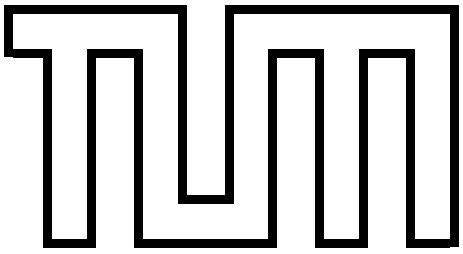


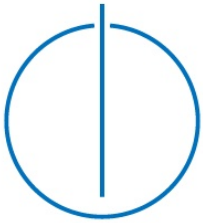
Technische Universität München

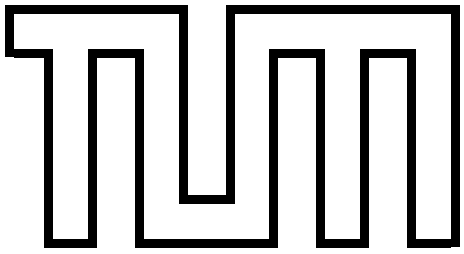
Fakultät für Informatik

Master's Thesis in Informatik

Leveraging Deep Learning Solutions for Predictive Maintenance of Industrial Datasets

Rupam Bhattacharya





Technische Universität München

Fakultät für Informatik

Master's Thesis in Informatik

Leveraging Deep Learning Solutions for Predictive Maintenance of Industrial Datasets

Verwendung von Deep Learning zur Praediktiven Wartung von Industrieller Datensatz

Author: Rupam Bhattacharya

Supervisor: Prof. Dr. Jan Křetínský

Advisor: Prof. Dr. Jan Křetínský

Submission: 15.10.2017

I assure the single handed composition of this master's thesis only supported by declared resources.

München, 15.10.2017

(Rupam Bhattacharya)

Acknowledgements

This Master thesis would not be possible without the support of many people. First and foremost, I would like to thank my thesis supervisor Prof. Dr. Jan Křetínský for providing me the opportunity to work under his supervision. I am immensely grateful to him in formulating this thesis topic and providing me with a perfect balance of guidance and freedom. I am also thankful to my wonderful team at BMW Innovation Lab (FG-96) in Munich - Kay Wünsche, Boulos El Asmar, Marc Kamradt and Mohammed Zahra who have helped me in providing useful insights including handful of tricks to train the models as well as constructive feedback about my work. I am thankful to them for their understanding and allowing me to be flexible. I am also thankful to the deep learning meetup community in Munich for organizing so many informative sessions which helped me to enhance my learning curve.

I am thankfuI to my cousins Ritamshirsha Choudhuri and Shouvik Bhattacharya for their valuable inputs. I am also thankful to my dear friends - Santanu Mohanta, Debsubhra Chakraborty, Saion Chatterjee, Bhaskar Chatterjee, Debdatta Dey, Shubhankar Chowdhury, Arijeet Panda, Antonio Ragagnin and Monika Machunik for their constant support like proof reading my work and motivation especially during the hard times. Lastly, I am also thankful to my family - my parents Dr Ramendu Kumar Bhattacharya and Samita Bhattacharjee and my brother Soham Bhattacharya for believing in my abilities.

Abstract

Predictive maintenance which is an age old problem, have been gaining attention of late due to the popularity of Internet of Things and applications of machine learning. Over the past few years, deep learning solutions have produced state of the art results in various domains. This thesis aims to provide solutions for providing predictive maintenance of batteries in industrial machines, in a way that both maintenance and repair activities can be predicted beforehand. It leverages deep learning solutions by exploring the application of sequence learning to represent the failure profiles and understand the patterns that lead to failures in batteries. It tries to address these problems by using recurrent neural networks.

As this work is in collaboration with BMW FG-96 Innovation Lab, we aspire to extend the effectiveness of Deep Architectures to real-time scenarios where accuracy and reliability is a priority.

Contents

Chapter 1

Introduction

"The brain's long-term memory has often been compared to an archive, but that's not its primary purpose. Instead of faithfully recording the past, it keeps rewriting history. Recalling an event in a new context can lead to new information being inserted in the memory. Coaching of eyewitnesses can cause people to reconstruct their memory so that no trace of the original is left."[1].

Artifical intelligence (AI), which is a term coined by John McCarthy, an American computer scientist, in 1956 at The Dartmouth Conference, has come a long way. Going as far back as to the Greek antiquity, AI went from fiction to very plausible reality in 1950 when Alan Turing came up with the idea of machines that think. He proposed Turing test to determine if a computer can actually think like a human. The rise of the personal computer in the 1980s sparked even more interest in machines that think. The success of Deep Blue, the IBM chess program that beat Garry Kasparov in the 1990s and the success of AlphaGo program more recently in 2016, when it beat a human professional Go player for the first time, have contributed to evolution of artifical intelligence. AI now has become one of the most booming research topics and is shaping our lives in many different ways starting from weather forecasts, email spam filtering, google's search predictions to biomedical applications. It has also led to emergence of autonomous driving, speech recognition, object detection, computer vision and many other research fields.

The last few years have also seen the emergence of big data, where almost 80 percent of the world's data has been generated over the last 3 years. The growth of internet and abundance of documents produced by the humanity has led to enormous growth of the amount of unstructured data in particular, which led to the need of intelligent systems to automatically extract knowledge from it.

Machine learning, a research area of computer science, which enables computers to learn without being explcitly programmed, has been instrumental for exploring the potential hidden in big data. Extracting value from big and disparate data sources with far less reliance on human direction, it is able to discover and display the patterns buried in the data. There are three types of machine learning algorithms: Supervised Learning, in which data sets are labeled so that patterns can be detected and used to label new data sets; Unsupervised Learning, in which data sets aren't labeled and are sorted according to similarities or differences and Reinforcement Learning, in which data sets aren't labeled but, after performing an action or several actions, the AI system is given feedback.

Deep learning is one of the research areas of machine learning, which has been very popular over the last decade, due to the successes in object detection and spech recognition owing to growth of massive amounts of computational power. Deep learning is biologically inspired by the structure of the human brain called artificial neural networks and can be trained to learn hierarchies of representations of the data. With emergence of more and more data, the performance of deep learning algoritms continues to increase as larger neural networks can be constructed and trained with more and more data.

This thesis aims to use deep learning algorithms in the area of predictive maintenance. Predictive maintenance techniques, which are designed to repair machines before they break down, can make enormous cost-savings and open the way to new business models. Most of the work related to predictive maintenance uses traditional machine learning models like rule based classification, support vector machine and fuzzy logic which focusses on feature engineering (manual construction of right features using domain expertise) to solve the problem of predictive maintenance. However for high dimensional data, such models have not been quite efficient as these models are not universal in nature and the features vary for different problems. As the volume of data is growing at an enormous rate, it becomes necessary to build a model which could learn from the failures in real time and able to undertake prediction activities. Also, although there are predictive maintenance templates available, a comprehensive architecture to apply the model is still under research. In this thesis, predictive maintenance of batteries in industrial datasets is considered where both maintenance and repair activities can be predicted beforehand. Sequence learning technique viz. recurrent neural networks is applied to represent the failure profiles by learning about failures from failure events of the batteries and understand the patterns that lead to failures in batteries simultaneously. Grid LSTM, a recurrent neural network model which can be applied to vectors, sequences or higher dimensional data has been applied over the sequential industrial data so that the model

can continuously learn about the failure patterns. The thesis also provides an architecture to deal with preventive maintenance problems using Apche Spark and AWS (Amazon Web Services).

Chapter 2

Organisation

This section tries to give an overview of the thesis.

Chapter 3 explains about the deep learning solutions in general where as **chapter 4** talks about predictive maintenance in industries. In **Chapter 5**, some relevant machine learning and deep learning algorithms are being discussed which have been used in the thesis. **Chapter 6** talks about related work in sequence learning as well as some related predictive maintenance solutions which have already been proposed.

Chapter 7 explains the dataset - battery dataset and failure events where as **chapter 8** talks about descriptive mining techniques which has been used to process the data to make it suitable for the project. **Chapter 9** outlines some predictive maintenance approaches. **Chapter 10** introduces the deep learning algorithm - Grid LSTM which has been used for the project. **Chapter 11** provides an architecture which could be used in industries to deal with the problem of predictive maintenance. **Chapter 12** discusses about the implementation of Grid LSTM and the performance is discussed in **Chapter 13**. The results of the project and the future scope of the project has been described in **Chapter 14** and **Chapter 15** respectively.

Chapter 3

Deep Learning

The recent years have seen a resurgence in the field of neural networks. Deep Learning has been able to create exciting AI applications to build self-driving cars, accurate speech recognition, systems that can detect and understand images, and much more. Despite all the recent progress, the potential of deep learning has been hugely untapped in many fields such as precision agriculture, consumer finance, medicine and other sectors where there is a clear opportunity for deep learning to have meaningful impact, but not much of progress have been done yet[3].

Deep learning traces it's origins to representation learning and artificial neural networks (ANNs). Traditional Machine Learning algorithms observe data drawn from some distribution, and then try to predict some aspect of this distribution. Representation Learning, on the other hand, does not try to predict observables, but tries to learn something about the underlying structure of the date. It tries to do the same by automatically building a high-level representation of data. Artifical neural networks are a family of models inspired by biological neural networks (the central nervous systems of animals, in particular the brain). They are used to estimate or approximate functions that can depend on a large number of inputs and are generally unknown. Artificial neural networks are generally presented as systems of interconnected "neurons" which exchange messages between each other. The connections have numeric weights that can be tuned based on experience, making neural networks adaptive to inputs and capable of learning.

As shown in the figure above, artifical neural neworks are typically organized in layers. Layers are made up of a number of interconnected nodes which contain an activation function. Patterns are presented to the network via the input layer, which communicates to one or more hidden layers where the actual processing is done via a system of weighted

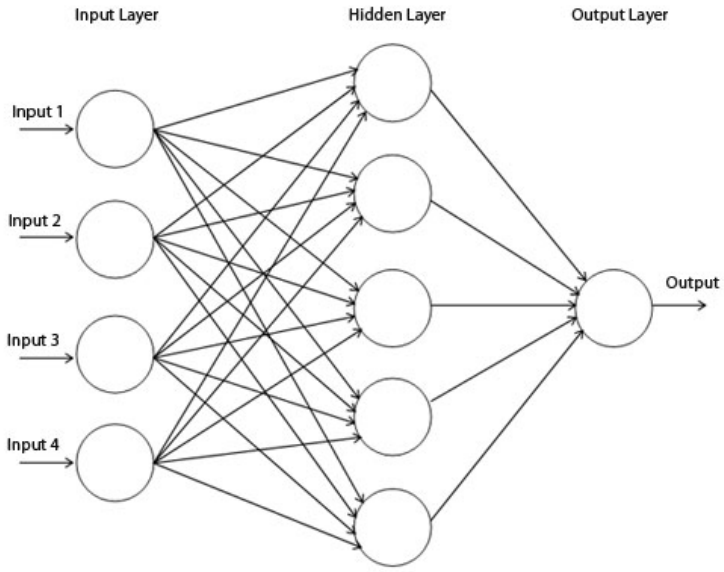


Figure 3.1: Artificial Neural Network Architecture taken from "Wikipedia"

connections. The hidden layers then link to an output layer where the answer is output. ANN's essentially learn with the help of a learning rule which modifies the weights of the connections according to the input patterns that it is presented with.

A formal definition of deep learning is given by [Deng, 2013][2] which states, deep learning is a hierarchical method of learning representions or abstractions of data. The features are not complex in a point, they are distributed in different layers and different units one layer after another. Deep learning networks are multilayer neural networks which have more layers between input and output as compared to other artifical neural networks, which allows richer intermediate representations to be built. It is different from representation learning due to the fact that while representation learning is a high level representation, deep-learning algorithms attempt to learn multiple levels of representation of increasing complexity./abstraction and the number of levels can be data-selected.

The main factors which have led to the recent upsurge in popularity of deep learning networks are: (i) scale of data and (ii) scale of computation. As we humans spend more time on internet from our computers and mobile devices, for the past two decades we've been rapidly accumulating data. Over the last few years, we have been able to figure out ways to scale computation so as to build deep learning networks that can take advantage of this voluminous amount of data. Researchers have started shifting deep learning to GPUs and evenutally shifting to HPC (High Performance Computing/Supercomputing)

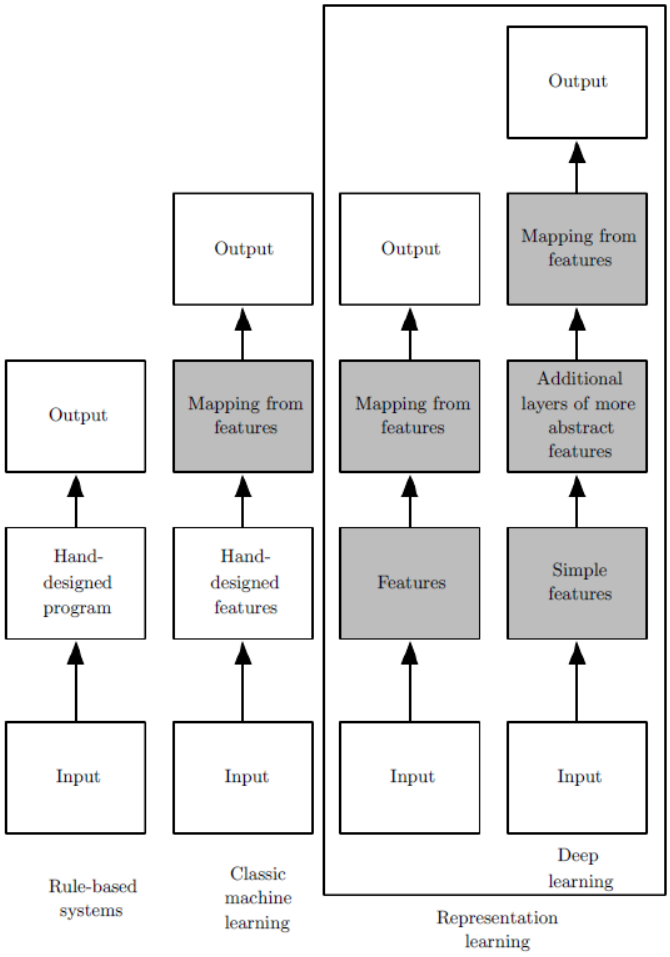


Figure 3.2: Evolution of Deep Learning Algorithms taken from "Deep Learning by Ian Goodfellow and Yoshua Bengio and Aaron Courville"

tactics for scaling up deep learning. These methods are considered to make researchers more efficient and help accelerate the progress in deep learning as a whole.

Chapter 4

Predictive Maintenance

The topic of predictive maintenance has become increasingly important in recent years, and has become related to the domains like Industry 4.0, the Internet of Things and Big Data. Technology manufacturer companies are equipping machines and industrial systems with sensors that enable remote monitoring along with the right software. To automate the machining process, it has become important to control and regulate entire production systems in a holistic and dynamic manner. Defective components that could soon lead to a system shutdown need to be identified independently of the usual maintenance schedule, and can be replaced before damage actually occurs. High reliability and availability become increasingly important as the industrial systems get more complex and depend on more actors.

Predictive maintenance does not rely on operators – whether cars, aircraft, CNC mills or automobiles – to set maintenance schedules. It enables continuous monitoring of components inside the machine – using sensors that can measure vibration, temperature or humidity, for example. Specialized maintenance softwares gather this sensor data, evaluate it and recognize in advance when a component might be prone to fail. Defective components that could soon lead to a system shutdown are identified independently of the usual maintenance schedule, and can be replaced before damage actually happens.

Predictive maintenance brings big cost savings by limiting downtimes, saving energy and reducing maintenance costs. That's why it becomes important to build predictive maintenance models that would proactively detect a defect before it becomes apparent and causes costly downtime. The future of predictive maintenance is into predictive producing where a production unit would become a veritable smart factory.

Predictive maintenance solutions can be classified into two categories - on-board

solutions and off-board solutions. While on-board solutions are computationally inexpensive methods and have small memory foot print, off-board solutions require larger computational resources and access to Historical Data. The commonly used approaches for predicitve maintenance include - Remaining Useful Life Prediction, Deviation Detection and Supervised Classification. All of these approaches can use both historical and real time data as input, but use of both these data for the three approaches mentioned above, is still under research. Out of the three mentioned approaches, deviation detection is relatively easy as uncertainty in linking deviations to failures is left out.

Chapter 5

Fundamentals

The capacity and capabilities of our human brain is immense. We can do a plethora of tasks with utmost ease. When we group images into a fixed number of labels, it is just a small part of it. If we holistically consider the process of learning, it is actually based on all our past experiences. Every time we gather new information, the brain stores it for future context. This powerful way of learning is what makes our brain really special. Our brain is trained to learn from a sequence of events and not necessarily just a single instance. For example, when we see a car, irrespective of their shapes and sizes, we know it's a car. It is because of the way we have learnt to recognize them. In this chapter, we are going to outline some fundamental concepts which are beneficial for the understanding of the thesis.

5.1 Sequence Learning

Sequential data in real life includes video, speech, stock market, sensor data, and so on. Sequences have complex temporal dependencies and a lot of hidden information. When we try to build a model for sequential data, we need to understand the characteristics of sequential data. For example, we try to predict if it's going to be a sunny day tomorrow based on all the parameters available today. There are a bunch of things that affect weather like temperature, pressure, humidity, and so on. If we just use a static snapshot of data available 12 hours earlier, then we won't be able to predict accurately. We need to understand the pattern over the last few days, in order to accurately predict what's going to happen tomorrow.

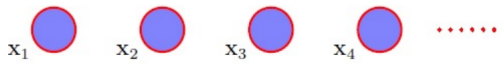


Figure 5.1: The simplest approach to modeling a sequence of observations is to treat them as independent, corresponding to graph without links taken from Bishop 2006 [4]

When we build convolutional neural network models, we explicitly consider the 2D nature of the input data i.e. images. So when we build the model for sequential data, we need to explicitly consider the sequential nature of input data. When we are building convolutional neural network models, we assume that the input data has a fixed size. For sequential data analysis, it is not possible. So when we build a model, it should know how to model sequences of arbitrary length. The model needs to know how to account for long term dependencies. It shouldn't ask the user to specify a time window within which it should look for dependencies. One of the main issues with sequential data is that we don't know exactly what time instant affects future outcomes. We don't know how far back we should go back to look for patterns. This is one of the things the model has to learn to do on it's own.

For many applications we consider a sequence, a collection of random variables to be in an identically independent distribution (which is usually called in short i.i.d.); which means each random variable possess the same probability distribution as the others and all are mutually independent. However it does not work always as the assumption of statistical independence is not enough and sometimes ends up in poor results. In order to understand the patterns for sequential data, specifically when we are building a probabilistic model, we need to relax the i.i.d. assumption. We can see the graphical interpretation with i.i.d. assumption in the above figure.

There are different models in order to understand the sequential characteristics of the patterns, such as Dynamic Bayesian Networks, Linear Dynamical Systems, Sliding Window Method, Recurrent Sliding Windows, Conditional Random Fields, Mixture Distributions, Hidden Markov Models, Discriminative Methods, Maximum Entropy Markov Models, Graph Transformer Networks, Input-Output Markov Models and so on. The Markov model is one of the most important models to represent the sequential patterns and is a member of Probabilistic Graphical Models' family. There are two kinds of Probabilistic Graphical Models, one is Directed Graphical Models (acyclic) also known as Bayesian network, and the other is Undirected Graphical Model or Markov Random Field or Markov Network. In the subsequent sections, for the scope of the thesis, we focus on Hidden Markov Models and subsequent neural network models [3].

5.2 Hidden Markov Models

Hidden Markov Models (HMM) are used in different commercial applications in speech processing, speech recognition, and speaker recognition. HMMs are also used in handwriting recognition, machine translation, and sign language recognition. HMM's are dynamical Bayesian networks, where we can observe a sequence of emissions, but we cannot figure out the sequence of states the model went through to generate the emissions. Analysis of HMM's tries to recover the sequence of states from the observed data.

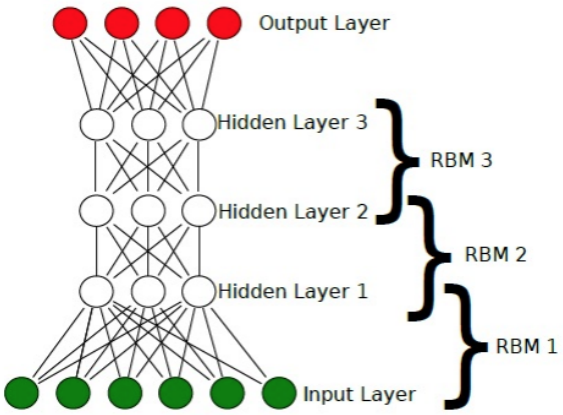


Figure 5.2: A representation of Hidden Markov Model where probability of being in a particular state at step i is known, once we know what state we were at step i-1. Probability of seeing a particular emission at step i is known, when we know what state, we were at step i. [6]

.

Although, HMMs benefit from relatively fast and powerful training and decoding algorithms, they have some limitations:

- Hidden Markov models suffer from poor discrimination. The models are trained independently one another. The algorithms use maximum likelihood inference which is a generative model.
- Hidden Markov models have a priori choice of model topology and statistical distributions.
- Hidden Markov models assume that the state sequences are first-order Markov chains. For example, in some cases it may be possible the dependency between random variables is from higher orders or even it might be differ from one pair to another.

- Sometimes in speech (where they are mainly used), no acoustical context is used, so that possible correlations between successive acoustic vectors is overlooked [3].

  To overcome these limitations, HMM's are used together with neural networks (hybrid models). In the subsequent chapters, we discuss about neural networks which are relevant for the thesis.

5.3 Optimization Algorithms

Optimization algorithms are used to minimize (or maximize) a Loss function (another name for Error function) which is simply a mathematical function dependent on the prediction model's ability to learn to compute the target values from the set of predictors. For instance, we call the Weights and the Bias values of the neural network as the internal learnable parameters of the model. The loss and the bias values are used in computing the output values and are learned and updated in order to have an optimal solution. The main goal is to minimize the loss by the network's training process.

There are two types of optimization algorithms - first order and second order optimization algorithms. First order optimization algorithms minimize or maximize a Loss function using its Gradient values with respect to the parameters. Second-order optimization algorithms use the second order derivative which is also called Hessian to minimize or maximize the Loss function. The Hessian is a Matrix of Second Order Partial Derivatives. Since the second derivative is costly to compute, the second order is not used much as compared to the first order optimization algorithms which are easy to compute and less time consuming, converging pretty fast on large data sets [8]. We will consider one of the most common first order optimization algorithms - Gradient Descent.

5.3.1 Gradient Descent

Gradient Descent is the most important technique in training and optimizing neural networks. The parameters of the prediction model are updated and tuned in a way to minimize the Loss function. Artificial Neural Network models are trained by a famous technique called Backpropagation , in which the dot product of Inputs signals and their corresponding weights are calculated first.Then an activation function is applied to those sum of products, which transforms the input signal to an output signal. It is also important

to model complex Non-linear functions and introduces Non-linearities to the Model which enables the Model to learn almost any arbitrary functional mappings. The network is then propagated backwards with the Error terms. The Weights values are updated using Gradient Descent, in which the gradient of Error(E) function is calculated with respect to the Weights or the parameters. The parameters (Weights) are updated in the opposite direction of the Gradient of the Loss function w.r.t to the Model's parameters. Although Gradient Descent calculates the gradient of the whole data set, it performs only one update. Hence, sometimes it can be very slow and hard to control for very large datasets and don't fit in the memory.

Variants of Gradient Descent - Stochastic Gradient Descent and Mini Batch Gradient Descent are used to overcome the problems mentioned before. While Stochastic Gradient Descent performs a parameter update for each training example and is usually faster, however due to the frequent updates and fluctuations, it ultimately complicates the convergence to the exact minimum and will keep overshooting due to the frequent fluctuations. Mini Batch Gradient Descent takes the best of both techniques - Gradient Descent and Stochastic Gradient Descent and performs an update for every batch with training examples in each batch. It is able to reduce the variance in the parameter updates , which leads to a much better and stable convergence. Mini-batch gradient descent is now mostly used for training neural networks. Since choosing a proper learning rate could be hard, Gradient Descent is further optimized with various algorithms mentioned below[8].

- Momentum: The high variance oscillations in SGD makes it hard to reach convergence. Momentum is a technique which was invented to accelerate Stochastic Gradient Descent by navigating along the relevant direction and soften the oscillations in irrelevant directions.

- Nesterov accelerated gradient: Momentum however gets pretty high and cannot slow down as we reach the minima i.e the lowest point on the curve. Unfortunately, it could miss the minima entirely and continue moving up. Nesterov accelerated gradient suggests to first make a big jump based on the previous momentum and then calculate the Gradient. Correction is made then which results in an parameter update. Now this anticipatory update prevents the gradient to go too fast and not miss the minima and makes it more responsive to changes.

- Adagrad: Adagrad allows the learning Rate to adapt based on the parameters. So it makes big updates for infrequent parameters and small updates for frequent parameters. For this reason, it works well with sparse data. Howver Adagrad suffers

with Decaying Learning Rate - where the Learning rate gets smaller and smaller. The ability of the Model to learn fastly decreases and which in turn leads to slow convergence and takes very long to train and learn.

- AdaDelta : AdaDelta is an extension of AdaGrad which tends to remove the Decaying Learning Rate problem. Instead of accumulating all previous squared gradients, Adadelta limits the window of accumulated past gradients to some fixed size.
- Adam: Adam which stands for Adaptive Moment Estimation is the most efficient of all the algorithms mentioned above. It is able to calculate individual momentum changes for each parameter and store them separately. In addition to storing an exponentially decaying average of past squared gradients like AdaDelta ,Adam also keeps an exponentially decaying average of past gradients, similar to momentum[8].

For training a deep neural network model or a highly complex neural network model, Adam is mostly being used due to its significant advantages over the other algorithms.

5.4 Deep Belief Networks

Deep Belief Networks (DBNs) are generative probabilistic models with one visible layer at the bottom and many hidden layers upto the output. Each hidden layer unit earns the statistical representation via the links to the lower layers. The more higher the layers, the more complex are the representations [7].

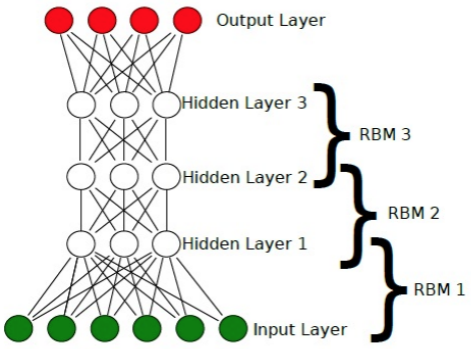


Figure 5.3: The schematic of a Deep Belief Network. The number of layer and the number of units on each layer in the schema are only examples taken from Hamel and Eck, 2010 [5]

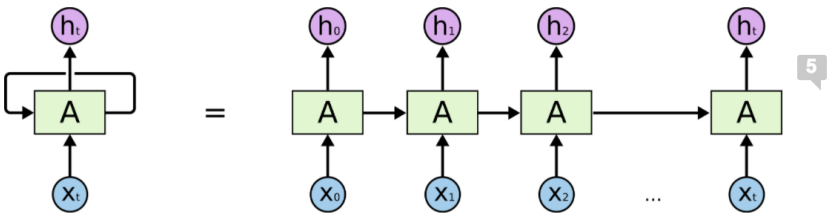
The main problem with the deep neural network architectures was the learning process,

since the ordinary gradient descent algorithm does not work well and sometimes it makes the training quite impossible for DBN's. To solve this problem, a greedy layerwise unsupervised pre-training can be used [5]. After the pre-training, successful supervised learning can be done by a procedure called fine-tuning, using gradient descent, which we discussed before.

5.5 Recurrent Neural Networks

We humans don't start thinking from scratch every second. As we read sentences, we can understand each words by getting the context from our understanding of our previous words. Traditional artifical neural networks are unable to do this which led to the emergence of recurrent neural networks. Recurrent neural networks are neural networks which have loops in them allowing information to persist. This is the precise reason, they are used for sequential data.

In the diagram, we can look at the neural network, A, which looks at some input x_t and outputs a value h_t. A loop allows information to be passed from one step of the network to the next. It can be thought of as multiple copies of the same network, which passes a message to it's successor [9].

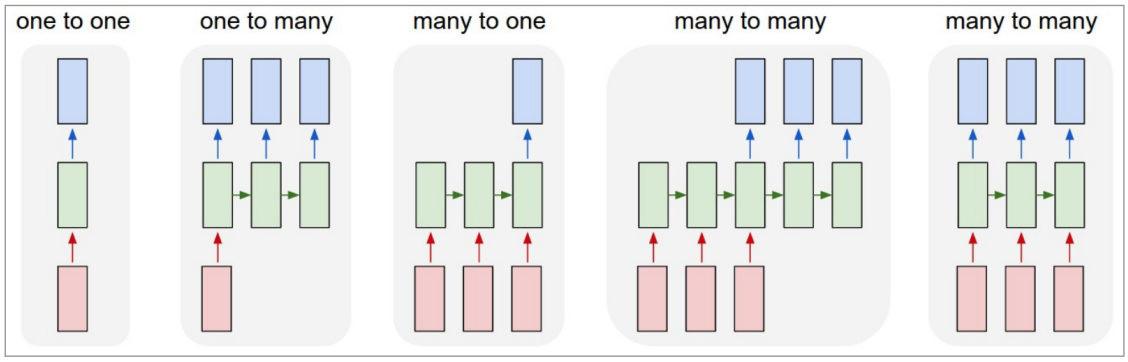


An unrolled recurrent neural network.

Figure 5.4: Taken from Chris Olah's Blog on LSTM [9]

Recurrent nets allow to operate over sequences of vectors: Sequences in the input, the output, or in the most general case both. RNNs combine the input vector with their state vector with a fixed (but learned) function to produce a new state vector [10].

RNN's however are not so effective when we need more context while looking at recent information to perform the present task. The gap between the relevant information and the point where it is needed to become very large and as a result, RNNs become unable to learn to connect the information.



Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

Figure 5.5: Taken from Karpathy's blog on RNN's [10]

5.6 Long Short Term Memory (LSTM)

Long Short Term Memory networks – usually just called "LSTMs" – are a special kind of RNNs, capable of learning long-term dependencies. Introduced by Hochreiter and Schmidhuber (1997), they work tremendously well on a large variety of problems, and are now widely used. LSTMs are explicitly designed to avoid the long-term dependency problem [9].

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

LSTMs also have a chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way [9].

The key to LSTMs is the cell state, the horizontal line running through the top of the diagram. The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates [9].

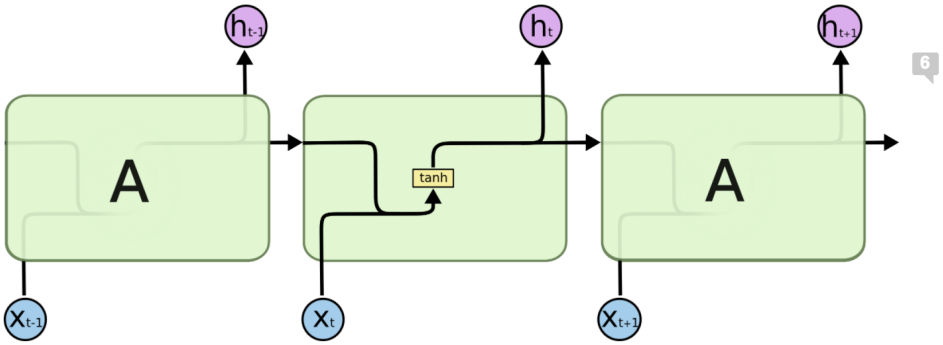


Figure 5.6: Taken from Colah's Blog [9]

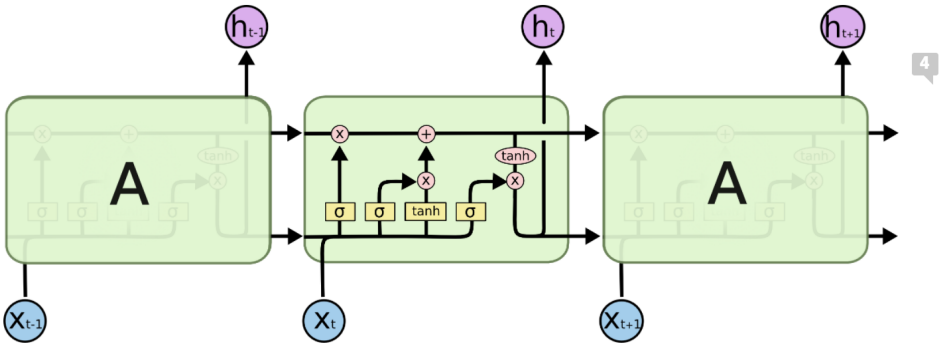


Figure 5.7: Taken from Colah's Blog [9]

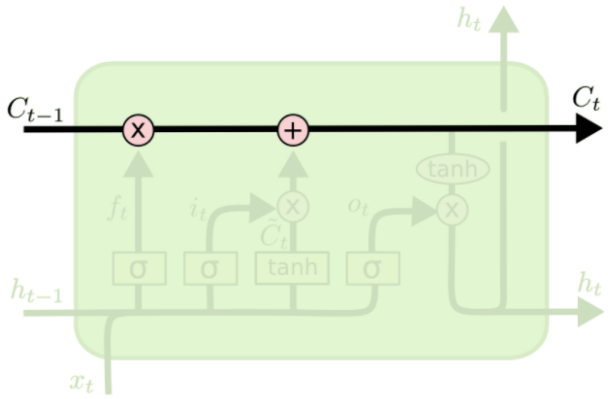


Figure 5.8: Taken from Colah's Blog [9]

Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer

outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means "let nothing through," while a value of one means "let everything through!"

An LSTM has three of these gates, to protect and control the cell state [9].

Chapter 6

Related Work

A number of solutions have been used to deal with sequential data. This section outlines some approaches.

6.1 Deep Belief Network - Hidden Markov Model (Hybrid Approach)

Deep Belief Networks are used to extract the probabilities for sequence modeling; they play the role of temporal ordering where as Hidden Markov Models are used to model dynamic properties of the data. Deep Belief Networks are trained with unsupervised fashion as a pre-training stage followed by supervised tuning such as back propagation algorithm in order to model posterior densities over Hidden Markov Model states for a given frame of input. The input comprises of sequence of features which are fed to DBN as training set. In supervised learning phase, the cross-entropy loss for each HMM state predictions is optimized [11].

6.2 Conditional Deep Belief Networks

Building blocks of Conditional Deep Belief Network are Conditional Restricted Boltzmann Machines (CRBMs). The conditional dependency deals with temporal contents in data. RBMs and stacked RBMs form a family of generative models that is promising in modeling sequential data, since they exhibit a great representational power. RBMs are transformed

into Conditional RBMs by adding 2 dimensions – one is horizontal and directed (depth in time) and the other is vertical and undirected. 2-level Conditional RBMs are able to learn good representations of the observed data, by capturing different factors of variation in the input space and efficiently decoupling them [12].

6.3 Recurrent Temporal Restricted Boltzmann Machine

Recurrent Temporal Restricted Boltzmann Machine is similar to Conditional Restricted Boltzmann Machine, except that, unlike CDRBMs, there is no direct link between previous time steps and current hidden layer. It is possible to introduce time dependencies of more than one step behind. Contrastive divergence algorithm are being used for training with only one step reconstruction. Weights on time dependency edges are learned using back propagation algorithm [13].

6.4 Sequential Deep Belief Networks

Sequential Deep Belief Networks are less generalized as compared to Temporal Restricted Boltzmann Machine, but they cover broader concept of time dependencies. Conditional random fields which are suitable for sequence labeling are used to represent conditional dependencies; but requires to be carefully designed by the engineer regarding the tasks at hand, which is time consuming and arduous task. Sequential Deep Belief Networks add hidden layers to CRFs to extract the features which are not explicit in data. Sequential Restricted Boltzmann Machines are similar to ordinary Restricted Boltzmann Machines except that they spread over time. Sequential Deep Belief Networks are formed by stacking multiple block of SRBN on top of each other, where time is an additional dimension to the structure [14].

6.5 Deep Spatio-Temporal Inference Networks

Deep Spatio-Temporal Inference Network is an unsupervised algorithm, where pattern recognition is done by adding a layer of supervised machine learning methods. It is a

discriminative learning architecture, which is a combination of unsupervised learning for dynamic patterns and Bayesian inference. High-dimensional sensory data is divided into small segments which are then modelled. The architecture comprises of multiple nodes, each of which characterizes the sequence of patterns from previous nodes. Raw data is fed to the system at the lowest layer (input layer). A belief state, which is a probability mass function over sequences of input is learned by the nodes which is received by the upper layers. DeSTIN works without any prior estimation of temporal window size, rather learning it adaptively [15].

6.6 Hierarchial Temporal Memory - Realtime Anomaly Detection

Hierarchial Temporal Memory is an anomaly detection technique based on online sequence memory algorithm – Hierarchical Temporal Memory. They are Efficient, extremely tolerant to noisy data, continuously adapts to change sin the statistics of data and detects very subtle anomalies while minimizing false positives. HTM networks are continuously learning and model the spatiotemporal characteristics of their inputs. HTM networks do not directly output anomaly score – raw anomaly score and anomaly likelihood are computed using a prediction vector which incorporates inferred information about current sequences. Multiple independent models are then combined by obtaining a final anomaly likelihood score [16].

6.7 Hierarchial Temporal Memory - Continuous Online Sequence Learning with Unsupervised Network Model

In this work, models are able to continuously learn a large number of variable-order temporal sequences using an unsupervised Hebbian-like learning rule. Sparse temporal codes formed by the model can robustly handle branching temporal sequences by maintaining multiple predictions until there is sufficient disambiguating evidence. The model exhibits properties critical for sequence learning – continuous online learning, ability to handle multiple projections and branching sequences with high order statistics,

robustness to sensor noise and fault tolerance and good performance without task-specific hyper-parameter tuning. The model also achieves comparable accuracy to other state of the art algorithms [17].

6.8 Multi Dimensional Recurrent Neural Networks (MD-RNN)

There is no direct way of applying recurrent neural networks to data with more than one spatio-temporal dimension. Multi-dimensional Hidden Markov Models suffer from two serious drawbacks:

- Time required to run the Viterbi algorithm, and thereby calculate the optimal state sequences, grows exponentially with the size of the data.
- Number of transition probabilities, and hence the required memory, grows exponentially with the data dimensionality.

MD-RNN extends the potential applicability of RNNs to vision, video processing, medical imaging and many other areas, while avoiding the scaling problems that have plagued other multi-dimensional models. MD-RNN replaces the single recurrent connection found in standard RNNs with as many recurrent connections as there are dimensions in the data. During the forward pass, at each point in the data sequence, the hidden layer of the network receives both an external input and its own activations from one step back along all dimensions [18].

6.9 Bi-directional LSTM and Hidd en Markov Model - LSTM

Bidirectional LSTM outperforms both unidirectional LSTM and conventional Recurrent Neural Networks (RNNs) in TIMIT dataset. Hidden Markov Model –Bidirectional LSTM hybrid model has outperformed context dependent and context independent HMM, LSTM and bidirectional recurrent neural networks on TIMIT Speech Corpus. Best results were achieved with HMM-BLSTM hybrid using a weighted error signal. Hybrid systems had fewer parameters than context dependent HMM as high number of states are required for

HMM to model contextual dependencies [19].

6.10 Clockwise Recurrent Neural Networks (CW-RNN)

Recurrent neural networks are usually difficult to train successfully when long term memory is required. CW-RNN, simple, yet powerful modification to the standard RNN architecture. In Clockwork RNN (CW-RNN), the hidden layer is partitioned into separate modules, each processing inputs at its own temporal granularity, making computations only at its prescribed clock rate. CW-RNN reduces the number of RNN parameters, improves the performance significantly in the tasks tested, and speeds up the network evaluation since not all modules are executed at every time step. CW-RNN have a smaller number of weights compared to Standard RNNs, because slower modules are not connected to faster ones. CW-RNN outperformed both RNN and LSTM networks for TIMIT spoken word classification [20].

6.11 Sequence to Sequence Learning with Neural Networks

Sequence to Sequence Learning is a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure.. Multilayered Long Short-Term Memory (LSTM) maps the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Reversing the order of source labels (not target labels) improves LSTM's performance markedly, because doing so introduces many short term dependencies between the source and the target label which made the optimization problem easier. The Connectionist Sequence Classification is another popular technique for mapping sequences to sequences with neural networks, although it assumes a monotonic alignment between the inputs and the outputs. Deep LSTMs significantly outperformed shallow LSTMs [21].

6.12 Gated Recurrent Neural Networks on Sequence Modeling

Gated recurrent unit (GRU) makes each recurrent unit to adaptively capture dependencies of different time scales. Similar to LSTM, GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cells. Any important feature, decided by either the forget gate of the LSTM unit or the update gate of the GRU, will not be overwritten but be maintained as it is. In the LSTM unit, the amount of the memory content that is seen, or used by other units in the network is controlled by the output gate. On the other hand the GRU exposes its full content without any control. LSTM unit controls the amount of the new memory content being added to the memory cell independently from the forget gate where as GRU controls the information flow from the previous activation when computing the new candidate activation, but does not independently control the amount of the candidate activation being added (the control is tied via the update gate) [22].

6.13 Gated Feedback Recurrent Neural Networks (GF-RNN)

Gated-feedback RNN (GF-RNN), extends the existing approach of stacking multiple recurrent layers by allowing and controlling signals flowing from upper recurrent layers to lower layers using a global gating unit for each pair of layers. The recurrent signals exchanged between layers are gated adaptively based on the previous hidden states and the current input. GF-RNN outperforms the conventional approaches to build deep stacked RNNs as the GF-RNN can adaptively assign different layers to different timescales and layer-to-layer interactions (including the top-down ones which are not usually present in a stacked RNN) by learning to gate these interactions. A deterioration in performance is observed when the proposed gated-feedback architecture is used together with a tanh activation function, unlike when it was used with more sophisticated gated activation functions [23].

6.14 Vanilla - LSTM

The most popular LSTM architecture Vanilla LSTM outperforms its 8 variants:

- No Input Gate (NIG)
- No Forget Gate (NFG)
- No Output Gate (NOG)
- No Input Activation Function (NIAF)
- No Output Activation Function (NOAF)
- No Peepholes (NP)
- Coupled Input and Forget Gate (CIFG)
- Full Gate Recurrence (FGR)

Certain modifications such as coupling the input and forget gates or removing peephole connections simplify LSTM without significantly hurting performance. The forget gate and the output activation function are the critical components of the LSTM block. Learning rate and network size are the most crucial tunable LSTM hyper-parameters. Hyper-parameters can be tuned independently i.e. the learning rate can be calibrated first using a fairly small network, thus saving a lot of experimentation time [24].

6.15 Sequence Training and Adaptation of Highway Deep Neural Networks

Highway Deep Neural Network is a type of depth-gated feedforward neural network, which is easier to train with more hidden layers and also generalized better compared to conventional plain deep neural networks (DNNs). HDNNs are equipped with two gate functions – Transform and Carry gate. Transform gate used to scale the output of a hidden layer. Carry gate used to pass through the input directly after elementwise rescaling. These two gates that are tied across all the hidden layers are able to control the information flow over the whole network. Considerable improvements can be achieved by only updating these gate functions in both sequence training and adaptation experiments [25].

6.16 Depth Gated LSTM

Depth Gated LSTM is an extension of long short-term memory (LSTM) neural networks using a depth gate to connect memory cells of adjacent layers. Linear dependency between lower and upper layer recurrent unit is done through a gating function, which is referred to as depth gate. The depth gate is a function of the lower layer memory cell, the input to and the past memory cell of this layer. Related work includes Grid LSTM and highway networks which share the same idea of stacking networks with both linear but gated connections and nonlinear paths. DGLSTM is a specific and simple case of Grid LSTM that has gate applied to time and depth only on memory cells. Depth Gated LSTM is able to improve machine translation and language modeling performances [26].

6.17 Frequency LSTM, Time-Frequency LSTM and ReNet-LSTM

Frequency LSTM (F-LSTM) uses LSTM architecture, except that it models a sequential process in frequency. Time Frequency LSTM (TF-LSTM) extends Frequency LSTM to model the sequential process of the signal in both time and frequency jointly. Grid-LSTM is very similar to a TF-LSTM except there are separate LSTMs which move in time and frequency. A ReNet-LSTM is an F-LSTM unrolled in frequency and T-LSTM unrolled in time. These two LSTMs are treated completely independently, meaning the recurrent state is not shared when the F-LSTM and T-LSTMs overlap. The benefit of ReNet is computational efficiency, as the two LSTMs can be run in parallel. Grid LSTM offers the best performance of all 2D-LSTMs, particularly when the input is noisy or the filter bank is learned [28].

6.18 Phased LSTM: Accelerating Recurrent Neural Network Training for Long or Event-based Sequences

Current RNN models are ill-suited to process irregularly sampled data triggered by events generated in continuous time by sensors or other neurons. Such data can occur, for

example, when the input comes from novel event-driven artificial sensors that generate sparse, asynchronous streams of events or from multiple conventional sensors with different update intervals. Phased LSTM model extends the LSTM unit by adding a new time gate which is controlled by a parametrized oscillation with a frequency range that produces updates of the memory cell only during a small percentage of the cycle. Phased LSTM network achieves faster convergence than regular LSTMs on tasks which require learning of long sequences. The model naturally integrates inputs from sensors of arbitrary sampling rates, thereby opening new areas of investigation for processing asynchronous sensory events that carry timing information [29].

Chapter 7

Dataset Description

This chapter presents the datasets that is going to be used in the evaluation for identifying the sequence of failure events from the sequence of battery data. A test battery dataset for calculating remaining useful life of aero engines was considered which contains failure events for about half a million events from last couple of years. Batteries are continuously cycled with randomly generated current profiles. Reference charging and discharging cycles are also performed after a fixed interval of randomized usage in order to provide reference benchmarks for battery state of health.

- Battery Data: This dataset contains 931 atrributes which consists of features like time, battery type, state of charge, historical state of charge, stat aging insistance value, stat aging capacity value, number of clamping cycles, tension value, charge time, discharge time, charge time value, discharge time value, sum of state of charge, displayed state of charge, charging cycle time, discharging cycle time, state of health, etc.
- Aero Engine Data: Aero Engine Data has 155 attributes which all engine related data like engine model, start time, end time, down time, battery type, timestamp,
- Failure Events: This dataset contains 118 attributes which contain all the failure events , source, source type and timestamps of failures for all types of batteries. I

Chapter 8

Descriptive Mining Approaches

Data preprocessing was being done to clean the data. Null value Attributes and Replacing Missing Value Attributes were removed. The dataset had data from different time range, the events from the failure, aero engine and battery dates were put within same time range. Battery types and engine tyoes were segmented. The battery data and engine data were also to calculate the running time for each of the engine type and battery type. The failure nodes were segregated from the failure event dataset and matched to each battery type and engine type. Initially, Information gain and Gini Index were used to assign weights to the datasets to understand the coorelation.

The attributes ofThe battery data was serialized using Google Protocol Buffer into tf.Sequence.example format as Tensorflow is being used for the sequence to sequence labelling. The labels of every sequence are rescaled to be in range of [-1, 1]. The labels were generated for the training data for binary clasification where the model can successfully detect the failure patterns by looking at the past failure events. To ensure that the predicted outputs exhibit the same value scale, a tanh activation function is used.

Chapter 9

Predictive Mining Approaches

As we have discussed before, there are three approaches for predictive maintenance - , Deviation Detection, Remaining Useful Life Prediction and Supervised Classification. The following section gives an overview of the three approaches.

9.1 Deviation Detection

Deviation detection methods are distinguished from the other methods as they do not give specific information of what is wrong. To make deviation detection methods complete (and comparable to RUL and Classification methods) they need to be combined with historical examples of similar deviations linked to known failures. This introduces one more uncertainty into the complete prediction, which reduces the prediction accuracy. Threshold values or ranges could be set for particular feature or feature set for predictive maintenance. Deviation detection could also be made in the model space where a model distance measure which is not based on the model parameters, but rather on characteristics of the model provides the measure of deviation.

9.2 Remaining Useful Life Prediction (RUL)

The failing component is inherently known in the Remaining Useful Life Prediction approach as these models are developed for a specific component. One RUL model per machine is derived during training and stored in a database as references of degradation patterns. The database is deployed on-board and an algorithm tries to match the

degradation of the monitored system to one of the references systems stored in the database. Machine Learning models are used to calculate the most likely RUL by looking at the similarities to the reference models. Remaining Useful Life Prediction approach is basically a regression problem for predictive maintenance.

9.3 Supervised Classification

Supervised Classification models monitor the individual correlations and classify accordingly to failure events. The state-driven data labeling technique is used for classification in failure type detection and predictive maintenance. In lifetime-state data labeling for predictive maintenance, it is necessary to categorize the historic data into the lifetime states used. Lifetime-states are generally equally distributed over the lifetime of a component. Accuracy of classification usually changes, and becomes better at the end of the useful lifetime of a component. Fuzzy rule based system classification is an alternative approach to using deep architectures.

Both Remaining Useful Life and Supervised Classification approaches are considered in this work. The figure below outlines the whole workflow required for predictive maintenace solutions.

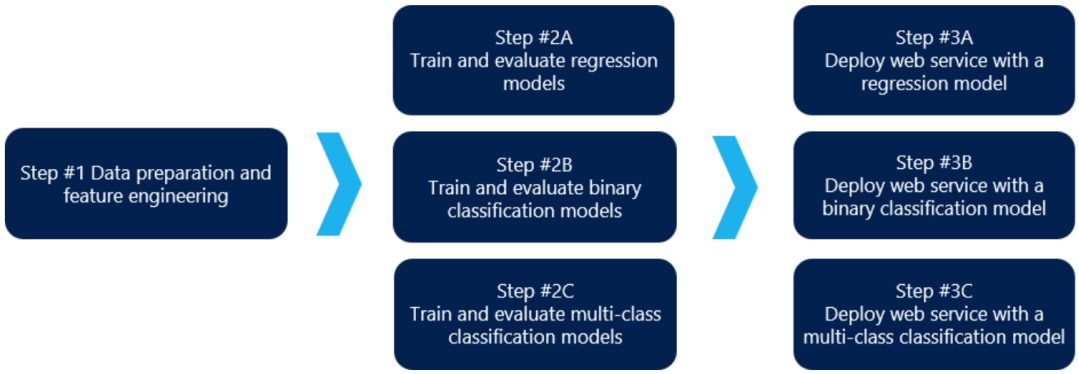


Figure 9.1: Predictive Maintenance Template taken from Microsoft AzureML Team

Chapter 10

Model Selection

The effectiveness and recent success of LSTM's over sequential data makes them good candidate for our use-case compared to traditional ones like Deep Belief networks and Hidden Markov models. LSTMs operate over sequences of vectors: sequences in the input or the output or both and enable sequential processing even in absence of sequences. Deep Belief Networks have shown remarkable results to extract a hierarchical representation from a dataset where as Restricted Boltzmann Machine (RBMs) is a good candidate for feature extraction. Hierarchical Temporal Memory shows remarkable results but not peer support available as mostly researchers only from Numenta Inc work on it. The subsequent section gives an overview about Grid LSTM, which is the selected algorithm.

10.1 Grid LSTM

Grid LSTM is a network of LSTM cells arranged in a multidimensional grid that can be applied to vectors, sequences or higher dimensional data such as images. The network differs from existing deep LSTM architectures in that the cells are connected between network layers as well as along the spatiotemporal dimensions of the data. The network provides a unified way of using LSTM for both deep and sequential computation. Grid LSTM has been used to define a novel two-dimensional translation model, the Reencoder where translation is done in a two-dimensional mapping. One dimension processes the source sequence whereas the other dimension produces the target sequence citenal.

In multidimensional LSTM models, the inputs are not arranged in a sequence, but in a N-dimensional grid, such as the two-dimensional grid of pixels in an image. At each input

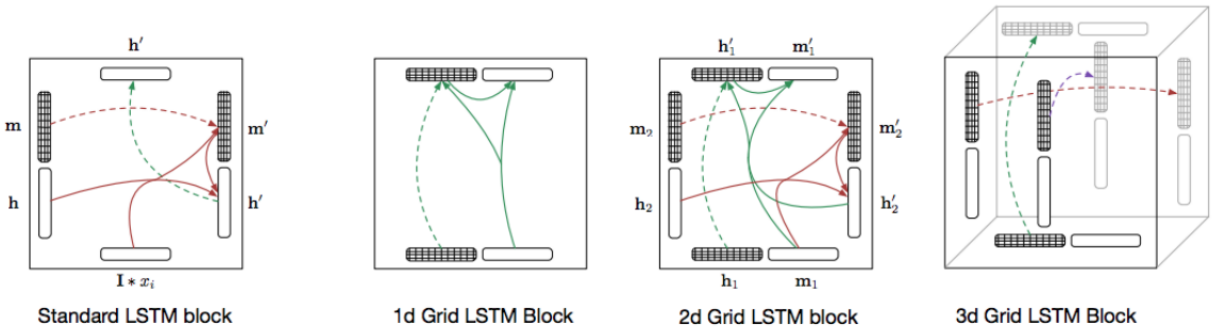


Figure 10.1: Blocks form the standard LSTM and those that form Grid LSTM networks of N = 1, 2 and 3 dimensions. The dashed lines indicate identity transformations. The standard LSTM block does not have a memory vector in the vertical dimension; by contrast, the 2d Grid LSTM block has the memory vector m1 applied along the vertical dimension.. [27]

.

x in the array the network receives N hidden vectors h_1, ..., h_N and N memory vectors m_1, ..., m_N and computes a hidden vector h and a memory vector m that are passed as the next state for each of the N dimensions. The network concatenates the transformed input $I \times x$ and the N hidden vectors h_1, ..., h_N into a vector H and computes g^u , g^o and g^c , as well as N forget gates g_i^f. These gates are then used to compute the memory vector as follows [27]:

$$\mathbf{m} = \sum_{i}^{N} \mathbf{g}_i^f \odot \mathbf{m}_i + \mathbf{g}^u \odot \mathbf{g}^c$$

Figure 10.2: Memory Vector for Multi-Dimensional recurrent Neural Networks. [27]

.

As the number of paths in a grid grows combinatorially with the size of each dimension and the total number of dimensions N, the values in m can grow at the same rate due to the unconstrained summation in Eq. 4. This can cause instability for large grids, and adding cells along the depth dimension increases N and exacerbates the problem. This motivates the simple alternate way of computing the output memory vectors in the Grid LSTM [27].

Grid LSTM deploys cells along any or all of the dimensions including the depth of the network. In the context of predicting a sequence, the Grid LSTM has cells along two dimensions, the temporal one of the sequence itself and the vertical one along the depth. To modulate the interaction of the cells in the two dimensions, the Grid LSTM proposes a simple mechanism where the values in the cells cannot grow combinatorially as shown

in the equation [27].

All the recent variants of LSTM – Time frequency based LSTM, Grid LSTM and Phased LSTM show remarkable results, but since Grid LSTM offers the best performance of all 2D-LSTMs, particularly when the input is noisy or the filter bank is learned, we choose Grid LSTM for our usecase.

Chapter 11

Architecture

This section presents a novel architecture where a cloud based solution for real time predictive maintenance of industrial datasets can be done. The streaming industrial dataset is on Hadoop architecture. We can use Apache Splunk to do data visualization and data transformation. We have the failure events in Apache Kafka which have different topics for the failure patterns. Data is ingested from Apache Kafka to Apache Spark. For the prediction model to work on real time streaming data, we use Apache Spark. We run our deep learning models with TensorFlow.on top of Spark. Apache Spark revolves around the concept of a resilient distributed dataset (RDD), which is a fault-tolerant collection of elements that can be operated on in parallel. We deploy the Apache Spark cluster on Amazon Web Services(AWS) using AWS EMR. Spark can be installed alongside the other Hadoop applications available in Amazon EMR. We send the Spark job to the EMR cluster and gather our results on Amazon S3.

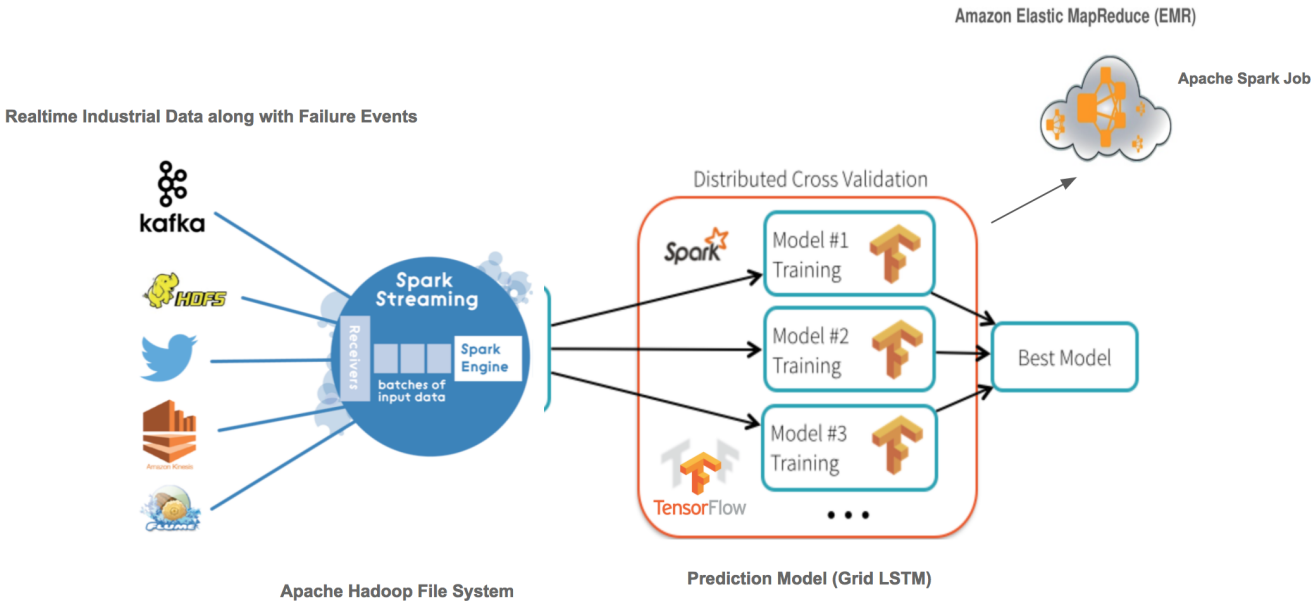


Figure 11.1: Architecture for Real time Predictive Maintenance of Industrial Data

.

Chapter 12

Implementation

This section outlines our neural network model in Tensorflow. When using LSTMs in the time-series domain, one important parameter to pick is the sequence length which is the window for LSTMs to look back. This may be viewed as similar to picking window size = 5 cycles for labelling the sequences. The idea of using LSTMs is to let the model extract abstract features out of the sequence of failed events in the window rather than engineering those manually. The expectation is that if there is a pattern in these events within the window prior to failure, the pattern should be encoded by the LSTM. One critical advantage of LSTMs is their ability to remember from long-term sequences (window sizes) which is hard to achieve by traditional feature engineering. For example, computing rolling averages over a window size of 50 cycles may lead to loss of information due to smoothing and abstracting of values over such a long period, instead, using all 50 values as input may provide better results. While feature engineering over large window sizes may not make sense, LSTMs are able to use larger window sizes and use all the information in the window as input.

The LSTM layers expect an input in the shape of a numpy array of 3 dimensions (samples, time steps, features) where samples is the number of training sequences, time steps is the look back window or sequence length and features is the number of features of each sequence at each time step.

We use multilayered LSTM's to map the input sequence of the battery values to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence of failure events from the vector. Same length and variable sequences are both being implemented .The first layer is a Grid LSTM layer with 100 units followed by another Grid LSTM layer with 50 units. Dropout is also applied after each LSTM layer to control

overfitting. Final layer is a Dense output layer with single unit and sigmoid activation since this is a binary classification problem. Model prediction is done at each time step. We use a regressor to approximate a sequence of vectors. We repeat the same procedure replacing the Grid LSTM layers with only LSTM layers.

We leverage Apache Spark to do both hyper parameter tuning as well as desploy the models at scale.

- Hyperparameter Tuning: We use Spark to find the best set of hyperparameters for neural network training. Even though TensorFlow itself is not distributed, the hyperparameter tuning process is very much parallel and can be distributed using Spark. In this case, we can use Spark to broadcast the common elements such as data and model description, and then schedule the individual repetitive computations across a cluster of machines in a fault-tolerant manner.
- Deploying models at scale: We use Spark to apply our trained neural network model on our dataset. The model is first distributed to the workers of the clusters, using Spark's built-in broadcasting mechanism. Then this model is loaded on each node and applied to the sequences.

Chapter 13

Results

This chapter presents the evaluation results of our model on all datasets of chapter 7. Since the model exhibits a vast number of hyper parameters and each training can take several days, performing an extensive grid search is not possible. All experiments have been computed on a single Nvidia GeForce GTX Titan X using the TensorFlow open source software library for machine intelligence in version r0.10 with CUDA 7.5.18 and cuDNN 4. This quite new deep learning framework is developed by Google as a second-generation system based on their experiences with DistBelief [Dea+12]. A computation in TensorFlow is expressed as a stateful dataflow graph, where each node represents a mathematical operation and the graph edges a data tensor that can flow from one node to the next [Aba+15]. It has gained increasing attention since its first public release in November 2015.

13.1 Training

For each failure events, we construct a LSTM model (with 80 percent for training and 20 percent for testing). We optimize with weighted training method and denoising LSTM to avoid overfitting. We allow the model to classify the sequences that affect the battery data. Mean pooling is used on sequences to generate input for an extra neural network layer to further learn the feature representation. The output of the neural network layer is learned by the survival model. The failure metric is correlated to the prediction of failure sequences from the failure events.

13.2 Hyperparameter Tuning

We match either whole sequence or split sequence for the dataset to predict the failure sequence for more than 10000 events, layer sizes applied being 2, 3, 4, 5 using different time steps. We provide batch of two different sequences but having same sequence length for different time steps. We use hyper parameter tuning to calculate the standard cross-entropy loss. We further optimize with Adam optmization algorithm. The model is incrementally explored in order to understand its behavior regarding changes in the hyperparameters. Starting from the network using binary crossentropy only, perceptual motivated loss terms like GDL and SSIM are added one after another until it ends up in the triplet loss. While the network purely trained on the BCE loss function performs best regarding the binary cross-entropy on the validation set, it nevertheless performs worst according to the sharpness difference metric shown in Figure 13.1 (b).

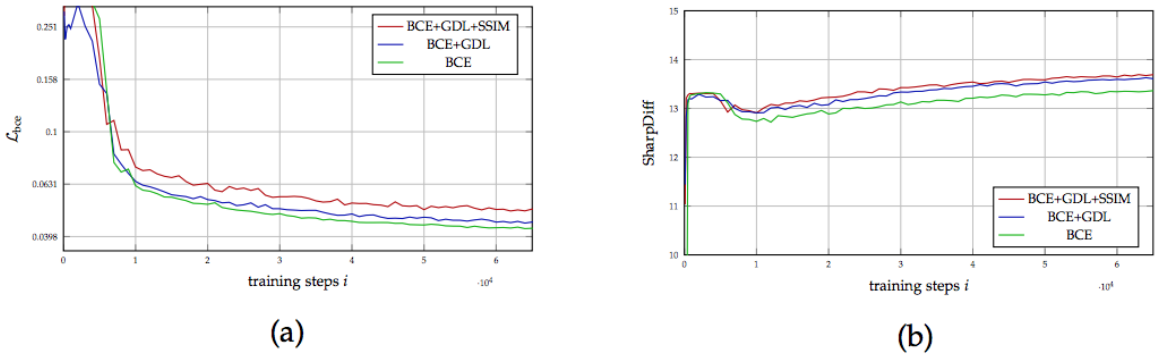


Figure 13.1: Comparison of validation results using a 2-layer network with varying loss function combinations. All models have been trained given an input sequence of 8 variables to predict the next 8 variables

.

13.3 Test Results on LSTM and Grid LSTM

For the final evaluation of our model with the dataset, the qualitative and quantitative performance of the best configuration is assessed on the test set. Inspired by the findings of the model exploration, as well as a rudimentary grid-search, the model for this final test ends up to use the default settings as described earlier, but with an exponential learning rate decay of $\alpha = 0.95$ and a loss layer that utilizes no perceptual motivated loss term. The

LSTm and Grid-LSTM models of this network with different numbers of recurrent layers are trained for 100,000 steps. After training the model for about 1,000,000 training steps which takes almost one week, our Grid LSTM model outperforms the standard LSTM model with a higher precision and recall than the standard LSTM model.

A comparison of the best performing model (our model - Grid-LSTM) along with standard LSTM model us outlined in the table.

	Accuracy	Precision	Recall	F1-score
Grid LSTM	0.979195	0.950000	0.94	0.940000
LSTM	0.930980	0.942391	0.87	0.879565

Figure 13.2: Comparison of Results between LSTM and Grid LSTM

.

Chapter 14

Conclusion

We presented existing unsupervised deep learning approaches for predicting sequences of batteries to identify the failure events for realtime predictive maintenance. We build a neural network model that combines it's strengths. We have been also able to provide an architecture for real time predictive maintenance of industrial dataset using Apache Spark and AWS. As the results from the previous chapter show, we have been able to leverage the the Reencoder transition model in Grid LSTM for predicting sequences of failure events in a better way than the standard LSTM Model. After applying batch normalization and the scheduled sampling learning strategy, it is surprising that our model was able to outperform the standard LSTM model.

We can also conclude that the choice of the appropriate loss function has a huge impact regarding the generated future frame predictions, if not even the most tremendous effects. However, as the evaluation in chapter 6 clearly shows, there is no perfect solution for this purpose. But it must be emphasized that the detailed properties of the sequences have to be analyzed in detail, in order to be able to achieve good results. Nevertheless, the fine-tuning of neural networks in context of sequence labelling tasks remains to be very difficult even with a good loss function at hand.

Finally, in can be assumed that the proposed model could be currently trained more effectively using a different deep learning framework than TensorFlow, at least at the time of this writing. This can be argued with the fact that the current batch normalization layer in this framework currently depends on some operations where no GPU kernel is implemented yet. Such a bottleneck might be the root cause why the training process of our model is so slow that it requires up to four days to train the network for only 100,000 steps.

Chapter 15

Future Work

There are at least the following five proposals for future work:

- Firstly, the proposed network model should be examined and fine-tuned in more detail. We strongly belief that this architecture is able to obtain even better results after performing a more extensive hyperparameter search, training it for many more iterations. Unfortunately, this is beyond the timeframe of this thesis, that is why some evaluations were performed on networks which still had further potential in case of more training iterations.
- Secondly, because the application of the scheduled sampling learning strategy for recurrent networks has improved our results in such an extent, it would be worthwhile experimenting with new variants of this approach. For instance, the recurrent network could dynamically grow in the course of the training process. Thereby, it could start to predict a single frame only until the validation loss reaches a specified threshold.. As a result, such a network should be able to predict longer sequences with a higher stability.
- Thirdly, we can try different architectures with different number of layers and nodes.
- Fourthly, we can try predicting RUL (regression). Our work considered a binary classdification approach. It remains to be seen whether RUL based approach could outperform the binary classification approach.
- Lastly, the proposed network architecture itself can be further extended to cope with different unsupervised tasks. Our model only considers identifying failure patterns for battery data but it can be applied to a varied range of predictive maintenance scenarios where multiple other data sources are involved such as maintenance

records.

Chapter 16

Appendix

16.1 Source Code

The source code of the project is available at https://github.com/TrustyandTrue/PredictiveMaintenance

List of Figures

Bibliography

[1] Martin E.P. Seligman, John Tierney *We Aren't Built to Live in the Moment* May 2017: NY Times.

[2] Samy Bengio, Li Deng, Hugo Larochelle, Honglak Lee, and Ruslan Salakhutdinov *Guest Editors' Introduction: Special Section on Learning Deep Architectures* August 2013: IEEE Transactions on Pattern Analysis and Machine Intelligence

[3] Pooyan Safari *Deep Learning For Sequential Pattern Recognition* December 2013

[4] CM Bishop. Pattern recognition and machine learning. 2006. ISBN 9780387310732

[5] Philippe Hamel and Douglas Eck. Learning Features from Music Audio with Deep Belief Networks. ISMIR, (Ismir):339–344, 2010. URL http://musicweb.ucsd.edu/ sdubnov/Mu270d/DeepLearning/FeaturesAudioEck.pdf.

[6] Kshitij Tayal. Retrieved from https://www.slideshare.net/kshitijtayal/cpg-island-identification-with-hi

[7] Honglak Lee, Peter Pham, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. pages 1–9, 2009a.

[8] Anish Singh Walia. Retrieved from https://medium.com/towards-data-science/types-of-optimization-a

[9] Chris Olah. Retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[10] Andrej Karpathy Retrieved from http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[11] G. Dahl, D. Yu, L. Deng, A. Acero, "Large vocabulary continuous speech recognition with context-dependent DBN-HMMs", Proc. ICASSP, 2011.

[12] Taylor, G. W., Hinton, G. E., and Roweis, S. T. (2006). Modeling human motion using binary latent variables. Advances in Neural Information Processing Systems. MIT Press.

[13] Ilya Sutskever, Geoffrey Hinton, and Graham Taylor. The Recurrent Temporal Restricted Boltzmann Machine. NIPS'08 Proceedings of the 21st International Conference on Neural Information Processing Systems Pages 1601-1608. 2008.

[14] Galen Andrew, Jeff Bilmes. Sequential Deep Belief Networks. Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference. 2012

[15] Itamar Arel, Derek Rose, Robert Coop. DeSTIN: A Scalable Deep Learning Architecture with Application to High-Dimensional Robust Pattern Recognition. 2009

[16] Subutai Ahmad, Scott Purdy. Hierarchical Temporal Memory – Real-time Anomaly Detection. 2016

[17] Yuwei Cui, Subutai Ahmad, Jeff Hawkins. Hierarchical Temporal Memory – Continuous Online Sequence Learning with Unsupervised Network Model.2016.

[18] Alex Graves,Santiago Fernandez, Jurgen Schmidhuber. Multi Dimensional Recurrent Neural Networks. 2007.

[19] Alex Graves. Bi-directional LSTM and Hidden Markov Model - LSTM. 2012.

[20] Jan Koutnik, Klaus Greff, Faustino Gomez, Jürgen Schmidhuber. Clockwork Recurrent Neural Networks. 2014

[21] Ilya Sutskever, Oriol Vinyals, Quoc V. Le. Sequence to Sequence Learning with Neural Networks. 2014

[22] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, Yoshua Bengio. Gated Recurrent Neural Networks on Sequence Modeling. 2014.

[23] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, Yoshua Bengio. Gated Feedback Recurrent Neural Networks. 2015.

[24] Jurgen Schmidhuber, Bas R. Steunebrink, Jan Koutnik, Rupesh Kumar Srivastava, Klaus Greff. Vanilla-LSTM. 2015.

[25] Liang Lu. Sequence Training and Adaptation of Highway Deep Neural Networks. 2016.

[26] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, Chris Dyer. Depth Gated LSTM. 2015.

[27] Nal Kalchbrenner, Ivo Danihelka, Alex Graves. Grid LSTM. 2016.

[28] Tara N. Sainath, Bo Li . Frequency LSTM, Time-Frequency LSTM and ReNet-LSTM. 2016.

[29] Daniel Neil, Michael Pfeiffer, Shih-Chii Liu. Phased LSTM: Accelerating Recurrent Network Training for Long or Event based Sequences. 2016.