Literature review and pre-study - Bachelor Thesis DD152X

Karl Söderbärg
KTH Royal Institute of Technology
Stockholm, Sweden
kasoderb@kth.se

Max Xie
KTH Royal Institute of Technology
Stockholm, Sweden
maxxie@kth.se

March 13, 2018

1 Introduction

1.1 Background and Scope

1.1.1 Description of the field that the thesis work is done in (e.g. connection to research/development, state-of-the-art)

This thesis will attempt to help institutions to better handle operational risk and improve data quality with the help of machine learning. Using machine learning to improve data quality in datasets by finding anomalies and outliers has been relevant for a long time. In this paper we will be focusing on a subcategory of machine learning, namely, deep learning. Using deep learning to improve data quality is a field that has not been very well explored, and no clear state-of-the-art method has yet been discovered.

Operational Risk has long been a problem for financial institutions. Operational risk has several definitions, however, the most prominent one, defined by the Basel II committee is as follows: "The risk of loss resulting from inadequate or failed internal processes, people and systems, or from external events. For example, an operational risk could be losses due to an IT failure; transaction errors; external events like a flood, an earthquake, or a fire." (Karam and Planchet, 2012). It is hence hard to account for in risk modelling. However, this does not make them less dangerous. The classic example of an operational risk is the English rouge trader Nick Leeson who, through a series of deception and fraud accumulated an operational loss of $1.3 billion, which in turn caused the bankruptcy of the English bank Barings. Many similar cases exist, caused by both frauds and mistakes, and they have all had a devastating effect on the responsible bank. However, recent studies have shown that on top of the direct monetary damage, operational loss events also pose a reputational risk. The reputational damage causes volatility in share price and may effect sales and customer relations. A study from 2017 of 300 events of operational loss events in British and American banks showed that the average loss due to reputational damage was $113 million. (Jiang, 2018)

Attempts have been made to mitigate the damage caused by operational risk through the use of special insurance policies and regulation to protect financial institutions. However, no previous research, to our knowledge, has attempted to reduce occurrence of operational risks using machine learning.

Our hope is that this thesis can serve as a proof of concept, that proves deep learning for solving operational risk as a tractable option. To our disposal, we also have access to vast amounts of bank data, with which we can use to train and test our systems. Our aim it that by combining the two fairly unexplored fields of operational risks and deep learning we may highlight a new approach to mitigate and neutralize certain types of operational risks. This would enhance the security for both financial institutions, and its customers.

1.1.2 The interest of the employer. Background to the specific task that should be solved.

Specifically, this project will be focused on minimizing operational risks for Handelsbanken AB. The interest behind this task is due to the vast amount of data that the company creates and handles on a daily basis. Since millions of rows of data (usually in Excel) are created or modified every day, and often by humans, the system is prone to human error. Data can flow through many departments and systems within the company, getting extracted and modified along the way. This creates a situation where one small error at one point can cause unexpected consequences a long way down the line, much like a butterfly effect. There has been previous cases in other similar companies where this type of errors have caused large, unnecessary costs. Finding a good solution to these kinds of operational risks can thus potentially be very beneficial and have big real world implications.

1.1.3 The goal with the thesis and its news value. What is the desired result? Why would anyone want to read the finished work? And who would these people be?

The goal with this report is to examine the usage of deep learning as a means to lower the operational risks and increase data quality, specifically in businesses that are handling vast amounts of data, such as a bank. In our case the errors are mainly due to human error. However, we would like to achieve a result that could be applied to increase data quality, regardless of what factors caused the data quality to be poor. Today, the previous work done in this subject is very limited, therefore, we find that there might exist a news value in doing this work.

Specifically, we would like to examine if it is possible to achieve some generalized algorithm that will be able to increase data quality regardless of the internal structure of the data and regardless of the factors that caused the data quality to be poor. Such a generalized algorithm could be of a large value to society as the amount of data in our world increases exponentially. Errors that occur early can, down the line, be amplified and eventually cause unexpected consequences, it is therefore of paramount importance that these types of errors are detected early. The people that this paper mainly would be of interest to are companies and institutions that today handle vast amounts of data, especially if corporate decisions are made based on this data.

1.1.4 Social and ethical aspects. Are there any ethical aspects or a bigger social perspective that affects the work?

Poor data quality is a big driver for random and unnecessary losses for companies and institutions. Improving this will help companies make better decisions and investments into society. The ethical aspects, of course, might vary depending on how the work is used. We think that when applied to a state authority it might help counteract erroneous data that eventually could lead to bad decisions, negatively effecting certain citizens or groups of citizens in a country. In other words, we think that helping people making more intelligent and correct decisions also will help people become better at making contributions to society. From a social perspective it shall be noted that only companies with large amounts of data (often the very large companies) will see direct benefit from the result of this work. Although, this might create further imbalance, we reason that more security at great institutions also will benefit their customers and smaller players.

1.2 SCIENTIFIC QUESTION

1.2.1 The question that will be examined. Stated as a explicit and revisable question.

Is it possible to use unsupervised deep learning to build a tractable model that identifies anomalies and hence, neutralizes operational risk in large sets of data?

1.2.2 Specified definition of the problem. (For instance, what is the nature of the task and what challenges are there?)

With the approach we have chosen, we are faced with a machine learning problem. More specifically, we will be given a large data set from the core of a banks information flow. As of this writing we have not gained access to the data yet. Due to the sensitivity of the data, it will be delivered to us in a encrypted format, and we also won't be knowing its exact source or what the data is being used for.

The dataset will not be labeled, but we have been informed that it is safe to assume that an absolute majority of the data points will be unlabeled and non-anomalous. Furthermore, we know that the dataset will contain some unlabeled but anomalous points. An anomalous field is an outlier piece of information that we hope will attract the attention of our neural network, just like how a human could look at it and notice that something was amiss.

The nature of the task is to build neural networks to find anomalies in large sets of data. The main challenge is that we don't quite know whether or not this is feasible, due to the lack of published previous work on this exact topic. Building the model itself will therefore be the biggest challenge. Another challenge is that we need a good understanding of the practical implications of what we are trying to solve. For example, how sensitive do we want our model to be to match the risks that we find in real world data? Will we be preferring many false positives, or should we risk true positives to slip by? Another challenge is guaranteeing the cleanliness of the datasets. Mainly when it comes to the test set, since it needs to be quite large with intentional anomalous data injected into it, which are the data points that we test the model to detect. The challenge arise when we, before the injection or anomalous data, need to have a data set, that is guaranteed to have no anomalous data whatsoever. Which is a hard task when the data sets are both encrypted and very large.

1.2.3 Expected scientific result. In what ways is this work relevant and which is the hypothesis that is going to be tested? How will this hypothesis be tested?

Hypothesis 1.1 *Machine Learning with a neural network can be used to ensure data quality by identifying anomalies in bank data, with such a satisfactory result so that mitigation of some operational risk is possible.*

However, our main expectation of our report is that it is of a exploratory nature, exploring the hypothesis and its feasibility.

The expected result is that our neural network will be able to identify anomalies in the given dataset with a satisfactory accuracy. Which will be evaluated using a validation set of labeled sets of test data. We do not yet

have a clear apprehension as to what threshold of the result we will use to class an acceptable anomaly detecting model. We reason that this threshold will depend on the data we are given, as well as what real world implications different results might have. If we manage to solve this task using this approach, we will have a solution to a data quality problem that will have direct ramifications and benefit on risk modelling for financial institutions.

2 Theory

2.1 Theory. Theoretical review on relevant parts of the field that is going to be examined and the methods that are going to be used.

2.1.1 Machine Learning - What we are doing

Machine learning can generally be described as a computer system that has the ability to "learn", that is to progressively improve performance on a specific task as the amount of available data increases, without being explicitly programmed. It is a field that has been examined for a long time, and with time, has developed in several directions.

The popularity among the different methods have through time varied a lot as new discoveries and methods have been discovered. One of the most basic machine learning method is the "linear regression", where the task is to fit a line on a given set of data points. Linear regression fits the definition of machine learning since the line, because the computer learns and achieves a lower error as the amount of data is increased.

One method that has been around for a very long time is the Support Vector Machine. Which, for a long time was the model that achieved state-of-the-art results in a wide variety of tasks in the field of machine learning, up on till Hinton's use of a Neural Network, beating the Support Vector Machine state-of-the-art result on the MNIST dataset, leading to a Neural Network Renaissance.

2.1.2 Neural Networks - How they work

Neural networks is a category in machine learning that has gained a lot of attention lately. In simple terms, it is based on mapping some input to an output via matrix multiplications and non-linear transformations. The learning takes place by iteratively adjusting the matrices that are used for multiplication and linearly transforming whatever input it receives. Since, we don't only want to limit ourselves to linear transformations, neural networks are built by combining many different algorithms, and the usage of non-linear functions.

There are many decisions that has to be made before implementing a neural network. Those include choosing some viable architecture, that is how many layers the network should consist of and how many nodes each layers should have. Furthermore, one needs to choose a good non-linear transformation, where the most common ones are the sigmoid function, rectified linear unit and the hyperbolic tan function. Lastly, a good optimization function needs to be chosen and the hyperparameters needs to be tuned, which is oftentimes considered one of the hardest parts in training a network. Of course, there are a lot more to be done to neural network that will make it function better, but that is outside the scope of this report.

2.1.3 Auto Encoders - What they are

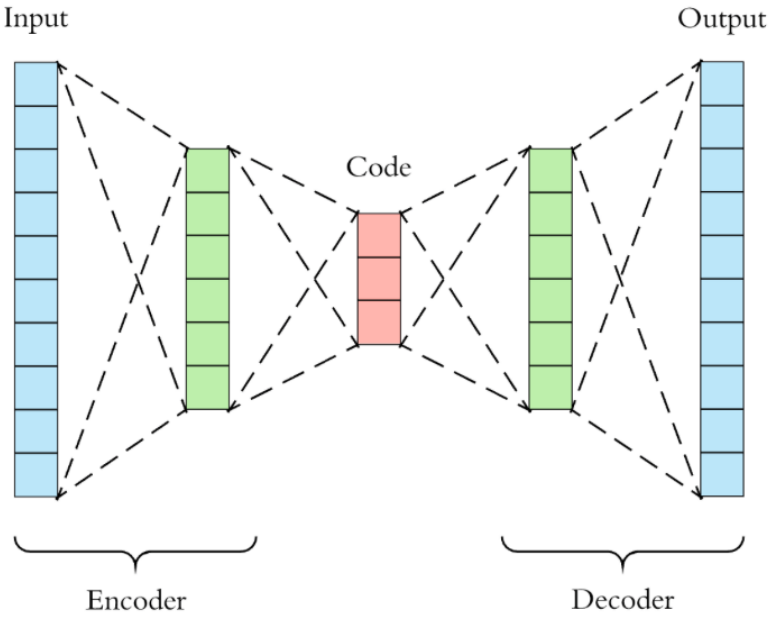
Auto Encoders describe an idea given by a certain type of neural network architecture that comes with some particular attributes. The idea is a neural network that is trained to attempt to copy its input to its output. In theory, the input is first translated into a intermediary code that is used to represent the input. This code is then re-translated to an output that tries to reconstruct the input. In other words, the network is trying to learn an encoder function $h = f(x)$ and a decoder function $r = g(h)$ so that $g(f(x)) = x$. This might seems counter-intuitive at first as this would easily be solved by letting $g(f(x))$ be the identity function. Therefore, we actively try to prevent it from learning the identity function, and we have several regularizing methods to choose from.

Traditionally, the main purpose of the autoencoder is to learn features and reduce the dimensionality of the input data, as well as generative modelling. This is most often done by constructing a neural network architecture where the number of nodes in the middle hidden layers in the network consist of less nodes than that of the input and output layer, forming an hour glass like structure. It is theorized that this forces the input data to be compressed into a lower, code dimension, therefore the network has to prioritize which features are important and how to find an efficient representation of the input data.

Digging deeper into autoencoders we find that they exploit the assumption that all examples concentrate around some low-dimensional manifold or some small set of manifolds. One important principle is that the autoencoder can afford to represent only the variations that are needed to reconstruct the training examples. This requires good knowledge about the underlying distribution and the structure the training examples. Not only can such knowledge be used to generate completely new examples no one has every seen before, but it can also serve as an efficient non-linear anomaly detector.

If it is not intuitive how an autoencoder can be used to detect anomalies, consider the following example. We build an autoencoder that takes a picture of a dog as the input and tries to reconstruct the same dog picture as the output. Assuming we have trained it with sufficient amount of dog pictures it will be able to reconstruct a

Figure 1: This is a general visual description of an autoencoder. Note that the intermediary code is built by having an hourglass like structure, where the amount of units in the bottleneck is significantly less than those in the input layer. Also note that the amount of units in the input layer and the output layer, which may seem trivial since the objective is to make the output as similar to the input as possible.



dog picture with low reconstruction error. The reconstruction error is defined as a function that measures the dissimilarities between the input and the reconstructed output. A low reconstruction error implicitly means that the autoencoder has learned a particular low-dimensional manifold that the training examples are concentrated around. Now if we were to present a cat picture as input to the autoencoder instead, since the code of this picture is a point in space far away from the learned manifold, we expect it to render a high reconstruction error, and thus anomalies are detected on the inputted data point.

2.1.4 Our approach

Our main objective is to find whether or not we can build some tractable model that can detect anomalies on whatever data we give to it. Although, it is hard to know which model and strategies will be the best, we do have some qualified guesses as to which will deliver more satisfactory results. One algorithm that we will focus extra much on, is the previously mentioned autoencoder. In this approach we plan to use some regularized autoencoder, that, according to several papers, can efficiently learn the underlying distributions of some data, even if the internal relationships in the data is non-linear. As we previously mentioned, we reason that learning such a distribution can be an efficient way of finding anomalies. There have been some previous examples of this in published papers. However, all of them that we have found so far are on visual data. Furthermore, we will be adding on several algorithms and regularization methods to our model to further improve its performance, such as dropout, zero-bias and unsupervised pre-training. Each strategy and method will be motivated by previous published scientific research, some of which will be found in this pre-study.

If there are more time we will try some other interesting approaches that we hypothesize might work well. Among them are specific ensemble architecture for several regular feedforward neural networks and other generative models, like the generative adversarial network, deep Boltzmann machine and the deep belief network. We would also like to examine how well we can combine deep learning methods with traditional statistical methods to better motivate our choices.

3 Previous work

To be frank, we have not found any previous work on the subject of the task that we are trying to solve. That is using machine learning as a means to counteract operational risks. Even anomaly detection using autoencoders and neural networks have only seen limited publications. Our approach will therefore be to start at a very low level of abstraction by studying fundamentals and strategies that we can use to optimize our model. We therefore reason that our studies should have a wide focus, by building a large toolbox of possible methods and strategies. We motivate this with the fact that there are no published work that offer clear guidelines of which methods works well and which don't, for this kind of task. Thus, we also expect our working process to be iterative where we test different scientifically backed strategies, so that we can motivate our choices and keep the methods that work and

discard those that don't. Hopefully, we end up with a viable solution that we can back and motivate with previous scientific results.

3.1 Learning From Data By Yaser S. Abu-Mostafa, Malik Magdon-Ismail and Hsuan-Tien Lin

3.1.1 Chapter 1 - The Learning Problem

A machine learning algorithm can be as simple as:

$$h(x) = sign\Big(\Big(\sum_{i=1}^{d} w_i x_i\Big) + b\Big)$$

In machine learning, different models can roughly be divided into two categories. Supervised learning where the training examples are paired with labels that tells what the correct output should be, and unsupervised learning where no such explicit labels or information exists. A good rule of thumb is that unsupervised learning requires more training data.

An important aspect to machine learning is that the algorithms are faced with an in-sample error which is the fraction between the total number of training examples and the number of times the true function f and the hypothesis function h disagree:

$$E_{in}(h) = \frac{1}{N}\sum_{n=1}^{N}\big[h(x_n) \neq f(x_n)\big]$$

Furthermore, the models are also faced with an out-of-sample error which corresponds to the probability that the hypothesis function h fails to correctly predict test data, which are not included in the training:

$$E_{out}(h) = P\big[h(\mathbf{x}) \neq f(\mathbf{x})\big]$$

Since $f(x)$ is unknown, the feasibility of the model can be evaluated with Hoeffdings Inequality for any $\epsilon > 0$:

$$P\big[E_{in}(g) - E_{out}(g)| > \epsilon\big] \leq 2Me^{-2\epsilon^2 N}$$

Where N is the size of the the training dataset. M is the size of the set of all hypothesis functions that we want to test.

3.1.2 Training versus Testing

The Hoeffding Inequality as presented above will not be useful in the case of infinite hypotheses, because M, and hence the bound will approach infinity. The theory of generalization solves this problem by formalizing the concept of "effective number of hypotheses". In the case of infinite hypotheses there will often be an overlap among some hypotheses. From this concept we derive the VC (Vapnik-Chervonenkis) Generalization Bound for the statistical tolerance δ:

$$E_{out}(g) - E_{in}(g) \leq \sqrt{\frac{8}{N} ln \frac{4m_H(2N)}{\delta}}$$

The growth function m_H is the maximum number of dichotomies that can be generated by the hypothesis set H on any N points. The VC dimension is the order of the polynomial bound on m_H. Hence it is concluded that while a more complex learning model might be more exact in representing the training the, there is also a penalty for using a more complex model. Statistical Bias-Variance analysis is a conceptual alternative to evaluating the problem with VC analysis which can be helpful when developing a model.

3.1.3 Chapter 3 - The Linear Model

Non-separable data in one dimension can be separated linearly in an other dimension.

The concept of linear regression is discussed.

Gradient Descent is a general technique for minimizing a twice-differentiable function.

SGD (stochastic gradient descent) is a version of gradient descent that in each iteration instead of trying to minimize the error on all data points minimizes the error in one randomly chosen data point. This randomness gives the advantage of helping the algorithm avoid flat regions and local minima in a complicated surface.

The linear model (for classification or regression) is an often overlooked resource. Efficient learning algorithms exist for linear models, they are low overhead. There are also very robust and have good generalization properties.

3.1.4 Chapter 4 - Overfitting

Overfitting is the phenomenon of fitting an algorithm too well on the training data hence picking a hypothesis with lower E_{in} which results in a higher E_{out}. Catalysts for overfitting include high hypothesis function complexity and noise in the data set. The simplest cure for overfitting is to add more training data or to introduce some regularizing term.

Regularization is an important technique that does not require more data points. The essence is to concoct a measure $\Omega(h)$ for the complexity of a hypothesis and instead of minimizing $E_{in}(h)$ alone, it minimizes the combination of E_{in} and $\Omega(h)$. Most methods of $\Omega(h)$ are heuristic methods that are grounded in a mathematical framework developed for special cases.

Validation is another important counter measure against overfitting. The simple idea is that we reserve a piece of our training data for testing the produced algorithm. Adding this to the evaluation, of course, has the consequence of leaving us with less training data, but effectively prevents us from producing a model that only is effective on the data it has been trained on.

Cross validation, a strategy that is closely related to the concept of k-folds, significantly mitigates the problem of reducing the training data set. In essence the method divides the training data set into k groups. Each group is then separated to act as the validation set giving us k different error measures as:

$$e_n = E_{val}(g_n^-) = e(g_n^-(\mathbf{x}_n), y_n)$$

The cross validation estimate is then the average value of the e_n's:

$$E_{cv} = \frac{1}{N}\sum_{n=1}^{k} e_n$$

Using advanced validation techniques is powerful because the usefulness of estimating E_{out} expands outside the scope of comparing the single hypotheses h. An important usage of validation includes selecting the best hypothesis set H (our models) and to tweak the model in selection of optimal hyperparameters, such as the learning rate and how harsh the regularizer should be.

3.2 Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville

3.2.1 Summary

The purpose of the textbook "Deep Learning" is to compress the field into a coherent text book explaining the key concepts of the field. It does do by offering a good overview of the history of deep learning methods. It starts from the beginning and ends with describing the state-of-art methods as of the year of its publication (2016). This textbook served as a good source of inspiration when thinking and breaking down our task. It also offered good guidance as to which papers have been influential in this field and other good pieces of advice for people who wish to dive deeper into the subject.

Summarizing the entire book within this pre-study is not tractable, although there were some key takeaways that we think will be very relevant to our task. They are spread across several chapters and we will be discussing some of them below.

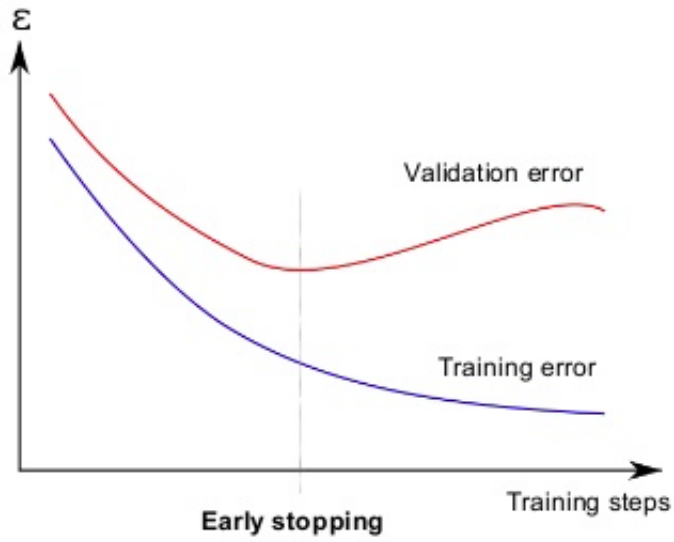
3.2.2 Chapter 7 - Regularization for Deep Learning

Beside from the normal regularization methods, such as L^2 and L^1 regularization. The textbook discusses other more new regularization methods. Two subjects this chapter emphasizes are dataset augmentation and noise robustness. It has been known for a long time that neural networks prove not to be very robust to noise. One way to increase a model's noise robustness is to use data augmentation by intentionally adding random noise to the input data. The textbook motivates this method with the following quote: "In other words, it [adding random noise to input data] pushes the model into regions where the model is relatively insensitive to small variations in the weights, finding points that are not merely minima, but minima surrounded by flat regions."

Furthermore, the idea of early stopping is discussed in this chapter. When training, we run through the same training data set several times, each such iteration is called an epoch. Given that the model have a high enough representational capacity, it will eventually overfit the training data, which we prevent using early stopping. The concept is depicted on the figure below, where each training step is an epoch. One basic algorithm is to introduce a patience variable that resets to zero every time an iteration produces a better result on the cross-validation set. However, if it produces a worse result, the patience increases, until a threshold is reached on which the training is stopped.

The chapter continues by discussing the usage of sparse representations, where the main idea is to penalize activations of the units in the network. In other words, we encourage that the matrices representing the connection between layers to mainly consist of zeros, this is a widely used regularization method. Moving on, we also discuss bootstrap aggregating, where we train several different models separately, and then have the models vote on the predictions for test examples, this is called model averaging. Here it is really important that the different models are as uncorrelated as possible, so that they can correct each others mistakes. However, the book also states that:

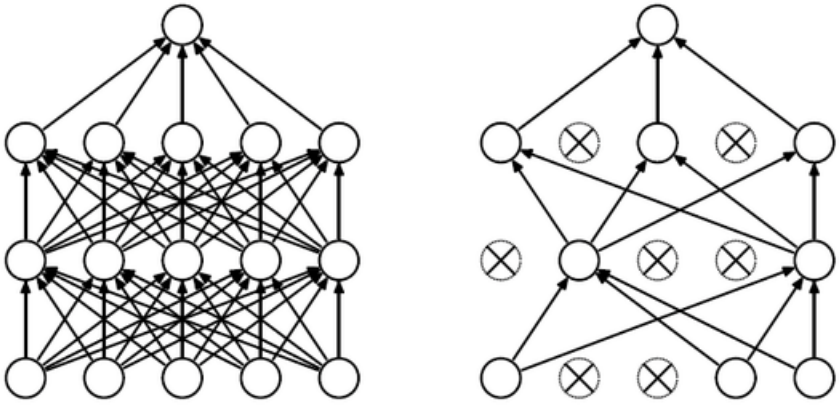
Figure 2: A visual demonstration to the motivation of early stopping.



"Neural networks reach a wide enough variety of solution points that they can often benefit from model averaging even if all the models are trained on the same dataset. Differences in random initialization, in random selection of minibatches, in hyperparameters, or in outcomes of nondeterministic implementations of neural networks are often enough to cause different members of the ensemble to make partially independent errors. Model averaging is an extremely powerful and reliable method for reducing generalization error."

Lastly, we have the idea of Dropout, coined by Srivastava in 2014. It trains several subnetworks created from original network, where we randomly have chosen units to remove. We then apply a bootstrap aggregating method to this ensemble of subnetworks for inference. The motivation for this method is that dropout trains an ensemble of models that share the same hidden units, and therefore forces each individual hidden unit to perform well regardless of which other hidden units that are also active in the network. Therefore, it regularizes so that each hidden unit not only to perform well by itself but also within its context. This method has been proven to work very well, as it works on most types of models and it can be trained with most optimization methods.

Figure 3: Dropout randomizes several versions of the network where randomly chosen units' output value is multiplied with zero, i.e. removed. This is one example of how one such model could look like.



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

3.2.3 Chapter 8 - Optimization for training deep models

In this chapter we learn what challenges we can encounter when trying to optimize a deep neural network. One central issue is finding the best local minima. With a cost function spanning a non-linear field, with multiple differently valued local minimas there is a risk that a gradient based model will get stuck in a local minima with a relatively high cost. However, on the other hand, the book states that experts now believe that, for large neural

networks, most local minima have low cost value. Therefore, with this in mind, it is no longer as important to achieve the global minimal cost. The cost of the most minimas in some cost function are therefore considered, "low enough".

Problems also arise when the model encounter saddle points and other flat regions. The textbook states that we can expect that the ratio between the number of saddle points and the number of local minima to grow exponentially as input dimension increases. Although this field is not yet fully understood, Goodfellow argue that continuous-time gradient descent, analytically, could efficiently escape a saddle point. However, this could pose a serious issue for models that optimize using Newton's method, since, in contrast to gradient descent, it is designed to solve for a point where the gradient is zero-valued, risking to immediately jump to a saddle point.

Furthermore, there is also a risk of encountering cliff structures in the cost function landscape. These consist of a "wall" where the value of the cost function abruptly increases or decreases. Having a too high learning rate can cause the model to jump onto the top of the cliff, reversing much of the iterative optimizing work. This problem can be avoided using gradient clipping. Which clips the gradient and prevents it to take a too big step. $gradient_{new} = \frac{gradient}{\|gradient\|} * threshold$

Moving on, we discuss the different optimization algorithms that the developer can choose from. Beginning with momentum whose idea is to accumulate an exponentially decaying moving average of past gradients and continues to move in the direction of those previous gradients. To help it slow down and converge on a local minima, we also introduce a friction variable that decreases the speed by some value that is proportional to the current speed. The idea with momentum was to design and speed up the learning while still working well in situations of high curvature, noisy gradients or in small but consistent gradients.

One of the harder hyperparameters to tune in a deep neural network is the learning rate. This is because there are no clear rules how to pick a good learning rate. This might seem worrying since the learning rate might be one of the most crucial hyperparameters. Therefore, algorithms using adaptive learning rates have been developed, some examples of them are AdaGrad, RMSProp and Adam. Those algorithms differ somewhat, but the main idea remains the same, which is to be able to change the learning rate dynamically during training, depending on the current situation in the optimization process.

With all these optimization algorithms it can be difficult to tell which one to use in a given situation, the research community have unfortunately not yet reached a common consensus on this point. However the textbook states that: "Stochastic gradient descent with momentum, which was used to train neural networks in the 1980s, remains in use in modern state-of-the-art neural network applications." Which can seem reassuring.

3.2.4 Chapter 11 - Practical methodology

"A good machine learning practitioner also needs to know how to choose an algorithm for a particular application and how to monitor and respond to feedback obtained from experiments in order to improve a machine learning system."

In this chapter the book offers the reader some practical methodology that will be useful when implementing the models for practical use. As previously mentioned it states that a reasonable default choice of an optimization algorithm is the SGD (Stochastic gradient descent) with momentum and decaying learning rate. If the training set consists of several million examples or more there is not much need for regularization, the exception being early stopping which should be used in all cases.

Choosing good hyperparameters for a given machine learning problem is very difficult, especially since we don't have a very good understanding of how to do that efficiently, and more specifically, how to know that we have picked good hyperparameters. The textbook argue that it is, in principle, possible to develop hyperparameter optimization algorithms that analyzes a learning algorithm and chooses its hyperparameters. However, this requires hyperparameters of their own, although, these new hyperparameters tend to be simpler. Another popular way of picking good hyperparameters are using random search. Basically, it is done by randomly choosing values for each hyperparameter, train the model and cross validate it. The textbook argues that random search can be exponentially more efficient than grid search, where you prepare sets of hyperparameters and iterate through every one. The reason for this is that there are no wasted experimental runs in random search, which exist in grid search as it tend to unnecessarily repeat two or more equivalent experiments.

When it comes to debugging it can be said that machine learning systems are not easy to debug, this is due to several reasons. When a network produces a result, there is little or no way to tell whether or not this result is good or not. You could have gained this result, even with some of the parts in the system broken, without you ever knowing that they are, in fact, broken. The advice that the textbook offers boils down to six main points.

1. Visualize the model in action.
2. Visualize the worst mistakes.
3. Reason about software using training and test error.
4. Fit a tiny dataset.
5. Compare back-propagated derivatives to numerical derivatives.
6. Monitor histograms of activations and gradients.

3.2.5 Chapter 12 - Applications

This chapter discusses some practical advice on the application level. It begins with emphasizing why running neural networks on a GPU is superior than running it on a CPU. That is because the GPU is optimized when it comes to computing matrix multiplications, which is an operation on which neural networks rely heavily. The GPU also have a high memory bandwidth, which a regular CPU does not have.

The chapter moves on to discuss data parallelism and model parallelism. These are tricks that try to reduce the computational cost of training or inference, by dividing the problem onto several independent computing units. Model parallelism is described as a model where different machines can collaborate on a single data point in order to train the model or infer a prediction. This is often done by allowing the machines to compute different parts of the model. Data parallelism, on the other hand, is described as dividing up the data into batches and let each machine get one batch to train the model, the idea is to combine the result from every machine and produce one single optimized model. However, data parallelism is rather unusual and difficult to accomplish, due to, for instance, the fact that gradient descent is a completely sequential algorithm. The textbook present a possible solution, by using asynchronous stochastic descent, which is a method were many processor cores share the parameters for the model. Each core computes their own gradient on their own batch of data, updates the parameters and update the parameters of every other core. Although, this method let cores overwrite the progress of other cores, it seems to speed up the training overall. Today, distributed asynchronous gradient descent is one of the primary strategies for training large deep networks and is a common practice for most major deep learning groups in the industry.

The chapter moves on to talk about dynamic structure, which are a collection of methods that try to achieve a good approximation of the real inference, and at the same time, cut computational time and cost. One way of doing this is conditional computation, which decide which computations to exclude after having seen the input. This helps by not forcing us to pay the cost for full inference for each and every datapoint. Conditional computation can be achieved with the strategy of cascading, where a sequence of classifiers are set up, often in the order of increasing capacity. New datapoints that are fed into the cascade begin at the model with the least capacity, if it passes it goes on to the next, more complex model. However, if it fails, the inference process is stopped and the datapoint gets classified with whatever class is associated with the datapoint not passing. This way we can early eliminate all of the "easy" datapoints without having to run them through the entire much more complex classification model. Therefore, this is often used to detect objects that occur rarely, since the earlier, more simple classifiers have low capacity and trained to have high recall, they won't wrongly reject an input where this, sought after, rare object is present. A problem that often occurs in cascading is that the first classifiers in the cascade tend to become overloaded, whilst the more complex classifiers tend to be underloaded.

We can also use a neural network called the gater whose task is to look at the incoming datapoint and pick one of several experts networks, whichever the gater finds the most appropriate, the strategy of only picking one expert network is called a hard mixture of experts. It is worth to note that this strategy won't work well when the number of gating decisions is large since it is likely to be combinatorial. One alternative design is to have a gater output a set of weights for different expert networks. Those networks computes their predictions, which are then weighted and averaged according to the set of weights specified by the gater.

3.2.6 Chapter 14 - Autoencoders

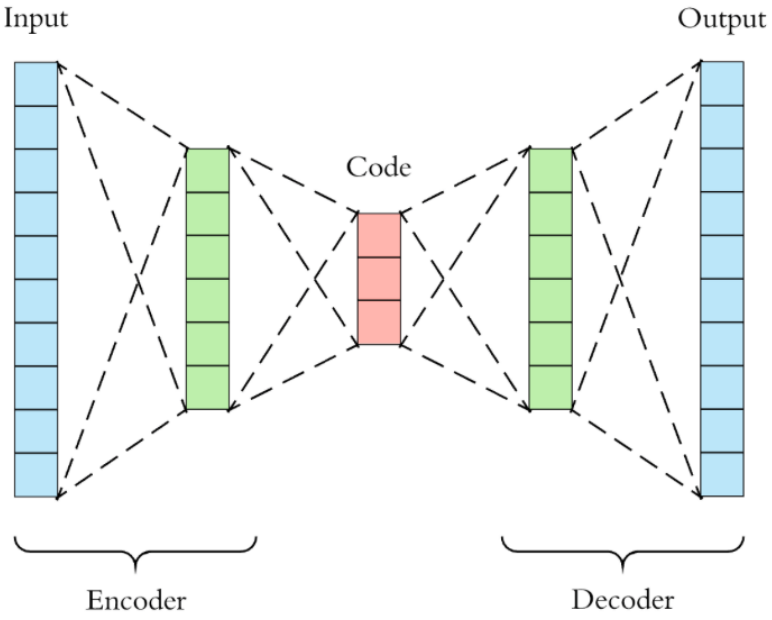
In this chapter, we go back to the concept of autoencoders, which is a neural network that we train to copy its input to its output. The idea is that one network compresses the input in a hidden layer h. This compressed representation of the input is called the code. This code is then ran through another network that attempts to reconstruct it to the original input. Thus, we can say that the autoencoder consists of two parts. The encoder function that produces the code, $h = f(x)$ and the decoder function the reconstructs it, $r = g(h)$. At the same time we also restrict the network from learning the identity function $g(f(x)) = x$, by applying some regularizing factor. Thus, the model is forced to prioritize which features should be copied and therefore has to learn salient properties of the data.

The cost function we will be trying to minimize is some $L(x, g(f(x)))$ where L measures the difference between the original input x and the reconstructed version $g(f(x))$. Just like principal component analysis, autoencoders can be used for dimensionality reduction and feature learning, it is however known that its encoder and decoder functions can learn more powerful nonlinear generalizations of the data than principal component analysis. One of the most relevant feature of the autoencoder is its ability to find useful information about the distribution of the data. However, if we allow the autoencoder to much capacity it might learn the copying task without extracting much information about the distribution of the data. This can be seen as a type of overfitting, which is a smooth segue to the next part, how to regularize autoencoders.

One key aspect is to match the capacity of the autoencoder to the complexity of the underlying data distribution. Therefore we want the autoencoder to learn other abilities than just copying its input to its output. When it comes to finding good regularizers for the autoencoder, we often have to make a trade-off between two opposing forces.

1. Learning a representation of the input x. This representational code is deemed good if it is able to be reconstructed fairly well by the decoder function. An autoencoder is said to be overfit if it does this task too well, e.g. when it has learned the identity function.

Figure 4: We reuse this figure again for the purpose of facilitating the explanation of the autoencoder concept.



2. Preventing overfitting with good regularization, often by limiting the capacity of the autoencoder or adding reconstruction cost. Generally this means that we encourage the model to be less sensitive to the input data and stops it from learning the identity function.

One of the most important features of a regularized autoencoder, is that the code of the input x can implicitly be represented by being concentrated along some low-dimensional manifold. "Hence the encoder learns a mapping from the input space x to a representation space, a mapping that is only sensitive to changes along the manifold directions, but that is insensitive to changes orthogonal to the manifold. [...] by making the reconstruction function insensitive to perturbations of the input around the data points, we cause the autoencoder to recover the manifold structure." So, the manifold characterizes the representation of the data points, on a given example, this representation is called its embedding. Being insensitive to small changes in the input data directly translates to it being regularized. In conclusion, autoencoders learn manifolds by balancing the two opposing forces above.

One way to achieve good regularization is to train a denoising autoencoder. Which is an autoencoder that adds random noise to the input before running it through the autoencoder, and then compares the reconstruction to the original uncorrupted datapoint. This way the system is discouraged form simply learning an identity function, whilst also forcing f and g to implicitly learn the structure of $p_{data}(x)$. A popular method to do this is by estimating something called the score, where the score is a particular gradient field $\nabla_x log\, p(x)$. Further, we try to get the model to estimate the same score at every training point as the underlying data distribution, by using the fact learning the gradient field of $logp_{data}$ is a way to learn the structure of p_{data} itself. This works because of the special nature of the cost function of the denoising autoencoder, which lets the autoencoder learn a vector field of gradients. Furthermore, we also modify the criterion to the following function.

$$||g(f(\widetilde{x})) - x||^2$$

Where $\widetilde{x}$ is the corrupted version of the input x.

Moving on, another popular way of regularizing autoencoders is the contractive autoencoder. This is an explicit regularizer that attempts to make the derivatives of the encoder f as small as possible.

$$\Omega(h) = \lambda \left|\left| \frac{\delta f(x)}{\delta x} \right|\right|_F^2$$

$\Omega(h)$ is then added to the cost function. It penalizes large derivatives of $f(x)$ with respect to its variable x, and is controlled by a regularization parameter λ. The contractive autoencoder is thus trained to achieve smooth behaviour, thereby resisting small perturbations of its input. Smoothness implies that small changes in the input value won't affect the output as much. This also makes us more confident in applying the smoothness assumption to our model and implies that a small neighbourhood of input points are encouraged to map to an even smaller neighborhood of output points.

3.3 Chapter 15 - Representation learning

"Representation learning is particularly interesting because it provides one way to perform unsupervised and semi-supervised learning. We often have very large amounts of unlabeled data and relatively little labeled training

Figure 5: By adding some random noise onto the input with a corruption function $C(\widetilde{x}|x)$ and training it to find its way back to the original, uncorrupted input, we can get a vector field that converges toward the gradient field of the true underlying distribution $logp_{data}$. The line is some low-dimensional representation of the observed datapoints, called a manifold. Autoencoders assume that all datapoints belong to one or more manifolds that could be used to represent all possible correct datapoints.

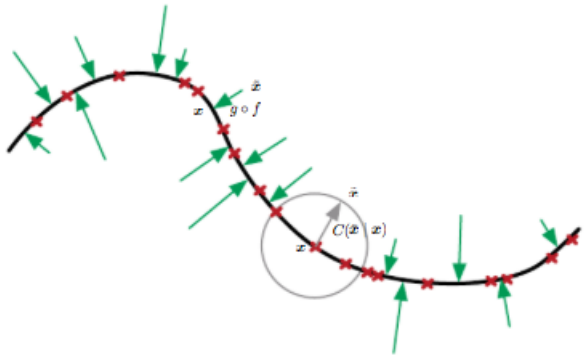
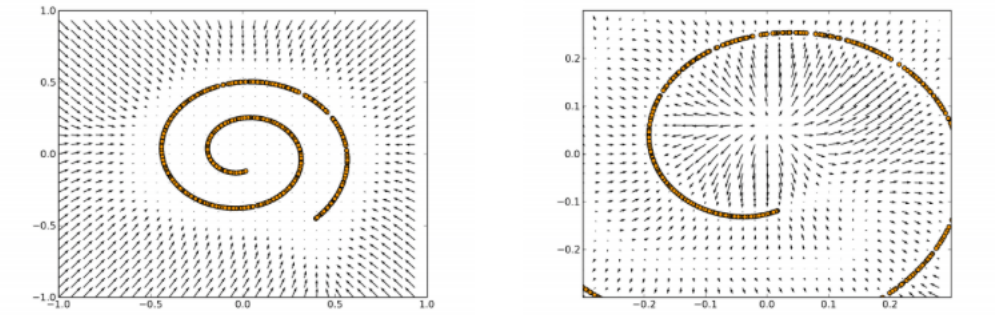


Figure 6: Having trained the denoising autoencoder for enough datapoints, the result will eventually become a more complete vector field, such as the ones below. The length of the arrows are proportional to the norm of the reconstruction error in those points. Points where the norm of the arrow is zero represent local minima, maxima and saddle points on the approximation of the underlying data distribution. Having a datapoint moving in the direction of the arrow hence increases its probability or being a correct datapoint. Thus, the purpose of denoising autoencoders is not only to learn how to denoise the input but also to find a good internal representation.



data. Training with supervised learning techniques on the labeled subset often result in severe overfitting. Semi-supervised learning offers the chance to resolve this overfitting problem by also learning from the unlabeled data. Specifically, we can learn good representations for the unlabeled data, and then use these representations to solve the supervised learning task."

In this chapter we discuss what learning a representation means and how the notion of representation can help us design deep architectures. In simple terms, we define a good representation to be one that makes some subsequent learning task easier. Learning this representation is usually done by a process called pre-training. Unsupervised pre-training, e.g. learning the data distribution of the input data, can on many tasks improve the result significantly, but on many other tasks it doesn't offer any improvements and can sometimes even hurt the system. The main goal of unsupervised pre-training is usually to use the information about the input distribution in order to choose good initial parameters and weights for some model. Having good initial weights is believed to be crucial in order for the model to find good minima in the cost function landscape. The book states that we can expect unsupervised learning to be most effective when the amount of labeled datapoints is small, and that it could work well when the underlying function to be learned is very complicated.

Unsupervised pre-training has however gone out of fashion, since new methods have largely replaced most aspects of it.

Further, we discuss the semi-supervised disentangling of causal factors. Another perspective of what makes a good representation is one where the features of the representations correspond to underlying independent causes of the real data, this way the representation separates the causes from each other. The authors reason that if we learn a representation h that describes many of the underlying causes of the input examples x, then we can use h to predict our output y. The official term for these underlying causes is "latent variables". Regularized autoencoders implicitly try to learn these latent variables to describe the input.

Figure 7: The main task of a contracting autoencoder is to learn a good manifold of the data. The following image shows an example on the MNIST dataset, where the input datapoints have been arranged in accordance to their respective order on the lower-dimensional manifold. Walking along the manifold, we find different input points which were then used to output the following image. It is worth to note the continuous nature of the manifold and its insensitivity to small perturbations. Such a manifold could create an embedding in which we define a domain in which all possible handwritten digits "live". This can in turn be used to detect which images does not contain a handwritten image. The usage of manifolds however, assumes that the data can be described with a continuous curve.

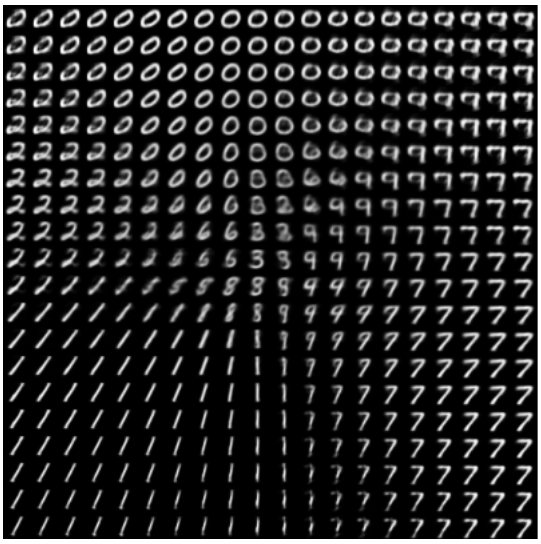
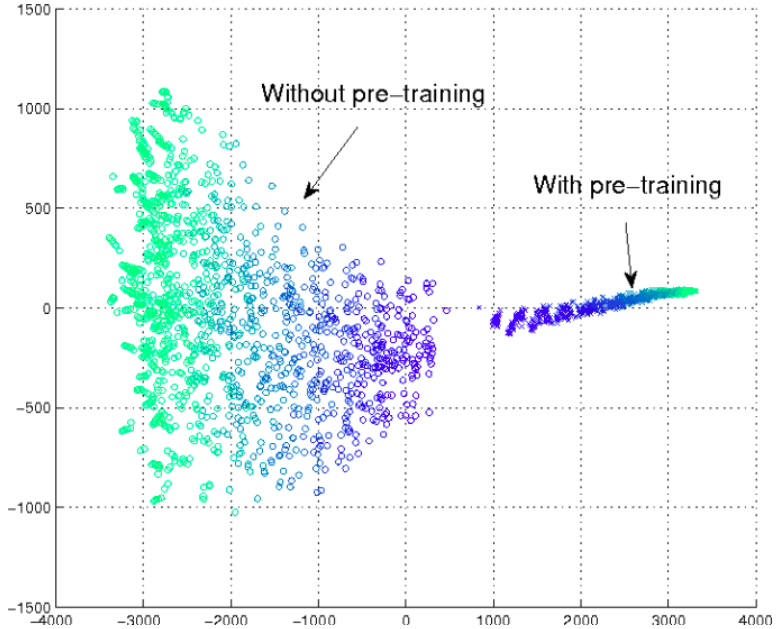


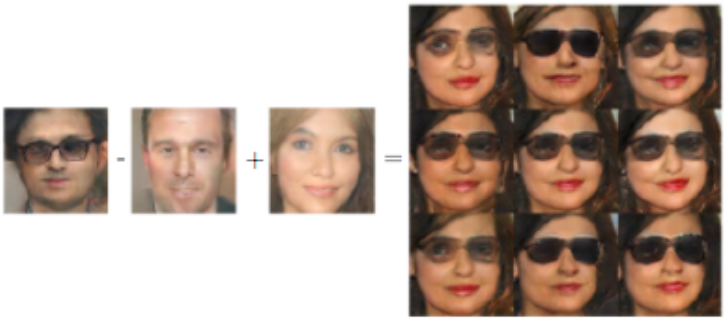
Figure 8: Neural networks that have been pre-trained, consistently produce hypothesis functions that lie close to each other, whilst neural networks with no pre-training tend to produce functions that are more spread out. This is believed to be due to that the pre-training process tend to choose initial weights that are close and within some certain region, resulting in the model ending up finding local minima in regions that also are close to each other. In contrast to when not using unsupervised pre-training where initial weights are mostly chosen by random, leading to the model ending up in a larger variance of hypothesis functions as the cost function ends up in local minima spread across many different regions.



We could take this one step further and try to achieve distributed representation, which is a representation where all the latent variables that one found are completely independent from each other. These are powerful because they can take n features and k values to produce k^n different concepts. These representations also have the advantage of being able to represent very complicated structures with only a relatively small number of parameters.

Fewer parameters means that we have fewer parameters to fit, and therefore require a smaller amount of data to train the model to generalize well.

Figure 9: Here is an example of a face generator that has learned the independent latent variables in representing a face, we can for example see that it has learned that the feature of someone wearing sunglasses is a complete separate feature from the gender of the person. By building vectors using using n latent variables and k possible values for each latent variable, we can represent k^n different concepts or faces. It is also worth to note that every element in the vectors don't relate to other elements in any way. Resulting in the vectors being able to be added and subtracted by each other in order to add/remove features, e.g. whether or not a person is wearing sunglasses or not.



There are many different ways and assumptions that encourage a learning algorithm to learn the underlying causal factors of some data, these include:

1. Smoothness, the assumption that the underlying hypothesis function f is smooth, i.e. $f(x + \epsilon d) \approx f(x)$ for a unit vector d and a small ϵ.

2. Linearity, the assumption that the relationship between some of the variables is linear. This assumptions can help with extrapolating results.

3. Multiple explanatory factors, the assumption that data is generated by multiple underlying explanatory factors. The task would easily be solved, knowing the state of these factors.

4. Causal factors, the assumption that there is a representation h that describe the causes of the observed data x, ,and not the other way around.

5. Depth, the assumption that high-level, abstract concepts can be defined with the help of simpler concepts, in the form of a hierarchy.

6. Shared factors across tasks, the assumption that each different class y_i is associated with a different subset from a common pool of relevant factors h. i.e. training how to recognize cats can help the model to recognize dogs. (eyes, fur, legs etc.)

7. Manifolds, the assumption that the input data has a probability mass that concentrates around a low-dimensional curve, where all the points are locally connected and take up a tiny volume. Autoencoders heavily rely on this assumption, as it tries to explicitly learn the structure of the manifold.

8. Natural clustering, the assumption that each manifold in the input space belong to a separate class. This assumption is used by many models.

9. Temporal and spatial coherence, the assumption that the most important explanatory factors change slowly over time. It is easier to make predictions on underlying factors rather than the raw input, e.g. pixels on an image.

10. Sparsity, the assumption that most model features are irrelevant for making prediction on some given datapoint, therefore it is best to interpret any feature that can be either active or not active, to be not active.

11. Simplicity of factor dependencies, the assumption that in good representations, the dependencies between factors are simple, or even better would be non-existent.

3.4 Generalized Denoising Auto-Encoders as Generative Models By: Yoshua Bengio, Li Yao, Guillaume Alain Pascal Vincent

3.4.1 Summary

This paper mainly discusses why denoising autoencoders are working, since the only difference between regular autoencoders is that it adds some extra noise on its input values. "The gain we expect from using the denoising framework is that if $X_{corrupt}$ is a local pertubation of X, then the true $P(X|X_{corrupt})$ can be well approximated by a much simpler distribution than $P(X)$." This is a very strong feature of the denoising autoencoder. This study dives deeper in the fact that recent work has shown that denoising and contractive autoencoders implicitly capture the structure of the underlying data-generating density.

The report looks at three problems. Firstly, how to connect the training procedure of regularized autoencoders to the implicit estimation of the underlying data-generating distribution when the data are discrete. The second and third problem that are addressed are, how to use different forms of corruption process and to measure reconstruction error. Further the paper also uses the Jacobian matrix of the encoding function to provide an estimator of a local Gaussian approximation of the density, as well as a theorem that tries to describe when and how an estimator converges to some underlying distribution.

Theorem 3.1 *If $P_{\theta_n}(X|X_{corrupt})$ is a consistent estimator of the true conditional distribution $P(X|X_{corrupt})$ and T_n defines an ergodic Markov Chain, then as the number of examples $n \to \infty$, the asymptotic distribution $\pi_n(X)$ of the generated samples converges to the data-generating distribution $P(X)$.*

Corollary 3.1.1 ***If*** *$P_\theta(X|X_{corrupt})$ is a consistent estimator of the true conditional distribution $P(X|X_{corrupt})$,* ***and*** *both the data-generating distribution and denoising model are contained in and non-zero in a finite-volume region V (i.e., $\forall X_{corrupt}, \forall X \notin V, P(X) = 0, P_\theta(X|X_corrupt) = 0$) **and** $\forall X_{corrupt}, \forall X \in V, P(X) > 0, P_\theta(X|X_corrupt) > 0, C(X|X_{corrupt}) > 0$ **and** these statements holds true in the limit of $n \to \infty$,* ***then*** *the asymptotic distribution $\pi_n(X)$ of the generated samples converges to the data-generating distribution $P(X)$.*

The conditions that the theory and following corollary presented makes everything more manageable, as it, for instance, makes it impossible for the Markov chain to wander toward infinity since we assume a finite volume in which model and data is limited. With this we also guarantee to never have a case where there are no corruption.

The energy function that we want to optimize in such cases needs to be estimated to $energy(X) \approx -logP(X|X_{corrupt}) + logC(X_{corrupt}|X)$. This however, assumes that $P(X|X_{corrupt})$ is well estimated for all $(X, X_{corrupt})$ pairs. However, the questions of how much we can trust this energy function estimator and how $X_{corrupt}$ should be chosen, still remains. The paper theorizes that the estimator should work well in all cases in theory, but states that, in practice, it might be a bad estimation when X and $X_{corrupt}$ are far from each other. The solution could be to use a different energy function in different regions of the input space. It is also possible to use the original function in a way that gives us a way to compare the probabilities of points that are nearby. It could also be used to obtain an estimator of the relative energy between two far away points.

The paper continues by discussing the problem and solution of spurious modes, the cause for this problem is described in the report as: "[...] if the corruption is too local, the DAE's behaviour far from the training examples can create spurious modes in the regions insufficiently visited during training." The solution is a method called walkback, where we train the denoising autoencoder Markov chain to walk back toward the training examples. Which leads us to the first proposition.

Proposition 3.1 *Let $P(X)$ be the implicitly defined asymptotic distribution of the Markov chain alternating sampling from $P(X|X_{corrupt})$ and $C(X_{corrupt}|X)$, where C is the original local corruption process. Under the assumptions of corollary 1, minimizing the training criterion in walkback training algorithm for generalized DAEs (combining Algorithms 1 and 2) produces a $P(X)$ that is consistent estimator of the data generating distribution $P(X)$.*

This means that the walkback training algorithm will estimate the same distribution as the regular denoising autoencoder. However, it may do it more efficiently.

In the end of the report, a small experimental validation is conducted and its results presented in the following graph.

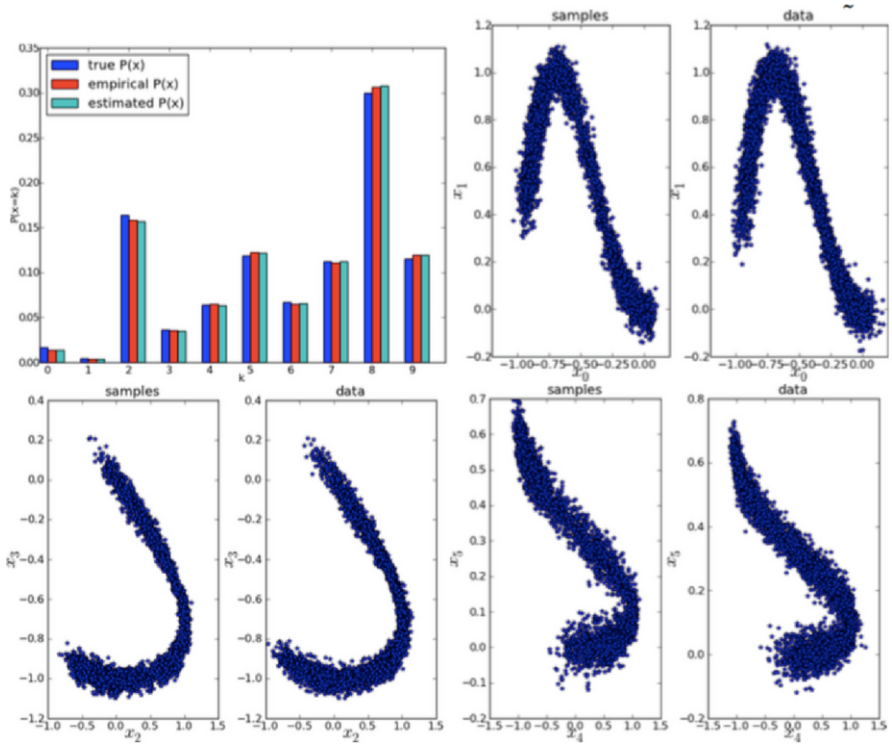
The conclusions of this report is that it has been proven that training a denoising autoencoder is an implicit way of estimating the underlying data-generating process. The paper has also been proven that a Markov chain that samples from both the denoising model and the corruption process converge to that estimation. Although, the walkback has not been proven, it was shown that it might be a working algorithm in order to converge to the same distribution faster. Which opens up the question, whether or not a model's $P(X|X_{corrupt})$ need to be able to represent multi-modal distributions over X given $X_{corrupt}$, for future research.

3.5 What Regularized Auto-Encoders Learn from the Data-Generating Distribution By G.Alain and Y.Bengio

3.5.1 Summary

Regularization forces the auto-encoder to become less sensitive to the input, but minimizing reconstruction error forces it to remain sensitive to variations along the manifold of high density.

Figure 10: The result from the experimental validation testing how well denoising autoencoders using the walkback algorithm can estimate the underlying distribution. The top left histogram describes how well this algorithm is doing on discrete data with only 10 different values. The other figures shows how how well model-generated samples imitate the training samples. Every such figure consist of 5000 generated examples trained on only 500 training examples.



An auto-encoder maps inputs x to an internal representation f through the encoder function f, and then maps back f to the input space through a decoding function g. The composition of f and g is called the reconstruction function r, with $r(x) = g(f(x))$, and a reconstruction loss function penalizes the error made, with $r(x)$ viewed as a prediction of x. When the auto-encoder is regularized, e.g., via a sparsity regularizer, a contractive regularizer, or a denoising form of regularization, the regularizer basically attempts to make r as simple as possible, i.e., as constant as possible, as unresponsive to x as possible.

The contractive auto-encoder, or CAE, is a particular form of regularized auto-encoder which is trained to minimize the following regularized reconstruction error:

$$L_{CAE} = E\left[l(x, r(x)) + \lambda \left|\left| \frac{\delta f(x)}{\delta x} \right|\right|_F^2\right]$$

Where

$$r(x) = g(f(x))$$

and

$$||A||_F^2$$

is the sum of the squares of the elements of A. A denoising auto-encoder, or DAE, is trained to minimize the following denoising criterion:

$$L_{CAE} = E\big[l(x, r(N(x))\big]$$

where N is a stochastic corruption of x and the expectation is over the training distribution and the corruption noise source.

In particular, the commonly used sparse auto-encoders seem to fit the qualitative pattern where a scorelike vector field arises out of the opposing forces of minimizing reconstruction error and regularizing the auto-encoder. What previous work on denoising and contractive auto-encoders suggest is that regularized auto-encoders can capture the local structure of the density through the value of the encoding function and its derivative.

3.6 Improving Data Quality through Deep Learning and Statistical Models By Wei Dai, Kenji Yoshigoe, William Parsley

3.6.1 Summary

A possible approach to detect outliers and anomalies in a dataset is to create a y-vector through statistical analysis. Probabilistic and statistical models can be used for outlier detection. The authors use statistical analysis to define outlier by assuming that all data is distributed with some normal distribution.

Through integrating deep learning and statistical quality control the authors proves the possible usefulness of supervised learning algorithms such as a regular back-propagating neural network applied on a previously unsupervised data set.

The paper breaks down a possible work process from the point of getting the data to producing a result. Data preparation focuses on identifying basic problems and cleaning the data; Machine learning contains neutral network and statistical quality control model for discovering complex outlier data; Data visualization then displays risk data, error data, and correct data. At the data preparation stage, the data profiling program reads the data file; particularly for the purpose of transforming string type and date type data values into numeric values; data quality problems should be discovered at this stage as well, such as incorrect sex code or date.

The authors statistical definition defines the Lower and Upper Control Limit is defined as $UCL = \mu + 3\sigma$ and $LCL = \mu - 3\sigma$. A result outside this limit is flagged as a possible outlier (risk data) and should therefore be examined. σ is the standard deviation and μ is expected value. This paper relies on the important assumption that a particual field of data (the salary) can be the product of the remaining information on each data point.

3.7 Autoencoders, Minimum Description Length and Helmholtz Free Energy by Geoffrey E. Hinton and Richard S. Zemel

3.7.1 Summary

Vector Quantization (VQ), also called clustering or competitive learning is powerful because it uses a very non-linear mapping from the input vector to the code but weak because the code is a purely local representation. PCA on the other hand is weak because it is linear and but powerful becaue it is distributed.

The expected number of random bits required to pick a code stochastically is the entropy of the probability distribution over codes:

$$H = -\sum_i p_i log(p_i)$$

So, allowing for the fact that these random bits have been successfully communicated, the true expected combined cost is

$$F = \sum_i p_i E^i - H$$

F has exactly the form of Helmholtz free energy.

The probability distrubition that minimizes F is

$$p_i = \frac{e_i^E}{\sum_j e_i^E}$$

The number of possible distributed codes is m d where d is the number of VQs (clusters) and m is the number of units within a VQ. The weights from the hidden units to the output units determine what output is produced by each possible distributed code.

A natural approach to unsupervised learning is to use a generative model that defines a probability distribution over observable vectors.

The obvious maximum likelihood learning procedure is then to adjust the parameters of the model so as to maximize the sum of the log probabilities of a set of observed vectors.

For non-linear models that use distributed codes it is usually intractable to compute these derivatives since they require that we integrate over all of the exponentially many codes that could have been used to generate each particular observed vector.

The optimal way to code an observed vector is to use the correct posterior probability distribution over codes given the current model parameters.

3.8 Why does Unsupervised Pre-training Help Deep Learning? by D.Erhan, Y.Bengio, A.Courville, P.Manzagol, P.Vincent and S.Bengio

Why does unsupervised pre-training help deep learning? This report discusses the subject of pre-training, how it works and why it can lead to models that generalize better. The influence of pre-training is motivated by the empirical results of experiments conducted in the report. The report begins with discussing a long-lived problem in the realm of deep learning, which is that the objective function that the algorithm tries to optimize oftentimes

have a very non-convex behaviour, frequently resulting in multiple local minima in the model parameter space. Therefore, there is little certainty whether or not the optimization process managed to pick good parameters. The earlier in the training process the parameters are altered, the more influence it will have on where the optimization algorithm will end up. Initial parameters and weights, limit the model parameters in particular regions of parameter space, the report called these restricted regions "basins of attraction". The amount of training data does not help, but unsupervised pre-training might. The report present one way to combat this problem, by pre-training each layer with an unsupervised algorithm whose purpose is just to learn the main variations of the input. The report is assuming that knowing $p(x)$ could help the model with the main prediction task $p(y|x)$. It all breaks down in using the information of $p(x)$ to pick a more qualified point in parameter space, rather than just picking a point at random. The supervised process and objective function stays exactly the same, regardless if we are using unsupervised pre-training or not.

Moving on the report is setting up for a series of experiments to demonstrate what influence pre-training could have on the generalization error. For the experiments, the models used for the unsupervised pre-training is the beep belief network and the denoising autoencoder, the pre-training is greedily applied to all layers of some multi-level perceptron network. Since autoencoders have proven to be very efficient at catching the statistical structures on unlabelled data, it is a very intuitive choice in order to learn $p(x)$, as long as it does not learn the identity mapping. After the greedy layer wise pre-training we proceed with fine-tuning the model. In this report it is done by adding a output logistic regression layer that uses softmax units to estimate $p(class|x)$. The datasets the experiments test on are all based on visual data, they are: MNIST, InfiniteMNIST and Shapeset. Here below we will list the different experiments and their general conclusions.

1. **Experiment 1:** Does pre-training provide a better conditioning process for supervised learning?

   Conclusion: The experiment found no significant differences in improved conditioning. The differences found were neglected by the report.

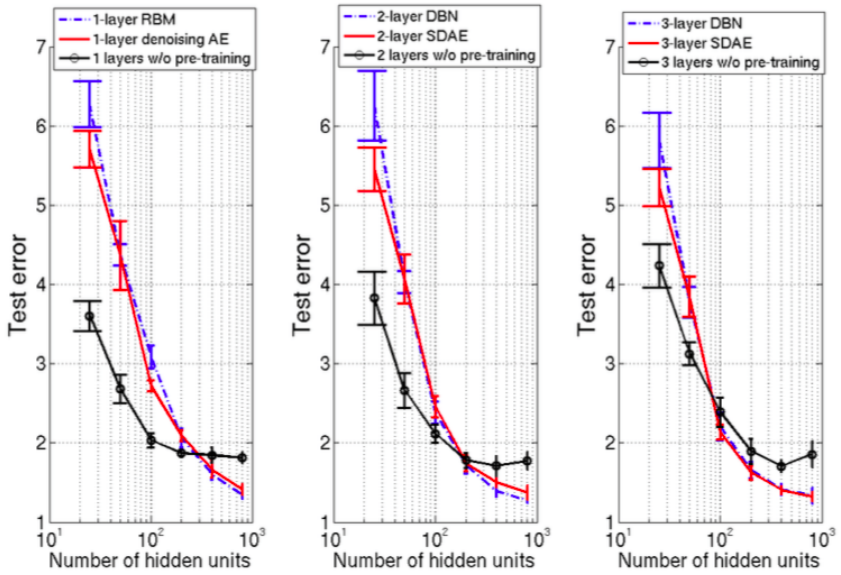
2. **Experiment 2:** The effect of pre-training on training error.

   Conclusion: Test error generally got better. However, the training error was left relatively unchanged.

3. **Experiment 3:** The influence of the layer size.

   Conclusion: Both the layer size and the amount of hidden units was related to the positive influence that the pre-training had on the generalization error. It worked well for networks of increasing size, but was harmful for networks that were too small.

Figure 11: Unsupervised pre-training gain better performance as the size of the network increases. The figure also reveals that unsupervised pre-training in fact can be harmful to the efficiency of the model when applied in the wrong situations.



4. **Experiment 4:** Challenging the optimization hypothesis.

   Conclusion: There are a lot of empirical data that favours the regularization theorem, i.e. that unsupervised pre-training rather can be regarded as a means of regularization rather than optimization.

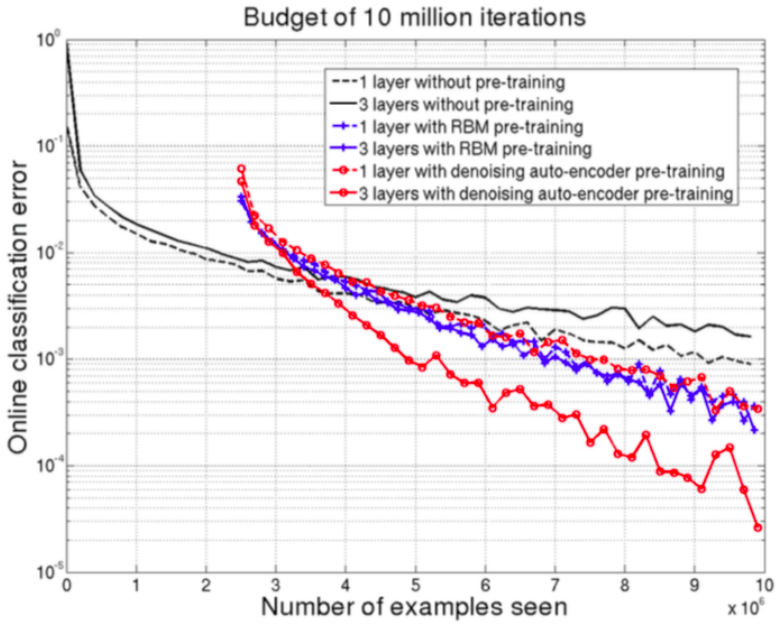
5. **Experiment 5:** Comparing pre-training to L_1 and L_2 regularization.

   Conclusion: Pre-training generally outperforms L_1 and L_2 regularization. Furthermore, the efficiency of unsupervised pre-training does not converge to zero as the training dataset grows larger, whilst the efficiency of L_1 and L_2 do.

6. **Experiment 6:** Effect of pre-training with very large data sets. (In an online learning environment)

   Conclusion: Pre-training thrives as the amount of data grows, in the experiment the denoising autoencoders performed best for a large amount of data.

Figure 12: In this online setting we can see that the amount of available training data have an impact on the performance of the unsupervised pre-training process. However, given too little data, unsupervised pre-training can actually be harmful.



7. **Experiment 7:** The Effect of Example Ordering (In an online learning environment)

   Conclusion: Training examples from the beginning and the end of the training process have the most influence on where the parameters are settling. Although pre-training doesn't stop this, it limits the variance of the final parameters due to the ordering of the training examples.

8. **Experiment 8:** Pre-training only k-layers (In an online learning environment)

   Conclusion: It seems like pre-training as many layers as possible achieves the lowest generalization error.

The conclusions for this report is that unsupervised pre-training adds robustness to a deep architecture. The optimization algorithm of deeper architecture have a higher chance of getting stuck in a bad local minima. Through the experiments it was shown that pre-trained networks consistently achieved better generalization, because unsupervised pre-training offered a way to get good initial weights on the network. It was also concluded that pre-training can be explained as a regularizer rather than an optimizer, with that in mind we could compare pre-training with other regularizers such as the L_1 and the L_2 where we concluded that pre-training outperforms them both. This also led us to the conclusion that pre-training, in contrast to other regularizer don't have a decaying performance as the training set grows.

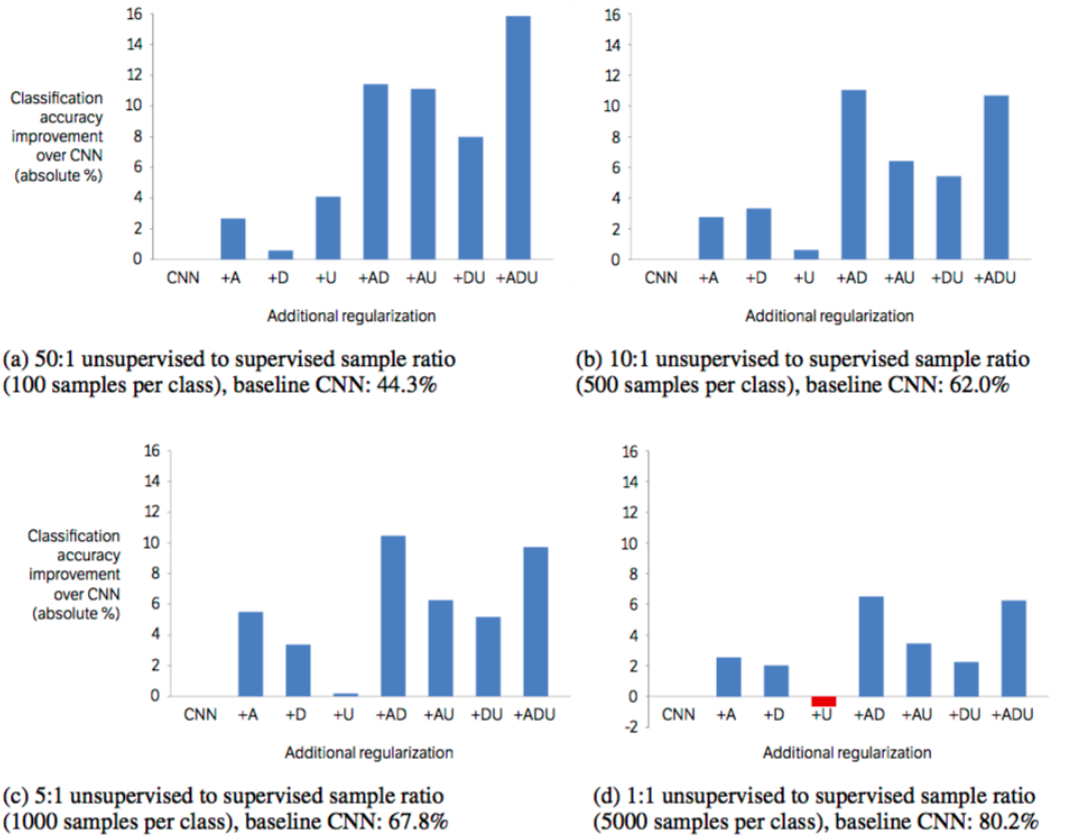
3.9 An analysis of unsupervised pre-training in light of recent advances by T.Paine, P.Khorrami, W.Han, T.Huang

This paper was published five years after the publication of the previous paper (Why does unsupervised pre-training help deep learning?) and examines how well unsupervised pre-training is doing in comparison and in combination with other newly discovered methods that in some way compete with the unsupervised pre-training. Those methods are data augmentation and dropout. The main question discussed in this paper is thus, "Is unsupervised pre-training still useful given recent advances?".

Although, unsupervised pre-training has "fallen out of favor" at the time of the writing of this paper, it states that unsupervised pre-training might still be useful. Especially in the case when the ratio between unlabeled examples and labeled examples in the training set is large. However, when the opposite is true we can instead find that unsupervised pre-training can hurt performance.

The experiments conducted in this paper is done using a convolutional neural network to classify different images on the CIFAR-10 dataset, with varying ratios between unlabeled and labeled examples (50:1, 10:1, 5:1, 1:1). The results confirm the hypothesis of the paper, that higher ratios between unlabeled and labeled examples favor the usage of unsupervised pre-training, and that the combination of several regularization methods is viable.

Figure 13: This are the results of the experiments where different combinations of regularizing methods are used, and with varying ratios. One conclusion that can be drawn is that unsupervised pre-training is more efficient if the unsupervised dataset is much bigger than the supervised dataset. Also note that for higher ratios, including unsupervised pre-training actually harm performance. A: Data augmentation, D: Dropout, U: Unsupervised pre-training.



3.10 Dropout: A simple way to prevent neural networks from overfitting by N.Srivastava, G.Hinton, A.Krizhevsky, I.Sutskever, R.Salakhutdinov

This paper offers a breakthrough in the realm of deep learning. It presents the new method of dropout which more or less solves the problem of co-adaption. Co-adaption is the problem when the hidden units are adapted to each other, each relying on that other units will fix its own mistakes. This can cause the model to perform well on training data but generalize badly. The authors managed to achieve several state-of-the-art results on many datasets using this new method. "In a standard neural network, the derivative received by each parameter tells it how it should change so the final loss function is reduced, given what all other units are doing. Therefore, units may change in a way that they fix up the mistakes of the other units. [...] dropout prevents co-adaption by making the presence of other hidden units unreliable. Therefore, a hidden unit cannot rely on other specific units to correct its mistakes."

This paper states that combining many models almost always improves performance, especially when the different methods are different, but doing so with multiple deep networks can be extremely expensive. Even though one is able to train multiple deep networks the next problem occurs when trying to infer predictions from such a ensemble of networks. This is the definition of the problem that dropout tries to solve. As the idea is to prevent

overfitting and offer a way of approximately combining exponentially many different neural networks architectures efficiently.

The algorithm itself has already been briefly explained, but the main idea is to build "new" models by randomly remove different units from the network, we usually pick $p = 0.5$ meaning that each node individually has a 50% chance of being removed, however it is recommended to let p be a tunable hyperparameter. However, the problem of such an ensemble being intractable to infer from still remains. This problem is solved by using approximating the inference by only inferring from a single model with no dropped units, whose weights are scaled-down versions of the trained weights.

Dropout can also be used on several models, including feed-forward neural nets, as well autoencoders and graphical models such as the Boltzmann Machines. It was also shown that dropout also works well when combined with other optimizing and regularizing methods, such as momentum, annealed learning rates and L_2 weight decay. The authors saw that dropout prevents overfitting in such a way that the model no longer had any use for the usage of early stopping, and that the results could be improved further by replacing the rectified linear units with maxout units. It was also found that one form of regularization was particularly useful for dropout. Which is the max-norm regularization, which forces the norm weights to be limited in a sphere, i.e. the constraint $||w||_2 \leq c$ for some maximal norm limit c. The authors hypothesize that this constraint, combined with large decaying learning rates and high momentum can offer a significant improvement over just using dropout. The authors justify this by explaining that since the parameters are constrained in a sphere of radius c we are able to use a large learning rate without having to worry about the parameters diverging outside the constraint. This allows the parameters to discover more new regions of the weight space that normally could be hard to reach. Over time the learning rate decays and the parameters settles in a well chosen minimum.

According to this paper, combining unsupervised pre-training and dropout works fine if certain precautions are made. Dropout can be used to finetune nets that have been pre-trained, although the weights that the pre-training sets for the model should be scaled up by a factor of $1/p$, so that the output of the pre-trained units stay the same during the dropout learning process. The learning rate when training the dropout networks also has to be relatively small so that they do not wipe out the weights obtained by the pre-training process.

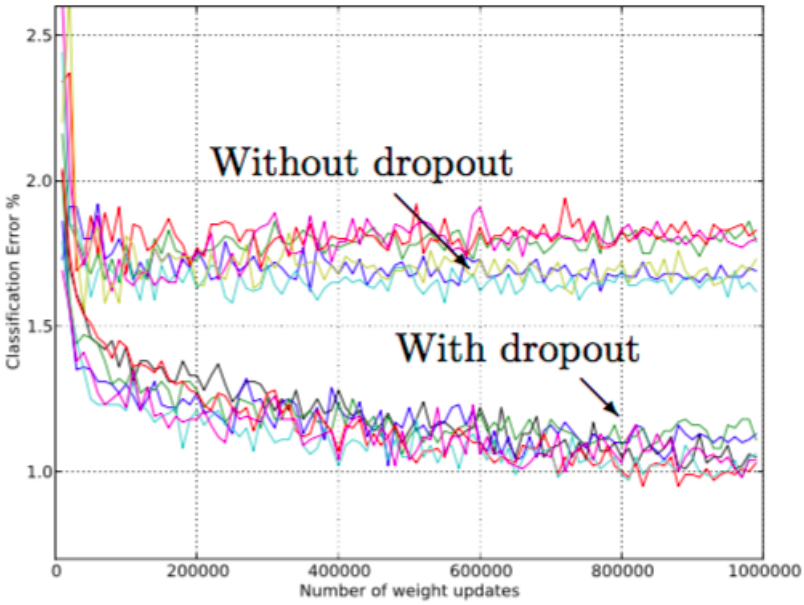
Moving on, the paper presents the results when testing dropout on a variety of different types of data, many of which it achieved state-of-the-art results. Starting out with image data it managed to achieve state-of-the-art result on the datasets: MNIST, SVHN, CIFAR-10, CIFAR-100 and ImageNet. The result on the MNIST dataset is summarized in figure 14.

Figure 14: Results on using different methods, with and without dropout, on the MNIST dataset. We can see that the state-of-the-art result is achieved with a Boltzmann machine pre-trainer combined with a dropout finetuner. We can also see how this fares compared to other popular methods.

Method	**Unit Type**	**Architecture**	**Error %**
Standard Neural Net (Simard et al., 2003)	Logistic	2 layers, 800 units	1.60
SVM Gaussian kernel	NA	NA	1.40
Dropout NN	Logistic	3 layers, 1024 units	1.35
Dropout NN	ReLU	3 layers, 1024 units	1.25
Dropout NN + max-norm constraint	ReLU	3 layers, 1024 units	1.06
Dropout NN + max-norm constraint	ReLU	3 layers, 2048 units	1.04
Dropout NN + max-norm constraint	ReLU	2 layers, 4096 units	1.01
Dropout NN + max-norm constraint	ReLU	2 layers, 8192 units	0.95
Dropout NN + max-norm constraint (Goodfellow et al., 2013)	Maxout	2 layers, (5 × 240) units	0.94
DBN + finetuning (Hinton and Salakhutdinov, 2006)	Logistic	500-500-2000	1.18
DBM + finetuning (Salakhutdinov and Hinton, 2009)	Logistic	500-500-2000	0.96
DBN + dropout finetuning	Logistic	500-500-2000	0.92
DBM + dropout finetuning	Logistic	500-500-2000	**0.79**

In the appendix the reader can find some intuitive advice on how to set the hyperparameters of the model. It begins with general advice for setting the network size where it says that if the model has n units per layer in a standard neural net, it should have at least n/p units in the entire network. For the hyperparameters learning rate and momentum, it is recommended to have the dropout net to use a learning rate that is 10-100 times larger than the learning rate that was optimal for a regular neural net. Alternatively, one could use high momentum. Although, a momentum of 0.9 is popular for standard nets, it was found that momentum values around 0.95 to 0.99 work better for models with dropout. In conclusion, high learning rate and/or momentum speeds up learning significantly. High learning rate and momentum can cause the weights to explode, to prevent this we

Figure 15: Graph describing how different architectures perform with and without dropout. We can see that different architectures have a consistently improved performance no matter what architecture we choose.



present max-norm regularization which is dictated by the variable c (the maximal norm of a given weight vector). Typical values for this variable range from 3.0 to 4.0. Lastly, some intuition of choosing the hyperparameter for the dropout rate is given. The value p typically range between 0.5 and 0.8 for hidden units, and is around 0.8 for input units. Note that $p = 1$ indicate no dropout and $p = 0$ indicate that all units are dropped. Smaller values on p therefore requires more hidden units n. Small p can slow down training and can cause underfitting, whilst large p might not drop enough units to prevent overfitting.

3.11 Operational Risks in Financial Sectors by E.Karam and F.Planchet

3.11.1 Summary

This report attempts to offer an official definition of what we call operational risks, as well as how it should be measured and what we can do to counteract those types of risks. In the report they are stating that market and credit risk long has been seen as the two biggest factors when it comes to risk for financial institutions, but this paper shows that operational risks also should be seen as a significant factor in the risk assessment.

Although, the definition of operational risks are not that clear, the report offer us some ways to look at it in order to understand. The most official definition, given by the Basel II Committee states that operational risks are defined as: "[. . .] the risk of loss resulting from inadequate or failed processes, people and systems, or from external events. For example, an operational risk could be losses cue to an IT failure; transcriptions errors; external events like a flood, an earthquake, or a fire." The loss aggregation of the risk in this report is simplified to only two parameters, frequency and severity. Loss aggregation is easiest described as the cumulative sum of harm that some certain risk can cause to the company. The report then divide the possible operational risks into four different categories of loss aggregation. Negligible risks, marginal risks, catastrophic risks and impossible risks. Negligible risks are the one with low frequency and low severity, these are mostly ignored. Marginal risks are those with high frequency and low severity, these are not very harmful individually, but accumulated, can stack up losses. Catastrophic risks are those with low frequency but high severity, rare but very harmful, this could be rogue traders or natural disasters. And lastly, impossible risks are the ones with high severity and high frequency, a company cannot operate with impossible risks and should be solved as fast as possible.

Today there are mainly three ways of calculating how much capital that is required in the case of operational risks. These are, the basic indicator approach (BIA), the standardized approach (SA) and the advanced measurement approach (AMA). Both the BIA and SA calculations relies on taking some sort of weighted average of the gross income of the past three years to calculate the capital requirements to handle operational risks. The AMA on the other hand describe a wider variety of methods, the most interesting one being the loss distribution approach (LDA), which is a parametric method where one gathers data on losses that were due to operational risks. These data are then used to estimate the distributions for the loss frequency and the loss severity. Since, we want the distribution of the loss aggregation, which, in the report is assumed to consist of the parameters loss frequency and loss severity, we combine the two distributions. We, however need to approximate this new loss aggregation distribution with a Monte Carlo method.

The report continues by digging deeper into the method of choosing good distributions to represent loss frequency and loss severity. Having fitted a distribution for the loss severity it explains methods of testing this distribution, these include, the probability plot, the Q-Q plots, the Kolmogorov-Smirnov Goodness of Fit Test and the Anderson-Darling Goodness to Fit Test. For testing the loss frequency distribution the report advice using the Chi-squared goodness to fit test. However, in order to build these two distributions we need to collect data on frequency and severity of losses for a particular operational risk type. The report further states that there currently (2012) is a lack of operational risk loss data, which is a major issue. One possible implementation of the work that we will be conducting could therefore be to detect operational faults in order to gather data on its frequency, and by doing this, facilitating the calculation of the loss aggregation.

Furthermore, the report says that in practice, small operational losses are not detected or declared, for instance, all losses under $5000, which can make the data become truncated, and create unwanted biases of the aggregated loss. The most common approach to this problem in practice is just to ignore it, which according to the report is the worst possible alternative. Here we see another possible application of the algorithm that we have set out to create, since our algorithm could help people also detecting the less severe losses we could fill in the gap in the distribution that truncates it, pushing it closer to reality.

Next we talk about how to work with extremes for catastrophic risks. There is a whole branch of statistics that are dedicated to this called the extreme value theory (EVT), this approach mainly want to find the characteristics on the tail of a certain distribution without limiting the analysis to only a single parametric family on the whole distribution. This is especially important as some of the extreme cases might never have even been observed. This is exemplified with two anecdotes of rogue traders that cost their banks $1.3 billion and $1.1 billion respectively. Therefore, it might be smart to assume that the body and the tail of the data should not be sampled from the distribution or even the same family of distributions. To this the report presents the three-type theorem, stating that when finding a good distribution for extreme values in a set of datapoints, there are essentially only three reasonable distributions: the Weibull, the Gumbel, and the Freshet distributions, all of which have long tails. "This result is important as the asymptotic distributions of the maxima always belongs to one of these three distributions, regardless of the original distribution." As a side note it is also stated that extreme value theory not only can be used in financial calculations, but also in hydrology and structural engineering to detect extreme values that can have catastrophic consequences, further offering inspiration of areas to apply the algorithm.

The report ends with discussing insurance as a tool to cover for operational risk. A good insurance would in turn mean lower capital requirements. However, calculating the pricing for such insurances might still not be an easy feat, yet another area where it could become relevant with our algorithm.

3.12 Xingnan Jiang (2018) Operational risk and its impact on North American and British banks, Applied Economics, 50:8, 920-933, DOI: 10.1080/00036846.2017.1346363

3.12.1 Summary

In addition to direct monetary damage an operational loss event may affect the reputation of the involved firm. This gives rise to reputational risk. Studies of reputational risk on the financial sector are scarce, hence the impact and danger of operational risks may be even greater than what has previously been anticipated.

This study examines the impact of operational loss events on the firms that announce them, and the potential reputational damage that follows the release of the loss announcements by analysing over 300 operational loss events recorded in the largest publicly traded banks in USA, Canada and the UK. It analyses characteristics of operational losses such as the loss region, size, disclosure status, impact type and claimant type, and how they may influence the market to react differently following an operational loss announcement.

The study reveals significant loss in the companies' reported earnings followed by increased stock volatility, and subsequent declines in analysts' estimates. The reactions tend to be greater when the restatement is due to fraud. In terms of loss severity, the study sample reports an average loss of $113 million. With respect to business line, Commercial Banking including lending and loan services reports the highest number of loss events, and the biggest average loss is found in Corporate Finance, which is also the business line that produces the highest total loss amount in the sample.

The paper concludes that although, by regulation, it is not mandatory for banks to incorporate reputational risk into the calculation of the capital holding, it may be in the banks' best interest to do so, because evidence suggests, reputational losses can potentially be costly. The results indicate that the potential damage of operational risk may have been underestimated.

4 Conclusion of the pre-study: Methodology

4.1 Methodology of examination. From what theories and concepts above will result in a preliminary description of, for instance, what algorithms that is going to be tested, or data that will be used.

As concluded from this literary review, our desired approach to this problem is mainly to use an autoencoder neural network. In this relatively unexplored field, where there are no clear state-of-the-art result or method.

Autoencoder neural networks are however a modern and advanced method to approach unsupervised anomaly detection, that has only started to show their true potential within the last few years of writing this report. Autoencoder neural networks offers a unique ability to find non-linear dependencies in large unlabelled data sets, which we predict will be of great benefit for the result we wish to accomplish.

As previously mentioned, since there are no previous work that we could rely our methodology on we need to find good strategies on our own. Strategies listed above, such as dropout, early stopping, unsupervised pre-training etc. will be tried in an iterative manner to find a good methodology for this type of task. The methodology itself will therefore mostly consist of building the model and trying different combination of strategies to examine how they affect an anomaly detecting network. Hopefully, this might offer some guidelines for future research as to what strategies works well for this type of task.

In practice, we shall build this network with the programming language Python and we intend to utilize the following tools: *Anacondas, Jupyther, Scikit-learn, Pandas and TensorFlow.*

As the data set we intend to use has not yet been disclosed to us we cannot yet be certain of what measures we will have to take in order to format our the data set and prepare it for use. The formatting is however also predicted to be done with the programming language python, with suitable libraries.