IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.Doi Number

Review of Deep Learning Algorithms and Architectures

Ajay Shrestha, Ausif Mahmood (Senior Member, IEEE)
Department of Computer Science and Engineering, University of Bridgeport, 126 Park Ave, Bridgeport, CT 06604, USA

Corresponding author: Ajay Shrestha (e-mail: shrestha@my.bridgeport.edu).

ABSTRACT Deep learning (DL) is playing an increasingly important role in our lives. It has already made a huge impact in areas such as cancer diagnosis, precision medicine, self-driving cars, predictive forecasting, speech recognition, etc. The painstakingly handcrafted feature extractors used in the traditional learning, classification and pattern recognition systems are not scalable for large-sized data sets. In many cases depending on the problem complexity, deep learning can also overcome limitations of earlier shallow networks that prevented efficient training and abstractions of hierarchical representations of multi-dimensional training data. Deep Neural Network (DNN) uses multiple (deep) layers of units with highly optimized algorithms and architectures. The paper reviews several optimization methods to improve accuracy of the training and reduce training time. We delve into the math behind training algorithms used in recent deep networks. We describe current shortcomings, enhancements and implementations. The review also covers different types of deep architectures such as deep convolution networks, deep residual networks, recurrent neural networks, reinforcement learning, variational autoencoders, and others.

INDEX TERMS Machine Learning Algorithm, Optimization, Artificial Intelligence, Deep Neural Network Architectures, Convolution Neural Network, Backpropagation, Supervised and unsupervised learning

1. Introduction

Neural Network is a machine learning (ML) technique that is inspired by and resembles the human nervous system and the structure of the brain. It consists of processing units organized in input, hidden and output layers. The nodes or units in each layer are connected to nodes in adjacent layers. Each connection has a weight value. The inputs are multiplied by the respective weights and summed at each unit. The sum then undergoes a transformation based on the activation function, which is in most cases is a sigmoid function, tan hyperbolic or rectified linear unit (ReLU). These functions are used because they have a mathematically favorable derivative, making it easier to compute partial derivatives of the error delta with respect to individual weights. Sigmoid and tanh functions also squash the input into a narrow output range or option, i.e., 0/1 and -1/+1 respectively. They implement saturated nonlinearity as the outputs plateaus or saturates before/after respective thresholds. ReLu on the other hand exhibits both saturating and non-saturating behaviors with $f(x) = max(0, x)$. The output of the function is then fed as input to the subsequent unit in the next layer. The result of the final output layer is used as the solution for the problem.

Neural Networks can be used in a variety of problems including pattern recognition, classification, clustering, dimensionality reduction, computer vision, natural language processing (NLP), regression, predictive analysis, etc. Here is an example of image recognition.

Figure 1 shows how a deep neural network called Convolution Neural Network (CNN) can learn hierarchical levels of representations from a low-level input vector and successfully identify the higher-level object. The red squares in the figure are simply a gross generalization of the pixel values of the highlighted section of the figure. CNNs can progressively extract higher representations of the image after each layer and finally recognize the image.

The implementation of neural networks consists of the following steps:

1. Acquire training and testing data set
2. Train the network
3. Make prediction with test data

The paper is organized in the following sections:

1. Introduction to Machine Learning
    a. Background and Motivation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

2. Classifications of Neural Networks
3. DNN Architectures
4. Training Algorithms
5. Shortcomings of Training Algorithms
6. Optimization of Training Algorithms
7. Architectures & Algorithms – Implementations
8. Conclusion

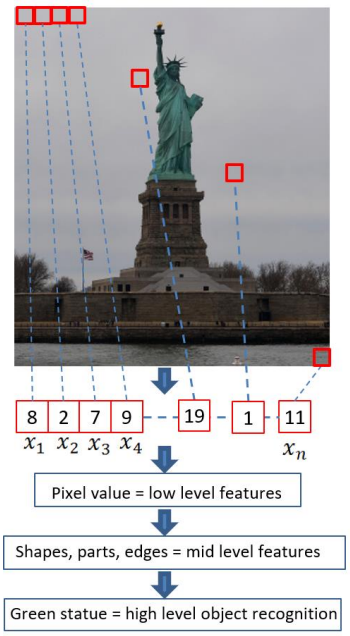

Figure 1. Image recognition by a CNN

1.1. Background

In 1957, Frank Rosenblatt created the perceptron, the first prototype of what we now know as a neural network [1]. It had two layers of processing units that could recognize simple patterns. Instead of undergoing more research and development, neural networks entered a dark phase of its history in 1969, when professors at MIT demonstrated that it couldn't even learn a simple XOR function[2].

In addition, there was another finding that particularly dampened the motivation for DNN. The universal approximation theorem showed that a single hidden layer was able to solve any continuous problem [3]. It was mathematically proven as well [4], which further questioned the validity of DNN. While a single hidden layer could be used to learn, it was not efficient and was a far cry from the convenience and capability afforded by the hierarchical abstraction of multiple hidden layers of DNN that we know now. But it was not just the universal approximation theorem that held back the progress of DNN. Back then, we didn't have a way to train a DNN either. These factors prolonged the so-called AI winter, i.e., a phase in the history of artificial intelligence where it didn't get much funding and interest, and as a result didn't advance much either.

A breakthrough in DNN occurred with the advent of backpropagation learning algorithm. It was proposed in the 1970s [5] but it wasn't until mid-1980s [6] that it was fully understood and applied to neural networks. The self-directed learning was made possible with the deeper understanding and application of backpropagation algorithm. The automation of feature extractors is what differentiates a DNNs from earlier generation machine learning techniques.

DNN is a type of neural network modeled as a multilayer perceptron (MLP) that is trained with algorithms to learn representations from data sets without any manual design of feature extractors. As the name Deep Learning suggests, it consists of higher or deeper number of processing layers, which contrasts with shallow learning model with fewer layers of units. The shift from shallow to deep learning has allowed for more complex and non-linear functions to be mapped, as they cannot be efficiently mapped with shallow architectures. This improvement has been complemented by the proliferation of cheaper processing units such as the general-purpose graphic processing unit (GPGPU) and large volume of data set (big data) to train from. While GPGPUs are less powerful that CPUs, the number of parallel processing cores in them outnumber CPU cores by orders of magnitude. This makes GPGPUs better for implementing DNNs. In addition to the backpropagation algorithm and GPU, the adoption and advancement of ML and particularly Deep Learning can be attributed to the explosion of data or bigdata in the last 10 years. ML will continue to impact and disrupt all areas of our lives from education, finance, governance, healthcare, manufacturing, marketing and others [7].

1.2. Motivation

Deep learning is perhaps the most significant development in the field of computer science in recent times. Its impact has been felt in nearly all scientific fields. It is already disrupting and transforming businesses and industries. There is a race among the world's leading economies and technology companies to advance deep learning. There are already many areas where deep learning has exceeded human level capability and performance, e.g., predicting movie ratings, decision to approve loan applications, time taken by car delivery, etc. [8]. On March 27, 2019 the three deep learning pioneers (Yoshua Bengio, Geoffrey Hinton, and Yann LeCun) were awarded the Turing Award, which is also referred to as the "Nobel Prize" of computing[9]. While a lot has been accomplished, there is more to advance in deep learning. Deep learning has a potential to improve human lives with more accurate diagnosis of diseases like cancer [10], discovery of new drugs, prediction of natural disasters [11]. E.g., [12] reported that an deep learning network was able to learn from 129,450 images of 2,032 diseases and was able to diagnose at the same level as 21 board certified dermatologists. Google AI [10] was able to beat the average accuracy of US board certified general pathologists in grading prostate cancer by 70% to 61%.

The goal of this review is to cover the vast subject of deep learning and present a holistic survey of dispersed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

information under one article. It presents novel work by collating the works of leading authors from the wide scope and breadth the deep learning. Other review papers [13-16] focus on specific areas and implementations without encompass the full scope of the field. This review covers the different types of deep learning network architectures, deep learning algorithms, their shortcomings, optimization methods and the latest implementations and applications.

2. Classification of Neural Network

Neural Networks can be classified into the following different types.

1. Feedforward Neural Network
2. Recurrent Neural Network (RNN)
3. Radial Basis Function Neural Network
4. Kohonen Self Organizing Neural Network
5. Modular Neural Network

In feedforward neural network, information flows in just one direction from input to output layer (via hidden nodes if any). They do not form any circles or loopbacks. Figure 2a shows a particular type of implementation of a multilayer feedforward neural network with values and functions computed along the forward pass path. Z is the weighed sum of the inputs and y represents the non-linear activation function f of Z at each layer. W represents the weights between the two units in the adjoining layers indicated by the subscript letters and b represents the bias value of the unit.

Unlike feedforward neural networks, the processing units in RNN form a cycle. The output of a layer becomes the input to the next layer, which is typically the only layer in the network, thus the output of the layer becomes an input to itself forming a feedback loop. This allows the network to have memory about the previous states and use that to influence the current output. One significant outcome of this difference is that unlike feedforward neural network, RNN can take a sequence of inputs and generate a sequence of output values as well, rendering it very useful for applications that require processing sequence of time phased input data like speech recognition, frame-by-frame video classification, etc.

Figure 2b demonstrates the unrolling of a RNN in time. E.g., if a sequence of 3-word sentence constitutes an input, then each word would correspond to a layer and thus the network would be unfolded or unrolled 3 times into a 3-layer RNN.

Here is the mathematical explanation of the diagram: x_t represents the input at time t. U, V, and W are the learned parameters that are shared by all steps. O_t is the output at time t. S_t represents the state at time t and can be computed as follows, where f is the activation function, e.g., ReLU.

$$S_t = f(Ux_t + Ws_{t-1}) \quad (1)$$

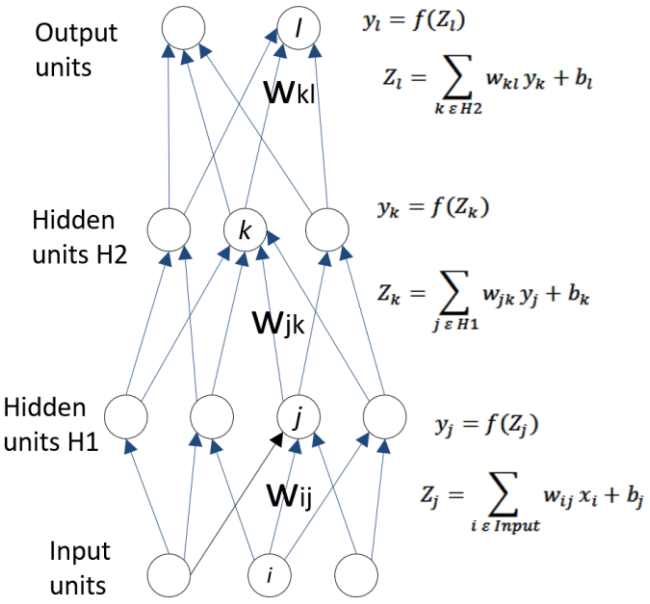

Figure 2a. Feedforward neural network [6]

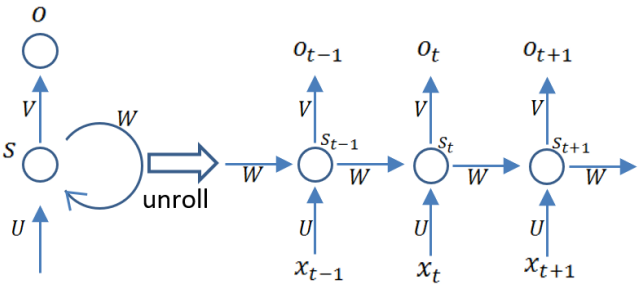

Figure 2b. The unrolling of RNN in time [6]

Radial basis function neural network is used in classification, function approximation, time series prediction problems, etc. It consists of input, hidden and output layers. The hidden layer includes a radial basis function (implemented as gaussian function) and each node represents a cluster center. The network learns to designate the input to a center and the output layer combines the outputs of the radial basis function and weight parameters to perform classification or inference[17].

Kohonen self-organizing neural network self organizes the network model into the input data using unsupervised learning. It consists of two fully connected layers, i.e., input layer and output layer. The output layer is organized as a two-dimensional grid. There is no activation function and the weights represent the attributes (position) of the output layer node. The Euclidian distance between the input data and each output layer node with respect to the weights are calculated. The weights of the closest node and its neighbors from the input data are updated to bring them closer to the input data with the formula below[18].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

$$w_i(t+1) = w_i(t) + \alpha(t)\eta_{j*i}(x(t) - w_i(t)) \quad (2)$$

Where $x(t)$ is the input data at time t, $w_i(t)$ is the ith weight at time t and η_{j*i} is the neighborhood function between the $ith\ and\ jth$ nodes.

Modular neural network breaks down large network into smaller independent neural network modules. The smaller networks perform specific task which are later combined as part of a single output of the entire network [19].

DNNs are implemented in the following popular ways:

1. Sparse Autoencoders
2. Convolution Neural Networks (CNNs or ConvNets)
3. Restricted Boltzmann Machines (RBMs)
4. Long Short-Term Memory (LSTM)

Autoencoders are neural networks that learn features or encoding from a given dataset in order to perform dimensionality reduction. Sparse Autoencoder is a variation of Autoencoders, where some of the units output a value close to zero or are inactive and do not fire. Deep CNN uses multiple layers of unit collections that interact with the input (pixel values in the case of image) and result in desired feature extraction. CNN finds it application in image recognition, recommender systems and NLP. RBM is used to learn probability distribution within the data set.

All these networks use backpropagation for training. Backpropagation uses gradient descent for error reduction, by adjusting the weights based on the partial derivative of the error with respect to each weight.

Neural Network models can also be divided into the following two distinct categories:

1. Discriminative
2. Generative

Discriminative model is a bottom-up approach in which data flows from input layer via the hidden layers to the output layer. They are used in supervised training for problems like classification and regression. Generative models on the other hand are top-down and data flows in the opposite direction. They are used in unsupervised pre-training and probabilistic distribution problems. If the input x and corresponding label y are given, a discriminative model learns the probability distribution p(y|x), i.e., the probability of y given x directly, whereas a generative model learns the joint probability of p(x,y), from which P(y|x) can be predicted [20]. In general whenever labelled data is available discriminative approaches are undertaken as they provide effective training, and when labelled data is not available generative approach can be taken [21].

Training can be broadly categorized into three types:

1. Supervised
2. Unsupervised
3. Semi-supervised

Supervised learning consists of labeled data which is used to train the network, whereas unsupervised learning there is no labeled data set, thus no learning based on feedback. In unsupervised learning, neural networks are pre-trained using generating models such as RBMs and later could be fine-tuned using standard supervised learning algorithms. It is then used on test data set to determine patterns or classifications. Big data has pushed the envelope even further for deep learning with its sheer volume and variety of data. Contrary to our intuitive inclination, there is no clear consensus on whether supervised learning is better than the unsupervised learning. Both have their merits and use cases. [22] demonstrated enhance results with unsupervised learning using unstructured video sequences for camera motion estimation and monocular depth. Modified Neural Networks such as Deep Belief Network (DBM) as described by Xue-Wen Chen et al. [23] uses both labeled and unlabeled data with supervised and unsupervised learning respectively to improve performance. Developing a way to automatically extract meaningful features from labeled and unlabeled high dimensional data space is challenging. Yann LeCun et al. asserts that one way we could achieve this would be to utilize and integrate both unsupervised and supervised learning [24]. Complementing unsupervised learning (with un-labeled data) with supervised learning (with labeled data) is referred to as semi-supervised learning.

DNN and training algorithms have to overcome two major challenges: premature convergence and overfitting. Premature convergence occurs when the weights and bias of the DNN settle into a state that is only optimal at a local level and misses out on the global minima of the entire multi-dimensional space. Overfitting on the other hand describes a state when DNNs become highly tailored to a given training data set at a fine grain level that it becomes unfit, rigid and less adaptable for any other test data set.

Along with different types of training, algorithms and architecture, we also have different machine learning frameworks (Table 1) and libraries that have made training models easier. These frameworks make complex mathematical functions, training algorithms and statistically modeling available without having to write them on your own. Some provide distributed and parallel processing capabilities, and convenient development and deployment features. Figure 3 shows a graph with various deep learning libraries along with their Github stars from 2015-2018. Github is the largest hosting service provider of source code in the world [25]. Github stars are indicative of how popular a project is on Github. TensorFlow is the most popular DL library.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

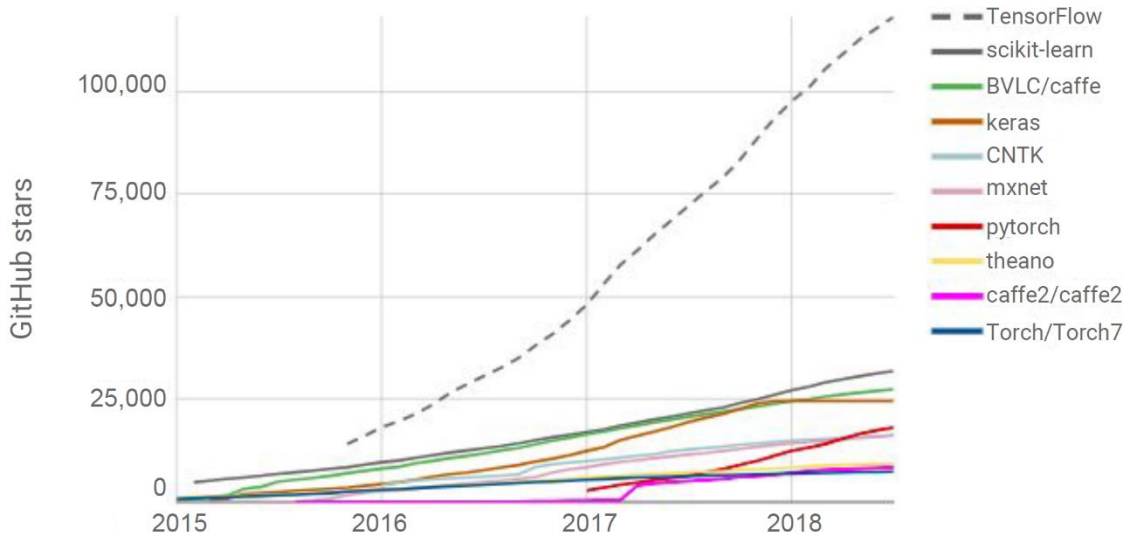

Figure 3. Github stars by Deep Learning Library [26]

TABLE 1. POPULAR DEEP LEARNING FRAMEWORKS AND LIBRARIES

Framework	Institution	License	1st Release
Caffe	Berkeley AI Research	BSD / Free	2015
Microsoft Cognitive Toolkit	Microsoft	MIT License / Free	2016
Gluon	AWS and Microsoft	Open Source	2017
Keras	Individual Author	MIT License / Free	2015
MXNet	Apache Software Foundation	Apache 2.0 / Free	2015
TensorFlow	Google Brain	Apache 2.0 / Free	2015
Theano	University of Montreal	BSD / Free	2008
Torch	Ronan Collobert et al.	BSD / Free	2002
PyTorch	Facebook	BSD / Free	2016
Chainer	Preferred Networks	BSD / Free	2015
Deeplearning4j	Adam Gibson et al.	Apache 2.0 / Free	2014

3. DNN Architectures

Deep neural network consists of several layers of nodes. Different architectures have been developed to solve problems in different domains or use-cases. E.g., CNN is used most of the time in computer vision and image recognition, and RNN is commonly used in time series problems/forecasting. On the other hand, there is no clear winner for general problems like classification as the choice of architecture could depend on multiple factors. Nonetheless [27] evaluated 179 classifiers and concluded that parallel random forest or parRF_t, which is essentially parallel implementation of variation of decision tree, performed the best. Below are three of the most common architectures of deep neural networks.

1. Convolution Neural Network
2. Autoencoder
3. Restricted Boltzmann Machine (RBM)
4. Long Short-Term Memory (LSTM)

3.1. Convolution Neural Network

CNN is based on the human visual cortex and is the neural network of choice for computer vision (image recognition) and video recognition. It is also used in other areas such as NLP, drug discovery, etc. As shown in Figure 4, a CNN consists of a series of convolution and sub-sampling layers followed by a fully connected layer and a normalizing (e.g., softmax function) layer. Figure 4 illustrates the well-known 7 layered LeNet-5 CNN architecture devised by LeCun et al. **[28]** for digit recognition. The series of multiple convolution layers perform progressively more refined feature extraction at every layer moving from input to output layers. Fully connected layers that perform classification follow the convolution layers. Sub-sampling or pooling layers are often inserted between each convolution layers. CNN's takes a 2D $\boldsymbol{n\,x\,n}$ pixelated image as an input. Each layer consists of groups of 2D neurons called filters or kernels. Unlike other neural networks, neurons in each feature extraction layers of CNN are not connected to all neurons in the adjacent layers. Instead, they are only connected to the spatially mapped fixed sized and partially overlapping neurons in the previous layer's input image or feature map. This region in the input is called *local receptive field*. The lowered number of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

connections reduces training time and chances of overfitting. All neurons in a filter are connected to the same number of neurons in the previous input layer (or feature map) and are constrained to have the same sequence of weights and biases. These factors speed up the learning and reduces the memory requirements for the network. Thus, each neuron in a specific filter looks for the same pattern but in different parts of the input image. Sub-sampling layers reduce the size of the network. In addition, along with local receptive fields and shared weights (within the same filter), it effectively reduces the network's susceptibility of shifts, scale and distortions of images **[29]**. Max/mean pooling or local averaging filters are used often to achieve sub-sampling. The final layers of CNN are responsible for the actual classifications, where neurons between the layers are fully connected. Deep CNN can be implemented with multiple series of weight-sharing convolution layers and sub-sampling layers. The deep nature of the CNN results in high quality representations while maintaining locality, reduced parameters and invariance to minor variations in the input image **[30]**.

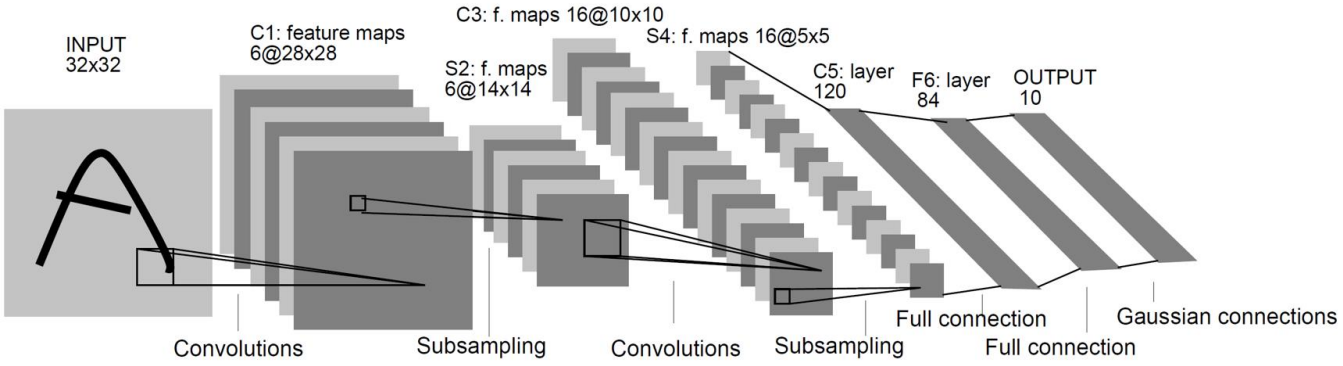

Figure 4. 7-layer Architecture of CNN for character recognition [28]

In most cases, backpropagation is used solely for training all parameters (weights and biases) in CNN. Here is a brief description of the algorithm. The cost function with respect to individual training example (x, y) in hidden layers can be defined as [31]:

$$J(W, b; x, y) = \frac{1}{2}||h_{w,b}(x) - y||^2 \qquad (3)$$

The equation for error term δ for layer l is given by [31]:

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) . f'(z^{(l)}) \qquad (4)$$

Where $\delta^{(l+1)}$ is the error for $(l + 1)$th layer of a network whose cost function is $J(W, b; x, y)$. $f'(z^{(l)})$ represents the derivate of the activation function.

$$\nabla_{w^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l+1)})^T \qquad (5)$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)} \qquad (6)$$

Where a is the input, such that $a^{(1)}$ is the input for 1st layer (i.e., the actual input image) and $a^{(l)}$ is the input for $l - th$ layer.

Error for sub-sampling layer is calculated as [31]:

$$\delta_k^{(l)} = upsample\left((W_k^{(l)})^T \delta_k^{(l+1)}\right) . f'(z_k^{(l)}) \qquad (7)$$

Where k represent the filter number in the layer. In the sub-sampling layer, the error has to be cascaded in the opposite direction, e.g., where mean pooling is used, *upsample* evenly distributes the error to the previous input unit. And finally, here is the gradient w.r.t. feature maps [31]:

$$\nabla_{w_k^{(l)}} J(W, b; x, y) = \sum_{i-1}^{m} (a_i{}^{(l)}) * rot90\left(\delta_k^{(l+1)}, 2\right) \qquad (8)$$

$$\nabla_{b_k^{(l)}} J(W, b; x, y) = \sum_{a,b} \left(\delta_k^{(l+1)}\right)_{a,b.} \qquad (9)$$

Where $(a_i{}^{(l)}) * \delta_k^{(l+1)}$ represents the convolution between error and the $i - th$ input in the $l - th$ layer with respect to the $k - th$ filter.

Algorithm 1 below represents a high-level description and flow of the backpropagation algorithm as used in a CNN as it goes through multiple epochs until either the maximum iterations are reached or the cost function target is met.

In addition to discriminative models such as image recognition, CNN can also be used for generative models such as deconvolving images to make blurry image sharper. [32] achieves this by leveraging Fourier transformation to regularize inversion of the blurred images and denoising. Different implementations of CNN has shown continuous improvement of accuracy in computer vision. The improvements are tested against the same benchmark (ImageNet) to ensure unbiased results.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

ALGORITHM 1. CNN BACKPROPAGATION ALGORITHM PSEUDO CODE

1: *Initialization weights to randomly generated value (smal*
2: *Set learning rate to a small value (positive)*
3: *Iteration* $n = 1$; ***Begin***
4: ***for*** n
$<$ *max iteration OR Cost function criteria met,* ***do***
5: ***for*** *image* x_1 *to* x_i, ***do***
6: *a. Forward propagate through convolution, pooling and*
7: *b. Derive Cost Fuction value for the image*
8: *c. Calculate error term* $\delta^{(l)}$ *with respect to weights for e*
9: *Note that the error gets propagated from layer to lay*
10: *i. fully connected layer*
11: *ii. pooling layer*
12: *iii. convolution layer*
13: *d. Calculate gradient* $\nabla_{w_k^{(l)}}$ *and* $\nabla_{b_k^{(l)}}$ *for weights* $\nabla_{w_k^{(l)}}$*and*
14: *Gradient calculated in the following sequence*
15: *i. convolution layer*
16: *ii. pooling layer*
17: *iii. fully connected layer*
18: *e. Update weights*
19: $w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} + \Delta w_{ji}^{(l)}$
20: *f. Update bias*
21: $b_j^{(l)} \leftarrow b_j^{(l)} + \Delta b_j^{(l)}$

Here are the well-known variation and implementation of the CNN architecture.

1. AlexNet:
   a. CNN developed to run on Nvidia parallel computing platform to support GPUs
2. Inception:
   a. Deep CNN developed by Google
3. ResNet:
   a. Very deep Residual network developed by Microsoft. It won 1st place in the ILSVRC 2015 competition on ImageNet dataset.
4. VGG:
   a. Very deep CNN developed for large scale image recognition
5. DCGAN:
   a. Deep convolutional generative adversarial networks proposed by [33]. It is used in unsupervised learning of hierarchy of feature representations in input objects.

3.2. Autoencoder

Autoencoder is a neural network that uses unsupervised algorithm and learns the representation in the input data set for dimensionality reduction and to recreate the original data set. The learning algorithm is based on the implementation of the backpropagation.

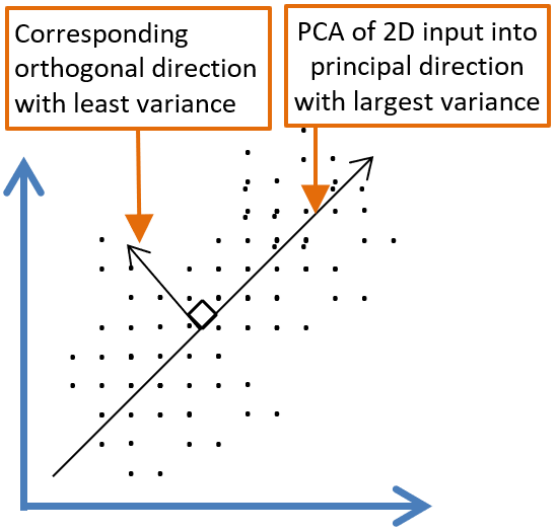

Figure 5. Linear representation of a 2D data input using PCA

Autoencoders extend the idea of principal component analysis (PCA). As shown in Figure 5, a PCA transforms multi-dimensional data into a linear representation. Figure 5 demonstrates how a 2D input data can be reduced to a linear vector using PCA. Autoencoders on the other hand can go further and produce nonlinear representation. PCA determines a set of linear variables in the directions with largest variance. The p dimensional input data points are represented as m orthogonal directions, such that $m \leq p$ and constitutes a lower (i.e., less than m) dimensional space. The original data points are projected into the principal directions thus omitting information in the corresponding orthogonal directions. PCA focuses more on the variances rather than covariances and correlations and it looks for the linear function with the most variance [34]. The goal is to determine the direction with the least mean square error, which would then have the least reconstruction error.

Autoencoders use encoder and decoder blocks of non-linear hidden layers to generalize PCA to perform dimensionality reduction and eventual reconstruction of the original data. It uses greedy layer by layer unsupervised pre-training and fin-tuning with backpropagation [35]. Despite using backpropagation, which is mostly used in supervised training, autoencoders are considered unsupervised DNN because they regenerate the input $x^{(i)}$ itself instead of a different set of target values $y^{(i)}$, i.e., $y^{(i)} = x^{(i)}$. Hinton et al. were able to achieve a near perfect reconstruction of 784-pixel images using autoencoder, proving that it is far better than PCA [36].

While performing dimensionality reduction, autoencoders come up with interesting representations of the input vector in the hidden layer. This is often attributed to the smaller number of nodes in the hidden layer or every second layer of the two-layer blocks. But even if there are higher number of nodes in the hidden layer, a sparsity constraint can be enforced on the hidden units to retain interesting lower

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

dimension representations of the inputs. To achieve sparsity, some nodes are restricted from firing, i.e., the output is set to a value close to zero.

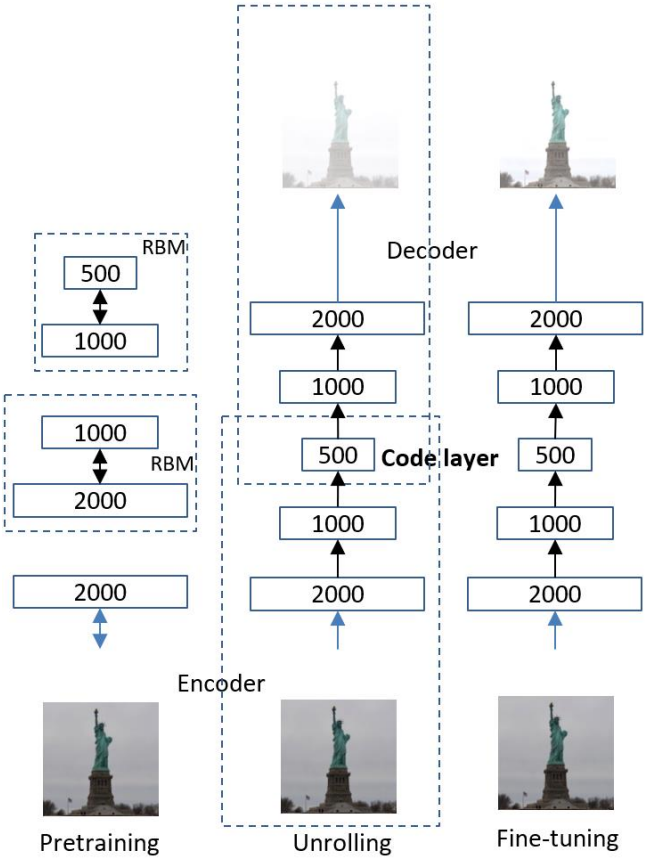

Figure 6. Training stages in Autoencoder [36]

Figure 6 shows single layer feature detector blocks of RBMs used in pre-training, which is followed by unrolling **[36]**. Unrolling combines the stacks of RBMs to create the encoder block and then reverses the encoder block to create the decoder section**, and** finally the network is fine-tuned with backpropagation **[36]**.

Figure 7 illustrates a simplified representation of how autoencoders can reduce the dimension of the input data and learn to recreate it in the output layer. Wang et al. [37] successfully implemented a deep autoencoder with stacks of RBM blocks similar to Figure 6 to achieve better modeling accuracy and efficiency than the proper orthogonal decomposition (POD) method for dimensionality reduction of distributed parameter systems (DPSs). The equation below describes the average of activation function $a_j^{(2)}$ of jth unit of 2nd layer when the xth input activates the neuron [38].

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}[a_j^{(2)} x^{(i)}] \qquad (10)$$

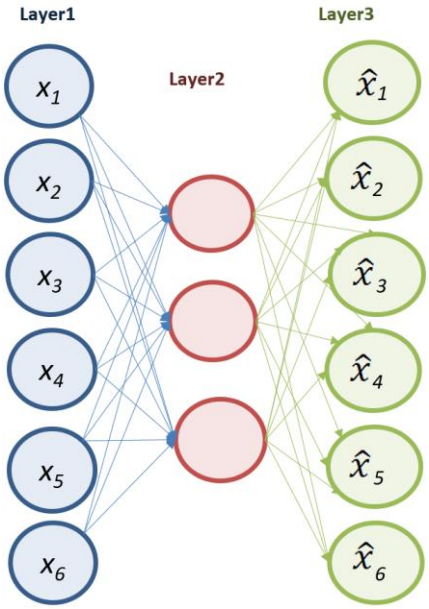

Figure 7. Autoencoder nodes

A sparsity parameter $\rho$is introduced such that ρ is very close to zero, e.g., 0.03 and $\hat{\rho} = \rho$. To ensure that $\hat{\rho} = \rho$, a penalty term $KL(\rho||\hat{\rho}_j)$ is introduced such that the Kullback–Leibler (KL) divergence term $KL(\rho||\hat{\rho}_j) = 0$, if $\hat{\rho}_j = \rho$, else becomes large monotonically as the difference between the two values diverges [38]. Here is the updated cost function [38]:

$$J_{sparse}(W,b) = J(W,b) + \beta\sum_{j=1}^{s2} KL(\rho||\hat{\rho}_j)] \qquad (11)$$

Where s2 equals the number of units in 2nd layer and β is the parameter than controls sparsity penalty term's weight.

3.3. Restricted Boltzmann Machine (RBM)

Restricted Boltzmann Machine is an artificial neural network where we can apply unsupervised learning algorithm to build non-linear generative models from unlabeled data [39]. The goal is to train the network to increase a function (e.g., product or log) of the probability of vector in the visible units so it can probabilistically reconstruct the input. It learns the probability distribution over its inputs. As shown in Figure 8, RBM is made of two-layer network called the visible layer and the hidden layer. Each unit in the visible layer is connected to all units in the hidden layer and there are no connections between the units in the same layer.

The energy (E) function of the configuration of the visible and hidden units, (v, h) is expressed in the following way [40]:

$$\mathrm{E(v,h)} = -\sum_{i\,\varepsilon\,visible} a_i v_i - \sum_{j\,\varepsilon\,hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \qquad (12)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

v_i and h_j are the vector states of the visible unit *i* and hidden unit *j*. a_i and b_j represents the bias of visible and hidden units. W_{ij} denotes the weight between the respective visible and hidden units.

The partition function, Z is represented by the sum of all possible pairs of visible and hidden vectors [40].

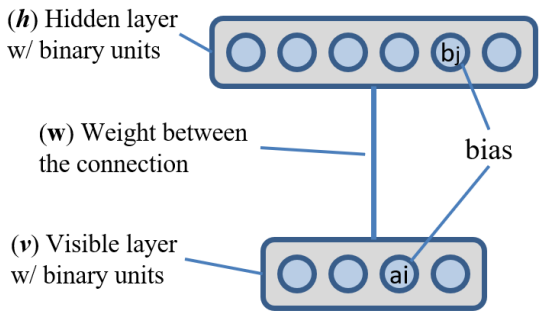

Figure 8. Restricted Boltzmann Machine

$$Z = \sum_{v,h} e^{-E\,(v,h)} \tag{13}$$

The probability of every pair of visible and hidden vectors is given by the following [40].

$$p(v,h) = \frac{1}{Z} e^{-E\,(v,h)} \tag{14}$$

The probability of a particular visible layer vector is provided by the following [40].

$$p(v) = \frac{1}{Z} \sum_{h} e^{-E\,(v,h)} \tag{15}$$

As you can see from the equations above, the partition function becomes higher with lower energy function value. Thus during the training process, the weights and biases of the network are adjusted to arrive at a lower energy and thus maximize the probability assigned to the training vector. It is mathematically convenient to compute the derivative of the log probability of a training vector.

$$\frac{\partial \log p(v)}{\partial w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \tag{16}$$

In the equation [40] above $\langle v_i h_j \rangle_{data}$ and $\langle v_i h_j \rangle_{model}$ represents the expectations under the respective distributions.

Thus, the adjustments in the weights can be denoted as follows [40], where ϵ is the learning rate.

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}) \tag{28}$$

3.4. Long Short-Term Memory (LSTM)

LSTM is an implementation of the Recurrent Neural Network and was first proposed by Hochreiter et al. in 1997 [41]. Unlike the earlier described feed forward network architectures, LSTM can retain knowledge of earlier states and can be trained for work that requires memory or state awareness. LSTM partly addresses a major limitation of RNN, i.e., the problem of vanishing gradients by letting gradients to pass unaltered. As shown in the illustration in Figure 9, LSTM consists of blocks of memory cell state through which signal flows while being regulated by input, forget and output gates. These gates control what is stored, read and written on the cell. LSTM is used by Google, Apple and Amazon in their voice recognition platforms [42].

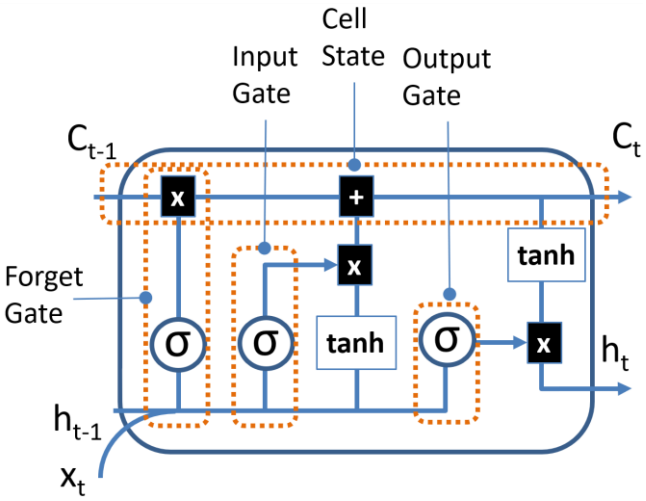

Figure 9. LSTM Block with memory cell and gates

In figure 9, C, x, h represent cell, input and output values. Subscript t denotes time step value, i.e., $t-1$ is from previous LSTM block (or from time $t-1$) and t denotes current block values. The symbol σ is the sigmoid function and $tanh$ is the hyperbolic tangent function. Operator + is the element-wise summation and x is the element-wise multiplication. The computations of the gates are described in the equations below[41, 43].

$$f_t = \sigma(W_f x_t + w_f h_{t-1} + b_f) \tag{17}$$

$$i_t = \sigma(W_i x_t + w_i h_{t-1} + b_i) \tag{18}$$

$$o_t = \sigma(W_o x_t + w_o h_{t-1} + b_o) \tag{19}$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \sigma_c(W_c x_t + w_c h_{t-1} + b_c) \tag{20}$$

$$h_t = o_t \otimes \sigma_h(c_t) \tag{21}$$

Where f, i, o are the forget, input and output gate vectors respectively. W, w, b *and* $\otimes$ represent weights of input, weights of recurrent output, bias and element-wise multiplication respectively.

There is a smaller variation of the LSTM known as gated recurrent units (GRU). GRUs are smaller in size than LSTM as they don't include the output gate, and can perform better than LSTM on only some simpler datasets[44, 45].

LSTMs recurrent neural networks can keep track of long-term dependencies. Therefore, they are great for learning

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

from sequence input data and building models that rely on context and earlier states. The cell block of LSTM retains pertinent information of previous states. The input, forget and output gates dictates new data going into the cell, what remains in the cell and the cell values used in the calculation of the output of the LSTM block respectively [41, 43]. Naul et al. demonstrated LSTM and GRU based autoencoders for automatic feature extractions [46].

3.5. Comparison of DNN Networks

Table 2 provides a compact summary and comparison of the different DNN architectures. The examples of implementations, applications, datasets and DL software frameworks presented in the table are not implied to be exhaustive. In addition, some of the categorization of the network architectures could be implemented in hybrid fashion. E.g., even though RBMs are generative models and their training is considered unsupervised, they can have elements of discriminative model when training is finetuned with supervised learning. The table also provides examples of common applications for using different architectures.

TABLE 2. DNN NETWORK COMPARISON TABLE

Network Type	Architecture	Network Model	Training Type	Training Algorithm	Implementation Sample	Common Application	Popular Dataset Sample	DL Framework (sample)
Feedforward Neural Network	CNN	Discriminative	Supervised	Gradient Descent based Backpropagation	Siamese Network, Deep CNN	Image recognition/classification	MNIST	TensorFlow, Caffe, Theano, Torch, Deeplearning4j, Microsoft Cognitive Toolkit, Keras, MXNet, PyTorch
	Residual Network	Discriminative	Supervised	Gradient Descent based Backpropagation	Deep ResNet; HighwayNet; DenseNet	Image recognition	ImageNet	TensorFlow, PyTorch, Keras
	Autoencoder	Generative	Unsupervised	Backpropagation	Sparse Autoencoders, Variational Autoencoders	Dimensionality Reduction; Encoding	MNIST	TensorFlow, Deeplearning4j, Keras
	Adversarial Networks	Generative & Discriminative	Unsupervised	Backpropagation	Generative Adversarial Network	Generate realistic fake data; Reconstruction of 3D models; Image improvement	CIFAR10	TensorFlow, Keras
	RBM	Generative with Discriminative finetuning	Unsupervised	Gradient Descent based Contrastive divergence	Deep Belief Network; Deep Boltzmann Machine	Dimensionality Reduction; Feature learning; Topic modeling	MNIST	TensorFlow, Deeplearning4j, Keras, MXNet, Theano, Torch
Recurrent Neural Network	LSTM	Discriminative	Supervised	Gradient Descent & Backpropagation through Time	Deep RNN, Gated Recurrent Unit (GRU), Neural Machine Translation (NMT)	Natural Language Processing; Language Translation	MNIST Stroke Sequence	TensorFlow, Caffe, Theano, Torch, Deeplearning4j, Microsoft Cognitive Toolkit, Keras, MXNet, PyTorch
Radial Basis Function NN	RBF Network	Discriminative	Supervised and Unsupervised	K-means Clustering; Least Square Function	Radial Basis Function NN	Function approximation; Time series prediction	Fisher's Iris data set	TensorFlow
Kohonen Self Organizing NN	Nodes arranged in hexagonal or rectangular grid	Generative	Unsupervised	Competitive Learning	Kohonen Self Organizing NN	Dimensionality Reduction; Optimization problems; Clustering analysis	SPAMbase	TensorFlow

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

4. Training Algorithms

The learning algorithm constitutes the main part of Deep Learning. The number of layers differentiates the deep neural network from shallow ones. The higher the number of layers, the deeper it becomes. Each layer can be specialized to detect a specific aspect or feature.

As indicated by Maryam M Jajafabadi et al. [47], in case of image (face) recognitions, first layer can detect edges and the second can detect higher features such as various part of the face, e.g., ears, eyes, etc., and the third layer can go further up the complexity order by even learning facial shapes of various persons. Even though each layer might learn or detect a defined feature, the sequence is not always designed for it, especially in unsupervised learning. These feature extractors in each layer had to be manually programmed prior to the development of training algorithms such as gradient descent. These hand-crafted classifiers didn't scale for lager dataset or adapt to variation in the dataset. This message was echoed in the 1998 paper [28] by Yann Lecun et al., where they demonstrate that systems with more automatic learning and reduced manually designed heuristics yields far better pattern recognition.

Backpropagation provides representation learning methodology, where raw data can be fed without the need to manually massage it for classifiers, and it will automatically find the representations needed for classification or recognition [6]. The goal of the learning algorithm is to find the optimal values for the weight vectors to solve a class of problem in a domain.

Some of the well-known training algorithms are:

1. Gradient Descent
2. Stochastic Gradient Descent
3. Momentum
4. Levenberg–Marquardt algorithm
5. Backpropagation through time

4.1. Gradient Descent

Gradient descent (GD) is the underlying idea in most of machine learning and deep learning algorithms. It is based on the concept of Newton's Algorithm for finding the roots (or zero value) of a 2D function. To achieve this, we randomly pick a point in the curve and slide to the right or left along the x-axis based on negative or positive value of the derivative or slope of the function at the chosen point until the value of the y-axis, i.e., function or f(x) becomes zero. The same idea is used in gradient descent, where we traverse or descend along a certain path in a multi-dimensional weight space if the cost function keeps decreasing and stop once the error rate ceases to decrease. Newton's method is prone to getting stuck in local minima if the derivative of the function at the current point is zero. Likewise, this risk is also present when using gradient descent on a non-convex function. In fact, the impact is amplified in the multi-dimensional (each dimension represents a weight variable) and multi-layer landscape of DNN and it result in a sub-optimal set of weights. Cost function is one half the square of the difference between the desired output minus the current output as shown below.

$$C = \frac{1}{2}\left(y_{\text{expected}} - y_{actual}\right)^2 \tag{22}$$

Backpropagation methodology uses gradient descent. In backpropagation, chain rule and partial derivatives are employed to determine error delta for any change in the value of each weight. The individual weights are then adjusted to reduce the cost function after every learning iteration of training data set, resulting in a final multi-dimensional (multi-weight) landscape of weight values [6]. We process through all the samples in the training dataset before applying the updates to the weights. This process is repeated until objective (aka cost function) doesn't reduce any further.

Figure 10 shows the error derivatives in relation to outputs in each hidden layer, which is the weighted summation of the error derivates in relation to the inputs in the unit in the above layer. E.g., when $\partial E/\partial z_k$ calculated, the partial error derivative with respect to w_{jk} to is equal to$y_j \partial E/\partial z_k$.

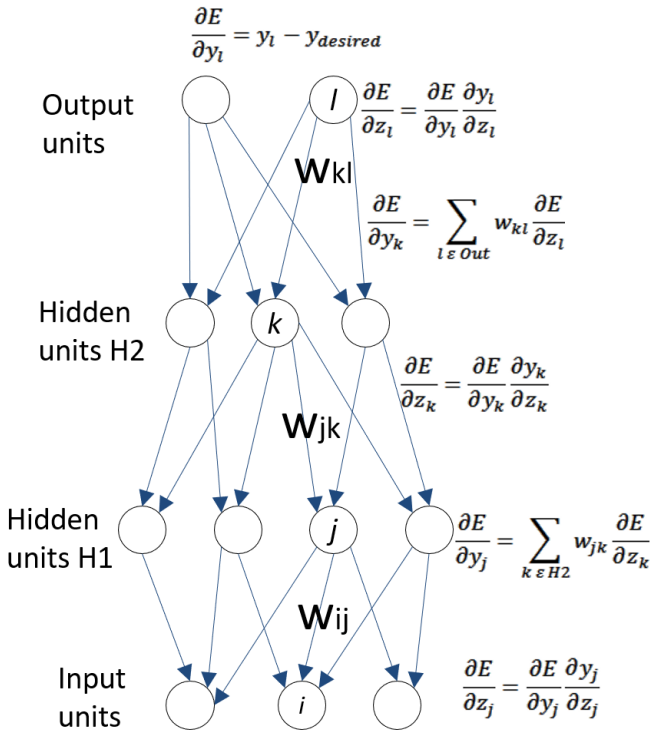

Figure 10. Error calculation in Multilayer Neural Network [6]

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

4.2. Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is the most common variation and implementation of gradient descent. In gradient descent, we process through all the samples in the training dataset before applying the updates to the weights. While in SGD, updates are applied after running through a minibatch of *n* number of samples. Since we are updating the weights more frequently in SGD than in GD, we can converge towards global minimum much faster.

4.3. Momentum

In the standard SGD, learning rate is used as a fixed multiplier of the gradient to compute step size or update to the weight. This can cause the update to overshoot a potential minima, if the gradient is too steep, or delay the convergence if the gradient is noisy. Using the concept of momentum in physics, the momentum algorithm presents a velocity v variable that configured as an exponentially decreasing average of the gradient [48]. This helps prevent costly descent in the wrong direction. In the equation below, $\alpha \in [0,1)$ is the momentum parameter and ϵ is the learning rate.

$$Velocity\ Update: v \leftarrow \alpha v - \epsilon g \quad (23)$$

$$Actual\ Update: \theta \leftarrow \theta + v \quad (24)$$

4.4. Levenberg-Marquardt algorithm

Levenberg-Marquadt algorithm (LMA) is primarily used in solving non-linear least squares problems such as curve fitting. In least squares problems, we try to fit a given data points with a function with the least amount of sum of the squares of the errors between the actual data points and points in the function. LMA uses a combination of gradient descent and Gauss-Newton method. Gradient descent is employed to reduce the sum of the squared errors by updating the parameters of the function in the direction of the steepest-descent, while the Gauss-Newton method minimizes the error by assuming the function to be locally quadratic and finds the minimum of the quadratic [49].

If the fitting function is denoted by $\hat{y}(t;\boldsymbol{p})$ and m data points denoted by (t_i, y_i), then the squared error can be written as [49]:

$$x^2(\boldsymbol{p}) = \sum_{i=1}^{m} \left[\frac{y(t_i) - \hat{y}(t_i;\mathbf{p})}{\sigma_{yi}}\right]^2 \quad (25)$$

$$= (y - \hat{y}(p))^T W \left(y - \hat{y}(p)\right) \quad (26)$$

$$= y^T W y - 2y^T W \hat{y} + \hat{y}^T W \hat{y} \quad (27)$$

where the measurement error for y (t_i), i.e., σ_{yi} is the inverse of the weighting matrix W_{ii}.

The **gradient descent** of the squared error function in relation to the *n* parameters can be denoted as [49]:

$$\frac{\partial}{\partial \mathbf{p}} x^2 = 2(y - \hat{y}(p))^T W \frac{\partial}{\partial \mathbf{p}} \left(y - \hat{y}(p)\right) \quad (28)$$

$$= 2(y - \hat{y}(p))^T W \left[\frac{\partial \hat{y}(p)}{\partial \mathbf{p}}\right] \quad (29)$$

$$= 2(y\ \hat{y})^T W\ \mathbf{J} \quad (30)$$

$$\mathrm{h_{gd}} = \alpha\, \mathbf{J}^T W\, (y - \hat{y}) \quad (31)$$

where J is the Jacobian matrix of size *m x n* used in place of the $[\partial \hat{y}/\partial \mathrm{p}]$, and $\mathrm{h_{gd}}$ is the update in the direction of the steepest gradient descent.

The equation for the Gauss-Newton method update ($\mathrm{h_{gn}}$) is as follows [49]:

$$[\mathrm{J}^T \mathrm{W} \mathrm{J}]\mathrm{h_{gn}} = \mathrm{J}^T \mathrm{W}(y - \hat{y}) \quad (32)$$

The Levenberg- Marquardt update [$\mathrm{h_{lm}}$] is generated by combining gradient descent and Gauss-Newton methods resulting in the equation below [49]:

$$[\mathrm{J}^T \mathrm{W} \mathrm{J} + \lambda\, \mathrm{diag}\,(\mathrm{J}^T \mathrm{W} \mathrm{J})]\, \mathrm{h_{lm}} = \mathrm{J}^T \mathrm{W}(y - \hat{y}) \quad (33)$$

4.5. Backpropagation through time

Backpropagation through time (BPTT) is the standard method to train the recurrent neural network. As shown in Figure 2b, the unrolling of RNN in time makes it appears like a feedforward network. But unlike the feedforward network, the unrolled RNN has the same exact set of weight values for each layer and represents the training process in time domain. The backward pass through this time domain network calculates the gradients with respect to specific weights at each layer. It then averages the updates for the same weight at different time increments (or layers) and changes them to ensure the value of weights at each layer continues to stay uniform.

4.6. Comparison of Deep Learning Algorithms

Table 3 provides a summary and comparison of common deep learning algorithms. The advantages and disadvantages are presented along with techniques to address the disadvantages. Gradient descent-based training is the most common type of training. Backpropagation through time is the backpropagation tailored for recurrent neural network.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

Contrastive divergence finds its use in probabilistic models such as RBMs. Evolutionary algorithms can be applied to hyperparameter optimizations or training models by optimizing weights. Reinforcement learning could be used in game theory, multi-agent systems and other problems where both exploitation and exploration need to be optimized.

TABLE 3. DEEP LEARNING ALGORITHM COMPARISON TABLE

Algorithm	Advantages	Disadvantages	Techniques to address disadvantages
(Batch) Gradient Descent	Scales well after optimizations	Takes a long time to converge as weights are updated after the entire dataset pass	Mini-Batch Gradient Descent
		Local minima	Please see table 4
Stochastic Gradient Descent	Scales well after optimizations	Noisy error rates since it is calculated at every sample; Accuracy requires random order	Mini-Batch Gradient Descent; Shuffle data after every epoch
		Local minima	Please see table 4
Back Propagation through Time	Performs better than metaheuristics (e.g., genetic algorithm)	Hard to be used in the application where online adaption is required as the entire time series must be used	Truncate part of time instead of entire time
Contrastive divergence	Can create samples that appear to come from input data distribution; Generative models; Pattern completion	Difficult to train	Get sampling from Monte Carlo Markov Chain
Evolutionary Algorithms	Is able to explore and exploit solutions space effectively	Takes long time to run as it needs to test different combinations	Utilize cloud and GPUs
Reinforcement Learning (Q-learning)	Is able to balance exploration and exploitation	In some cases, reward is extremely rare	Work backwards from the reward state

5. Shortcomings of Training Algorithms

There are several shortcomings with the standard use of training algorithms on DNNs. The most common ones are described here.

5.1. Vanishing and Exploding Gradients

Deep neural networks are prone to vanishing (or exploding) gradients due to the inherent way in which gradients (or derivates) are computed layer by layer in a cascading manner with each layer contributing to exponentially decreasing or increasing derivatives. Weights are increased or decreased based on gradients to reduce the cost function or error. Very small gradients can cause the network to take a long time to train, whereas large gradients can cause the training to overshoot and diverge. This is made worse by the non-linear activation functions like sigmoid and tanh functions that squash the outputs to a small range. Since change in weight have nominal effect on the output training could take much longer. This problem can be mitigated using linear activation function like ReLu and proper weight initialization.

5.2. Local Minima

Local minima is always the global minima in a convex function, which makes gradient descent based optimization fool proof. Whereas in nonconvex functions, backpropagation based gradient descent is particularly vulnerable to the issue of premature convergence into the local minima. A local minima as shown in Figure 11, can easily be mistaken for global absolute minima.

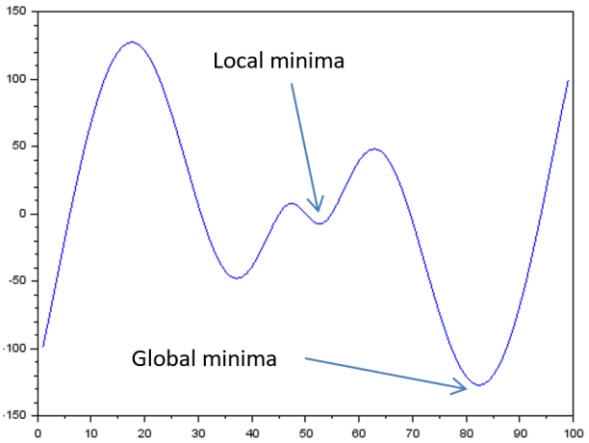

Figure 11. Gradient Descent

5.3. Flat Regions

Just like local minima, flat regions or saddle points (Figure 12) also pose similar challenge for gradient descent based optimization in nonconvex high-dimensional functions. The training algorithm could potentially mislead by this area as the gradient comes to a halt at this point.

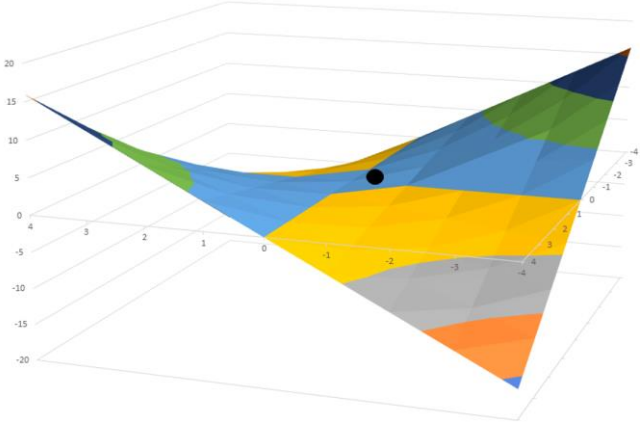

Figure 12. Flat (saddle point marked with black dot) region in a nonconvex function

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

5.4. Steep Edges

Steep edges are another section of the optimization surface area where the steep gradient could cause the gradient descent-based weight updates to overshoot and miss a potential global minima.

5.5. Training Time

Training time is an important factor to gauge the efficiency of an algorithm. It is not uncommon for graduate students to train their model for days or weeks in the computer lab. Most models require exorbitant amount of time and large datasets to train. Often times many of the samples from the datasets do not add value to the training process and in some cases, they introduce noise and adversely affect the training.

5.6. Overfitting

As we add more neurons to DNN, it can undoubtedly model the network for more complex problems. DNN can lend itself to high conformability to training data. But there is also a high risk of overfitting to the outliers and noise in the training data as shown in Figure 13. This can result in delayed training and testing times and result in the lower quality prediction on the actual test data. E.g., in classification or cluster problems, overfitting can create a high order polynomial output that separates the decision boundary for the training set, which will take longer and result in degraded results for most test data set. One way to overcome overfitting is to choose the number of neurons in the hidden layer wisely to match the problem size and type. There are some algorithms that can be used to approximate the appropriate number of neurons but there is no magic bullet and the best bet is to experiment on each use case to get an optimal value.

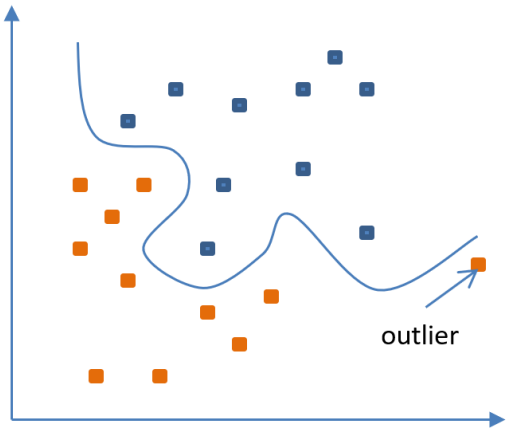

Figure 13. Overfitting in Classification

6. Optimization of Training Algorithms

The goal of the DNN is to improve the accuracy of the model on test data. Training algorithms aims to achieve the end goal by reducing the cost function. The common root cause of three out of five shortcomings mentioned above is primarily due to the fact that the training algorithms assume the problem area to be a convex function. The other problem is high number of nodes and the sheer possible combination of weight values they can have. While weights are learned by training on the dataset, there are additional crucial parameters referred to as hyperparameters that aren't directly learnt from training dataset. These hyperparameters can take a range of values and add complexity of finding the optimal architecture and model. There is significant room for improvement to the standard training algorithms. Here are some of the popular ways to enhance the accuracy of the DNNs.

6.1. Parameter Initialization Techniques

Since the solution space is so huge, the initial parameters have an outsized influence on how fast or slow the training converges, if at all or if it prematurely converges to a suboptimal point. Initialization strategies tend to be heuristic in nature. [50] proposed normalized initialization where weights are initialized in the following manner.

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right] \qquad (34)$$

[51] proposed another technique called sparse initialization, where the number of non-zero incoming weights were capped at a certain limit causing them to retain high diversity and reduce chances of saturation.

6.2. Hyperparameter Optimization

The learning rate and regularization parameters constitutes the commonly used hyperparameters in DNN. Learning rate determines the rate at which the weights are updated. The purpose of regularization is the prevent overfitting and regularization parameter affects the degree of influence on the loss function. CNN's have additional hyperparameters i.e., number of filters, filter shapes, number of dropouts and max pooling shapes at each convolution layer and number of nodes in the fully connected layer. These parameters are very important for training and modeling a DNN. Coming up with an optimal set of parameter values is a challenging feat. Exhaustively iterating through each combination of hyperparameter values is computationally very expensive. For example, if training and evaluating a DNN with the full dataset takes ten minutes, then with seven hyperparameters each with eight potential values will take (8^7 x 10 min), i.e., 20,971,520 minutes or almost 40 years to exhaustively train and evaluate the network on all combinations of the hyperparameter values. Hyperparameter can be optimized with different metaheuristics. Metaheuristics are nature inspired guiding principles that can help in traversing the search space more intelligently yet much faster than the exhaustive method.

Particle Swarm Optimization (PSO) is another type of metaheuristic that can be used for hyperparameter optimization. PSO is modeled around the how birds fly

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

around in search of food or during migration. The velocity and location of birds (or particles) are adjusted to steer the swarm towards better solution in the vast search space. Escalante et al. used PSO for hyperparameter optimization to build a competitive model that ranked among the top relative to other comparable methods[52].

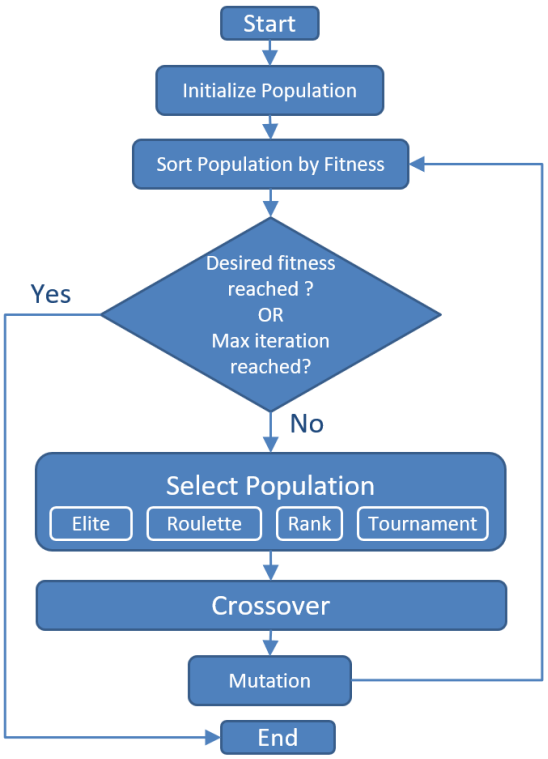

Figure 14a. Genetic Algorithm [53]

Genetic algorithm (GA) is a metaheuristic that is commonly used to solve combinatorial optimization problems. It mimics the selection and crossover processes of species reproduction and how that contributes to evolution and improvement of the species prospect of survival. Figure 14a shows a high-level diagram of the GA. Figure 14b illustrates the crossover process where parts of the respective genetic sequence are merged from both the parents to form the new genetic sequence in the children. The goal is to find a population member (a sequence of numbers resembling DNA nucleotides) that meets the fitness requirement. Each population member represents a potential solution. Population members are selected based on different methods, e.g., elite, roulette, rank and tournament.

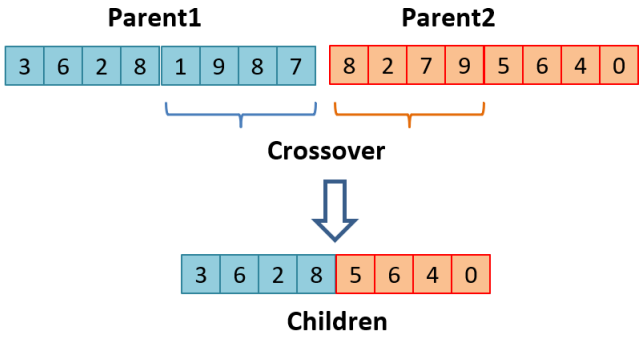

Figure 14b. Crossover in Genetic Algorithm

Elite method ranks population members by fitness and only uses high fitness members for the crossover process. The mutation process then makes random changes to the number sequence and the entire process continues until a desired fitness or maximum number of iterations are reached. [53, 54] propose parallelization and hybridization of GA to achieve better and faster results. Parallelization provide both speedup and better results as we can periodically exchange population members between the distributed and parallel operations of genetic algorithms on different set of population members. Hybridization is the process of mixing the primary algorithm (GA in this case) with other operations, like local search. Shrestha and Mahmood [53] incorporated 2-Opt local search method into GA to improve the search for optimal solution. [55] postulates that correctly performed exchanges (e.g., in GA) breeds innovation and results in creation solutions to hard problems just like in real life where collaboration and exchanges between individuals, organizations and societies. In additional to GA, other variations of evolution-based metaheuristics have also been used to evolve and optimize deep learning architectures and hyperparameters. E.g., [56] proposed CoDeepNEAT framework based on deep neuroevolution technique for finding an optimized architecture to match the task at hand.

6.3. Adaptive Learning Rates

Learning rates have a huge impact on training DNN. It can speed up the training time, help navigate flat surfaces better and overcome pitfalls of non-convex functions. Adaptive learning rates allow us to change the learning rates for parameters in response to gradient and momentum. Several innovative methods have been proposed. [48] describes the following:

1. Delta-bar Algorithm
2. AdaGrad
3. RMSProp
4. Adam

In **Delta-bar** algorithm, the learning rate of the parameter is increased if the partial derivative with respect to it stays in the same sign and decreased if the sign changes. **AdaGrad**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

is more sophisticated [57] and prescribes an inversely proportional scaling of the learning rates to the square root of the cumulative squared gradient. AdaGrad is not effective for all DNN training. Since the change in the learning rate is a function of the historical gradient, AdaGrad becomes susceptible to convergence.

RMSProp algorithm is a modification of AdaGrad algorithm to make it effective in a nonconvex problem space. RMSProd replaces the summation of squared gradient in AdaGrad with exponentially decaying moving average of the gradient, effectively dropping the impact of historical gradient [48]. **Adam** which denotes adaptive moment estimation is the latest evolution of the adaptive learning algorithms that integrates the ideas from AdaGrad, RMSProp and momentum [58]. Just like AdaGrad and RMSProd, Adam provides an individual learning rate for each parameter. Adam includes the benefits of both the earlier methods does a better job handling non-stationary objectives and both noisy and sparse gradients problems [58]. Adam uses first moment (i.e., mean as used in RMSProp) as well as second moments of the gradients (uncentered variance) utilizing the exponential moving average of squared gradient [58].

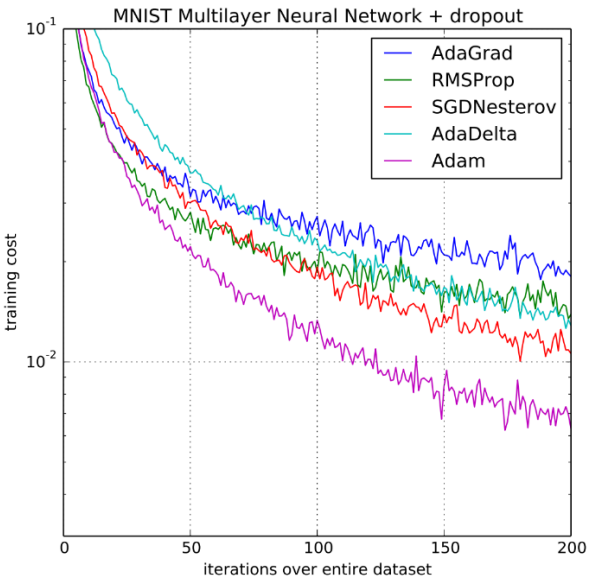

Figure 15. Multilayer network training cost on MNIST dataset using different adaptive learning algorithms [58]

Figure 15 shows the relative performance of the various adaptive learning rate mechanisms where Adam outperform the rest.

6.4. Batch Normalization

As the network is getting trained with variations to weights and parameters, the distribution of actual data inputs at each layer of DNN changes too, often making them all too large or too small and thus making them difficult to train on networks, especially with activation functions that implement saturating nonlinearities, e.g., sigmoid and tanh functions. Iofee and Szegedy [59] proposed the idea of batch normalization in 2015. It has made a huge difference in improving the training time and accuracy of DNN. It updates the inputs to have a unit variance and zero mean at each mini-batch.

6.5. Supervised Pretraining

Supervised pretraining constitutes breaking down complex problems into smaller parts and then training the simpler models and later combining them to solve the larger model. Greedy algorithms are commonly used in supervised pretraining of DNN.

6.6. Dropout

There are few commonly used methods to lower the risk of overfitting. In the dropout technique, we randomly choose units and nullify their weights and outputs so that they do not influence the forward pass or the backpropagation. Figure 16 shows a fully connected DNN on the left and a DNN with dropout to the right. The other methods include the use of regularization and simply enlarging the training dataset using label preserving techniques. Dropout works better than regularization to reduces the risk of overfitting and also speeds up the training process. [60] proposed the dropout technique and demonstrated significant improvement on supervised learning based DNN for computer vision, computational biology, speech recognition and document classification problems.

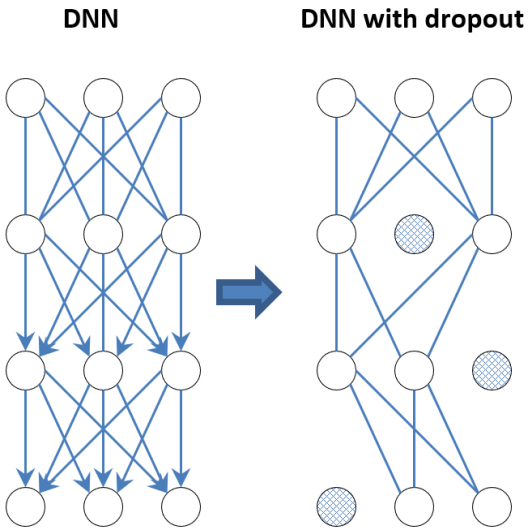

Figure 16. DNN with and without Dropout

6.7. Training Speed up with Cloud and GPU processing

Training time is one of the key performance indicators of machine learning. Cloud computing and GPUs lend themselves very well to speeding up the training process. Cloud provides massive amounts of compute power and now all major cloud vendors include GPU powered servers that can easily be provisioned and used for training DNNs on demand at competitive prices. Cloud vendor Amazon Web

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

Services' (AWS) P2 instances provides up to 40 thousand parallel GPU cores and its P3 GPU instances are further optimized for machine learning [61].

6.8. Summary of DL Algorithms Shortcomings and Resolutions Techniques

Table 4 provides a summary of deep learning algorithm shortcomings and resolutions techniques. The table also lists the cause and effect[s] of the shortcomings.

TABLE 4. DL ALGORITHM SHORTCOMINGS & RESOLUTION TECHNIQUES

Shortcomings	Cause	Effect	Optimizations to address Shortcomings
Vanishing and Exploding Gradients	Propagation of derivatives	Long training time; Overshoot global minima; Halts training	ReLu activation function; Weight initialization; Connection between the forget gate activations and the gradients computation in LSTM (RNN); Skipping connections (ResNet); Faster hardware (GPUs)
Local Minima / Flat regions	Gradient descent of non-convex problem space	Convergence into local minima	Adaptive learning rates; Parameter Re-initialization / initialization techniques; Adaptive learning rates
Steep Edges	Gradient descent of non-convex problem & steep spaces	Overshoot to miss global minima	Adaptive learning rates
Overfitting	Oversized nodes (network) relative to dataset; poor dataset	Poor accuracy of the model	Regularization; Dropout; Choosing correct size of network; Better training data
High Training Time	Large network (weights & hyperparameters), high dimensional data, others	Poor use of compute resources; wasted time	Adaptive learning rates; Using Cloud and GPUs
Hyperparameter selection	Extremely large solution space (NP-hard problem) for high dimensional problems	Convergence into local minima	Metaheuristics (e.g., Genetic Algorithm) for hyperparameter optimization; Random search; Sequential Model-based Algorithm Configuration

7. Architectures & Algorithms – Implementations

This section describes different implementations of neural networks using a variety of training methods, network architectures and models. It also includes models and ideas that have been incorporated into machine learning in general.

7.1. Deep Residual Learning

The ability to add more layers to DNN has allowed us to solve harder problems. Microsoft Research Asia (MSRA) applied a 100/1000 layer deep residual network (ResNet) on CIFAR-10 dataset and won 1st place in the ILSVRC 2015 competition with a 152-layer DNN on the ImageNet dataset [62]. Figure 17 demonstrates a simplified version of Microsoft's winning deep residual learning model. Despite the depth of these networks, simply adding more layers to DNN does not improve or guarantee results. To the contrary, it degrades the quality of the solution. This makes training DNN not so straight forward. The MSRA team was able to overcome the degradation by making the hoping stacked layers match a residual mapping instead of the desired mapping with the following function [62]:

$$F(x) := H(x) - xv \qquad (30)$$

Where *F(x)* is the residual mapping and *H(x)* is the desired mapping, and then by recasting the desired mapping at the end [62]. According to MSRA team, it is much easier to optimize the residual mapping.

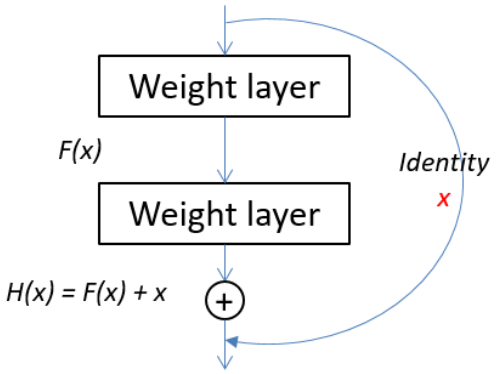

Figure 17. Deep Residual Learning model by MSRA at Microsoft

7.2. Oddball Stochastic gradient descent

All training data are not created equal. Some will have higher training error than the others. Yet, we assume that they are the same and thus use each training examples the same number of times. Andrew Simpson [63] argues that this assumption is invalid and makes a case in his paper for the number of times a training examples is used to be proportional to its respective training error. So, if a training example has a higher error rate, it will be used to train the network higher number of times than the other training example. Andrew Simpson [63] proves his methodology, termed Oddball Stochastic Gradient Descent with a training set of 1000 video frames. Simpson [63] created a training selection probability distribution for training example based on the error value and pegged the frequency of using the training example based on the distribution.

7.3. Deep belief Network

Xue-wen Chen et al. [23] highlights the fact that conventional neural network can easily get stuck in local

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

minima when the function is non-convex. They propose a DNN architecture called large scale deep belief network (DBN) that uses both labeled and unlabeled to learn feature representations. DBN are made up of layers of RBM stacked together and learn probability distribution of the input vectors. They employ unsupervised pre-training and fine-tuned supervised algorithms and techniques to mitigate the risk of getting trapped in local minima. Below is the equation [23] for change in weights, where c is the momentum factor and α is the learning rate, and v and h are visible and hidden units respectively.

$$\Delta w_{ij}(t+1) = c\Delta w_{ij}(t) + \alpha(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}) \quad (35)$$

Equation [23] for probability distribution for hidden and visible inputs.

$$p(h_j = 1 \mid \mathrm{v}; \mathrm{W}) = \sigma\left(\sum_{i=1}^{I} w_{ij} v_i + a_j\right) \quad (36)$$

$$p(v_i = 1 \mid \mathrm{h}; \mathrm{W}) = \sigma\left(\sum_{j=1}^{J} w_{ij} h_j + b_i\right) \quad (37)$$

7.4. Big data

Big data provides tremendous opportunity and challenge for deep learning. Big data is known for the 4 Vs (volume, velocity, veracity, variety). Unlike the shallow networks, the huge volume and variety of data can be handled by DNNs and significantly improve the training process and the ability to fit more complex models. On the flip side, the sheer velocity of data that is generated in real-time can be daunting to process. Maryam M Jajafabadi et al. [47] raises similar challenges learning from real-time streaming data such as credit cards usage to monitor for fraud detection. They propose using parallel and distributed processing with thousands of CPU cores. In addition, we should also use cloud providers that support auto-scaling based on usage and workload. Not all data represent the same quality. In the case of computer vision, images from constrained sources, e.g., studios are much easier to recognize that the ones from unconstrained sources like surveillance cameras. [64] proposes a method to utilize multiple images of the unconstrained source to enhance the recognition process.

Deep learning can help mine and extract useful patterns from big data and build models for inference, prediction and business decision making. There is massive volumes of structured and unstructured data and media files getting generated today making information retrieval very challenging. Deep learning can help with semantic indexing to enable information to be more readily accessible in search engines [14, 65]. This involves building models that provide relationships between documents and keywords the contain to make information retrieval more effective.

7.5. Generative top down connection (generative model)

Much of the training is usually implemented with bottom-up approach, where discriminatory or recognition models are developed using backpropagation. A bottom-up model is one that takes the vector representation of input objects and computes higher level feature representations at subsequent layer with a final discrimination or recognition pattern at the output layer. One of the shortcomings of backpropagation is that it requires labeled data to train. Geoffrey Hinton proposed a novel way of overcoming this limitation in 2007 [66]. He proposed a multi-layer DNN that used generative top-down connection as opposed to bottom-up connection to mimic the way we generate visual imagery in our dream without the actual sensory input. In top-down generative connection, the high-level data representation or the outputs of the networks are used to generate the low-level raw vector representations of the original inputs, one layer at a time. The layers of feature representations learned with this approach can then be further perfected either in generative models such as auto-encoders or even standard recognition models [66].

In the generative model in Figure 18, since the correct upstream cause of the events in each layer is known, a comparison between the actual cause and the prediction made by the approximate inference procedure can be made, and the recognition weights, r_{ij} can be adjusted to increase the probability of correct prediction.

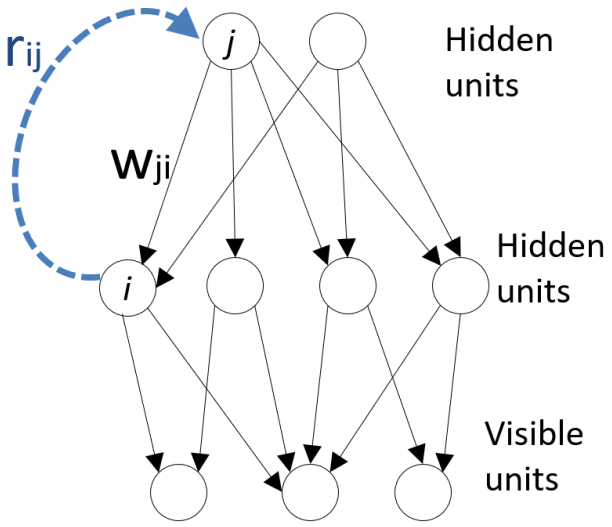

Figure 18. Learning multiple layers of representation

Here is the equation [66] for adjusting the recognition weights r_{ij}.

$$\Delta r_{ij} \, \alpha \, h_i \left(h_j - \sigma\left(\sum_i h_i r_{ij} \right) \right) \quad (38)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

7.6. Pre-training with unsupervised Deep Boltzmann Machines.

Vast majority of DNN training is based on supervised learning. In real life, our learning is based on both supervised and unsupervised learning. In fact, most of our learning is unsupervised. Unsupervised learning is more relevant in today's age of big data analytics because most raw data is unlabeled and un-categorized [47]. One way to overcome the limitation of backpropagation, where it gets stuck in local minima is to incorporate both supervised and unsupervised training. It is quite evident that top-down generative unsupervised learning is good for generalization because it is essentially adjusting the weights by trying to match or recreate the input data on layer at a time [67]. After this effective unsupervised pre-training, we can always fine-tune it with some labeled data. Geoffrey Hinton and Ruslan Salakhutdinov describe multiple layers of RBMs that are stacked together and trained layer by layer in a greedy, unsupervised way, essentially creating what is called the Deep Belief Network. They further modify stacks to make them un-directed models with symmetric weights, thus creating the Deep Boltzmann Machines (DBM). Four layered deep belief network and deep Boltzmann machines are shown in Figure 19. In [67] the DBM layers were pre-trained one at a time using unsupervised method and then tweaked using supervised backpropagation on the MNIST and NORB datasets as shown in Figure 20. They [67] received favorable results validating benefits of combining supervised and unsupervised learning methods.

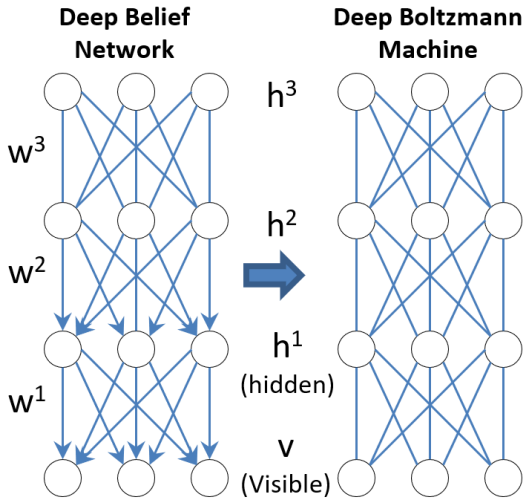

Figure 19. Four-layer DBN & four-layer Deep Boltzmann Machine

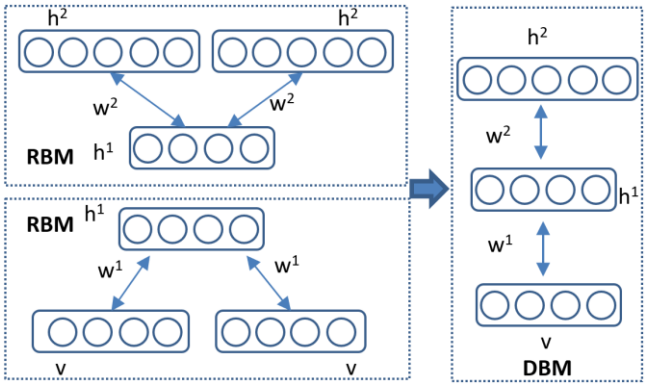

Figure 20. Pretraining of stacked & altered RBM to create a DBM [67]

Here are the equations showing probability distributions over visible and two hidden units in DBM (after unsupervised pre-training) [67].

$$p(v_i = 1|\mathrm{h}^1) = \sigma\left(\sum_j W_{ij}^1 h_j\right) \quad (39)$$

$$p(h_m^2 = 1|\mathrm{h}^1) = \sigma\left(\sum_j W_{jm}^2 h_j^1\right) \quad (40)$$

$$p\left(h_j^1 = 1\middle|\mathrm{v}, \mathrm{h}^2\right) = \sigma\left(\sum_i W_{ij}^1 v_i + \sum_m W_{jm}^2 h_m^2\right) \quad (41)$$

Post unsupervised pre-training, the DBM is converted into a deterministic multi-layer neural network by fine-tuning the network with supervised learning using labeled data as demonstrated in Figure 21. The approximate posterior distribution $q(\mathrm{h}|\mathrm{v})$ is generated for each input vector and the marginals $q(h^2_j=1|v)$ are added as an additional input for the network as shown in the figure above and subsequently, backpropagation is used to fine-tune the network [67].

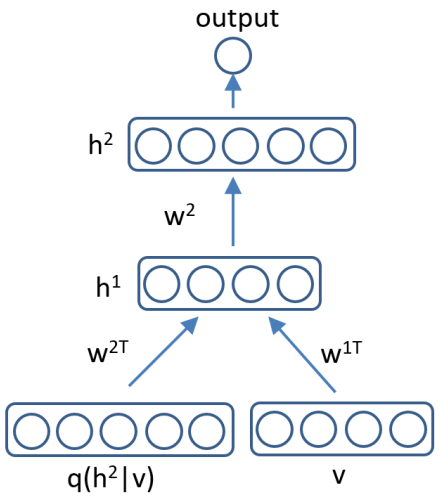

Figure 21. DBM getting initialized as deterministic neural network with supervised fine-tuning [67]

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

7.7. Extreme Learning Machine (ELM)

There have been other variations of learning methodologies. While more layers allow us to extract more complex features and patterns, some problems might be solved faster and better with less number of layers. [68] proposed a four-layered CNN termed *DeepBox* that outperformed larger networks in speed and accuracy. for evaluating objectness. ELM is another type of neural network with just one hidden layer. Linear models are learnt from the dataset in a single iteration by adjusting the weights between the hidden layer and the output, whereas the weights between the input and the hidden layers are randomly initialized and fixed [69].

ELM can obviously converge much faster than backpropagation, but it can only be applied to simpler problems of classifications and regression. Since proposing ELM in 2006, Buang-Bin Huang et al. came up with a multilayer version of ELM in 2016 [70] to take on more complex problems. They combined unsupervised multilayer encoding with the radom initialization of the weights and demonstrate faster convergence or lower training time than the state of the art multilayer perceptron training algorithm.

7.8. Multiobjective Sparse Feature Learning Model

Moaguo et al. [71] developed a multi-objective sparse feature learning (MO-SFL) model based on auto encoder, where they used an evolutionary algorithm to optimize two competing objectives of sparsity of hidden units and the reconstruction error (input vendor of AE). It fairs better than models where the sparsity is determined by human intervention or less than optimal methods.

Since the time complexity of evolutionary algorithms are high, they [71] utilize self-adaptive multi-objective differential evolution (DE) based on decomposition (Sa-MODE/D) to cut down on time and demonstrate it has better results than standard AE (auto encoder), SR-RBM (Sparse response RMB) and SESM (sparse encoding symmetric machine) by testing with MNIST dataset and compare the results with other implementations. Their learning procedure continuously iterates between evolutionary optimization step and the stochastic gradient descent to optimize the reconstruction error [71].

- Step 1: Multi-objective optimization to select the most optimal point in the pareto frontier for both objectives
- Step 2: Optimize parameters θ and θ' with stochastic gradient descent in the following reconstruction error function (of Auto Encoder), where *D* is the training data set and *L (x,y)* is the loss function with x representing the input and y representing the output, i.e., reconstructed input.

$$\sum_{x \in D} L(x, g_{\theta'}(f_{\theta}(\mathrm{x}))) \qquad (42)$$

Figure 22 shows a pareto frontier function that can be used to achieve a compromise between two competing objectives functions.

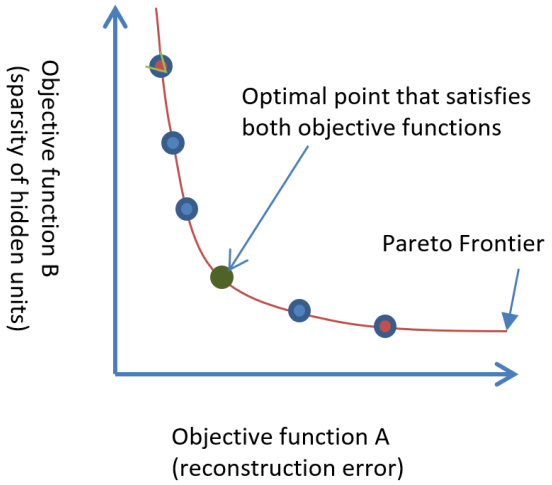

Figure 22. Pareto Frontier

7.9. Multiclass Semi-Supervised Learning Based on Kernel Spectral Clustering

Mehrkanoon et al. [72] proposed a multiclass learning algorithm based on Kernel Spectral Clustering (KSC) using both labeled and unlabeled data. The novelty of their proposal is the introduction of regularization terms added to the cost function of KSC, which allow labels or membership to be applied to unlabeled data examples. It is achieved in the following way [72]:

- Unsupervised learning based on kernel spectral clustering (KSC) is used as the core model
- A regularization term is introduced and labels (from labeled data) are added to the model

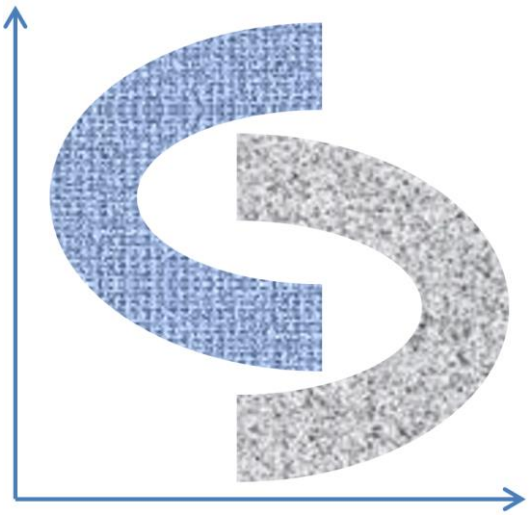

Figure 23. Spectral Clustering Representation

Figure 23 illustrates data points in a spectral clustering representation. Spectral clustering (SC) is an algorithm that divides the data points in a graph using Laplacian or double derivative operation, whereas KSC is simply an extension of SC that uses Least Squares Support Vector Machines methodology [73].

Since unlabeled data is more abundantly available relative to labeled data, it would be beneficial to make the most of it with unsupervised or in this case semi-supervised learning.

7.10. Very Deep Convolutional Networks for Natural Language Processing

Deep CNN have mostly been used in computer vision, where it is very effective. Conneau et al. [74] used it for the first time to NLP with up to 29 convolution layers. The goal is to analyze and extract layers of hierarchical representations from words and sentences at the syntactic, semantic and contextual level. One the major setbacks for lack of earlier deep CNN for NLP is because of deeper networks tend to cause saturation and degradation of accuracy. This is in addition to the processing overhead of more layers. Kaiming et al. [62] states that the degradation is not caused by overfitting but because deeper systems are difficult to optimize. [62] addressed this issue with shortcut connections between the convolution blocks to let the gradients to propagate more freely and they, along with [74] were able to validate the benefits of the shortcuts with 10/101/152-layers and 49 layers respectively. Conneau et al. [74] architecture consists of series of convolution blocks separated by pooling that halved the resolution followed by k-max pooling and classification at the end.

7.11. Metaheuristics

Metaheuristics can be used to train neural networks to overcome the limitation of backpropagation-based learning. When implementing metaheuristics as training algorithm, each weight of the neural network connection is represented by a dimension in the multi-dimensional solution search space of the problem we are trying to solve. The goal is to come as near as possible to the optimal values of weights, i.e., a location in the search space that represents the global best solution. Particle Swarm Optimization (PSO) is a type of metaheuristic inspired by the movement of birds in the sky consists of particles or candidate solutions move about in a search space to reach a near optimal solution. In their paper [75], N. Krpan and D. Jakobovic ran parallel implementations using backpropagation and PSO. Their results demonstrate that while parallelization improves the efficacy of both algorithms, parallel backpropagation is efficient only on large networks, whereas parallel PSO has wider influence on various sizes of problems.

Similarly, W. Dong and M. Zhou [76] complemented PSO with supervised learning control module to guide the search for global minima of an optimization problem. The supervised learning module provided real-time feedback with *back diffusion* (BD) to retain diversity and *social attractor renewal* to overcome stagnation [76]. Metaheuristics provide high level guidance inspired by nature and applies them to solve mathematical problems. In a similar way [77] proposes incorporating the concepts of intelligent teacher and privileged information, which is essentially extra information available during training but not during evaluation or testing, into the DNN training process.

7.12. Genetic Algorithm

Genetic Algorithm is a metaheuristic that can be effectively used in training DNN. GA mimics the evolutionary processes of selection, crossover and mutation. Each population member represents a possible solution with a set of weights. Unlike PSO, which includes only one operator for adjusting the solution, evolutionary algorithms like GA includes various steps, i.e., selection, crossover and mutation methods [52]. Population members undergo several iterations of selection and crossover based on known strategies to achieve better solution in the next iteration or generation. GA has undergone decades of improvement and refinements since it was first proposed in 1976 **[78]**. There are several ways to perform selections, e.g., elite, roulette, rank, tournament **[79]**. There are about dozen ways to perform crossovers by Larrañaga et al. alone **[80]**. Selection methodologies represent exploration of the solution space and crossovers represent the exploitation of the selected solution candidates. The goal is to get better solution wider exploration and deeper exploitation. Additional tweaking can be introduced with mutation. Parallel clusters of GA can be executed independently in islands and few members exchanged between the island every so often **[81]**. In addition, we can also utilize local search such as greedy algorithm, Nearest Neighbor or K-opt algorithm to further improve the quality of the solution.

Lin et al. **[82]** demonstrated a successful incorporation of GA that resulted in better classification accuracy and performance of a Polynomial Neural Network. Standard GA operations including selection, crossover and mutation were used on parameters that included partial descriptions (PDs) of inputs in the first layer, bias and all input features **[82]**.

GA was further enhanced with the incorporation of the concept of mitochondrial DNA (mtDNA). In evolution, it is quite evident from casual observation and simple reason that crossover of population members with too much similarity does not yield much variance in the offspring. Likewise, we can infer that in GA, selection and crossover between solutions that are very similar would not result is high degree of exploration of the multi-dimensional solution space. In fact, it might run the risk of getting pigeonholed into a restricted pattern.

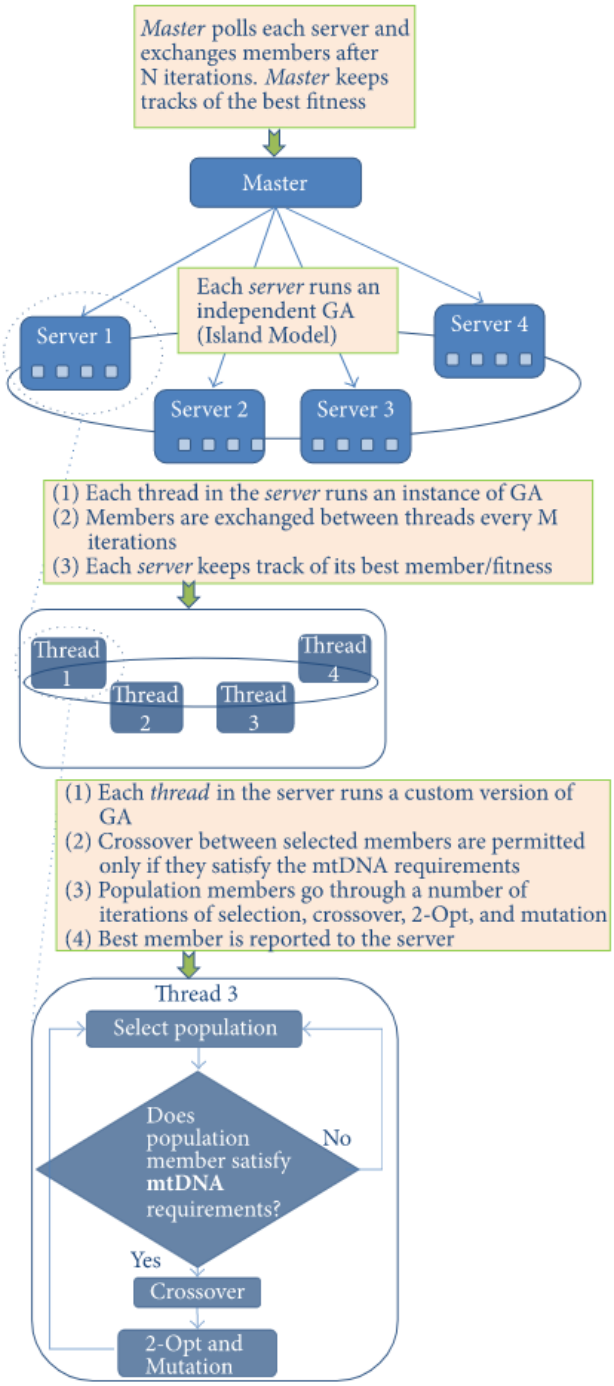

Figure 24. Continental Model with mtDNA [53]

Diversity is the key to overcoming the risk of getting stuck in local minima. This risk can be mitigated by exploiting the idea of mtDNA. mtDNA represents one percent of the human chromosomes **[83]**. The concept of incorporating mitochondrial DNA into GA was introduced by Shrestha and Mahmood **[53]**. They describe a way to restrict crossover between population members or solution candidates based proximity on their mtDNA value **[53]**. Unlike the rest of the 99% DNA, mtDNA is only inherited from the female, thus it is a more continuous marker of lineage or genetic proximity. The premise behind this is that offspring of population members with similar genetic makeup doesn't help with overcoming the local minima. Figure 24 describes the parallel and distributed nature of their full implementation **[53]** along with the GA operators (selection, mutation and mtDNA incorporated crossover). The training process is enhanced **[53]** with the implementation of continental model, where distributed servers run multiple threads, each running an instance of GA with mtDNA. Population members are then exchanged between the servers after fixed number of iterations as shown in Figure 24.

7.13. Neural Machine Translation (NMT)

Neural Machine Translation is a turnkey solution used in translation of sentences. While it provides some improvement over the traditional Statistical machine translation (SMT), it is not scalable for large models or datasets. It also requires lot of computational power for training and translation, and has difficult with rare words. For these reason, large tech companies like Google and Microsoft have both improved on NMT and have their own implementations of NMT, labeled as Google Neural Machine Translation (GNMT) and Skype Translator respectively. GMNT as shown in Figure 257 consists of encoder and decoder LSTM blocks organized in layers was presented in 2016 in [84]. It overcomes the shortcomings of NMT with enhanced deep LSTM neural network that includes 8 encoder and 8 decoder layers, and a method to break down rare difficult words to infer their meaning. On Conference on Machine Translation in 2014, GNMT received results at par with state-of-the-art for English-to-French and English-to-German language benchmarks [84].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

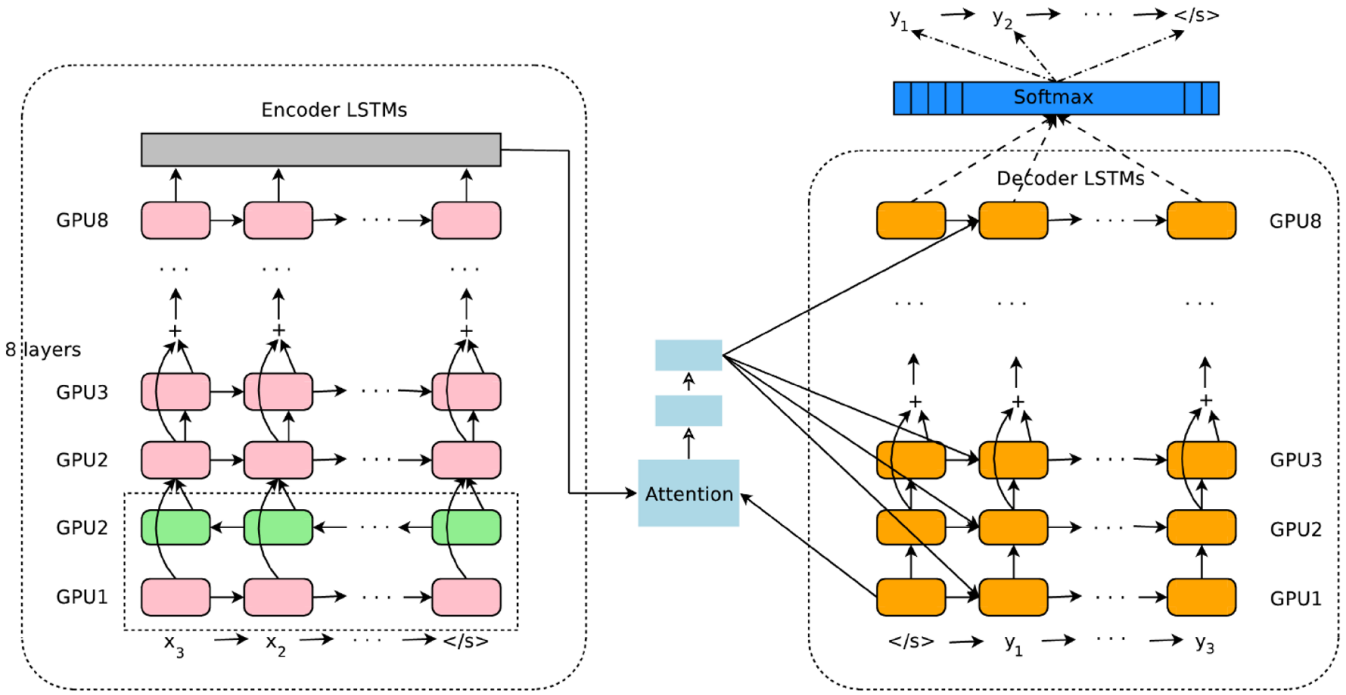

Figure 25. GNMT Architecture [84] with encoder neural network on the left and decoder neural network on the right.

7.14. Multi-Instance Multi-Label Learning

Images in real life include multiple instances (objects) and need multiple labels to describe them. E.g., a picture of an office space could include a laptop computer, a desk, a cubicle and a person typing on the computer. Zhang et al. [85] proposed MIML (Multi-Instance Multi-Label learning) framework and corresponding MIMLBOOST and MIMLSVM algorithms for efficient learning of individual object labels in complex high level concepts, e.g., like the office space. The goal is to learn $f : 2^x \rightarrow 2^y$ from dataset $\{(X_1, Y_1), (X_2, Y_2), \ldots, (X_m, Y_m)$, where $X_i \subseteq X$ represents a set of instances $\{x_{i1}, x_{i2}, \ldots x_{i,ni,}\}$, $x_{ij} \in X$ $(j = 1, 2, \ldots, n_i)$, and $Y_i \subseteq Y$ represents a set of instances $\{y_{i1}, y_{i2}, \ldots y_{i,li,}\}$, $y_{ik} \in Y$ $(k = 1, 2, \ldots, l_i)$, where n_i is the number of instances in X_i and l_i is the number of labels in Y_i [85]. MIMLBOOST uses category-wise decomposition into traditional single instance & single label supervised learning, whereas MIMLSVN utilizes cluster-based feature transformation. So, instead of trying to learn the idea of complex entities (e.g., *office space)*, [85] took the alternate route and learned the lower level individual objects and inferred the higher level concepts.

7.15. Adversarial Training

Machine learning training and deployment used to be done in isolated computers, but now they are increasing being done in a highly interconnected commercial production environment. Take a face recognition system where a network could be trained on a fleet of servers with a training dataset imported from an external data source, and the trained model could be deployed on another server which accepts APIs calls with real time inputs (e.g., images of people entering a building) and responds with matches. The interconnected architecture exposes the machine learning to a wide attack surface. The real-time input or training dataset can be manipulated by an adversary to compromise the output (image match by the network) or the entire model respectively.

Adversarial machine learning is a relatively new field of research that takes into out these new threats to machine learning. According to **[86]** adversaries (e.g., email spammer) can exploit the lack of stationary data distribution and manipulate the input (e.g., an actual spam email) as a normal email. **[86]** demonstrates these and other vulnerabilities and discusses how application domain, features and data distribution can be used to reduce the risk and impact of such adversarial attacks.

7.16. Gaussian Mixture Model

Gaussian mixture model (GMM) is a statistical probabilistic model used to represent multiple normal gaussian distributions within a larger distribution using an EM (estimation maximization) algorithm in an unsupervised setting. E.g., a GMM could be used to represent the height distribution for a large population group with two gaussian distributions, for male and female sub-groups. Figure 26 below demonstrates a GMM with three gaussian distributions within itself.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2019.2912200, IEEE Access

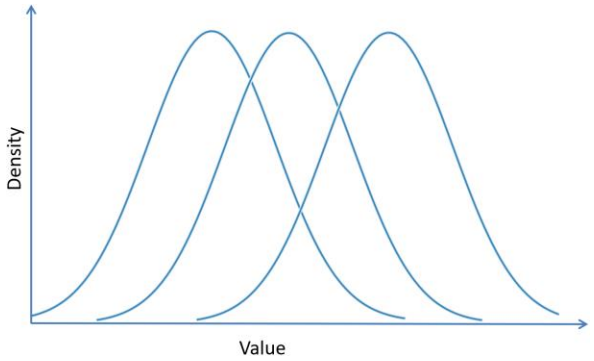

Figure 26. GMM example with three components

GMM has been used primarily in speech recognition and tracking objects in video sequences. GMM are very effective in extracting speech features and modeling the probability density function to a desired level of accuracy as long as we have sufficient components, and the estimation maximization makes it easy to fit the model **[87]**. The probability density function for the GMM is given by the following **[87]**:

$$p(x) = \sum_{m=1}^{M} c_m \mathrm{N}(x; \mu_m, \Sigma_m), \qquad (c_m > 0) \qquad (43)$$

Where M is the number of number of gaussian components, c_m is the weight of the M-th gaussian, and $(x; \mu_m, \Sigma_m)$ represents the random variable x, which following the mean vector μ_m.

7.17. Siamese Networks

The purpose of siamese network is to determine the degree of similarity between two images. As shown in figure 27 below, siamese network consists of two identical CNN networks with identical weights and parameters. The two images to be compared are passed separately through the two twin CNNs and the respective vector representations outputs are evaluated using contrastive divergence loss function. The function is defined as following [88]:

$$L\left(W, Y, \overrightarrow{X_1}, \overrightarrow{X_1}\right) = (1-Y)\frac{1}{2}(D_w)^2 + (Y)\frac{1}{2}(max(0, m - D_w))^2 \qquad (44)$$

D_w represents the Euclidean distance between the two output vectors as shown in figure 27. The output of the contrastive divergence loss function, Y is either 1 (indicates images are not the same) or 0 (indicates images are the same). m represents a margin value greater than 0. The idea of siamese networks has been extended to come up with triplet networks, which includes three identical networks and is used to assess the similarity of a given image with two other images.

Since the softmax layer outputs must match the number of classes, a standard CNN becomes impractical for problems that have large number of classes. This issue doesn't apply to siamese network as the number of outputs of the softmax in the twin networks doesn't have the requirement to match the number of classes[89]. This ability to scale to many more classes for classification extends the use of siamese networks beyond what a traditional CNN is used for. Siamese network can be used for handwritten check recognition, signature verification, text similarity, etc.

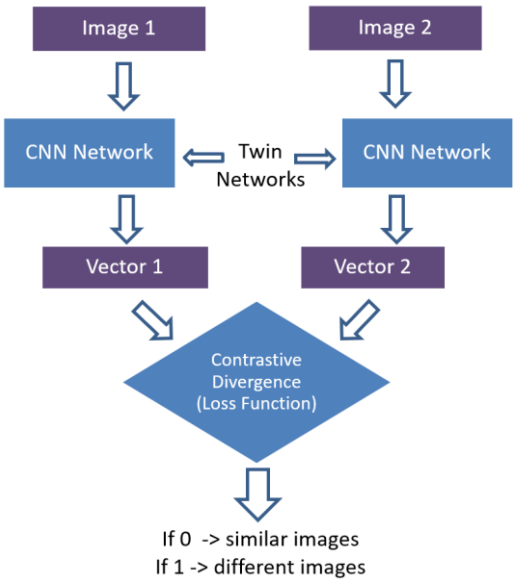

Figure 27. Siamese network

7.18. Variational Autoencoders

As the name suggests, variational autoencoder (VAE), are a type of autoencoder and consists of encoder and decoder parts as shown in figure 28. It falls under the generative model class of neural networks and are used in unsupervised learning. VAEs learn a low dimensional representation (latent variable) that model the original high dimensional dataset into a gaussian distribution. Kullback–Leibler (KL) divergence method is a good way to compare distributions. Therefore, the loss function in VAE is a combination of cross entropy (or mean squared error) to minimize reconstruction error and KL divergence to make the compressed latent variable follow a gaussian distribution. We then sample from the probability distribution to generate new dataset samples that are representative of the original dataset. It has found various applications including generating images in video games to de-noising pictures.

Figure 28. Variational Autoencoder

In figure 28, x is the input and z is the encoded output (latent variable). $P(x)$ represents the distribution associated with x. $P(z)$ represents the distribution associated with z. The goal is to infer $P(z)$ based on $P(z|x)$ that follows a

certain distribution. The mathematical derivation for VAEs were originally proposed in [90]. Suppose we wanted to infer $P(z|x)$ based on some $Q(z|x)$, then we can try to minimize the KL divergence between the two:

$$D_{KL}[Q(z|x)||P(z|x)] = \sum_{z} \mathrm{Q(z|x)} \log[\frac{\mathrm{Q(z|x)}}{\mathrm{P(z|x)}}] \quad (45)$$

$$= \mathrm{E}\,[\log[\frac{\mathrm{Q(Z|X)}}{\mathrm{P(Z|X)}}]] \quad (46)$$

$$= \mathrm{E}\ \log[\,\mathrm{Q(z|x)} - log\mathrm{P(z|x)}] \quad (47)$$

Where D_{KL} is the Kullback–Leibler (KL) divergence and E represents expectation.

Using Baye's rule:

$$P(z|x) = \frac{P(x|z)P(z)}{P(x)} \quad (48)$$

$$D_{KL}[Q(z|x)||P(z|x)] = E\,[\log Q\,(z|x) - Log\frac{P(x|z)P(z)}{P(x)}] \quad (49)$$

$$= E\,[\log Q\,(z|x) - \log P\,(x|z) - \log P\,(z)] + \log P\,(x) \quad (50)$$

To allow us to easily sample $P(z)$ and generate new data, we set $P(z)$ to normal distribution, i.e., $N(0,1)$. If $Q(z|x)$ is represented as gaussian with parameters $\mu(\mathrm{x})$ and $\sum(x)$, then the KL divergence between $Q(z|x)$ and $P(z)$ can be derived in closed form as:

$$D_{KL}\big[N\big(\mu(\mathrm{x}), \Sigma(\mathrm{x})\big)\,|\quad |N(0,1)\big] = (1/2)\sum_{k}(\exp\big(\Sigma(\mathrm{x})\big) + \mu^2(x) - 1 - \Sigma(\mathrm{x})) \quad (51)$$

7.19. Deep Reinforcement Learning

The primary idea about reinforcement learning is about making an agent learn from the environment with the help of random experimentation (exploration) and defined reward (exploitation). It consists of finite number of states (s_i, representing agent and environment), actions (a_i) by the agent, probability (P_a) of moving from one state to another based on action a_i, and reward $R_a(s_i, s_{i+1})$ associated with moving to the next state with action a. The goal is to balance and maximize the current reward (R) and future reward ($\gamma.\max[Q(s', a')]$) by predicting the best action as defined by this function $Q(s, a)$. γ in the equation represent a fixed discount factor. $Q(s, a)$ is represented as the summation of current reward (R) and future reward ($\gamma.\max[Q(s', a')]$) as shown below.

$$Q(s, a) = R + \gamma.\max[Q(s', a')] \quad (52)$$

Reinforcement learning is specifically suited for problems that consists of both short-term and long-term rewards, e.g., games like chess, go, etc. AlphaGo, Google's program that beat the human Go champion also uses reinforcement learning[91]. When we combine deep network architecture with reinforcement learning, we get deep reinforcement learning (DRL), which can extend the use of reinforcement to even more complex games and areas such as robotics, smart grids, healthcare, finance etc. [92]. With DRL, problems that were intractable with reinforcement learning can now be solved with higher number of hidden layers of deep networks and reinforcement learning based Q-learning algorithm that maximizes the reward for actions taken by the agent [13].

7.20. Generative Adversarial Network (GAN)

GANs consists of generative and discriminative neural networks. The generative network generates completely new (fake) data based on input data (unsupervised learning) and the discriminative network attempts to distinguish whether the data is real (from training set) or generated. The generative network is trained to increase the probability of deceiving the discriminative network, i.e., to make the generated data indistinguishable from the original. GANs were proposed by Goodfellow et al., [93] in 2014. It has been very popular as it has many applications both good and bad. E.g., [94] were able to successfully synthesize realistic images from text.

7.21. Multi-approach method for enhancing deep learning

Deep learning can be optimized at different areas. We discussed training algorithm enhancements, parallel processing, parameter optimizations and various architectures. All these areas can be simultaneously implemented in a framework to get the best results for specific problems. The training algorithms can be finetuned at different levels by incorporating heuristics, e.g., for hyperparameter optimization. The time to train a deep learning network model is a major factor to gauge the performance of an algorithm or network. Instead of training the network with all the data set, we can pre-select a smaller but representative data set from the full training distribution set using instance selection methods **[95]** or Monte Carlo sampling **[48]**. An effective sampling method can result in preventing overfitting, improving accuracy and speeding up of the learning process without compromising on the quality of the training dataset. Albelwi and Mahmood **[96]** designed a framework that combined dataset reduction, deconvolution network, correlation coefficient and an updated objective function. Nelder-Mead method was used in optimizing the parameters of the objective function and the results were comparable to latest known results on the MNIST dataset **[96]**. Thus, combining optimizations at multiple levels and

using multiple methods is a promising field of research and can lead to further advancement in machine learning.

8. Conclusion

In this tutorial, we provided a thorough overview of the neural networks and deep neural networks. We took a deeper dive into the well-known training algorithms and architectures. We highlighted their shortcomings, e.g., getting stuck in the local minima, overfitting and training time for large problem sets. We examined several state-of-the-art ways to overcome these challenges with different optimization methods. We investigated adaptive learning rates and hyperparameter optimization as effective methods to improve the accuracy of the network. We surveyed and reviewed several recent papers, studied them and presented their implementations and improvements to the training process. We also included tables to summarize the content in a concise manner. The tables provide a full view on how different aspects of deep learning are correlated.

Deep Learning is still in its nascent stage. There is tremendous opportunity for exploitation of current algorithms/architectures and further exploration of optimization methods to solve more complex problems. Training is currently constrained by overfitting, training time and is highly susceptible to getting stuck in local minima. If we can continue to overcome these challenges, deep learning networks will accelerate breakthroughs across all applications of machine learning and artificial intelligence.

Conflicts of Interest: The authors declare no conflict of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and/or in the decision to publish the results.

ORCID:

Ajay Shrestha: http://orcid.org/0000-0001-5595-5953

References

1. Rosenblatt, F., *The perceptron: a probabilistic model for information storage and organization in the brain.* Psychol Rev, 1958. **65**(6): p. 386-408.
2. Minsky, M. and S. Papert, *Perceptrons; an introduction to computational geometry*. 1969, Cambridge, Mass.,: MIT Press. 258 p.
3. Cybenko, G., *Approximation by superpositions of a sigmoidal function.* Mathematics of Control, Signals and Systems, 1989. **2**(4): p. 303-314.
4. Hornik, K., *Approximation capabilities of multilayer feedforward networks.* Neural Networks, 1991. **4**(2): p. 251-257.
5. Werbos, P.J., *Beyond regression : new tools for prediction and analysis in the behavioral sciences*. 1975.
6. LeCun, Y., Y. Bengio, and G. Hinton, *Deep learning.* Nature, 2015. **521**(7553): p. 436-44.
7. Jordan, M.I. and T.M. Mitchell, *Machine learning: Trends, perspectives, and prospects.* Science, 2015. **349**(6245): p. 255-60.
8. Ng, A., *Machine Learning Yearning: Technical Strategy for AI Engineers In the Era of Deep Learning*. 2019: deeplearning.ai.
9. Metz, C., *Turing Award Won by 3 Pioneers in Artificial Intelligence*, in *New York Times*. 3/27/2019, The New York Times Company. p. B3.
10. Nagpal, K., et al., *Development and Validation of a Deep Learning Algorithm for Improving Gleason Scoring of Prostate Cancer.* CoRR, 2018. **abs/1811.06497**.
11. Nevo, S., et al., *ML for Flood Forecasting at Scale.* CoRR, 2019. **abs/1901.09583**.
12. Esteva, A., et al., *Dermatologist-level classification of skin cancer with deep neural networks.* Nature, 2017. **542**: p. 115.
13. Arulkumaran, K., et al., *Deep Reinforcement Learning: A Brief Survey.* IEEE Signal Processing Magazine, 2017. **34**(6): p. 26-38.
14. Gheisari, M., G. Wang, and M.Z.A. Bhuiyan. *A Survey on Deep Learning in Big Data*. in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*. 2017.
15. Pouyanfar, S., et al., *A Survey on Deep Learning: Algorithms, Techniques, and Applications.* ACM Comput. Surv., 2018. **51**(5): p. 1-36.
16. Vargas, R., A. Mosavi, and R. Ruiz, *Deep learning: A review*. 2017.
17. Buhmann, M.D. and M.D. Buhmann, *Radial Basis Functions*. 2003: Cambridge University Press. 270.
18. Akinduko, A.A., E.M. Mirkes, and A.N. Gorban, *SOM: Stochastic initialization versus principal components.* Information Sciences, 2016. **364-365**: p. 213-221.
19. Chen, K., *Deep and Modular Neural Networks*, in *Springer Handbook of Computational Intelligence*, J. Kacprzyk and W. Pedrycz, Editors. 2015, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 473-494.
20. Ng, A.Y. and M.I. Jordan, *On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes*, in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. 2001, MIT Press: Vancouver, British Columbia, Canada. p. 841-848.

21. Bishop, C.M. and J. Lasserre. *Generative or Discriminative ? Getting the Best of Both Worlds*. 2007.
22. Zhou, T., et al., *Unsupervised Learning of Depth and Ego-Motion from Video.* CoRR, 2017. **abs/1704.07813**.
23. Chen, X.W. and X. Lin, *Big Data Deep Learning: Challenges and Perspectives.* IEEE Access, 2014. **2**: p. 514-525.
24. LeCun, Y., K. Kavukcuoglu, and C. Farabet. *Convolutional networks and applications in vision*. in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. 2010.
25. Gousios, G., et al., *Lean GHTorrent: GitHub data on demand*, in *Proceedings of the 11th Working Conference on Mining Software Repositories*. 2014, ACM: Hyderabad, India. p. 384-387.
26. AI-Index. *Top deep learning Github repositories*. AI Index 2019; Available from: https://github.com/mbadry1/Top-Deep-Learning.
27. Fern, M., et al., *Do we need hundreds of classifiers to solve real world classification problems?* J. Mach. Learn. Res., 2014. **15**(1): p. 3133-3181.
28. Lecun, Y., et al., *Gradient-based learning applied to document recognition.* Proceedings of the IEEE, 1998: p. 2278--2324.
29. LeCun, Y. and Y. Bengio, *Convolutional networks for images, speech, and time series*, in *The handbook of brain theory and neural networks*, A.A. Michael, Editor. 1998, MIT Press. p. 255-258.
30. Taylor, G.W., et al. *Convolutional Learning of Spatio-temporal Features*. in *Computer Vision – ECCV 2010*. 2010. Berlin, Heidelberg: Springer Berlin Heidelberg.
31. Ng, A. *Convolutional Neural Network*. Unsupervised Feature Learning and Deep Learning (UFLDL) Tutorial 2018 [cited 2018 7/21/2018]; Available from: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/.
32. Schuler, C.J., et al. *A Machine Learning Approach for Non-blind Image Deconvolution*. in *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013.
33. Radford, A., L. Metz, and S. Chintala, *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.* CoRR, 2015. **abs/1511.06434**.
34. Jolliffe, I.T., *Principal component analysis*. 2nd ed. Springer series in statistics. 2002, New York: Springer. xxix, 487 p.
35. Noda, K., et al. *Multimodal integration learning of object manipulation behaviors using deep neural networks*. in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013.
36. Hinton, G.E. and R.R. Salakhutdinov, *Reducing the Dimensionality of Data with Neural Networks.* Science, 2006. **313**(5786): p. 504.
37. Wang, M., et al., *Deep Learning-Based Model Reduction for Distributed Parameter Systems.* IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2016. **46**(12): p. 1664-1674.
38. Ng, A. *Autoencoders*. Unsupervised Feature Learning and Deep Learning (UFLDL) Tutorial 2018 [cited 2018 7/21/2018]; Available from: http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders.
39. Teh, Y.W. and G.E. Hinton, *Rate-coded Restricted Boltzmann Machines for Face Recognition.* 2001: p. 908--914.
40. Hinton, G.E., *A Practical Guide to Training Restricted Boltzmann Machines*, in *Neural Networks: Tricks of the Trade: Second Edition*, G. Montavon, G.B. Orr, and K.-R. Müller, Editors. 2012, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 599-619.
41. Hochreiter, S. and J. Schmidhuber, *Long Short-term Memory*. Vol. 9. 1997. 1735-80.
42. Metz, C., *Apple is bringing the AI Revolution to your Phone*, in *Wired*. 2016.
43. Gers, F.A., J. Schmidhuber, and F.A. Cummins, *Learning to Forget: Continual Prediction with LSTM.* Neural Computation, 2000. **12**(10): p. 2451-2471.
44. Chung, J., et al., *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.* eprint arXiv:1412.3555, 2014: p. arXiv:1412.3555.
45. Cho, K., et al., *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.* eprint arXiv:1406.1078, 2014: p. arXiv:1406.1078.
46. Naul, B., et al., *A recurrent neural network for classification of unevenly sampled variable stars.* Nature Astronomy, 2018. **2**(2): p. 151-155.
47. Najafabadi, M.M., et al., *Deep learning applications and challenges in big data analytics.* Journal of Big Data, 2015. **2**(1): p. 1.
48. Goodfellow, I., Y. Bengio, and A. Courville, *Deep learning*. Adaptive computation and machine learning. 2016, Cambridge, Massachusetts: The MIT Press. xxii, 775 pages.
49. Gavin, H.P., *The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems.* 2016.
50. Xavier, G. and B. Yoshua, *Understanding the difficulty of training deep feedforward neural networks*, in *In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics* 2010, PMLR. p. 249-256.
51. Martens, J., *Deep learning via Hessian-free optimization*, in *Proceedings of the 27th International Conference on International*

Conference on Machine Learning. 2010, Omnipress: Haifa, Israel. p. 735-742.

52. Escalante, H.J., M. Montes, and L.E. Sucar, *Particle Swarm Model Selection.* J. Mach. Learn. Res., 2009. **10**: p. 405-440.
53. Shrestha, A. and A. Mahmood, *Improving Genetic Algorithm with Fine-Tuned Crossover and Scaled Architecture.* Journal of Mathematics, 2016. **2016**: p. 10.
54. Sastry, K., D. Goldberg, and G. Kendall, *Genetic Algorithms.* 2005.
55. Goldberg, D.E., *The design of innovation: Lessons from and for competent genetic algorithms* 2013: Springer, Boston, MA.
56. Miikkulainen, R., et al., *Evolving Deep Neural Networks.* CoRR, 2017. **abs/1703.00548**.
57. Duchi, J., E. Hazan, and Y. Singer, *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization.* J. Mach. Learn. Res., 2011. **12**: p. 2121-2159.
58. Kingma, D.P. and J. Ba, *Adam: A Method for Stochastic Optimization.* CoRR, 2014. **abs/1412.6980**.
59. Ioffe, S. and C. Szegedy, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* CoRR, 2015. **abs/1502.03167**.
60. Srivastava, N., et al., *Dropout: a simple way to prevent neural networks from overfitting.* J. Mach. Learn. Res., 2014. **15**(1): p. 1929-1958.
61. Services, A.W. *Amazon EC2 P2 & P3 Instances*. Amazon EC2 Instance Types 2018 [cited 2018 7/21/2018]; Available from: https://aws.amazon.com/ec2/instance-types/p2/ & https://aws.amazon.com/ec2/instance-types/p3/
62. He, K., et al. *Deep Residual Learning for Image Recognition*. in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.
63. Simpson, A.J.R., *Uniform Learning in a Deep Neural Network via "Oddball" Stochastic Gradient Descent.* CoRR, 2015. **abs/1510.02442**.
64. Best-Rowden, L., et al., *Unconstrained Face Recognition: Identifying a Person of Interest From a Media Collection.* IEEE Transactions on Information Forensics and Security, 2014. **9**(12): p. 2144-2157.
65. Letsche, T.A. and M.W. Berry, *Large-scale information retrieval with latent semantic indexing.* Information Sciences—Informatics and Computer Science, Intelligent Systems, Applications: An International Journal 1997. **100**(1-4): p. 105-137.
66. Hinton, G.E., *Learning multiple layers of representation.* Trends in Cognitive Sciences, 2007. **11**(10): p. 428-434.
67. Salakhutdinov, R. and G. Hinton, *Deep Boltzmann Machines*, in *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*, D. David van and W. Max, Editors. 2009, PMLR: Proceedings of Machine Learning Research. p. 448--455.
68. Kuo, W., B. Hariharan, and J. Malik, *DeepBox: Learning Objectness with Convolutional Networks.* CoRR, 2015. **abs/1505.02146**.
69. Huang, G.-B., Q.-Y. Zhu, and C.-K. Siew, *Extreme learning machine: Theory and applications.* Neurocomputing, 2006. **70**(1): p. 489-501.
70. Tang, J., C. Deng, and G.B. Huang, *Extreme Learning Machine for Multilayer Perceptron.* IEEE Transactions on Neural Networks and Learning Systems, 2016. **27**(4): p. 809-821.
71. Gong, M., et al., *A Multiobjective Sparse Feature Learning Model for Deep Neural Networks.* IEEE Transactions on Neural Networks and Learning Systems, 2015. **26**(12): p. 3263-3277.
72. Mehrkanoon, S., et al., *Multiclass Semisupervised Learning Based Upon Kernel Spectral Clustering.* IEEE Transactions on Neural Networks and Learning Systems, 2015. **26**(4): p. 720-733.
73. Langone, R., et al., *Kernel Spectral Clustering and applications.* CoRR, 2015. **abs/1505.00477**.
74. Conneau, A., et al., *Very Deep Convolutional Networks for Natural Language Processing.* CoRR, 2016. **abs/1606.01781**.
75. Krpan, N. and D. Jakobovic. *Parallel neural network training with OpenCL*. in *2012 Proceedings of the 35th International Convention MIPRO*. 2012.
76. Dong, W. and M. Zhou, *A Supervised Learning and Control Method to Improve Particle Swarm Optimization Algorithms.* IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2017. **47**(7): p. 1135-1148.
77. Vapnik, V. and R. Izmailov, *Learning using privileged information: similarity control and knowledge transfer.* J. Mach. Learn. Res., 2015. **16**(1): p. 2023-2049.
78. Sampson, J.R., *Adaptation in Natural and Artificial Systems (John H. Holland).* SIAM Review, 1976. **18**(3): p. 529-530.
79. Mohd Razali, N. and J. Geraghty, *Genetic Algorithms Performance with Different Selection Strategy in Solving TSP*. 2010.
80. Larrañaga, P., et al., *Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators.* Artificial Intelligence Review, 1999. **13**(2): p. 129-170.
81. Whitley, D., *A genetic algorithm tutorial.* Statistics and Computing, 1994. **4**(2): p. 65-85.
82. Lin, C.T., M. Prasad, and A. Saxena, *An Improved Polynomial Neural Network Classifier Using Real-Coded Genetic Algorithm.* IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2015. **45**(11): p. 1389-1401.

83. Guo, Y., et al., *The use of next generation sequencing technology to study the effect of radiation therapy on mitochondrial DNA mutation.* Mutation Research/Genetic Toxicology and Environmental Mutagenesis, 2012. **744**(2): p. 154-160.
84. Wu, Y., et al., *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.* CoRR, 2016. **abs/1609.08144**.
85. Zhou, Z.-H., et al., *Multi-instance multi-label learning.* Artificial Intelligence, 2012. **176**(1): p. 2291-2320.
86. Huang, L., et al., *Adversarial machine learning*, in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. 2011, ACM: Chicago, Illinois, USA. p. 43-58.
87. Yu, D. and L. Deng, *Automatic Speech Recognition: A Deep Learning Approach*. 2015: Springer, London.
88. Hadsell, R., S. Chopra, and Y. LeCun. *Dimensionality Reduction by Learning an Invariant Mapping*. in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. 2006.
89. Shrestha, A. and A. Mahmood. *Enhancing Siamese Networks Training with Importance* in *Proceedings of the 11th International Conference on Agents and Artificial Intelligence*. 2019. Prague, Czech Republic,: SciTePress.
90. Kingma, D.P. and M. Welling, *Auto-Encoding Variational Bayes.* ArXiv e-prints, 2013.
91. Silver, D., et al., *Mastering the game of Go with deep neural networks and tree search.* Nature, 2016. **529**: p. 484.
92. François-Lavet, V., et al., *An Introduction to Deep Reinforcement Learning.* CoRR, 2018. **abs/1811.12560**.
93. Goodfellow, I.J., et al. *Generative Adversarial Networks*. arXiv e-prints, 2014.
94. Reed, S., et al. *Generative Adversarial Text to Image Synthesis*. arXiv e-prints, 2016.
95. Brighton, H. and C. Mellish, *Advances in Instance Selection for Instance-Based Learning Algorithms.* Data Mining and Knowledge Discovery, 2002. **6**(2): p. 153-172.
96. Albelwi, S. and A. Mahmood, *A Framework for Designing the Architectures of Deep Convolutional Neural Networks.* Entropy, 2017. **19**(6).