A SURVEY OF TECHNIQUES ALL CLASSIFIERS CAN LEARN FROM DEEP NETWORKS: MODELS, OPTIMIZATIONS, AND REGULARIZATION

A PREPRINT

Alireza Ghods
Department of Computer Science
Washington State University
Pullman, WA 99163
alireza.ghods@wsu.edu

Diane J. Cook
Department of Computer Science
Washington State University
Pullman, WA 99163
djcook@wsu.edu

September 30, 2019

ABSTRACT

Deep neural networks have introduced novel and useful tools to the machine learning community. Other types of classifiers can potentially make use of these tools as well to improve their performance and generality. This paper reviews the current state of the art for deep learning classifier technologies that are being used outside of deep neural networks. Non-network classifiers can employ many components found in deep neural network architectures. In this paper, we review the feature learning, optimization, and regularization methods that form a core of deep network technologies. We then survey non-neural network learning algorithms that make innovative use of these methods to improve classification. Because many opportunities and challenges still exist, we discuss directions that can be pursued to expand the area of deep learning for a variety of classification algorithms.

***K*eywords** Deep Learning · Deep Neural Networks, · , · Regularization

1 Introduction

The objective of supervised learning algorithms is to identify an optimal mapping between input features and output values based on a given training dataset. A supervised learning method that is attracting substantial research and industry attention is Deep Neural Networks (DNN). DNNs have a profound effect on our daily lives; they are found in search engines [1] [2], self-driving cars [3] [4] [5], health care systems [6], and consumer devices such as smartphones and cameras [7]. Convolutional Neural Networks (CNN) have become the standard for processing images [8] [9] [10], whereas Recurrent Neural Networks (RNN) dominate the processing of sequential data such as text and voice [11] [12] [13] [14]. DNNs allow machines to automatically discover the representations needed for detection or classification of raw input [15]. Additionally, the neural network community developed unsupervised algorithms to help with the learning of unlabeled data. These unsupervised methods have found their way to real-world applications, such as creating generative adversarial networks (GANs) that design clothes [16]. The term ***deep*** has been used to distinguish these networks from shallow networks which have only one hidden layer; in contrast, DNNs have multiple hidden layers. The two terms ***deep learning*** and ***deep neural networks*** have been used synonymously. However, we observe that deep leaning itself conveys a broader meaning, which can also shape the field of machine learning outside the realm of neural network algorithms.

The remarkable recent DNN advances were made possible by the availability of massive amounts of computational power and labeled data. However, these advances do not overcome all of the difficulties associated with DNNs. For example, there are many real-world scenarios, such as analyzing power distribution data [17], for which large annotated datasets do not exist due to the complexity and expense of collecting data. While applications like clinical interpretation of medical diagnoses require that the learned model be understandable, most DNNs resist interpretation

due to their complexity [18]. DNNs can be insensitive to noisy training data [19] [20] [21], and they also require appropriate parameter initialization to converge [22] [23].

Despite these shortcomings, DNNs have reported higher predictive accuracy than other supervised learning methods for many datasets, given enough supervised data and computational resources. Deep models offer structural advantages that may improve the quality of learning in complex datasets as empirically shown by Bengio [24]. Recently, researchers have designed hybrid methods which combine unique DNN techniques with other classifiers to address some of these identified problems or to boost other classifiers. This survey paper investigates these methods, reviewing classifiers which have adapted DNN techniques to alternative classifiers.

1.1 Research Objectives and Outline

While DNN research is growing rapidly, this paper aims to draw a broader picture of deep learning methods. Although some studies provide evidence that DNN models offer greater generalization than classic machine learning algorithms for complex data [25] [26] [27] [28] [29], there is no "silver bullet" approach to concept learning [30]. Numerous studies comparing DNNs and other supervised learning algorithms [31] [32] [33] [34] [35] observe that the choice of algorithm depends on the data - no ideal algorithm exists which generalizes optimally on all types of data. Recognizing the unique and important role other classifiers thus play, we aim to investigate how non-network machine learning algorithms can benefit from the advances in deep neural networks. Many deep learning survey papers have been published that provide a primer on the topic [36] or highlight diverse applications such as object detection [37], medical record analysis [38], activity recognition [39], and natural language processing [40]. In this survey, we do not focus solely on deep neural network models but rather on how deep learning can inspire a broader range of classifiers. We concentrate on research breakthroughs that transform non-network classifiers into deep learners. Further, we review deep network techniques such as stochastic gradient descent that can be used more broadly, and we discuss ways in which non-network models can benefit from network-inspired deep learning innovations.

The literature provides evidence that non-network models may offer improved generalizability over deep networks, depending on the amount and type of data that is available. By surveying methods for transforming non-network classifiers into deep learners, these approaches can become stronger learners. To provide evidence of the need for continued research on this topic, we also implement a collection of shallow and deep learners surveyed in this paper, both network and non-network classifiers, to compare their performance. Figure 1 highlights deep learning components that we discuss in this survey. This graph also summarizes the deep classifiers that we survey and the relationships that we highlight between techniques.

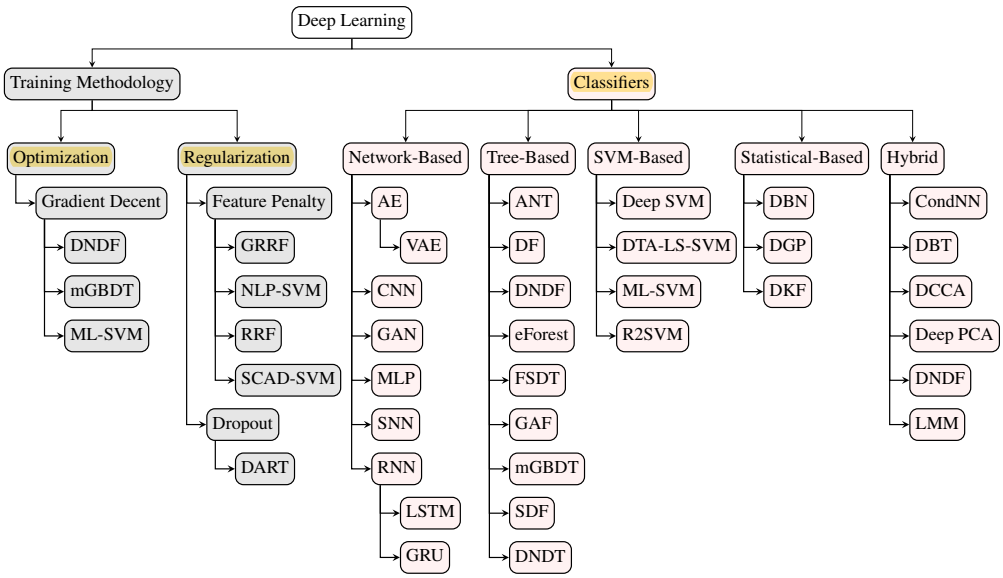

Figure 1: Content map of the methods covered in this survey.

2 Brief Overview of Deep Neural Networks

2.1 The Origin

In 1985, Rosenblatt introduced the Perceptron [41], an online binary classifier which flows input through a weight vector to an output layer. Perceptron learning uses a form of gradient descent to adjust the weights between the input and output layers to optimize a loss function [42]. A few years later, Minsky proved that a single-layer Perceptron is unable to learn nonlinear functions, including the XOR function [43]. Multilayer perceptrons (MLPs, see Table 4 for a complete list of abbreviations) addressed the nonlinearity problem by adding layers of hidden units to the networks and applying alternative differentiable activation functions, such as sigmoid, to each node. Stochastic gradient descent was then applied to MLPs to determine the weights between layers that minimize function approximation errors [44]. However, the lack of computational power caused DNN research to stagnate for decades, and other classifiers rose in popularity. In 2006, a renaissance began in DNN research, spurred by the introduction of Deep Belief Networks (DBNs) [45].

2.2 Deep Neural Network Architectures

Due to the increasing popularity of deep learning, many DNN architectures have been introduced with variations such as Neural Turing Machines [46] and Capsule Neural Networks [47]. In this paper, we summarize the general form of DNNs together with architectural components that not only appear in DNNs but can be incorporated into other models. We start by reviewing popular types of DNNs that have been introduced and that play complementary learning roles.

2.3 Supervised Learning

2.3.1 Multilayer Perceptron

A multilayer perceptron (MLP) is one of the essential bases of many deep learning algorithms. The goal of a MLP is to map input X to class y by learning a function $y = f(X, \theta)$, where θ represents the best possible function approximation. For example, in Figure 2 the MLP maps input X to y using function $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$, where $f^{(1)}$ is the first layer, $f^{(2)}$ is the second layer, and $f^{(3)}$ represents the third, output layer. This chain structure is a common component of many DNN architectures. The network depth is equal to the length of the chain, and the width of each layer represents the number of nodes in that layer [48].

In networks such as the MLP, the connections are not cyclic and thus belong to a class of DNNs called *feedforward networks*. Feedforward networks move information in only one direction, from the input to the output layer. Figure 2 depicts a particular type of feedforward network which is a fully-connected multilayer perceptron because each node at one layer is connected to all of the nodes at the next layer. Special cases of feedforward networks and MLPs have drawn considerable recent attention, which we describe next.

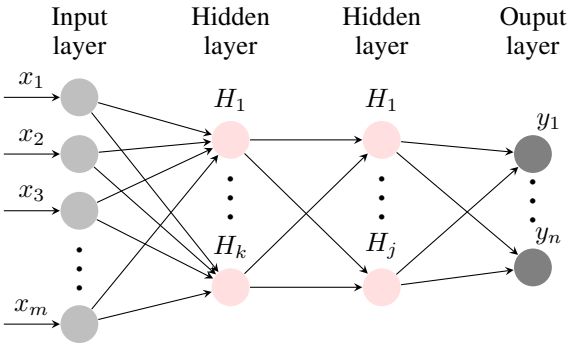

Figure 2: An illustration of a three-layered MLP with j nodes at the first hidden layer and k at the second layer.

2.3.2 Deep Convolutional Neural Network

A convolutional neural network (CNN) [49] is a specialized class of feedforward DNNs for processing data that can be discretely presented. Examples of data that can benefit from CNNs include time series data that can be presented as samples of discrete regular time intervals and image data presented as samples of 2-D pixels at discrete locations.

Most CNNs involve three stages: a convolution operation; an activation function, such as the rectified linear activation (ReLU) function [50]; and a pooling function, such as max pooling [51]. A convolution operation is a weighted average or smooth estimation of a windowed input. One of the strengths of the convolution operation is that the connections between nodes in a network become sparser by learning a small kernel for unimportant features. Another benefit of convolution is parameter sharing. A CNN makes an assumption that a kernel learned for one input position can be used at every position, in contrast to a MLP which deploys a separate element of a weight matrix for each connection. Applying the convolution operator frequently improves the network's learning ability.

A pooling function replaces the output of specific nearby nodes by their statistical summary. For example, the max-pooling function returns the maximum of a rectangular neighborhood. The motivation behind adding a pooling layer is that statistically down-sampling the number of features makes the representation approximately invariant to small translations of the input by maintaining the essential features. The final output of the learner is generated via a Fully-Connected (FC) layer that appears after the convolutional and max-pooling layers (see Figure 3 for an illustration of the process).

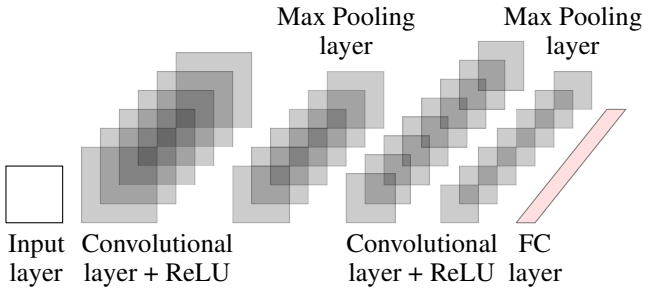

Figure 3: An illustration of a three-layered CNN made of six convolution filters followed by six max pooling filters at the first layer, and eight convolution filters followed by seven max pooling filters at the second layer. The last layer is a fully connected layer (FC).

2.3.3 Recurrent Neural Network

A recurrent Neural Network (RNN) is a sequential model that can capture the relationship between items in a sequence. Unlike traditional neural networks, wherein all inputs are independent of each other, RNNs contain artificial neurons with one or more feedback loops. Feedback loops are recurrent cycles over time or sequence, as shown in Figure 4. An established RNN problem is exploding or vanishing gradients. For a long data sequence, the gradient could become increasingly smaller or increasingly larger, which halts the learning. To address this issue, Hochreiter et al. [52] introduced a long short-term memory (LSTM) model and Cho et al. [53] proposed a gated recurrent unit (GRU) model. Both of these networks allow the gradient to flow unchanged in the network, thus preventing exploding or vanishing gradients.

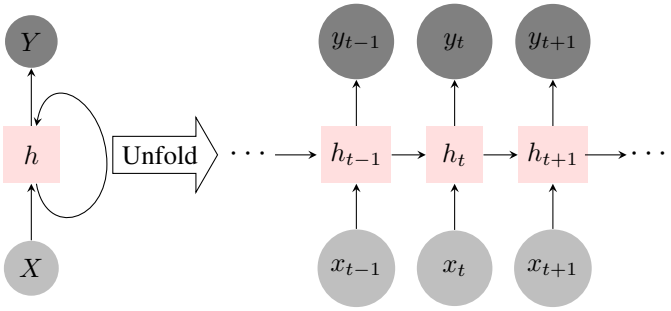

Figure 4: An illustration of a simple RNN and its unfolded structure through time t.

2.3.4 Siamese Neural Network

There are settings in which the number of training samples is limited, such as in facial recognition scenarios where only one image is available per person. When there is a limited number of examples for each class, DNNs struggle

with generalizing the model. One strategy for addressing this problem is to learn a similarity function. This function computes the degree of difference between two samples, instead of learning each class. As an example, let x_1 represent one facial image and x_2 represent a second. If $d(x1, x2) \leq \tau$, we can conclude that the images are of the same person while $d(x_1, x_2) > \tau$ implies that they are different people. Siamese Neural Networks (SNN) [54] build on this idea by encoding examples x_i and x_j on two separate DNNs with shared parameters. The SNN learns a function d using encoded features as shown in Figure 5. The network then outputs $y > 0$ for similar objects (i.e., when d is less then a threshold value) and $y < 0$ otherwise. Thus, SNNs can be used for similarity learning by learning a distance function over objects. In addition to their value for supervised learning from limited samples, SNNs are also beneficial for unsupervised learning tasks [55] [56].

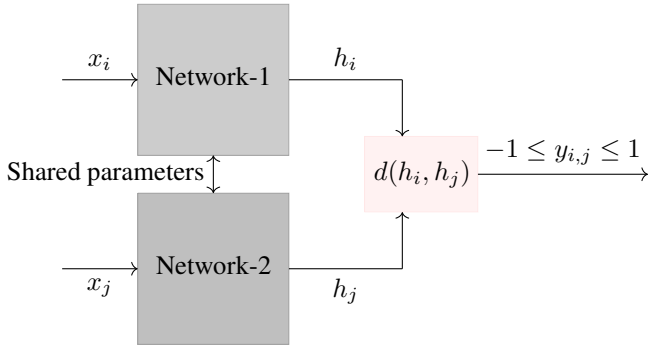

Figure 5: An illustration of an SNN. In this figure, x_i and x_j are two data vectors corresponding to a pair of instances from the training set. Both networks share the same weights and map the input to a new representation. By comparing the outputs of the networks using a distance measure such as Euclidean, we can determine the compatibility between instances x_i and x_j.

2.4 Unsupervised Learning

2.4.1 Generative Adversarial Network

Until this point in the survey, we have focused on deep learning for its power in classifying data points. However, researchers have exploited deep learning for other uses as well, such as generating synthetic data that shares characteristics with known real data. One way to create synthetic data is to learn a generative model. A generative model learns the parameters that govern a distribution based on observation of real data points from that distribution. The learned model can then be used to create arbitrary amounts of synthetic data that emulate the real data observations. Recently, researchers have found a way to exploit multiplayer games for the purpose of improving generative machine learning algorithms. In the adversarial training scenario, two agents compete against each other, as inspired by Samuel [57] who designed a computer program to play checkers against itself. Goodfellow et al. [58] put this idea to use in developing Generative Adversarial Networks (GANs), in which a DNN (generator) tries to generate synthetic data that is so similar to real data that it fools its opponent DNN (discriminator), whose job is to distinguish real from fake data (see Figure 6 for an illustration). The goal of GANs is to simultaneously improve the ability of the generator to produce realistic data and of the discriminator to distinguish synthetic from real data. GANs have found successful application in diverse tasks including translating text to images [59], discovering drugs [60], and transforming sketches to images [61] [62].

2.4.2 Autoencoder

Yet another purpose for deep neural networks is to provide data compression and dimensionality reduction. An Autoencoder (AE) is a DNN that accomplishes this goal by creating an output layer that resembles the input layer, using a reduced set of terms represented by the middle layers [48]. Architecturally, an AE combines two networks. The first network, called the encoder, learns a new representation of input x with fewer features $h = f(x)$; the second part, called the decoder, maps h onto a reconstruction of the input space $\hat{y} = g(h)$, as shown in Figure 7. The goal of an AE is not simply to recreate the input features. Instead, an AE learns an approximation of the input features to identify useful properties of the data. AEs are vital tools for dimensionality reduction [63], feature learning [64], image colorization [65], higher-resolution data generation [66], and latent space clustering [67]. Additionally, other versions of AEs such as variational autoencoders (VAEs) [68] can be used as generative models.

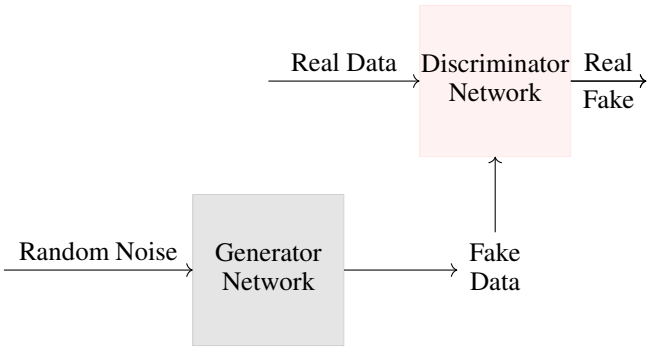

Figure 6: An illustration of a GAN. The goal of the discriminator network is to distinguish real data from fake data, and the goal of the generator network is to use the feedback from the discriminator to generate data that the discriminator cannot distinguish from real.

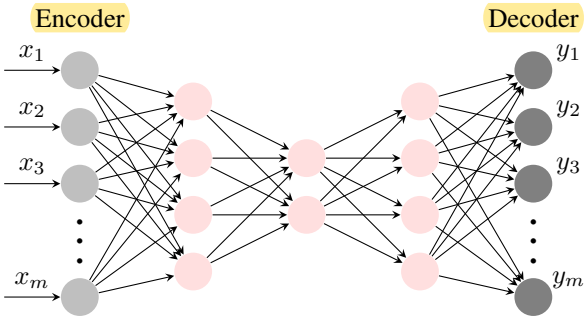

Figure 7: An illustration of an AE. The first part of the network, called the encoder, compresses input into a latent-space by learning the function $h = f(x)$. The second part, called the decoder, reconstructs the input from the latent-space representation by learning the function $\hat{y} = g(h)$.

2.5 Optimization for Training Deep Neural Networks

In the previous section, we described common DNN architecture components. In this section, we offer a brief overview of optimization approaches for training DNNs. Learning methods may optimize a function $f(x)$ (e.g., minimize a loss function) by modifying model parameters (e.g., changing DNN weights). However, as Bengio et al. [69] point out, DNN optimization during training may be further complicated by local minima and ill-conditioning (see Figure 8 for an illustration of an ill-condition).

The most common type of optimization strategy employed by DNNs is gradient descent. This intuitive approach to learns the weights of connections between layers which reduce the network's objective function by computing the error derivative with respect to a Ir-level layer of the network. Input x is fed forward through a network to predict $\hat{y}$. A cost function $J(\theta)$ measures the error of the network at the output layer. Gradient descent then directs the cost value to flow backward through the network by computing the gradient of the objective function $\nabla_\theta J(\theta)$. This process is sometimes alternatively referred to as backpropagation because the training error propagates backward through the network from output to input layers. Many variations of gradient descent have been tested for DNN optimization, such as stochastic gradient descent, mini-batch gradient descent, momentum [70], Ada-Grad [71], and Adam [72].

Deep network optimization is an active area of research. Along with gradient descent, many other algorithms such as derivative-free optimization [73] and feedback-alignment [74] have appeared. However, none of these algorithms are as popular as the gradient descent algorithms.

2.6 Regularization

Regularization was an optimization staple for decades prior to the development of DNNs. The rationale behind adding a regularizer to a classifier is to avoid the overfitting problem, where the classifier fits the training set too closely instead of generalizing to the entire data space. Goodfellow et al. [48] defined regularization as "any modification to a learning algorithm that is intended to reduce its generalization error but not its training error". While regularization

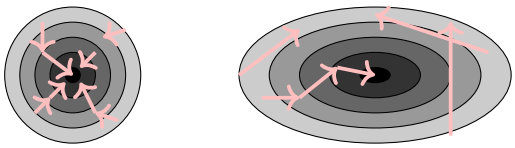

Figure 8: The left-hand side loss surface depicts a well-conditioned model where local minima can be reached from all directions. The right-hand side loss surface depicts an ill-conditioned model where there are several ways to overshoot or never reach the minima.

methods such as bagging have been popular for neural networks and other classifiers, recently the DNN community has developed novel regularization methods that are unique to deep neural networks. In some cases, backpropagation training of fully-connected DNNs results in poorer performance than shallow structures because the deeper structure is prone to being trapped in local minima and overfitting the training data. To improve the generalizability of DNNs, regularization methods have thus been adopted during training. Here we review the intuition behind the most frequent regularization methods that are currently found in DNNs.

2.6.1 Parameter Norm Penalty

A conventional method for avoiding overfitting is to penalize large weights by adding a p-norm penalty function to the optimization function of the form $f(x)+$ *p-norm*(x), where the p-norm p for weights w is denoted as $||w||_p = (\sum_i |w_i|^p)^{\frac{1}{p}}$. Popular p-norms are the L_1 and L_2 norms which have been used by other classifiers such as logistic regression and SVMs prior to the introduction of DNNs. L_1 adds a regularization term $\Omega(\theta) = ||w||_1$ to the objective function for weights w, while L_2 adds a regularization term $\Omega(\theta) = ||w||_2$. The difference between the L_1 and L_2 norm penalty functions is that L_1 penalizes features more heavily by setting the corresponding edge weights to zero compared to L_2. Therefore, a classifier with the L_1 norm penalty tends to prefer a sparse model. The L_2 norm penalty is more common than the L_1 norm penalty. However, it is often advised to use the L_1 norm penalty when the amount of training data is small and the number of features is large to avoid noisy and less-important features. Because of its sparsity property, the L_1 penalty function is a key component of LASSO feature selection [75].

2.6.2 Dropout

A powerful method to reduce generalization error is to create an ensemble of classifiers. Multiple models are trained separately, then as an ensemble they output a combination of the models' predictions on test points. Some examples of ensemble methods included bagging [76], which trains k models on k different folds of random samples with replacement, and boosting [77], which applies a similar process to weighted data. A variety of DNNs use boosting to achieve lower generalization error [45] [78] [79].

Dropout [80] is a popular regularization method for DNNs which can be viewed as a computationally-inexpensive application of bagging to deep networks. A common way to apply dropout to a DNN is to deactivate a randomly-selected 50% of the hidden nodes and a randomly-selected 20% of the input nodes for each mini-batch of data. The difference between bagging and dropout is that in bagging the models are independent of each other, while in dropout each model inherits a subset of parameters from the parent deep network.

2.6.3 Data Augmentation

DNNs can generalize better when they have more training data; however, the amount of available data is often limited. One way to circumvent this limitation is to generate artificial data from the same distribution as the training set. Data augmentation has been particularly effective when used in the context of classification. The goal of data augmentation is to generate new training samples from the original training set (X, y) by transforming the X inputs. Data augmentation may include generating noisy data to improve robustness (denoising) or creating additional training data for the purpose of regularization (synthetic data generation). Dataset augmentation has been adopted for a variety of tasks such as image recognition [81] [82], speech recognition [83], and activity recognition [84]. Additionally, GANs [85] [86] and AEs [87] [88], described in Sections 2.4.1 and 2.4.2, can be employed to generate such new examples.

Injecting noise into a copy of the input is another data augmentation method. Although DNNs are not consistently robust to noise [89], Poole et al. [90] show that DNNs can benefit from carefully-tuned noise.

3 Deep Learning Architectures Outside of Deep Neural Networks

Recent research has introduced numerous enhancements to the basic neural network architecture that enhance network classification power, particularly for deep networks. In this section, we survey non-network classifiers that also make use of these advances.

3.1 Supervised Learning

3.1.1 Feedforward Learning

A DNN involves multiple layers of operations that are performed sequentially. The idea of creating a sequence of operations, each of which manipulates the data before passing them to the next operator, may be used to improve many types of classifiers. One way to construct a model with a deep feedforward architecture is to use stacked generalization [91] [92]. Stacked generalization classifiers are comprised of multiple layers of classifiers stacked on top of each other as found in DNNs. In stacked generalization classifiers, one layer generates the next layer's input by concatenating its own input to its output. Stacked generalization classifiers typically only implement forward propagation, in contrast to DNNs which propagate information both forward and backward through the model.

In general, learning methods that employ stacked generalization can be categorized into two strategies. In the first stacked generalization strategy, the new feature space for the current layer comes from the concatenation of the predicted output of the previous layer with the original feature vector. Here, layers refer not to layers of neural network operations, but instead refer to sequences of other types of operations. Examples of this strategy include Deep Forest (DF) [93] and the Deep Transfer Additive Kernel Least Square SVM (DTA-LS-SVM) [94]. At any given layer, for each instance x, DF extends x's previous feature vector to include the previous layer's predicted class value for the instance. The prediction represents a distribution over class values, averaged over all trees in the forest. Furthermore, Zhou et al. [93] introduce a method called Multi-Grained Scanning for improving the accuracy of DFs. Inspired by CNNs and RNNs where spatial relationships between features are critical, Multi-Grained Scanning splits a D-dimensional feature vector into smaller segments by moving a window by moving a window over the features. For example, given 400 features and a window size of 100, the original features convert to 301 features of length 100, $\{< 1-100 >, < 2-101 >, \ldots, < 301-400 >\}$, where the new instances have the same labels as the original instances. The new samples which are described by a subset of the original features might have incorrectly-associated labels. At a first glance, it seems these noisy data could hurt the generalization. But as Breiman illustrates [95], perturbing a percentage of the training labels can actually help generalization.

Furthermore, Ho [96] demonstrates that feature sub-sampling can enhance the generalization capability for RFs. Zhou et al. [93] tested three different window sizes ($D/4$, $D/8$, and $D/16$), where data from each different window size fits a different level of a DF model. Then the newly-learned representation from these three layers are fed to a multilayer DF, applying subsampling when the transformed features are too long. Multi-Grained Scanning can improve the performance of a DF model for continuous data, as Zhou et al. [93] report that accuracy increased by 1.24% on the MNIST [97] dataset. An alternative method, DTA-LS-SVM, applies an Additive Kernel Least Squares SVM (AK-LS-SVM) [98] [99] at each layer and concatenates the original feature vector x with the prediction of the previous level to feed to the next layer. In addition, DTA-LS-SVM incorporates a parameter-transfer approach between the source (previous-layer learner) and target (next-layer learner) to enhance the classification capability of the higher level.

In the second stacked generalization strategy, the current layer's new feature space comes from the concatenation of predictions from all previous layers with the original input feature vector. Examples of this strategy include the Deep SVM (D-SVM) [100] and the Random Recursive SVM (R2-SVM) [101]. The D-SVM contains multiple layers of SVMs, where the first layer is trained in the normal fashion. Following this step, each successive layer employs the kernel activation from the previous layer with the desired labels. The R2-SVM is a multilayer SVM model which at each layer transforms the data based on the sigmoid of a projection of all previous layers' outputs. For the data (X, Y) where $X \in R^D$ and $Y \in R^C$, the random projection matrix is $W \in R^{D \times C}$, where each element is sampled from $N(0, 1)$. The input data for the next layer is:

$$X_{l+1} = \sigma(d + \beta W_{l+1}[o_1^T, o_2^T, ..., o_l^T]^T), \tag{1}$$

where β is a weight parameter that controls the degree with which a data sample in X_{l+1} moves from the previous layer, $\sigma(.)$ is the sigmoid function, W_{l+1} is the concatenation of l random projection matrices $[W_{l+1,1}, W_{l+1,2}, ..., W_{l+1,l}]$, one for each previous layer, and o is the output of each layer. Addition of a sigmoid function to the recursive model prevents deterioration to a trivial linear model in a similar fashion as MLPs. The purpose of the random projection is to push data from different classes in different directions.

It is important to note here that stacked generalization can be found in DNNs as well as non-network classifiers. Examples of DNNs with stacked generalization include Deep Stacking Networks [102] [103] and Convex Stacking

Architectures [104] [102]. This is clearly one enhancement that benefits all types of classifier strategies. However, there is no evidence that stack generalization could add nonlinearity to the model.

DNN classifiers learn a new representation of data at each layer with a goal that the newly-learned representation maximally separate the classes. Unsupervised DNNs often share this goal. As an example, Deep PCA's model [105] is made of two layers that each learn a new data representation by applying a Zero Components Analysis (ZCA) whitening filter [106] followed by a principal components analysis (PCA) [107]. The final data representation is derived from concatenating the output of the two layers. The motivation behind applying a ZCA whitening filter is to force the model to focus on higher-order correlations. One motivation for combining output from the first and second layers could be to preserve the learned representation from the first layer and to prevent feature loss after applying PCA at each layer. Experiments demonstrate that Deep PCA exhibits superior performance for face recognition tasks compared to standard PCA and a two-layer PCA without a whitening filter. However, as empirically confirmed by Damianou et al. [108], stacking PCAs does not necessarily result in an improved representation of the data because Deep PCA is unable to learn a nonlinear representation of data at each layer. Damianou et al. [108] fed a Gaussian to a Deep PCA and observed that the model learned just a lower rank of the input Gaussian at each layer.

As pointed out earlier in this survey, the invention of the deep belief net (DBN) [45] drew the attention of researchers to developing deep models. A DBN can be viewed as a stacked restricted Boltzmann machine (RBM), where each layer is trained separately and alternates functionality between hidden and input units. In this model, features learned at hidden layers then represent inputs to the next layer. A RBM is a generative model that contains a single hidden layer. Unlike the Boltzmann machine, hidden units in the restricted model are not connected to each other and contain undirected, symmetrical connections from a layer of visible units (inputs). All of the units in each layer of a RBM are updated in parallel by inputting the current state of the unit to the other layer. This updating process repeats until the system is sampling from an equilibrium distribution. The RBM learning rule is shown in Equation 2.

$$\frac{\partial \log P(v)}{\partial W_{ij}} \approx < v_i h_j >_{data} - < v_i h_j >_{reconstruction} \tag{2}$$

In this equation, W_{ij} represents the weight vector between a visible unit v_i and a hidden unit h_j, and $< . >$ is the average value over all training samples. Since the introduction of DBNs, many other different variations of Deep RBMs have been proposed such as temporal RBMs [109], gated RBMs [110], and cardinality RBMs [111].

Another novel form of a deep belief net is a deep Gaussian process (DGP) model [108]. DGP is a deep directed graph where multiple layers of Gaussian processes map the original features to a series of latent spaces. DGPs offer a more general form of Gaussian Processes (GPs) [112] where a one-layer DGP consists of a single GP, f. In a multilayer DGP, each GP, f_l, maps data from one latent space to the next. As shown in Equation 3, each data point Y is generated from the corresponding function f_l with ϵ Guassian noise applied to data X_l that is obtained from a previous layer.

$$Y = f_l(X_l) + \epsilon_l, \epsilon_l \sim \mathcal{N}(0, \sigma_l^2 I) \tag{3}$$

Figure 9 illustrates a DGP expressed as a series of Gaussian processes mapping data from one latent space to the next. Functions f_l are drawn from a Gaussian process, i.e. $f(x) \sim \mathcal{GP}(0, k(x, x'))$. In this setting, the covariance function k defines the properties of the mapping function. DGP can be utilized for both supervised and unsupervised learning. In the supervised setting, the top hidden layer is observed, whereas in the unsupervised setting, the top hidden layer is set to a unit Gaussian as a fairly uninformative prior. DGP is a powerful non-parametric model but it has only been tested on small datasets. Also, we note that researchers have developed deep Gaussian process models with alternative architectures such as recurrent Gaussian processes [113], convolutional Gaussian processes [114] and variational auto-encoded deep Gaussian processes [115]. There exists a vast amount of literature on this topic that provides additional insights on deep Gaussian processes [116] [117] [118].

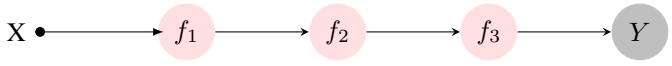

Figure 9: A deep Gaussian process with two hidden layers.

As we discussed, non-network classifiers have been designed that contain multiple layers of operations, similar to a DNN. We observe that a common strategy for creating a deep non-network model is to add the prediction of the previous layer or layers to the original input feature. Likewise, novel methods can be applied to learn a new representation of data at each layer. We discuss these methods next.

3.1.2 Siamese Model

As discussed in Section 2.3.4, a SNN represents a powerful method for similarity learning. However, one problem with SNNs is overfitting when there is a small number of training examples. The Siamese Deep Forest (SDF) [119] is a method based on DF which offers an alternative to a standard SNN. The SDF, unlike the SNN, uses only one DF. The first step in training a SDF is to modify the training examples. The training set consists of the concatenation of each pair of samples in the original set. If sample points x_i and x_j are semantically similar, the corresponding class label is set to zero; otherwise, the class label is set to one. The difference between the SDF and the DF in training is that the Siamese Deep Forest concatenates the original feature vector with a weighted sum of the tree class probabilities. Training of SDF is similar to DF; the primary difference is that SDF learns the class probability weights w for each forest separately at each layer. Learning the weights for each forest can be accomplished by minimizing the function in Equation 4.

$$\min_{w} J_q(w) = \min_{w} \sum_{i,j} l(x_i, x_j, y_{ij}, w) + \lambda R(w) \tag{4}$$

Here, w represents a concatenation of vectors w^k, $k = 1, ..., M$, q is the SDF layer, $R(w)$ is a regularization term, and λ is a hyper-parameter to control regularization. Detailed instructions on minimizing Equation 4 are found in the literature [119]. The results of SDF experiments indicate that the SDF can achieve better classification accuracy than DF for small datasets. In general, all non-network models that learn data representations can take advantage of the Siamese architecture like SDF.

3.2 Unsupervised Learning

3.2.1 Generative Adversarial Model

A common element found in GANs is inclusion of a FC layer in the discriminator. One issue with the FC layer is that it cannot deal with the ill-condition in which local minima are not surrounded by spherical wells as shown in Figure 8. The Generative Adversarial Forest (GAF) [120] replaces the FC layer of the discriminator with a deep neural decision forest (DNDF), which is discussed in Section 4. GAF and DNDF are distinguished based on how leaf node values are learned. Instead of learning leaf node values iteratively, as DNDF does, GAF learns them in parallel across the ensemble members. The strong discriminatory power of the decision forest is the reason the authors recommend this method in lieu of the fully-connected discriminator layer.

In this previous work, the discriminator is replaced by an unconventional model. We hypothesize that replacing the discriminator with other classifiers such as Random Forest, SVM, of K nearest neighbor based on the data could result in a diverse GAN strategies, each of which may offer benefits for alternative learning problems.

3.2.2 Autoencoder

As we discussed in Section 2.4.2, AEs offer strategies for dimensionality reduction and data reconstruction from compressed information. The autoencoding methodology can be found in neural networks, non-networks, and hybrid methods. As an example, the multilayer SVM (ML-SVM) autoencoder is a variation of ML-SVM with the same number of output nodes as input features and a single hidden layer that consists of fewer nodes than the input features. ML-SVM is a model with the same structure as a MLP. The distinction here is that the network contains SVM models as its nodes. A review of ML-SVM is discussed in Section 4. The outputs of hidden nodes are fed as input to each SVM output node c as follows:

$$g_c(f(X|\theta)) = \sum_{i=1}^{l} (\alpha_i^{c*} - \alpha_i^c) K_o(f(x_i|\theta), f(x|\theta)) + b_c, \tag{5}$$

where α_i^{c*} and α_i^c are the support vector coefficients, K_o is the kernel function, and b_c is their bias. The error back-propagates through the network to update the parameters.

Another exciting emerging research area is the combination of Kalman filters with deep networks. A Kalman filter is a well-known algorithm that estimates the optimal state of a system from a series of noisy observations. The classical Kalman filter [121] is a linear dynamical system and therefore is unable to model complex phenomena. For this reason, researchers developed nonlinear versions of Kalman filters. In a seminal contribution, Krishnan et al. [122] introduced a model that combines a variational autoencoder with Kalman filters for counterfactual inference of patient information. In a standard autoencoder, the model learns a latent space that represents the original data minus extraneous information or "signal noise". In contrast, a variational autoencoder (VAE) [68] adds a constraint to the encoder that it learn a Gaussian distribution of the original input data. Therefore, a VAE is able to generate

a latent vector by sampling from the learned Gaussian distribution. Deep Kalman filters (DKF) learn a generative model from observed sequences $\vec{x} = (x_1, \cdots, x_T)$ and actions $\vec{u} = (u_1, \cdots u_{T-1})$, with a corresponding latent space $\vec{z} = (z_1, \cdots, z_T)$, as follows:

$$\begin{aligned} z_1 &\sim \mathcal{N}(\mu_0, \Sigma_0) \\ z_t &\sim \mathcal{N}(G_\alpha(z_{t-1}, u_{t-1}, \Delta_t), S_\beta(z_{t-1}, y_{t-1}, \Delta_t)) \\ x_t &\sim \Pi(F_k(z_t)), \end{aligned} \tag{6}$$

where $\mu_0 = 0$ and $\Sigma_0 = I_d$, Δ_t represents the difference between times t and $t-1$, and Π represents a distribution (e.g., Bernoulli for binary data) over observation x_t. The functions G_α, S_β, F_k are parameterized by a neural net. As a result, the autoencoder will learn $\theta = \{\alpha, \beta, k\}$ parameters. Additionally, Shashua et al. [123] introduced deep Q-learning with Kalman filters and Lu et al. [124] presented a deep Kalman filter model for video compression.

As we highlighted in this section, non-network methods have been designed that are inspired by AEs. Although ML-SVM mimics the architecture of AEs, its computational cost prevents the algorithm from being a practical choice. DKF takes advantage of the VAE idea by learning a Kalman Filter in its middle layer. Additionally, Feng et al. [125] introduced an encoder forest, a model inspired by the DNN autoencoder. Because the encoder forest is not a deep model, we do not include the details of this algorithm in our survey.

4 Deep Learning Optimization Outside of Deep Neural Networks

As discussed in Section 2.5, gradient descent has been a prominent optimization algorithm for DNNs; however, it has been underutilized by non-network classifiers. Some notable exceptions are found in the literature. We discuss these here.

A resourceful method for constructing a deep model is to start with a DNN architecture and then replace nodes with non-network classifiers. As an example, the multilayer SVM (ML-SVM) [126] replaces nodes in a MLP with standard SVMs. ML-SVM is a multiclass classifier which contains SVMs within the network. At the output layer, the ML-SVM contains the same number of SVMs as the number of classes learned at the perceptron output layer. Each SVM at the ML-SVM output layer is trained in a one-versus-all fashion for one of the classes. When observing a new data point, ML-SVM outputs the class label corresponding to the SVM that generates the highest confidence. At each hidden layer, SVMs are associated with each node that learn latent variables. These variables are then fed to the output layer. At hidden layer $f(X|\theta)$ where X is the training set and θ denotes the trainable parameters of the SVM, ML-SVM maps the hidden layer features to an output value as follows:

$$g(f(X|\theta)) = \sum_{i=1}^{l} y_i^c \alpha_i^c K_o(f(x_i|\theta), f(X|\theta)) + b_c, \tag{7}$$

where g is the output layer function, $y_i^c \in \{-1, 1\}$ for each class c, K_o is the kernel function for the output layer, α_i^c are the support vector coefficients for SVM nodes of the output layer, and b_c is their bias. The goal of ML-SVM is to learn the maximum support vector coefficient of each SVM at the output layer with respect to the objective function $J_c(.)$, as shown in Equation 8.

$$\min_{w^c, b, \xi, \theta} J_c = \frac{1}{2}||w^c||^2 + C\sum_{i}^{l} \xi_i \tag{8}$$

Here, w^c represents the set of weights for class c, C represents a trade-off between margin width and misclassification risk and ξ_i are slack variables. ML-SVM applies gradient ascent to adapt its support vector coefficient towards a local maximum of $J_c(.)$. The support vector coefficient is defined as zero for values less than zero and is assigned to C for values larger than C. The data is backpropagated through the network similar to traditional MLPs by calculating the gradient of the objective function.

The SVMs in the hidden layer are identical. Given the same inputs, they would thus generate the same outputs. To diversity the SVMs, the hidden layers train on a perturbed version of the training set to eliminate producing similar outputs before training the combined ML-SVM model. The outputs of hidden layer nodes are constrained to generate values in the range $[-1 \ldots 1]$. Despite the effort of ML-SVMs to learn a multi-layer data representation, this approach is currently not practical because adding a new node incurs a dramatic computational expense for large datasets.

Kontschieder et al. [127] further incorporate gradient descent into a Random Forest (RF), which is a popular classification method. One of the drawbacks of a RF is that does not traditionally learn new internal representations like DNNs. The Deep Network Decision Forest (DNDF) [127] integrates a DNN into each decision tree within the forest to reduce the uncertainty at each decision node. In DNDF, the result of a decision node $d_n(x, \Theta)$ corresponds to the

output of a DNN $f_n(x, \Theta)$, where x is an input and Θ is the parameter of a decision node. DNDF must have differentiable decision trees to be able to apply gradient descent to the process of updating decision nodes. In a standard decision tree, the result of a decision node $d_n(x, \Theta)$ is deterministic. DNDF replaces the traditional decision node with a sigmoid function $d_n(x, \Theta) = \sigma(f_n(x; \Theta))$ to create a stochastic decision node. The probability of reaching a leaf node l is calculated as the product of all decision node outputs from the root to the leaf l, which is expressed as μ_l in this context. The set of leaf nodes $\mathcal{L}$ learns the class distribution π, and the class with the highest probability is the prediction of the tree. The aim of DNDF is to minimize its empirical risk with respect to the decision node parameter Θ and the class distribution π of $\mathcal{L}$ under the log-loss function for a given data set.

The optimization of the empirical risk is a two-step process which is executed iteratively. The first step is to optimize the class distribution of leaf nodes $\pi_{\mathcal{L}}$ while fixing the decision node parameters and the corresponding DNN. At the start of optimization (iteration 0), class distribution π^0 is set to a uniform distribution across all leaves. DNDF then iteratively updates the class distribution across the leaf nodes as follows for iteration t+1:

$$\pi_{l_y}^{(t+1)} = \frac{1}{Z_l^{(t)}} \sum_{(x,y') \in \mathcal{T}} \frac{\mathbb{1}_{y=y'} \pi_{l_y^{(t)}} \mu_l(x|\Theta)}{\mathbb{P}_T[y|x, \Theta, \pi^{(t)}]}, \tag{9}$$

where $Z_l^{(t)}$ is a normalization factor ensuring that $\sum_y \pi_{l_y}^{t+1} = 1$, $\mathbb{1}_q$ is the indicator function on the argument q, and $\mathbb{P}_T$ is the prediction of the tree.

The second step is to optimize decision node parameters Θ while fixing the class distribution $\pi_{\mathcal{L}}$. DNDF employs gradient descent to minimize log-loss with respect to Θ as follows:

$$\frac{\partial L}{\partial \Theta}(\Theta, \pi; x, y) = \sum_{n \in \mathcal{N}} \frac{\partial L(\Theta, \pi; x, y)}{\partial f_n(x; \Theta)} \frac{\partial f_n(x; \Theta)}{\partial \Theta}. \tag{10}$$

The second term in Equation 10 is the gradient of the DNN. Because this is commonly known, we only discuss calculating the gradient of the differentiable decision tree. Here, the gradient of the differentiable decision tree is given by:

$$\frac{\partial L(\Theta, \pi; x, y)}{\partial f_n(x; \Theta)} = d_n(x; \Theta) A_{n_r} - \bar{d}_n(x; \Theta) A_{n_l}, \tag{11}$$

where d_n is the probability of transitioning to the left child, $\bar{d}_n = 1 - d_n$ is the probability of transitioning to the right child calculated by a forward pass through the DNN, and n_l and n_r indicate the left and right children of node n. To calculate the term A in Equation 11, DNDF performs one forward pass and one backward pass through the differentiable decision tree. Upon completing the forward pass, a value A_l can be initially computed for each leaf node as follows:

$$A_l = \frac{\pi_{l_y} \mu_l}{\sum_l \pi_{l_y} \mu_l}. \tag{12}$$

Next, the values of A_l for each leaf node are used to compute the values of A_m for each internal node m. To do this, a backward pass is made through the decision tree, during which the values are calculated as $A_m = A_{n_l} + A_{n_r}$, where n_l and n_r represent the left and the right children of node m, respectively.

Each layer of a standard DNN produces the output o_i at layer i. As mentioned earlier, the goal of the DNN is to learn a mapping function $F_i : o_{i-1} \rightarrow o_i$ that minimizes the empirical loss at the last layer of DNN on a training set. Because each F_i is differentiable, a DNN updates its parameters efficiently by applying gradient descent to reduce the empirical loss.

Adopting a different methodology, Frosst et al. [128] distill a neural network into a soft decision tree. This model benefits from both neural network-based representation learning and decision tree-based concept explainability. In the Frosst soft decision tree (FSDT), each tree's inner node learns a filter w_i and a bias b_i, and leaf nodes l learn a distribution of classes. Like the hidden units of a neural network, each inner node of the tree determines the probability of input x at node i as follows:

$$p_i(x) = \sigma(\beta(x w_i + b_i)) \tag{13}$$

where σ represents the sigmoid function and β represents an inverse temperature whose function is to avoid soft decisions in the tree. Filter activation routes the sample x to the left branch for values of p_i less than 0.5, and to the right branch otherwise. The probability distribution Q^l for each leaf node l represents the learned parameter ϕ^l at that leaf over the possible k output classes:

$$Q_k^l = \frac{\exp(\phi_k^l)}{\sum_{k'} \exp(\phi_{k'}^l)}. \tag{14}$$

The predictive distribution over classes is calculated by traversing the greatest-probability path. To train this soft decision tree, Frosst et al. [128] calculate a loss function L that minimizes the cross entropy between each leaf, weighted by input vector x path probability and target distribution T, as follows:

$$L(x) = -\log\Big(\sum_{l \in LeafNodes} P^l(x) \sum_k T_k \log Q_k^l\Big) \tag{15}$$

where $P^l(x)$ is the probability of reaching leaf node l given input x. Frosst et al. [128] also introduce a regularization term to avoid internal nodes routing all data points on one particular path and encourage them to equally route data along the left and right branches. The penalty function calculates a sum over all internal nodes from the root to node i, as follows:

$$C = -\lambda \sum_{i \in InnerNodes} 0.5 \log(\alpha_i) + 0.5 \log(1 - \alpha_i) \tag{16}$$

where λ is a hyper-parameter set prior to training to determine the effect of the penalty. The cross entropy α for a node i is the sum of the path probability $P^i(x)$ from the root to node i multiplied by the probability of that node p_i divided by the path probability, as follows:

$$\alpha_i = \frac{\sum_x P^i(x) p_i(x)}{\sum_x P^i(x)}. \tag{17}$$

Because the probability distribution is not uniform across nodes in the penultimate level, this penalty function could actually hurt the generalization. The authors address this problem by decaying the strength of penalty function λ exponentially with the depth d of the node to 2^d. Another challenge is that in any given batch of data, as the data descends the tree, the number of samples decreases exponentially. Therefore, the estimated probability loses accuracy further down the tree. Frosst et al. recommend addressing this problem by decaying a running average of the actual probabilities with a time window that is exponentially proportional to the depth of the nodes [128]. Although the authors report that the accuracy of this model was less than the deep neural network, the model offers an advantage of concept interpretability.

Both DNDF and the soft decision tree fix the depth of the learned tree to a predefined value. In contrast, Tanno et al. [129] introduced the Adaptive Neural Tree (ANT) which can grow to any arbitrary depth. The ANT architecture is similar to a decision tree but at each internal node and edge, ANT learns a new data representation. For example, an ANT may contain one or more convolution layers followed by a fully-connected layer at each inner node, one or more convolution layers followed by an activation function such as *ReLU* or *tanh* at each edge, and a linear classifier at each leaf node.

Training an ANT requires two phases: growth and refinement. In the growth phase, starting from the root in breadth-first order one of the nodes is selected. The learner then evaluates three choices: 1) split the node and add a sub-tree, 2) deepen edge transformation by adding another layer of convolution, or 3) keep the current model. The model optimizes the parameters of the newly-added components by minimizing log likelihood via gradient descent while fixing the parameters of the previous portion of the tree. Eventually, the model selects the choice that yields the lowest log likelihood. This process repeats until the model converges. In the refinement phase, the model performs gradient descent on the final architecture. The purpose of the refinement phase is to correct suboptimal decisions that may have occurred during the growth phase. The authors evaluate their method on several standard testbeds and the results indicate that ANT is competitive with many deep network and non-network learners for these tasks.

Yang et al. [130] took a different approach; instead of integrating artificial neurons into the tree, they obtained a decision tree using a neural network. The Deep Neural Decision Tree (DNDT) employs a soft binning function to learn the split rules of the tree. DNDT construct a one-layer neural network with softmax as its activation function. The objective function of this network is:

$$\text{softmax}\big(\frac{wx + b}{\tau}\big). \tag{18}$$

Here, for a continuous variable x, we want to bin it to $n + 1$, $w = [1, 2, \cdots, n + 1]$ is an untrainable constant, b is a learnable bin or the cutting rule in the tree, and τ is a temperature variable. After training this model, the decision tree is constructed via the Kronecker product $\otimes$. Given an input $x \in R^D$ with D features, the tree rule to reach a leaf node is:

$$z = f_1(x_1) \otimes f_2(x_2) \otimes \cdots \otimes f_D(x_D) \tag{19}$$

Here, z is an almost-one-hot encoded vector that indicates the index of leaf node. One of the shortcomings of this method is that it cannot handle a high-dimensional dataset because the cost of calculating the Kronecker product becomes prohibitive. To overcome this problem, authors learn a classifier forest by training each tree on a random subset of features.

In some cases, the mapping function is not differentiable. Feng et al. [131] propose a new learning paradigm for training a multilayer Gradient Boosting decision tree (mGBDT) [131] where F_i is not differentiable. Gradient boosting decision tree (GBDT) is an iterative method which learns an ensemble of regression predictors. In GBDT, a decision tree first learns a model on a training set, then it computes the corresponding error residual for each training sample. A new tree learns a model on the error residuals, and by combining these two trees GBDT is able to learn a more complex model. The algorithm follows this procedure iteratively until it meets a prespecified number of trees for training.

Since gradient descent is not applicable to mGBDT, Feng et al. [131] obtain a "pseudo-inverse" mapping. In this mapping, G_i^t represents the pseudo-inverse of F_i^{t-1} at iteration t, such that $G_i^t(F_i^{t-1}(o_{i-1})) \sim o_{i-1}$. After performing backward propagation and calculating G_i^t, forward propagation is performed by fitting a pseudo-label z_{i-1}^t from G_i^t to F_i^{t-1}. The last layer F_m computes z_m^t based on the true labels at iteration t, where $i \in \{2 \dots m\}$. After this step, pseudo-labels for previous layers are computed via pseudo-inverse mapping. To initialize mGBDT at iteration $t = 0$, each intermediate (hidden) layer outputs Gaussian noise and F_i^0 represent depth-constrained trees that will later be refined. Feng et al. [131] thus create a method that is inspired by gradient descent yet is applicable in situations where true gradient descent cannot be effectively applied.

In this section, we examine methods that apply gradient descent to non-network models. As we observed, one way of utilizing gradient descent is to replace the hidden units in a network with a differentiable algorithm like SVM. Another common theme we recognized was to transform deterministic decision-tree nodes into stochastic versions that offer greater representational power. Alternatively, trees or other ruled-based models can be built using neural networks.

5 Deep Learning Regularization Outside of Deep Neural Networks

We have discussed some of the common regularization methods used by DNNs in Section 2.6. Now we focus on how these methods have been applied to non-network classifiers in the literature. It is worth mentioning that while most models introduced in this section are not deep models, we investigate how non-network models can improve their performance by applying regularization methods typically associated with the deep operations found in DNNs.

5.1 Parameter Norm Penalty

Problems arise when a model is learned from data that contain a large number of redundant features. For example, selecting relevant genes associated with different types of cancer is challenging because of a large number of redundancies may exist in the gene's long string of features. There are two common ways to eliminate redundant features: the first way is to perform feature selection and then train a classifier from the selected features; the second way is to simultaneously perform feature selection and classification. As we discussed in Section 2.6.1, DNNs apply a L_1 or L_2 penalty function to penalize large weights. In this section, we investigate how the traditional DNN idea of penalizing features can be applied to non-network classifiers to simultaneously select high-ranked features and perform classification.

Standard SVMs employ the L_2 norm penalty to penalize weights in a manner similar to DNNs. However, the Newton Linear Programming SVM (NLP-SVM) [132] replaces the L_2 norm penalty with the L_1 norm penalty. This has the effect of setting small hyperparameter coefficients to zero, thus enabling NLP-SVM to select important features automatically. A different way to penalize non-important features in SVMs is to employ a Smoothly Clipped Absolute Deviation (SCAD) [133] function. The L_1 penalty function can be biased because it imposes a larger penalty on large coefficients; in contrast, SCAD can give a nearly unbiased estimation of large coefficients. SCAD learns a non-convex penalty function as shown in Equation 20.

$$p_\lambda(|w|) = \begin{cases} \lambda|w| & \text{if } |w| \leq \lambda \\ -\frac{(|w|^2 - 2a\lambda|w| + \lambda^2)}{2(a-1)} & \text{if } \lambda < |w| \leq a\lambda \\ \frac{(a+1)\lambda^2}{2} & \text{if } |w| > a\lambda \end{cases} \tag{20}$$

SCAD equates with L_1 penalty function until $|w| = \lambda$, then smoothly transitions to a quadratic function until $|w| = a\lambda$, after which it remains a constant for all $|w| > a\lambda$. As shown by Fan et al. [134], SCAD has better theoretical properties than the L_1 function.

One limitation of decision tree classifiers is that the number of training instances that can be selected at each branch in the tree decreases with the tree depth. This downward sampling may cause less relevant or redundant features to be selected near the bottom of the tree. To address this issue, Dang et al. [135] proposed to penalize features that were never selected in the process of making a tree. In a Regularized Random Forest (RRF) [135], the information gain for

a feature j is specified as follows:

$$Gain(j) = \begin{cases} \lambda.Gain(j) & j \notin F \\ Gain(f_i) & j \in F \end{cases} \tag{21}$$

where F is the set of features used earlier in the path, $f_i \in F$, and $\lambda \in [0, 1]$ is the penalty coefficient. RRF avoids including a new feature j, except when the value of $Gain(j)$ is greater than $\max_i \big(Gain(f_i)\big)$.

To improve RRF, Guided RRF (GRRF) [136] assigns a different penalty coefficient λ_j to each feature instead of assigning the same penalty coefficient to all features. GRRF employs the importance score from a pre-trained RF on the training set to refine the selection of features at a given node. The importance score of feature j in an RF with T trees is the mean of gain for features in the RF. The important scores evaluate the contribution of features for predicting classes. The GRRF uses the normalized importance score to control the degree of regularization of the penalty coefficient as follows:

$$\lambda_j = (1 - \gamma)\lambda_0 + \gamma Imp'_j, \tag{22}$$

where $\lambda_0 \in (0, 1]$ is the base penalty coefficient and $\gamma \in [0, 1]$ controls the weight of the normalized importance score. The GRRF and RRF are computationally inexpensive methods that are able to select stronger features and avoid redundant features.

5.2 Dropout

As detailed in Section 2.6.2, dropout is a method that prevents DNNs from overfitting by randomly dropping nodes during the training. Dropout can be added to other machine learning algorithms through two methods: by dropping features or by dropping models in the case of ensemble methods. Dropout has also been employed by dropping input features during training [137] [138]. Here we look at techniques that have been investigated for dropping input features, particularly in non-network classifiers.

Rashmi et al. [139] applied dropout to Multiple Additive Regression Trees (MART) [140] [141]. MART is a regression tree ensemble which iteratively refines its model by continually adding trees that fit the loss function derivatives from the previous version of the ensemble. Because trees added at later iterations may only impact a small fraction of the training set and thus over-specialize, researchers previously used shrinkage to exclude a random subset of leaf nodes during each tree-adding step. More recently, Rashmi et al. integrated the deep-learning idea of dropout into MART. Using dropout, a subset of the trees are temporarily dropped. A new tree is created based on the loss function for the on-dropped trees. This new tree is combined with the previously-dropped trees into a new ensemble. This method, Dropout Multiple Additive Regression Trees (DART) [139], weights the votes for the new and re-integrated trees to have the same effect on the final model output as the original set of trees. Other researchers have experimented with permanently removing a strategic subset of the dropped trees as well [142].

5.3 Early Stopping

The core concept of early stopping is to terminate DNN training once performance on the validation set is not improving. One potential advantage of Deep Forest [93] over DNNs is that DF can determine the depth of a model automatically. In DF, if the model performance does not increase on the validation set after adding a new layer, the learning terminates. Unlike DNNs, DF may avoid the tendency to overfit as more layers are added. Thus, while early stopping does not necessarily enjoy the primary outcome of preventing such overfitting, it can provide additional benefits such as shortening the validation cycle in the search for the optimal tree depth.

5.4 Data Augmentation

As discussed in Section 2.6.3, data augmentation is a powerful method for improving DNN generalization. However, little research has investigated the effects of data augmentation methods on non-network classifiers. As demonstrated by Wong et al. [143], the SVM classifier does not always benefit from data augmentation, in contrast to DNNs. However, Xu [144] ran several data augmentation experiments on synthetic datasets and observed that data augmentation did enhance the performance of random forest classifiers. Offering explanations for the circumstances in which such augmentation is beneficial is a needed area for future research.

6 Hybrid Models

Hybrid models can be defined as a combination of two or more classes of models. There are many ways to construct hybrid models, such as DNDF [127] which integrates a deep network into a decision forest as explained in Section 4. In this section we discuss other examples of hybrid models.

One motivation for combining aspects of multiple models is to find a balance between classification accuracy and computational cost. Energy consumption by mobile devices and cloud servers is an increasing concern for responsive applications and green computing. Decision forests are computationally inexpensive models because of the conditional property of decision trees. Conversely, while CNNs are less efficient, they can achieve higher accuracy because of their representation-learning capabilities. Ioannou et al. [145] introduced the Conditional Neural Network (CondNN) to reduce computation in a CNN model by introducing a routing method similar to that found in decision trees. In CondNN, each node in layer l is connected to a subset of nodes from the previous layer, $l-1$. Given a fully trained network, for every two consecutive layers a matrix $\Lambda_{(l-1,l)}$ stores the activation values of these two layers. By rearranging elements of $\Lambda_{(l-1,l)}$ based on highly-active pairs for each class in the diagonal and zeroing out off-diagonal elements, the CondNN develops explicit routes $\Lambda^{route}_{(l,l-1)}$ through nodes in the network. CondNN incurs profoundly lower computation cost compared to other DNNs at test time; whereas, CondNN's accuracy remains similar to larger models. We note that DNN size can be also be reduced by employing Bayesian optimization, as investigated by Blundell et al. [146] and by Fortunato et al. [147]. These earlier efforts provide evidence that Bayesian neural networks are able to decrease network size even more than CondNNs while maintaining a similar level of accuracy.

Another direction for blending a deep network with a non-network classifier is to improve the non-network model by learning a better representation of data via a deep network. Zoran et al. [148] introduce the differentiable boundary tree (DBT) in order to integrate a DNN into the boundary tree [149] to learn a better representation of data. The newly-learned data representation leads to a simpler boundary tree because the classes are well separated. The boundary tree is an online algorithm in which each node in the tree corresponds to a sample in the training set. The first sample together with its label are established as the tree root. Given a new query sample z, the sample traverses through the tree from the root to find the closest node n based on some distance function like the Euclidean distance function. If the label of the nearest node in the tree is different from the query sample, a new node containing the query z is added as a child of the closest node n in the tree; otherwise the query node z is discarded. Therefore, each edge in the tree marks the boundary between two classes and each node tends to be close to these boundaries.

Transitions between nodes in a standard boundary tree are deterministic. DBT combines a SoftMax cost function with a boundary tree, resulting in stochastic transitions. Let x be a training sample and c be the one-hot encoding label of that sample. Given the current node x_i in the tree and a query node z, the transition probability from node x_i to node x_j, where $x_j \in \{child(x_i), x_i\}$ is the SoftMax of the negative distance between x_j and z. This is shown in Equation 23.

$$\begin{aligned} p(x_i \rightarrow x_j|z) &= \underset{i,j \in child(i)}{\text{SoftMax}}\left(-d(x_j, z)\right) \\ &= \frac{\exp(-d(x_j, z))}{\sum\limits_{j' \in \{i, j \in child(i)\}} \exp(-d(x_j, z))} \end{aligned} \tag{23}$$

The probability of traversing a particular path in the boundary tree, given a query node z, is the product of the probability of each transition along the path from the root to the final node x_{final^*} in the tree. The final class log probability of DBT is computed by summing the probabilities of all transitions to the parent of x_{final^*} together with the probabilities of the final node and its siblings. The set $sibling(x_i)$ consists of all nodes sharing the same parent with node x_i and the node x_i itself. As discussed earlier, a DNN $f_\theta(x)$ transforms the inputs to learn a better representation. The final class log probabilities for the query node z are calculated as follows:

$$\begin{aligned} \log p(c|f_\theta(z)) = &\sum_{x_i \rightarrow x_j \in path^\dagger | f_\theta(z)} \log p(f_\theta(x_i) \rightarrow f_\theta(x_j)|f_\theta(z)) \\ &+ \log \sum_{x_k \in sibling(x_{final^*})} p(parent(f_\theta(x_k)) \rightarrow f_\theta(x_k)|f_\theta(z))c(x_k). \end{aligned} \tag{24}$$

In Equation 24, $path^\dagger$ denotes $path^*$ (the path to the final node x_{final^*}) without the last transition, and $sibling(x)$ represents node x and all other nodes sharing the same parent with node x. The gradient descent algorithm can be applied to Equation 24 by plugging in a loss function to learn parameter θ of the DNN. However, gradient descent cannot be applied easily to DBT because of the node and edge manipulations in the graph. To address this issue, DBT transforms a small subset of training examples via a DNN and builds a boundary tree based on the transformed examples. Next, DBT transforms a query node z via the same DNN and calculates the log probability of a class

according to Equation 24. The DNN employs gradient descent to update its parameters by propagating the gradient of log loss probability. DBT discards this boundary tree and iteratively builds a new boundary tree as described until a convergence criteria is met. In the described method, the authors set a specific threshold for the loss value to terminate the training. DBT is able to achieve greater accuracy with a simpler tree than original boundary tree as shown by the authors on the MNIST dataset [97]. One of the biggest advantages of DBT is its interpretability. However, DBT is computationally an expensive method because a new computation graph needs to be built, which makes batching inefficient. Another limitation is that the algorithm needs to switch between building the tree and updating the tree. Therefore, scaling to large datasets is fairly prohibitive.

Yet another way of building a hybrid model is to learn a new representation of data with a DNN, then hand the resulting feature vectors off to other classifiers to learn a model. Tang [150] explored replacing the last layer of DNNs with a linear SVM for classification tasks. The activation values of the penultimate layer are fed as input to an SVM with a L_2 regularizer. The weights of the lower layer are learned through momentum gradient descent by differentiating the SVM objective function with respect to activation of the penultimate layer. The author's experiments on the MNIST [97] and CIFAR-10 [151] datasets demonstrate that replacing a CNN's SoftMax output layer with SVM yields a lower test error. Tang et al. [150] postulate that the performance gain is due to the superior regularization effect of the SVM loss function.

It is worth mentioning that in their experiment on MNIST [97], Tang first used PCA to reduce the features and then fed the reduced feature vectors as input to their model. Also, Niu et al. [152] replaced the last layer of a CNN with an SVM which similarly resulted in lowering test error of the model compare to a CNN on the MNIST dataset. Similar to these methods, Zareapoor et al. [153], Nagi et al. [154], Bellili et al. [155], and Azevedo et al. [156] replace the last layer of a DNN with an SVM. In these cases, their results from multiple datasets reveal that employing a SVM as the last layer of a neural network can improve the generalization of the network.

Zhao et al. [157] replace the last layer of a deep network with a visual hierarchical tree to learn a better solution for image classification problems. A visual hierarchical tree with L levels organizes N objects classes based on their visual similarities in its nodes. Deeper in the tree, groups become more separated wherein each leaf node should contain instances of one class. The class similarity between the class c_i and c_j is defined as follows:

$$S_{i,j} = S(c_i, c_j) = \exp\Big(- \frac{d(x_i, x_j)}{\sigma}\Big). \tag{25}$$

Here, $d(x_i, x_j)$ represents the distance between the deep representation of instances of classes c_i and c_j, and σ is automatically determined by a self-tuning technique. After calculating matrix S, hierarchical clustering is employed to learn a visual hierarchical tree.

In a traditional visual hierarchical tree, some objects might be assigned to incorrect groups. A level-wise mixture model (LMM) [157] aims to improve this visual hierarchical tree by learning a new representation of data via a DNN then updating the tree during training. For a given tree, matrix Ψ_{y_i,t_i} denotes the probability of objects with label y belonging to group t in the tree. First, LMM updates the DNN parameters and the visual hierarchical tree as is done with a traditional DNN. The only difference is a calculation of two gradients, one based on the parameters of the DNN and other one based on the parameters of the tree. Second, LMM updates the matrix Ψ_{y_i,t_i} for each training sample separately and updates the parameters of the DNN and the tree afterwards. To update the Ψ, the posterior probability of the assigning group t_i for the object x_i is calculated based on the number of samples having the same label y as the label of x_i in group t. For a given test image, LMM learns a new representation of the image based on the DNN and then obtains a prediction by traversing the tree. One of the advantages of a LMM is that over time, by learning a better representation of data via DNN, the algorithm can update the visual hierarchical tree.

In some cases, two different data views are available. As an example, one view might contain video and the another sound. Canonical correlation analysis (CCA) [158] and kernel canonical correlation analysis (KCCA) [159] offer standard statistical methods for learning view representations that each the most predictable by the other view. Nonlinear representations learned by KCCA can achieve a higher correlation than linear representations learned by CCA. Despite the advantages of KCCA, the kernel function faces some drawbacks. Specifically, the representation is bound to the fixed kernel. Furthermore, because is model is nonparametric, training time as well as the time to compute the new data representation scales poorly with the size of a training set.

Andrews et al. [160] proposed to apply deep networks to learn a nonlinear data representation instead of employing a kernel function. Their resulting deep canonical correlation analysis (DCCA) consists of two separate deep networks for learning a new representation for each view. The new representation learned by the final layer of networks H_1 and H_2 is fed to CCA. To compute the objective gradient of DCCA, the gradient of the output of the correlation objective

with respect to the new representation can be calculated as follows:

$$\frac{\partial_{corr}(H_1, H_2)}{\partial H_1} \tag{26}$$

After this computation, backpropagation is applied to find the gradient with respect to all parameters. The details of calculating the gradient in Equation 26 are provided by the authors [160].

While researchers have also created LSTM methods that employ tree structures [161] [162], these methods utilize the data structure to improve a network model rather than employing tree-based learning algorithms. Similarly, other researches integrate non-network classifiers into a network structure. Cimino et al. [163] and Agarap [164] introduce hybrid models. These two methods apply LSTM and GRU, respectively, to learn a network representation. Unlike traditional DNNs, the last layer employs a SVM for classification.

The work surveyed in this section provides evidence that deep neural nets are capable methods for learning high-level features. These features, in turn, can be used to improve the modeling capability for many types of supervised classifiers.

Table 1: Summary of classifiers which integrate deep network components into non-network classifiers.

	Methods	**Classifiers**
Architecture	Feedforward	ANT [129], DNDT [130], DBN [45], Deep PCA [105], DF [93], DPG [116], R2-SVM [101], D-SVM [100], DTA-LS-SVM [94], SFDT [128]
	Autoencoder	DKF [122], eForest [125], ML-SVM [126]
	Siamese Model	SDF [119]
	Generative Adversarial Model	GAF [120]
Optimization	Gradient Decent	DNDF [127], mGBDT [131], ML-SVM [126]
Regularization	Parameter Norm Penalty	NLP-SVM [132], GRRF [136], RRF [135] , SCAD-SVM [133]
	Dropout	DART [139]
Hybrid Model		CondCNN [145], DBT [148], DCCA [160], DNDF [127], DNN+SVM [150] [152] [153] [154] [155] [156], LMM [157]

In this survey, we aim to provide a thorough review of non-network models that utilize the unique features of deep network models. Table 1 provides a summary of such non-network models, organized based on four aspects of deep networks: model architecture, optimization, regularization, and hybrid model fusing. A known advantage of traditional deep networks compared with non-network models has been the ability to learn a better representation of input features. Inspired by various deep network architectures, deep learning of non-network classifiers has resulted in methods to also learn new feature representations. Another area where non-network classifiers have benefited from recent deep network research is applying backpropagation optimization to improve generalization. This table summarizes published efforts to apply regularization techniques that improve neural network generalization. The last category of models combines deep network classifiers and non-network classifiers to increase overall performance.

7 Experiments

In this paper, we survey a wide variety of models and methods. Our goal is to demonstrate that diverse types of models can benefit from deep learning techniques. To highlight this point, we empirically compare the performance of many techniques described in this survey. This comparison includes deep and shallow networks as well as and non-network learning algorithms. Because of the variety of classifiers that are surveyed, we organize the comparison based on the learned model structure.

First, we compare the models that are most similar to DNNs. These models should be able to learn a better representation of data when a large dataset is available. We report the test error provided by the authors for MNIST and CIFAR-10 dataset in Table 2. If the performance of a model was not available for any of these datasets, we ran that experiment with the authors' code. Default parameters are employed for parameter values that are not specified in the original papers. In the event that the authors did not provide their code, we did not report any results. These omissions prevent the report of erroneous performances that result from implementation differences.

The MNIST dataset has been a popular testbed dataset for comparing model choices within the computer vision and deep network communities. MNIST instances contain 28×28 pixel grayscale images of handwritten digits and their labels. The MNIST labels are drawn from 10 object classes, with a total of 6000 training samples and 1000 testing samples. The CIFAR-10 is also a well-known dataset containing 10 object classes with approximately 5000 examples per class, where each sample is a 32×32 pixel RGB image.

Table 2: Classification error rate (%) comparison.

Dataset/**Models**	**RF**	**ANT**† [129]	**DF** [93]	**R2SVM** [101]	**SFDT** [128]	**DNDF** [127]	**CondCNN** [145]	**DBT** [148]	**DNN+SVM** [150]
MNIST	3.21	0.29	0.74	4.03*	5.55	0.7	-	1.85	0.87
CIFAR-10	50.17	7.71	31.0	20.3	-	-	10.12	13.06	11.9

* Based on the authors' code.
† The reported result reflects an ANT ensemble.

In the next set of experiments, we compare models designed to handle small or structured datasets, as shown in Table 3. The authors of these methods tested a wide range of datasets to evaluate their their approach. In this survey, we conducted a series of experiments on the UCI human activity recognition (HAR) dataset [165]. Here, human activity recognition data were collected from 30 participants performing six scripted activities (walking, walking upstairs, walking downstairs, sitting, standing, and laying) while wearing smartphones. The dataset contains 561 features extracted from sensors including an accelerometer and a gyroscope. The training set contains 7352 samples from 70% of the volunteers and the testing set contains 2947 samples from the remaining 30% of volunteers.

Table 3: Classification error rate (%) comparison.

Dataset/**Models**	**RF**	**SVM**	**MLP**	**DART** [139]	**RRF** [135]	**GRRF** [136]	**mGBDT** [131]
HAR	6.96	4.69	4.69	6.55	3.77	3.74	7.68

From the table, we observe that models representing multiple layers of machine learning models such as DF, R2SVM, and mGBDT did not perform well on the MNIST and CIFAR-10 datasets. Compared to DNNs, these models are computationally more expensive, require an excessive amount of resources, and do not offer advantages of interpretability.

Another category of models utilizes a neural network to learn a better representation of data. These models such as DNDF, and DNN+SVM applied a more traditional machine learning model on the newly-learned representation. This technique could be beneficial when the neural network has been trained on a large dataset. For example, DNDF utilized GoogLeNet for extracting features, and subsequently achieved a 6.38% error rate on the ImageNet testset. In contrast, the GoogLeNet error rate is 10.02%. Another class of models enhances the tree model by integrating artificial neurons such as ANT, SFDT, and DBT. These models cleverly combine neural networks with decision trees that improve interpretation while offering representation-learning benefits.

Another hybrid strategy focused on decreasing the computational complexity of the DNNs. CondCNN is such a neural network that employs a routing mechanism similar to a decision tree to achieve this goal. Another successful line of research is to add regularizers frequently used by the neural network to other classifiers similar to DART, RRF, and GRRF.

The results from our experiments reveal that both network classifiers and non-network classifiers benefit from deep learning. The methods surveyed in this paper and evaluated in these experiments demonstrate that non-network machine learning models do improve performance by incorporating DNN components into their algorithms. Whereas models without feature learning such as RF usually do not perform well on unstructured data such as images, we observe that adding deep learning to these models drastically improve their performance, as shown in Table 2. Additionally, non-deep models may achieve improved performance on structured data by adding regularizers, as shown in Table 3. The methods surveyed in this paper demonstrate that deep learning components can be added to any type of machine learning model, and are not specific to DNNs. The incorporation of deep learning strategies is a promising direction for all types of classifiers, both network and non-network methods.

8 Conclusions and Directions for Ongoing Research

DNNs have emerged as a powerful force in the machine learning field for the past few years. This survey paper reviews the latest attempts to incorporate methods that are traditionally found in DNNs into other learning algorithms. DNNs work well when there is a large body of training data and available computational power. DNNs have consistently yielded strong results for a variety of datasets and competitions, such as winning the Large Scale Visual Recognition Challenge [166] and achieving strong results for energy demand prediction [167], identifying gender of a text author [168], stroke prediction [169], network intrusion detection [170], speech emotion recognition [171], and taxi destination prediction [172]. Since there are many applications which lack large amounts of training data or for which the interpretability of a learned model is important, there is a need to integrate the benefits of DNNs with other classifier

algorithms. Other classifiers have demonstrated improved performance on some types of data, therefore the field can benefit from examining ways of combining deep learning elements between network and non-network methods.

Although some work to date provides evidence that DNN techniques can be used effectively by other classifiers, there are still many challenges that researchers need to address, both to improve DNNs and to extend deep learning to other types of classifiers. Based on our survey of existing work, some related areas where supervised learners can benefit from unique DNN methods are outlined below.

The most characteristic feature of DNNs is a deep architecture and its ability to learn a new representation of data. A variety of stacked generalization methods have been developed to allow other machine learning methods to utilize deep architectures as well. These methods incorporate multiple classification steps in which the input of the next layer represents the concatenation of the output of the previous layer and the original feature vector as discussed in Section 3.1.1. Future work can explore the many other possibilities that exist for refining the input features to each layer to better separate instances of each class at each layer.

Previous studies provide evidence that DNNs are effective data generators [173] [174], while in some cases non-network classifiers may actually be the better discriminators. Future research can consider using a DNN as a generator and an alternative classifier as a discriminator in generative adversarial models. Incorporating this type of model diversity could improve the robustness of the models.

Gradient descent can be applied to any differentiable algorithm. We observed that Kontschieder et al. [127], Frosst et al. [128], Tanno et al. [129], and Zoran et al. [148] all applied gradient descent to two different tree-based algorithms by making them differentiable. In the future, additional classifiers can be altered to be differentiable. Applying gradient descent to other algorithms could be an effective way to adjust the probability distribution of parameters.

Another area which is vital to investigate is the application of network-customized regularization methods unique to non-network classifiers. As discussed in Section 5, the non-network classifiers can benefit from the regularization methods that are unique to DNNs. However, there exist many different ways that these regularization methods can be adapted by non-network classifiers to improve model generalization.

An important area of research is interpretable models. There exist applications such as credit score, insurance risk, health status because of their sensitivity, models need to be interpretable. Further research needs to exploit the use of DNNs in interpretable models such as DNDT [130].

As we discussed in this survey, an emerging area of research is to combine the complementary benefits of statistical models with neural networks. Statistical models offer mathematical formalisms as well as possible explanatory power. This combination may provide a more effective model than either approach used in isolation.

There are cases in which the amount of ground truth-labeled data is limited, but a large body of labeled data from the same or similar distribution is available. One possible area of ongoing exploration is to couple the use of DNNs for learning from unlabeled data with the use of other classifier strategies for learning from labeled data. The simple model learned from labeled data can be exploited to further tune and improve learned representation patterns in the DNN.

We observe that currently, there is a general interest among the machine learning community to transfer new deep network developments to other classifiers. While a substantial effort has been made to incorporate deep learning ideas into the general machine learning field, continuing this work may spark the creation of new learning paradigms. However, the benefit between network-based learners and non-network learners can be bi-directional. Because a tremendous variety of classifiers has shown superior performance for a wide range of applications, future research can focus not only on how DNN techniques can improve non-network classifiers but on how DNNs can incorporate and benefit from non-network learning ideas as well.

Acknowledgment

The authors would like to thank Tharindu Adikari, Chris Choy, Ji Feng, Yani Ioannou, Stanislaw Jastrzebski and Marco A. Wiering for their valuable assistance in providing code and additional implementation details of the algorithms that were evaluated in this paper. We would also like to thank Samaneh Aminikhanghahi and Tinghui Wang for their feedback and guidance on the methods described in this survey. This material is based upon work supported by the National Science Foundation under Grant No. 1543656.

Table 4: The list of abbreviations and their descriptions utilized in this survey.

Abbreviation	Description
AE	Autoencoder
ANT	Adaptive Neural Tree
CNN	Convolutional Neural Network
CondNN	Conditional Neural Network
DART	Dropout Multiple Additive Regression Trees
DBT	Differentiable Boundary Tree
DBN	Deep Belief Network
DCCA	Deep Canonical Correlation Analysis
Deep PCA	Deep principal components analysis
DF	Deep Forest
DGP	Deep Gaussian Processes
DKF	Deep Kalman Filters
DNDT	Deep Network Decision Tree
DNDF	Deep Network Decision Forest
DNN	Deep Neural Network
DSVM	Deep SVM
DTA-LS-SVM	Deep Transfer Additive Kernel Least Square SVM
eForest	Encoder Forest
FC	Fully Connected
GAF	Generative Adversarial Forest
GAN	Generative Adversarial Network
GRRF	Guided Regularized Random Forest
LMM	Level-wise Mixture Model
mGBDT	Multilayer Gradient Decision Tree
ML-SVM	Multilayer SVM
MLP	Multilayer perceptron
NLP-SVM	Newton Linear Programming SVM
R2-SVM	Random Recursive SVM
RBM	Restricted Boltzmann Machine
RNN	Recurrent Neural Network
RRF	Regularized Random Forest
SCAD-SVM	Smoothly Clipped Absolute Deviation SVM
SDF	Siamese Deep Forest
SNN	Siamese Neural Network
FSDT	Frosst Soft Decision Tree
VAE	Variational Autoencoder

References

[1] Cade Metz. AI is transforming google search. the rest of the web is next, 2016.

[2] Nathan Sikes. Deep learning and the future of search engine optimization, 2015.

[3] Alex Davies. The numbers don't lie: Self-driving cars are getting good, 2017.

[4] Neal Boudette. Tesla's self-driving system cleared in deadly crash, 2017.

[5] Autonomous vehicle disengagement reports 2017, 2017.

[6] Fei Jiang, Yong Jiang, Hui Zhi, Yi Dong, Hao Li, Sufeng Ma, Yilong Wang, Qiang Dong, Haipeng Shen, and Yongjun Wang. Artificial intelligence in healthcare: past, present and future. *Stroke and vascular neurology*, 2(4):230–243, 2017.

[7] Liang-Chieh Chen and Yukun Zhu. Semantic image segmentation with deeplab in tensorflow, 2018.

[8] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International Conference on Machine Learning*, pages 1058–1066, 2013.

[9] Benjamin Graham. Fractional max-pooling. *arXiv preprint arXiv:1412.6071*, 2014.

[10] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[11] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701, 2015.

[12] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[13] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4945–4949. IEEE, 2016.

[14] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International Conference on Machine Learning*, pages 173–182, 2016.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[16] Thuy Ong. Amazon's new algorithm designs clothing by analyzing a bunch of pictures, 2017.

[17] Chenghui Tang, Yishen Wang, Jian Xu, Yuanzhang Sun, and Baosen Zhang. Efficient scenario generation of multiple renewable power plants considering spatial and temporal correlations. *Applied Energy*, 221:348–357, 2018.

[18] Rich Caruana, Yin Lou, Johannes Gehrke, Paul Koch, Marc Sturm, and Noemie Elhadad. Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1721–1730. ACM, 2015.

[19] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 427–436, 2015.

[20] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[21] David Krueger, Nicolas Ballas, Stanislaw Jastrzebski, Devansh Arpit, Maxinder S Kanwal, Tegan Maharaj, Emmanuel Bengio, Asja Fischer, and Aaron Courville. Deep nets don't learn via memorization. 2017.

[22] Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

[23] Siddharth Krishna Kumar. On weight initialization in deep neural networks. *arXiv preprint arXiv:1704.08863*, 2017.

[24] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[25] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[26] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[27] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[28] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. IEEE, 2013.

[29] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

[30] David H Wolpert, William G Macready, et al. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[31] Ross D. King, Cao Feng, and Alistair Sutherland. Statlog: comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence an International Journal*, 9(3):289–333, 1995.

[32] Tjen-Sien Lim, Wei-Yin Loh, and Yu-Shan Shih. A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine learning*, 40(3):203–228, 2000.

[33] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

[34] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *Proceedings of the 25th international conference on Machine learning*, pages 96–103. ACM, 2008.

[35] Philipp Baumann, DS Hochbaum, and YT Yang. A comparative study of the leading machine learning techniques and two new optimization algorithms. *European journal of operational research*, 272(3):1041–1057, 2019.

[36] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and SS Iyengar. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys (CSUR)*, 51(5):92, 2018.

[37] Benjamin Shickel, Patrick James Tighe, Azra Bihorac, and Parisa Rashidi. Deep ehr: A survey of recent advances in deep learning techniques for electronic health record (ehr) analysis. *IEEE journal of biomedical and health informatics*, 22(5):1589–1604, 2018.

[38] Junwei Han, Dingwen Zhang, Gong Cheng, Nian Liu, and Dong Xu. Advanced deep-learning techniques for salient and category-specific object detection: a survey. *IEEE Signal Processing Magazine*, 35(1):84–100, 2018.

[39] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 119:3–11, 2019.

[40] William Grant Hatcher and Wei Yu. A survey of deep learning: platforms, applications and emerging research trends. *IEEE Access*, 6:24411–24432, 2018.

[41] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[42] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

[43] Marvin Minsky and Seymour A Papert. *Perceptrons: An introduction to computational geometry*. MIT press, 1969.

[44] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[45] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[46] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[47] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3856–3866. Curran Associates, Inc., 2017.

[48] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[49] Yann LeCun et al. Generalization and network design strategies. *Connectionism in perspective*, pages 143–155, 1989.

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[51] Yi-Tong Zhou and Rama Chellappa. Computation of optical flow using a neural network. In *IEEE International Conference on Neural Networks*, volume 1998, pages 71–78, 1988.

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

[53] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[54] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[55] Rachid Riad, Corentin Dancette, Julien Karadayi, Neil Zeghidour, Thomas Schatz, and Emmanuel Dupoux. Sampling strategies in siamese networks for unsupervised speech representation learning. *arXiv preprint arXiv:1804.11297*, 2018.

[56] Zara Alaverdyan, Julien Jung, Romain Bouet, and Carole Lartizien. Regularized siamese neural network for unsupervised outlier detection on brain multiparametric magnetic resonance imaging: application to epilepsy lesion screening. 2018.

[57] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[58] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[59] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. *arXiv preprint arXiv:1605.05396*, 2016.

[60] Artur Kadurin, Alexander Aliper, Andrey Kazennov, Polina Mamoshina, Quentin Vanhaelen, Kuzma Khrabrov, and Alex Zhavoronkov. The cornucopia of meaningful leads: Applying deep adversarial autoencoders for new molecule development in oncology. *Oncotarget*, 8(7):10883, 2017.

[61] Wengling Chen and James Hays. Sketchygan: Towards diverse and realistic sketch to image synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9416–9425, 2018.

[62] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Gaugan: semantic image synthesis with spatially adaptive normalization. In *ACM SIGGRAPH 2019 Real-Time Live!*, page 2. ACM, 2019.

[63] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[64] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[65] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer, 2016.

[66] Detian Huang, Weiqin Huang, Zhenguo Yuan, Yanming Lin, Jian Zhang, and Lixin Zheng. Image super-resolution algorithm based on an improved sparse autoencoder. *Information*, 9(1):11, 2018.

[67] Chih-Kuan Yeh, Wei-Chieh Wu, Wei-Jen Ko, and Yu-Chiang Frank Wang. Learning deep latent space for multi-label classification. In *AAAI*, pages 2838–2844, 2017.

[68] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[69] Yoshua Bengio. Deep learning of representations: Looking forward. In *International Conference on Statistical Language and Speech Processing*, pages 1–37. Springer, 2013.

[70] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[71] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[72] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[73] Luis Miguel Rios and Nikolaos V Sahinidis. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2013.

[74] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in neural information processing systems*, pages 1037–1045, 2016.

[75] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[76] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[77] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285, 1995.

[78] Mohammad Moghimi, Serge J Belongie, Mohammad J Saberian, Jian Yang, Nuno Vasconcelos, and Li-Jia Li. Boosted convolutional neural networks. In *BMVC*, 2016.

[79] Jesse Eickholt and Jianlin Cheng. Dndisorder: predicting protein disorder using boosting and deep networks. *BMC bioinformatics*, 14(1):88, 2013.

[80] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[81] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[82] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.

[83] Navdeep Jaitly and Geoffrey E Hinton. Vocal tract length perturbation (vtlp) improves speech recognition. In *Proc. ICML Workshop on Deep Learning for Audio, Speech and Language*, volume 117, 2013.

[84] Hiroki OHASHI, Mohammad AL-NASER, Sheraz AHMED, Takayuki AKIYAMA, Takuto SATO, Phong NGUYEN, Katsuyuki NAKAMURA, and Andreas DENGEL. Augmenting wearable sensor data with physical constraint for dnn-based human-action recognition. 2017.

[85] Christopher Bowles, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger Gunn, Alexander Hammers, David Alexander Dickie, Maria Valdés Hernández, Joanna Wardlaw, and Daniel Rueckert. Gan augmentation: Augmenting training data using generative adversarial networks. *arXiv preprint arXiv:1810.10863*, 2018.

[86] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.

[87] Javier Jorge, Jesús Vieco, Roberto Paredes, Joan-Andreu Sánchez, and José-Miguel Benedí. Empirical evaluation of variational autoencoders for data augmentation. In *VISIGRAPP (5: VISAPP)*, pages 96–104, 2018.

[88] Xiaofeng Liu, Yang Zou, Lingsheng Kong, Zhihui Diao, Junliang Yan, Jun Wang, Site Li, Ping Jia, and Jane You. Data augmentation via latent space interpolation for image classification. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 728–733. IEEE, 2018.

[89] Yichuan Tang and Chris Eliasmith. Deep networks for robust visual recognition. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1055–1062. Citeseer, 2010.

[90] Ben Poole, Jascha Sohl-Dickstein, and Surya Ganguli. Analyzing noise in autoencoders and deep networks. *arXiv preprint arXiv:1406.1831*, 2014.

[91] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[92] Kai Ming Ting and Ian H Witten. Issues in stacked generalization. *Journal of artificial intelligence research*, 10:271–289, 1999.

[93] Zhi-Hua Zhou and Ji Feng. Deep forest: Towards an alternative to deep neural networks. *arXiv preprint arXiv:1702.08835*, 2017.

[94] Guanjin Wang, Guangquan Zhang, Kup-Sze Choi, and Jie Lu. Deep additive least squares support vector machines for classification with model transfer. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2017.

[95] Leo Breiman. Randomizing outputs to increase prediction accuracy. *Machine Learning*, 40(3):229–242, 2000.

[96] Tin Kam Ho. Random decision forests. In *Document analysis and recognition, 1995., proceedings of the third international conference on*, volume 1, pages 278–282. IEEE, 1995.

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

[98] Gavin C Cawley. Leave-one-out cross-validation based model selection criteria for weighted ls-svms. In *Neural Networks, 2006. IJCNN'06. International Joint Conference on*, pages 1661–1668. IEEE, 2006.

[99] Hao Yang and Jianxin Wu. Practical large scale classification with additive kernels. In *Asian Conference on Machine Learning*, pages 523–538, 2012.

[100] Azizi Abdullah, Remco C Veltkamp, and Marco A Wiering. An ensemble of deep support vector machines for image categorization. In *Soft Computing and Pattern Recognition, 2009. SOCPAR'09. International Conference of*, pages 301–306. IEEE, 2009.

[101] Oriol Vinyals, Yangqing Jia, Li Deng, and Trevor Darrell. Learning with recursive perceptual representations. In *Advances in Neural Information Processing Systems*, pages 2825–2833, 2012.

[102] Li Deng, Dong Yu, and John Platt. Scalable stacking and learning for building deep architectures. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 2133–2136. IEEE, 2012.

[103] Brian Hutchinson, Li Deng, and Dong Yu. Tensor deep stacking networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1944–1957, 2013.

[104] Li Deng and Dong Yu. Deep convex net: A scalable architecture for speech pattern classification. In *Twelfth Annual Conference of the International Speech Communication Association*, 2011.

[105] Venice Erin Liong, Jiwen Lu, and Gang Wang. Face recognition using deep pca. In *Information, Communications and Signal Processing (ICICS) 2013 9th International Conference on*, pages 1–5. IEEE, 2013.

[106] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[107] Jonathon Shlens. A tutorial on principal component analysis. *arXiv preprint arXiv:1404.1100*, 2014.

[108] Andreas Damianou and Neil Lawrence. Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215, 2013.

[109] Ilya Sutskever and Geoffrey Hinton. Learning multilevel distributed representations for high-dimensional sequences. In *Artificial intelligence and statistics*, pages 548–555, 2007.

[110] Roland Memisevic and Geoffrey Hinton. Unsupervised learning of image transformations. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[111] Kevin Swersky, Ilya Sutskever, Daniel Tarlow, Richard S Zemel, Ruslan R Salakhutdinov, and Ryan P Adams. Cardinality restricted boltzmann machines. In *Advances in neural information processing systems*, pages 3293–3301, 2012.

[112] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[113] César Lincoln C Mattos, Zhenwen Dai, Andreas Damianou, Jeremy Forth, Guilherme A Barreto, and Neil D Lawrence. Recurrent gaussian processes. *arXiv preprint arXiv:1511.06644*, 2015.

[114] Mark Van der Wilk, Carl Edward Rasmussen, and James Hensman. Convolutional gaussian processes. In *Advances in Neural Information Processing Systems*, pages 2849–2858, 2017.

[115] Zhenwen Dai, Andreas Damianou, Javier González, and Neil Lawrence. Variational auto-encoded deep gaussian processes. *arXiv preprint arXiv:1511.06455*, 2015.

[116] Andreas Damianou. *Deep Gaussian processes and variational propagation of uncertainty*. PhD thesis, University of Sheffield, 2015.

[117] Matthew M Dunlop, Mark A Girolami, Andrew M Stuart, and Aretha L Teckentrup. How deep are deep gaussian processes? *The Journal of Machine Learning Research*, 19(1):2100–2145, 2018.

[118] David Duvenaud, Oren Rippel, Ryan Adams, and Zoubin Ghahramani. Avoiding pathologies in very deep networks. In *Artificial Intelligence and Statistics*, pages 202–210, 2014.

[119] Lev V Utkin and Mikhail A Ryabinin. A siamese deep forest. *arXiv preprint arXiv:1704.08715*, 2017.

[120] Yan Zuo, Gil Avraham, and Tom Drummond. Generative adversarial forests for better conditioned adversarial learning. *arXiv preprint arXiv:1805.05185*, 2018.

[121] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[122] Rahul G. Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

[123] Shirli Di-Castro Shashua and Shie Mannor. Deep robust kalman filter. *arXiv preprint arXiv:1703.02310*, 2017.

[124] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Zhiyong Gao, and Ming-Ting Sun. Deep kalman filtering network for video compression artifact reduction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 568–584, 2018.

[125] Ji Feng and Zhi-Hua Zhou. Autoencoder by forest. *arXiv preprint arXiv:1709.09018*, 2017.

[126] Marco A Wiering and Lambert RB Schomaker. Multi-layer support vector machines. *Regularization, optimization, kernels, and support vector machines*, page 457, 2014.

[127] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Computer Vision (ICCV), 2015 IEEE International Conference on*, pages 1467–1475. IEEE, 2015.

[128] Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *arXiv preprint arXiv:1711.09784*, 2017.

[129] Ryutaro Tanno, Kai Arulkumaran, Daniel C Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. *arXiv preprint arXiv:1807.06699*, 2018.

[130] Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018.

[131] Ji Feng, Yang Yu, and Zhi-Hua Zhou. Multi-layered gradient boosting decision trees. *arXiv preprint arXiv:1806.00007*, 2018.

[132] Glenn M Fung and Olvi L Mangasarian. A feature selection newton method for support vector machine classification. *Computational optimization and applications*, 28(2):185–202, 2004.

[133] Hao Helen Zhang, Jeongyoun Ahn, Xiaodong Lin, and Cheolwoo Park. Gene selection using support vector machines with non-convex penalty. *bioinformatics*, 22(1):88–95, 2005.

[134] Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association*, 96(456):1348–1360, 2001.

[135] Houtao Deng and George Runger. Feature selection via regularized trees. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.

[136] Houtao Deng and George Runger. Gene selection with guided regularized random forest. *Pattern Recognition*, 46(12):3483–3489, 2013.

[137] Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 90–94. Association for Computational Linguistics, 2012.

[138] Sida Wang and Christopher Manning. Fast dropout training. In *international conference on machine learning*, pages 118–126, 2013.

[139] Rashmi Korlakai Vinayak and Ran Gilad-Bachrach. Dart: Dropouts meet multiple additive regression trees. In *Artificial Intelligence and Statistics*, pages 489–497, 2015.

[140] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[141] Jerome H Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[142] Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, and Salvatore Trani. X-dart: Blending dropout and pruning for efficient learning to rank. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1077–1080. ACM, 2017.

[143] Sebastien C Wong, Adam Gatt, Victor Stamatescu, and Mark D McDonnell. Understanding data augmentation for classification: when to warp? *arXiv preprint arXiv:1609.08764*, 2016.

[144] Ruo Xu. Improvements to random forest methodology. 2013.

[145] Yani Ioannou, Duncan Robertson, Darko Zikic, Peter Kontschieder, Jamie Shotton, Matthew Brown, and Antonio Criminisi. Decision forests, convolutional networks and the models in-between. *arXiv preprint arXiv:1603.01250*, 2016.

[146] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.

[147] Meire Fortunato, Charles Blundell, and Oriol Vinyals. Bayesian recurrent neural networks. *CoRR*, abs/1704.02798, 2017.

[148] Daniel Zoran, Balaji Lakshminarayanan, and Charles Blundell. Learning deep nearest neighbor representations using differentiable boundary trees. *arXiv preprint arXiv:1702.08833*, 2017.

[149] Charles Mathy, Nate Derbinsky, José Bento, Jonathan Rosenthal, and Jonathan S Yedidia. The boundary forest algorithm for online supervised and unsupervised learning. In *AAAI*, pages 2864–2870, 2015.

[150] Yichuan Tang. Deep learning using linear support vector machines. *arXiv preprint arXiv:1306.0239*, 2013.

[151] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research).

[152] Xiao-Xiao Niu and Ching Y Suen. A novel hybrid cnn–svm classifier for recognizing handwritten digits. *Pattern Recognition*, 45(4):1318–1325, 2012.

[153] Masoumeh Zareapoor, Pourya Shamsolmoali, Deepak Kumar Jain, Haoxiang Wang, and Jie Yang. Kernelized support vector machine with deep learning: an efficient approach for extreme multiclass dataset. *Pattern Recognition Letters*, 2017.

[154] Jawad Nagi, Gianni A Di Caro, Alessandro Giusti, Farrukh Nagi, Luca Maria Gambardella, et al. Convolutional neural support vector machines: Hybrid visual pattern classifiers for multi-robot systems. In *ICMLA (1)*, pages 27–32, 2012.

[155] Abdel Bellili, Michel Gilloux, and Patrick Gallinari. An hybrid mlp-svm handwritten digit recognizer. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 28–32. IEEE, 2001.

[156] Washington W Azevedo and Cleber Zanchet. A mlp-svm hybrid model for cursive handwriting recognition. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 843–850. IEEE, 2011.

[157] Tianyi Zhao, Baopeng Zhang, Ming He, Wei Zhanga, Ning Zhou, Jun Yu, and Jianping Fan. Embedding visual hierarchy with deep networks for large-scale visual recognition. *IEEE Transactions on Image Processing*, 2018.

[158] Harold Hotelling. Relations between two sets of variates. In *Breakthroughs in statistics*, pages 162–190. Springer, 1992.

[159] David R Hardoon, Sandor Szedmak, and John Shawe-Taylor. Canonical correlation analysis: An overview with application to learning methods. *Neural computation*, 16(12):2639–2664, 2004.

[160] Galen Andrew, Raman Arora, Jeff Bilmes, and Karen Livescu. Deep canonical correlation analysis. In *International conference on machine learning*, pages 1247–1255, 2013.

[161] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[162] David Alvarez-Melis and Tommi S Jaakkola. Tree-structured decoding with doubly-recurrent neural networks. 2016.

[163] Andrea Cimino and Felice Dell'Orletta. Tandem lstm-svm approach for sentiment analysis. In *of the Final Workshop 7 December 2016, Naples*, page 172, 2016.

[164] Abien Fred M Agarap. A neural network architecture combining gated recurrent unit (gru) and support vector machine (svm) for intrusion detection in network traffic data. In *Proceedings of the 2018 10th International Conference on Machine Learning and Computing*, pages 26–30. ACM, 2018.

[165] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In *ESANN*, 2013.

[166] Large scale visual recognition challenge 2017 (ilsvrc2017), 2017.

[167] Nikolaos G Paterakis, Elena Mocanu, Madeleine Gibescu, Bart Stappers, and Walter van Alst. Deep learning versus traditional machine learning methods for aggregated energy demand prediction. In *Innovative Smart Grid Technologies Conference Europe (ISGT-Europe), 2017 IEEE PES*, pages 1–6. IEEE, 2017.

[168] Alexander Sboev, Ivan Moloshnikov, Dmitry Gudovskikh, Anton Selivanov, Roman Rybka, and Tatiana Litvinova. Deep learning neural nets versus traditional machine learning in gender identification of authors of rusprofiling texts. *Procedia Computer Science*, 123:424–431, 2018.

[169] Chen-Ying Hung, Wei-Chen Chen, Po-Tsun Lai, Ching-Heng Lin, and Chi-Chun Lee. Comparing deep neural network and other machine learning algorithms for stroke prediction in a large-scale population-based electronic medical claims database. In *Engineering in Medicine and Biology Society (EMBC), 2017 39th Annual International Conference of the IEEE*, pages 3110–3113. IEEE, 2017.

[170] Chuanlong Yin, Yuefei Zhu, Jinlong Fei, and Xinzheng He. A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, 5:21954–21961, 2017.

[171] Haytham M Fayek, Margaret Lech, and Lawrence Cavedon. Evaluating deep learning architectures for speech emotion recognition. *Neural Networks*, 92:60–68, 2017.

[172] Alexandre De Brébisson, Étienne Simon, Alex Auvolat, Pascal Vincent, and Yoshua Bengio. Artificial neural networks applied to taxi destination prediction. *arXiv preprint arXiv:1508.00021*, 2015.

[173] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[174] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. *arXiv preprint arXiv:1711.03213*, 2017.