ENHANCED DEEP NETWORK DESIGNS USING MITOCHONDRIAL DNA BASED GENETIC ALGORITHM AND IMPORTANCE SAMPLING

Ajay D. Shrestha

Under the Supervision of Dr. Ausif Mahmood

DISSERTATION

SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

AND ENGINEERING

THE SCHOOL OF ENGINEERING

UNIVERSITY OF BRIDGEPORT

CONNECTICUT

October, 2019

ENHANCED DEEP NETWORK DESIGNS USING MITOCHONDRIAL DNA BASED GENETIC ALGORITHM AND IMPORTANCE SAMPLING

Ajay D. Shrestha

Under the Supervision of Dr. Ausif Mahmood

Approvals

Committee Members

Name	**Signature**	**Date**
Dr. Ausif Mahmood	[illegible]	11-19-2019
Dr. Khaled Elleithy	[illegible]	11/19/19
Dr. Julius Dichter	[illegible]	11.18.2019
Dr. Miad Faezipour	[illegible]	11, 26, 2019
Dr. Syed Rizvi	[illegible]	10-31-2019

Ph.D. Program Coordinator

Dr. Khaled M. Elleithy	[illegible]	11/18/19

Chairman, Computer Science and Engineering Department

Dr. Ausif Mahmood	[illegible]	11-1-2019

Dean, School of Engineering

Dr. Tarek M. Sobh	T. Sobh	11/21/2019

ENHANCED DEEP NETWORK DESIGNS USING MITOCHONDRIAL DNA BASED GENETIC ALGORITHM AND IMPORTANCE SAMPLING

ENHANCED DEEP NETWORK DESIGNS USING MITOCHONDRIAL DNA BASED GENETIC ALGORITHM AND IMPORTANCE SAMPLING

ABSTRACT

Machine learning (ML) is playing an increasingly important role in our lives. It has already made huge impact in areas such as cancer diagnosis, precision medicine, self-driving cars, natural disasters predictions, speech recognition, etc. The painstakingly handcrafted feature extractors used in the traditional learning, classification and pattern recognition systems are not scalable for large-sized datasets or adaptable to different classes of problems or domains. Machine learning resurgence in the form of Deep Learning (DL) in the last decade after multiple AI (artificial intelligence) winters and hype cycles is a result of the convergence of advancements in training algorithms, availability of massive data (big data) and innovation in compute resources (GPUs and cloud). If we want to solve more complex problems with machine learning, we need to optimize all three of these areas, i.e., algorithms, dataset and compute. Our dissertation research work presents the original application of nature-inspired idea of mitochondrial DNA (mtDNA) to improve deep learning network design. Additional fine-tuning is provided with Monte Carlo based method called importance sampling (IS). The primary performance indicators for machine learning are model accuracy, loss and training time. The goal of our dissertation is to provide a framework to address all these areas by optimizing network designs (in the form of hyperparameter optimization) and dataset using enhanced Genetic Algorithm (GA) and importance sampling. Algorithms are by far the most important aspect of machine learning. We demonstrate the application of mitochondrial DNA to complement the standard genetic algorithm for architecture optimization of deep Convolution Neural Network (CNN). We use importance sampling to reduce the dataset variance and sample more often from the instances that add greater value from the training outcome perspective. And finally, we leverage massive parallel and distributed processing of GPUs in the cloud to speed up training. Thus, our multi-approach method for enhancing deep learning combines architecture optimization, dataset optimization and the power of the cloud to drive better model accuracy and reduce training time.

ACKNOWLEDGEMENTS

I owe a debt of gratitude to my family for their love and encouragement. I am grateful to the school of engineering and its faculty for providing a supportive and learning environment. I am indebted to the University of Bridgeport for giving me the opportunity to realize my academic goals and build lasting connections. I am also thankful to all my past teachers and alma maters.

I would like to express my deep gratitude to my advisor Professor Ausif Mahmood. I have benefitted greatly from his teaching and research mentorship. I feel fortunate to be his advisee. I am also very appreciative of the guidance provided by Professor Khaled Elleithy and the committee members.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1: INTRODUCTION

Model accuracy, loss and training time are the key performance indicators of any machine learning framework or algorithm. Accuracy and loss are impacted by multiple factors. Backpropagation, the de facto training method, has its share of shortcomings. They are manifested in the form of getting stuck in local minima, overshooting the potential global minima or overfitting a model to a data set. The explosion of smart devices, sensors and the Internet of Things (IoT) have resulted in an exponential increase in the amount of data. Dataset sample size and dimensions have a highly outsized impact on the problem space and training time. The issue is exacerbated with the addition of any new parameter value to the neural network. These are significant challenges and overcoming them could result in enormous benefits to the field of machine learning. We need relevant data with less noise and optimized deep architecture designs to reduce the loss function and yield better accuracy. Our research tackles these problems by proposing a framework for coming up with an enhanced deep learning architecture with dataset sampling and hyperparameter optimization.

1.1 Research Problem and Contribution

Only weights and biases are learned by gradient descent-based training of Deep Neural Networks (DNN). The other hyperparameters shape the architecture of the network and have a huge influence on the quality of the model but finding optimal values for them is not a trivial solution. The hyperparameter space grows exponentially and they also display non-linearity and interactions. A network with twelve hyperparameters, each with five potential values can have about 250 million unique combinations of hyperparameter sequences. If training on each set takes 6 minutes, exhaustive training on all potential combinations of the hyperparameters to find the optimal values will take almost 3000 years. Expert knowledge or random selection are some alternate options, but they are not scalable or consistently reliable. Metaheuristics such as evolutionary algorithms are a great choice for solving combinatorial optimization problems like hyperparameter optimization. While other researchers have used evolutionary algorithms such as standard implementation of genetic algorithm (GA), we introduce additional nature-inspired enhancements to GA for better exploration of the hyperparameter solution space to optimize the DNN architecture.

The accuracy of machine learning (ML) model is determined to a great extent by its training dataset. Yet the dataset optimization is often not the center of the focus to improve ML models. Datasets used in the training process affect the convergence of the training process and accuracy of the models. We complement the training with Monte Carlo based variance reduction method called *importance sampling*. We demonstrate that these fine-tunings result in tangible improvements in the network accuracy on MNIST dataset that outperforms standard use of GA.

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. [1]. 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[2]. 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 [3], discovery of new drugs, prediction of natural disasters [4]. E.g., [5] reported that a 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 [3] was able to beat the average accuracy of US board certified general pathologists in grading prostate cancer by 70% to 61%.

The literature survey covers the vast subject of deep learning and presents a holistic survey of dispersed information under one section. Other review papers [6-9] focus on specific areas and implementations without encompassing the full scope of the field. The research chapter delves into the specific research topic. It presents novel work by collating the works of leading authors from the wide scope and breadth the deep learning.. The review and research sections describe the different types of deep learning network architectures, deep learning algorithms, their shortcomings, optimization methods and the latest implementations and applications.

1.3 Contribution of Research

Our work looks at the ML problem from multiple perspective with the end goal of improving the overall network accuracy and reducing training time. It provides a novel approach of combining enhanced genetic algorithm for hyperparameter and architecture optimization, Monte Carlo based importance sampling method, and parallel and distributed gpu processing in the cloud to speed up the training.

Exhaustive ways to solve combinatorial optimization problem is not tractable. Traveling salesman problem (TSP) is such a problem, where we need to find the shortest path through every city from the original city and back, provided the distances between every pair of cities are given. TSP is an NP-hard problem and its time complexity rises factorially with each new city. Such problems are well suited for metaheuristics like genetic algorithm. Genetic algorithm like other metaheuristics is inspired by the natural phenomenon of natural selection, genetic crossover and mutation. It provides a means to navigate the hyperparameter solution space in an intuitive way. It uses both exploration and exploitation of the space to overcome the local minima and get close to the near-optimal solution. We use the concept of mitochondrial DNA (mtDNA) to further enhance the genetic algorithm.

Not all instances from the training dataset are equally significant from the perspective of their impact to training. Importance sampling can help us focus more on the important ones and stop wasting CPU or GPU cycles on using the un-important instances at every epoch or mini-batch. E.g., we achieve high accuracy with just one epoch on the MNIST dataset using a relatively small network. It would make more sense to sample more from the important instances in subsequent epochs.

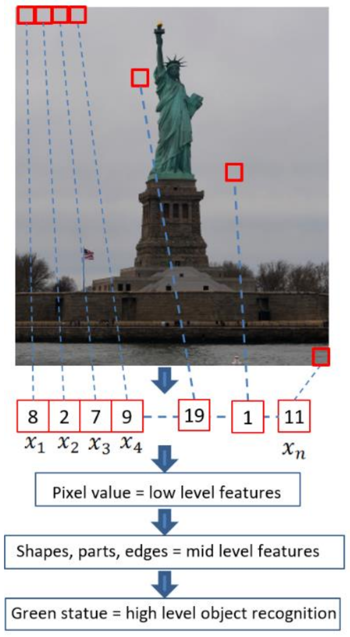
We demonstrate empirically that our approach can achieve improvement in training and testing errors on MNIST dataset compared to training when these fine-tunings are not used. These approaches return better results independently when the stand on their own and are complimentary to each other when combined.

CHAPTER 2: LITERATURE SURVEY

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. Figure 1 is an example of image recognition. It 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.


8
2
7
9
19
1
11
x_1 x_2 x_3 x_4
x_n
Pixel value = low level features
Shapes, parts, edges = mid level features
Green statue = high level object recognition
Figure 1. Image recognition by a CNN

The implementation of neural networks consists of the following steps:

1. Acquire training and testing data set
2. Prepare data for training
3. Train the neural network model
4. Make prediction with test data

Our work focuses on optimizing the acquisition of training dataset and the training steps, so we can make better prediction from the trained model.

The paper is organized in the following sections:

1. Introduction
2. Literature Survey
3. Research & Related Work
4. Implementation
5. Results
6. Conclusion

In 1957, Frank Rosenblatt created the perceptron, the first prototype of what we now know as a neural network [10]. 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[11].

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 [12]. It was mathematically proven as well [13], which further questioned the validity of DNN. While a single hidden layer could be used to learn, it was not efficient and didn't provide the capability presented 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 [14] but it wasn’t until mid-1980s [15] 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 [16].

2.1 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.

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

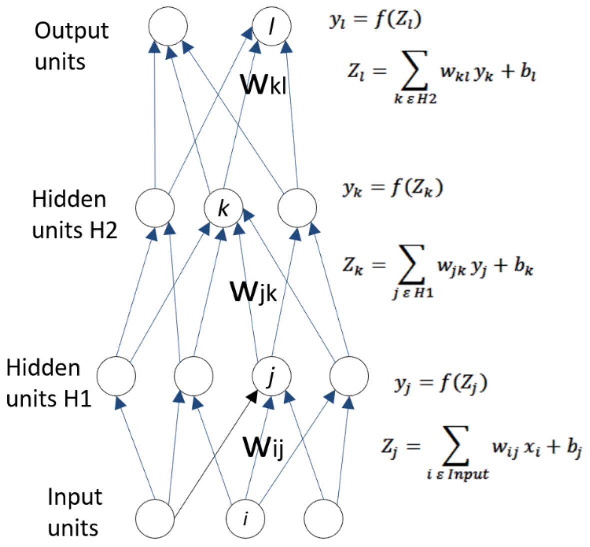

Output units
Hidden units H2
Hidden units H1
Input units
l
k
j
i
Wkl
Wjk
Wij
$y_l = f(Z_l)$
$Z_l = \sum_{k \varepsilon H2} w_{kl} y_k + b_l$
$y_k = f(Z_k)$
$Z_k = \sum_{j \varepsilon H1} w_{jk} y_j + b_k$
$y_j = f(Z_j)$
$Z_j = \sum_{i \varepsilon Input} w_{ij} x_i + b_j$

Figure 2a. Feedforward neural network [15]

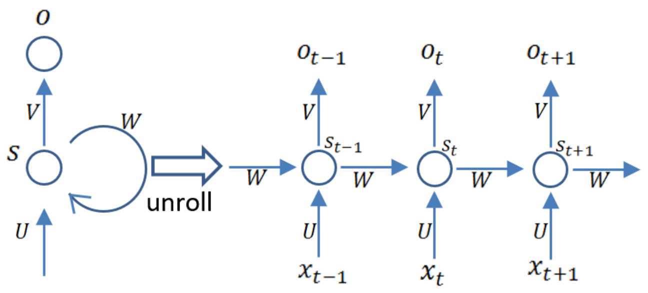


Figure 2b. The unrolling of RNN in time [15]

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].

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

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 of an 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 enhanced 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.

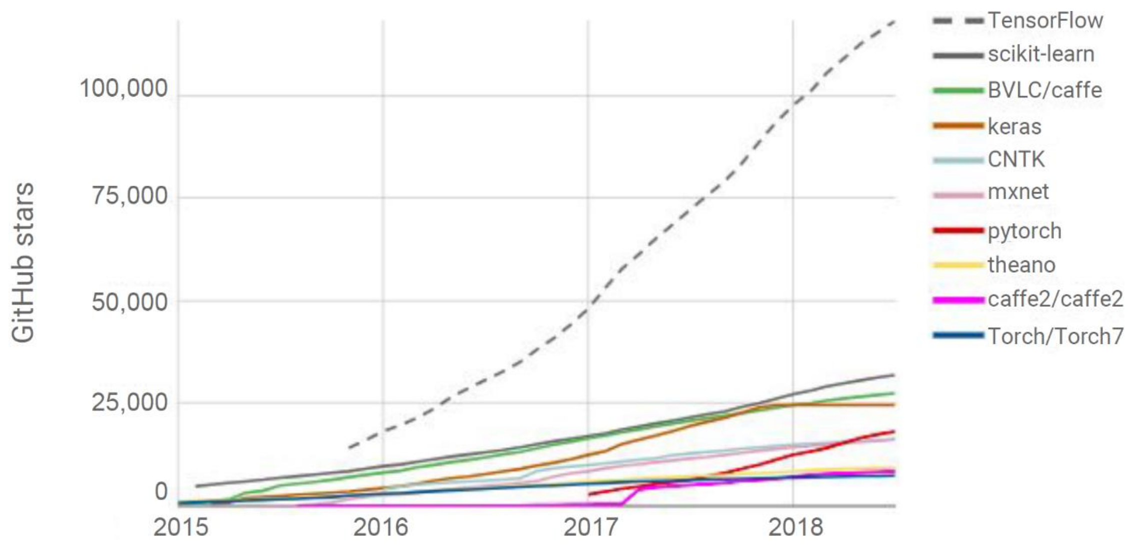


Figure 3. Github stars by Deep Learning Library [26]

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

2.2 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 (CNN)
2. Autoencoder
3. Restricted Boltzmann Machine (RBM)
4. Long Short-Term Memory (LSTM)

2.2.1 Convolution Neural Network

CNN is based on the human visual cortex and is the neural network of choice for computer vision (image recognition). It has been applied to areas such has cancer diagnosis, self-driving cars, etc. As shown in Figure 4a, 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 4a 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.

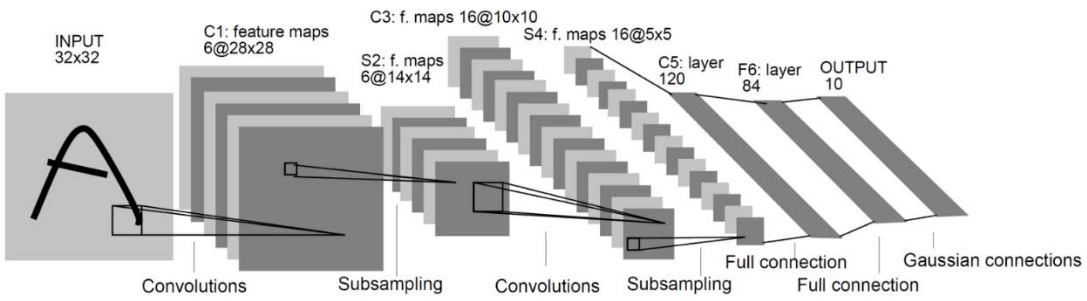


Figure 4a. 7-layer Architecture of CNN for character recognition [28]

CNN's takes a 2D $n\, x\, n$ 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 connections reduces training time and chances of overfitting. As show in Figure 4b, 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.

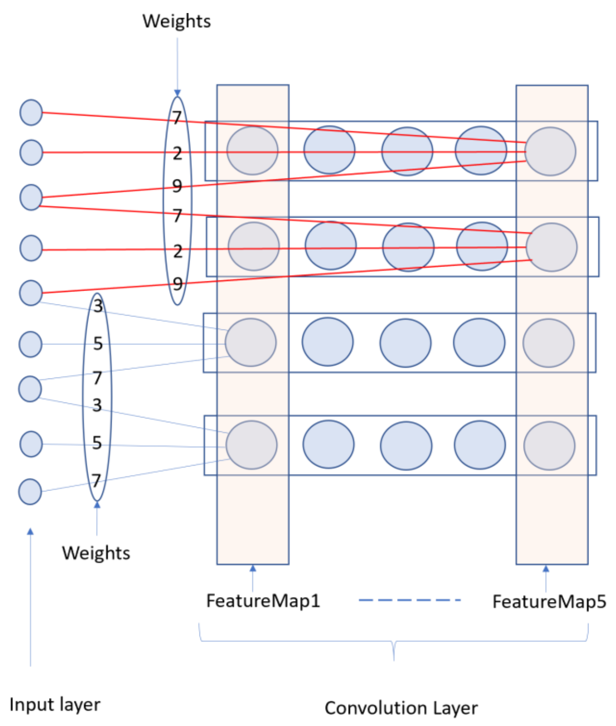


Figure 4b. Single convolution layer of CNN

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]. 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 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.

Algorithm 1: CNN Backpropagation Algorithm Pseudo Code

```
1: $Initialization\ weights\ to\ randomly\ generated\ value\ (small)$
2: $Set\ learning\ rate\ to\ a\ small\ value\ (positive)$
3: $Iteration\ n = 1;\ \boldsymbol{Begin}$
4:  $\boldsymbol{for}\ n <\ max\ iteration\ OR\ Cost\ function\ criteria\ met,\ \boldsymbol{do}$
5:   $\boldsymbol{for}\ image\ x_1\ to\ x_i,\ \boldsymbol{do}$
6:     $a.\ Forward\ propagate\ through\ convolution, pooling\ and\ then\ fully\ connected\ layers$
7:     $b.\ Derive\ Cost\ Fuction\ value\ for\ the\ image$
8:     $c.\ Calculate\ error\ term\ \delta^{(l)}\ with\ respect\ to\ weights\ for\ each\ type\ of\ layers.$
9:       $Note\ that\ the\ error\ gets\ propagated\ from\ layer\ to\ layer\ in\ the\ following\ sequence$
10:      $i.\ fully\ connected\ layer$
11:      $ii.\ pooling\ layer$
12:      $iii.\ convolution\ layer$
13:    $d.\ Calculate\ gradient\ \nabla_{w_k^{(l)}}\ and\ \nabla_{b_k^{(l)}}\ for\ weights\ \nabla_{w_k^{(l)}} and\ bias\ respectively\ for\ each\ layer$
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: CNN developed to run on Nvidia parallel computing platform to support GPUs

2. Inception: Deep CNN developed by Google

3. ResNet: Very deep Residual network developed by Microsoft. It won 1st place in the ILSVRC 2015 competition on ImageNet dataset.

4. VGG: Very deep CNN developed for large scale image recognition

5. DCGAN: Deep convolutional generative adversarial networks proposed by [33]. It is used in unsupervised learning of hierarchy of feature representations in input objects.

2.2.2 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.

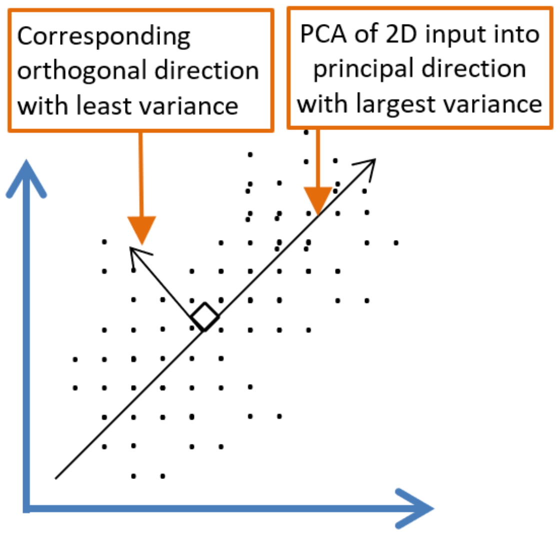

Corresponding orthogonal direction with least variance
PCA of 2D input into principal direction with largest variance

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 fine-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 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 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].

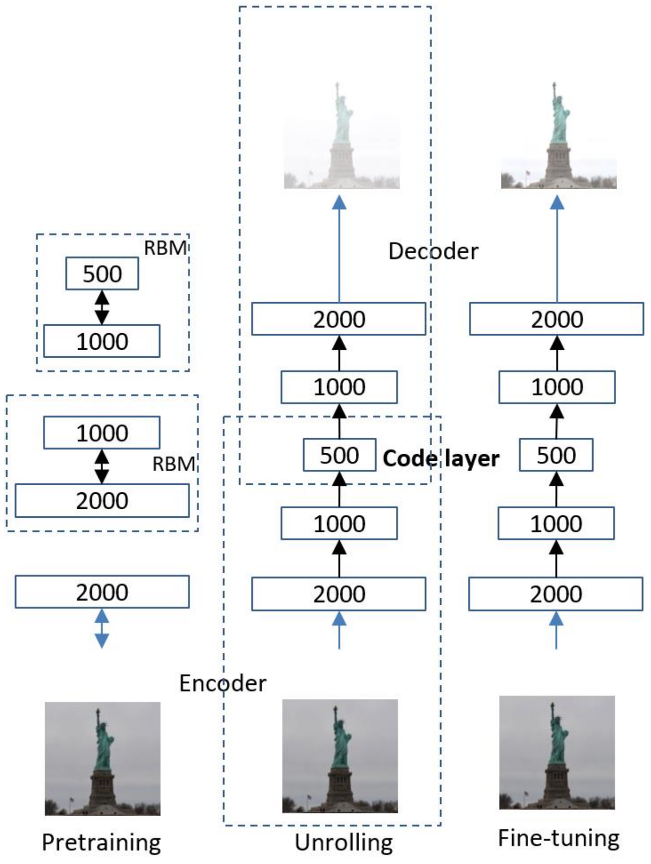


Figure 6. Training stages in Autoencoder [36]

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].

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

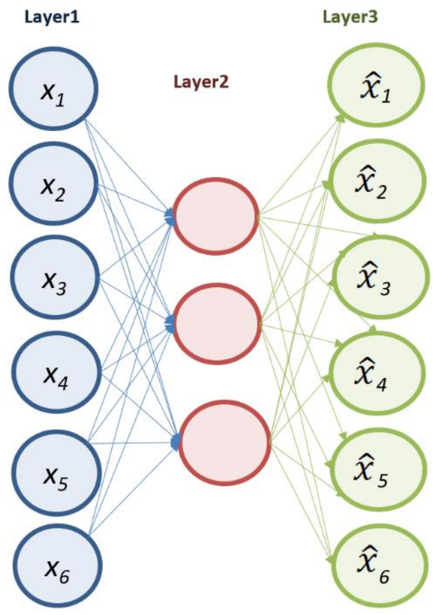


Figure 7. Autoencoder nodes

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.

2.2.3 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]:

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

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].

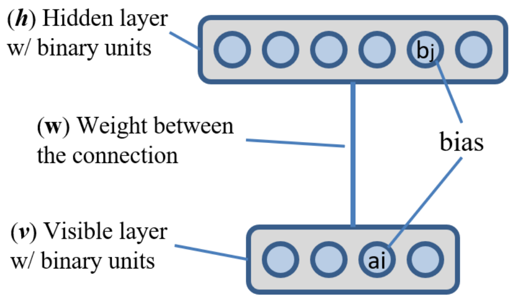


Figure 8. Restricted Boltzmann Machine

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

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.

$$\Delta w_{ij} = \epsilon(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}) \qquad (17)$$

2.2.4 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].

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.

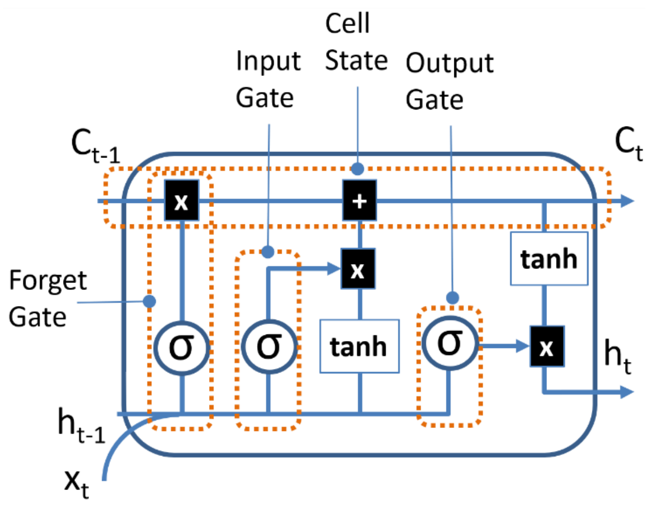


Figure 9. LSTM Block with memory cell and gates

The computations of the gates are described in the equations below[41, 43].

$$f_t = \sigma(W_f x_t + w_f h_{t-1} + b_f) \tag{18}$$

$$i_t = \sigma(W_i x_t + w_i h_{t-1} + b_i) \tag{19}$$

$$o_t = \sigma(W_o x_t + w_o h_{t-1} + b_o) \tag{20}$$

$$c_t = f_t \otimes c_{t-1} + i_t \otimes \sigma_c(W_c x_t + w_c h_{t-1} + b_c) \tag{21}$$

$$h_t = o_t \otimes \sigma_h(c_t) \tag{22}$$

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 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].

2.2.5 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 fine-tuned 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

2.3 Dataset

Access to data is a critical factor for training robust ML models. The explosion of smart devices, sensors, cloud and edge computing have resulted in massive volume, variety and velocity of data. While this has been a tremendous boon to ML, normalizing and sanitizing this data before utilizing them for training is a challenge. The goal of dataset optimization is to reduce the dimensions or the size of the dataset with the intent of improving the accuracy and/or reducing training time, without compromising on the quality. There are several type of dataset optimization methods that are currently in practice.

Dataset optimization including importance sampling has been used to fine-tune the training process and achieve both training speed-up and accuracy improvement. Here are some of the earlier related work.

[47] enhanced variations of stochastic optimization (prox-SMD and prox-SDCA) with importance sampling to reduce the variance resulting in better convergence rate of the training. [48] optimized the training of CNN and RNN with importance sampling by computing an upper bound for the gradient and estimation for the variance reduction.

Dataset and training data in general come with lot of noise that do not contribute to the training process and in some cases hinder training. An effective sampling from the full dataset is a great way to address this concern. [49] showed that the sampling based of word frequency compression and speaker distribution can have a positive impact on training.

2.4 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. [50], 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 [15]. 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

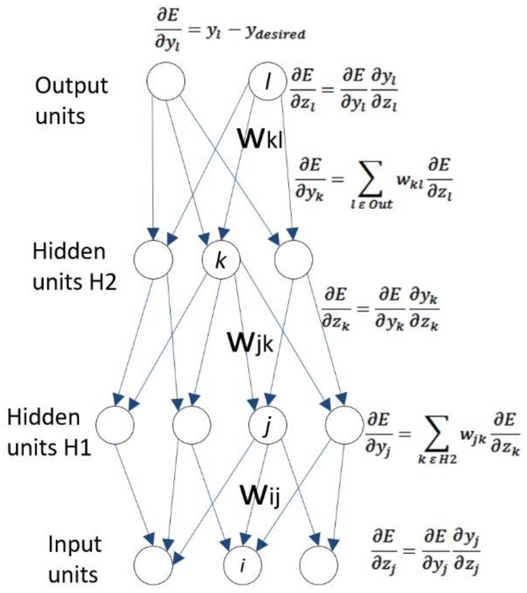
2.4.1 Gradient Descent

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.

$$\mathrm{C} = \frac{1}{2}\left(y_{\text{expected}} - y_{actual}\right)^2 \tag{23}$$

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 [15]. We process through all the samples in the training dataset before applying the updates to the weights. This process is repeated until 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$y_j \partial E/\partial z_k$.



$\frac{\partial E}{\partial y_l} = y_l - y_{desired}$
Output units
l
$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$
Wkl
$\frac{\partial E}{\partial y_k} = \sum_{l \varepsilon Out} w_{kl}\frac{\partial E}{\partial z_l}$
Hidden units H2
k
$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$
Wjk
Hidden units H1
j
$\frac{\partial E}{\partial y_j} = \sum_{k \varepsilon H2} w_{jk}\frac{\partial E}{\partial z_k}$
Wij
Input units
i
$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$

Figure 10. Error calculation in Multilayer Neural Network [15]

2.4.2. 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.

2.4.3. 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 [51]. 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.

$$Velocity\ Update: v \leftarrow \alpha v - \epsilon g \tag{24}$$

$$Actual\ Update: \theta \leftarrow \theta + v \tag{25}$$

2.4.4 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 [52].

If the fitting function is denoted by *ŷ(t;**p**)* and *m* data points denoted by *(t_i,y_i)*, then the squared error can be written as [52]:

$$x^2(\boldsymbol{p}) = \sum_{i=1}^{m} \left[\frac{y(t_i) - \hat{y}(t_i;\mathbf{p})}{\sigma_{yi}}\right]^2 \qquad (26)$$

$$= (y - \hat{y}(p))^T W \left(y - \hat{y}(p)\right) \qquad (27)$$

$$= y^T W y - 2y^T W \hat{y} + \hat{y}^T W \hat{y} \qquad (28)$$

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 [52]:

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

$$= 2(y - \hat{y}(p))^T W \left[\frac{\partial \hat{y}(p)}{\partial \mathbf{p}}\right] \qquad (30)$$

$$= 2(y\ \hat{y})^T W\, \mathbf{J} \qquad (31)$$

$$\mathrm{h_{gd}} = \alpha\, \mathbf{J}^T W\, (y - \hat{y}) \quad (32)$$

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 [52]:

$$[\mathrm{J}^T \mathrm{WJ}]\mathrm{h_{gn}} = \mathrm{J}^T \mathrm{W}(y - \hat{y}) \quad (33)$$

The Levenberg- Marquardt update $[\mathrm{h_{lm}}]$ is generated by combining gradient descent and Gauss-Newton methods resulting in the equation below [52]:

$$[\mathrm{J}^T \mathrm{WJ} + \lambda\, \mathrm{diag}\, (\mathrm{J}^T \mathrm{WJ})]\, \mathrm{h_{lm}} = \mathrm{J}^T \mathrm{W}(y - \hat{y}) \quad (34)$$

2.4.5 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.

2.4.6 Comparison of Deep Learning Algorithms

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

2.5 Shortcomings of Training Algorithms

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

2.5.1 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.

2.5.2 Local Minima

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

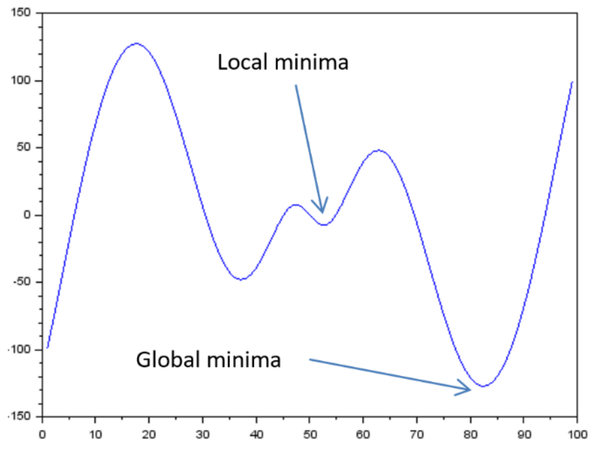


Figure 11. Gradient Descent

2.5.3 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 be misled by this area as the gradient comes to a halt at this point.

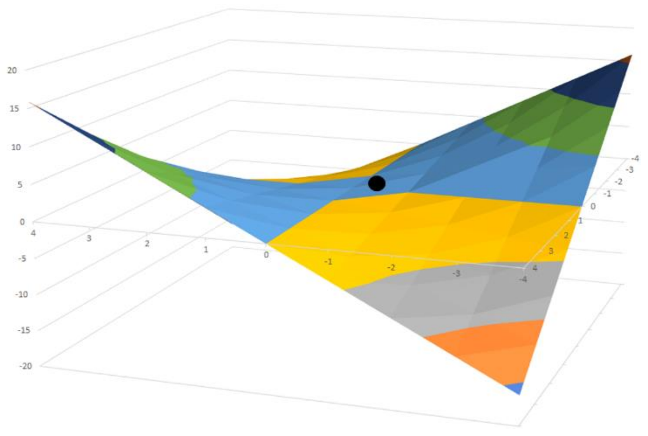


Figure 12. Flat (saddle point marked with black dot) region in a nonconvex function

2.5.4 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.

2.5.5 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 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.

2.5.6 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.

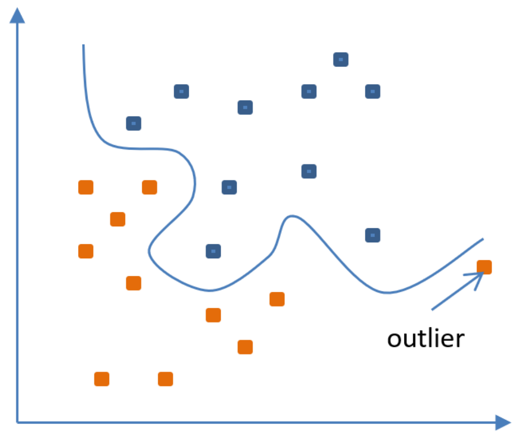


Figure 13. Overfitting in Classification

2.6 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.

2.6.1 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 [53]. Figure 14 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 [53]:

$$F(x) := H(x) - xv \tag{35}$$

Where *F(x)* is the residual mapping and *H(x)* is the desired mapping, and then by recasting the desired mapping at the end [53]. According to MSRA team, it is much easier to optimize the residual mapping.

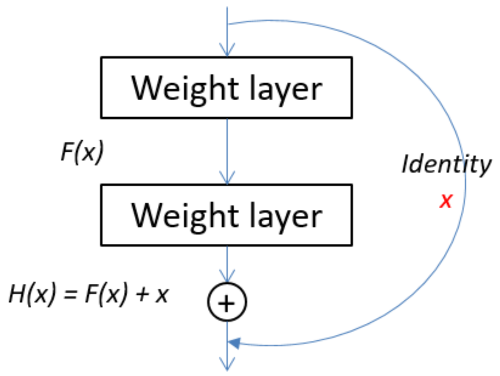


Figure 14. Deep Residual Learning model by MSRA at Microsoft

2.6.2 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 [54] 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 [54] proves his methodology, termed Oddball Stochastic Gradient Descent with a training set of 1000 video frames. Simpson [54] 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.

2.6.3 Deep Belief Network

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

$$\Delta w_{ij}(t+1) = c\Delta w_{ij}(t) + \alpha(\langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{model}) \tag{36}$$

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

$$p(h_j = 1\,|\mathrm{v};\mathrm{W}) = \sigma\left(\sum_{i=1}^{I} w_{ij}v_i + a_j\right) \tag{37}$$

$$p(v_i = 1\,|\mathrm{h};\mathrm{W}) = \sigma\left(\sum_{j=1}^{J} w_{ij}h_j + b_i\right) \tag{38}$$

2.6.4 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. [50] 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. [55] 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 [7, 56]. This involves building models that provide relationships between documents and keywords the contain to make information retrieval more effective.

2.6.5 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 [57]. 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 [57].

In the generative model in Figure 15, 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.

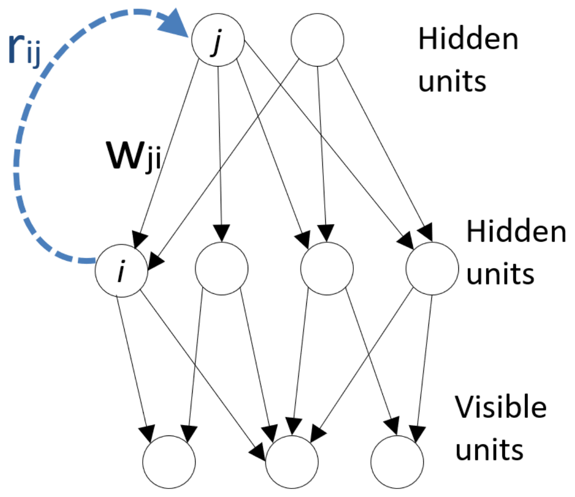


Figure 15. Learning multiple layers of representation

Here is the equation [57] for adjusting the recognition weights r_{ij}.

$$\Delta r_{ij} \propto h_i \left(h_j - \sigma(\sum_i h_i r_{ij}) \right) \tag{39}$$

2.6.6 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 [50]. 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 one layer at a time [58]. 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 16. In [58] 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 17. They [58] received favorable results validating benefits of combining supervised and unsupervised learning methods.

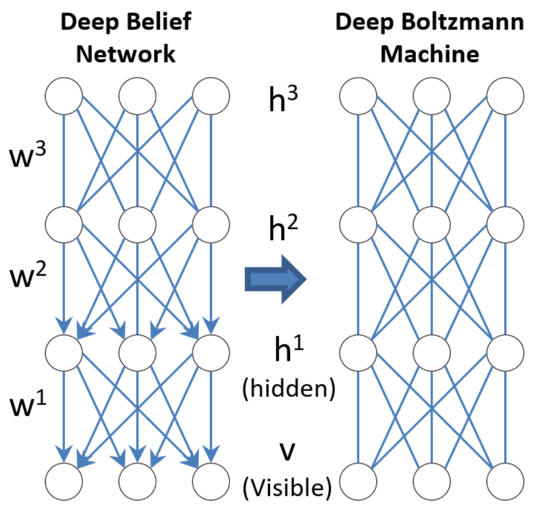


Figure 16. Four-layer DBN & four-layer Deep Boltzmann Machine

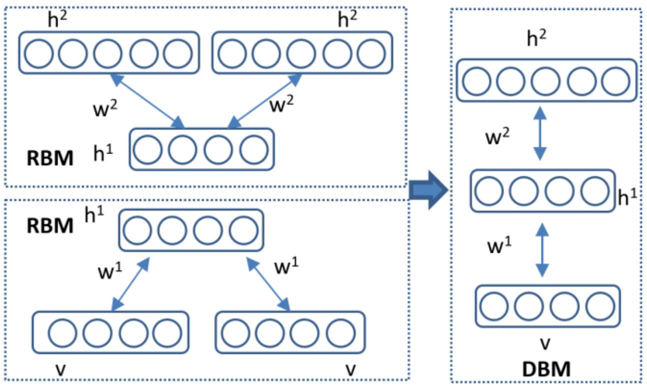


Figure 17. Pretraining of stacked & altered RBM to create a DBM [58]

Here are the equations showing probability distributions over visible and two hidden units in DBM (after unsupervised pre-training) [58].

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

$$p(h_m^2 = 1|\mathrm{h}^1) = \sigma\left(\sum_j W_{jm}^2 h_j^1\right) \quad (41)$$

$$p\left(h_j^1 = 1\middle|\mathrm{v}, \mathrm{h}^2\right) = \sigma\left(\textstyle\sum_i W_{ij}^1 v_i + \sum_m W_{jm}^2 h_m^2\right) \quad (42)$$

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 18. The approximate posterior distribution $q(h|v)$ 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 18 and subsequently, backpropagation is used to fine-tune the network [58].

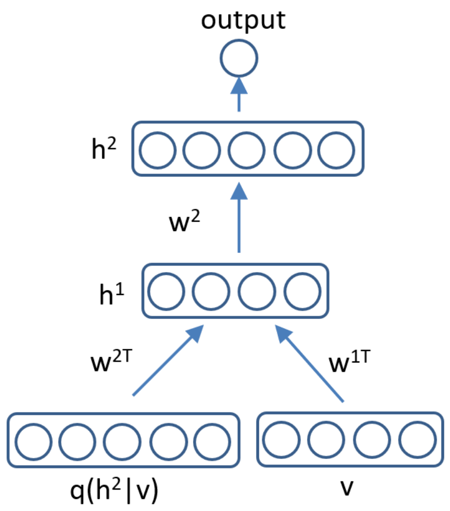


Figure 18. DBM getting initialized as deterministic neural network with supervised fine-tuning [58]

2.6.7 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. [59] proposed a four-layered CNN termed *DeepBox* that outperformed larger networks in speed and accuracy. 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 [60].

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 [61] to take on more complex problems. They combined unsupervised multilayer encoding with the random initialization of the weights and demonstrate faster convergence or lower training time than the state-of-the-art multilayer perceptron training algorithm.

2.6.8 Multi-objective Sparse Feature Learning Model

Moaguo et al. [62] 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 [62] 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 comparing the results with other implementations. Their learning procedure continuously iterates between evolutionary optimization step and the stochastic gradient descent to optimize the reconstruction error [62]. Here are the steps.

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.

$$\sum_{x \in D} L(x, g_{\theta'}(f_{\theta}(\mathrm{x}))) \qquad (43)$$

Figure 19 shows a pareto frontier function that can be used to achieve a compromise between two competing objectives functions.

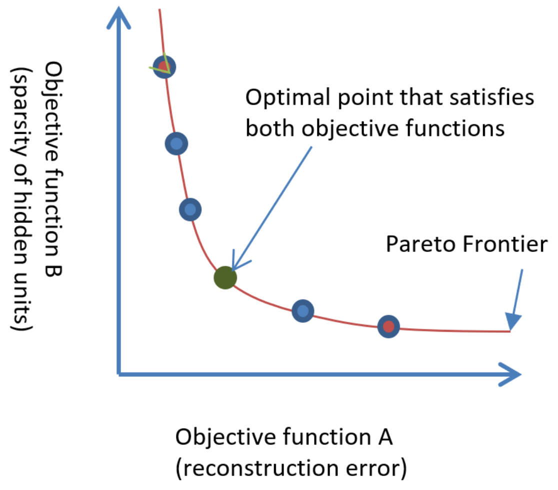


Figure 19. Pareto Frontier

2.6.9 Multiclass Semi-Supervised Learning Based on Kernel Spectral

Mehrkanoon et al. [63] 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 allows labels or membership to be applied to unlabeled data examples. It is achieved in the following way [63]:

- 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.

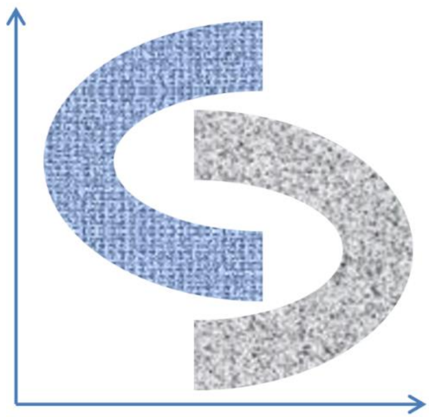


Figure 20. Spectral Clustering Representation

Figure 20 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 [64].

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

2.6.10 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. [65] 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. [53] states that the degradation is not caused by overfitting but because deeper systems are difficult to optimize. [53] addressed this issue with shortcut connections between the convolution blocks to let the gradients to propagate more freely and they, along with [65] were able to validate the benefits of the shortcuts with 10/101/152-layers and 49 layers respectively. The architecture proposed by Conneau et al. [65] consists of series of convolution blocks separated by pooling that halved the resolution followed by k-max pooling and classification at the end.

2.6.11 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. It consists of particles or candidate solutions that move about in a search space to reach a near optimal solution. In their paper [66], 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 [67] 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 [67]. Metaheuristics provide high level guidance inspired by nature and applies them to solve mathematical problems. In a similar way [68] 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.

2.6.12 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 [69]. 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 [70]. There are several ways to perform selections, e.g., elite, roulette, rank, tournament [71]. There are about dozen ways to perform crossovers as reviewed by Larrañaga et al. alone [72]. 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 with 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 [73]. 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. [74] 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 [74].

2.6.13 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. It also has difficulty with rare words. For these reason, large tech companies like Google and Microsoft have improved NMT with their own implementations of NMT, labeled as Google Neural Machine Translation (GNMT) and Skype Translator respectively. GMNT as shown in Figure 21 consists of encoder and decoder LSTM blocks organized in layers, was presented in 2016 in [75]. 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 meanings. 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 [75].

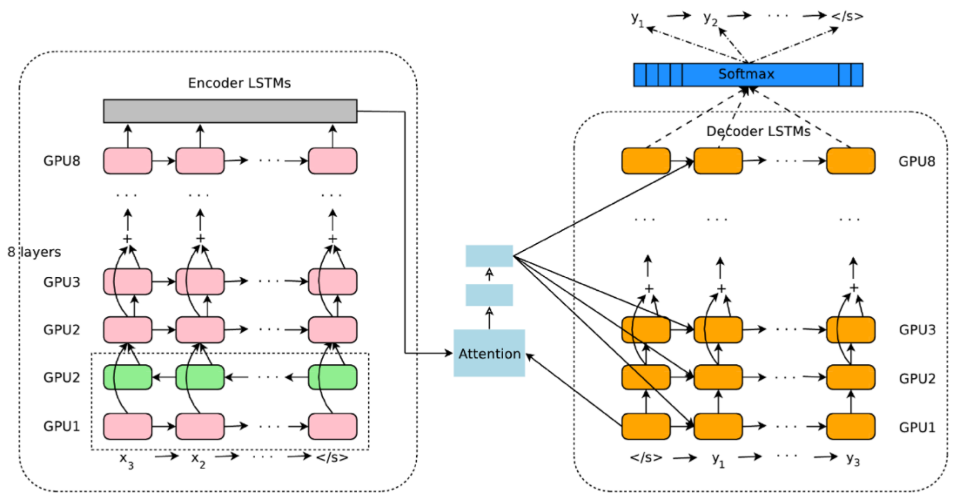


Figure 21. GNMT Architecture [75] with encoder neural network on the left and decoder neural network on the right.

2.6.14 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. [76] 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 $\{(X_1, Y_1), (X_2, Y_2), \dots, (X_m, Y_m),$ where $X_i \subseteq X$ represents a set of instances $\{x_{i1}, x_{i2}, \dots x_{i,ni,}\}$, $x_{ij} \ \in X \ (j = 1, 2, \dots, n_i)$, and $Y_i \subseteq Y$ represents a set of instances $\{y_{i1}, y_{i2}, \dots y_{i,li,}\}$, $y_{ik} \ \in Y \ (k = 1, 2, \dots, l_i)$, where n_i is the number of instances in X_i and l_i is the number of labels in Y_i [76]. 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),* [76] took the alternate route and learned the lower level individual objects and inferred the higher level concepts.

2.6.15 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 account these new threats to machine learning. According to [77] 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. [77] 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.

2.6.16 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 22 below demonstrates a GMM with three gaussian distributions within itself.

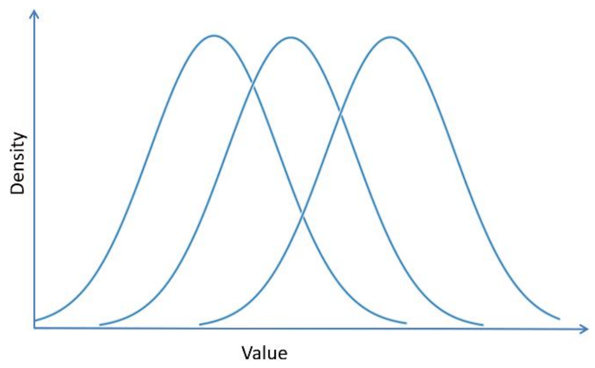


Figure 22. GMM example with three components

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 [78]. The probability density function for the GMM is given by the following [78]:

$$p(x) = \sum_{m=1}^{M} c_m \mathrm{N}\,(x; \mu_m, \Sigma_m), \qquad (c_m > 0) \qquad (44)$$

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.

2.6.17 Siamese Networks

The purpose of siamese network is to determine the degree of similarity between two images. As shown in Figure 23 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 [79]:

$$L\left(W, Y, \overrightarrow{X_1}, \overrightarrow{X_1}\right) = (1 - Y)\frac{1}{2}(D_w)^2 + (Y)\frac{1}{2}(max(0, m - D_w))^2 \tag{45}$$

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 [80]. 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.

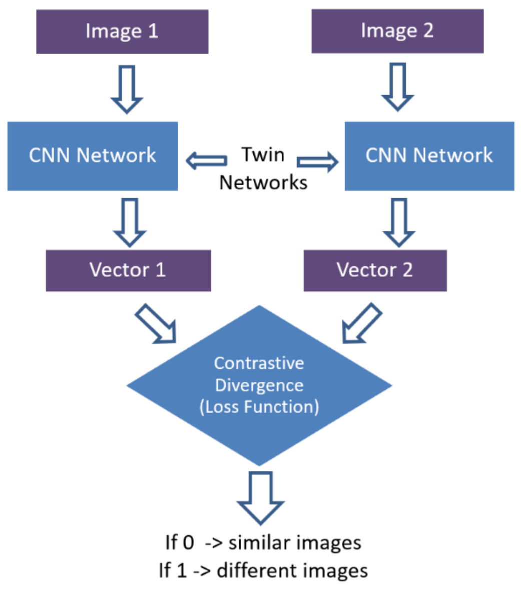


Figure 23. Siamese network

2.6.18 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 24. 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.

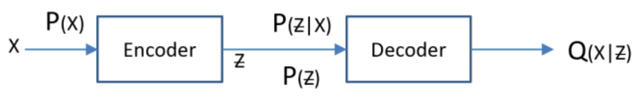


Figure 24. Variational Autoencoder

In Figure 24, 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 [81]. 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:

$$D_{KL}[Q(z|x)||P(z|x)] = \sum_{z} \mathrm{Q(z|x)} \log[\frac{\mathrm{Q(z|x)}}{\mathrm{P(z|x)}}] \quad (46)$$

$$= \mathrm{E}\ [\log[\frac{\mathrm{Q(Z|X)}}{\mathrm{P(Z|X)}}]] \quad (47)$$

$$= \mathrm{E}\ \log[\ \mathrm{Q(z|x)} - log\mathrm{P(z|x)}] \quad (48)$$

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

Using Baye's rule:

$$P(z|x) \quad = \frac{P(x|z)P(z)}{P(x)} \quad (49)$$

$$D_{KL}[Q(z|x)||P(z|x)] \ = E\ [\log Q\ (z|x) - Log\,\frac{P(x|z)P(z)}{P(x)}] \quad (50)$$

$$= E\ [\log Q\ (z|x) - \log P\ (x|z) - \ \log P\ (z)] + \log P\ (x) \quad (51)$$

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 μ(x) and

$\sum(x)$, then the KL divergence between $Q(z|x)$ and $P(z)$ can be derived in closed form as:

$$D_{KL}\left[N\left(\mu(\mathrm{x}), \Sigma(\mathrm{x})\right) | \quad | N(0,1)\right] = (1/2) \sum_{k} (\exp\left(\Sigma(\mathrm{x})\right) + \mu^2(x) - 1 - \Sigma(\mathrm{x})) \qquad (52)$$

2.6.19 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.

$$Q(s, a) = R + \gamma . \max\,[Q(s', a')] \qquad (53)$$

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[82]. 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. [83]. 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 [6].

2.6.20 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 (i.e., from training set) or generated (fake). 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., [84] in 2014. It has been very popular as it has many applications both good and bad. E.g., [85] were able to successfully synthesize realistic images from text.

2.6.21 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 fine-tuned 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 preselect a smaller but representative data set from the full training distribution set using instance selection methods [86] or Monte Carlo sampling [51]. 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 [87] 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 [87]. 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.

CHATPER 3: RESEARCH & RELATED WORK

The research section gets more specific about the topics pertaining to the implementation of ideas in the dissertation. It describes the concept of mitochondrial DNA and how it can extend the genetic algorithm. It takes a deeper dive into the training algorithm optimization and dataset optimization. Finally, it describes how cloud can be used to complement ML training.

3.1 Genetic Algorithm & mtDNA

There has been considerable amount of research to improve the GA operators to solve combinatorial optimization problems such as TSP. The development of several selection strategies mentioned earlier, i.e., elite, roulette, rank and tournament are a testimony of that effort. These strategies have been implemented and run against TSPLIB benchmarks[88] by different researchers. These selection operators each have their own characteristics, benefits and shortcomings. Razali et al. [89] concluded in the paper that rank based selection strategy yielded better results but took more computation time, while tour method is faster for small sized problems [89]. Selection methods represent only one side of the TSP problem. The other major side is crossover functions, which contributes significantly to the success of the algorithm. There are about eleven crossover operators reviewed by Larrañaga et al. [72] in their paper. Majority of them are based on specific patterns of information mixing and interchange between the parents, e.g., *order crossover*

(OXI), introduces several uniform length cut points in the path of the parents and produces offspring with several sub paths from the parents intact and assimilated in the children [90]. Another crossover operator, i.e., *genetic edge recombination* crossover add more meaningful logic in its workings by assuming the edges of the tour are important and attempts to preserve them in the offspring [91].

There are published literatures on restrictive crossover. Galan et al. [92] proposed a mating strategy that balances between exploration (selection criteria) and exploitation (fitness criteria) by developing a parameter called *mating index*, which controls the degree of exploration (or diversity) of parents based on the hardness of the problem. Strategies like *incest prevention* [93] prevents mating between similar individuals. Assortative Mating is another strategy used to improve GA results. Ochoa et al. [94] demonstrates the relation between mutation rates and assoratative mating choices, i.e., higher mutation rates work well with assortative mating whereas lower mutation rates work well with dissortative mating to confer better fitness. The idea behind these strategies is based on the principle that offspring of similar individuals do not result in higher fitness. Introducing controlled mating based on similarity of genes does yield better results but they are also computationally costly as the lengthy chromosomes have to be compared.

The dissertation presents a further optimized idea of restrictive mating to complement the standard crossover operators. The idea is based on the premise that it would not be beneficial to select the offspring of the same parents (or close lineage) as new parents to crossover with each other. In fact, it could be detrimental to maintaining diversity and exploring greater search space. As an alternate to exhaustive comparison of the genes to determine genetic diversity between the parents, we present an algorithm that is computationally lean. We exploit the concept of mtDNA to enhance the GA.

3.1.1 Mitochondrial DNA (mtDNA)

Humans have 23 pairs of chromosomes with one copy of each pair inherited separately from each parent [89]. The DNA in these chromosomes is referred to as nuclear DNA [95]. In addition, humans also have mtDNA [95], which consists of only 1% of the total DNA [96], thus coding for far less genes. Though insignificant by orders of magnitude when compared to the nuclear DNA in their contribution to inheritable traits (genes), mtDNA's unique characteristic in inheritance can play an important role in guiding the search for optimal solution. The DNA sequence in the 23 pairs of chromosomes is inherited equally into the offspring from both the parents during reproduction, whereas the sequence in mtDNA is inherited only from the maternal side [96]. This allows us to keep track of population members with similar genetic traits and common inheritance via maternal lineage. Diversity is the key to preventing premature convergence and achieving near optimal solution. Crossovers between similar population members with close DNA proximity will not yield results better than the prior generation in most cases. The idea in this paper is to create a data structure to tag and track the mtDNA in every population member and restrict the crossover between population members with similar mtDNA. mtDNA is widely used in evolutionary genetics and population study [97], and its concept could potentially be beneficial to GA search exploration.

3.1.2 Using mtDNA GA

The primary objectives of GA are to help get us better solution after every iteration and to prevent from prematurely converging into local minima. The primary way to address the later goal is to introduce the right amount of diversity in the parents.

Most of the crossover operators tend to be very refined and granular at the node information level and seems to overlook the bigger picture. As the GA undergoes several iterations of crossover, the risk of convergence increases too and, in some cases, crossovers between the similar population members' offspring will not yield results any better than the previous generation because their parents would have similar genetic information to begin with. With less genetic variance in the parents, we cannot expect better or different results in the offspring. It is self-evident that genetic variability sows the seed for evolution and newer offspring [98]. One way to track genetic similarity is by tracking the family lineage. And the most effective way to track inheritance in the real world is through mtDNA [99].

The concept of mtDNA (Mitochondrial DNA) is implemented in this paper to control the crossover function to prevent population members with same mtDNA from reproducing for *n* number of generations. To avoid the overreaching consequences of this condition, this requirement is dictated only on a percentage of crossovers. mtDNA is defined as a separate attribute of the population member class. Since mtDNA gets inherited solely from the female parent, it doesn't alter as it is passed down to the offspring. This attribute was exploited to guide and control crossovers. Algorithm 2 presents a high-level overview of the mtDNA algorithm and its pseudo code.

Algorithm 2: mtDNA pseudo code and Algorithm

```
1. $Initialization$
            $Assign\ mtDNA\ attribute\ to\ each\ population\ member$
                $\textbf{If}\ population = initial\quad \textbf{then}\ mtDNA$
                            $\leftarrow random\ unique\ value$
                $\textbf{If}\ population = offspring\ of\ crossover$
                        $\textbf{then}\ mtDNA \leftarrow mtDNA\ of\ female\ (2nd\ parent)$
2. $\textbf{for}\ i \leftarrow 1\ to\ Maximum\ Iterations$
            $a. Selection$
              $Check\ mtDNA\ attributes\ of\ crossover\ pairs\ at$
              $(total\ iteration\ \%\ 100) < N\ iterations, where\ N < 100$

                $\textbf{If}\ parent1\ mtDNA = parent2\ mtDNA$
                            $\textbf{then}\ abort\ crossover\ \&\ find\ another\ pair$
                $\textbf{If}\ parent1\ mtDNA \neq parent2\ mtDNA$
                        $\textbf{then}\ allow\ crossover$
                $\textbf{Reset}\ mtDNA\ attribute\ of\ all\ members\ to\ unique$
                      $values\ after\ X\ iterations.$
                          $X < \log_2 P, where\ P = total\ population$
            $b. Crossover\ \&\ mtDNA\ transfer$
        $Children's\ mtDNA \leftarrow mtDNA\ of\ the\ female\ parent$
```

All four selection methods (tour, elite, roulette and rank) described earlier were utilized during the implementation. To transfer genes to children during crossover, 1/4 to 3/4 tour cut was made on parent one and transmitted to the children. The rest was transferred in cyclic order from the second parent, skipping any cities that were already derived from the first parent, thus ensuring every city is represented in the child with no repetition. In addition to leveraging mtDNA in the implementation, various selection methods, Island Model, 2-Opt and distributed processing using multiple servers (*Continental Model*) were also utilized.

Figure 25 provides a high-level workflow of the GA implementation in this paper. Custom versions of GA were run on each of the four threads on each server.

Island Model was implemented with multi-core processors in server by running multiple threads in parallel. Each thread ran its own version of GA. Periodically after every X number of iterations/generations on each of the threads running the GA, a handful of randomly selected population members were exchanged between the threads. This process not only added more computing resources and improved the execution time of GA but also increased diversity and reduced initial sampling bias.

2-Opt was implemented by reviewing the best member (best solution so far) every X iterations/generations for local optimization. Two links/edges of the best member were swapped exhaustively to check if it improves the solution.

Island Model was further scaled with distributed processing by executing the above-mentioned implementation on several servers using Web Services (Service Oriented Architecture - SOA). We aptly named it *Continental Model*. Population members were randomly exchanged between these independently run Island Model GA implementations in different servers after a fixed number of iterations to achieve diversity and to reduce the likelihood of premature convergence.

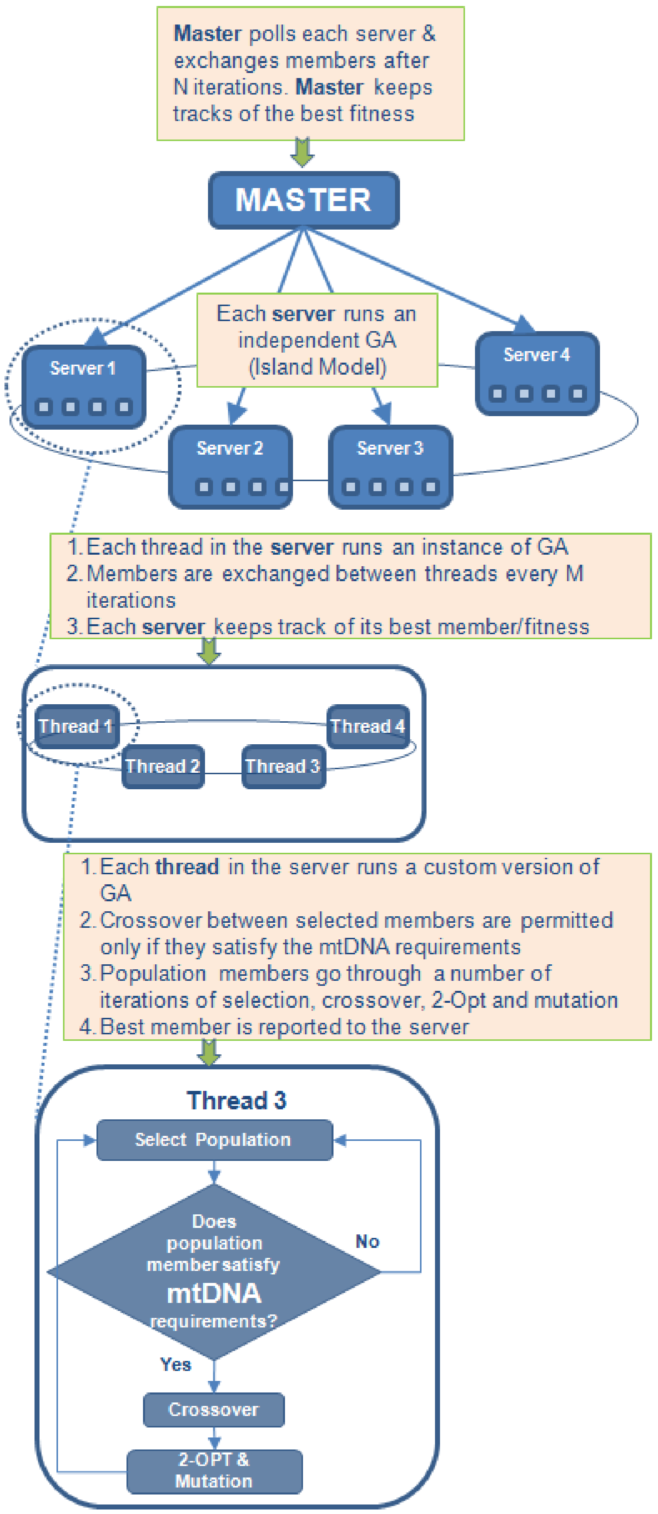

Master polls each server & exchanges members after N iterations. Master keeps tracks of the best fitness
MASTER
Each server runs an independent GA (Island Model)
Server 1
Server 2
Server 3
Server 4
1. Each thread in the server runs an instance of GA
2. Members are exchanged between threads every M iterations
3. Each server keeps track of its best member/fitness
Thread 1
Thread 2
Thread 3
Thread 4
1. Each thread in the server runs a custom version of GA
2. Crossover between selected members are permitted only if they satisfy the mtDNA requirements
3. Population members go through a number of iterations of selection, crossover, 2-Opt and mutation
4. Best member is reported to the server
Thread 3
Select Population
Does population member satisfy mtDNA requirements?
No
Yes
Crossover
2-OPT & Mutation

Figure 25: Continental Model GA with mtDNA

3.2 Optimization of Training Algorithms

The goal of the DNN training is to improve the accuracy of the model on test data and subsequently on real time 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 because the training algorithms assume the problem area to be a convex function. The other problems is high number of nodes and the sheer possible combinations 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 on a wide 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 DNN algorithms.

3.2.1. 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. [100] proposed normalized initialization where weights are initialized in the following manner.

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right] \qquad (54)$$

W represents the weight and U represents the uniform distribution in the interval, whereas n is the size of the previous layer.

[101] 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.

3.2.2 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. 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 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 [69].

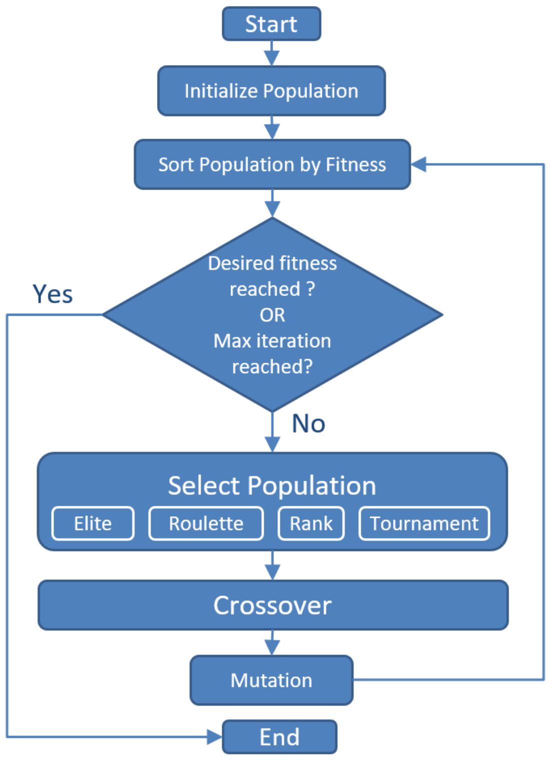


Figure 26. Genetic Algorithm

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 26 shows a high-level diagram of the GA. Figure 27 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.

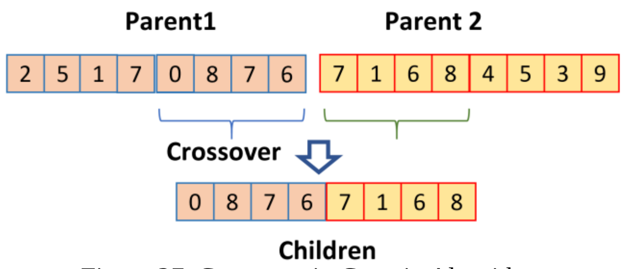


Figure 27. Crossover in Genetic Algorithm

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. [102, 103] 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. We incorporated 2-Opt local search method into GA to improve the search for optimal solution. [104] 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 lead to new breakthroughs. 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., [105] proposed CoDeepNEAT framework based on deep neuro-evolution technique for finding an optimized architecture to match the task at hand.

3.2.3 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. [51] 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 is more sophisticated [106] 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 [51]. Adam which denotes adaptive moment estimation is the latest evolution of the adaptive learning algorithms that integrates the ideas from AdaGrad, RMSProp and momentum [107]. Just like AdaGrad and RMSProd, Adam provides an individual learning rate for each parameter. Adam includes the benefits of both the earlier methods and does a better job handling non-stationary objectives, and both noisy and sparse gradients problems [107]. 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 [107].

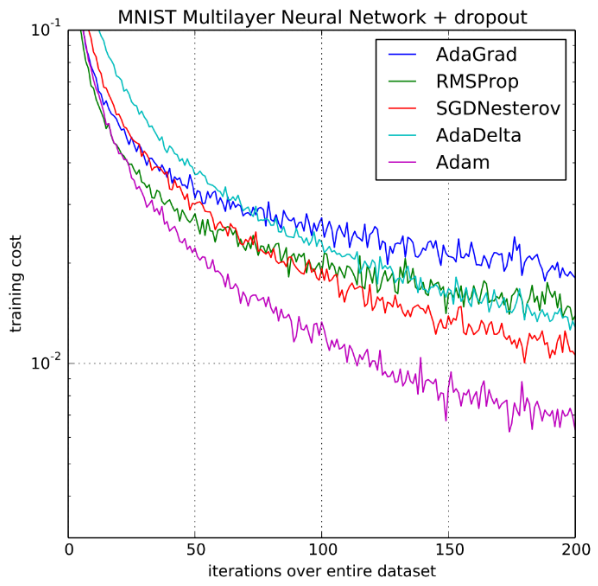


Figure 28. Multilayer network training cost on MNIST dataset using different adaptive learning algorithms [107]

Figure 28 shows the relative performance of the various adaptive learning rate mechanisms where Adam outperform the rest.

3.2.4 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 [108] 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.

3.2.5 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.

3.2.6 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 29 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. [109] proposed the dropout technique and demonstrated significant improvement on supervised learning based DNN for computer vision, computational biology, speech recognition and document classification.

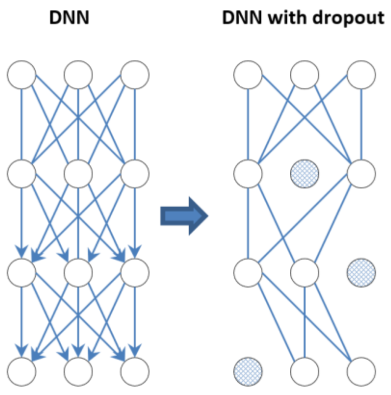


Figure 29. DNN with and without Dropout

3.2.7 Summary of DL Algorithms Shortcomings and Resolutions

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;
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

3.3 Dataset Optimization

Dataset or training data is a key contributor to the success of ML. It also provides an opportunity for optimization. The availability of data along with enhancements in training algorithms and commoditization of compute resources in the form of cloud and GPUs have been the driving force behind the exponential rise of artificial intelligence and machine learning. While data is everywhere nowadays, thanks to bigdata, not all data is good data from a training standpoint. In most cases they must be normalized and sanitized before they can be used in any training. The following attributes of the dataset can be reviewed and optimized: size, noise, dimensions, co-variance, variance, quality and distribution of the dataset, based on the problem we are trying to solve. E.g., as the dimensions become very high, finding the nearest neighbor instances become practically impossible compute-wise and we have to settle for sub-par approximates [110]. We need to reduce dimensions on large sized (dimension-wise) dataset otherwise they become intractable to work with. Principle component analysis (PCA) and auto-encoders are commonly used in dimensionality reduction. Even in the standard datasets like MNIST, CIFAR10, etc., not all samples have the same level of contribution to the training. E.g., a basic CNN can achieve high accuracy in a single epoch. There is diminishing returns for repetitive uniform training on the same/entire dataset for multiple epochs. Here are the commonly employed dataset optimization methods. Optimizing datasets can help us achieve significant speed up in the training process without compromising on the accuracy.

3.3.1. Instance Selection

Instance selection is a dataset reduction method used to decrease the training time and improve the accuracy of the model. The number of samples in the dataset puts a huge burden on training time of supervised training. Instance selection reduces the number of samples (instances) in the dataset using different approaches based on the goal. It can also help to remove noisy samples from the dataset that do not add any value to or in some cases mislead the training.

According to [111], instance selection can be divided into two groups: Wrapper and Filter. The wrapper method (e.g., k-NN classifier) discards instances that do not improve the accuracy of the model, whereas filter methods selects instances based on desired location, i.e., instances around decision boundary or interior instances. Instances around boundary could be noise or outliers and dropping them can smoothen the decision boundary and improve accuracy. Dropping boundary instances could also lead to over generalization. On the other hand, retaining all interior instances with similar attributes might not contribute to training accuracy.

There is a directly proportional relationship between retention rate of instances (of instance selection method) and the accuracy of the model and an inversely proportional relationship with training time. [112] proposes RDI (Remove Dense Instances) method that balance the two competing constraints, such that we remove instances from denser regions, which does not impact the accuracy much while significantly reducing the training time. Instance selection itself can be an optimization problem as selecting each instance is a binary decision problem, thus giving rise to 2^x potential subsets from x sample instances [113]. Therefore, it could be solved using metaheuristics.

3.3.2. Variable Selection & Dimension Reduction

MNIST dataset includes 60k 28x28 pixel images. Each image is a vector of 784 (28x28) variables or dimensions. While 784 is large, it is miniscule compared to 60k variables in gene selection problem, where models are trained to classify whether gene expression profiles of patients mRNA sample are cancerous or healthy [114]. Consider another example where blood pressure forecasting needs to be done based on over 500k dimensions, i.e., the number of single nucleotide polymorphisms or individual DNA mutations common in a population [115]. This can be looked at as a dimensionality reduction problem. [115] reports that once the dimensions (or features) becomes larger than the number of samples (or observations), the *least square* method doesn't work because the *mean squared error* reduces to zero even when the features are not related.

Several ranking methods based on the contributing factor of the variable both at individual variable level and at a collective subset level have been proposed to select the variables in [114]. Principal component analysis (PCA) is another statistical approach to reduce the dimensions. Here are three methods described in [115].

a) Best Subset Selection Method

This method explores whether to include all possible combinations of the dimensions, thus resulting in 2^n potential subsets. Each combination has to fit on individual run of least squares regression, making it computationally very expensive.

There have been other alternatives that propose only exploring a small portion of all possible combinations of the subsets.

b) Shrinkage Method

This method attempts shrink or constraint the coefficient estimates toward zero to fit a model with all features. Before fitting a model, all predictors (features) are standardized to have one standard deviation.

c) Partial Least Square (PLS)

PLS selects a new set of features using least squares method in a supervised way by utilizing the labeled outputs as well as the original predictors.

3.3.3. Monte Carlo Method

Monte Carlo method constitutes a set of algorithms used in optimization, bayesian inference and drawing representative samples from a probability distribution in very high dimensional space. It is used is several different disciplines including engineering, design, finance, law, business, etc. In machine learning, it can be used to approximate computationally expensive sums and integrals in training, and in sampling instances from large datasets. Marcov Chain Monte Carlo (MCMC) and Importance sampling (IS) are very popular implementation of Monte Carlo method for sampling complex distributions.

Bayes’ theorem states the following about posterior distribution:

$$P\ (X \mid Y) = \frac{\mathrm{P}\ (\mathrm{Y} \mid X)\ P(X)}{P\ (Y)} \qquad (55)$$

Where P(X | Y) is the probability of X occurring provided Y is true and the opposite for P(Y | X).

MCMC constructs a Markov chain with a stationary distribution π as the target distribution and the samples produced X_i are revised as X_i' [116].

3.3.4. Importance Sampling

Importance sampling is a variance reduction method where the goal is to sample more often from the instances that help improve the accuracy of the model than others. This is done by sampling from a different distribution than the original distribution that reduces the variance of the gradient estimation. Importance sampling provides good representation of the dataset. It also helps with convergence as it reduces major fluctuations in the gradient estimates. The new distribution maintains the original expectation value. Below is the mathematic explanation [117].

$$\mathbb{E}\, f(x) = \int f(x)\, p(x)\ dx \tag{56}$$

Where $\mathbb{E}f(x)$is the expectation for function of x. $p(x)$ is the original probability distribution of x.

$$\mathbb{E}_{p(x)}\, [f(x)] = \int \left[f(x)\frac{\mathrm{p(x)}}{\mathrm{q(x)}}\mathrm{q(x)}\right] dx \tag{57}$$

Where $q(x)$ is the proposed distribution of x that maintains the original expectation value.

$$\mathbb{E}_{p(x)}\, [f(x)] = \mathbb{E}_{q(x)}[f(x)\, w(x)]\, dx \tag{58}$$

Where $w(x) = \frac{\mathrm{p(x)}}{\mathrm{q(x)}}$ is referred to as importance weight function.

[118] proves the advantages of importance sampling over another popular sampling method (Bernoulli), where 50 points sampled from 1000 points using

importance sampling represented all 10 clusters whereas Bernoulli sampling failed the same test. In addition to sampling a good representation of the original dataset, importance sampling also improves the convergence of the training process by limiting large swings in the gradient estimates [47, 48, 119]. This validates the use of importance sampling to improve the training time and cluster quality.

3.4 Training Speed up with Cloud and GPU processing

GPUs have been the workhorse of machine learning and compute resources have been further commoditized by cloud. We can leverage the massive scaling power of cloud providers in a cost-effective manner. Combining the power of cloud and GPUs can extend the outcome of deep learning. 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. In addition to helping us train and build ML models faster, Cloud can also help us explore better architectures with its massive computational power. All major cloud vendors include GPU powered servers that can easily be provisioned and used for training DNNs on demand at competitive prices. We can leverage the parallel and distributed processing power of cloud to not only reduce the training time but also deploy machine learning models to live systems and respond to real-time test input at a much faster rate than traditionally possible. Cloud vendor Amazon Web Services’ (AWS) P2 instances provides up to 40 thousand parallel GPU cores and its P3 GPU instances are further optimized for machine learning [120].

CHAPTER 4: IMPLEMENTATION

This section presents that actual implementation of the dissertation topic and contribution of the work. There are three ideas implemented here. First, the mtDNA enhanced genetic algorithm is implemented on a neural network to solve for function approximation problem. Then the value-add of importance sampling is verified on a siamese network. Next, we combine both these ideas with hyperparameter and architecture optimization for designing better CNN model. We also fine-tune the training with importance sampling and finally achieve training speed up with GPUs in the cloud.

4.1 mtDNA GA in Artificial Neural Network

We used GA to train the Neural Network for function approximation. A multi-layer feedforward ANN (artificial neural network), with 1 node in the input layer, 26 nodes in the first hidden layer, 26 nodes in the second hidden layer and 1 node in the output layer was chosen. The GA implementation for Neural Network is similar to the GA implementation for TSP. In place of the city numbers (in TSP), the values of the weights are randomly initialized in a solution set and crossed over with another set of weights in the case of Neural Network. But unlike in TSP, the values of the weights do not need to be unique within a solution set.

mtDNA was introduced in GA here just like in TSP. The resulting children from crossovers were tagged with the same mtDNA attribute of the female parent for *X* iterations as defined in Algorithm 2 and crossovers prevented between those with same mtDNA. mtDNA value was reset after X iterations to ensure crossovers weren't too restrictive. We used mtDNA implementation of GA to train the Neural Network for function approximation for functions show in Figure 30.

Here are the details from the implementation:

1. Population size = 200
2. Selection Ratio = 20
3. Mutation Ratio = 4
4. mtDNA reset every X iterations, where $X = 15$
5. Nodes: 1 (input layer): 26 (hidden layer1): 26 (hidden layer2): 1 (output layer)

GA was used to train ANN for the following functions with and without the mtDNA logic (Algorithm 2).

Function A: $f(x) = \frac{x \sin(50\, x)}{e^2}$

Function B: $f(x) = 250 \sin(2x) \sin(x)$

Function C: 1D version of Schewefel function

$$f(x, y) = -x \sin\left(\sqrt{|\mathrm{x}|}\right) - \mathrm{y} \sin\left(\sqrt{|\mathrm{y}|}\right)$$

Function D: 1D version of $f(x, y) =$

$$(16\, x\, (1 - x)\, y(1 - y) \sin(9\, \pi\, x) \sin(9\, \pi\, y))^2$$

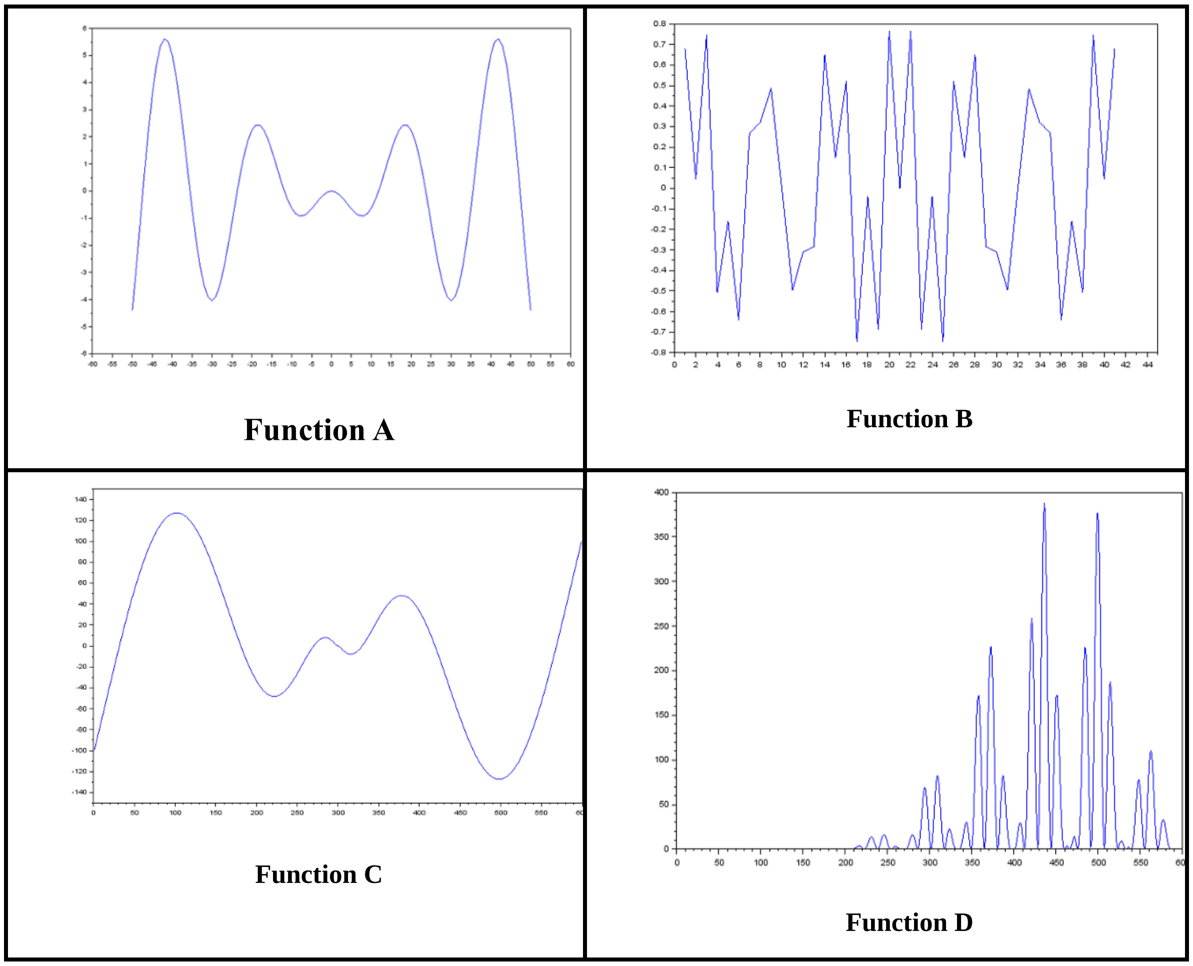


Figure 30: Four functions used for training Neural Network

4.2 Siamese – Importance Sampling Implementation

We implemented importance sampling on siamese networks using TensorFlow [121, 122] ML framework. Since the goal was to test the validity of employing importance sampling on siamese network, we did not optimize the network for best performance to compete with published accuracy results on the dataset. Rather the focus was on demonstrating relative difference between siamese network that uses importance sampling versus one that doesn’t.

We started with a standard 3-layer CNN. We trained it on MNIST dataset, both with full/uniform dataset and then with importance sampling. We confirmed that the training and testing accuracy were better when importance sampling was used. Next, we saved the instances/values of the sampled dataset for the next phase.

We built a siamese network. The twin networks were setup with three fully connected layers with 784 x 1024 x 1024 x 400 nodes. The instances of the samples/values selected by importance sampling were used to complement the training. The testing errors were compared with results from siamese network that used the full dataset without any importance sampling. A mini-batch of 64 samples were used for 80 epochs.

The siamese network was trained and tested on MNIST dataset. The dataset was shuffled at every time a mini-batch was selected for training. The trained network was evaluated on the full test dataset.

Algorithm 3: Pseudocode for implementing importance sampling on siamese networks

```
1. Importance Sampling on Standard CNN
        Build CNN network and load dataset (MNIST)
            Start importance sampling
            Train 10 epochs ← original dataset
            Train 10 epochs ← importance sampling instances
2. Siamese Networks with Importance Sampling
        Build siamese network and load importance sampled instances
          Train siamese network
            a. Fine − tune training with instances from importance sampling
            b. Train with original dataset
3       Test trained models on Test data on full dataset training and importance sampling training
```

We also used another way of picking the samples for training and testing. The training of siamese network differs from traditional training of CNN. A trained siamese network is used to indicate whether the two images passed through the twin networks are same or different. Therefore, we trained the network alternately with similar and differently labelled input samples as well. Each mini-batch consisted of 50% similar and 50% differing label samples. The testing results were captured. Algorithm 3 shows the flow used in the implementation.

4.3 mtDNA Based Genetic Algorithm to Optimize Hyperparameters and Build CNN

In this section we incorporated the idea of mtDNA based genetic algorithm to build a robust CNN to maximize accuracy of the model. We came up with the concept and used it for learning the weights for function approximation and solving TSP. We extended it to hyperparameter optimization. Table 5 lists the hyperparameters that will be optimized.

Table 5: Implementation plan table

Layer	Hyperparameter	Value
Convolution Layer	Enabled	0 or 1
	Number of Layers	6 (Max)
	Batch Normalization	0 or 1
	Activation Function	Relu (1), Sigmoid (2)
	Dropout	0-10
	Number of Kernels	256 (Max)
Fully Connected Layer	Enabled	0 or 1
	Number of Nodes	1024 (Max)
	Batch Normalization	0 or 1
	Activation Function	Relu (1), Sigmoid (2)
	Dropout	0-10
	Optimizer	Adadelta, Adagrad, Adam, RMSProp

Here are the steps that were followed (visually represented in Figure 31):

1. Define each network setup as a specific combination of above hyperparameters (Table 5).
2. The specific combination of hyperparameters is represented as a genetic sequence of a population member that is being optimized in the context of genetic algorithm.
3. Each member (would-be parent) is tagged with a mtDNA data structure (attribute).
4. Population members are selected for crossover but two partners with the same mtDNA value are restricted from crossover operation.
5. The crossover between population members are conducted to derive a new genetic sequence (children).
6. Mutation is also introduced
7. The crossover is iterated over number of generations. mtDNA values are reset after X number of generations to prevent over-restrictive crossover as defined in Algorithm 2 above.
8. The whole process is repeated for number of epochs.

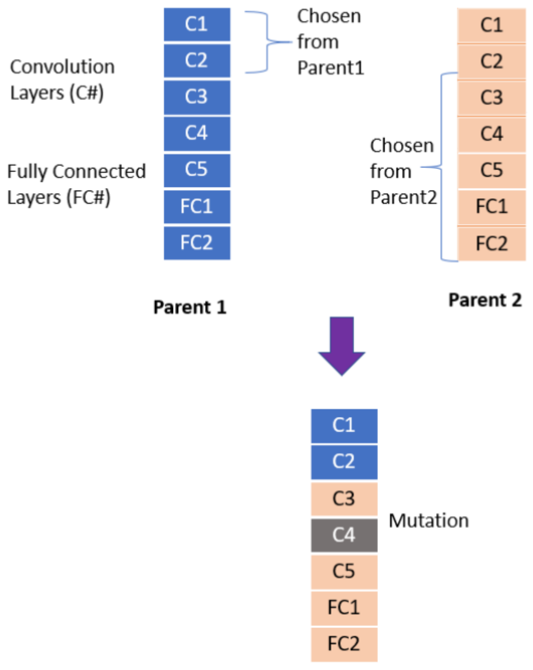


Figure 31: Child after cross-over

4.4 Putting the Plan Together

The proposed plan for enhancing deep CNN is summarized in Table 6 below.

Table 6: Implementation plan table

	Method	Goal & Benefits	How	Specifics
1.	Architecture Optimization	Improve Accuracy	Using Genetic Algorithm	mtDNA (Mitochondrial DNA) based fine-tuning
2.	Dataset Optimization	Improve Accuracy, Reduce Overfitting	Monte Carlo method	Importance Sampling
3.	Computation Optimization	Reduce Training Time	Scalable, Distributed & Paralleling Processing	Use of Cloud and GPUs

These approaches and specific implementations were chosen for their uniqueness and potential for significant contributions. Individually these approaches have proven effective and they complement each other when combined as well.

4.5 Implementation Steps

1. Setup machine learning environment with applicable Keras and TensorFlow libraries/framework [121, 122].
2. Choose Dataset: MNIST / CIFAR10
3. Choose Network to optimize: CNN
   a. Identify/set architecture parameters (Table 5) values/ranges.
   b. Choose optimization method:
      i. Genetic Algorithm with mtDNA logic
   c. Set training constraints
      i. Number of generations
      ii. Population size
      iii. Number of epochs

4. Choose Cloud service: AWS (Amazon Web Services)
5. Run multiple epochs of training
    a. Sample dataset using *importance sampling*.
    b. Optimize hyperparameters with mtDNA GA
    c. Run training on server with GPU (p2.xlarge) with 2500 cores.
        i. P2 instances provides up to 40 thousand parallel GPU cores [120]
6. Collect accuracy from the runs above with and without mtDNA implementation

CHAPTER 5: RESULTS

This section reports the results from the implementation presented in Chapter 4 above. There are results from implementing mtDNA GA, importance sampling, hyperparameter and architecture optimization using cloud/GPUs to improve neural network training.

5.1 mtDNA GA in Artificial Neural Networks

The ANN was trained separately using mtDNA implementation of GA and GA by itself. After the weights were set, the ANN was used to approximate the four functions (Figure 30) and the square error was computed. The results from the mtDNA implementation of GA as listed in Algorithm 2 were better than the results when GA was used by itself across all four functions.

Table 7: RMS for different functions, with and without mtDNA GA training

Function	**Training Algorithm**	**RMS Error per Iteration**					
		100	200	300	500	1000	5000
A	Standard GA	1.	1.78	1.78	1.77	1.75	0.232
	mtDNA GA	1.	1.77	1.77	1.76	0.92	0.203
B	Standard GA	2.	2.92	2.90	2.87	2.82	2.666
	mtDNA GA	2.	2.91	2.90	2.87	2.80	2.610
C	Standard GA	0.	0.46	0.42	0.37	0.29	0.282
	mtDNA GA	0.	0.41	0.35	0.30	0.21	0.121
D	Standard GA	0.	0.59	0.59	0.59	0.58	0.371
	mtDNA GA	0.	0.58	0.58	0.58	0.53	0.314

Table 7 shows the results from the several (100, 200, 300, 500, 1000 and 5000) iterations/executions using both the training methods. Figure 32 represents the data from the Table 4 in graphical form.

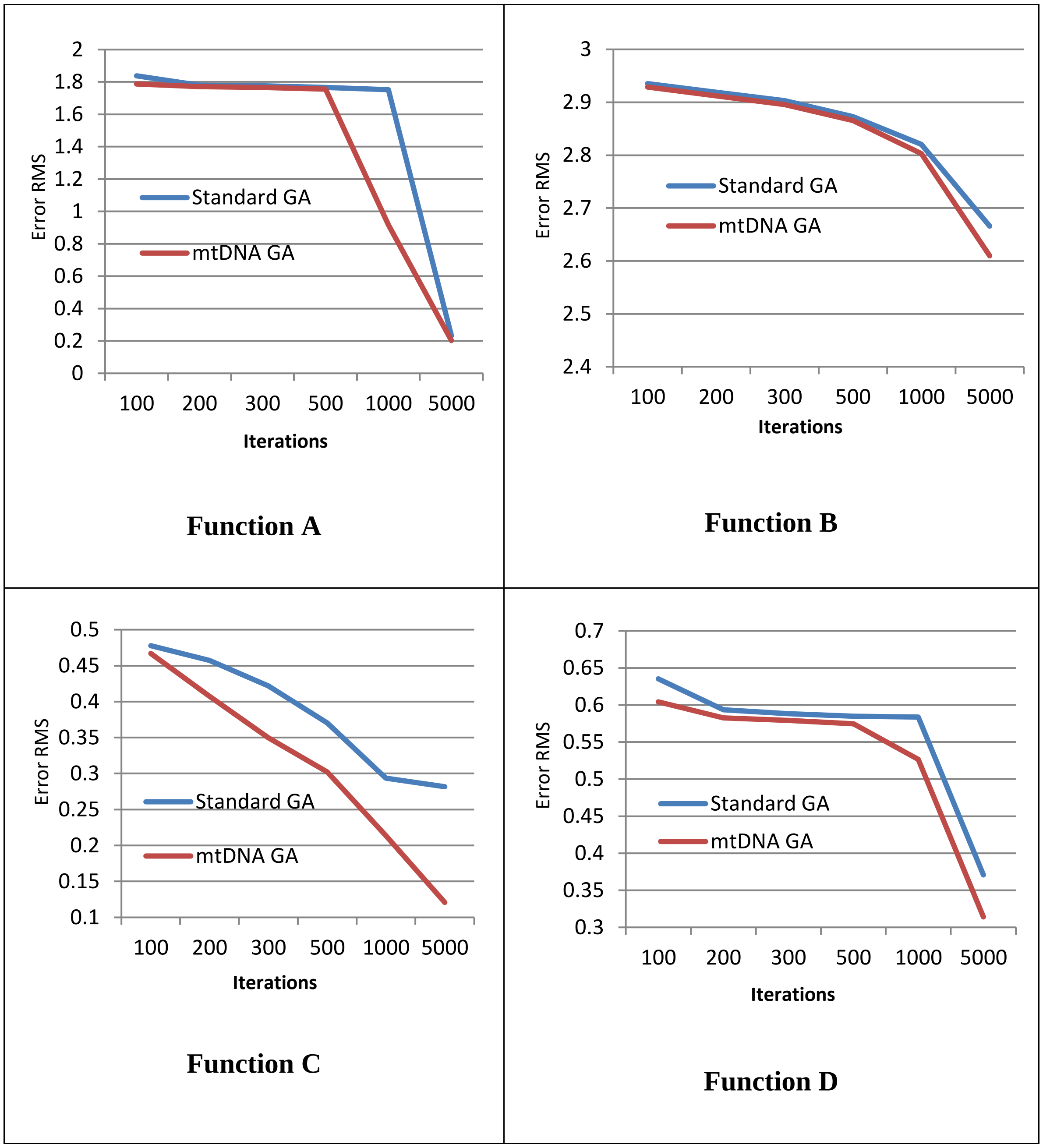


Figure 32: Graph shows Error with GA mtDNA and standard GA

The results from mtDNA incorporated GA trained ANN consistently outperforms the GA-only trained ANN for the given four function approximations.

5.2 Siamese Network Training with Importance Sampling

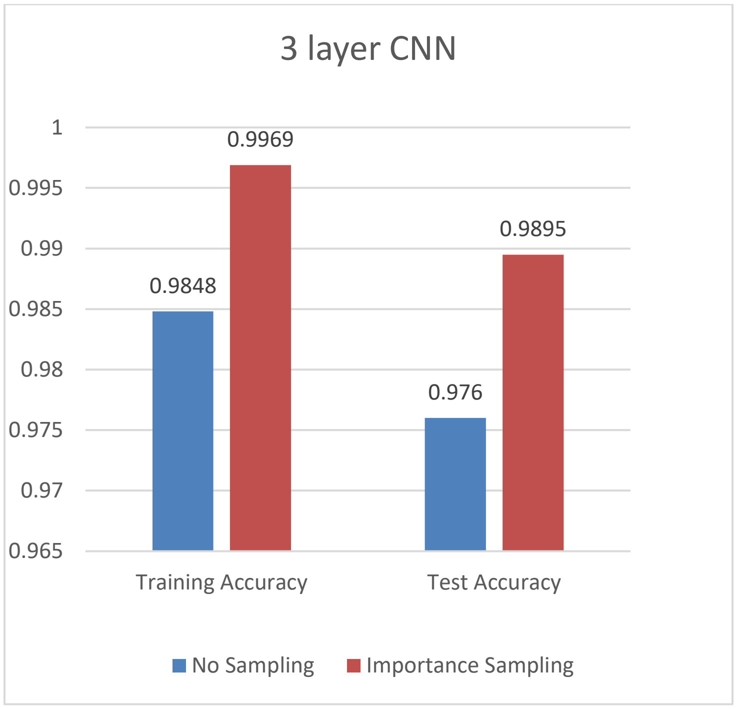


Figure 33: Training loss and testing accuracy on CNN

We were able to achieve improvement on both training and testing data when the network was fine-tuned with importance sampling. Since our goal was to provide evidence of relative positive impact of using importance sampling, we did not intentionally compare our results to state of the art.

Figure 33 shows the training and testing accuracy of using importance sampling vs using the full dataset with no sampling on standard CNN. This provided us validation that importance sampling improves accuracy and gave us more confidence in using sampled instances for siamese network as well. Figure 34 shows the testing loss on siamese networks with and without importance sampling fine-tuning on regular dataset and dataset that includes 50% similar (labels) samples. The results demonstrate the efficacy of using importance sampling in getting better accuracy of the trained model.

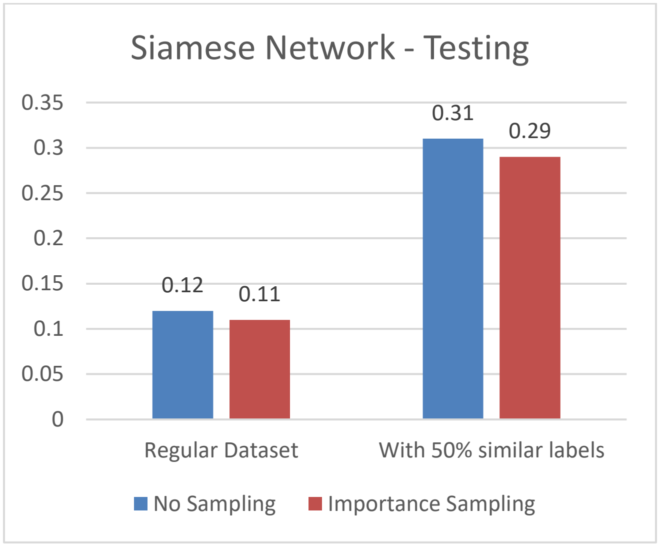


Figure 34: Testing loss on siamese Networks

5.3 Build CNN with mtDNA GA and Importance Sampling

Here are results from implementing mtDNA GA and importance sampling for hyperparameter and architecture optimization with cloud computing. We compared the results of constructing the deep CNN with the following two ways.

- With standard Genetic Algorithm
- With mtDNA based Genetic Algorithm and Importance Sampling

Figure 35 displays the testing accuracy on the models trained with these two options. Table 8 provides the CNN architecture details (hyperparameters and size) of the models trained on the two options.

Table 8: CNN architecture trained with Standard GA vs mtDNA GA w/ Importance Sampling (on MNIST)

CNN Architecture	**Hyperparameter**	**Standard GA**	**mtDNA GA and IS**
Convolution Layer 1	Enabled	1	1
	Number of Kernels	128	256
	Batch Normalization	0	0
	Activation Function	0	0
	Dropout	3	2
	Max Pooling	1	1
Convolution Layer 2	Enabled	0	1
	Number of Kernels	64	64
	Batch Normalization	0	0
	Activation Function	0	1
	Dropout	10	8
	Max Pooling	2	0
Convolution Layer 3	Enabled	1	0
	Number of Kernels	32	8
	Batch Normalization	1	1
	Activation Function	1	0
	Dropout	3	7
	Max Pooling	0	2
Convolution Layer 4	Enabled	0	1
	Number of Kernels	8	128
	Batch Normalization	1	1
	Activation Function	1	0
	Dropout	1	5
	Max Pooling	0	0
Convolution Layer 5	Enabled	1	1
	Number of Kernels	128	16
	Batch Normalization	1	0
	Activation Function	0	1
	Dropout	5	7
	Max Pooling	0	0
Convolution Layer 6	Enabled	0	0
	Number of Kernels	8	16
	Batch Normalization	0	0
	Activation Function	0	1
	Dropout	3	2
	Max Pooling	0	0
Fully Connected Layer	Enabled	1	1
	Number of Nodes	32	512
	Batch Normalization	0	1
	Activation Function	1	0
	Dropout	6	7
	Optimizer	Adadelta	RMSProp
Testing	**Loss**	**0.0335**	**0.0188**
	Accuracy	**0.9901**	**0.994**

The accuracy of the model when mtDNA GA and importance sampling were used on MNIST was 99.4%. When those additions were not used, the accuracy was 99%. Thus, adding mtDNA and importance sampling contributed to improvement of .4% to make the total accuracy of 99.4%, which is significant in the context of MNIST dataset. We also able to get better results with our implementation on CIFAR10 dataset, i.e., 73% accuracy vs 67%, when mtDNA GA and importance sampling were not used.

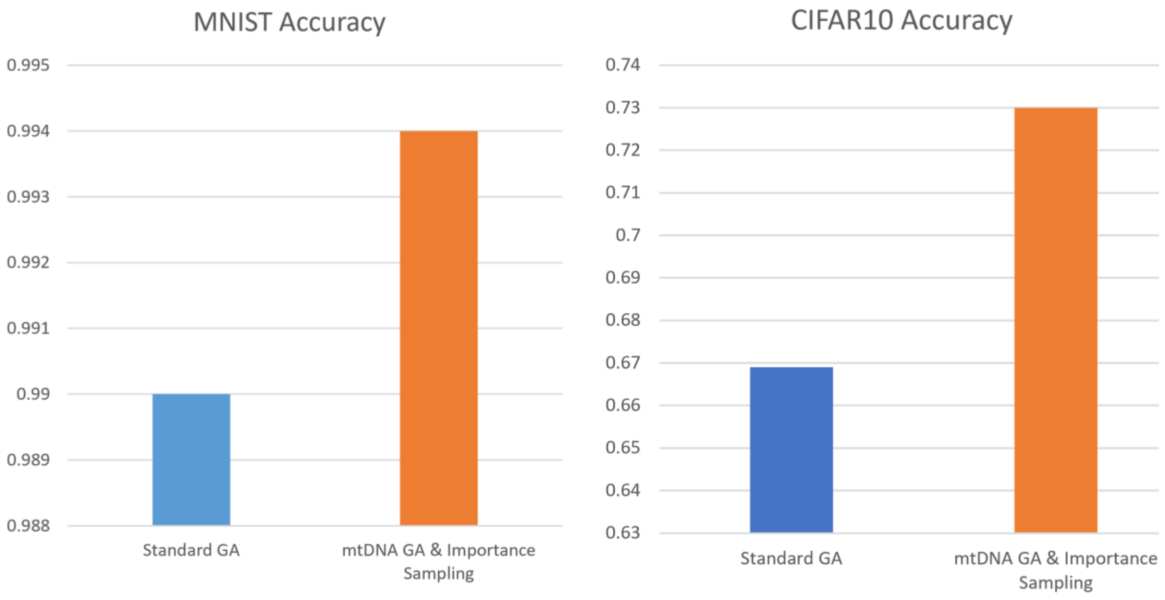


Figure 35: Testing accuracy on CNN

CHAPTER 6: CONCLUSION

In first half of this paper covered 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 have presented two important ideas of mtDNA based Genetic Algorithm and the use of importance sampling to optimize the deep neural network training and design.

The mtDNA logic introduced in the paper is a novel idea and is inspired by nature just like many of the optimization algorithms, e.g., genetic algorithm, swarm intelligence, ant colony optimization, neural network, etc. Like these nature inspired algorithms and systems, the concept of mtDNA is not very complex but can be instrumental in improving outcomes of genetic algorithm. Maintaining diversity is the key to preventing premature convergence into local minima. The characteristics of mtDNA can be exploited to track diversity and restrict crossover between parents of same genetic traits, thus yielding better fitness value in the offspring. The mtDNA concept articulated and implemented here mimics the natural order where it is an established fact that biodiversity favors evolution and produces more adaptable offspring. We were able to use the concept of mtDNA in GA to improve the outcome of neural network training. We validated the efficacy of mtDNA enhanced genetic algorithm for optimizing the deep neural network hyperparameters by getting better results on MNIST and CIFAR10 datasets when compared to standard GA. They hyperparameter space is huge and the parameters themselves display non-linear and interactive influence on performance. Solving it with expert knowledge, fixed formula or manually exhaustive methods is not feasible. Our implementation of mtDNA GA provide a heuristic based guided navigation of the solution space and results in better outcome. It is grounded on the premise that the right amount of diversity can improve the fitness in the next generation after crossover operations. It helps improve both exploration and exploitation of the solution space.

We were able to demonstrate that importance sampling, a variance reduction method can successfully improve the training and testing accuracy of the siamese network and CNN. This the first known attempt to combine importance sampling with siamese network. We have empirically demonstrated the validity of using importance sampling to fine-tune the training.

Finally, we were able to incorporate both mtDNA based genetic algorithm and importance sampling to build and train deep CNN and leverage the power of the cloud to achieve differentiable improvement in model accuracy.

REFERENCES

[1] A. Ng, Machine Learning Yearning: Technical Strategy for AI Engineers In the Era of Deep Learning, deeplearning.ai, 2019.

[2] C. Metz, "Turing Award Won by 3 Pioneers in Artificial Intelligence," New York Times, The New York Times Company, 3/27/2019, pp. B3.

[3] K. Nagpal et al., "Development and Validation of a Deep Learning Algorithm for Improving Gleason Scoring of Prostate Cancer," CoRR, vol. abs/1811.06497, 2018.

[4] S. Nevo et al., "ML for Flood Forecasting at Scale," CoRR, vol. abs/1901.09583, 2019.

[5] A. Esteva et al., "Dermatologist-level classification of skin cancer with deep neural networks," Nature, vol. 542, 01/25/online 2017, pp. 115.

[6] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," IEEE Signal Processing Magazine, vol. 34, no. 6, 2017, pp. 26-38.

[7] M. Gheisari, G. Wang, and M. Z. A. Bhuiyan, "A Survey on Deep Learning in Big Data," IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC), 2017, vol. 2, pp. 173-180.

[8] S. Pouyanfar et al., "A Survey on Deep Learning: Algorithms, Techniques, and Applications," ACM Comput. Surv., vol. 51, no. 5, 2018, pp. 1-36.

[9] R. Vargas, A. Mosavi, and R. Ruiz, "Deep learning: A review," ed, 2017.

[10] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," Psychol Rev, vol. 65, no. 6, Nov 1958, pp. 386-408.

[11] M. Minsky and S. Papert, Perceptrons; an introduction to computational geometry. Cambridge, Mass.,: MIT Press, 1969, pp. 258.

[12] G. Cybenko, "Approximation by superpositions of a sigmoidal function," Mathematics of Control, Signals and Systems, vol. 2, no. 4, 1989 pp. 303-314.

[13] K. Hornik, "Approximation capabilities of multilayer feedforward networks," Neural Networks, vol. 4, no. 2, 1991, pp. 251-257.

[14] P. J. Werbos, "Beyond regression : new tools for prediction and analysis in the behavioral sciences," 1975.

[15] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, May 28 2015, pp. 436-44.

[16] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," Science, vol. 349, no. 6245, Jul 17 2015, pp. 255-60.

[17] M. D. Buhmann and M. D. Buhmann, Radial Basis Functions. Cambridge University Press, 2003, pp. 270.

[18] A. A. Akinduko, E. M. Mirkes, and A. N. Gorban, "SOM: Stochastic initialization versus principal components," Information Sciences, vol. 364-365, 2016/10/10/ 2016, pp. 213-221.

[19] K. Chen, "Deep and Modular Neural Networks," in Springer Handbook of Computational Intelligence, J. Kacprzyk and W. Pedrycz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 473-494.

[20] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes," presented at the Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, Vancouver, British Columbia, Canada, 2001.

[21] C. M. Bishop and J. Lasserre, "Generative or Discriminative - Getting the Best of Both Worlds," 2007.

[22] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised Learning of Depth and Ego-Motion from Video," CoRR, vol. abs/1704.07813, / 2017.

[23] X. W. Chen and X. Lin, "Big Data Deep Learning: Challenges and Perspectives," IEEE Access, vol. 2, 2014, pp. 514-525.

[24] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in Proceedings of 2010 IEEE International Symposium on Circuits and Systems, 2010, pp. 253-256.

[25] G. Gousios, B. Vasilescu, A. Serebrenik, and A. Zaidman, "Lean GHTorrent: GitHub data on demand," presented at the Proceedings of the 11th Working Conference on Mining Software Repositories, Hyderabad, India, 2014.

[26] AI-Index. (2019). Top deep learning Github repositories. Available: https://github.com/mbadry1/Top-Deep-Learning

[27] M. Fern´andez-Delgadondez-Delgado, E. Cernadas, S. e. Barro, and D. Amorim, "Do we need hundreds of classifiers to solve real world classification problems?," J. Mach. Learn. Res., vol. 15, no. 1, 2014, pp. 3133-3181.

[28] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, 1998, pp. 2278-2324.

[29] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," in The handbook of brain theory and neural networks, A. A. Michael, Ed.: MIT Press, 1998, pp. 255-258.

[30] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler, "Convolutional Learning of Spatio-temporal Features," in Computer Vision – ECCV 2010, Berlin, Heidelberg, Springer Berlin Heidelberg, 2010, pp. 140-153.

[31] A. Ng. (2018, 7/21/2018). Convolutional Neural Network. Available: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

[32] C. J. Schuler, H. C. Burger, S. Harmeling, and B. Schölkopf, "A Machine Learning Approach for Non-blind Image Deconvolution," in 2013 IEEE Conference on Computer Vision and Pattern Recognition, 2013, pp. 1067-1074.

[33] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," CoRR, vol. abs/1511.06434, 2015.

[34] I. T. Jolliffe, Principal component analysis, 2nd ed. (Springer series in statistics). New York: Springer, 2002, pp. xxix, 487.

[35] K. Noda, H. Arie, Y. Suga, and T. Ogata, "Multimodal integration learning of object manipulation behaviors using deep neural networks," in 2013 IEEE/RSJ

International Conference on Intelligent Robots and Systems, 2013, pp. 1728-1733.

[36] G. E. Hinton and R. R. Salakhutdinov, "Reducing the Dimensionality of Data with Neural Networks," Science, 10.1126/science.1127647 vol. 313, 2006, no. 5786, p. 504.

[37] M. Wang, H. X. Li, X. Chen, and Y. Chen, "Deep Learning-Based Model Reduction for Distributed Parameter Systems," IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 46, no. 12, 2016, pp. 1664-1674.

[38] A. Ng. (2018, 7/21/2018). Autoencoders. Available: http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders

[39] Y. W. Teh and G. E. Hinton, "Rate-coded Restricted Boltzmann Machines for Face Recognition," 2001, pp. 908-914.

[40] G. E. Hinton, "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, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 599-619.

[41] S. Hochreiter and J. Schmidhuber, Long Short-term Memory. 1997, pp. 1735-80.

[42] C. Metz. (2016) Apple is bringing the AI Revolution to your Phone. Wired. Available: https://www.wired.com/2016/06/apple-bringing-ai-revolution-iphone/

[43] F. A. Gers, J. Schmidhuber, and F. A. Cummins, "Learning to Forget: Continual Prediction with LSTM," Neural Computation, vol. 12, no. 10, 2000, pp. 2451-2471.

[44] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," eprint arXiv:1412.3555, 2014, p. arXiv:1412.3555.

[45] K. Cho et al., "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," eprint arXiv:1406.1078, 2014, pp. arXiv:1406.1078.

[46] B. Naul, J. S. Bloom, F. Pérez, and S. van der Walt, "A recurrent neural network for classification of unevenly sampled variable stars," Nature Astronomy, vol. 2, no. 2, 2018/02/01 2018, pp. 151-155.

[47] P. Zhao and T. Zhang, "Stochastic Optimization with Importance Sampling for Regularized Loss Minimization," presented at the Proceedings of the 32nd International Conference on Machine Learning, Proceedings of Machine Learning Research, 2015.

[48] A. Katharopoulos and F. Fleuret, "Not All Samples Are Created Equal: Deep Learning with Importance Sampling," CoRR, vol. abs/1803.00942, 2018.

[49] R. Riad, C. Dancette, J. Karadayi, N. Zeghidour, T. Schatz, and E. Dupoux, "Sampling strategies in Siamese Networks for unsupervised speech representation," Computing Research Repository (CoRR), vol. abs/1804.11297, 2018.

[50] M. M. Najafabadi, F. Villanustre, T. M. Khoshgoftaar, N. Seliya, R. Wald, and E. Muharemagic, "Deep learning applications and challenges in big data analytics," Journal of Big Data, vol. 2, no. 1, 2015/02/24 2015, pp. 1.

[51] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning (Adaptive computation and machine learning). Cambridge, Massachusetts: The MIT Press, 2016, pp. xxii, 775.

[52] H. P. Gavin, "The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems," 2016.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778.

[54] A. J. R. Simpson, "Uniform Learning in a Deep Neural Network via "Oddball" Stochastic Gradient Descent," CoRR, vol. abs/1510.02442, 2015.

[55] L. Best-Rowden, H. Han, C. Otto, B. F. Klare, and A. K. Jain, "Unconstrained Face Recognition: Identifying a Person of Interest From a Media Collection," IEEE Transactions on Information Forensics and Security, vol. 9, no. 12, 2014, pp. 2144-2157.

[56] T. A. Letsche and M. W. Berry, "Large-scale information retrieval with latent semantic indexing," Information Sciences—Informatics and Computer Science, Intelligent Systems, Applications: An International Journal vol. 100, no. 1-4, Aug. 1997, pp. 105-137.

[57] G. E. Hinton, "Learning multiple layers of representation," Trends in Cognitive Sciences, vol. 11, no. 10, 2007, pp. 428-434.

[58] R. Salakhutdinov and G. Hinton, "Deep Boltzmann Machines," presented at the Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research, 2009.

[59] W. Kuo, B. Hariharan, and J. Malik, "DeepBox: Learning Objectness with Convolutional Networks," CoRR, vol. abs/1505.02146, 2015.

[60] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," Neurocomputing, vol. 70, no. 1, 2006, pp. 489-501.

[61] J. Tang, C. Deng, and G. B. Huang, "Extreme Learning Machine for Multilayer Perceptron," IEEE Transactions on Neural Networks and Learning Systems, vol. 27, no. 4, 2016, pp. 809-821.

[62] M. Gong, J. Liu, H. Li, Q. Cai, and L. Su, "A Multiobjective Sparse Feature Learning Model for Deep Neural Networks," IEEE Transactions on Neural Networks and Learning Systems, vol. 26, no. 12, 2015, pp. 3263-3277.

[63] S. Mehrkanoon, C. Alzate, R. Mall, R. Langone, and J. A. K. Suykens, "Multiclass Semisupervised Learning Based Upon Kernel Spectral Clustering," IEEE Transactions on Neural Networks and Learning Systems, vol. 26, no. 4, 2015, pp. 720-733.

[64] R. Langone, R. Mall, C. Alzate, and J. A. K. Suykens, "Kernel Spectral Clustering and applications," CoRR, vol. abs/1505.00477, 2015.

[65] A. Conneau, H. Schwenk, L. Barrault, and Y. LeCun, "Very Deep Convolutional Networks for Natural Language Processing," CoRR, vol. abs/1606.01781, 2016.

[66] N. Krpan and D. Jakobovic, "Parallel neural network training with OpenCL," in 2012 Proceedings of the 35th International Convention MIPRO, 2012, pp. 1053-1057.

[67] W. Dong and M. Zhou, "A Supervised Learning and Control Method to Improve Particle Swarm Optimization Algorithms," IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 47, no. 7, 2017, pp. 1135-1148.

[68] V. Vapnik and R. Izmailov, "Learning using privileged information: similarity control and knowledge transfer," J. Mach. Learn. Res., vol. 16, no. 1, 2015, pp. 2023-2049.

[69] H. J. Escalante, M. Montes, and L. E. Sucar, "Particle Swarm Model Selection," J. Mach. Learn. Res., vol. 10, 2009, pp. 405-440.

[70] J. R. Sampson, "Adaptation in Natural and Artificial Systems (John H. Holland)," SIAM Review, vol. 18, no. 3, 1976, pp. 529-530.

[71] N. Mohd Razali, J. Geraghty, Genetic Algorithms Performance with Different Selection Strategy in Solving TSP, 2010.

[72] P. Larrañaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators," Artificial Intelligence Review, vol. 13, no. 2, 1999, pp. 129-170.

[73] D. Whitley, "A genetic algorithm tutorial," Statistics and Computing, vol. 4, no. 2, 1994, pp. 65-85.

[74] C. T. Lin, M. Prasad, and A. Saxena, "An Improved Polynomial Neural Network Classifier Using Real-Coded Genetic Algorithm," IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 45, no. 11, 2015, pp. 1389-1401.

[75] Y. Wu et al., "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," CoRR, vol. abs/1609.08144, 2016.

[76] Z.-H. Zhou, M.-L. Zhang, S.-J. Huang, and Y.-F. Li, "Multi-instance multi-label learning," Artificial Intelligence, vol. 176, no. 1, 2012, pp. 2291-2320.

[77] L. Huang, A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. D. Tygar, "Adversarial machine learning," presented at the Proceedings of the 4th ACM workshop on Security and artificial intelligence, Chicago, Illinois, USA, 2011.

[78] D. Yu and L. Deng, Automatic Speech Recognition: A Deep Learning Approach. Springer, London, 2015.

[79] R. Hadsell, 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, vol. 2, pp. 1735-1742.

[80] A. Shrestha and A. Mahmood, "Enhancing Siamese Networks Training with Importance " in Proceedings of the 11th International Conference on Agents and Artificial Intelligence, Prague, Czech Republic,, 2019, vol. 2: ICAART SciTePress, 2019, pp. 610-615.

[81] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," ArXiv e-prints, 2013.

[82] D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," Nature, Article vol. 529, 2016, p. 484.

[83] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An Introduction to Deep Reinforcement Learning," CoRR, vol. abs/1811.12560, 2018.

[84] I. J. Goodfellow et al., "Generative Adversarial Networks," arXiv e-prints, Accessed on: June 01, 2014, Available: https://ui.adsabs.harvard.edu/\#abs/2014arXiv1406.2661G

[85] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative Adversarial Text to Image Synthesis," arXiv e-prints, Accessed on: May 01, 2016, Available: https://ui.adsabs.harvard.edu/\#abs/2016arXiv160505396R

[86] H. Brighton and C. Mellish, "Advances in Instance Selection for Instance-Based Learning Algorithms," Data Mining and Knowledge Discovery, vol. 6, no. 2, 2002, pp. 153-172.

[87] S. Albelwi and A. Mahmood, "A Framework for Designing the Architectures of Deep Convolutional Neural Networks," Entropy, vol. 19, no. 6, 2017.

[88] G. Reinelt, "TSPLIB—A Traveling Salesman Problem Library," ORSA Journal on Computing, vol. 3, no. 4, 1991, pp. 376-384.

[89] N. M. Razali and J. Geraghty, "Genetic Algorithm Performance with Different Selection Strategies in Solving TSP," presented at the Proceedings of the World Congress on Engineering 2011, London, U.K., July 6-8, 2011, 2011.

[90] L. Davis, "Applying adaptive algorithms to epistatic domains," presented at the Proceedings of the 9th international joint conference on Artificial intelligence - Volume 1, Los Angeles, California, 1985.

[91] D. Whitley, T. Starteather, and D. Shaner, "The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination," in Handbook of Genetic Algorithms, E. L. Davis, Ed., 1991.

[92] S. F. Galán, O. J. Mengshoel, and R. Pinter, "A Novel Mating Approach for Genetic Algorithms," Evolutionary Computation, vol. 21, no. 2, 2012, pp. 197-229.

[93] L. J. Eshelman and J. D. Schaffer, "Preventing premature convergence in genetic algorithms by preventing incest," presented at the In Proceedings of the 4th International Con-ference on Genetic Algorithms (ICGA), San Diego, CA, July, 1991.

[94] G. Ochoa, C. Madler-Kron, and J. Rodriguez R., K. , "Assortative mating in genetic algorithms for dynamic problems, Applications of Evolution-ary Computing," 2005.

[95] I. J. Wilson, M. E. Weale, and D. J. Balding, "Inferences from DNA data: population histories, evolutionary processes and forensic match probabilities," Journal of the Royal Statistical Society: Series A (Statistics in Society), vol. 166, no. 2, 2003, pp. 155-188.

[96] Y. Guo 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, vol. 744, no. 2, 2012, pp. 154-160.

[97] M. Li, A. Schönberg, M. Schaefer, R. Schroeder, I. Nasidze, and M. Stoneking, "Detecting Heteroplasmy from High-Throughput Sequencing of Complete Human Mitochondrial DNA Genomes," The American Journal of Human Genetics, vol. 87, no. 2, 2010, pp. 237-249.

[98] W. Amos and J. Harwood, "Factors affecting levels of genetic diversity in natural populations," Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences, 10.1098/rstb.1998.0200 vol. 353, no. 1366, 1998, p. 177.

[99] E. Hagelberg et al., "Molecular genetic evidence for the human settlement of the Pacific: analysis of mitochondrial DNA, Y chromosome and HLA markers," Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences, 10.1098/rstb.1999.0367 vol. 354, no. 1379, 1999, p. 141.

[100] G. Xavier and B. Yoshua, "Understanding the difficulty of training deep feedforward neural networks," presented at the In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics 2010/03/31, 2010. Available: http://proceedings.mlr.press/v9/glorot10a.html

[101] J. Martens, "Deep learning via Hessian-free optimization," presented at the Proceedings of the 27th International Conference on International Conference on Machine Learning, Haifa, Israel, 2010.

[102] A. Shrestha and A. Mahmood, "Improving Genetic Algorithm with Fine-Tuned Crossover and Scaled Architecture," Journal of Mathematics, 2016, Art. no. 4015845, p. 10.

[103] K. Sastry, D. Goldberg, and G. Kendall, "Genetic Algorithms," 2005.

[104] D. E. Goldberg, The design of innovation: Lessons from and for competent genetic algorithms Springer, Boston, MA, 2013.

[105] R. Miikkulainen et al., "Evolving Deep Neural Networks," CoRR, vol. abs/1703.00548, 2017.

[106] J. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization," J. Mach. Learn. Res., vol. 12, 2011, pp. 2121-2159.

[107] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," CoRR, vol. abs/1412.6980, 2014.

[108] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," CoRR, vol. abs/1502.03167, 2015.

[109] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," J. Mach. Learn. Res., vol. 15, no. 1, 2014, pp. 1929-1958.

[110] M. Muja and D. G. Lowe, "Scalable Nearest Neighbor Algorithms for High Dimensional Data," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 36, no. 11, 2014, pp. 2227-2240.

[111] J. A. Olvera-López, J. A. Carrasco-Ochoa, J. F. Martínez-Trinidad, and J. Kittler, "A review of instance selection methods," Artificial Intelligence Review, vol. 34, no. 2, 2010, pp. 133-143.

[112] X. Sun and P. K. Chan, "An Analysis of Instance Selection for Neural Networks to Improve Training Speed," in 2014 13th International Conference on Machine Learning and Applications, 2014, pp. 288-293.

[113] W. D. Bennette, "Instance selection for model-based classifiers," Graduate Theses and Dissertations, Dissertation vol. 13783, 2014.

[114] I. Guyon, Andr, #233, and Elisseeff, "An introduction to variable and feature selection," J. Mach. Learn. Res., vol. 3, 2003, pp. 1157-1182.

[115] G. James, D. Witten, T. Hastie, and R. Tibshirani, "An Introduction to Statistical Learning," New York, NY : Springer, vol. 103, 2013.

[116] D. A. Forsyth, J. Haddon, and S. Ioffe, "The Joy of Sampling," International Journal of Computer Vision, vol. 41, no. 1, 2001, pp. 109-134.

[117] G. Alain, A. Lamb, C. Sankar, A. Courville, and Y. Bengio, "Variance Reduction in SGD by Distributed Importance Sampling," eprint arXiv:1511.06481, 2015.

[118] R. Chitta, R. Jin, and A. K. Jain, "Stream Clustering: Efficient Kernel-Based Approximation Using Importance Sampling," in 2015 IEEE International Conference on Data Mining Workshop (ICDMW), 2015, pp. 607-614.

[119] F. Shang, K. Zhou, J. Cheng, I. W. Tsang, L. Zhang, and D. Tao, "VR-SGD: A Simple Stochastic Variance Reduction Method for Machine Learning," CoRR, vol. abs/1802.09932, 2018.

[120] A. W. Services. (2018, 7/21/2018). Amazon EC2 P2 & P3 Instances. Available: https://aws.amazon.com/ec2/instance-types/p2/, https://aws.amazon.com/ec2/instance-types/p3/

[121] F. Chollet and others, "Keras," https://keras.io, 2015.

[122] M. Abadi et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," arXiv preprint, 2015.