A Deep Learning Odyssey:

An Invitation for Actuaries to join this journey.

Syed Danish Ali, Rahul Ahuja

Abstract

While Deep Learning has shown remarkable success in the area of unstructured data like image classification, text analysis and speech recognition, there is very little literature on Deep Learning performed on structured/relational data. This investigation also focuses on applying Deep Learning on structured data because actuaries are more comfortable with structured data then unstructured data and so it can be a useful imitative step for other actuaries to apply Deep Learning to solve their own business problems. After extensive investigations, it does seem that Deep Learning has the potential to do well in the area of structured data. More specifically, we review Deep Learning, which is the most promising candidate so far for anomaly detection and for classification. We investigate class imbalance as it is a challenging problem for anomaly detection. In this report, Deep Multilayer Perceptron (MLP) was implemented using Theano in Python and experiments were conducted to explore the effectiveness of hyper-parameters. It was seen that increasing the depth of the neural network helped in detecting minority classes. Cost-sensitive learning technique was also observed to work quite well to deal with the class imbalance. One of the objectives of this project was to see the performance of dropout to regularize and avoid over-fitting in the network. Our conclusion is that while dropout with techniques worked for unstructured data, adding dropout to structured data does not seem to give any improvement. We also situate Deep Learning in its proper strategic context; highlight its applications and limitations as well as on-going developments in Deep Learning.

Keywords: Deep Learning, class imbalance, Multilayer Perceptron, cost-sensitive learning, Dark Knowledge, Big Data

1. Overview of Deep Learning

Up till recent past, the artificial intelligence portion of data science was looked upon cautiously due to its history of booms and flops [1]. In the latest stream of events, major improvements have taken place in this field and now Deep Learning, the new leading front for Artificial Intelligence, presents promising prospect for overcoming problems of big data. Deep Learning is a method of machine learning that undertakes calculations in a layered fashion starting from high level abstractions (vision, language and other Artificial Intelligence related tasks) to more and more specific features [2]. Deep Learning algorithms essentially attempt to model high-level abstractions of the data using architectures composed of multiple non-linear transformations. The machine is able to progressively learn as it digest more and more data and its ability to transform abstract concepts into concrete realities has opened up a diverse plethora of areas where it can be utilized. Deep Learning has various architectures such as Deep Neural Networks, Deep Belief Networks, Deep Boltzmann Machines and so on that are able to handle and decode complex structures that have multiple non-linear features [3].

Deep Learning offers us considerable insight into the relatively unknown unstructured data which is 80% of the data that we generate as per IBM [4]. While traditional data analysis before 2005 focused on just the tip of the iceberg, the big data revolution sprang up and now Deep Learning offers us a better glimpse into the unconscious segment of data that we know exists, but is constrained in realizing its true potential. Deep Learning helps us in both exploring the data and identifying connections in descriptive analytics but these connections also help us in forecasting what the result will likely be, given the particular combination as the machine learns from the data.

Deep Learning has inputs, hidden layers where they are transformed by the weights/biases and output which is achieved through choice of activation function from various functions available (Softmax, sigmoid, hyperbolic tangent, rectified linear, maxout and so on). The weights/biases are learned by feeding training data to the particular Deep Learning architecture. Deep Learning is different from neural networks as it has multiple hidden layers whereas neural network only has one [5].

A de-mystified the foundation of Deep Learning is mostly a way of using backpropagation with gradient descent and a larger number of hidden neural network layers which is certainly not new. However, revival of Deep Learning was possible after 2010 and onwards due to drastically more computational power from GPUs, bigger datasets, and some key algorithm tweaks mainly dropout and AdaGrad to increase accuracy rates. Moreover, the unique feature of Deep Learning is that it allows individual parts of the model to be trained independently of the other parts [6].

Deep Learning models can recognize human faces with over 97% accuracy, as well as recognize arbitrary images and even moving videos. Deep Learning systems now can process

real-time video, interpret them, and provide a natural language description. It is becoming increasingly established that Deep Learning can perform exceptionally well on problems involving perceptual data like speech recognition image classification and text analytics [7].

In a single formula, this is the formula for neural networks (for hyperbolic tangent activation function) [8]

$$p(x) = \beta_0 + \sum_{i=1}^{n_h} \beta_i \tanh\left(\alpha_{i,0} + \sum_{j=1}^{n} \alpha_{i,j} x_j\right).$$

So that essentially, p(x) = linear+ non- linear. This shows that intuitively, Deep Learning is a system of linear+ non-linear equations iterated over and over to the training data and multiple hidden layers and nodes with appropriate activation function to achieve accurate output.

The In a nutshell, the contents of this report can be encapsulated as follows:

1. Overview of Deep learning
2. Applications of deep learning
3. Key tools of big data for transformation: Review & Case Study
4. Deep Learning modeling exercises and experiments
    a. Introduction
    b. Literature Review
    c. Methods
    d. Results
    e. Discussion
5. Ongoing developments and Outlook
6. Limitations of deep learning
7. Strategic notes
8. Conclusion
9. References

2. Applications of Deep Learning

Unstructured data and text mining

It is well known that 80% of data is unstructured. Unstructured data is the messy stuff every quantitative analyst tries to traditionally stay away from. It can include images of accidents, text notes of loss adjusters, social media comments, claim documents and review of medical doctors etc. Unstructured data has massive potential but has never been traditionally considered as a source of insight before. Deep Learning is becoming the method of choice for its exceptional accuracy and capturing capacity for unstructured data.

Text mining utilizes a number of algorithms to make linguistic and contextual sense of the data. The usual techniques are text parsing, tagging, flagging and natural language processing [9]. There is a correlation between unstructured data and text mining as many unstructured data is qualitative free text like loss adjusters' notes, notes in medical claims, underwriters' notes, and critical remarks by claim administration on particular claims and so on. For instance, a sudden surge in homeowners' claims in a particular area might remain a mystery but through text analytics, it can be seen that they are due to rapid growth in mold in those areas. Another useful instance is utilizing text analytics when lines have little data or are newly introduced which is becoming more and more necessary for our fast moving society with new emerging risks arriving before the previous emerging risks even ossify [10].

Sentiment analysis/opinion mining over expert judgment on level of uncertainty in actuarial models like pricing, capital modeling and reserves can also prove fruitful. Natural Language Processing (such as in Stanford 'CoreNLP' software available free for download [11]) is a powerful source of making sense out of the texts.

Aside from exposures, the other side of ratemaking is losses and loss trends. By building Deep Learning models we can analyse images to estimate repair costs. Also Deep Learning techniques can be applied to automatically categorize the severity of damage to vehicles involved in accidents. This will more quickly update with us more accurate severity data for modelling pure premiums [12].

Claim professionals often have more difficulty in assessing loss values associated with claims that are commonly referred to as "creeping Cats." [13]

These losses typically involve minor soft tissue injuries, which are reserved and handled as such. Initially, these soft tissue claims are viewed as routine. Over time, however, they develop negatively. For example, return-to-work dates get pushed back, stronger pain medication is prescribed, and surgery may take place down the road. Losses initially reserved at $8,000–$10,000 then become claims costing $200,000–$300,000 or more. Since these claims may develop over an extended time period, they can be difficult to identify. Creeping cat is a big problem for emerging liabilities because mostly, we do not fully know what we are dealing with. Emerging risks like cyber-attacks, terrorism etc have shown to have huge creeping cat potential.

As discussed, Deep Learning models can review simulated or real (as per availability) claim data from agent based modeling, network theory for similarities and other factors shared by such losses, thereby alerting the claims professional to emerging risks that may have creeping Cat potential. With this information, strategies and resources can be applied at a point in time where they can be most effective in an effort to achieve the best possible outcome and control cost escalation. Additional loading on premiums can also be given on areas with higher Creeping Cat potential.

Another powerful application is fraud analytics especially since the advent of cyber insurance. With big data, insurers are collecting more and more sensitive and personal data on customers. Also they have started selling cyber insurance where the insurance company covers the losses arising from data theft, hacking, internal sabotage etc to the company's IT systems. With advent of Cyber insurance, fraud and intrusion has been explicitly commoditized as a risk factor. Fraud analytics are crucial for insurance company more so in the present than ever before, because while insuring other companies for cyber risk, it is paramount that the insurance company itself should have top-notch fraud analytics software so to ensure breaches of its privacy, data security are at the bar most minimum, given that more and more data is stored by insurance companies on their clients as well as frauds in claims and other internally caused frauds.

It would be most embarrassing and costly indeed, if the insurance company insuring other companies itself for cyber insurance becomes hacked or gets compromised. Aside from these reputational damages, the lack of extremely sensitive and personal data on the policyholders can spark class action liability damages action in legal courts as well.

With Deep Learning doctors can for the first time use the predictive power of Deep Learning to directly improve patients' medical outcomes. Deep Learning can readily handle a broad spectrum of diseases in the entire body, and all imaging modalities (x-rays, CT scans, etc). Deep Learning contextualizes the imaging data by comparing it to large datasets of past images, and by analyzing ancillary clinical data, including clinical reports and laboratory studies.

In initial benchmarking test against the publicly-available LIDC dataset, Enlitic technology, the startup utilizing Deep Learning for improving healthcare, detected lung cancer nodules in chest CT images 50% more accurately than an expert panel of radiologists. In initial benchmarking tests, Enlitic's Deep Learning tool regularly detected tiny fractures as small as 0.01% of the total x-ray image Enlitic's Deep Learning tool is designed to simultaneously support hundreds of diseases (not just a limited specialization of diseases or one disease) [14].

Another key combination is Deep Learning's integration and synergy with Big Data. Actuaries will have to understand and appreciate the growing use of big data and the potential disruptive impacts on the insurance industry. Actuaries will also need to become more proficient with the underlying technology and tools required to use big data in business processes [15]. Subsequently, this is the effort of the next section.

Deep Learning, in collaboration with other machine learning tools is make headways in possible applications. All major giants like Google, IBM, Baidu are aggressively expanding in this direction but startups are providing the most vivid applications so far. Kensho is a startup that aims to use software to perform tasks in minutes that would take analysts weeks or months. Just like searching via Google, the analyst can write their questions in the Kensho's search engine. The cloud based software, as per Forbes reporter Steven Bertoni, can find targeted answers to more than 65 million combination in the flick of a second by scanning over 90,000 actions which are as myriad as political events, new laws, economic reports, approval of drugs etc and their impact on nearly any financial instrument in the world. Another startup, Ufora is set to automate a large part of quantitative finance work undertaken by quants, especially on the stochastic modeling front. Even some hedge funds like Renaissance Technologies are proactively working on machine learning and Deep Learning algorithms to better see patterns in the financial data to exploit opportunities (which stocks are overrated or underrated, market is going strong on fundamentals or approaching the bubble stage and so on) to guide their investment strategies[17].

On the other hand, Firms like Narrative Science and Automated Insights working on text analytics are utilizing Deep Learning to create lively and interactive narrative reports out of data and numbers. This essentially means report written by a machine that reads like it is almost written by a human author. To elaborate this feature, Narrative Science' s Quill platform undertakes statistical analysis of applying time series, regression etc and then the semantic engine evaluates the important data signal from the unimportant noise as per the needs of the particular audience in question like different reasoning if it is related for a quant or a trader of investments. The patterns are spotted and made sense out of in a holistic manner. Particular fuzzy attention is given to anomalies and elements of results that are deviant from the main normal body of the results to ascertain their impact and proper interpretation. It remembers previous reports made so it doesn't become repetitive. Natural Language Generation is applied with a surgeon's precision and expertise in forming such a dynamic semantic engine.

This is indeed a leap forward as report writing consumes a lot of human time and efforts and because machines making such reports were before unheard of practically. Deep Learning allows us not just to explore and understand the data better, but also to perform forecasts better. For predictive analytics part, the startup MetaMind is working to help financial firms assess chances of selling of stocks by going through corporate financial disclosures [18]. It identifies from previous experiences when a particular combination of actions lead to a particular result to assess chances of the same result happening in the future.

Extrapolating this trend into the future, it is our opinion that such analytics might soon find their way into Mergers & Acquisitions (M & A) and will be able to come up with probability of some key event happening and the consequences of it when involved in a high stake M & A. Another application can be to apply Deep Learning applications to help us for one of the most vexing problems, i.e, financial crises. Economists, financial experts and social scientists have elaborated on a lot of key issues that lead to financial crises in general as well as specifically for a particular meltdown. These can form the modeling methodology for the

Deep Learning machine to analyze the cosmic scale of data available on any and every platform that it can garner. Such evaluation can perhaps help us to see patterns that we could have missed otherwise as well as to allow us to understand more accurately the sequential movements and mechanisms involved in a particular financial contagion and crisis. There is no guarantee that this will work. But perhaps it can shed some light inside the 'quantum black box' of financial crises. This seems to be the need of the hour with recurring financial hemorrhages such as EU crisis on Greek Debt and Brexit as well as the recent massive and escalating falls in Chinese stock exchanges reminding us of the bitter past we faced in Wall Street Crisis of 2008-09.

Given all these developments, there are still a myriad of issues that need clarification with not just Deep Learning in specific but also with big data generally. Automation of such unprecedented scale and intensity raises the possibility of mass redundancies in labor force across the economy. Are we comfortable with giving up our controls to such applications without knowing the full implications of such a move? Not every innovation brings positive results or sustains in the long run. Technology is progressing rapidly at an unstoppable pace but can we manage the social consequences and make it sustainable in the long term? Human efforts are seemingly being diverted from other fields into IT which consequently can imply a concentration of power in one overlord field to the potential detriment of others. Are we ready for this? From a consumer point of view how ethical is it that marketing personnel know you so well that it makes rational optimization very difficult on the part of the consumer?

These are all good questions and should be adequately and mutually tackled and addressed by all the stakeholders involved such as the data scientists, government, professions and consumers so as to be able to reach a mutual policy that can better alleviate such concerns. The core aim of the policy has to be to sustain technology for the benefit of our societies, to lead to value creation, to reduce scarcity and reduce fragility of our systems as well as to generate more resources for our prosperity instead of creating the monster of Frankenstein, as Terminator and other doomsday movies will have us believe.

3. Key tools of Big Data for Transformation: Review & Case Study

Review

The challenges of big data can be captured succinctly as follows [19][20]:

- Volume; ever increasing volume which breaks down traditional data-holding capacity
- Variety; more and more heterogeneous data from many formats and types are bombarding the data environment
- Velocity; more and more data is time sensitive now; frequent updates are taking place instead of relying on historical old data and data in real time is being generated now by the internet of things, amongst others.
- Veracity; how valid and reliable is the data? Since now we have so much data, any point of view can be supported by selective adaption of data.

For volume, Map Reduce [21] works to harness the potential of billions of items of data. The first part is that the data is mapped down into key and value pairs; the reduce job combines the mapped data into smaller set of data by eliminating repetition and redundancy amongst others. Hadoop is open-source for handling big data, applying MapReduce and a variety of other distribution systems and clusters. and there are variants produced by many different vendors such as Cloudera, Hortonworks, MapR and Amazon. There also other products such HPCC and cloud-based services such as Google BigQuery.

For variety, NoSQL (Not-Only SQL) [22] is a new way of handling variety of data. Relational databases use rows and columns in handling data but NoSQL uses a number of other components such as giving unique key or hash tagging to every item in the data to every item in the data. Companies utilize NoSQL because it captures so many elements of supply chain that were previously only based on experience or hunches (Techterms, 2013). MongoDB – an open-source NoSQL database and other instances of NoSQL are Cassandra, Couchbase, Apache River etc. Insurance companies can utilize NoSQL databases like MongoDB, Cloudera and Hadoop for feeding into Deep Learning because it captures so many elements that were deemed belonging to the domain of uncertainty before as they were too messy, unstructured and qualitative [23].

NewSQL is the latest addition to the database technology. It provides the same scalable performance and ability to handle unstructured data as NoSQL, while still maintaining the ACID (Atomicity, Consistency, Isolation, Durability) guarantees of the traditional SQL databases. NewSQL provides the power of NoSQL but in the SQL interface and relational data model that analysts are used to [24].

For velocity, Complex even processing or stream processing allows us to handle the velocity of time; real time generated by countless sensors involved in every bit and inch of the supply chain process can be automatically fed into stream processing which uses defined algorithms to analyze it almost instantly. Early alarm systems of supply chain would find this invaluable

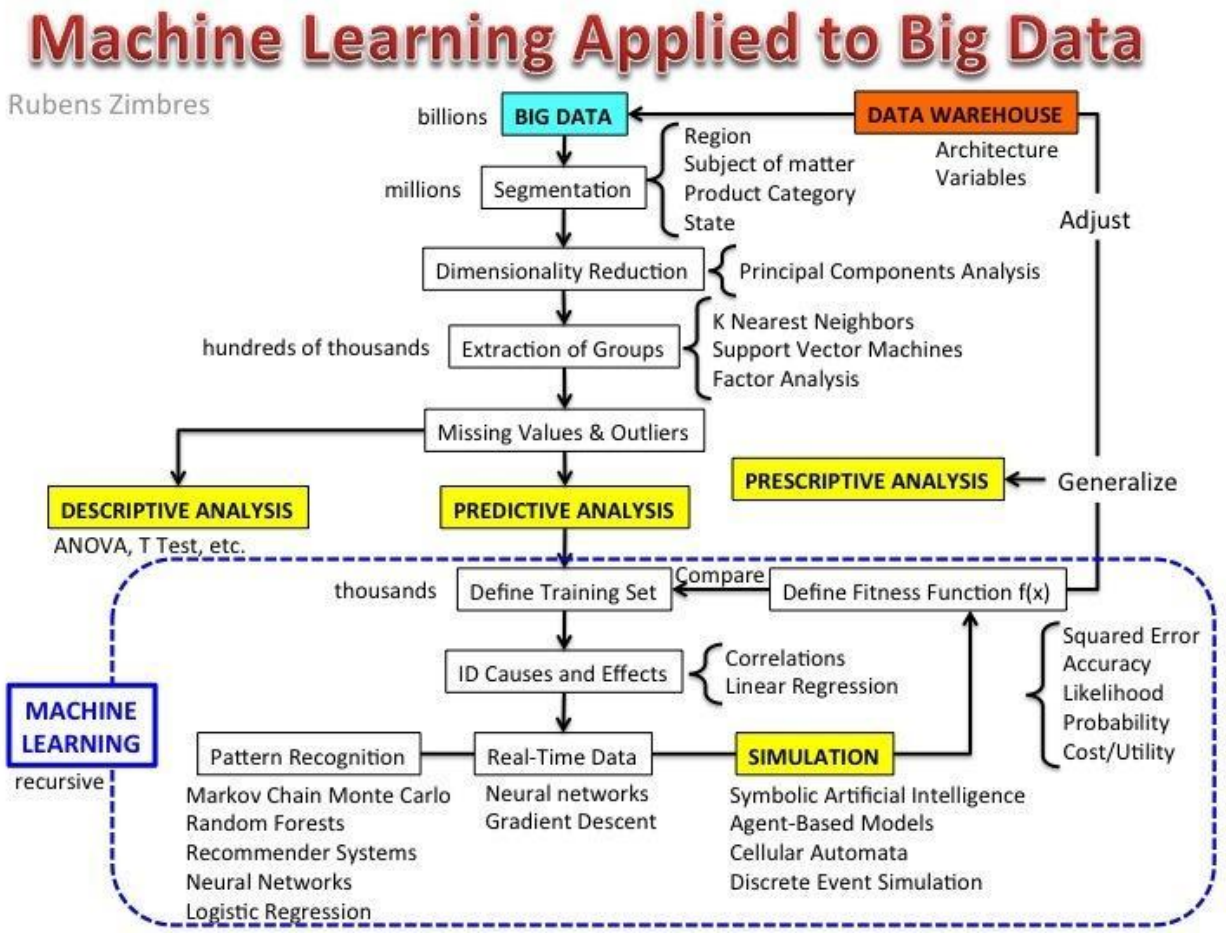
because red alert can be issued to company from the supply chain right when it occurs instead of giving the red alert after a sufficient time has elapsed which has given the disruption time to reckon havoc for the business. Apache spark has also been developed which is found to be around 100 times faster than hadoop for MapReduce purposes. Storm is an open-source distributed computation system designed for processing multiple data streams in real time.

For veracity, Machine learning and data mining constitute a number of models taken from mathematics, statistics and artificial intelligence that make sense of the big data as well as ensure reliability of results associated with the data so that the skepticism brought by veracity for the management can be sufficiently diminished. Also for veracity, apache mahout, machine learning in R and python are particularly useful platforms for filtering of data as well as clustering and classification of data into model points which greatly reduces having to focus on redundant data. Powerful visualizations have also covered ground to make intuitive human sense of the plethora of data and results available to us. These types of visualizations are far more flexible, representative and diverse than the usual spreadsheet based visualization tools. Instances of powerful visualization can be made through tableau, data driven documents, google interactive maps , Ggplot, Shiny and R Markdown of R and so on.

Moreover, behavioral finance has highlighted a number of ground breaking cognitive biases that human falls into; such as recency effect of seeing only the recent efforts and generalizing them to the long term anchoring which is relying on expert opinion or the primacy effect which is giving undue importance to first impressions. Anchoring has a special context in organizations which is relying unduly on the opinion of highly paid and powerful leaders in the company. Big data can clear up the clutter of such cognitive biases by introducing data-driven decision making into the company.

Even with robust sensitive checks in place to counter over-fitting, there is still a challenge around the concepts Game Theory and the Butterfly Effect of Chaos theory and their application in insurance analytics. Modelers need to at least partly take into account these reactions within strategies to maximize the benefit and minimize any potential damages. This includes more frequently updating analysis than before to assess if there are any discrepancies between model anticipation and reality. Assessment ideally needs to incorporate horizon scanning techniques and emerging risk assessment, which can be in-built into assumptions for stochastic analysis or the robustness of sensitivity analysis [25].

An excellent illustration by Rubens Zimbres captures the synergistic interaction between machine learning and big data and is shown as a mindmap as follows:



Key thing to note here is how big data goes from billions to less and less data which is more and more refined and useful. Machine learning is then applied onto the data to arrive at actionable insights. Key models of simulation mentioned here belong to the complexity science domain detailed before. As such, this mindmap presents a powerful consolidation of many algorithms by showing relationships and links between them, as these on their own, can seem confusing and dispersed.

Until now, cluster management products have been mainly focused on the upper layers of the cluster (e.g., Hadoop products, including the Hadoop Distributed File System [HDFS], MapReduce, Pig, Hive, HBase, and Zookeeper). The installation and maintenance of the underlying server cluster is handled by other solutions. Thus the overall Hadoop infrastructure is deployed and managed by a collection of disparate products, policies, and procedures, which can potentially lead to unpredictable and unreliable clusters [26].

Insurance providers are looking beyond algorithmic ratemaking techniques that are claim-centric, to ones that are person–centric. These techniques focus on analysing policyholder behavior across claims, providers, and other sources of information (e.g. how many similar claims were submitted by the same individual, reported by the same individual), and extend to data sources beyond the firewall to analytics based on external information (e.g. cohort

analysis - using a person's social graph to look for similar activities among connected individuals), and considering networks of people rather than just individuals [27].

This person-centric approach (very aptly harnessed by Deep Learning) requires integrating information across all providers involved in a claim, including counter-parties as well as partners (e.g. auto repair shops) requiring the schema-agnostic approach to data management mentioned earlier. Even when all the data lives within the firm, the agility provided by this approach makes it much more feasible to turn that data into useable information [28]. Telematics is the leading instance of ratemaking using such person-centric approach. In Kaggle Contest, there was even anonymized telematics data and the goal was to identify the unique driver. Moving vehicles are poised to become driverless cars which will bring its own set of challenges. Even currently, the Tesla Model S car can become a boat as well in times of emergency for short times, blurring the traditional line between Motor and Marine line of business [29].

Moving these data collection policies and the uses of this data from the subconscious to our consciousness is a first step in the process of potentially applying big data in a business context. The use of big data and analytics has rapidly evolved from a back-room niche to a strategic core competency.

Big Data Application Case Study [30]

This section highlights big data application on handling liability catastrophes which does not let insurers have a good night's sleep. The big data architecture combined with Deep Learning algorithms will be used here to capture, if not solve, the huge problem of Liability catastrophes.

Liability catastrophes are especially rare, high impact trends that are very difficult to know in advance. Even when the evidence starts becoming more and more corroborative and certain, the insurers may not collect adequate premium as actuarial modeling using historical claims data for ratemaking. This traditional way of ratemaking on historical data can mean that insurers are potentially underwriting their own graves and selling products features that will become their own gravediggers. Asbestos is the most famous liability catastrophe that has resulted in USD 85 billion in claims in US alone and bankruptcies in 73 insurers until 2004. Consequences of liability catastrophes cab include bodily injury, property damage or environmental damage Commercial general liability insurance covers such liability catastrophes usually.

To improve our chances of collecting adequate premium so that insurers do not go bankrupt when a new liability catastrophe arises, big data tools with Deep Learning and machine learning algorithms focused around emerging risk approach is being utilized. Framework of emerging risk is more fruitful for liability catastrophes because the impact of new technologies can be much more complex than expected, and it can take many years before the broader consequences are brought onto the surface. When a liability catastrophe does occur, the products or business practices involved are usually discontinued and the companies usually bankrupt. Hence, each new liability catastrophe is likely to happen in a different

industry and in a different way. Claims data therefore characteristically cannot be used to predict the next liability catastrophe, and this presents an open challenge for actuarial modeling.

Although the world has transformed since asbestos litigation emerged, the interplay between science, technology-driven-innovation and risk which can drive the accumulation of exposure has certainly not budged. Three prominent examples of emerging risks for current times are mobile telephones causing brain tumors, hydraulic fracturing causing earthquakes and nano-materials cause lung damage, range of plastics causing endocrinal damage causing autism and obesity and so on.

The main application is text mining in a big data environment, for which Deep Learning works best. Text parsing, sentiment analysis, opinion mining and natural language processing using Deep Learning algorithm can collate, identify, catalogue and track the massive universe of resources available on the internet in a data driven manner to identify key emerging risks. While the risks are diverse and too many, the means through which scientists establish causation are common to all of them. Each new hypothesis published in the scientific literature sits somewhere along the road to establishing causation, and it is possible from the algorithms mentioned here to estimate whether causation will ever be established based on the current pace at which the research is progressing.

Moreover, the data mining as well as exploratory analysis with Deep Learning algorithms of scientific literature can map these emerging risks with a particular insurer given its unique portfolio which can be deduced from public sources as well now. Some companies will be located on none of the emerging risks, and some will be located on many of them – and the map itself will vary according to what is on each insurer's list of emerging risks. Assembling companies into a portfolio and then adding up the risks across companies provides a possible method to measuring portfolio accumulation for a particular emerging risk.

These trends, mappings and probabilities can then be combined with quantitative estimates of mass litigations if these emerging risks were to occur. This show expected costs and these can vary across different industries, companies, regions and portfolios. This can serve as a vital guide for ratemaking for liability catastrophes and inform insurer on a number of key areas such as product specifications, list of exclusions, maximum sum insured, reinsurance arrangements, loading on premiums for liability catastrophes and so on. With historical analysis, insurers rely on scientific sceptics' approach where they are not convinced of liability catastrophes until they actually surface. This is a self-defeating approach and focusing instead on pragmatism and precautionary principle can serve the solvency interests of the insurers better. Big data tools with Deep Learning algorithms in an emerging risk framework can better aid risk-adjusted ratemaking of emerging liability catastrophes and apply the precautionary principle to work here.

4. Deep Learning modelling exercises and experiments

4.1 Introduction

4.1.1 Dataset used

A well-known and deeply studied dataset was chosen for this work to focus on understanding and implementing Deep Learning techniques rather than data pre-preparation. This data set does come with its fair share of warning though; it must not be used for IT security and intruder detection by IT security experts but there is no harm in using it to show a concept like we do here by Deep Learning classification.

4.1.2 Introduction to KDD Cup 1999

Knowledge Discovery and Data Mining (KDD) [31] is an international platform that organizes data mining competitions among academics researchers, and commercial entities. In 1999 there was a KDD cup competition related to intrusion detection. Since then KDD cup 1999 has become the most widely used dataset for the evaluation of intrusion detection system that detects intruding attacks seen as anomalies [32]. In this project KDD Cup 1999 dataset is used to build a Deep Learning model that can distinguish between and classify good connections and bad connections. The attacks fall into four main classes [33]:

1. Denial of Service (DoS) is a type of attack that ties up computing or memory resources such that the service cannot serve authorized requests rejecting entry to a machine.
2. User to Root attacks (U2R) is an exploitation in which an attempt is made to exploit susceptibility in the system to achieve user entitlements by using normal user accounts.
3. Probing is an attack in which the intruder examines a machine or networking device to determine susceptibility that can be exploited to compromise the system.
4. Remote to User (R2L) is an attack in which an intruder sends packets to a machine over the network to expose the machines susceptibilities and achieve local user entitlements.

10% of the KDD Cup 1999 dataset [34] consists of around 0.5 million samples in the training set and around 0.3 million samples in the test set. The distribution of the training set and test set is different. It is because there are some new attacks included in the test set that are not included in the training set making the problem challenging.

There are hundreds of papers available applying various machine learning algorithms on KDD Cup 1999 data. In [35] a Deep Belief Net (DBN) pre-trained by three or more layers by Restricted Boltzmann Machine (RBM) proved to perform better than the Multilayer Perceptron (MLP) with one hidden layer and a support vector machine. In another paper [36] twenty classifiers were tested on the KDD intrusion data set achieving prediction performance in the range of 45.67% to 92.81% with random forest classifier achieving the best results. This shows that there is huge interest in this classification problem.

4.1.3 Class Imbalance in KDD Cup 1999 Data Set

In the machine learning literature, it has been pointed out [37] that little work has been done in the area of classification by machine learning when there is a highly skewed distribution of the class labels in the data set. In many cases a classifier tends to be biased towards the majority class resulting in poor classification rates on minority classes. As we can see in the training and test class distribution of the KDD cup 1999 data U2R and R2L attacks constitute 0.24% of the training dataset but these attacks take up 5.27% in the test data [38].

Below [39] is the class distribution table of KDD Cup 1999:

Class	Training	Test
1	19.69%	19.48%
2	0.83%	1.34%
3	79.24%	73.90%
4	0.01%	0.07%
5	0.23%	5.20%

Below is the class distribution (figure 4.1.3.1a) of the training set.

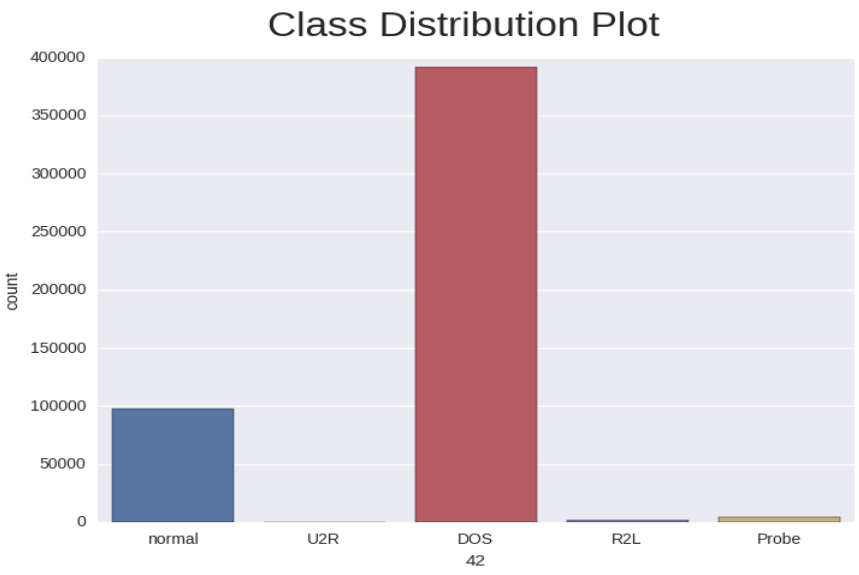


Figure 4.1.3.1a

Below is the class distribution (figure 4.1.3.1b) of the test set.

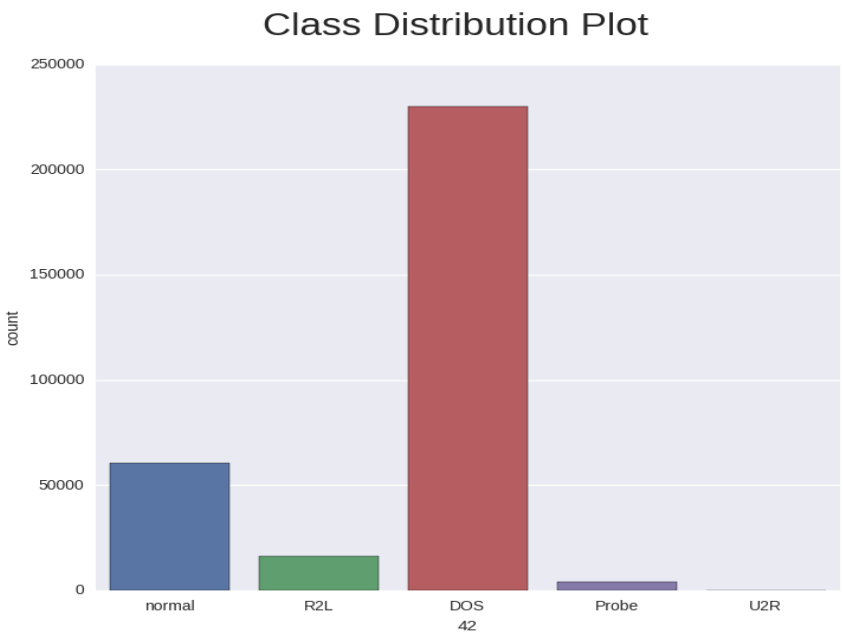


Figure 4.1.3.1b

It is important to note that the test data is not from the same probability distribution as the training data, and it includes specific attack types not in the training data. This makes the task more realistic. Some intrusion experts believe that most novel attacks are variants of known attacks and the "signature" of known attacks can be sufficient to catch novel variants. The datasets contain a total of 24 training attack types, with an additional 14 types in the test data only.

4.1.4 Results from the winning entry

The winning entry of the KDD Cup 1999 [40] is set as the benchmark for the project's experimental results of KDD Cup 1999. The winning entry was submitted by Dr. Bernhard Pfahringer of the Austrian Research Institute for Artificial Intelligence using C5.0 decision tree classifier [41] giving the benchmark for the comparison of our proposed machine learning algorithm. The winning entry achieved an average cost of 0.2331 per test example with the following confusion matrix:

predicted	0	1	2	3	4	% Correct
Actual						
0	60,262	243	78	4	6	99.5%
1	511	3,471	184	0	0	83.3%
2	5,299	1,328	223,226	0	0	97.1%
3	168	20	0	30	10	13.2%
4	14,527	294	0	8	1,360	8.4%
% Correct	74.6%	64.8%	99.9%	71.4%	98.8%	

The last row represents the recall rate and the last column represents the precision. The main issue is that the recall rate of 8.4% for the last class by the winning entry is quite low. The winning entry's classification technique is C5.0 Decision trees a mixture of boosting and bagging, taking into account the minimization of the so-called conditional risk which is a similar approach as cost-sensitivity (introduced later). Bagging decision trees are random forests that represent an ensemble of decision trees averaged while training on different parts of the training set by sampling with replacement. Boosting is an iterative procedure used to adaptively vary the training set's distribution in order for the base classifiers to focus on examples that are hard to classify. In boosting, weights are assigned to the data in such a way that examples that are misclassified gain weight and examples that are classified correctly lose weight [42]. Below is the cost matrix that will indicate of the cost of misclassifying between the various class labels.

	normal	probe	DOS	U2R	R2L
normal	0	1	2	2	2
probe	1	0	2	2	2
DOS	2	1	0	2	2
U2R	3	2	2	0	2
R2L	4	2	2	2	0

Source [43]

4.1.5 Evaluation Metrics

There are number of ways to evaluate the performance of a classifier:

Recall is the fraction of relevant instances that are classified.

Precision is the fraction of classified instances that are relevant.

$$Precision = \frac{tp}{tp + fp}$$

$$Recall = \frac{tp}{tp + fn}$$

where tp: true positive which is the number of classes correctly predicted as belonging to the positive class

fp*:* false positive which is the number of classes incorrectly predicted as belonging to the positive class

fn: false negative which is the number of classes which were not predicted as belonging to the positive class but should have been.

Confusion matrix also known as contingency table is a table with rows and columns that reports the true positive, false positives, false negatives and true negatives as depicted in figure 1.3.4a.

		Predicted Class	
		Positive Class	Negative Class
Actual Class	Positive Class	True Positive (TP)	False Negative (FN)
	Negative Class	False Positive (FP)	True Negative (TN)

Figure 4.1.5.a

F-1 score [44] is the weighted average of the precision and recall that lies between 0 and 1. If the equal weighting is given to the precision and recall then the following formula is used;

$$F_\beta = (1 + \beta^2) \cdot \frac{precision * recall}{(\beta^2 * precision) + recall}$$

where β corresponds to the relative importance of precision over recall. Instead of F1-score the average cost per test example is considered as an evaluation metric for the overall performance of the various learning algorithm techniques throughout this report.

When it comes to evaluating the performance of the classifier, it is better to rely on the precision, recall rates and F1-scores [45] rather than accuracy levels. Let's say we have a data set with a (0.9, 0.1) class distribution, there is the possibility that a classifier predicts

everything as major class and ignores the minor class. In that case we get an accuracy level of 90% turning out to be poor indicator of the performance of the classifier. There are better measures of performance of the techniques like the confusion matrix, recall (sensitivity), precision and F1-score.

4.2. Literature Review

Deep Learning is based on well-known neural network techniques which will be described before more recent advances from the emerging field of Deep Learning.

4.2.1 History of Neural Networks

It is useful to know a brief history of neural networks as they are the building blocks of today's technological breakthrough in the area of Deep Learning. A neural network can be seen as simple processing units that are massively parallel that can store knowledge and apply this knowledge to make predictions [46]. Neural Network mimics the brain in the way the network acquires knowledge from its environment through a learning process and intervention connection strengths known as synaptic weights are used to store the acquired knowledge. In the learning process the synaptic weights of the network are modified in an orderly fashion to attain the desired objective. In 1950 the neuropsychologist Karl Lashley's thesis was published in which he described the brain as a distributed system [47].

Another reason that the neural network is compared with the human brain is that they operate like non-linear parallel information-processing systems that can rapidly perform certain computations such as pattern recognition and perception [48]. As a result these networks perform very well in areas like speech, audio and image recognitions where the inputs/signals are inherently nonlinear.

McCulloch and Pitts [49] were pioneers of neural networks who wrote a research article on the model with two inputs and single output in 1943. The following are the features of that model:

A neuron would only be activated if:

- one of the inputs is active
- The weights for each input are equal
- The output of the model is binary

There is a certain threshold level computed by summing up the input values for which the output is either zero or one.

In Hebb's 1949 book 'The Organization of Behaviour' [50] the idea that the connectivity of the brain is continuously changing in response to changes in tasks was proposed for the first time. The rule implies that the connection between two neurons is active at the same time. This soon became the source of inspiration for the development of computational models of learning and adaptive systems. Artificial neural networks have the ability to learn based on

supplied data, known as adaptive learning, while the ability of a neural network to create its own organization or representation of information is known as self-organisation.

After 15 years the perceptron developed by Rosenblatt in 1958 [51] emerged as the next model of the neuron. Perceptron is the simplest neural network that linearly separates the data into two classes. He randomly interconnected the perceptron and used a trial and error method to change the weights for the learning.

After 1969 the research came to a dead end in this area for the next 15 years after the mathematicians Marvin Minsky and Seymour Parpert published [52] a mathematical analysis of the perceptron and found that the perceptron was not capable of representing many important problems, like the exclusive-or function (XOR). Secondly there was an issue that the computers did not have enough processing power to effectively handle large neural networks.

In 1986 the development of the back-propagation algorithm was reported by Rumelhart, Hinton, and Williams [53] that can solve problems like XOR, beginning a second generation of neural networks. In that same year, the celebrated two-volume book, Parallel Distributed Processing: Explorations in the Microstructures of Cognition, edited by Rumelhart and McClelland [54], was published. That book has been a major influence in the use of back-propagation, which has emerged as the most popular learning algorithm for the training of multilayer perceptrons.

4.2.2 Multilayer Perceptron (MLP)

A multilayer perceptron (MLP) has one or more hidden layers along with the input and output layers, each layer contains several neurons that interconnect with each other by weight links. The number of neurons in the input layer will be the number of attributes in the dataset, neurons in the output layer will be the number of classes given in the dataset.

4.2.2.1 Architecture of Multilayer Perceptron

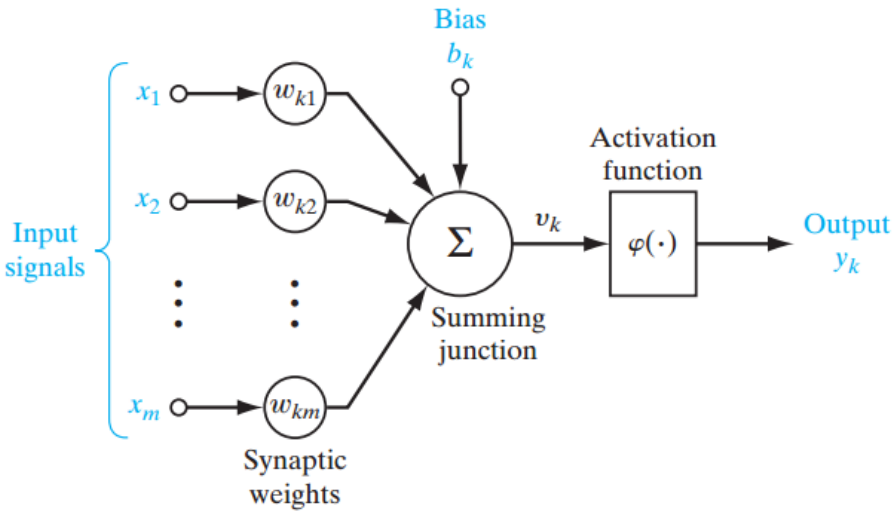


Figure 4.2.2.1a [Source 55]

In the figure 4.2.2.1a, W_{km} represents the m weights that are seen as a set of synapses or connecting links between one layer and another layer within the network. This parameter indicates how important each feature (x_j) is. Below is the adder function of features of the input multiplied by their respective synaptic connection.

$$u_k = \sum_{j=1}^{m} w_{kj} x_j$$

The bias, b_k, acts as an affine transformation to the output of the adder function u_k giving v_k the induced local field.

$$v_k = u_k + b_k$$

4.2.2.2 Initialisation of the parameters

Initialisation of the parameters, weights and biases plays an important role in determining the final model. There is lot of literature on initialisation strategy [56]. A good random initialisation strategy can avoid getting stuck at local minima. Local minima problem is when the network gets stuck in the error surface and does not go down while training although there is still capacity for learning. However experiment by using various initialisation strategies is out of the scope of this project. The initialisation strategy should be selected according to the activation function used. For $tanh$ the initialisation interval should be $\left[-\sqrt{\frac{6}{fan_{in}+fan_{out}}}, \sqrt{\frac{6}{fan_{in}+fan_{out}}}\right]$ where fan_{in} is the number of units in the (i-1)-th layer, and fan_{out} is the number of units in the ith layer. Similarly for the sigmoid activation

function the initialisation interval should be $\left[-4\sqrt{\frac{6}{fan_{in}+fan_{out}}}, 4\sqrt{\frac{6}{fan_{in}+fan_{out}}}\right]$. These initialization strategies ensure that information propagated upwards and backwards in the network at the early stage of training.

4.2.2.3 Activation function

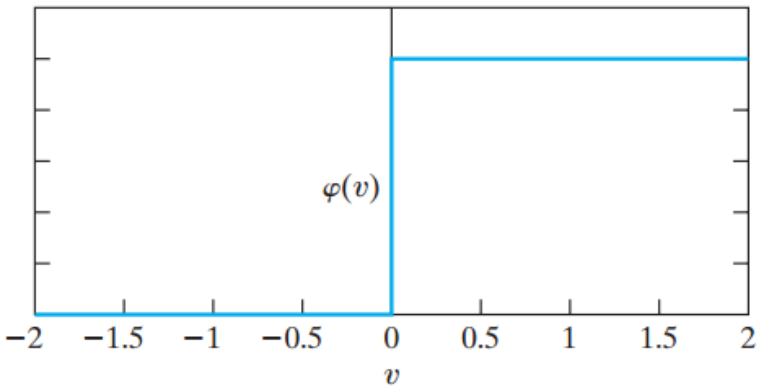
The activation function defines the output of a neuron in terms of the induced local field v.

$$y_k = \varphi(u_k + b_k)$$

where $\varphi(.)$ is the activation function. There are various types of activation functions, the following are the commonly used ones;

1. Threshold function

$$\varphi(v) = \begin{cases} 1 & if\ v \geq 0 \\ 0 & if\ v < 0 \end{cases}$$



It indicates that either the neuron is fully active or not. However this function is not differentiable which is quite vital when using the backpropagation algorithm (explained later).

2. Sigmoid function

The sigmoid function is a logistic function bounded by 0 and 1, as with the threshold function, but this activation function is continuous and differentiable.

$$\varphi(v) = \frac{1}{1 + exp^{(-av)}}$$

where a is the slope parameter of the above function. Moreover it is nonlinear that helps to increase the performance [57] making sure that small changes in the weights and bias causes small changes in the output of the neuron [58].

3. Hyperbolic tangent function

$$\varphi(v) = \tanh(v)$$

This function enables activation functions to range from -1 to +1.

4. Rectified Linear Activation function (ReLU)

ReLUs are the smooth approximation to the the sum of many logistic units and produce sparse activity vectors [59]. Below is the equation of the function;

$$y = \max\left\{0, b + \sum_{i=1}^{k} x_i * w_i\right\}$$

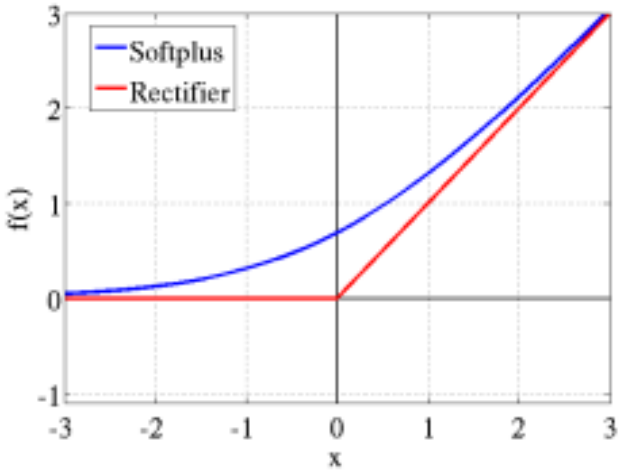


Figure 4.2.2.3a (softplus $f(x) = log(1 + e^x)$ is the smooth approximation to the rectifier) Source [60]

5. Maxout function

In 2013 Goodfellow [61] finds out the Maxout network using a new activation function is a natural companion to dropout (introduced later)[62]. Maxout units facilitate optimization by dropout and improve the accuracy of dropout's fast approximate model averaging technique. A single Maxout unit can be interpreted as making a piece wise linear approximation to an arbitrary convex function. Maxout networks learn not just the relationship between hidden units, but also the activation function of each hidden unit. See Below is the graphical depiction of how this works.

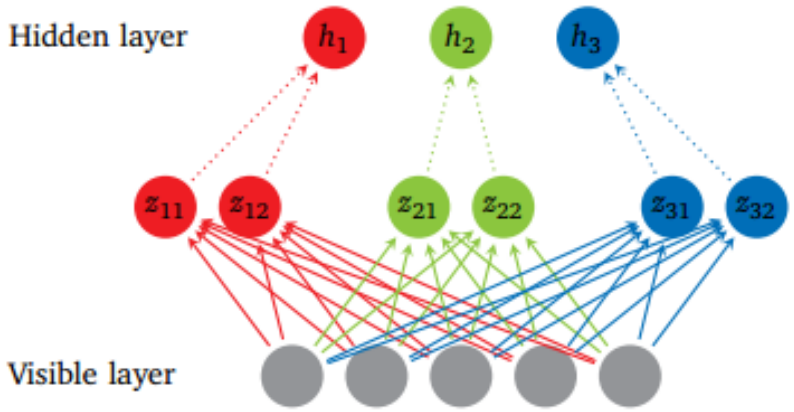


Figure 4.2.2.3b, the Maxout network with 5 visible units, 3 hidden units and 2 pieces for each hidden unit. (Source [63]).

$$h_i = max_{j \in [1,k]} z_{ij}$$

$$z_{ij} = \sum_m x_m W_{mij} + b_{ij} = xW_{\dots ij} + b_{ij}$$

where $W_{\dots ij}$ is the mean vector of size of the input obtained by accessing the matrix $W \in \mathbb{R}^{m \times n \times k}$ at the second coordinate i and third coordinate j. The number of intermediate units (k) is called the number of pieces used by the Maxout nets.

4.2.2.4 Backpropagation Algorithm

The backpropagation algorithm can be used to train feedforward neural networks or multilayer perceptrons. It is a method to minimise the cost function by changing weights and biases in the network. To learn and make better predictions a number of epochs (training cycles) are executed where the error determined by the cost function is backward propagated by gradient descent until a sufficiently small error is achieved.

4.2.2.5 Gradient descent

1. Mini-Batch Gradient Descent

Let's say in 100-sized mini-batch, 100 training examples are shown to the learning algorithm, and then weights are updated. After all mini-batches are presented sequentially, then average of the accuracy levels and training cost levels are calculated for each epoch. [64].

2. Stochastic Gradient Descent

Stochastic gradient descent is used in the real-time on-line processing, where the parameters are updated while presenting only one training example, and so average of accuracy levels and training costs are taken for the entire training dataset at each epoch.

3. Full batch Gradient Descent

In this method all the training examples are shown to the learning algorithm and the weights are updated.

Cost Function

There are various cost functions. Below are some examples;

1. Mean Squared Error Function

$$E = \frac{1}{2}\sum_{i\ \epsilon\ Output}(y_i - o_i)^2$$

where y_i is the predicted output o_i is the actual output

2. Cross-Entropy Function

$$-\frac{1}{n}\sum_{i=1}^{n}\log(f(x_i, l_i, W))$$

where the f function is the model's predicted probability for the input $x_i's$ label to be l_i, W are its parameters, and n is the training-batch size

3. Negative Log-Likelihood Loss (NLL) function

NLL is the cost function used in all the experiments of the report.

$$NLL(\theta, D) = -\sum_{i=0}^{|D|}\log P(Y = y^{(i)}|x^{(i)}, \theta)$$

where $y^{(i)}$is the value of the output is, $x^{(i)}$ is the value of the feature input, θ is the parameters and D is the training set.

4.2.2.7 Learning rate

Learning rate controls the change in the weight from one iteration to another. As a general rule smaller learning rates are considered as stable but cause slower learning. On the other hand higher learning rates can be unstable causing oscillations and numerical errors but speed up the learning.

$$\begin{pmatrix} Weight \\ correction \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} learning - \\ rate\ parameter \\ \eta \end{pmatrix} * \begin{pmatrix} local \\ gradient \\ \delta_j(n) \end{pmatrix} * \begin{pmatrix} input\ signal \\ of\ neuron\ j, \\ y_i(n) \end{pmatrix}$$

4.2.2.8 Momentum

Momentum provides inertia to escape local minima; the idea is to simply add a certain fraction of the previous weight update to the current one, helping to avoid becoming stuck in local minima.

$\Delta w_{ij}(n) = \alpha \Delta w_{ji}(n-1) + \eta \delta_j(n) y_i(n)$, where α is the momentum

4.2.2.9 Softmax

Softmax is a neural transfer function that is generalized form of logistic function implemented in the output layer that turns the vectors z_j into the probabilities that add up and constraint to 1.

$$y_j = \frac{\exp(z_j)}{\sum_{i=1}^{k} \exp(z_i)}, where\ z_j = \sum_{i=1}^{k} x_i * w_{i,j} + b$$

4.2.2.10 Summary of Multilayer Perceptron (MLP)

For classification a softmax function may be incorporated in the output layer that will give the probability of each occurring class. Activation function is used to compute the predicted output of each neuron in each layer by using inputs, weights and bias. The back propagation method trains the multilayer neural network by modifying its synaptic connection weights between the layers to improve model performance based on the error correction learning function which needs to be continuous and differentiable. The following parameters have been evaluated in the experiments:

* Number of hidden layers.

* Number of neurons in the hidden layers.

* Learning rate and Momentum.

* Types of Activation function.

4.3 Overview of Deep Learning

Before 2006 various failed attempts at training deep supervised feedforward neural networks were made that resulted in overfitting of the performance on the unseen data that is training error is going down while validation error starts to go up. A deep network usually means an artificial neural network that has more than one hidden layer [65]. Training the deep hidden layers required more computational power. Having a greater depth seemed to be better because intuitively neurons can make the use of the work done by the neuron in the layer below resulting in distributed representation of the data. Bengio suggests that the neurons in the hidden layers are seen as feature detectors learned by the neuron in the below layer. This result in better generalisation that is a subset of neurons learns from data in a specific region of the input space.

Moreover deeper architectures can be more efficient as fewer computational units are needed to represent the same functions, achieving greater efficiency. The core idea behind the distributed representation is the sharing of statistical strengths where different components of the architecture are re-used for different purposes [66].

Deep neural architectures are composed of multiple layers utilising non-linear operations, such as in neural nets with many hidden layers. There are often various factors of variation in the dataset, like aspects of the data separately and often independently may vary. Deep Learning algorithms can capture these factors that explain the statistical variations in the data, and how they interact to generate the kind of data we observe. Lower level abstractions are more directly tied to particular observations; on the other hand higher level ones are more abstract because their connection to perceived data is more remote. The focus of deep architecture learning is to automatically discover such abstractions, from low level features to the higher level concepts. It is desirable for the learning algorithms to enable this discovery without manually defining necessary abstractions. Training samples in the dataset must be at least as numerous as the variations in the test set otherwise the learning algorithm cannot generalize. Deep Learning methods aim to learn feature hierarchies, composing lower level features into higher level abstractions. Deep neural nets with a huge number of parameters are very powerful machine learning systems [67]. However, overfitting is a serious problem in deep networks. Overfitting is when the validation error starts to go up while the training error declines. Dropout [68] is one of the regularisation techniques for addressing this problem which is discussed later.

Today one of the most important factors for the increased success of Deep Learning techniques is advancement in the computing power. Graphical Processing Units (GPU) and cloud computing are crucial for applying Deep Learning to many problems. Cloud computing allows clustering of computers and on demand processing that helps to reduce the computation time by parallelising the training of the neural network. GPUs, on the other hand, are special purpose chips for high performance mathematical calculations, speeding up the computation of matrices.

In 2006-7 three papers [69] revolutionized the Deep Learning discipline. The key principles in their work were that each layer can be pre-trained by unsupervised learning, done one layer at a time. Finally supervised training by backpropagation of the error is used to fine-tune all the layers, effectively giving better initialization by unsupervised learning than by random initialization.

4.3.1 Restricted Boltzmann Machine and Deep Belief Networks

One of the unsupervised algorithms is Restricted Boltzmann Machines (RBM) that is used to pre-train deep belief network [70]. The RBM is a simplified version of the Boltzmann Machine, inspired by statistical mechanics, which models energy based probabilities for the underlying distributions of the given data sets, from which conditional distributions can be derived.

Boltzmann Machines are bi-directionally connected networks of stochastic processing units of visible units and hidden units. The raw data corresponds to the 'visible' neurons and samples to observed states and the feature detectors correspond to 'hidden' neurons. In a Boltzmann Machine visible neurons provide the input to the network and the environment in which it operates. During training visible neurons are clamped (set to a defined value) determined by the training data. Hidden neurons on the other hand operate freely [71]. However Boltzmann Machine are difficult to train because of its connectivity. An RBM has restricted connectivity to make learning easier; there are no connections between hidden units in a single layer forming a bipartite graph, depicted in figure 4.2.3.1a. The advantage of this is that the hidden units are updated independently and in parallel given the visible state. These networks are governed by an energy function that determines the probability of the hidden/visible states. Each possible joint configuration of the visible and hidden units has a Hopfield energy determined by the weights and biases. The energies of the joint configurations are optimised by Gibbs sampling that learns the parameters by minimising the lowest energy function of the RBM.

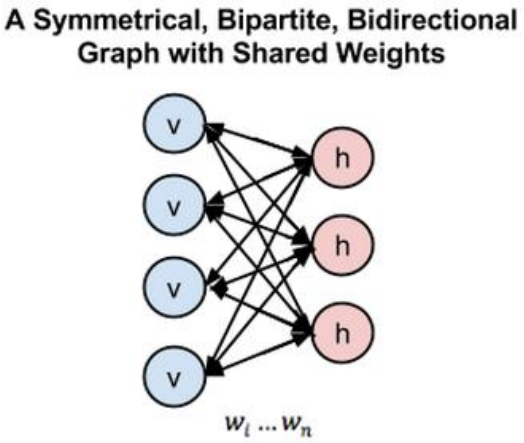


Figure 4.3.1a, left layer represents the visible layer and right layer represents the hidden layer Source: [72]

In Deep Belief Network (DBN), RBM is trained by input data with important features of the input data captured by stochastic neurons in the hidden layer. In the second layer the activations of the trained features are treated as input data. The learning process in the second RBM layer can be viewed as *learning feature of features.* Every time a new layer of features is added to the deep belief network, a variational lower bound on the log-probability of the original training data is improved. [73]

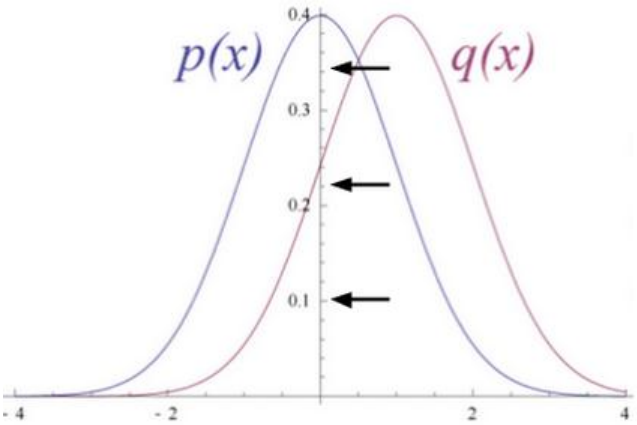


Figure 4.3.1b shows RBM converts its data distribution into a posterior distribution over its hidden units. Source: [74]

The weights of the RBM are randomly initialised causing the difference in the distribution of p(x) and q(x). During learning, weights are iteratively adjusted to minimise the error between p(x) and q(x). In figure 2.3a q(x) is the approximate of the original data and p(x) is the original data. The rule for adjusting the synaptic weight from neuron one and another is independent of whether both the neurons are visible or hidden or one of each [75]. The updated parameters by the layers of RBM are used as initialisation in DBNs that fine-tunes all the layers by supervised training of backpropagation.

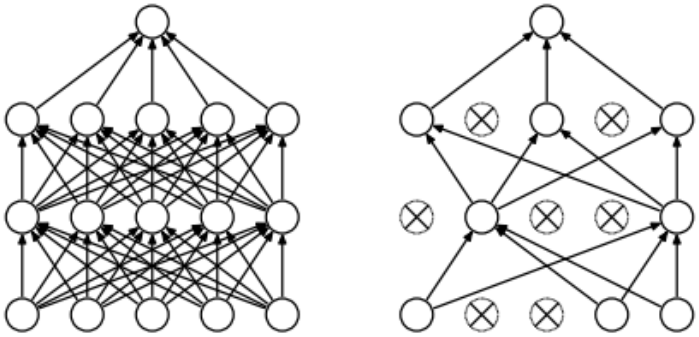
For the IDS data of KDD Cup 1999, it is appropriate to use multimodal (Bernoulli-Gaussian) RBM as KDD Cup 1999 consists of mixed data types, specifically continuous and categorical. In multimodal RBM there are two different channel input layers used in the RBM, one is Gaussian input unit used for continuous features and the other one is Bernoulli input unit layer where binary features are used. Using multimodal RBM is beyond the scope of the project [76].

4.3.2 Dropout

Recent developments have introduced powerful regularisers to deep networks to reduce overfitting. In machine learning regularisation is additional information usually introduced in the form of a penalty to penalize complexity of the model that leads to overfitting [77].

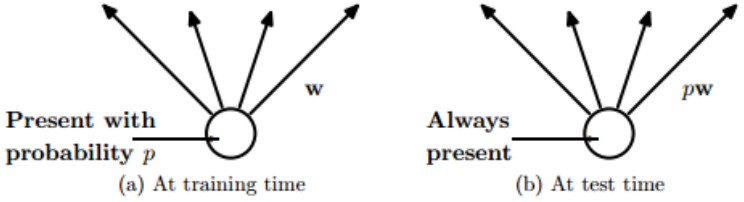
Dropout is a regularisation technique for deep neural networks introduced by Hinton [78] which consists of preventing co-adaptation of feature detectors by randomly turning off a

portion of neurons at every training iteration but using the entire network (with weights scaled down) at test time. Dropout reduces over-fitting by being equivalent to training an exponential number of models that share weights. There exists an exponential number of different dropout configurations for a given training iteration, so a different model is almost certainly trained every time. At test time, the average of all models is used, which acts as a powerful ensemble method.



(a) Standard Neural Net (b) After applying dropout.

In figure 4.3.3a dropout randomly drops the connections between the neural network layers



(a) At training time (b) At test time

In figure 4.3.3b at training time the connections are dropped with probability p, while at the test time weights are scaled to $p\mathbf{w}$

Averaging many models usually has been the key for many winner of the machine learning competitions. Many different types of model are used and then combined to make predictions at test time. Random forest is a very powerful bagging algorithm which is created by averaging many decision trees giving them different training sample sets with replacement. It is well known that the decision trees are easy to fit to data and fast at test time so averaging different individual trees by giving them different training sets is affordable.

However, using the same approach with deep neural networks will prove to be very computationally expensive. It is already costly to train individual deep neural networks and training multiple deep neural networks and then averaging seems to be impractical. Moreover a single network that is efficient at test time is needed rather than having lots of large neural nets.

Dropout is an efficient way to average many large neural nets. Each time while training the model hidden units can be omitted with some probability as in figure 2.3.3a, which is usually

p = 0.5, when the training example is presented. As a result a 'mean network' model that has all the outgoing weights halved is used at test time as in figure 2.3.3b. The mean network is equivalent to taking the geometric mean of the probability distributions over labels predicted by all $2\text{^}N$ possible networks with a single hidden layer of N units and 'softmax' output layer.

As per the [79] is mathematical proof of how dropout can be seen as an ensemble method.

$p_e(y|x)$ is the prediction of the "ensemble" using the geometric mean.

$p_d(y|x)$ is the prediction of a single sub model.

d is the binary vector that tells which inputs to include into the softmax classifier.

$p_d(y|x) = softmax(W * (x * d))$

Suppose there are N different units. There will be $2\text{^}N$ possible assignments to d, and so;

$p_e(y|x) = \frac{\prod_d p_d(\mathrm{y|x})^{(1/2\text{^N})}}{\sum_{y'} \prod_d p_d(y'|\mathrm{x})^{(1/2\text{^N})}}$ where y is the single and y' is the vector of the classes index.

The sum of the probabilities of the output by a single sub-model is used to normalise $p_e(y|x)$.

$p_e(y|x) \propto \prod_d p_d(\mathrm{y|x})^{(1/2\text{^N})}$

$= \prod_d \mathrm{softmax}(\mathrm{W} * (\mathrm{x} * \mathrm{d}))^{\left(\frac{1}{2^N}\right)}$

$= \frac{\prod_d e^{(W*(x.*d))\cdot}}{\sum_{y'} e^{(W*(x.*d))^{1/2\text{^}N}}}$ as per the definition of softmax

$= \frac{\prod_d e^{(W*(x.*d))^{1/2\text{^N}}}}{\prod_d \sum_{y'} e^{(W*(x.*d))^{1/2\text{^}N}}}$

$\propto \prod_d e^{(W*(x.*d))[y]^{1/2\text{^N}}}$

$= \prod_d e^{(1/2^{\mathrm{N}})(W*(x.*d))\cdot}$

$= e^{1/2\text{^}N} \sum_d \mathrm{W} * (\mathrm{x} * \mathrm{d})$

$= e^{\left(\left(\frac{1}{2}\right)*W*y\right)}$

So the predicted probability must be proportional to this. To renormalize the above expression, it is divided by $\sum_{y'} e^{\left(\left(\frac{1}{2}\right)*W*y\right)}$ which means the predicted probability distribution is $softmax((\frac{1}{2})\, W * x)$

Another way to view dropout is that it is able to prevent co-adaption among the feature detectors. Co-adaption of the feature detector means that if a hidden unit knows which other hidden units are present, it can co-adapt with them on the training data. However, on the test dataset complex co-adaptions are likely to fail to generalise.

Dropout can also be used in the input layer at a lower rate, typically 20% probability. The concept here is the same as de-noising auto encoders developed in [80]. In this method some of the inputs are omitted. This hurts the training accuracy but improves generalization acting in a similar way as adding noise to the dataset while training.

In 2013 a variant of dropout is introduced called Dropconnect [81]. Instead of dropping hidden units with certain probability weights are randomly dropped with certain probability. It has been shown that a Dropconnect network seemed to perform better than dropout on the MNIST data set.

4.3.3 Techniques to deal with class imbalance

A class imbalance problem arises when one of the classes (minority class) is heavily under-represented in comparison to the other classes (majority class). This problem has real world significance where it is costly to misclassify minority classes such as detecting anomalous activities like fraud or intrusion. There are various techniques to deal with the class imbalance problem.

4.3.4 SMOTE: Synthetic Minority Over-sampling Technique

One widely used approach to address the class imbalance problem is resampling of the data set. The sampling method involves pre-processing and balances the training data set by adjusting the prior distribution for minority and majority classes. SMOTE is an over-sampling approach in which the minority class is over-sampled by creating "synthetic" examples rather than by over-sampling with replacement.

In [82] it has been suggested that oversampling the minority class by replacement does not improve the results significantly rather it tends to over-fit the classification of the minority class. Instead the SMOTE algorithm operates in 'feature space' rather than 'data space'. It creates synthetic samples by oversampling the minority class which tends to generalize better. The idea is inspired by creating extra training data by operating on real data so that there is more data that helps to generalize prediction. In this algorithm firstly K nearest neighbours are computed for the minority class. Then synthetic samples of the minority class are computed in the following manner: a random number of nearest neighbours is chosen and distance between that neighbour and the original minority class data point is taken. This distance is multiplied by a random number between 0 and 1 and adds the result to the feature vector of the original minority class data as an additional sample, thus creating synthetic minority class samples.

4.3.5 Cost-sensitive learning in neural networks

Cost sensitivity learning [83] seems to be quite an effective way to address the class imbalance for classification problems. Three cost sensitive methods have been described that are specific to neural networks.

1. Incorporate the prior probabilities of the class in the output layer of the neural network while testing unseen examples (explained later in detail).

$$P'(i) = \frac{CostVector[i] * P(i)}{\sum_j CostVector[j]P(j)}$$

2. Adjusted learning rates based on the costs. Higher learning rates should be assigned to examples with high misclassifications costs making a larger impact on the weight changes for those examples.

$$\eta(p) = \frac{\eta * CostVector[class(p)]}{max_i CostVector[i]}$$

3. Modifying the mean square error function. As a result, the learning done by backpropagation will minimize misclassification costs. The new error function is:

$$E = \sum_{p \in Examples} \frac{1}{2} \sum_{i \in Output} ((y_i - o_i) * K[class(p), i])^2$$

with the cost factor being $K[i, j]$.

This new error function results in a new delta rule used in the updating of the weights of the network:

$$\delta_j = \begin{cases} (y_j - o_j) o_j (1 - o_j) K^2[c, j], \\ o_j (1 - o_j) \sum_k \delta_k w_{kj}, \end{cases}$$

where the first equation represents the error function for output neurons and the second equation represents the error function for hidden neurons.

4.4 Methods

4.4.1 Computational tools

Python is the primary programming language used in this project. Python has a lot of libraries that can be used for data manipulation and analysis. Scikit-learn [84] is the most popular machine learning Python library that offers a variety of algorithms along with utilities for calculating confusion matrices, accuracy levels, recall and precision tables to evaluate the performance of a learning algorithm. A Python dictionary function was used to store the results of the network and the Python pickle function was used to retrieve stored results.

Python libraries like NumPy and Pandas were used extensively for data manipulation. A NumPy random seed is used to make sure that for every run the results are reproducible. Since the weights and biases are initialised randomly following a normal distribution, NumPy random seed is used so that the weights and biases initialised are the same in every run. Finally the Matplotlib library was used for plotting. String formatting has been used to get separate graphs for different sets of parameters in the network.

4.4.1.1 Deep Learning library for the implementation

Theano [85] is the Deep Learning Python library that has been used in this project. Introduced as a CPU and GPU compiler by Bergstra at the Lisa lab of the University of Montreal in 2010, it allows defining, optimizing, and evaluating mathematical expressions involving multi-dimensional arrays efficiently. Below are some of the appealing features of Theano:

1. Tight integration with NumPy:

Theano syntax is very similar to NumPy syntax. For example,

```
Import theano.tensor as T
Import numpy as np
```

`T.dot(x,W_x)` and `np.dot(x,W_x)` Both implement the same function, the difference between the two is that NumPy uses numeric variables while Theano uses symbolic variables.
Theano is used for symbolic mathematical expressions that are compiled to a function that can operate on numerical data. For example,

```
x = T.vector("x")
y = T.vector("y")
fn = x*y
p = theano.function(inputs=[x, y], outputs=fn)
```

We can assign variables x and y using NumPy arrays to the numerical data of these variables which can then be used by compiled function like *p* above. To update a variable used in an equation (for example, while learning), Theano needs it to be in a special wrapper called a shared variable. Below are the model parameters for the first hidden layer in feedforward neural networks.

```
W_x = theano.shared(W_x, name="W_x")
b_h = theano.shared(b_h, name="b_h")
```

2. Transparent use of a GPU:

Theano is designed to easily use Graphical Processing Units (GPUs). In order to use the GPUs, all we need to is to create a file named .theanorc in the home folder on the server with just the following lines [86]:

```
[global]
device=gpu
floatX=float32
```

Operations on data of type float32 are accelerated along with matrix multiplication and large element-wise operations especially when the arguments are large enough [87]. The use of GPUs gives around 40x speedup over the use of CPUs particularly in larger networks. This feature of GPUs is one of the reasons for the revival and success of Deep Learning in the twenty-first century.

1. Efficient symbolic differentiation:

Theano performs derivatives for functions with one or many inputs allowing users to quickly prototype complex machine learning models fit by gradient descent without manually differentiating the gradient. Hard coding of the derivative for each function is not needed and human errors can be avoided when implementing backpropagation. For example, function T.grad determines the partial derivative of cost with respect to the parameters T.grad(cost=L, wrt=W)

2. Speed and stability optimizations:

Theano increases numerical stability. There are stable implementations for log(1+x) or log(sigmoid(x)) even when x is really tiny. Optimisations such as loop fusion are applied to improve speed.

3. Dynamic C code generation:

Low level C code is used to improve performance.

4. Theano is highly expressive:

Anyone looking at the Theano code with a background in neural networks will be able to tell what the Theano code is doing. This is not the case for every Deep Learning library like PyLearn2, Torch, Lasagne, OpenDeep or any other library. For example, if dropout was implemented using a Deep Learning library it might be that we would have been unable to understand how dropout is integrated into feedforward neural networks.

Theano has a large community [88] that has been very helpful in developing the codes and assisting in their use. Error messages produced by Theano are quite different from the error

produced by the standard Python packages because Theano codes are compiled. Therefore at times it had been difficult to understand the error message and debug the error.

4.4.2 MNIST Experiment

In order to confirm that if our implementation of dropout is correct we decided to verify it on the well-known MNIST handwritten image dataset.

The famous MNIST handwritten digits image containing dataset digits 0 to 9 has been used for the experiments with 50,000 training examples, 10,000 validation examples and 10,000 testing examples. An image is represented as a 1-dimensional array of 784 (28 x 28) float values between 0 and 1 (0 for black and 1 for white). There is no need for pre-processing and formatting the data. MNIST dataset is one of the most well studied datasets in the area of computer vision and machine learning. It is based on two data sets collected by NIST, the United States' National Institute of Standards and Technology. The NIST data sets were stripped down and put into a more convenient format by Yann LeCun, Corinna Cortes, and Christopher J. C. Burges [89].

An initial experiment was conducted to see the effect of dropout with a learning rate of 0.1 and no momentum. The figures 4.4.2a and 4.4.2b below show the comparison.

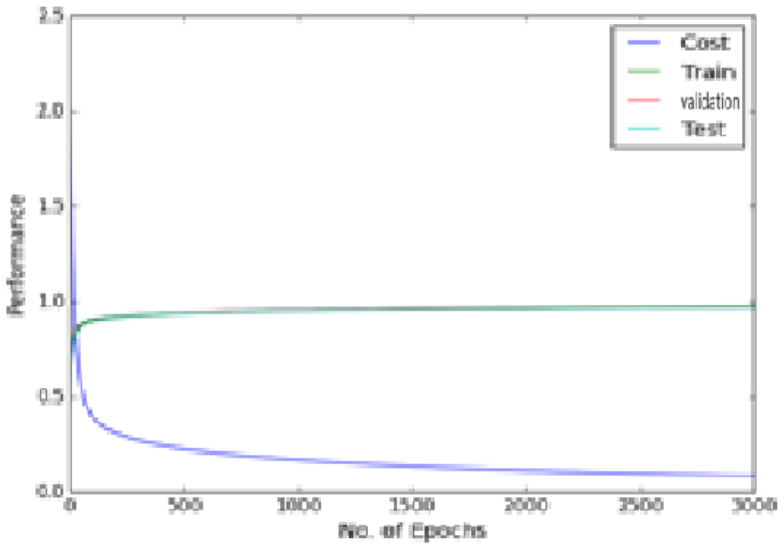


Figure 4.4.2a is the experimental result of 2 hidden layers standard MLP of 800 hidden nodes with 3000 epochs

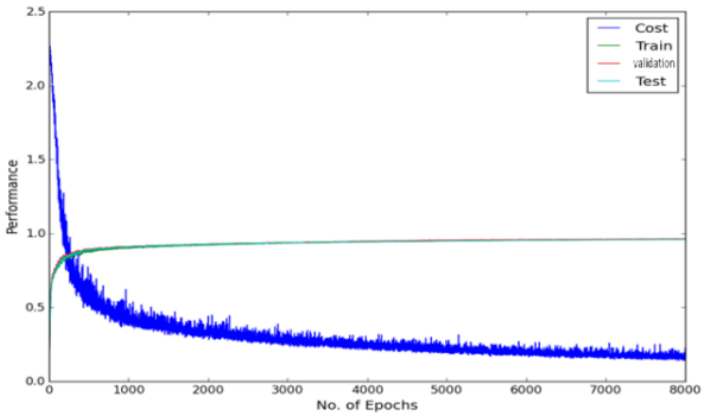


Figure 4.4.2b is the experimental result of dropout on 2 hidden layers MLP of 800 hidden nodes and 8000 epochs as dropout made the learning of the network slower. Hence larger number of epochs is required.

As can be seen in the figure 4.4.2b there is a fluctuation in the training cost value (obtained by the cost function), and accuracies created by the dropout noise. Moreover it turns out that problem of local minima in the error surface is encountered in the dropout network (figure 4.4.2c depicts error surface with local minima). That is cost value plateaus at the higher value than at the value without using dropout, making the slow learning process for the network. Training cost value without using dropout is 0.083 while using dropout is 0.225 suggesting problem of local minima at the end of epoch of 3000 and 8000 respectively. Varying learning rate and momentum rate helps to solve the problem of local minima [90]. It is little wonder that Hinton used various techniques to make dropout work.

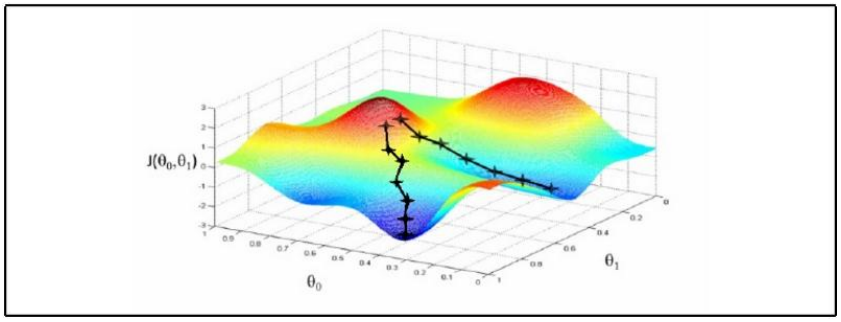
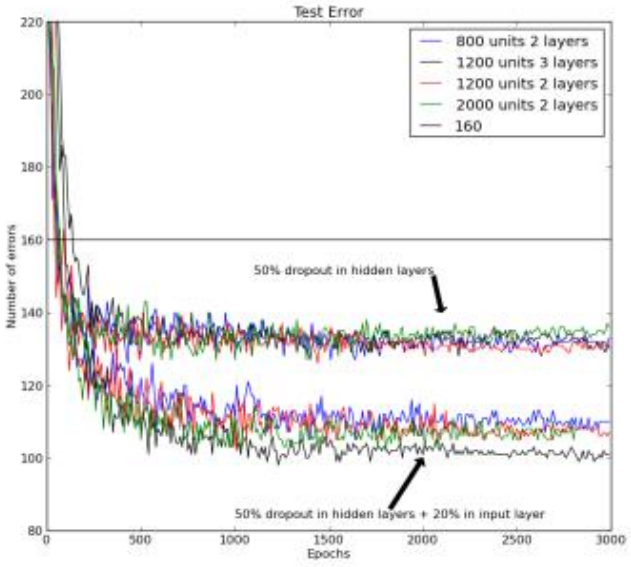


Figure 4.4.2c depicts an error surface with poor local minima (source [91])



Hinton experiment results on MNIST (source [92])

Hinton [93] starts off his experiment with learning rate of 10 and exponentially decays the rate at 0.998 for each training epoch and then stabilizes the learning rate at 500 epochs. In order to prevent the model from blowing up, due to the very high learning rate, he puts an upper bound constraint on the squared length that is L2 norm of the incoming weight vector for each individual hidden unit. After cross-validation it turns out that maximum squared length of l = 15 give the best results. If a weight-update goes beyond the upper bound, weights are renormalized accordingly. Similarly he initially set the momentum rate at 0.5 and linearly increases to 0.99 over the first 500 epochs speeding up the training, in the subsequent epoch momentum rate stays constant at 0.99. We had to use 8000 epochs because learning is significantly slower with dropout. As per Hinton, adding dropout in the input layer with 20% random noise improves the result by quite a large margin. Apart from these hyper parameters, in the network architecture he uses stochastic gradient descent as an optimizer with mini-batches of 100 and a cross-entropy objective function.

In order to combat the problem of local minima we used some of the techniques of varying the learning rate and momentum over the epochs. The experiment started off at a learning rate

of 0.1 and momentum of 0.2. The learning rate increased stepwise at 500, 1500, 3000 epochs decreasing to 0.08, 0.06 and 0.04 respectively at the same time momentum rate increased to 0.4, 0.6 and 0.8 at the same time. The testing accuracy level increased to 96.65% by varying learning rate and momentum as mentioned. The figure 3.2d below shows the performance of the described networks

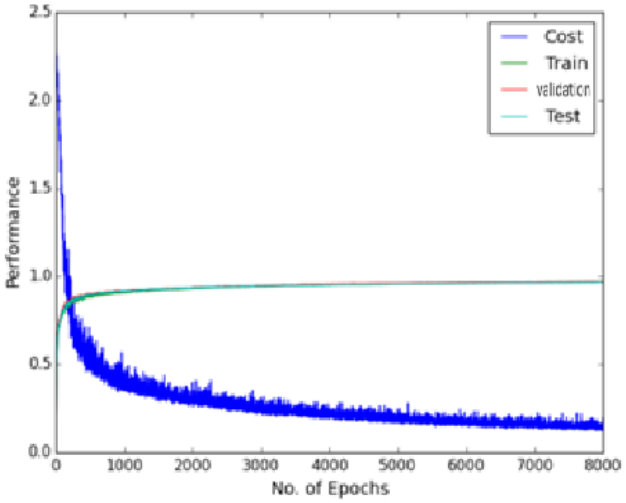


Figure 4.4.2d

Table below is the comparison of our three experiments on the MNIST dataset. In the table test accuracy levels of three experiments can be seen. The number of epochs required to get the training costs were different for all three experiments.

Training Costs	Test Accuracy without Dropout and no techniques [94] (A)	Test Accuracy with Dropout and with no techniques (B)	Test Accuracy with dropout and techniques (C)
0.183	95.00%	- [95]	95.78%
0.417	90.17%	90.80%	91.45%
1.310	77.27%	79.43%	79.63%
1.722	67.68%	74.50%	75.76%

It can be seen that at various training cost levels experiment B result is better than experiment A's. It took a network without dropout only 10 epochs to reach training cost of 1.310, whereas around 110 epochs were required to get the same training costs and thus better accuracy level by a network with dropout. Interestingly if learning rate and momentum are varied as in experiment C, not only accuracy level increases but also it takes fewer epochs, around 95 epochs, boosting the overall performance.

Later we did the same experiment with 100-sized mini-batches along with the same architectures and parameters as before. It turns out that the mini-batches made the learning easier as it started off (in the initial epochs) with the higher training and testing accuracy levels and learning also converged quicker and at the higher level with testing accuracy level of 98.6%, getting closer to the benchmark.

4.4.3 KDD Cup 1999 dataset pre-processing

It is very important to get data ready before doing any analysis. The treatment of categorical and continuous variables is quite different. Categorical variables are converted by one-hot encoding. In this encoding, a feature is obtained for each categorical variable in which one neuron/input represents each category, for example the following is the subset of two features in KDD Cup 1999 data set.

Type	Service
Tcp	Private
Tcp	Private
Tcp	ftp_data
ICmp	Eco_i
Tcp	telnet
Tcp	Http

Figure 3.3a, sample with 2 features of KDD 99

Protocol		Service				
TCP	ICMP	Private	ftp_data	eco_i	telnet	http
1	0	1	0	0	0	0
1	0	1	0	0	0	0
1	0	0	1	0	0	0
0	1	0	0	1	0	0
1	0	0	0	0	1	0
1	0	0	0	0	0	1

Figure 4.4.3b, table showing one-hot encoding of categorical samples.

The first column of figure 3.3b gives the binary representation of (tcp / not tcp) encoded as (0, 1) respectively, where tcp is represented in the above diagram as 1 and whichever example is not tcp is represented as 0. Similarly the second column of figure 3.3b gives the representation of (icmp / not icmp). When all the variables are encoded from the first column

of the categorical feature, the feature variables from the second feature type will be encoded in the same manner. As a result of one-hot encoding the input size has expanded from 42 units to 120 units.

Continuous features (attributes) have been transformed using the following rescaling;

$Standardisation: \frac{(x - mean(x))}{\text{standard deviation(x)}}$, where x is the value of the feature

One reason for scaling the data is to ensure that all variables are seen as equally important by smoothing out feature to feature variations. For example if feature A ranges between 1 and 100 while feature B ranges between 0.1 and 0.001, the network will assign smaller weights to feature A and larger weights to feature B. By rescaling all the features are brought to the same scale. Neural networks learn faster and give better result with pre-processed input variables as a result. All the samples in the dataset have been shuffled to ensure that if mini-batches are used then the training examples in the mini-batch will not necessarily have the same order of training sets in each mini-batch. Data shuffling randomises the order of the data. However instead of mini-batch training full gradient descent was carried out throughout this project.

In the raw data [96] there was various attack types that were categorize into four main attack classes by following [97] to compare with the winning entry's method given in section 4.3.3.

4.4.4 Details of MLP Experiments on KDD Cup 1999 data

One of the challenging tasks in neural networks is to find the optimal hyper-parameters to get the best performance. It is always difficult to tell which combination of parameters works well. In the end these are all still heuristics most of the time. In the neural networks the parameter space is huge and it is difficult to explore each hyper parameter extensively. The experiments started with one hidden layer along with the hyper-parameter tuning. Hidden layers were then stacked and evaluated. Four hidden layers were the maximum that we have stacked and reported the results of. Moreover some techniques mentioned in the literature review were incorporated in to the learning algorithm like SMOTE, dropout and cost-sensitive learning. The experiments were conducted in full batch mode of back-propagation learning that is weights updated after all training examples were presented to the learning algorithm in each epoch. The cost is computed and parameters are updated to minimize the cost value. At the time of prediction of unseen examples the final updated parameters are used and there are no parameter updates.

Weights and bias are initialised using Python NumPy arrays. Weights are initialised using random standard normal distribution with mean zero and standard deviation of 0.05. Biases are initialised with values of zero.

One hidden layer MLP implementation in Theano was taken from [98] and from there we worked all the way and changed the code to incorporate various other techniques. In the implementation one more hidden layer is stacked by initialising weights and biases the same

way as in one hidden layer MLP. Same goes on for three and four hidden layer MLPs given in the subsequent sections of the report.

Theano codes for momentum, dropout and rectified linear units were fetched from the online Theano user community [99]. The value of the momentum has to be multiplied by the parameter update of previous epoch to get the value of parameter updates of current epoch. To implement rectified linear units Theano's maximum function was used so that the value of the hidden layer stays non-negative resulting in the sparsity. Theano already had a module for tangent activation functions. In the implementation of dropout, Theano Bernoulli function is multiplied by each hidden layer function in order to drop the weights and biases with probability p at the time of training. For the test using unseen data separate hidden layers are written with the same value of initialised/updated weights and biases being used but this time weights are scaled down by probability of p.

SMOTE has been implemented by using Costcla Python library [100] which is built on top of Scikit- learn, Pandas and NumPy. Cost-sensitive learning classification has been implemented as detailed in [101] which is explained in section 4.2.2 of the report.

The vanishing gradient problem was solved by using Rectified Linear Units as activation functions as these introduce sparsity effect on the networks. In order to evaluate the results, confusion matrix, recall rate, precision and cost per test sample have been considered because of the class imbalance problem. Accuracy levels of training and testing set have also been calculated to see when the network actually over-fits. Cross-validation does not seem to be an appropriate way to evaluate the performance and get the optimal results. It is because the distribution of the training set and test set is quite different [102]. Cross-validation uses one subset of training data for learning and another subset of training data is used for validating. We used the subset of the training set for learning and the remaining subset of the training set for validating. It turned out that accuracy level of the validation set (subset of training data) was very high (higher than the accuracy level of the test set) confirming that the distribution of training and test set is quite different making the problem challenging.

All the experiments were done on CPU (not GPU unfortunately). Some of the experiments took days to finish. As the network was grown bigger, computation became time consuming task. The four hidden layer MLP network with dropout was the largest network experimented in this project and it took around four days to finish that experiment with no hyper-parameter tuning which is described in section 4.3.5. Tuning hyper-parameters of the networks was the most time consuming part of the exercise. Three and four hidden layers MLP usually took a day to finish running with each set of hyper-parameters.

4.5 Results

4.5.1 Standard one hidden layer MLP for KDD Cup 1999 data

Initially a single hidden layer MLP was used in the experiments to get the optimal configurations and architecture. Description of the hyper-parameter search is given as follows.

At the time of hyper-parameter tuning, a learning rate of 0.8 was chosen among the arbitrary set of 0.01, 0.1, 0.5, 0.8, 0.9, 1.0 and 1.2. Momentum of 0.8 was also chosen after hyper-parameter search. It is surprising to see a large learning rate along with large momentum to work well together because it is general practice to use a lower learning rate along with a large momentum to ensure that training cost oscillates smoothly down in the error surface while learning. However since we are using full batch gradient descent a higher learning rate seems reasonable [103]. Using an activation function of rectified linear units turns out to be better than the tangent units. Cost function of Negative Log-Likelihood (NLL) is chosen because of the softmax function in the ouput layer. It does not mean that NLL is the only cost function that can be used when softmax is in the output layer. Cost per test sample turns out to be 0.2472 by the standard single hidden layer MLP after hyper-parameter tuning. Cost per test sample is calculated by multiplying the confusion matrix given below and cost matrix given in section 1.3.3 and then divided by the total number of test examples which is around 0.3 million. Below are the graphical and tabular results of the network.

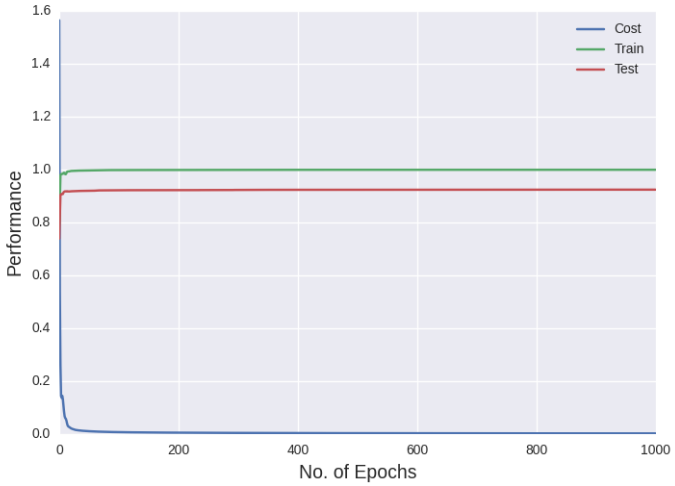


Figure 4.5.1a shows the cost, and training and test accuracies levels of single hidden layer MLP after all the hyper-parameter selections

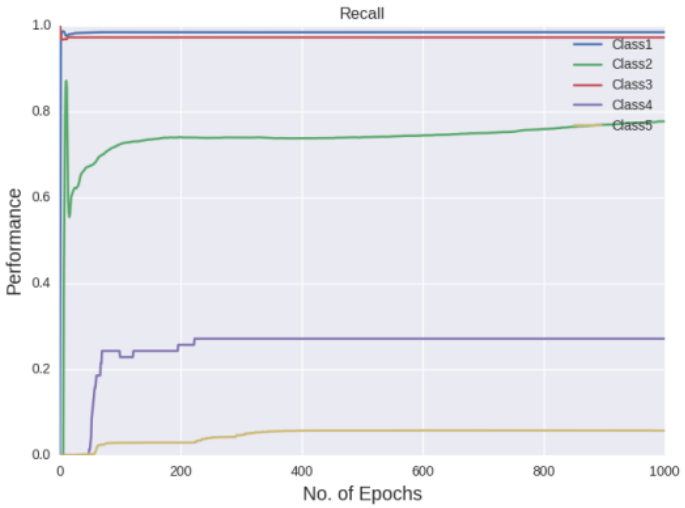


Figure 4.5.1b shows the recall curves of all the five classes after all the hyper-parameter selections

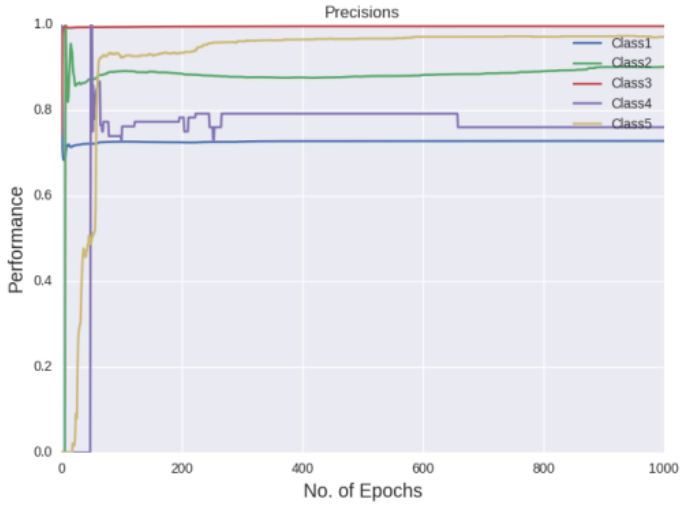


Figure 4.5.1c shows the precision curves of all the five classes after all the hyper-parameter selections

Confusion Matrix by the last 1000 epoch:

	Predicted Class				
Actual Class	59,679	200	698	4	12
	739	3,238	188	-	1
	6,153	132	223,567	-	-
	36	-	-	19	15
	15,378	24	7	2	936

Recall by the last 1000 epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.984	0.777	0.972	0.271	0.057

Precision by the last 1000 epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.728	0.901	0.996	0.760	0.971

In figure 4.5.1a we can see that there is a gap between the training and test accuracy. This gap has been observed in all the experiments of the project. Training accuracy has been around 99% and test accuracy has been around 92%. This gap is due to the different distribution of training and test datasets. When portion of the training set was taken as the validation set, that same gap was not observed.

The figure 4.5.1b and 4.5.1c gives out the recall and precision results respectively. As we can see the performance of the minority class of 4 and 5 is quite low. Hence we tried several approaches to deal with the class imbalance problem. After getting the results of one hidden layer of MLP, the objective of this project is to get the better results.

4.5.2 Experiments to deal with class imbalance

4.5.2.1 SMOTE-MLP for KDD Cup 1999 data

SMOTE has been described in section 2.4.1 as a technique to oversample the minority class by creating synthetic examples of minority class. SMOTE was used to increase the samples of minority class of U2R and Probe to 0.83% which is the proportion of R2L attack type from the proportion of 0.01% and 0.23% respectively. After the pre-processing step of oversampling the minority class along with data pre-processing step described in section 3.3 the processed KDD Cup 1999 data is tested on the single hidden layer MLP with the same architecture described in section 4.1 that is after hyper-parameter tuning.

The overall performance of the network on the data pre-processed by SMOTE has declined as can be seen in figure 4.2.1a precision and figure 4.2.1b recall. The lower performance of the network can be explained by the fact that the synthetic examples created by SMOTE makes them very similar to the majority attacks in the feature space. As a result the network is confusing between the minority class with the synthetic examples and the majority classes.

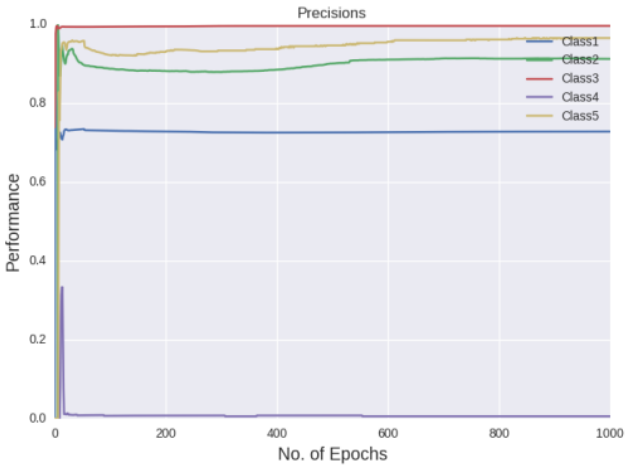


Figure 4.5.2a shows the precision curves of all the classes by the one hidden layer MLP with SMOTE

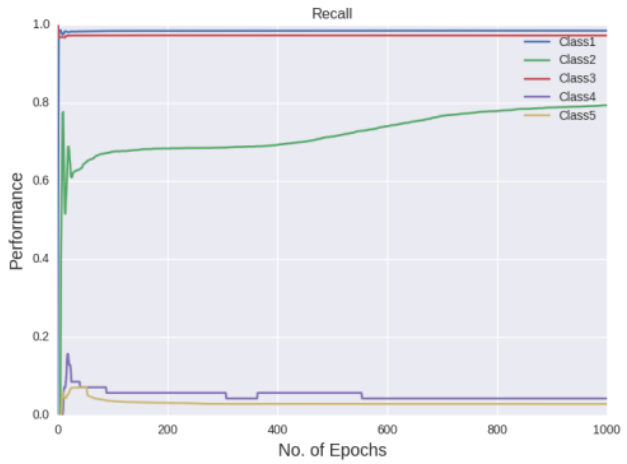


Figure 4.5.2b shows the recall curves of all the classes by the one hidden layer MLP with SMOTE.

Recall at the end of 1000th epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.984	0.794	0.973	0.0429	0.029

Precision at the end of 1000th epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.728	0.912	0.996	0.006	0.965

Confusion matrix at the end of 1000th epoch

	Predicted Class				
Actual Class	59,682	192	700	10	9
	667	3,307	192	-	-
	6,184	110	223,558	-	-
	59	-	-	3	8
	15,385	16	1	473	472

It can be seen from the above tabular results, the performance of the minority classes has decreased. One possible way to improve the results is to try tuning the parameter. If we refer to previous section on SMOTE, the difference between the data generated by K-nearest neighbour and the original point is multiplied by a random number in the range 0 to 1. It will be interesting to see if we vary the range of this random number. Let's say that this random number instead belong in the range of 0 to 0.5. If the minority classes with the synthetic samples stay indistinguishable from the majority attack classes, there might be an improvement.

4.5.2.2 Cost-sensitive learning in MLP for KDD Cup 1999 data

There are three methods mentioned in section 2.4.2 for the cost-sensitive learning of neural networks. The first method, cost-sensitive classification, was implemented into the single hidden layer MLP with the same architecture described in section 4.1 that is after hyper-parameter tuning. The cost matrix is given in section 1.3.3 where the cost of misclassifying the R2L as normal is 4 and similarly, the cost of misclassifying the normal as R2L is 2. $P(i)$ is an estimate of the prior probability that an example belongs to the $i-th$ class. The following table is the proportion of the training data set from the [104] is considered as $P(i)$ in the computation.

Training ($P(i)$)
19.69%
0.83%
79.24%
0.01%
0.23%

The cost vector represents the expected cost of misclassifying an example that belongs to the ith class. The following is the cost vector for the classification.

Cost vector for the class i:

$$CostVector[i] = \frac{1}{1-P(i)} \sum_{j \neq i} P(j) * Cost[i,j]$$ equation 4.5.2.2a

For example,

$$cost\ vector\ for\ class\ 1 = \frac{(cost\_matrix[0,1]*0.0083 + cost\_matrix[0,2]*0.7924 + cost\_matrix[0,3]*0.0001 + cost\_matrix[0,4]*0.0023)}{(1-0.1969)}$$

where, $cost\_matrix[0,1] = 1$ the cost of misclassifying class 1 as class 2 and 0.0083 is the proportion of the class 1 in the total training examples. In this way the cost vector for class 1, 2, 3, 4, and 5 calculated to be 1.989, 1.801, 1.960, 2.197, and 2.395 respectively. Then the class probabilities $P(y(i))$ have been computed by the softmax function from the output layer of the network. In cost-sensitive classification, the learning procedure while training stays intact and probability estimates of the classes is modified only during classification of the unseen examples using the following equation;

$$P'(i) = \frac{CostVector[i]*P(y(i))}{\sum_j \text{CostVector[j]P(y(j))}}$$ equation 4.2.2b

In the implementation $P'(i)$ is calculated as follows;

$P'(i) = CostVector[i] * P\big(y(i)\big)$, where $CostVector[i]$ is the cost vector of class i of equation 4.5.2.2a, $P(y(i))$ is the probability estimate from the output layer of the network using a softmax function.

The altered probability $P'(i)$ will take account of expected costs of misclassification and will favour the classes with higher expected misclassification costs. While testing there is a

function in the implementation Theano code that selects the class index with the highest index value and sets that class index value to 1 and assigns 0 to the rest of the class. Therefore there is no need to normalize $P'(i)$ with denominator of $\sum_j \text{CostVector}[j]P(y(j))$ given in the equation 4.2.2b.

	Predicted Class				
Actual Class	59,678	196	701	4	14
	995	2,980	191	-	-
	6,143	116	223,593	-	-
	32	-	-	14	24
	15,370	18	4	2	953

Recall at the end of 1000th epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.726	0.900	0.996	0.700	0.962

Precision at the end of 1000th epoch

Class 1	Class 2	Class 3	Class 4	Class 5
0.985	0.715	0.973	0.200	0.058

As a result of cost-sensitive learning in the neural networks the correct classification of minority class 5 has increased from 936 to 953 increasing the recall rate to 5.83% at the 1000th epoch. This shows that given the appropriate misclassification cost table, our learning algorithm can adjust so that expected cost of misclassification for various classes goes down.

Below is the table of the comparison of test accuracy of one hidden layer MLP and one hidden layer MLP with cost-sensitive learning. As can be seen there is an overfitting by MLP with cost-sensitive learning where test accuracy declines after 750 epochs while training accuracy improves because it is supposed to stay intact in the cost-sensitive learning procedure. On the other hand, one hidden layer MLP's test accuracy remains the same. Trade-off for cost-sensitive classification is that since the modified network is biased towards classifying the minority class that carries higher misclassification costs, the classification performance for the majority class may go down as it is can be seen in the below table.

Epochs	MLP	MLP-Cost Sensitivity
250	92.22%	92.21%
500	92.37%	92.35%
750	92.39%	92.36%
850	92.42%	92.35%
1000	92.42%	92.34%

One of the popular ways neural network models deal with over-fitting is to use an early stopping rule. Training of cost-sensitive learning neural network can be halted after around 750 epochs which is when model starts to over-fit.

4.6 Deep MLP Experiments

4.6.1 Standard two hidden layers MLP for KDD Cup 1999 data

As more hidden layers are stacked it became apparent that the distribution of the number of hidden nodes needs to be changed as well. If more hidden layers are added then it is not necessary to have 175 hidden nodes in the first hidden layer. There is a distributed representation of the feature learning by the hidden nodes in the multiple hidden layers of the neural networks. Same configuration has been used as for the one hidden layer MLP apart from the hidden nodes. After hyper-parameter search from the following set of hidden nodes (175, 85), (175, 175), (100, 75), (75, 75), (75, 50), (100, 50), (75, 100), (100, 100) and (175, 150)

175 and 150 hidden nodes have been chosen for first and second hidden layers respectively for two hidden layers MLP. Single hidden layer MLP with 175 hidden nodes gives a better result than two hidden layer MLP. Cost per test sample is 0.2507. Below is the confusion matrix, precision and recall of the test set;

		Predicted Class			
Actual Class	59,611	228	740	3	11
	886	3,105	175	-	-
	6,106	121	223,625	-	-
	35	-	1	24	10
	15,545	97	83	2	620

Precision at 1000th epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.984	0.874	0.996	0.828	0.967

Recall at 1000th epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.984	0.745	0.973	0.343	0.038

There seems to be no improvement in the performance of the two hidden layer MLP. Below is the performance of accuracy level over the 1000 epochs;

Epochs	Training	Test
250	0.9977	0.9215
500	0.9984	0.9219
750	0.9989	0.9221
1000	0.999	0.9223

4.6.2 Standard three hidden layers MLP for KDD Cup 1999 data

Set of (100, 50, 75) were chosen for nodes of three hidden layers from the sets (100,50,35), (100,50,75), (100,75,35), (100,75,75), (175,150,50),(175,100,50) after hyper-parameter tuning. Architecture is same as multilayer perceptron with one or two hidden layers. It would have been interesting to see the effect of changing the remaining parameters like learning rate, momentum, etc on deeper network experimental results. However due to time-constraints it was not possible to carry that out.

Interestingly this network performed quite well on the class 4 (minority). And cost per test data sample is 0.2466 which is better than the single hidden layer MLP. Below is the confusion matrix, precision recall and accuracy tables;

		Predicted Class			
Actual Class	59,618	276	681	4	14
	789	3,177	198	-	2
	6,144	131	2,235,777	-	-
	31	-	-	17	22
	15,307	23	5	1	1,011

Precision at the end 2000th epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.728	0.881	0.999	0.773	0.964

Recall at the end of 2000th epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.984	0.763	0.997	0.243	0.062

Epochs	Training	Test
500	99.94%	92.42%
1000	99.95%	92.44%
1500	99.96%	92.42%
2000	99.97%	92.403%

It can be seen that the precision and recall for minority class have improved. Number of epochs varies with the number of hidden layers. This is because it takes more time for the network to detect the minority classes and for the training cost to converge. Interestingly training accuracy improves quicker as more hidden layers are stacked.

4.6.3 Standard four hidden layers MLP for KDD Cup 1999 data

One of the objectives of this project is to see the performance of dropout. In this experiment higher number of nodes is tested with the intention to use dropout (later with the same setup) to regularize and avoid over-fitting in the network. This was done deliberately to ensure that the network is expressive enough to benefit from the dropout. In [105] Hinton said that if the network does not over-fit, make it bigger. In the three hidden layers MLP there seemed to be insignificant overfitting over 2000 epochs. Applying Hinton's advice, one more hidden layer was stacked and this time added more hidden nodes across the hidden layers. Number of hidden nodes that is used was as follows 375, 200, 150, 75 for first, second, third and fourth hidden layers respectively. Table below is the performance of four hidden layers MLP.

Epochs	Test Accuracy	Training Accuracy
500	92.46%	99.96%
1000	92.53%	99.97%
1500	92.63%	99.97%
2000	92.39%	99.95%
2500	92.39%	99.95%
3000	92.39%	99.95%

Cost per test sample is 0.2466. It seems like as more hidden layers are stacked the cost per test sample seems to go down. Over-fitting is seen after 1500 epochs as test accuracy goes down while training accuracy stays around the same.

Confusion matrix at the end of 3000th epoch:

		Predicted Class			
Actual Class	60,074	199	297	6	17
	553	3,319	293	-	1
	6,140	168	223,544	-	-
	33	-	4	20	13
	15,324	33	24	2	964

Precision at the end of 3000th epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.732	0.892	0.997	0.714	0.969

Recall at the end of 3000th epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.991	0.768	0.973	0.286	0.059

Interestingly both training and test accuracy levels of four hidden layers MLP are higher than single hidden layer MLP's. The performance of the minority classes has also improved as depicted by above tables. Now cost-sensitive learning classification will be integrated to boost the performance of the minority classes.

4.6.4 Cost-sensitive learning in four hidden layers MLP for KDD Cup 1999 data

Cost-sensitive classification method mentioned in section 4.2.2 was used here as well. Details are given in section 4.2.2. This technique is incorporated into four hidden layer MLP. Table below is the performance of four hidden layers MLP with cost sensitive classification;

Epochs	Test Accuracy	Train Accuracy
500	92.47%	99.96%
1000	92.56%	99.97%
1500	92.64%	99.97%
2000	92.41%	99.95%
2500	92.32%	99.96%
3000	92.32%	99.97%

Cost per test sample is 0.2424 and test accuracy of 92.64% was achieved at the end of 1500 epoch. These values have been the best that had been achieved throughout this project.

Confusion matrix at the 1500th epoch:

		Predicted Class			
Actual Class	60,318	192	62	5	16
	574	3,261	327	-	4
	6,068	165	223,618	-	1
	31	-	2	23	14
	15,315	34	20	3	975

Precision at the 1500th epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.733	0.893	0.998	0.742	0.965

Recall at the 1500th epoch:

Class 1	Class 2	Class 3	Class 4	Class 5
0.995	0.783	0.973	0.329	0.0596

However, in the range of 1000-1300 epochs the network was performing well on the last class (minority) with a cost per sample of 0.24477. Below is the confusion matrix, precision and recall tables:

		Predicted Class			
Actual Class	59,905	205	460	6	17
	549	3,327	289	-	1
	6,150	170	223,532	-	-
	32	-	3	19	16
	15,276	33	27	2	1,009

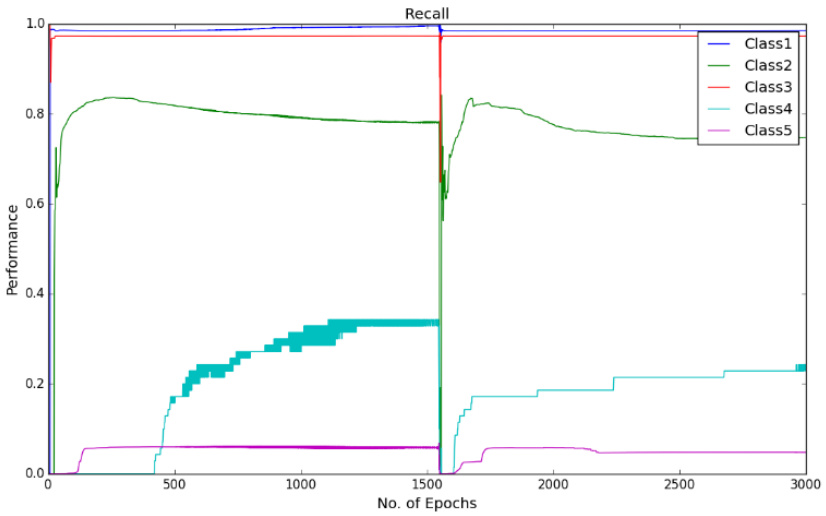
Precision

Class 1	Class 2	Class 3	Class 4	Class 5
0.731	0.891	0.997	0.704	0.967

Recall

Class 1	Class 2	Class 3	Class 4	Class 5
0.989	0.799	0.973	0.271	0.061

It is interesting that the neural networks for the class imbalance problem improves when more hidden layers are stacked, that is, if we build deeper networks, the performance on the minority classes gets better than with a shallower network. Throughout the iteration, the maximum correct number of classifications of minority class 5 has been 964, boosted to around 1000 correct when cost-sensitivity in the classification was incorporated. The performance of the minority class worsens after 1300 epoch, and after 1500 epochs the performance of the overall test accuracy starts to decline. Below is the graph of the recall for the classes.



Dropout used in cost-sensitive learning in four hidden layer MLP for KDD Cup 1999 data

4.6.5 Cost-sensitive learning with dropout in four hidden layers MLP for KDD Cup 1999

Finally, dropout regularisation is used with a cost-sensitive learning four hidden layer MLP to combat the over-fitting and investigate if the network performs. 2.5% of the noise level was added into the network by dropout. 2.5% is chosen because the use of dropout regularisation makes the learning a lot slower as seen in MNIST experiment in section 3.2. We varied learning rate and momentum in a similar way like done in our MNIST experiment. In the beginning of this experiment, learning rate of 0.8 and momentum of 0.2 was used. Then learning rate changed to 0.6, 0.2 and 0.05 stepwise, momentum changed to 0.4, 0.8 and 0.85 stepwise at the epoch of 500, 1000 and 1500 respectively. Below are the performance graphs of the network. Spikes and fluctuations are seen in the training cost curve, precision and recall curves particularly for the minority classes in figure 4.3.5a, 4.3.5b and 4.3.5c. This gives less confidence to use dropout in the network to make predictions.

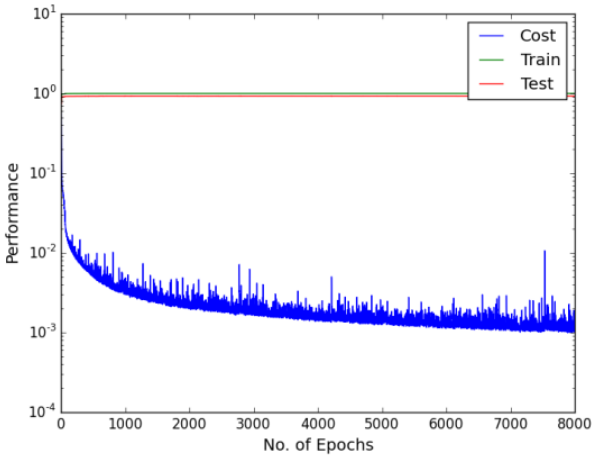


Figure 4.6.5a shows the cost, and training and test accuracies levels using Deep MLP network with dropout of 2.5%

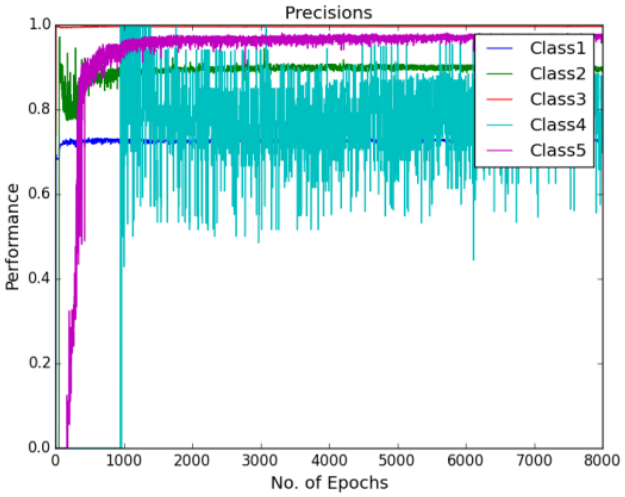


Figure 4.6.5b shows the precision curves of all the classes using Deep MLP network with dropout of 2.5%

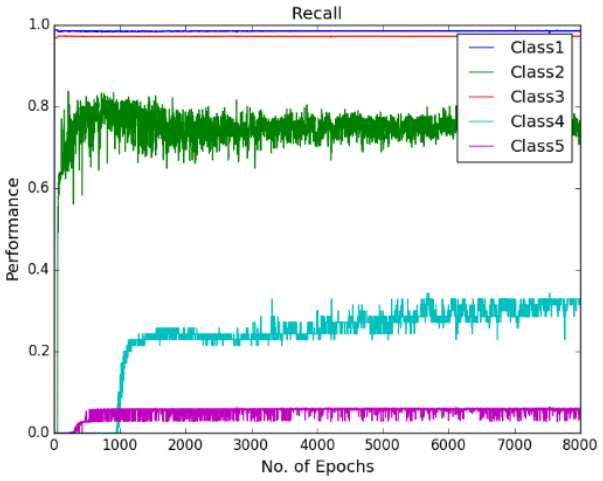


Figure 4.6.5b shows the recall curves of all the classes using Deep MLP network with dropout of 2.5%

Cost per test sample is 0.2477; it is calculated by the multiplying confusion matrix given below with cost matrix [106]. Below are the confusion matrix, recall and precision tables;

	Predicted Class				
Actual Class	59,655	210	710	5	13
	775	3,127	263	-	1
	6,181	141	223,530	-	-
	35	-	-	17	18
	15,343	28	4	-	972

Precision

Class 1	Class 2	Class 3	Class 4	Class 5
0.728	0.892	0.996	0.773	0.968

Recall

Class 1	Class 2	Class 3	Class 4	Class 5
0.985	0.751	0.972	0.243	0.059

There are other regularisers used in neural networks like weight decay L2 that deals with the overf-fitting. It will be interesting to see the comparison of dropout with other regularisers. Using dropout does not seem to give any advantage.

4.7 Discussion and Evaluation

This report addresses the problem of anomaly detection through proper classification. SMOTE, sampling approach to deal with the class imbalance problem can do well if features of the different classes are distinguishable. It will be interesting to see the performance of the data resampling by SMOTE on the binary classification to distinguish between the normal and intrusion connections in the KDD cup 1999 data.

Cost-sensitive learning of neural networks seems to be effective in dealing with the class imbalance. The overall accuracy of 92.64% turns out to be the best result achieved by us by cost-sensitive classification of deep four hidden layer neural network. In another research by Kukar et al [107], Adaptive learning rate and minimization of the misclassification costs were concluded to be more effective than the cost-sensitive classification (the method that is actually implemented in this project is cost-sensitive classification). It will be interesting to see the performance of the network with those two other cost-sensitive learning methods of adaptive learning rate and minimization of the misclassification costs.

Not only can cost-sensitive learning be adopted in network intrusion and classification problem but also in various other areas like in healthcare applications. For example the cost of misclassifying if a person is not likely to have a disease when they actually are likely to have one is more than the cost of misclassifying a person who is likely to have a disease but actually does not. We have applied cost sensitive learning here (like they do in healthcare) because here, like healthcare too, the cost of a false negative or misclassification can be huge for false positives and spark mass litigations in cyber breaches. Having to deal with a bit more false positives makes it worth it to have lower false negatives.

Cost-sensitive learning seemingly has worked quite well in most of the research projects when dealing with the class imbalance problem. In the research by Dalyac [108] the cost function called Bayesian cross entropy is introduced which tackles the problem of the image dataset using deep convolutional neural network. Bayesian cross entropy is simple modification of the equation of the cross entropy. In the Bayesian cross entropy, instead of assigning identical probability distribution to all the classes while trying to maximize the joint probability of occurrence, higher probability distribution to the minority class is assigned. This idea is very similar to the cost-sensitive learning. Below is the Bayesian cross entropy equation:

$$-\frac{1}{K*n}\sum_{i=1}^{n}\frac{\log(f(x_i,l_i,W))}{p(class(x_i))}$$, where K is the number of classes.

It turned out implementing any non-standard technique requires lot of work. If we had relied on a Deep Learning library like Pylearn2, the advantage would have been that we could have explored more techniques like Maxout, weight decay, etc. without worrying so much about the implementation, but the disadvantage would have been that we would not have had understood the implementation of techniques like Dropout extensively and may not have been able to implement highly customised changes.

While Deep Learning has shown remarkable success in the area of unstructured data like image classification, text analysis and speech recognition, there is very little literature on Deep Learning performed on structured/relational data which makes this investigation intriguing. After the extensive investigation carried out in this project, it does seem that Deep Learning has the potential to do well in the area of structured data. In [109] Deep Belief Networks have performed well (better than support vector machine and single hidden layer MLP) on the KDD Cup 1999 data with class imbalance. In this project it has been observed that if more hidden layers are stacked the performance on the minority class improves to some extent.

Theano is particularly challenging as it is difficult to debug errors since Theano uses compiled functions. It would have been nicer, cleaner and avoided a lot of repetitive code if we had implemented the code in Python as an Object Oriented Programming (OOP) language. OOP can scale quite nicely as the program complexity grows. You can think about a module as a dictionary that can store Python code that can be accessed with the '.' operator. Python also has another construct that serves similar purpose called a class. Classes can be thought of as a guide to the way the object should be structured. It is an easy way to group the similar variables and functions together. Each object belongs to the class and inherits the properties of the class but acts individually to other instances of the class. In [110] there are classes like Hidden Layer and MLP defined. The Hidden Layer for example defines how objects like parameters, weights and biases, and activation function behave. Hidden Layer class can be called by the dot operator and so instead of writing the whole equation codes for 2^{nd} and 3^{rd} hidden layers with the same set of functions used, we could have created the layers using a Hidden Layer class, making the object oriented programming approach neater and scalable. By introducing classes for layer activation function, the code would have been easier to read and change. Swapping activation function would equate to implementing a new class. This is the approach of Pylearn2, Keras and other Deep Learning libraries.

Aside from pylearn2, tensor flow and H20 are also good alternatives. H20 [111] can be used for Deep Learning in both Python and R. H20 has scalable, fast Deep Learning using mostly on the feedforward architecture. When using momentum updates for momentum training, H20 recommends using the Nesterov accelerated gradient method, which uses the nesterov accelerated gradient parameter. During training, the chance of oscillation or optimum skipping creates the need for a slower learning rate as the model approaches a minimum. As opposed to specifying a constant learning rate, learning rate annealing gradually reduces the learning rate to freeze into local minima in the optimization landscape (Zeiler, 2012). The unique advantage is that H20's implemented adaptive learning rate algorithm ADADELTA (Zeiler, 2012) automatically combines the benefits of learning rate annealing and momentum training to avoid slow convergence. This type of algorithm, called a deep autoencoder, has been used extensively for unsupervised, layer-wise pre-training of supervised Deep Learning tasks. It would have been interesting to see deep autoencoder in action.

It turned out that making dropout work was more difficult than actually implementing it. In section 3.2, MNIST experiment is described. It will be interesting to replicate the way Hinton conducted his MNIST experiment using dropout regularization. Hinton decays the learning of

very high initial value, increases momentum linearly and uses weight-norms. In this way the learning of the network may speed up to compensate for the regularisers like dropout slowing down the learning of the network. It will also be interesting to try a Maxout network in [112] as it shows the improvement in the performance of the dropout. It would have a been a bit easier if Deep Learning Python package like Pylearn2, Keras or other Deep Learning libraries are used. In this way we could have easily applied these techniques. Deep Boltzmann machines were also not used precisely because they are too slow.

It seems that if we are dealing with class imbalance problems then it is best to focus on the techniques that are specifically dealing with the class imbalance problem. On the other hand, Deep Learning shows the potential to optimise the results by that technique of stacking more hidden layers on the neural network.

Hinton in [113] has explicitly stated that dropout improves the performance of neural networks on supervised learning tasks in vision, speech recognition, document classification and computational biology. All these tasks have unstructured data. This questions if dropout actually works on the structured data like KDD 1999 Cup. Further experiments on structured data are required to solidify this tentative doubt.

An alternative to our method of using Multi-layer Perceptron can be utilizing Encoder-Decoder networks for structured data instead. Encoder-Decoder is a framework that aims to map highly structured input to highly structured output. This framework has been applied recently in machine translation where one language is translated by machine to another [114].

An alternative for handling class imbalance problem is using Deep Coding networks as they change parameters as context of data changes on another dataset (preferably not KDD 1999). This is very important for fraud and intrusion as fraudsters are continuously scheming new contexts and ways to deceive.

This project has shown that Deep Learning can be a powerful classification technique using categorical and continuous valued of the type typically collected by business processes. The expressive power of even relatively small neural networks has been seen in their ability to closely fit the large volume of training data. Deep Learning is clearly a powerful technique, and businesses may find many applications for it. However, it has also become clear through this project that considerable experience is required to get good results from Deep Learning. There are many architectures, techniques, and hyper-parameters that need to be carefully chosen before a model performs well on unseen data.

Insurance companies may gain much from applying Deep Learning techniques in many areas, such as proper classification and other methods mentioned in the previous sections. Many techniques are still recent and until the foundations of Deep Learning are better understood, or more general deep models are developed, businesses must be prepared to invest in gaining experience and a certain amount of trial and error should be carried out before usable results are obtained.

5. On-going Developments and Outlook

Overview

There are huge numbers of variants of deep architectures as it's a fast developing field and so it helps to mention other leading algorithms. The list is intended to be comprehensive but not exhaustive since so many algorithms are being developed [115] [116].

5. Deep High-order Neural Network with Structured Output (HNNSO).
6. Deep convex network.
7. Spectral networks
8. noBackTrack algorithm to solve the online training of RNN (recurrent neural networks) problem
9. Neural reasoner
10. Reccurrent Neural Networks
11. Long short term memory
12. Hidden Markov Models
13. Deep belief network
14. Convolutional deep networks
15. LAMSTAR are increasingly being used in medical and financial applications. LAMSTAR is Large memory storage and retrieval neural networks.
16. Deep Q-network agent. Google DeepMIND uses this and it is based on reinforcement learning which is a major branch of psychology, aside from evolution.

Additionally, Fuzzy logic models can also be used with other models such as decision trees, hidden Markov and Bayesian and artificial neural networks to model complicated risk issues like policyholder behaviours. A risk assessment and decision-making platform for ratemaking built on a fuzzy logic system can provide consistency when analyzing risks with limited data and knowledge. It allows people to focus on the foundation of risk assessment, which involves the cause-and-effect relationship between key factors as well as the exposure for each individual risk. Rather than a direct input for the likelihood and severity of a risk event, it supports human reasoning from the facts and knowledge to the conclusion in a comprehensive and reliable manner [117].

Hyper-parameter turning

One issue however with Deep Learning is trying to find the hyper-parameters that are optimum. The possible space for consideration is very large and it is difficult and computationally intensive to understand each hyper parameter in depth. We also cannot write down the actual formula for the response surface that we are optimizing.

One potential solution which the author of this report identifies is the possible use of genetic algorithm to find optimal hyper parameters. Genetic algorithms are already used on GLMs on R 'glmulti' package to select optimum GLM equation as per a given criteria usually Akaike Information Criterion or Bayesian Information Criterion.

Moreover, another algorithm has been used to optimize both structure and weights of a neural network. ES HyperNEAT is Evolving Substrate Hyperbolic Neuroevolution Of Augmenting Topologies developed by Ken Stanley. It uses a genetic algorithm to optimize both the structure and weights of a neural network. Following from this, maybe ES HyperNEAT framework can be extended to Deep Learning so that genetic al genetic algorithm can optimize both the structure and weights of the neural networks in Deep Learning as well [118].

Another problem is over fitting. Aside from Dropout, Machine unlearning can also be used to solve this. Machine unlearning puts a new layer of small number of summations between the training data and the learning algorithm so that the dependency between these two is eliminate. Now the learning algorithms depend only on the summations instead of the individual data from which over-fitting can arise more easily. No retraining of remodeling is required [119].

There are many other hyperparameter tuning algorithms now [120]: Spearmint, Sequential Model-based Algorithm Configuration (SMAC), Tree Structured Parzen estimator (TPE) and so on. These three can be implemented using 'Hyperopt' library in Python. Research shows that overall, SPEARMINT performed best for the low-dimensional continuous problems [121]. For the higher-dimensional problems, which also include conditional parameters, SMAC and TPE performed better [122].

The machine intelligence approach advocated by Ayasdi (startup company founded by Stanford professors) combines topology with machine learning to achieve data-driven insights instead of hypothesis driven insights. Machine learning on its own have significant limitations. Clustering, for example, requires an arbitrary choice of clusters which the analyst has to specify (hyper parameter tuning). With dimensionality reduction techniques, the danger is on missing the subtle insights included in the data that can potentially prove to be very useful to the analysis. Including topology with machine learning overcomes these drawbacks effectively [123].

The topology visualizations capture the subtle insights in the data while also representing the global behavior of the data. From the nodes identified by the topology network diagrams from the data, clusters are identified and each cluster is fit onto a model that fits it more properly so that instead of a one-size-fit-all model, different models are applied to different regions of data for maximum predictive potency [124]. Perhaps this machine intelligence approach can also be applied on Deep Learning for optimizing hyper-parameters tuning.

Dark Knowledge [125][126]

A very interesting recent development is Hinton's use of 'dark knowledge' to characterise Deep Learning. Hinton claims that 'dark knowledge' is what most Deep Learning models actually learn. Hinton et al shows that classifiers built on softmax function have a lot of hidden information contained within them as well as the correlations in the softmax are very informative and mysterious. For instance, when training to classify man, woman and alien, the correlation between man and woman will always have more correlation than with man

and alien or woman and alien as man and woman look more similar to each other being humans than an alien.

Hinton also highlights the research undertaken previously by Caruana et al [127] and tries to propagate it further. The main idea is that a complex model can be used to train a simpler model. Once a core Deep Learning model is trained, we do not need to train models again on the huge datasets. Instead, we can just posit the new 'student' model to mimic the 'teacher' model. This can lead to massive breakthroughs as far as our imagination can take us in applications as many Internet of Things components and smartphones do not have the memory or processing power to run their own Deep Learning models on the data that they have captured. They can instead rely on the 'teacher' model and still give us accurate results. Ofcourse, the teacher model needs to be accurate for student model to be accurate as well and we'd still need large unlabelled data set to train the teacher model.

A very effective way to increase performance and accuracy of Deep Learning as well as any machine learning algorithm is to train many different models on the same data and then average their predictions. This ensemble of models, termed the teacher model is too time consuming, memory expensive and cumbersome to be practiced by large number of users. Once the teacher model is trained, the student model mimics the teacher model via 'distillation' of the knowledge. Instead of training the student model on the labelled data, we can train it against the posteriors of the teacher model. The output distributions from the softmax function is utilized which contains the dark knowledge. However, as these posterior estimates suffer from low entropy, transformation from logarithm that 'raises the temperature' is necessary to activate and transfer the dark knowledge to the student model. Softened outputs reveal the dark knowledge in the ensemble as the soft targets contain almost all the knowledge.

Another way to harness the dark knowledge contained in Deep Learning is through specialist networks. Specialist network is a model that is trained to specifically identify and classify certain specialist classes. For instance, while a Deep Learning model can identify animals from humans, specialist network will aim to classify mammals from other animals and so on. A sufficient number of specialists can be trained on the data and their classifications averaged to arrive at accurate results rather than relying on a single generic Deep Learning model especially when data has a lot of classes and suffers heavily from class imbalance problems.

In Hinton's revised approach, certain numbers of adjustments are made. Each data point is assigned a target that matches the temperature adjusted softmax output. These softmax outputs are then clustered multiple times using k-means and the resultant clusters indicate easily confuseable data points that come from a subset of classes. Specialist networks are then trained only on the data in these clusters using a restricted number of classes. They treat all classes not contained in the cluster as coming from a single "other" class. These specialist networks are then trained using alternating one-hot, temperature-adjusted technique. The ensemble method constructed by combining the various specialist networks creates benefits for the overall network.

6. Limitations of Deep Learning

While Deep Learning has shown itself to be very powerful in applications, the underlying theory and mathematics behind it remains obscure and vague. Deep Learning works, but theoretically we do not understand much why it works. Some leading machine learning theorists like Vladimir Vapnik criticise Deep Learning for its ad-hoc approach that gives a strong flavour of brute force rather than technical sophistication. Deep Learning is not theory intensive; it is empirical based more (hence causing battle of viewpoints between empiricism and realism) and relies on clever tweakings [128]. This is why 'Deep Learning' is viewed as a black box and why we preferred to use Theano instead of other packages as it allowed us better view inside the workings of the model (which is still not enough to fully overcome the black box criticism).

Furthermore, there are no theoretical guarantees with Deep Learning. There is no theoretical proof that once we start randomly at one point and use stochastic gradient descent that we will achieve convergence and not be trapped in local minima. Other machine learning algorithms like logistic regression is guaranteed to converge over time and linear regression generates exact solutions [129].

We believe that Deep Learning, for its fundamental composition, resembles complexity science tools. This is as no mathematical maturity of advanced level is sought in Deep Learning unlike other many machine learning algorithms. Heuristics based learning and tweaks and experiments are engineering in Deep Learning on huge amounts of data with microscopic focus instead of generic focus. Thus, simple rules of Deep Learning interact with sophisticated data and break them into microscopic segments and then recombine them as outputs. Complexity Science adheres that complex systems can arise from massive iterations and interactions between agents and data following very simple rules and heuristics. Deep Learning starts from the bottoms-up and neural networks are even differentiable from top to bottom via back-propagation which makes it possible to trace the contribution of each input to the output and show which inputs will move the output most if slightly increased or decreased.

Also it must be emphasized that with sufficiently large data sets, the risk of over-fitting is low. Plus, we have so many algorithms like dropout for handling over-fitting. Validating the algorithm on testing data provides further accuracy and sanity checks. Parallel computing and increase in computing size means that we do not have to have tightly closed hypothesis to check on small data as done in statistics traditionally. We can follow up now with many hypothesis simultaneously with empirical experiments and go more microscopic on high dimensional data sets. When we arrive at the frontier of our formal understandings, intuition based Deep Learning can offer a path forward. There is always the chance that with continued research, we might be able to decode these black boxes in the future.

Deep Learning has been followed around with great hype by the media and spectators. First, the wave of hype posits Deep Learning as an 'apocalyptic' algorithm but recently this has turned around with negative opinions. It is over-reaction on both up and down sides by the

media, but researchers are aware that first a new algorithm gains prominence from its ability to add unique value addition and that over time, further research is bound to show faults in algorithms. This is not a weakness but is a strength as researchers can then focus on overcoming those specific limitations to make such algorithms more powerful in the future.

Recent research papers highlighting limitations of Deep Learning are [130]:

- 'Intriguing Properties of Neural Networks' by Google's Christian Szegedy and others.
- 'Deep Networks are Easily Fooled' by Anh Nguyen at University of Wyoming

The first paper shows that one can subtly alter images and these changes cannot be detected by humans yet these lead to misclassification in a trained Convolutional Neural Network. In giving this case its proper context, it is important for us to note that almost machine learning algorithms are susceptible to such adversarial chosen examples as done here. Value of a particular feature can be deliberately set very high or very low to induce misclassification in logistic regression. If some features have a lot of weight, even minor change can lead to misclassification. Similarly for decision tress, a single binary feature can be used to direct an example at the wrong partition by simply switching it at the final layer.

Hence, the proper context is that even machine learning models with theoretical guarantees and deep mathematical formulations are susceptible so such manipulations so we should not be shocked that Deep Learning too is vulnerable.

The second paper by Anh Nguyen discusses the opposite case. They create gibberish images to train gradient descent which then gets classified strongly into classes even though they should not have been classified in any class. This too is a genuine limitation of Deep Learning, but it's a drawback for other main machine learning algorithms as well.

7. Strategic Notes

The more data we have, the more likely we are to drown in it. ~ Nassim Nicholas Taleb; Fooled by Randomness (2001)

It is no accident that serial best experts preferred qualitative, context-specific explanation while generally using models and statistics to beat everyone else at judging (herds use models and stats).

Unfortunately, not all datasets can be made large just because storage is cheap and our storage items lack reasonable privacy controls. Rare diseases remain rare and rare events remain rare. To add maturity to our identification of such rare events, qualitative profiling and insight is compulsory and not just modeling on its own. Just because something formerly couldn't be measured didn't make it irrelevant. Recall Kant's, Jung's, Berlin's, Einstein's and Goethe's "beyond analysis" critique and advice: Intuition—experience and familiarity—links knowledge to understanding [131].

Along the same vein, In The Black Swan, Taleb describes "Mediocristan,"(Quadrants I and II) as a place where Gaussian distributions are applicable. By contrast, he calls Quadrant IV "Extremistan." It is Extremistan where we are interested for understanding complex systems. Actuaries like to build their models on the Gaussian distribution we are perhaps avoiding professional expertise by fooling ourselves by retreating to the comfort and safety of the womb of Mediocristan instead of facing Extremistan in all its unknown mystery and ambiguity [132].

To avoid being ambiguity averse, we can train ourselves to explore the unexplored. As actuaries, perhaps we could make a greater effort to uncover hidden patterns. Actuarial and statistical modeling is a double-edged sword. If applied correctly, it is a very powerful and effective tool to discover knowledge in data, but in the wrong hands it can also be distorted and generate absurd results. It is not only our results that can be absurd, but our risk-averse and ambiguity-averse mentalities as well [133]. As Voltaire said "doubt is not a pleasant condition but certainty is absurd."

Aristotle explains this further: "*It is the mark of an instructed mind to rest satisfied with that degree of precision which the nature of the subject limits, and not to seek exactness where only an approximation of the truth is possible.*"

This teaches us that we should be aware that precision implies confidence. We must be very alert to not fall into this trap. While point estimates are often required (we have to quote and file a specific premium), there are many cases where ranges of estimates are more appropriate. While statistical techniques can sometimes be used to generate precise confidence intervals, mostly statistical rigor is not possible or even necessary for emerging risks. By discussing a range of estimates, actuaries can provide more value to their stakeholders by painting a more complete picture of the potential impacts of decisions related to emerging liabilities [134].

Finally, we must ensure that actuarial output highlights fundamental questions at hand to stakeholders instead of confusing them with complicated numbers and lack of decisiveness. There is obviously a premium to be established but the management running the company does not care what the actual premium is—they need to know the likely impacts of that premium on the business. From a financial perspective we should avoid saying that we've priced for a certain margin because that exact margin is, in the end, going to be exactly wrong! The better approach would be to explain the range of possible outcomes and the impacts of each [135]. As Nassim Nicholas Taleb explains: "There are so many errors we can no longer predict, what you can predict is the effect of the error on you!"

8. Conclusion

Our technical conclusion is that while dropout with techniques worked for unstructured data, adding dropout to structured data does not seem to give any improvement. For strategic conclusion, this is an exciting and dangerous time for actuarial profession. The proliferation of big data, Deep Learning, and machine learning techniques and evolving of emerging risks at lightning speed has resulted in problem-solving means and aptitude which we have been previously unable to tackle, but the same advances have brought analytical professionals from other disciplines into spaces traditionally led by actuaries as well as its own share of technical challenges.

9. References

1. Jack Clark (Feb 3, 2015); Bloomberg Business,; "I'll be back: The Return of Artificial Intelligence".
2. Will Knight (May 31, 2015); MIT Technology Review; "Deep Learning catches on in new industries, from fashion to finance".
3. Yoshua Bengio, University of Montreal; "Learning deep architectures for AI".
4. IBM Website; Smarter Planet; improve decision making with content analytics
5. Jeff Heaton, SOA Predictive Analytics and Futurism Newsletter; Issue 9, 2014. An Introduction to Deep Learning
6. Sean Lorenz (June 2016); Domino Datalab; Deep learning with h20.ai
7. PwC; March 2016; Top Issues: AI in Insurance; hype or reality?
8. Dugas et al; Statistical Learning Algorithms Applied to Automobile Insurance Ratemaking
9. Stanford Natural Language Processing Group; available at: http://nlp.stanford.edu/software/
10. CAS Ellingsworth and Balakrishnan: 2008. Practical text mining in insurance
11. Stanford Natural Language Processing Group; available at: http://nlp.stanford.edu/software/
12. PwC; March 2016; Top Issues: AI in Insurance; hype or reality?
13. Lentz,W. GenRe Research (Nov 2013); Predictive Modeling—An Overview of Analytics in Claims Management
14. Entilic website; science and solutions
15. SOA; The Actuary Magazine December 2013/January 2014 – Volume 10, Issue 6, Ferris et al "Big Data".
16. Forbes, Steven Bertoni (Nov 11, 2014); Goldman Sachs leads $15 million investment in tech start up Kensho
17. Bloomberg Business, Kelly Bit (May 20, 2015): The $10 Hedge Fund Supercomputer that's sweeping Wall Street".
18. MetaMind company's website
19. Rozados, IV, Tjahjono, B, 2014; 6th International conference on operations and supply chain management, Bali.
20. IBM [a], NA. "Infographics & Animations", available at:http://www.ibmbigdatahub.com/infographic/four-vs-big-data
21. IBM [b], NA;"What is MapReduce" available at: http://www01.ibm.com/software/data/infosphere/hadoop/mapreduce/
22. Techterms, 2013. "NoSQL"
23. IBM White paper2013 : Harnessing the power of big data and analytics for insurance
24. https://en.wikipedia.org/wiki/NewSQL#cite_note-2
25. ASTIN Big Data/Data Analytics Working Party – Phase 1 Paper- April 2015
26. StackIQ white paper Capitalizing on Big Data Analytics for the Insurance Industry
27. Bharal, P and Halfon, A. ACORD and MarkLogic (2013). Making Sense of Big Data in Insurance
28. Ibid
29. http://arstechnica.com/cars/2016/06/tesla-model-s-floats-boat-video/ Anthony, S. June 20,2016. Arstecnica.com "Tesla Model S can be used as a boat in a pinch, Elon Musk confirms".
30. Lloyds and Praedicat, Innovation Series 2015. Emerging Liability Risks: Harnessing big data analytics.
31. Extract on KDD;*Cost-based Modeling and Evaluation for Data Mining With Application to Fraud and Intrusion Detection: Results from the JAM Project* by Salvatore J. Stolfo, Wei Fan, Wenke Lee, Andreas Prodromidis, and Philip K. Chan. https://kdd.ics.uci.edu/databases/kddcup99/task.html
32. Ibid
33. Swati Paliwal, Ravindra Gupta ," Denial-of-Service, Probing & Remote to User (R2L) Attack Detection using Genetic Algorithm", International Journal of Computer Applications ,Volume 60– No.19, December 2012.
34. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
35. Ni GAO, Ling GAO, Quanli Gao, Hai Wang. An Intrusion Detection Model Based on Deep Belief Networks. Second International Conference on Advanced Cloud and Big Data. 2014

36. Safaa et al; Evaluation of Different Data Mining Algorithms with KDD CUP 99 Data Set Journal of Babylon University/Pure and Applied Sciences/ No.(8)/ Vol.(21): 2013
37. R. Longadge and S. Dongre. Class Imbalance Problem in Data Mining Review. ArXiv e-prints, May 2013.
38. Charles Elkan, Results of the KDD'99 Classifier Learning Contest, September 1999, URL [http://cseweb.ucsd.edu/~elkan/clresults.html]
39. Note in this report class 0,1,2,3,4 have renamed as class 1,2,3,4,5 throughout particularly in the plots
40. Charles Elkan, Results of the KDD'99 Classifier Learning Contest, September 1999, URL [http://cseweb.ucsd.edu/~elkan/clresults.html]
41. B. Pfahringer. Winning the KDD99 classification cup: Bagged boosting. SIGKDD explorations, 1(2):65–66, 2000.
42. Tan, Pang-Ning, Steinbach, Michael, and Kumar, Vipin Introduction to data mining.
43. Charles Elkan, Results of the KDD'99 Classifier Learning Contest, September 1999, URL [http://cseweb.ucsd.edu/~elkan/clresults.html]
44. Tan, Pang-Ning, Steinbach, Michael, and Kumar, Vipin Introduction to data mining.
45. Ibid
46. S.S. Haykin, Neural networks: a comprehensive foundation. Upper Saddle River, N.J.: Prentice Hall, 1999.
47. Charles Elkan, 8/5/99, URL http://cseweb.ucsd.edu/~elkan/tabulate.html
48. S.S. Haykin, Neural networks: a comprehensive foundation. Upper Saddle River, N.J.: Prentice Hall, 1999.
49. Ibid
50. J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio. "Theano: A CPU and GPU Math Expression Compiler". Proceedings of the Python for Scientific Computing Conference (SciPy) 2010. June 30 - July 3, Austin, TX
51. S.S. Haykin, Neural networks: a comprehensive foundation. Upper Saddle River, N.J.: Prentice Hall, 1999.
52. https://github.com/mbeissinger/intro_deep/blob/master/MLP_theano.ipynb
53. Glorot, Xavier; Bordes, Antoine; Bengio, Yoshua; Deep Sparse Rectifier Neural Networks 2013
54. Alexandre Dalyac. Tackling Class Imbalance with Deep Convolutional Neural Networks. Imperial College London. 24 September 2014
55. S.S. Haykin, Neural networks: a comprehensive foundation. Upper Saddle River, N.J.: Prentice Hall, 1999.
56. http://deeplearning.net/tutorial/mlp.html
57. Bengio, Yoshua; Learning Deep Architectures for AI Foundations and Trends in Machine Learning Vol. No. 1 (2009) 1-127 2009
58. Michael Nickelson, August 2015, Neural Networks and Deep Learning, URL http://neuralnetworksanddeeplearning.com/chap1.html
59. Glorot, Xavier; Bordes, Antoine; Bengio, Yoshua; Deep Sparse Rectifier Neural Networks 2013
60. Ibid
61. I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In Proceedings of the 30th International Conference on Machine Learning, pages 1319– 1327. ACM, 2013.
62. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. JMLR, 2014.
63. Mihaela Rosca, Networks with emotions; An investigation into deep belief nets and emotion recognition, Department of Computing, Imperial College London. June 2014
64. S.S. Haykin, Neural networks: a comprehensive foundation. Upper Saddle River, N.J.: Prentice Hall, 1999.
65. Bengio, Yoshua; Learning Deep Architectures for AI Foundations and Trends in Machine Learning Vol. 2, No. 1 (2009) 1-127 2009
66. Bengio, Y., Courville, A. and Vincent, P., 2013. Representation learning: A review and new perspectives. Pattern Analysis and Machine Intelligence, IEEE Transactions on, 35(8), pp.1798-1828.
67. Ibid

68. Hinton, G.E., Srivastave, N., Krizhevsky, A.,Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580, 2012
69. The three papers are:
    a) Hinton, G. E., Osindero, S. and Teh, Y., A fast learning algorithm for deep belief nets Neural Computation 18:1527-1554, 2006
    b) Yoshua Bengio, Pascal Lamblin, Dan Popovici and Hugo Larochelle, Greedy Layer-Wise Training of Deep Networks, in J. Platt et al. (Eds), Advances in Neural Information Processing Systems 19 (NIPS 2006), pp. 153-160, MIT Press, 2007
    c) Marc'Aurelio Ranzato, Christopher Poultney, Sumit Chopra and Yann LeCun, Efficient Learning of Sparse Representations with an Energy-Based Model, in J. Platt et al. (Eds), Advances in Neural Information Processing Systems (NIPS 2006), MIT Press, 2007
70. Hinton, G. E., Osindero, S. and Teh, Y., A fast learning algorithm for deep belief nets Neural Computation 18:1527-1554, 2006
71. S.S. Haykin, Neural networks: a comprehensive foundation. Upper Saddle River, N.J.: Prentice Hall, 1999.
72. Deeplearning4j - Open-source, distributed deep learning for the JVM . [ONLINE] Available at: http://deeplearning4j.org/restrictedboltzmannmachine.html.
73. S.S. Haykin, Neural networks: a comprehensive foundation. Upper Saddle River, N.J.: Prentice Hall, 1999.
74. Deeplearning4j - Open-source, distributed deep learning for the JVM .
75. Ibid
76. If anyone wants to try a multimodal RBM in Matlab here is the link https://github.com/sFunzi/RelSim by Son Tran, PhD Student at City University's Machine Learning group.
77. Srihari, S. University at Buffalo. Regularization in Neural Networks.
78. Hinton, G.E., Srivastave, N., Krizhevsky, A.,Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580, 2012
79. https://www.kaggle.com/c/challenges-in-representation-learning-the-black-box-learning-challenge/forums/t/4491/dropout-maxout-and-deep-neural-networks
80. Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.A., 2010. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. The Journal of Machine Learning Research, 11, pp.3371-3408.
81. Wan, L., Zeiler, M., Zhang, S., Cun, Y.L. and Fergus, R., 2013. Regularization of neural networks using dropconnect. In Proceedings of the 30th International Conference on Machine Learning (ICML-13) (pp. 1058-1066).
82. N. V. Chawla, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer. SMOTE: Synthetic Minority Oversampling Technique. Journal of Artificial Intelligence Research, 16:321–357, 2002.
83. Kukar, M. and Kononenko, I., 1998, August. Cost-Sensitive Learning with Neural Networks. In ECAI (pp. 445-449).
84. Pedregosa, F. et al.,Scikit-learn: Machine Learning in Python , JMLR 12, pp. 2825-2830, 2011.
85. F. Bastien, et al "Theano: new features and speed improvements". NIPS 2012 deep learning workshop.
86. PhD Student at City University of Machine Learning Research group Srikanth gave us this information.
87. J. Bergstra, et al "Theano: A CPU and GPU Math Expression Compiler". Proceedings of the Python for Scientific Computing Conference (SciPy) 2010. June 30 - July 3, Austin, TX
88. https://groups.google.com/forum/#!forum/theano-users
89. http://yann.lecun.com/exdb/mnist/
90. Hamid, et al, 2012. Solving Local Minima Problem in Back Propagation Algorithm Using Adaptive Gain, Adaptive Momentum and Adaptive Learning Rate on Classification Problems. In International Journal of Modern Physics: Conference Series (Vol. 9, pp. 448-455). World Scientific Publishing Company.
91. Bishop, Christopher; Pattern Recognition and Machine Learning Springer 2010

92. Hinton, G.E., Srivastave, N., Krizhevsky, A.,Sutskever, I., and Salakhutdinov, R. R. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580, 2012
93. Ibid
94. Varying learning rate and momentum stepwise over the 3000 epochs
95. Training cost of 0.183 was not achieved within 8000 epochs in experiment B
96. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html
97. Charles Elkan, 8/5/99, URL http://cseweb.ucsd.edu/~elkan/tabulate.html
98. https://github.com/mbeissinger/intro_deep/blob/master/MLP_theano.ipynb
99. https://groups.google.com/forum/#!forum/theano-users
100. http://wwwen.uni.lu/snt/research/sigcom/computer_vision_lab/costcla_a_cost_sensitive_classification_library_in_python
101. Kukar, M. and Kononenko, I., 1998, August. Cost-Sensitive Learning with Neural Networks. In ECAI (pp. 445-449).
102. Charles Elkan, Results of the KDD'99 Classifier Learning Contest, September 1999, URL [http://cseweb.ucsd.edu/~elkan/clresults.html]
103. S.S. Haykin, Neural networks: a comprehensive foundation. Upper Saddle River, N.J.: Prentice Hall, 1999.
104. Charles Elkan, Results of the KDD'99 Classifier Learning Contest, September 1999, URL [http://cseweb.ucsd.edu/~elkan/clresults.html]
105. Hinton et al. 2012. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580
106. Charles Elkan, Results of the KDD'99 Classifier Learning Contest, September 1999, URL [http://cseweb.ucsd.edu/~elkan/clresults.html]
107. Kukar, M. and Kononenko, I., 1998, August. Cost-Sensitive Learning with Neural Networks. In ECAI (pp. 445-449).
108. Dalyac, A. September 2014. Tackling Class Imbalance with Deep Convolutional Neural Networks. Imperial College London.
109. Ni GAO, Ling GAO, Quanli Gao, Hai Wang. 2014. An Intrusion Detection Model Based on Deep Belief Networks. Second International Conference on Advanced Cloud and Big Data.
110. http://deeplearning.net/tutorial/mlp.html
111. Candel et al. Deep Learning with H20. June 2016 5th Edition.
112. I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In Proceedings of the 30th International Conference on Machine Learning, pages 1319– 1327. ACM, 2013.
113. Hinton, et al. 2012. Improving neural networks by preventing co-adaptation of feature detectors. arXiv:1207.0580,
114. Kurach,Karol, Andrychowicz, Marcin and Sutskever,Ilya. "Neural Random-Access Machines." arXiv:1511.06392 (2015).
115. Mayo M, Larochelle H (Oct 2015) KDNuggets.com. Top 5 arXiv Deep Learning Papers explained.
116. Mayo M, Larochelle H (Jan 2016) KDNuggets.com. 5 more arXiv Deep Learning Papers explained.
117. Shang K, Hossen Z, (2013); CAS/CIA/SOA Joint Risk Management Section; Applying Fuzzy Logic to Risk Assessment and Decision-Making
118. Risi, S. and Stanley, K. University of Central Florida; The ES-HyperNEAT Users Page
119. Cao and Yang, 2015. IEEE symposium on security and privacy pgs 463-480. Towards making systems forget with machine unlearning.
120. Katharina Eggensperger et al;Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters
121. J. Snoek, H. Larochelle, and R.P. Adams. Practical Bayesian optimization of machine learning algorithms. In Proc. of NIPS'12, 2012.
122. C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In Proc. of KDD'13, 2013.
123. Ayasdi: Topology & Topological Data analysis
124. Ibid
125. Dark Knowledge: Hinton et al; Google Inc. Presentation.

126. Distilling the Knowledge in a Neural Network: Hinton et al: arXiv:1503.02531v1 [stat.ML] 9 Mar 2015
127. Caruna et al: Do deep nets really need to go deep? arXiv:1312.6184v7 [cs.LG] 11 Oct 2014
128. Kdnuggets Oct 2015. Lipton, Z.C "Does Deep Learning come from the Devil?"
129. Kdnuggets Jul 2015. Lipton, Z.C**:"**Deep Learning and the Triumph of Empiricism"
130. Kdnuggets Jan 2015. Lipton, Z.C: "(Deep Learning's Deep Flaws')s Deep Flaws"
131. Werther; SOA 2013; Recognizing When Black Swans Aren't: Holistically Training Management to Better Recognize, Assess and Respond to Emerging Extreme Events
132. Mills, A. SOA Predictive Analytics and Futurism Newsletter; Issue 1, 2009. Should Actuaries Get Another Job? Nassim Taleb's Work And Its Significance For Actuaries
133. Ibid
134. Hileman, G. SOA Predictive Analytics and Futurism Newsletter; Issue 9, 2014. "Roughly Right".
135. Ibid

Biography of the Authors

Syed Danish Ali is a Senior Consultant at SIR consultants, a leading actuarial consultancy in the Middle East and South Asia. He is also a graduate of University of London in Sociology and Career ambassador of Institute and Faculty of Actuaries (IFoA UK). He has more than 60 publications across a range of international platforms and is a regular contributor to various research activities in IFoA, CAS, SOA and International Actuarial Association including the ASTIN journal. His non-actuarial publications include articles in Banking, Quantitative Finance, Data Science, International Relations & Foreign Policy, Astrophysics, Philosophy and Sociology. He can be contacted at sd.ali90@ymail.com

Rahul Ahuja is a member of Institute and Faculty of Acutaries UK, has Masters in Data Science from Cass Business School of City University UK and multiple years of actuarial consulting experience. He can be contacted at rahul.ahuja@live.com