990.

[81] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

[82] S.-K. Ng and G. J. McLachlan. Using the em algorithm to train neural networks: misconceptions and a new algorithm for multiclass classification. *IEEE transactions on neural networks*, 15(3):738–749, 2004.

[83] A. Nguyen, J. Yosinski, and J. Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. 2015.

[84] H. V. Nguyen and L. Bai. Cosine similarity metric learning for face verification. In *Asian conference on computer vision*, pages 709–720. Springer, 2010.

[85] V. P. Pallavi and V. Vaithiyanathan. Combined artificial neural network and genetic algorithm for cloud classification. *International Journal of Engineering and Technology*, 5(2): 787–794, 2013.

[86] D. B. Parker. Learning-logic. Invention Report S81-64, File 1, Office of Technology Licensing, Stanford University, Palo Alto, CA, 1985.

[87] F. Petroski Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[88] S. Pinker. *The language instinct: How the mind creates language*. Penguin UK, 2003.

[89] V. W. Porto, D. B. Fogel, and L. J. Fogel. Alternative neural network training methods. *IEEE Intelligent Systems*, (3):16–22, 1995.

[90] J. K. Pugh, L. B. Soros, and K. O. Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

[91] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. *ArXiv e-prints*, abs/1703.01041, 2017. URL http://arxiv.org/abs/1703.01041.

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

[93] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4780–4789, 2019.

[94] M. Requena-Prez, A. Albero-Ortiz, J. Monz-Cabrera, and A. Daz-Morcillo. Combined use of genetic algorithms and gradient descent optmization methods for accurate inverse permittivity measurement. *IEEE Transactions On Microwave Theory And Techniques*, 54 (2):615–624, 2006.

[95] S. Risi and J. Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(1):25–41, 2017.

[96] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958. URL http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=retrieve&db=pubmed&dopt=abstract&list_uids=13602029.

[97] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland. A general framework for parallel distributed processing. pages 45–76. 1986.

[98] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing*, pages 318–362. 1986.

[99] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2234–2242, 2016.

[100] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *ArXiv e-prints*, abs/1703.03864, 2017. URL `http://arxiv.org/abs/1703.03864`.

[101] F. Seide, G. Li, and D. Yu. Conversational speech transcription using context-dependent deep neural networks. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[102] M. J. Shafiee, E. Barshan, and A. Wong. Evolution in groups: A deeper look at synaptic cluster driven evolution of deep neural networks. *ArXiv e-prints*, abs/1703.02081, 2017. URL `http://arxiv.org/abs/1703.02081`.

[103] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[104] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[105] P. Simard, D. Steinkraus, and J. C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *International Conference on Document Analysis and Recogntion (ICDAR)*, volume 3, pages 958–962, 2003.

[106] R. E. Smith, B. A. Dike, and S. A. Stegmann. Fitness inheritance in genetic algorithms. In *Proceedings of the 1995 ACM Symposium on Applied Computing*, SAC '95, 1995.

[107] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *ArXiv e-prints*, abs/1505.00387, 2015.

[108] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10:99–127, 2002.

[109] K. O. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9 (2):93–130, 2003.

[110] K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research (JAIR)*, 21:63–100, 2004.

[111] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based indirect encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009.

[112] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *ArXiv e-prints*, abs/1312.6199, 2013.

[113] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[114] P. A. Szerlip, G. Morse, J. K. Pugh, and K. O. Stanley. Unsupervised feature learning through divergent discriminative feature accumulation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-2015)*, Menlo Park, CA, 2015. AAAI Press.

[115] K. Theofilatos, G. Beligiannis, and S. Likothanassis. Combining evolutionary and stochastic gradient techniques for system identification. *Journal of Computational and Applied Mathematics*, 227(1):147–160, 2009.

[116] T. Tieleman and G. Hinton. Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4, 2012.

[117] P. Verbancsics and K. O. Stanley. Constraining connectivity to encourage modularity in HyperNEAT. In *GECCO '11: Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 1483–1490, Dublin, Ireland, 12-16 July 2011. ACM. ISBN 978-1-4503-0557-0. doi: doi:10.1145/2001576.2001776.

[118] P. J. Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770, Heidelberg, Germany, 1982. Springer.

[119] D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14:347–361, 1990.

[120] W. Wienholt. Minimizing the system error in feedforward neural networks with evolution strategy. In *ICANN 93*, pages 490–493. Springer, 1993.

[121] D. R. Wilson and T. R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

[122] L. Xie and A. Yuille. Genetic cnn. *ArXiv e-prints*, abs/1703.01513, 2017. URL `http://arxiv.org/abs/1703.01513`.

[123] C. Yadav and L. Bottou. Cold case: The lost mnist digits. *arXiv preprint arXiv:1905.10498*, 2019.

[124] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[125] J. Ye. Improved cosine similarity measures of simplified neutrosophic sets for medical diagnoses. *Artificial intelligence in medicine*, 63(3):171–179, 2015.

[126] W. Yin, K. Kann, M. Yu, and H. Schütze. Comparative study of cnn and rnn for natural language processing. *arXiv preprint arXiv:1702.01923*, 2017.