CS5098: Deeplearning for Financial Investments

Contents

List of Figures

List of Tables

1. *Preface*

I. Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main word count of this project report is XX XXX words long, this including the all the sections of report including the Appendix. I give permission for it to be made available for use in accordance with the regulations of the University Library. Permission is also given, for the report to be made available on the public school intranet, permission is also given for this work to be used in research within the University of St Andrews, and for any software to be released on an open source basis. I retain the copyright in this work, and ownership of any resulting intellectual property.

II. Ethics

This projects includes free available information from the internet, free existing datasets used for academic research. The result of this projects are not available publicly and will be kept in the department of Computer Science department at the University of St Andrews. Above all, this project does not raise any ethical considerations or risk.

III. Abstract

2. *Introduction*

3. *Context Survey*

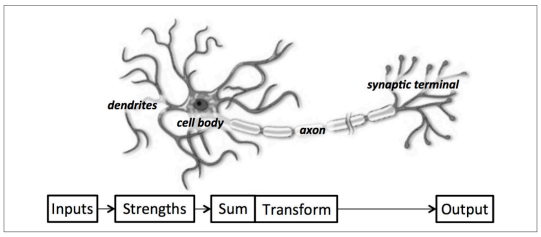
4. *Theory*

I. Neural Networks

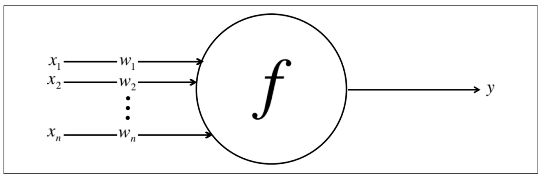
Neurons are the fundamentals units of the human brain. The brain consist of interconnected neurons though synapses, this structure is what we call a neural network. This phenomena is what enables us to experience everything around us. Biological fig. 4.1a neurons receives its inputs from dendrites, and the strength of each connection determines the contribution of the inputs to how stimulated the neuron would be to produce an output. Biological neurons are optimised to receive information from other neurons in the network. These neurons process information in their unique ways and then sends their results to other neurons in the network for further processing.

The artificial neural network (ANN) fig. 4.1b emerged from this phenomenom, where researchers goal was to create systems similar to the biological neural network to perform similar task. The first approach was pioneered by [McCulloch and Pitts, 1988] in 1943. Similar to the biological neurons the inputs of an ANN are represented as vectors $x_1, x_2, ..., x_n$ and this are connected to a neuron by a set of weights $w_1, w_2, ..., w_n$ and for each neuron the values connected to this neurons are summed up to form *logits* $z = \sum_{i=0}^{n} w_i x_i$ and this value in conjunction with the bias b is then passed through an activation function $y = f(z)$ and this gets sent to other neurons. This fundamental phenomena further extends to forming more complex networks such as convolutional and recurrent networks. This topics are discussed in depth further in this paper.

The formal way of representing an artificial neural network is $y = f(w \cdot x + b)$



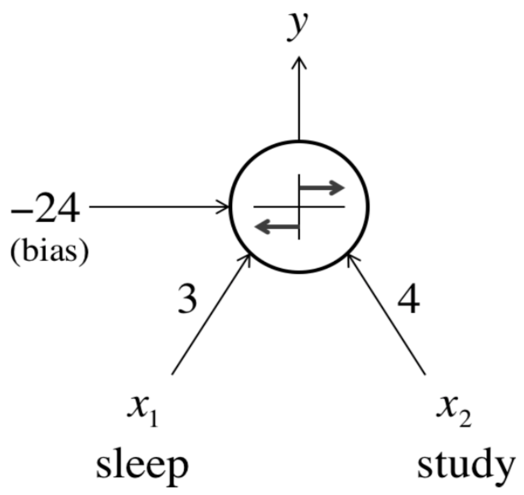
(a) Biological Neural Network



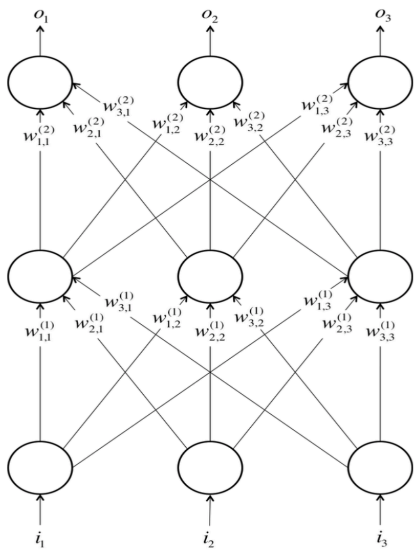
(b) Artificial Neural Network

I.I Feed forward networks & Deep Networks

The basic neural network is just a simple network with the input going straight to the output as illustrated here fig. 4.2a But in order to solve more complicated problems such as image classification, image recognition, hand written digit recognition, it is impossible for a single neural network to learn how to classify between different images. The human brain solves this problem because consists of a lot of neurons organised in layers [Mountcastle, 1957]. Information flows from the input layers I_i for example the human eyes into the internal layers called hidden units and this step propagates accross the network to the output layers denoted by O_i. Based on this concept the idea of feed forward networks fig. 4.2b were developed to learn complex representations of the input data set. Deep learning can be summarised as a feed forward network that has more then one hidden layer. This deeper layers go further to learning higher representations of the input information.



(a) Simple Neural Network



(b) Feed Forward Neural Network

When neural networks were developed they had several limitations, this was because they could only perform linear predictions this means they just understand the data that has been inputed into the network, they cannot learn complex relationships between different items. Non linearity was introduced to solve this problem. This introduced non-linearity to the network, makes the network learn higher order representations and interaction of different input values.

I.II Activation Functions

Activations functions fig. 4.3 used in a neural network introduces nonlinearity. The 3 main activation functions used are sigmoid eq. (I.1), hyperbolic tangent eq. (I.2) and ReLU eq. (I.3). The sigmoid function is the most used. The sigmoid would output values close to 0 if the values of the logits are relatively small and when the logits are relatively large. The hyperbolic tangent values ranges between -1 and 1.

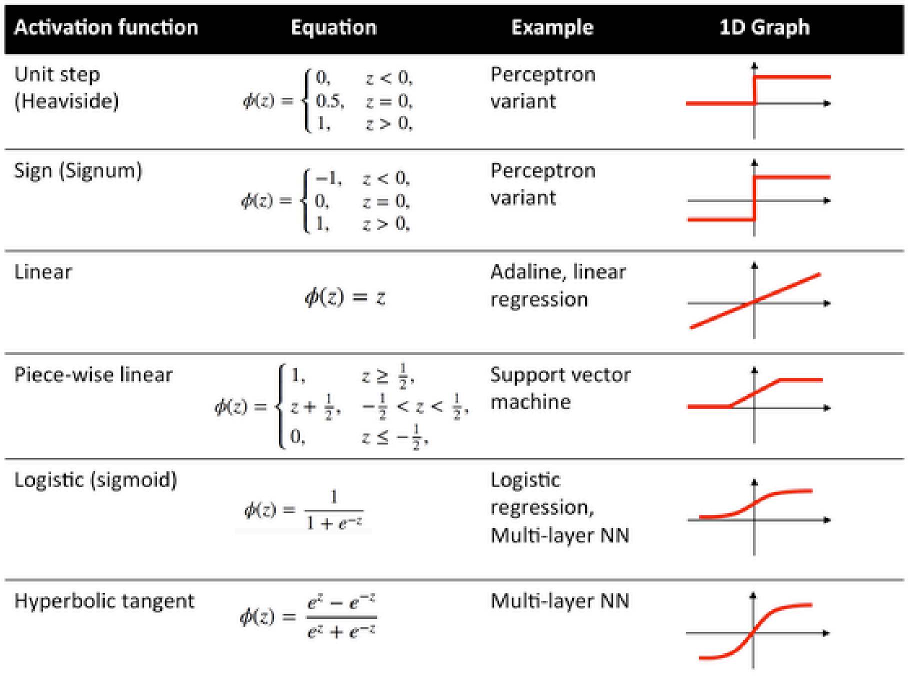
$$f(z) = \frac{1}{1+e^{-z}} \tag{I.1}$$

$$f(z) = tanh(z) \tag{I.2}$$

$$f(z) = max(0, z) \tag{I.3}$$

Another common function used in neural machine learning is called the softmax function. This is used to represent a set output vectors as a probability distribution over a set of labels. In a classification problem, this is used to represent the probability of how likely our classes are given a set of input values. The higher the probability the more likely the class.

$$y_{softmax_i} = \frac{\exp^{z_i}}{\sum_j \exp^{z_j}} \tag{I.4}$$



Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer NN	

Figure 4.3: Activation Functions

I.III Backpropagation

Neural networks learns uses a process called backpropagation. This was pioneered by David Rummelhart and Geoffery Hinton in 1986 [Rumelhart et al., 1988]. The idea of back propagation is to learn how fast the errors changes with respect to the weights of an individual connection. Back propagation learns by trying to minimise the errors provided by the outputs with respect to the weights from the hidden layers and it uses the method of steepest gradient descent. This method can be imagined as a ball rolling down the hill. Back propagation uses the derivative of the errors with respect to the weights of each neuron to modify the weights connected to the output layers all the way back to the weights connected to the input layers. Back propagation uses the chain rule derivatives of the outputs weights and the hidden neurons to modify the weights of the network. The deeper the network, the more chain derivatives multiplications has to be done.

Other method of gradient descent includes stochastic gradient descent which is commonly used, this randomises which weights to modify during training time. The errors are determined by several loss functions which includes but not limited to the least mean squared method for regression problems, cross entropy for classification problems, max marginal and other variations. In this paper the cost function explored includes the mean squared error eq. (I.5) and cross entropy eq. (I.6).

$$E = \frac{1}{2}(y - \hat{y}) \tag{I.5}$$

$$H(p,q) = -\sum_{x} p(x) \log_2 q(x) \tag{I.6}$$

We can formally define the equation for back-propagation chain rule to obtain the partial derivative of the error E with respect to weight w_{ij} for a neural network with a single hidden layer. o denotes the output of the hidden layer and net_j denotes the logits of the hidden layer and w_{ij} represent the weights.

$$\frac{\delta E}{\delta W_{ij}} = \frac{\delta E}{\delta o_j} \frac{\delta o_j}{\delta net_j} \frac{\delta net_j}{\delta w_{ij}} \tag{I.7}$$

I.III.1 Problems with back propagation

Neural networks suffers from 2 major problems during back-propagation, this includes vanishing and the exploding gradient problem. The vanishing gradient problem occurs when applying the chain rule to deeper layers, the gradient become so small they do not make any change to the lower layers making them not learn anything. This topic is further explored in section V. The exploding gradient problem is the inverse of the vanishing gradient problem, this is where the gradients gets exponentially gets large. This is commonly caused by having un-normalised values in the network.

I.IV Convolutional Neural Networks

Convolutional neural networks have made a lot of advances in computer vision from image classification [Krizhevsky et al., 2012], to image recognition [He et al., 2015] and image captioning [Donahue et al., 2014] and much more. Convolutional neural networks evolved from studying the brain's visual cortex. In 1958 David Hubel and Torsen Wisel [Hubel, 1959] performed a series of experiments on cats and they uncovered the structure of the visual cortex. Their work shows that many neurons in the visual cortex only react to visual stimuli located in a limited region of the receptive field. The research also shows that the visual cortex is made up of hierarchical layers where the lower layers learn low level representations of the image and the higher layers learn complex representations that comes from lower level representations.

The main idea behind convolution neural networks is that they are simply neural networks that uses convolution instead of general matrix multiplication in at least one of the layers [Goodfellow et al., 2016, p. 274]. Convolutional neural networks consists of a series of concepts, this includes Convolution as mentioned earlier, kernels which in image processing refers to filters E.g. Sobel edge detector, max pooling and last but not least fully connected layer. Convolutional neural can be formalised as Convolution, Linear Recification and Max pooling.

In the case of this paper we will refer to a image data (2d image) as a 2d tensor. Convolution fig. 4.5 is just a weight average of a series of data. In the case of convolutional networks it is a dot multiplication of the section of the data (2d tensor) and the kernel, the area of the average section of the tensor is donated by the kernel size. The second stage fig. 4.6a consists of the activation which the convolution is passed through. The ReLU activation function is commonly used, the output from the ReLU function is then sent to the pooling stage. Pooling functions fig. 4.6b returns a summary of nearby outputs [Goodfellow et al., 2016, p. 282]. Max pooling returns the maximum of a rectangular neighborhood and average pool returns the average. The output of this is what we know as a convolutional layer. A diagram showing this process is shown fig. 4.7

I.V Recurrent Neural Networks

Recurrent neural network (RNN) like convolutional network has made major advances in artificial intelligence, this includes problems like Named entity recognition [Lample et al., 2016], Machine translation [Amodei et al., 2015]. Recurrent in comparision to deep feed forward networks contains feedback loops. The whole processes can be imagined as a series of time, where the hidden values from the previous time step is passed on to the next, and this process persists all the way the end of the time series fig. 4.8. This concept can then be further used for solving several problems this includes mapping a singular input to a series of outputs can be used for captioning images. Mapping an input sequence to a single output can be used for document classification. Mapping a sequence of inputs to a sequence of outputs can be used for machine translation. Vanilla RNN suffers from a problem called the vanishing gradient problem. This was briefly discussed in section III, where as the time steps increases in order for the network to learn anything, it has to apply a long chain derivative multiplication. This is a problem as

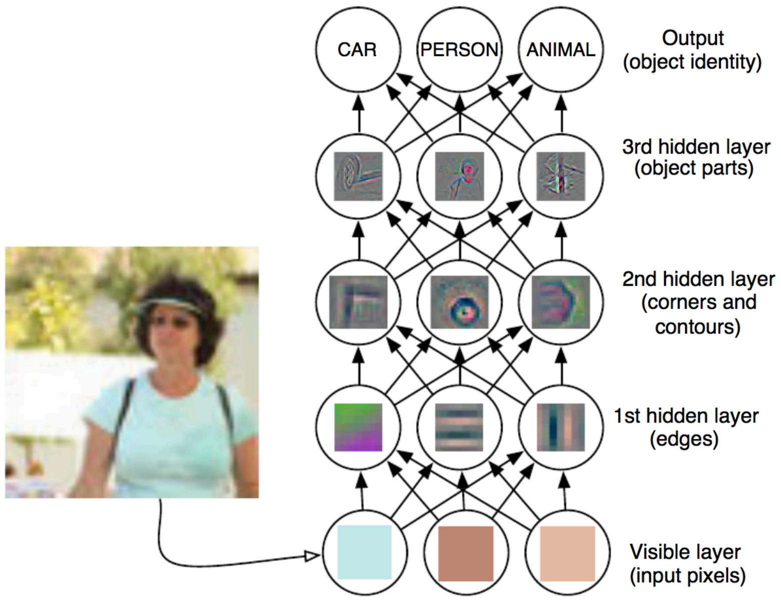


Figure 4.4: Convolutional Neural Network

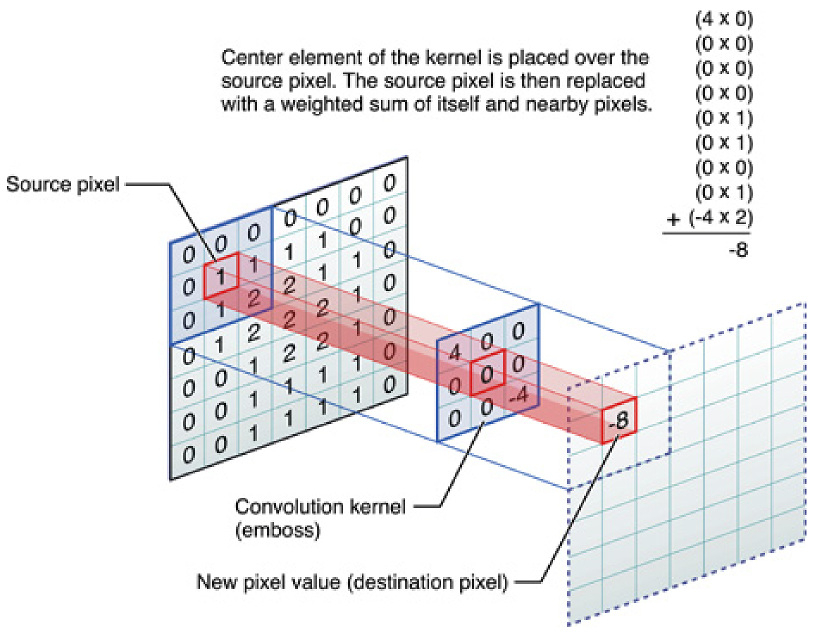
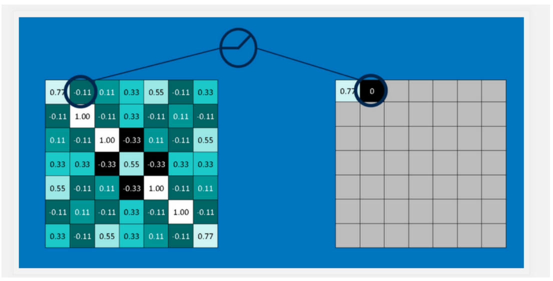

Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution kernel
(emboss)
New pixel value (destination pixel)

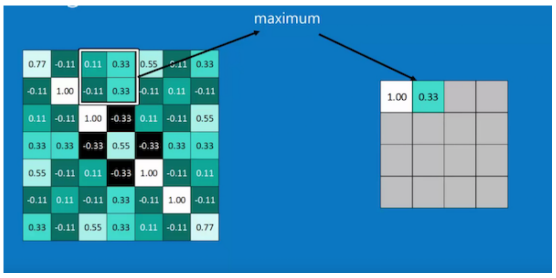
Figure 4.5: Convolution



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

0.77	0					

(a) Linear Rectifier



0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

1.00	0.33		

(b) Max pooling

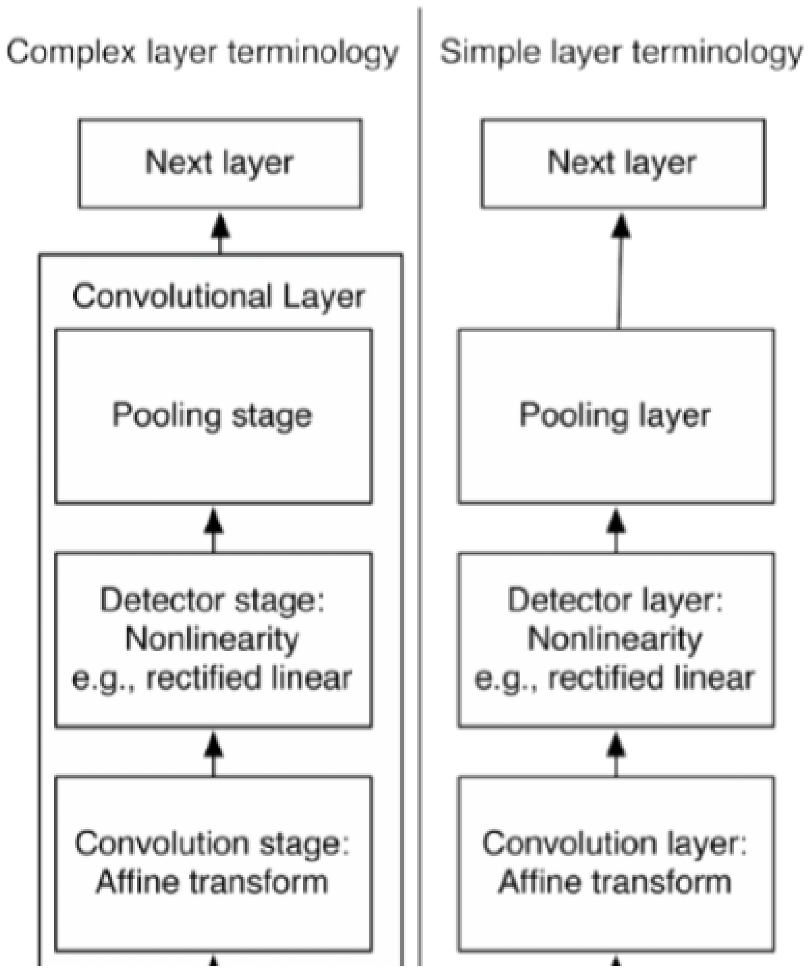

Complex layer terminology
Simple layer terminology
Next layer
Convolutional Layer
Pooling stage
Detector stage: Nonlinearity e.g., rectified linear
Convolution stage: Affine transform
Next layer
Pooling layer
Detector layer: Nonlinearity e.g., rectified linear
Convolution layer: Affine transform

Figure 4.7: Convolutional Layer

multiplying small values would yield to the value approaching 0 at a rate proportional to the number of time steps. In order to solve this problem LSTM networks and GRUs were developed. In this paper we would only explore LSTM networks, information on GRU's can be found here [Chung et al., 2014].

Vanilla RNN's can be formalised by the following equations, where, x represents the inputs, W represents the weights and h representing the hidden layer layer, t represents the time step and σ representing the activation function in this case sigmoid.

$$y_t = \sigma(h_t) \tag{I.8}$$
$$h_t = \sigma(W^{hh}h_{t-1} + W^{hx}x_t) \tag{I.9}$$
$$x_t = \{x_{i_t}, ..., x_{N_t}\} \tag{I.10}$$

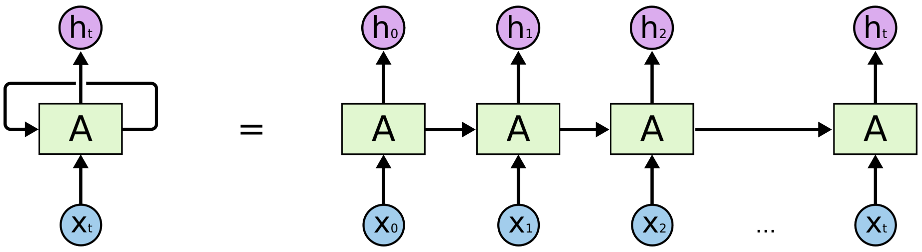


Figure 4.8: Recurrent Neural Network

More complex variations of RNN's have proven to give good results in terms of machine translation [Yao and Huang, 2016] and other sequence to sequence task. This is the process of having two different RNN network, where one does a forward pass and the other performs a backward pass and the results are concatenated and then used for prediction.

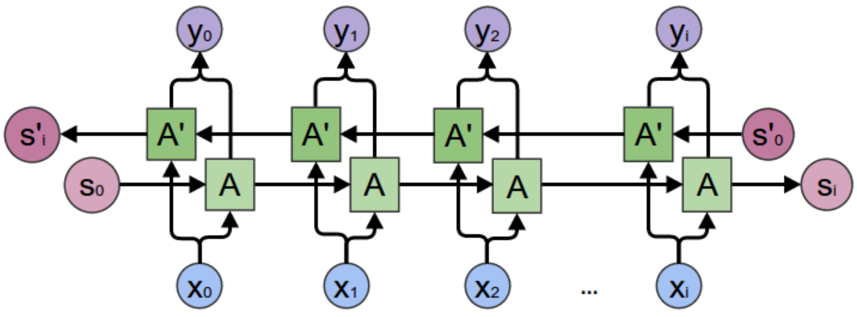


Figure 4.9: Bidirectional Neural Netowork

I.V.1 LSTM Networks

Long short term memory or LSTM [Hochreiter and Schmidhuber, 1997] was a mechanism developed to solve the vanishing gradient problem. This should not be confused with a layer of hidden units, this is not a layer of hidden units instead it is a mechanism of how each individual hidden neuron behaves. The main idea behind LSTM's is to be able to transmit important information into the future. This process is achieved by the hidden layers having a series of gates that contributes how much effect the past and current contribute towards future units. The LSTM fig. 4.10 consists of several components or gates this includes

1. Forget gate: this gate decides how much information should be forgotten.

$$f_t = \sigma(W_f \cdot [h_t - 1, x_t] + b_f)$$

2. Input gate: This value is used to determine how important the current input is.

$$i = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\widetilde{C} = tanh(W_{\widetilde{C}} \cdot [h_{t-1}, x_t] + b_{\widetilde{c}})$$

3. Memory cell: This is one of the most important component as this hold important information that has been learnt over time.

$$C_t = f_t * C_{t-1} + i_t * \widetilde{C}$$

4. Output gate: this determines how much of the cell is exposed

$$o = \sigma(W_o[h_{t_1}, x_t] + b_o)$$

$$h_t = o_t * tanh(C_t)$$

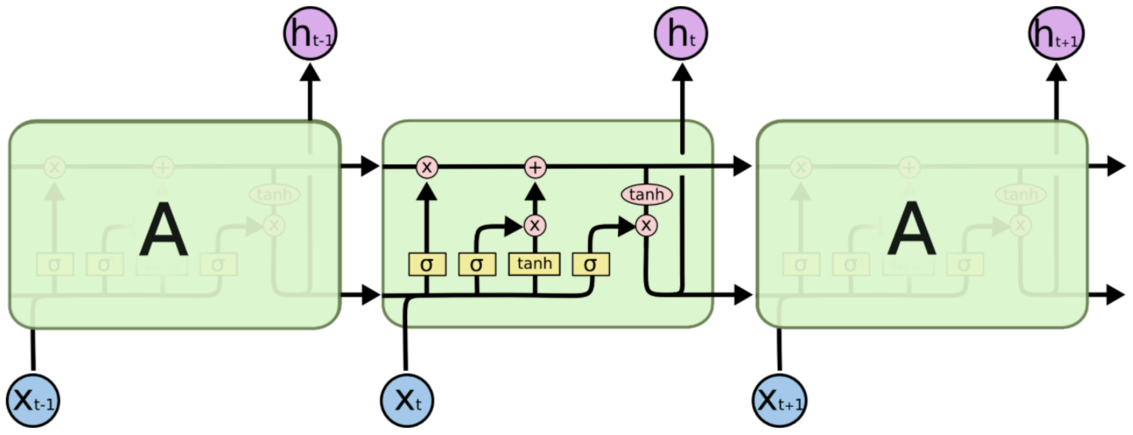


Figure 4.10: LSTM Network

I.VI Optimization

Optimization of deep neural networks has been a big field machine learning, some of this techniques includes but not limited to creating variations of gradient descent optimizers such as Adam [Kingma and Ba, 2014], Adagrad [Duchi et al., 2010] which builds from stocastic gradient descent, RMSprop which improves up on Adagrad [Hinton et al., 2012] [Goodfellow et al., 2016, p. 284] and Momentum optimizers. Other optimisation methods are discussed below

I.VI.1 Batch Normalization

Batch Normalisation [Ioffe and Szegedy, 2015] was introduced by S. Ioffe and C.Szegedy to address the vanishing and expoding gradient problem. They noticed the distribution of each layers inputs changes during training time as the values of the previous layers changes. They proposed a solution by adding an operation before the activation of each layer, which essentially scales and shifts the results using two parameters. The model will eventually learn the optimal values for these parameters during training time.

I.VI.2 Dropout

Dropout [Srivastava et al., 2014] is a simple but effective technique used for preventing overfitting in deep learning. The algorithm essentially sets using a probability p the input values to 0. This gets fed into the neural network during training time, and this is removed during test time. This is illustrated by the diagram section VI.2

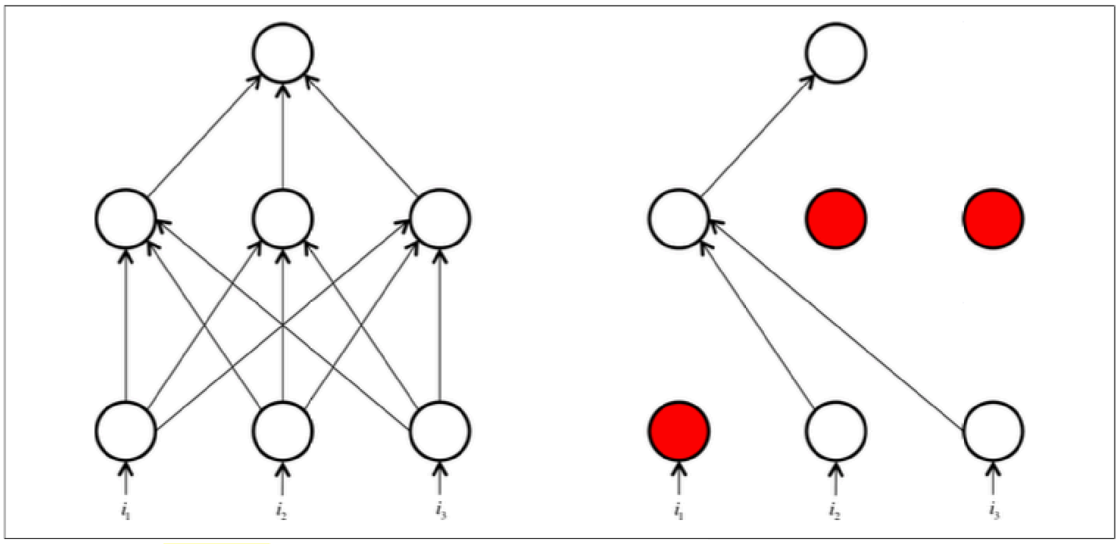


Figure 4.11: Dropout

I.VI.3 Regularisation

Another method for combatting overfitting is a method called regularisation. Regularisation modifies the cost function we try to minimise by appending additional terms that penalises large weights ϕ. Our new cost function then becomes eq. (I.11). As $f(\phi) it$ gets bigger as the values of ϕ gets larger and the regularization strength λ is another hyper parameter that can be modified. The most used form of regularisation is the $l2$ regularisation [Krogh and Hertz, 1991], this regularisation techniques specialises in penalising heavier weights vectors while preferring diffused weight vectors. Regularisation can be visualised fig. 4.12. Another common used regularisation technique is the $l1$ regularisation technique. This technique is used to increase the sparsity of the weight vectors. This is very useful in understanding what features are contributing to the networks decision. Other variations includes Max norm [Srivastava et al., 2014] which is another regularisation method that restricts the weights from getting too large.

$$loss = Error + \lambda f(\phi) \tag{I.11}$$

II. Representation Learning

This chapter aims to motivate the understanding for developing data representations and to explore a variety of techniques that are currently available. The main reason for data representation is to be able to represent and compress the high volume of data into meaningful representations that are well suited for the machines to learn. Representation learning is mostly used in cases where large amounts of unlabelled

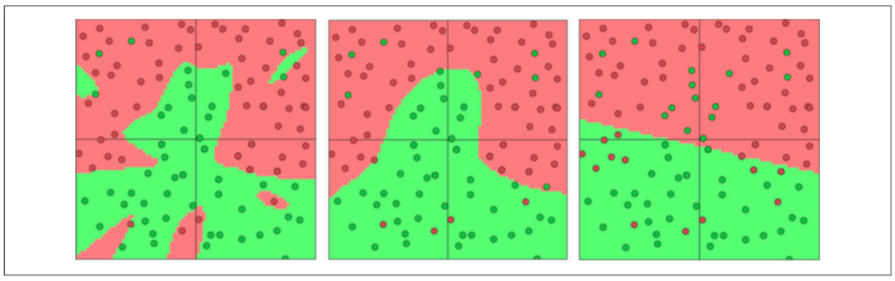


Figure 4.12: Shows applications of different regularisation strengths (0.01, 0.1 and 1) [Sutskever et al., 2014]

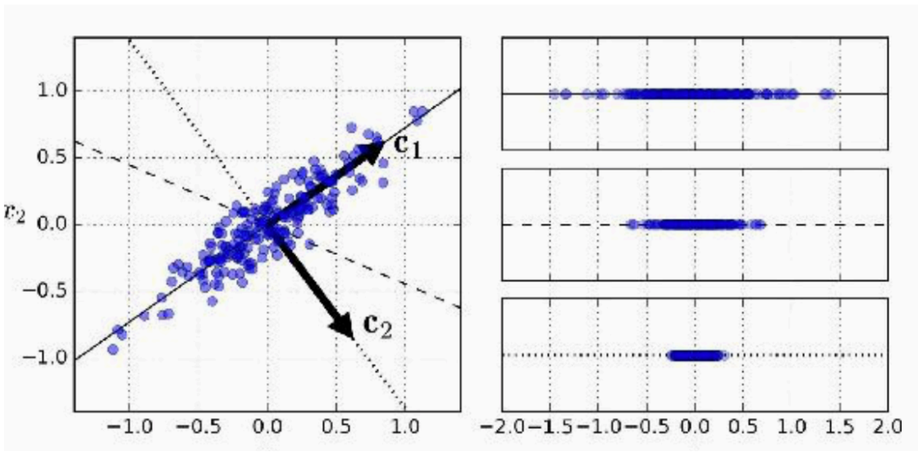


Figure 4.13: PCA: Right images shows decending variance from top to bottom

data. In terms of natural language processing, this processed is done by learning word embeddings or low dimensional word representations in terms of computer vision deconvolution [Noh et al., 2015] is a process used to learn image representations.

II.I Principal Component Analysis

Principal component analysis [Pearson, 1901] is a method for performing dimensional reduction of high vectors $n < d$ where n represents the lower dimensional vector, and d the dimension of the original data. It is a process used in finding a set of lower dimensional axes that conveys the most information about our current dataset. As shown in the image fig. 4.13 , the aim is to find an axis or dimension in the high dimensional space that preserves the maximum variance, this axis would be our first principal component. Afterwards it then proceeds to find the second axis orthogonal to the first one that accounts for the remaining variance, it then finds a third axis that is orthogonal to both axis and this process is repeated to the dimension length d of the dataset.

II.II Word Representations

II.II.1 Word2Vec

Word2Vec [Mikolov et al., 2013] is a neural learning frame work for learning word embeddings. It was developed by Tomas Mikolov in 2012. The process consists of representing words as vectors in some dimensional space and this vectors can be further used for performing several tasks that includes machine translation [Wolf et al., 2014] and sentiment analysis [Liu, 2017]. The paper proposes 2 different strategies for generating word embeddings this includes Continious bag of words (CBOW) and the Skip-gram model. Unlike other methods that uses concurrent counts. Word2Vec takes a different approach, the CBOW tries to predict a target word based on the context word while the skip-gram model performs the inverse of the CBOW method as it tries to predict a context word based on the target word. The main aim of Word2Vec is to learn similarities of words based on the context they are in. As illustrated in the diagram. We can see that the word "jumps" is similar to "leaps" as they appear in the same context.

From the word2vec paper after learning word representations, they emphasise this point by visualising a lower dimensional representations of words in the same context appears closer together. This includes countries and their capitals. The famous example is the "man is to woman as king is to queen".

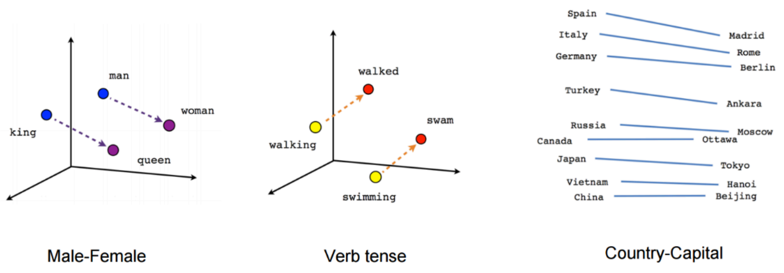
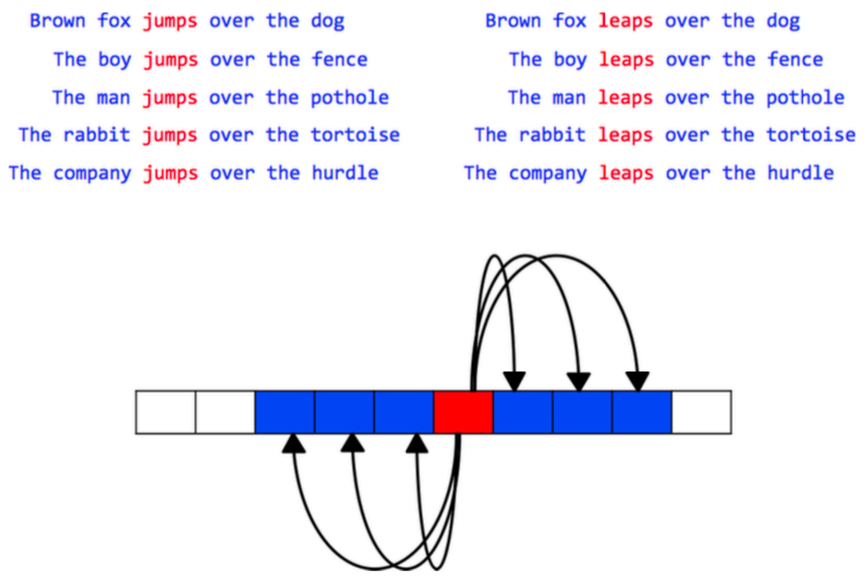


Figure 4.14: Word2Vec Representation

To formalise what Word2Vec is doing, each word is represented as a one hot encoding word of the vocabulary size $|V|$ fig. 4.16. This simply means that each word represented by a sparse matrix with a single index denoting where the word occurs. The aim is for the word to predict a context word in the skip-gram model which is another one hot encoding using a multilayer perceptron or feed forward network with one hidden layer. The aim is to learn weight d representations of each word in the vocab. The Word2Vec framework uses the NCE sampled loss function with cosine similarity to minimise its loss (section III.1). With this words that appears in the same context cluster together. The learned weights d are then used as vector representation for each word.



```
 Brown fox jumps over the dog          Brown fox leaps over the dog
   The boy jumps over the fence          The boy leaps over the fence
   The man jumps over the pothole        The man leaps over the pothole
The rabbit jumps over the tortoise    The rabbit leaps over the tortoise
The company jumps over the hurdle     The company leaps over the hurdle
```

Figure 4.15: Word2Vec showing the similarity between the word leaps and jump

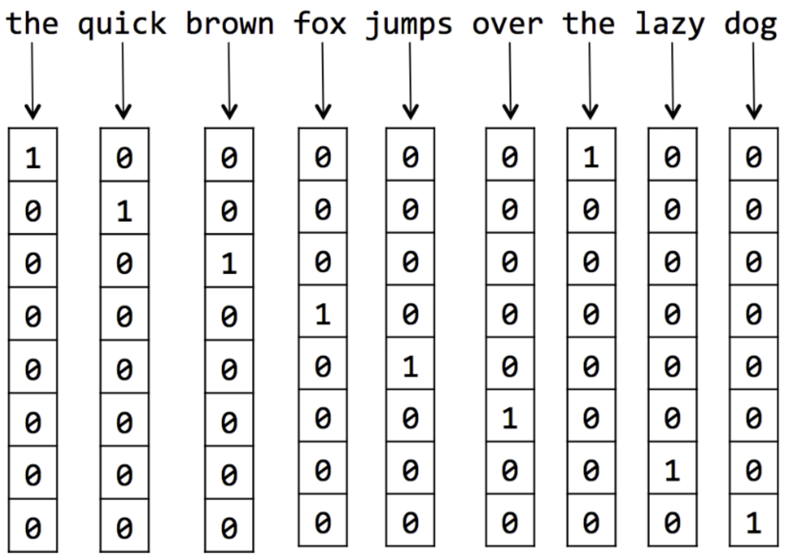


Figure 4.16: One hot encoding representation

II.II.2 GloVe

Gloval vectors or GloVe [Pennington et al., 2014] is a count based approach like LSA [Deerwester et al., 1990] which utilises concurrent statical information (counting what word appears before another word), and then perform dimensional reduction using matrix factorisation techniques like PCA section I. Unlike other methods like LSA, glove applies a log-smoothing transformation to the counts before performing dimensional reduction.

II.III Autoencoders

Autoencoders [Rumelhart et al., 1986] are deep learning strategies for performing unsupervised and transfer learning. Unlike PCA, auto encoders are simple circuits which aim to learn input representation with a minimal amount of distortion. Autoencoders are not only useful for learning feature representation of the input data, they can also be generalized to create new data from the trained data distribution [Kingma and Welling, 2014]. Autoencoders works by simply learning the same input data shown in fig. 4.17.

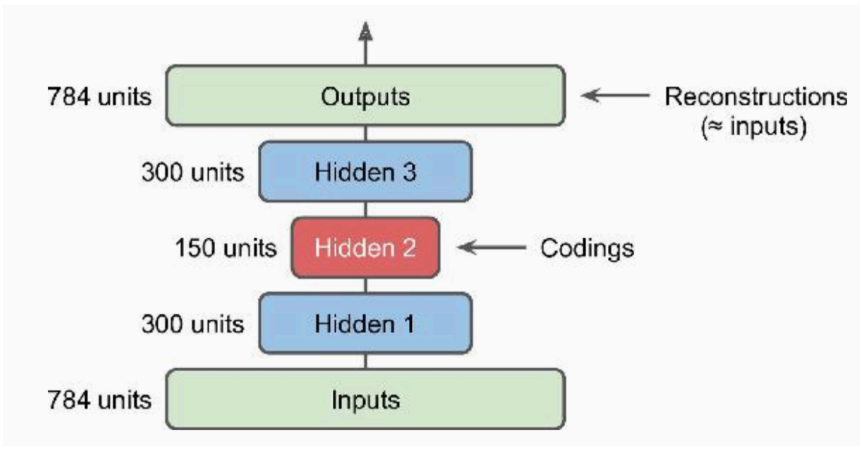


Figure 4.17: Deep Autoencoder

II.III.1 Sequence to Sequence Encoding

Sequence to sequence learning [Sutskever et al., 2014] is a machine learning approach that is used to map a series of input sequence to a series of output sequence. This architecture is used a lot in Machine translation [Neubig, 2017] E.g. English to french. To elaborate this point further, shown in the picture fig. 4.18, this architecture can be used to model a chatbot that can provide responses based on the input sequence. It receives as inputs the sequence "How are you <EOL>" and the program will respond with "I am fine <EOL>" where <EOL> denotes the end of line.

This architecture consists of two parts, an encoder, and a decoder. The encoder is reponsible for taking the input sequence and encoding it to a lower-dimensional representation. The decoder is responsible for taking this encoded representation and prodicing a series of outputs This architecture uses word representation mentioned in section II this includes the word2vec, or the GloVe model as word embeddings. It takes this word representations and passes them into the a recurrent neural network as inputs, and then takes the encoded point which represents the last hidden unit of the encoded sequence

In our model, this architecture was used for encoding the headlines into a lower dimensional representation. This was motivated by [Kiros et al., 2015], where the same principle of encoding word vectors was used in several tasks such as semantic relatedness, paraphrase detection, image-sentence ranking. In our model the inputs and output was represented as the headline. The aim was to take a lower representation of the sentence sequence. This was done by taking the encoder hidden units vectors as the summarisation of the input sequence.

For the first stage of encoding the headline into a point, the [Dyer, 2014] Noise contrastive estimation (NCE) cost function was used, as the sequence to sequence model performs a classification operation over the vocabulary size, which is very large. Using the softmax function for computing the probability distribution over the outputs this will lead to exponential computational time.

The NCE loss function uses a binary logistic regression function to compare the embeddings of the target word with the embedding of the context word and random sampled non-context words. The probability of the non context words are summed up and subtracted from the probabilities of the context word comparison. The aim is to minimise the result attained from this operation.

For the second stage, this process was repeated into minimising the distance of the encoded headline from it self, hence performing another encoding operation. The loss function used in this case was the least mean squared method.

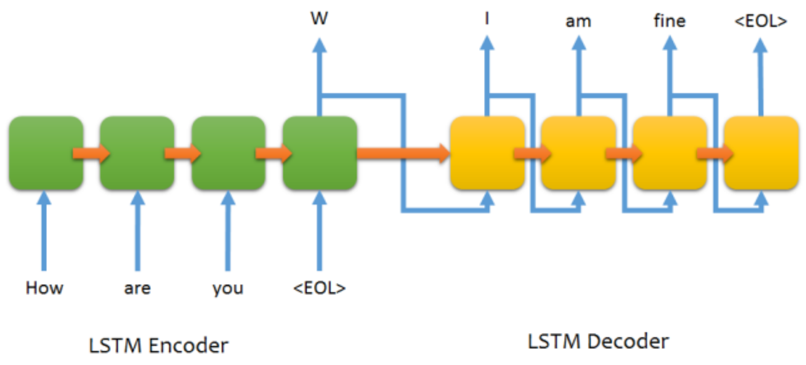


Figure 4.18: Sequence to Sequence Autoencoder

5. *Requirement Specification*

I. Aims and Objectives

Rather than try every neural architecture imaginable, we shall try and mimic best practices applied by investors daily. Hence, in the same way that convolutional neural networks (CNNs) mimic our own visual cortexes, our neural architecture shall mimic the discovery process followed by investors. Since we are focusing on technicals, as well as fundamentals appearing in textual form, we must attempt to find an architecture that looks for price chart patterns similarly as a day traded, and fundamentals as both an arbitrager and a longer term investor. Finally we must identify a method to combine these in an efficient way [Ngiam et al., 2011].

Taking these in step, day traders (not considering event arbitragers yet) attempt to gauge the direction of the herd and profit from it. They are in affect comparable to cattle herders. In trying to fulfil there aim they primarily concentrate on identifying patters in price charts, looking for $patterns$ appearing in conjunction with the passage of $time$. Elaborating, a day trader may look for support and resistance lines (bands in which prices move), or heads and shoulders [Investopedia, 2017] (a famous technical patter that some investors swear by), and make a trade based on his belief of the likelihood of this pattern continuing tomorrow, in the next hour or etc. The difficulty in mimicking this kind of analysis is that neither CNNs - which look for patterns and make decisions irrespective of where (time t or t-10) they occur - nor recurrent neural networks - which try and analyse the cumulative effects of the passage of time, or addition of variables - on their own replicate this process completely. Ideally we would like an architecture that allows us to capture both the presence of patterns and the passage of time. Consequently, we choose to input price and volume data into two parts of our network - one CNN network, in conjunction with multiday 'headlines vectors', and one long short term memory RNN (LSTM). (see architecture section for more details)

Other option: Consequently, we choose to implement a modern max pooling RNN [Lai et al., 2015] when considering price and volume data (see architecture section for more details).

6. *Design*

I. Headlines and NLP

I.I Encoding Single Headline

I.II Headline Encoding

II. Technical Analysis and Informaiton

Cesari and Cremonini (2003) suggested that technical analysis (TA) is the oldest investment appraisal technique used to beat markets. TA allows traders to evaluate the 'breadth and duration of price trends', providing evidence for imminent/immediate market movements before fundamental analysis (Doug Standefer, June 2003). It makes one assumption: that prices do not follow a markov process.

Whilst many papers have been published rebuffing the claims of technical analysts, perhaps most prominently a paper by Fama and French, numerous other papers have been published providing support for the technique. Regardless, we set out to try and use price and volume data to distinguish between buy, sell, and hold points in our n dimensional field.

Traditional technical analysis, as you have discovered, focuses on using price and volume data as inputs in signal identifying functions. Elaborating, individual price data can be passed through functions such as the Moving Average Convergence Divergence function (better known as MACD) or Relative Strength Index function (RSI). These try and extract important trading signals from the seemingly random noise that is a price chart, allowing investors' to identify immediate trading signals (eg: when RSI dips above/below 70/30). These functions have numerous hyperparameters (eg. lenght of input data and etc), which are usually optimised over the training set. The problem with these techniques is that they tend to overfit on the training set, and rarely perform well over the test set. This is because no actual trading signal is identified in the process - the technique identifies esoteric relationships that would have worked in the past, however no real method for separating out noise is found.

A plethora of other techniques exists, but rather than waste time outlining these I move on to our goal. Similarly to technical analysts we are taking price and volume data as inputs. The more information we use (adjusted closing price, open and close spread, volume, and etc - data downloaded from yahoo finance) the better our network should be able to identify clusters using technical information. However there is a difference between using more useful information, and using simplified, and possibly completely irrelevant, information. Expanding upon this point, using RSI or any other technical indicator is pointless as: all such information is present in the price chart already, the process is not supported by theory (no theory other than pure data mining), and it relies enormously on blindly setting hyperparameters (data mining of training set with no link to test set). Whilst the inherent amount of noise and lack of data may make our task of identifying clusters using raw price and volume data impossible using a neural network, methods (supported by theory) for efficiently extracting information from price and volume data exist. These are the methods we shall focus on.

Before continuing, below we describe two observations linked to a so called momentum effect that researchers have tried to explain using risk factor theory: the principle that risk should be compensated by reward. The medium to long term momentum effect, in simple terms, notes that securities that have moved up in value more than the market average are likely to continue moving up in value, and those that have declined in value are more likely to continue declining in value. (FOOTNOTE: Requires observing a medium to long term horizon of returns, and typically continues for a period longer than a month.) The short term momentum effect (FOOTNOTE: better linked to the liquidity premium) on the contrary notes that companies that outperform the market today (in the short term) tend to underperform tomorrow (in the short term).

The reason that we spent space explaining the above, is that many researchers attribute the success of technical analysis to these phenomena. Moreover, an understanding of the above allows us to determine the data preprocessing required to capture real persistent trading signals without having to worry too much about hyperparameter (FOOTNOTE: whilst cross validation would still help identifying the optimal return horizon and etc, at least we have theoretical backing for our choice) s. We refer to paper 1 summarized in our outline (TECHNICAL's section) that described the use of NNs for effective 'technical' signal capturing.

Per the paper mentioned, treating returns as panel data (both cross sectional and time series) allows us to capture an important trading signal. Elaborating, in every time period we can calculate the average and stdev daily/monthly return for all the companies being considered, and then use these to convert the returns to Z scores. Our model input would then consists of Z scores instead of returns. Similarly, to capture things such as the liquidity premium, we could calculate Z scores for volume as well. The above mentioned are two ways of 'actually extending' the data; unlike MA cross for instance.

SUMMARY OF WHAT NEEDED FOR TECHNICALS: DATA: Z score for price/return and volume nominal volume (potentially allows for better individual security pattern spotting - heard behaviour) adjusted close and adjusted open (adjusted open calculation outlined previously...... or just open close difference)

TECHNIQUE ADDITIONALS: if focusing on next day return we shall only have to focus on 20 days of data as we shall be capturing the short term momentum effect. Otherwise have daily and monthly returns (like paper 1.... and all other data mentioned above)

(Market forces: technically speaking…, Doug Standefer, June 2003)

7. *Implementation*

Most of the work done on this project was done using Tensorflow. This involves both the construction and visualisation. Other python packages was also used like Scikit Learn and Pandas. The first step was to encode the headlines in to a lower representation. The second step was to encode multiple headlines for each company for each day to a lower representation (Daily Encoder). The networks mainly used involved sequence to sequence auto encoders and machine translation. During training time a lot of factors was considered which includes the size of the hidden layers, learning rate parameters and types of optimisers. The main optimiser used was the Adam optimiser. In addition, an early stop procedure was added to make sure if the validation error does not improve after 30 epoch, the network gets terminated. The best epoch with the lowest validation error is selected as the best model.

I. Headlines Encoding

The first auto-encoder consisted of auto-encoding the headlines using machine translation. Unlike deep auto encoders which uses least squared method as its loss function, machine translation uses cross entropy with logits, and the errors is measured with perplexity. How well the network can translate the value from one to the other.

The inputs are represented as 3 dimensional tensors (Batch size, number of steps, frame dimension). Using an embedded matrix the frame dimension would be represented with a single integer value which would be used to index the embedding matrix for the corresponding vector. This value would then be fed into the network. This can be illustrated further by the image fig. 7.1. It is to be noted the lstm layers can be stacked up to form deeper layers. In the hyper parameters this would be denoted by the RNN depth parameter.

To further explain what this step comprises of This will take a headline "Google ($a300d$) buys ($300d$) youtube($300d$)" which would be 900 dimensional vector in total and compresses it to $300d$ vectors in the encoded state.

The hyper parameters chosen are listed as follows.

Hyperparameter	Value
Network type	LSTM
Batch size	64
Hidden neurons size	300
Regularisation parameter	0.3
Learning Rate	0.001
RNN Depth	1
Epoch	200

Table 7.1: Headline Encoding Hyper parameters

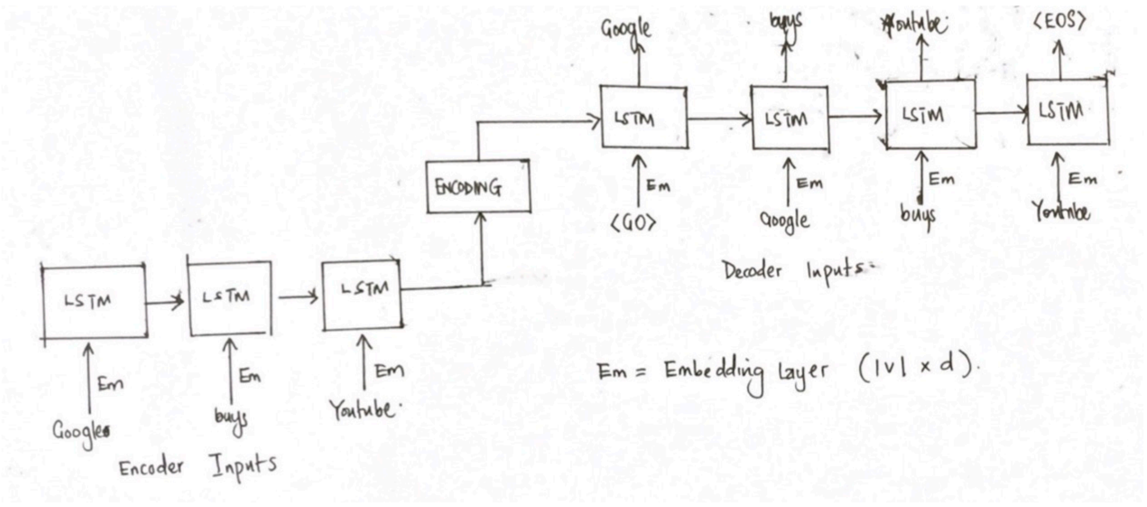


Figure 7.1: Headline Autoencoder

I.I Loss and Evaluation

The graph shows a steady conversion rate after the first epoch. and the validation loss decreases with the training loss.

Epoch	Training Error	Validation Error
1	10.60524	10.62446024
2	4.2282748	62949
3	2.816638	6.019030
4	2.122330	6.1285
5	1.7702149	6.33384
6	1.54610	6.581824

Table 7.2: Headline Encoding

II. Daily Encoding

For the daily encoding we used the learning through loss error propagation as mentioned in loss method mentioned in section I. But this time we changed a couple parameters. In this method we also included bi-directionality to see if this would improve learning time.

Hyperparameter	Value
Network type	LSTM
Batch size	32
Hidden neurons size	150
Regularisation parameter	0.3
Learning Rate	0.0001
RNN Depth	1
Epoch	200

Table 7.3: Day Encoding Hyperparameters

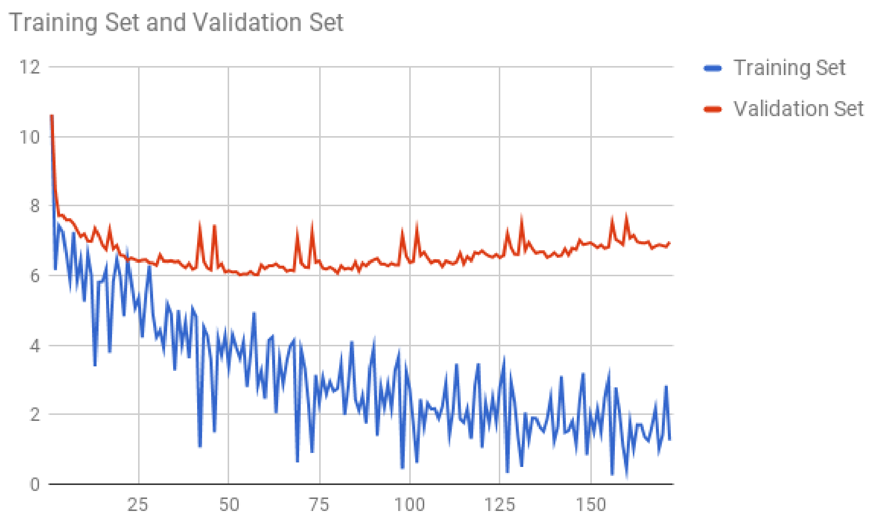


Figure 7.2: Headline Encoder Loss

II.I Visualisation

We used PCA to visualise the data to see what types of headlines clusters together, to get an understanding of the network is doing.

II.II Loss and Evaluation

The loss function used was least squared error. The values from each epoch are as follows. For visualisation purposes the errors were log transformed.

Epoch	Training Error	Validation Error
1	129.559	240.44
2	118	99
3	106	45
4	91	23
5	53	12.14
10	14.07	12.521
20	2.105	12.68
50	0.044	12.417

Table 7.4: Day Encoding Loss per epoch

III. Technicals Encoding

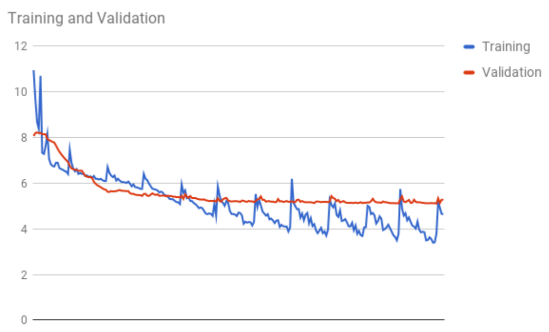


Figure 7.3: Encoded Day Loss

8. *Experiments*

9. *Evaluation, Critical Appraisal*

10. *Conclusion*

Bibliography

[Amodei et al., 2015] Amodei, D., Anubhai, R., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Chen, J., Chrzanowski, M., Coates, A., Diamos, G., Elsen, E., Engel, J., Fan, L., Fougner, C., Han, T., Hannun, A. Y., Jun, B., LeGresley, P., Lin, L., Narang, S., Ng, A. Y., Ozair, S., Prenger, R., Raiman, J., Satheesh, S., Seetapun, D., Sengupta, S., Wang, Y., Wang, Z., Wang, C., Xiao, B., Yogatama, D., Zhan, J., and Zhu, Z. (2015). Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595.

[Chung et al., 2014] Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

[Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, 41(6):391–407.

[Donahue et al., 2014] Donahue, J., Hendricks, L. A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Saenko, K., and Darrell, T. (2014). Long-term recurrent convolutional networks for visual recognition and description. *CoRR*, abs/1411.4389.

[Duchi et al., 2010] Duchi, J., Hazan, E., and Singer, Y. (2010). Adaptive subgradient methods for online learning and stochastic optimization. Technical Report UCB/EECS-2010-24, EECS Department, University of California, Berkeley.

[Dyer, 2014] Dyer, C. (2014). Notes on noise contrastive estimation and negative sampling. *CoRR*, abs/1410.8251.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

[Hinton et al., 2012] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[Hubel, 1959] Hubel, D. H. (1959). Single unit activity in striate cortex of unrestrained cats. *The Journal of Physiology*, 147(2):226–238.

[Investopedia, 2017] Investopedia (2017). Heads and shoulders pattern.

[Ioffe and Szegedy, 2015] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In *Proceedings of the Second International Conference on Learning Representations (ICLR 2014)*.

[Kiros et al., 2015] Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R. S., Torralba, A., Urtasun, R., and Fidler, S. (2015). Skip-thought vectors. *CoRR*, abs/1506.06726.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, pages 1097–1105, USA. Curran Associates Inc.

[Krogh and Hertz, 1991] Krogh, A. and Hertz, J. A. (1991). A simple weight decay can improve generalization. In *Proceedings of the 4th International Conference on Neural Information Processing Systems*, NIPS'91, pages 950–957, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Lai et al., 2015] Lai, S., Xu, L., Liu, K., and Zhao, J. (2015). Recurrent convolutional neural networks for text classification. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 2267–2273. AAAI Press.

[Lample et al., 2016] Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K., and Dyer, C. (2016). Neural architectures for named entity recognition. *CoRR*, abs/1603.01360.

[Liu, 2017] Liu, C. (2017). Packing topological minors half-integrally. *CoRR*, abs/1707.07221.

[McCulloch and Pitts, 1988] McCulloch, W. S. and Pitts, W. (1988). Neurocomputing: Foundations of research. chapter A Logical Calculus of the Ideas Immanent in Nervous Activity, pages 15–27. MIT Press, Cambridge, MA, USA.

[Mikolov et al., 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546.

[Mountcastle, 1957] Mountcastle, V. B. (1957). Modality and topographic properties of single neurons of cat9s somatic sensory cortex. *Journal of neurophysiology*, 20(4):408–434.

[Neubig, 2017] Neubig, G. (2017). Neural machine translation and sequence-to-sequence models: A tutorial. *CoRR*, abs/1703.01619.

[Ngiam et al., 2011] Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., and Ng, A. (2011). *Multimodal deep learning*, pages 689–696.

[Noh et al., 2015] Noh, H., Hong, S., and Han, B. (2015). Learning deconvolution network for semantic segmentation. *CoRR*, abs/1505.04366.

[Pearson, 1901] Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *In EMNLP*.

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA.

[Rumelhart et al., 1988] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to Sequence Learning with Neural Networks. *ArXiv e-prints*.

[Wolf et al., 2014] Wolf, L., Hanani, Y., Bar, K., and Dershowitz, N. (2014). Joint word2vec networks for bilingual semantic representations. *Int. J. Comput. Linguistics Appl.*, 5:27–42.

[Yao and Huang, 2016] Yao, Y. and Huang, Z. (2016). Bi-directional LSTM recurrent neural network for chinese word segmentation. *CoRR*, abs/1602.04874.