View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by DCU Online Research Access Service

A Configurable Deep Network for High-Dimensional Clinical Trial Data

Jim O' Donoghue, Mark Roantree
Insight Centre for Data Analytics
School of Computing
Dublin City University
Glasnevin, Dublin 9
Email:jodonghue, mroantree@computing.dcu.ie

Martin Van Boxtel
Department of Psychiatry and Neuropsychology
School for Mental Health and Neuroscience
Maastricht University
Netherlands
Email: martin.vanboxtel@maastrichtuniversity.nl

***Abstract*—Clinical studies provide interesting case studies for data mining researchers, given the often high degree of dimensionality and long term nature of these studies. In areas such as dementia, accurate predictions from data scientists provide vital input into the understanding of how certain features (representing lifestyle) can predict outcomes such as dementia. Most research involved has used traditional or shallow data mining approaches which have been shown to offer varying degrees of accuracy in datasets with high dimensionality. In this research, we explore the use of deep learning architectures, as they have been shown to have high predictive capabilities in image and audio datasets. The purpose of our research is to build a framework which allows easy reconfiguration for the performance of experiments across a number of deep learning approaches. In this paper, we present our framework for a configurable deep learning machine and our evaluation and analysis of two shallow approaches: regression and multi-layer perceptron, as a platform to a deep belief network, and using a dataset created over the course of 12 years by researchers in the area of dementia.**

I. Introduction

Dementia is a serious loss of cognitive ability beyond what might be expected from normal ageing. Worldwide, the number of people with dementia is currently estimated to be 44 million and expected to reach approximately 76 million by 2030, and 135 million by 2050 [18]. While dementia is a chronic and progressive illness, with no cure, and one of the most feared age-related conditions, there is now strong evidence that dementia can potentially be delayed by adopting lifestyle changes in mid-life in relation to cardiovascular health, mood, diet and physical and cognitive activity [9]. Many healthcare research projects have progressed into the realm of attempting to analyse and predict outcomes based on a person's lifestyle. The Innovative Mid-life Intervention for Dementia Deterrence (In-Mindd) project [21] is a collaboration between dementia and data science research teams across Europe with an aim to create a cloud-based profiling tool to assess the relative risk of developing dementia for individuals of middle-age. In previous work, this involved federating existing healthcare systems in projects such as [19][6] to create a single patient record similar to the standards proposed in the HL7 standard [2]. With In-Mindd, the project involved the development of an on-Cloud profiler and algorithm to predict a score indicating the risk of a patient developing dementia in later years.

In recent times, there have been many examples of data mining for sensor networks. In healthcare, efforts such as [20] and [23] used sensor networks to harvest high volumes of personal health data for usage in data mining activities. However, clinical studies cannot employ automated harvesting mechanisms for data collection. Instead, longitudinal clinical studies contain general and biometric data gathered on a cohort of participants at various time points over a number of years. Consequently these are high-dimensional (many features, or poor sample to feature ratio) and temporal datasets, which are prone to sparsity or missing values. As part of the evaluation process, we attempted to identify sets of features which were strong indicators of a predictable outcome. Deep learning has proven successful for learning models in high-dimensional [13], [24], [16] and temporal [14] datasets, as well as those with significant numbers of attribute values missing [25]. Deep learning, or deep architectures refer to algorithms where multiple-layers of hidden, usually latent variables are learned through many layers of non-linear operations [7], usually in the artificial neural networks context. However, the issue of high-dimensionality, discerning from a large number of features those which best predict the class or outcome, remains a challenge even using deep architectures.

Motivation. As part of the In-Mindd project, researchers are using an ontology containing the MAAS data set [22], [15] which recorded a very high number of features over a 12-year span in an attempt to discern the determinants of cognitive ageing from behavioural characteristics and other biometrics. However, it is unclear as to which set(s) of features would provide the best predictive capabilities for a particular outcome. As different approaches have been shown in the past to deliver varying degrees of success, our motivation is to develop a framework by which we can test different combinations of features in different deep learning configurations or machines as they are known. By evaluating the success of learning different features representations with different deep learning architectures, our goal is to demonstrate the configuration most applicable to prediction in clinical studies for dementia.

Contribution. Deep learning architectures have been primarily used in image, audio and video domains where the feature sets are often large and complex. More traditional or shallow architectures are used in text based datasets. However, we feel that the high dimensionality of clinical studies provides a perfect domain problem for the deep learning approach. In this research, we are one of the first projects to attempt to use a deep architecture for a dementia-based study. Our contribution

is to develop an easily-configurable machine that provides Regression, Multi-Layer Perception (MLP),and Deep Belief Network (DBN) approaches as part of an overall framework for identifying the feature sets for higher levels of predictive accuracy and comparing the above approaches as well as allowing for many experimental runs and test case scenarios.

Paper Structure. The remainder of this paper is structured as follows: In Section II, we discuss the related research in this area; in Section III, we briefly present terminology and an overview of the deep architectures involved, as well as a detailed discussion of our Configurable Deep Network; in Sections IV and V, we present a detailed series of experiments, with results and analysis to illustrate this the framework approach; and finally in Section VI, we offer some conclusions.

II. Related Research

Traditionally, feature representation or selection, and subsequent model learning are treated as separate processes in the medical context [4], [10]. Arauzo-Azofra et al. [4], review the suite of traditional feature selection methodologies and their efficacy with various shallow learning algorithms. They formalise the process of feature selection into two steps: the first scores an individual feature's predictive power; and the second applies a cutting criterion (methodology which removes all features under a certain threshold). It was found that when a certain cutting threshold was exceeded, greater reductions lead to poor prediction accuracy meaning relevant features were lost. They also found no generic method to perform best overall, as each learning algorithm worked better with a different feature selection method.

Similar work [10], proposed a new confidence metric for medical data classifications. To select relevant features, they applied a single variable classifier to only those instances with no missing data and measured their performance using the area under the receiver operation characteristic curve (AUC) measure. The final score calculated the average AUC over multiple classifiers and the features were ranked according to this score. There were originally nine features and the top four ranked were selected to train upon. This number was chosen as the accuracy of the models learned improved up until a sixth feature was discarded. Subsequently a number of multi-variate experiments were performed using different feature combinations building multiple models in all possible combinations, until performance degraded - for example: all nine features, the top four and so on.

Both of the aforementioned cases use a two-step, manually intensive process of choosing a feature representation to learn predictive models, with shallow algorithms in a medical context. They sometimes eliminate relevant features [4] and do not deal with high dimensional data as in one case [10] only nine features were contained in the original dataset. Such a manually intensive, two-step process would not scale to high-dimensional data. Individually ranking each feature [4], [10], building multiple models with different combinations of features [10] and having to determine which feature selection method is best out of a suite of methods [4] is intractable where thousands of features are present in a dataset.

The power of deep algorithms for feature learning and building accurate predictive models can be seen in a recent development by the Google research team [16]. In this work they successfully learn high-level features, training a face detector in a completely unsupervised fashion by using a deep sparse auto-encoder on a dataset of 10,000,000 images. It demonstrates that abstract and relevant features can be learned not only from labelled but also unlabelled data. Their work focuses on auto-encoders, whereas ours implements a Deep Belief Network (DBN) using a configurable framework and has to be trained on a much smaller number of instances. Work similar to that described, but instead using a DBN, can be found in [13]. Like Le et. al [16], and as is often-times the case with deep learning, they apply their methodologies to an artificial intelligence task, which in these cases is computer vision. We apply our DBN to clinical trial data analysis, in an extensible architecture that allows for the easy roll-out of multiple configurations, hence providing a building block approach to constructing a DBN. As clinical data provides a new domain for this approach, it is crucial to understand how different configurations perform.

There have been recent developments in bioinformatics that use DBNs for medical text classification [27] and healthcare decision making with electronic medical records [17]. As yet, none have applied deep learning to clinical trial datasets. The work of Yepes et al. use DBNs to assign descriptive Medical Subject Headings (MeSH®) to MEDLINE®citations and compare them with the results achieved by completing the same task with a Support Vector Machine (SVM). They found that in smaller datasets the SVM outperformed the DBN but when there was more data to train on the DBN significantly improved upon the shallow models. Their DBNs are somewhat limited in configuration, as both consisted of only 3 layers, with one hidden layer, where one configuration had 250 units in each layer and the other had 500. It is also arguable that these are shallow models as they consist of only three layers [7], where a lot of the power of deep networks comes from multiple hidden layers [16], although they do employ an unsupervised pre-training step as is characteristic of most deep architectures. Similar to our work, their code is written on Theano [5], [8], but they do not implement their own architecture, instead using a third-party implementation [1].

The work of Liang et al [17] also focus on text classification as opposed to our efforts with clinical trial data. They found that the DBN is successful and outperforms the standard SVM or decision tree (DT) models, extracting relevant features and performing more accurate classifications on data. They do not however, describe a single algorithm architecture that can implement both shallow and deep algorithms and be extended to incorporate different hidden units and activation functions. They employ a SVM as their classification output layer which could arguably be more powerful in classification than the softmax regression layer which we used. In both of the aforementioned projects they also do not provide possible heuristics on how to choose the hyper-parameters of the learning rate[1] and regularisation term[2] or model configurations for the DBN.

[1]Learning rate is a coefficient for the model parameter updates and decides how big of a step to take in gradient descent.

[2]The regularisation term is the coefficient for the regularisation terms of L1 or L2 norm and decides on how much to penalise the data values.

III. The Configurable Deep Network

Every deep network is composed of two main constructs: *nodes* which execute the activation functions, and *layers* which contain the nodes. The bottom or *input* layer contains a nodes which reflects each feature in the dataset. At the top of each configuration (with the exception of Restricted Boltzmann machines), there is an *output* layer which performs, for example, a classification function.

For most deep networks, each node in layer $L^{(i)}$ is connected to every node in layer $L^{(i-1)}$, and to every node in $L^{(i+1)}$. The activation energy from each node in Layer $L^{(i)}$ is passed to every node in layer $L^{(i+1)}$. In Figure 1, the links shown connecting nodes from $L^{(i-1)}$ to $L^{(i)}$ are only shown for the first node in $L^{(1)}$, but are in fact, repeated for every node in $L^{(1)}$. A vector of weights forms part of the input into each node, giving a particular weight to multiply by a particular activation energy depending on the activation energy's origin and destination.

Each layer $L^{(i)}$ contains a set of nodes $n_1^{(i)}$ to $n_m^{(i)}$, where each node $n_j^{(i)}$ contains a non-linear activation function through which the sum of the products of all inputs go through (Equations 1 and 2/3) to generate an energy value $a_j^{(i)}$. In all configurations shown in Figure 1, a matrix of weights is randomly generated for each layer $L^{(i)}$, containing a distinct set of weights $W_j^{(i)}$ for every node $n_j^{(i)}$ in $L^{(i)}$. Thus, the two sets of input parameters to every node $n_j^{(i)}$ are the weights $W_{ij}^{(i)}$ and the energy values $a_i^{(i-1)}$. This occurs for every $\{W_{ij}^{(i)}, a_j^{(i)}\}$ to reflect the connection between each node in $L^{(i)}$ to each node in $L^{(i-1)}$.

We now describe the four configurations currently implemented in our architecture in Python and built upon Theano [8], [5]: Regression, Multi-Layer Perceptron(MLP), Restricted Boltzman Machine (RBM) and Deep Belief Network (DBN), which are displayed in figure 1 and are based on the research presented in [7]. Each configuration has the following functions described below.

- `initialise`. This function instantiates the model parameters of a particular configuration: the number of inputs and outputs, nodes, weights and biases for that architecture. It associates the necessary hyper-parameters (like learning rate and the regularisation parameter) with the architecture.
- `buildhypothesis`. This function builds the symbolic expressions for the hypothesis i.e. the function that classifies or gives you Y for a particular sample (explained in further detail in the configurations below).
- `buildcost`. This function creates a symbolic cost expression based on the classification type or algorithm output. It also adds regularisation to the cost function if required. If applicable, it also builds an expression to compute output error.
- `buildmodel`. This uses Theano's automatic differentiation to compute the gradients of the cost function and thus, builds a symbolic expression for the update of the parameters of the model. It subsequently compiles these expressions into Theano functions for training and pre-training (if applicable), on training, validation and test data.
- `pretrain`. This function is only relevant to RBM and DBN configurations. Its purpose is the unsupervised pretraining of the layers, before the fine-tuning stage. It essentially puts the parameters of the model in a bound where they can effectively represent the data, narrowing the search space for the fine-tuning function and is similar to creating clusters based on the data.
- `train`. This function fine-tunes the model with respect to an outcome, not relevant in the case of the RBM.

A. Regression

The `Regression` class is the simplest module in our architecture and is called upon as the output layer in the `MLP` and `DBN` classes. If other classification algorithms are added to the architecture in the future like the Support Vector Machine, this module is easily interchangeable as an output layer. Currently our `Regression` layer can handle linear, binary-class and multi-class classification where, in the case of multi-class classification, the class is mutually exclusive.

In the case of running regression, as a model itself or as the output layer in a neural network, it is initialised based on the type of data one is modelling. In the case of linear data, only Equation 1 is executed in the *hypothesis* function, multiplying the input X by the model parameters (weights with the bias term) θ. In the case of binary classification and multi-class classification, the logistic sigmoid (Equation 2) or the softmax (generalised sigmoid Equation 3) expressions are called respectively on the output of the linear function, all shown below in matrix notation. In the case of linear data, the output is an estimate of what the continuous output should be, whereas in the case of the logistic and softmax functions it outputs a probability of class membership, where the different classes are class 1 to class K.

$$z^{(i)} = \theta^{(i)T} \dot{X^{(i)}} \tag{1}$$

$$sigmoid(z^{(i)}) = \frac{1}{1 + e^{-Z^{(i)}}} \tag{2}$$

$$softmax(z^{(i)}) = \frac{e^{z^{(i)}}}{\sum_{k=1}^{K} e^{z_k^{(i)}}} \quad forj = 1, \ldots, K. \tag{3}$$

Next the `buildcost` function is called, again dependant on the classification task. In the case of linear data, the mean square error is sufficient as seen in Equation 4 but for more complicated non-linear models of logistic and softmax regression, cross entropy cost and the negative log likelihood (or negative log probability) are employed as shown in Equation 5 and 6 respectively.

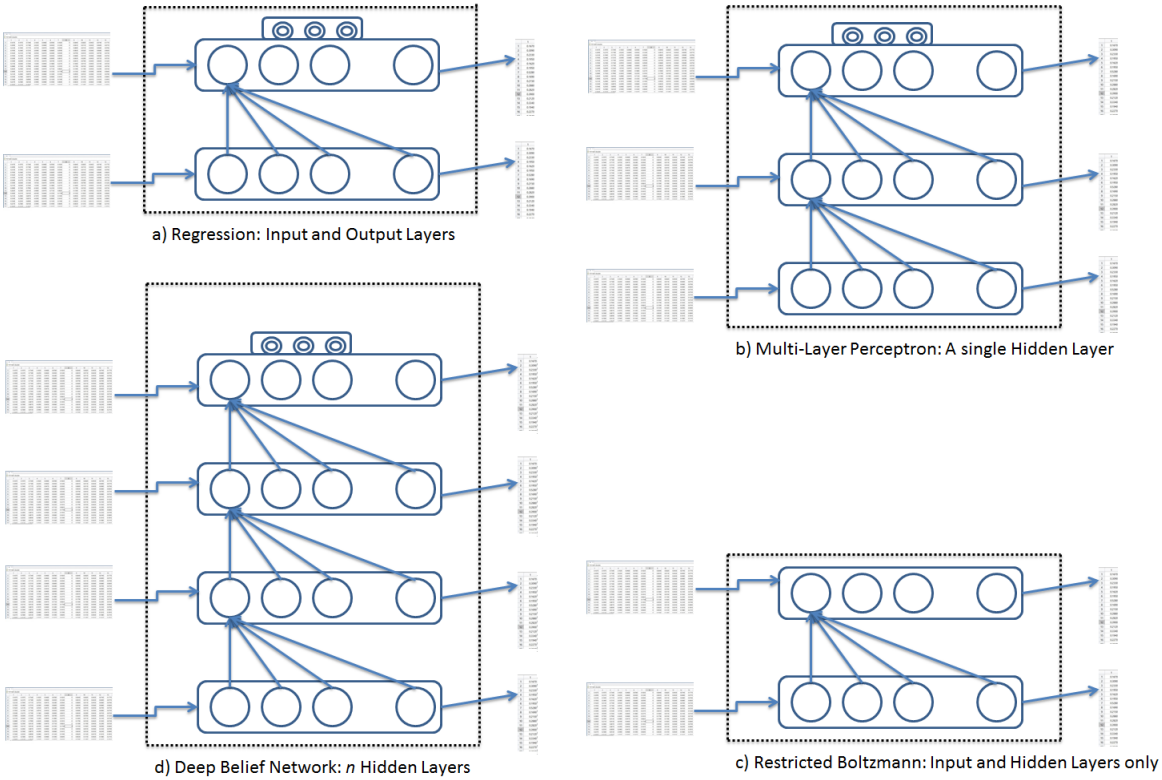

Fig. 1: Machine Configurations

$$\frac{1}{|\mathcal{D}|}\sum_{i=1}^{|\mathcal{D}|}(\hat{y}_i - y_i)^2 \tag{4}$$

$$-\sum_{i=0}^{|\mathcal{D}|} y_i * logP(\hat{y}_i = y_i|x_i,\theta) - (1-y_i) * log(1-P(\hat{y}_i = y_i|x_i,\theta)) \tag{5}$$

$$-\sum_{i=0}^{|\mathcal{D}|} logP(\hat{y}_i = y_i|x_i,\theta) \tag{6}$$

`buildmodel` then calculates the gradient of the cost function with respect to the model parameters and compiles them into Theano functions. Finally, these parameters are updated through stochastic gradient descent in the `train` function, where for each instance in the dataset the derivative of the cost with respect to the model parameters, multiplied by the learning rate is taken away from the current weights for each step.

B. Multi-Layer Perceptron

This is a simple form of one hidden layer neural network where the values are propagated through the network as explained in the start of this section. It is trained via back-propagation in a similar fashion to regression except that the parameters in each layer must be updated with respect to the cost of the predicted output compared to the actual output. It makes use of a further `Layer` class which determines the activation energy function to be used and holds the weights for that layer, taking the relevant activation energy value from the previous layer and multiplying it by the relevant weight before going through the activation function for each node in the layer. The `Layer` class allows for extension and the swapping of different activation functions, when required. The activation function used for this configuration was the logistic sigmoid function as in Equation 2

C. Restricted Boltzmann Machine

An RBM is a two layer neural network, with one hidden and one visible layer. The total energy of an RBM is given by the function shown in Equation 7, again in matrix notation, and the aim is to learn a model which occupies a low energy state for desirable configurations of the data. This is done through maximising the probability of the training data in the hidden and visible layers, learning the joint probability distribution over the visible and hidden layers as shown in Equation 8.

$$E(v,h) = -a^T v - b^T v - v^T W h \tag{7}$$

$$P(v,h) = \frac{1}{Z}e^{-E(v,h)} \tag{8}$$

To train a RBM, the contrastive divergence [12] algorithm is used. This performs a step of Gibbs sampling inside the gradient descent procedure. It takes the visible vector and computes the probabilities of the hidden layer. It subsequently

samples from this probability, computes the outer product of the v and h layers and this is called the positive phase. Next, a sample v' is taken from the hidden layer in an attempt to reconstruct the data. The probabilities of the hidden layer are recalculated from v' and the hidden values sampled from this are h'. The outer product of $v'h'$ is calculated, giving the negative gradient or negative phase. The weights are then updated by multiplying the learning rate against the result of subtracting the negative phase from the positive phase and subsequently, subtracting the result from the weights for each step of gradient descent for each instance in the dataset.

D. Deep Belief Network

A Deep Belief Network is an artificial deep neural network with multiple hidden layers. It is a generative model characterised by unsupervised pre-training, where subsequent dual-layers are trained as back to back RBMs - previously explained in Section III-C, and supervised fine-tuning where parameters are updated with respect to an outcome, in the same way as in a MLP, explained previously in Section III-B.

For our `DBN` class, the initialise phase spools up multiple `Layer` and `RBM` classes in parallel and shares the weights between them. The `pre-train` function trains the layers of RBMs and the `train` function uses the same weight matrices (that can be accessed as they are in a shared variable) and fine-tunes these in respect to the outcome of interest. What training layers initially as RBMs seeks to do, is to tune the weights to fit the data, narrowing the search space for optimising the weights to fit the outcome data. Once again, sigmoid activation functions are used. Finally, an output layer of regression is put on top of the model in order to classify.

IV. Evaluation: Preparation of Datasets and Set-up

A. The Dataset

The dataset used in our evaluation is the Maastricht ageing study, or MAAS dataset [15]. The Maastricht ageing study is a longitudinal clinical trial that took place over 12 years with biometric data on ageing individuals taken at fixed (3 year) intervals over the period of the study. The entire dataset contains 3441 unique records and 1835 unique attributes distributed over 86 'tests' or study subsections.

As this is an exploratory analysis into the efficacy of DBNs for modelling high-dimensional data in the health context, we chose to remove the temporal and sparse aspects of the data. The baseline data was analysed and 414 datasets extracted, containing disjoint sets of participants where all participants took part in exactly the same tests. This removed the temporal aspect of the data as well as the sparsity on a test level. From these datasets we chose a set containing 523 instances (participant records) and 556 features, taken from 14 'tests' or study sub-sections.

The 14 tests in the chosen dataset along with their subject matter were: "ax'_deelname" - whether the respondent will participate in the follow up and reasons; "bezoek_npp" - health complaints and treatments; "conv_oude_nieuwe_patnr_a1" - study wave the participant was in and their national patient number; "genmid_nu" - prescription drugs and how often they are taken; "genmid_vl" - further questions on drugs; "klachten" - information on whether treatments for ailments were sought; "nacrose" - relating to being under anesthesia; "pe_cognstat" - questions relating to memory complaints; "pe_medinfo" - general medical information; "pe_mia" - general memory questions; "pe_psyche" - psychological questions; "pe_voeg2" - general well-being i.e. pains, sleeping, etc.; "planvoort" - administrative information about their participation in the survey; "resp_basis" - where the participant lives, their education and information regarding their career. The variable we chose to predict was"vergeet" which equates to the answer to the question: "do you find yourself forgetful? (yes/no)".

To remove missing data on a feature level, all those features that had over 20% missing were deleted, leaving 347 features. For those remaining any missing data was imputed using a mean imputation.Next the data was scaled to give it unit variance, which makes sure no one feature dominates the model because of large data values. After scaling, some features contained only zeros and thus were removed as they would not have contributed to the model.

At this point features were still continuous, therefore it was required they be categorised into vectors of zeros and ones, due to binomial hidden and visible units in our RBMs. As such, a unique dictionary was created for each feature, similar to the bag of words representation [3]. For each feature, a list of only unique values was extracted, counted, and ordered. Then, for each individual feature value, a vector the same length as the number of unique elements in the feature was created and the zero was changed to a one at the index where that value was stored in the list of unique, ordered elements for that feature, thus mapping each feature to the original extracted key and leaving the feature in a categorical 'one-hot' encoded state. Once the features were categorised the cardinality of each input vector was 3567.

B. Parameter Initialisation

In all of our models the bias terms were initialised to zero. For the MLPs, RBMs, and DBNs we randomly initialised the weights between the upper and lower bounds shown in Equation 9, which are the bounds for the sigmoid activation function, taken from the work of Glorot et. al [11]. fan_{in} refers to the number of inputs into the current node from the previous layer and fan_{out} stands for the number of outputs to nodes in the subsequent layer. This ensures that values can be easily forward and back-propagated through the activation function and mitigates against exploding or vanishing gradients. The weight for the regression model were randomly initialised without upper or lower bounds.

$$[-4\ \sqrt{\frac{6}{fan_{in}+fan_{out}}}, 4\ \sqrt{\frac{6}{fan_{in}+fan_{out}}}] \quad (9)$$

C. Experimental Procedure

Before the DBN was trained, softmax regression and mlp models were learned with many different hyper-parameter configurations in a process known as grid search. This was performed as a proxy for choosing the hyper-parameters of best fit for the deep belief network, as the regression and

Experiment	Initial Train Cost	Train cost	Validation cost	Test cost	Test Error	Epsilon	Lambda	Steps	Data
8, 0, 0	13.188452	0.001	45.818	2.960	0.258	0.9	0.001	100	one-hot
8, 1, 0	**4.925**	**0.002**	**7.725**	**2.909**	**0.305**	**0.9**	**0.003**	**100**	**one-hot**
8, 2, 0	7.608	0.00334	22.615	1.809	0.225	0.9	0.009	100	one-hot
7, 0, 1	21.066	0.003	6.449	2.690	0.391	0.3	0.001	1000	one-hot
8, 1, 1	9.718	0.004	35.637	1.090	0.238	0.9	0.003	1000	one-hot
8, 0, 1	9.200	0.003919	15.913	1.250	0.305	0.9	0.001	1000	one-hot
4, 0, 2	12.103	0.004	14.097	2.73	0.298	0.03	0.001	10000	one-hot
4, 0, 2	16.553	0.004	16.351	3.106	0.338	0.03	0.001	10000	continuous
7, 0, 1	**6.193**	**0.004**	**8.180**	**2.816**	**0.298**	**0.3**	**0.001**	**1000**	continuous
5 0 2	11.149	0.005	9.223	1.090	0.291	0.09	0.001	10000	one-hot

TABLE I: Learning Rate and Regularisation Parameter Grid Search Results - Regression

Experiment	Initial Train Cost	Train cost	Validation Cost	Test cost	Test Error	Data	Epsilon	Lambda	Steps	Nodes
2	2.389	0.17	2.107	0.76	**0.232**	**continuous**	0.3	0.001	1000	337 **10** 2
4	5.319	0.231	4.609	0.889	0.225	continuous	0.3	0.001	1000	337 30 2
6	13.466	0.332	12.436	0.97	0.225	continuous	0.3	0.001	1000	337 100 2
8	33.467	0.456	30.394	1.176	0.238	continuous	0.3	0.001	1000	337 337 2
1	11.247	0.842	11.664	0.974	**0.291**	**one-hot**	0.9	0.003	100	3567 **10** 2
10	64.252	0.929	62.453	2.224	0.232	continuous	0.3	0.001	1000	337 900 2
12	73.305	1.426	78.562	3.5761	0.212	continuous	0.3	0.001	1000	337 1300 2
3	30.256	1.473	35.802	1.363	0.318	one-hot	0.9	0.003	100	3567 30 2
14	121.088	2.211	113.605	5.452	0.219	continuous	0.3	0.001	1000	337 2000 2
5	99.757	2.549	134.606	4.533	0.616	one-hot	0.9	0.003	100	3567 100 2
7	318.028	8.859	378.284	9.371	0.616	one-hot	0.9	0.003	100	3567 337 2
9	795.223	24.819	1057.241	22.025	0.384	one-hot	0.9	0.003	100	3567 900 2
11	1097.768	35.089	1487.919	31.769	0.384	one-hot	0.9	0.003	100	3567 1300 2
13	1581.871	61.142	2381.085	47.163	0.384	one-hot	0.9	0.003	100	3567 2000 2

TABLE II: Layer Grid Search Results - Multi-Layer Perceptron

MLP models can be seen as constituent parts of a DBN. Thus, it was thought individually fine-tuning the constituent parts should contribute to the efficacy of the overall DBN model. To compare the results before and after categorisation, the regression and MLP experiments were run on both categorical and continuous data.

D. Experimental Set-up

All experiments were run on a Dell Optiplex 790 running 64-bit Windows 7 Home Premium SP1 with an Intel Core i7-2600 quad-core 3.40 GHz CPU and 16.0GB of RAM. The code for the experiments was developed in Python using the Enthought Canopy (1.4.1.1975) distribution of 64-bit Python 2.7.6 and developed in PyCharm 3.4.1 IDE. The code makes use of the NumPy 1.8.1-1 and Theano 0.6.0 (and hence its dependencies). All algorithms were trained with stochastic gradient descent. The dataset was also broken into three parts for the training, validation and test sets, in a ratio of 5:2:3.

V. Evaluation: Results and Analysis

A. Regression Grid-Search Experiments

Table I shows the top ten configurations of hyper-parameters found via a grid search for regression, ranked by the lowest cross entropy cost achieved on the training set.

Experiment refers to a particular hyper-parameter configuration. Each configuration was only run twice, once on the categorical (one-hot) data and continuous data. For example code 8 8 0 would refer to the eighth learning rate value to be tested, the eighth regularisation value, and the first value for the number of training steps. *Initial Train Cost* refers to the cross entropy cost on the data before training. *Train/Test/Validation Cost* refer to the cross entropy cost achieved after training on the training, validation or test sets respectively. *Test Error* refers to a measure of predictive accuracy as can be calculated from Equation 10. *Epsilon* in each experiment refers to the value used for the learning rate and *Lambda* refers to the value used for the regularisation co-efficient. *Steps* tells us the number of steps of stochastic gradient descent taken and *Data* tells us whether continuous or categorical (one-hot) data was used. All figures are rounded to three decimal places.

$$1 - (\frac{true\_pos + true\_neg}{num\_predictions}) \tag{10}$$

Grid search involves testing a range of values for the hyper-parameter configuration. The range of the grid search for both the regularisation and learning rate values was from 0.001 to 1. This range was roughly divided into 10 giving us values of 0.001, 0.003, 0.009, 0.01, 0.03, 0.09 and so on up to 0.9. We chose three values for the number of steps for gradient descent, that is 100, 1000, and 10000. Given such a small sample size we estimated that any more would lead to over-fitting the data (not generalising well to unseen data). All combinations of the above parameters were tested, giving 246 experiments in total.

In bold, in Table I, are hyper-parameter configurations we found as the best performing for the categorical and continuous data respectively. The configuration chosen in both cases was that which achieved the second lowest cost on the training data, this was because for the configurations that performed best on the training data did not perform very well on the validation data and although the model chosen performed worse on the training data it had a significantly lower cost on the validation set, suggesting these hyper-parameters generalised better. We were able to learn a relatively good model with both types of data, achieving an error rate of less that 0.31 for both, meaning that over 69% of the instances were classified correctly for the

one-hot encoded data and over 70% for the continuous data. Immediately striking is that even though the categorical, one-hot encoded data achieves a lower cost, the continuous data performs better in making predictions achieving a lower test error. This may suggest that categorising the data removed some noise present in the continuous data but along with the noise it removed some information necessary to generalise better. Another distinction to be drawn between the categorical and continuous data is that learning rate (epsilon) is much higher for the categorical data and the regularisation parameter is also larger than in the continuous data but ten-times less steps were required suggesting that gradient descent was much larger and steeper on the categorical data and as a result, converging in a smaller number of steps.

B. MLP Grid-Search Experiments

The three figures in the extra column of *Nodes* in Table II refers to the the number of Nodes in the input, hidden and output layers respectively. Shown are the entirety of the 14 experiments performed on the data with a multi-layer perceptron. Seven different hidden layer configurations were tested on the categorical and continuous data, giving 14 experiments in total.

When instantiating a multi-layer perceptron there are considerations beyond the configurations of the hyper-parameters of learning rate, regularisation parameter and training steps. The number of nodes in the hidden layer must also be determined, which could itself be considered a hyper-parameter. As such, we performed a grid-search on the number of hidden nodes with the best performing hyper-parameters from the first set of experiments for the categorical and continuous data respectively. The search for the optimal number of hidden was in the range of 10 to 2000 inclusive, with values of 30, 337 (number of features before categorisation) 900 and 1300 also used for this series of experiments. We chose this sequence of node counts as they are broadly in line with gaps of values tested in our first series of experiments (Regression), although increased by several factors of ten.

From Table II it can be seen that the smallest number of hidden nodes performs best for both the categorical and continuous data - Experiments 2 and 10. The test error for the MLP is an improvement upon that of the test error in regression for both types of data, this time achieving a prediction accuracy of over 70% for both types of data and achieving a predictive accuracy of just under 78% for the continuous data. Therefore, a better representation of the data must have been learned by the MLP. The best performance coming from a small number of hidden nodes suggests that the number of features in the datasets that actually describe the outcome are relatively few, and we use this as a heuristic for choosing the number of nodes in our second last hidden layer in the DBN. Again, it is also clear that the continuous data has greater predictive power when compared to the one-hot encoded data, suggesting that information is lost when it is coded into binary features.

C. Deep Belief Network Versus Shallow

For the DBN run, we use the hyper-parameters found through grid-search for the regression layer on the binary data (again as our RBMs can only currently model binary hidden units), and the number of hidden nodes found to work best from the multi-layer perceptron in the second last layer of our DBN.

Table III compares the best results from the models learned from MLP and regression respectively on both the continuous and categorical data and the initial model learned by our DBN with the heuristic hyper-parameters applied from grid-searches in the regression and MLP models.

Although the Deep Belief Network performs comparably when the cost is examined, it performs worse then the regression and MLP models when predicting. This was partially expected for a number of reasons. First, where there are not a lot of instances to train the DBN, it has been found to perform worse compared to models learned with shallow algorithms [27]. Second, a DBN can take on very high numbers of configurations because of: the cardinality of hidden layers; the cardinality of nodes within each hidden layer; and in settings for other hyper-parameters, we performed only an initial run of the DBN and perhaps did not use the best possible configuration. We were not in a position to test multiple configurations as the time cost incurred was prohibitively expensive on the current hardware. Finally, the data the model was trained on was categorical which, as shown in previous experiments, built less accurate models compared to continuous data. Interestingly the DBN had lowest cost of all the models but performed worst when classifying on the test set. This could signify that the model over-fitted the data and did not generalise to the extent that was necessary or possible did not learn a relevant feature representation.

VI. Conclusions and Future Work

Clinical trials are an important area for data scientists as the provision of good predictive models can have a high impact on society. This is one of the main goals in the In-Mindd project which seeks to reduce the risk of dementia, using profiling and a predictive algorithm. As part of that research, we are building a strategy that involves deep learning architectures to address one of the primary issues with clinical study data: high dimensionality. However, these architectures are complex, with many combinations of weights, feature sets and hyper-parameters. We require a framework in which we can carefully measure the results of different machine and algorithm types (e.g. regression, MLP, DBN) and configurations (e.g. number of hidden layers and nodes and hyper-parameters). In this paper, we presented our Configurable Deep Network and a series of experiments used to evaluate the Regression and Multi-Layer Perceptron components.

While this research is ongoing, our initial results as presented in our evaluation sections, are quite promising. Apart from the ability to easily configure many experimental runs, it also demonstrated a function to tune results to make them more accurate. For current and future work, we are now focused on utilising the Deep Belief Network configuration and using results as feedback to rerun the original experiments. Although the initial DBN run performed poorly against the first 2 series of experiments, we did not make use of many of the main advantages of these models, such as sampling from the RBM to infer missing data and using Gaussian units to model continuous data in order to train a DBN and not lose

Model	Initial Train Cost	Train cost	Validation Cost	Test cost	Test Error	Data	Epsilon	Lambda	Steps	Nodes (In, Hidden, Out)
Regression 1	6.193	0.004	8.180	2.816	0.298	continuous	0.3	0.001	1000	n/a
Regression 2	4.925	0.002	7.725	2.909	0.305	one-hot	0.9	0.003	100	n/a
MLP 1	2.389	0.17	2.107	0.76	0.232	continuous	0.3	0.001	1000	337, 10, 2
MLP 2	11.247	0.842	11.664	0.974	0.291	one-hot	0.9	0.003	100	3567, 10, 2
DBN	0.815	0.693	0.791	0.703	0.616	one-hot	0.9	0.003	100	3567 200, 10, 2

TABLE III: Comparing Deep Versus Shallow

information encoding the features into a binary configuration. Apart from this there are state-of the art regularisation methods [26] which have been shown to vastly improve upon the predictive power of deep architecture.

As part of future research, there are a vast number of possible configurations to test. This was not possible during this initial set of experiments as the computational costs were prohibitively expensive. A new set of experiments are planned using a Graphical Processing Unit (GPU) which Theano exploits for much faster code-completion, which would in turn lead to a higher throughput of experiments. Our Configurable Deep Machine is designed for use with a GPU, hence allowing for a far higher number of tests.

ACKNOWLEDGMENT

This research is carried out under In-MINDD, which is funded by the European Union Seventh Framework Programme, Grant Agreement Number 304979

REFERENCES

[1] Deep learning with theano tutorial and software. Online: http://www.deeplearning.net/tutorial/ and git://github.com/lisa-lab/DeepLearningTutorials.git; Last accessed 06/02/2015.

[2] Health level 7 international website, 2014. Online; Last accessed 31/01/15.

[3] Chidanand Apte, Fred Damerau, and Sholom M Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems (TOIS)*, 12(3):233–251, 1994.

[4] Antonio Arauzo-Azofra, Jos Luis Aznarte, and Jos M. Bentez. Empirical study of feature selection methods based on individual feature evaluation for classification problems. *Expert Systems with Applications*, 38(7):8170 – 8177, 2011.

[5] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[6] Damir Bearevi and Mark Roantree. A metadata approach to multimedia database federations. *Information and Software Technology*, 46(3):195 – 207, 2004.

[7] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[8] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[9] Kay Deckers, Martin PJ Boxtel, Olga JG Schiepers, Marjolein Vugt, Juan Luis Muñoz Sánchez, Kaarin J Anstey, Carol Brayne, Jean-Francois Dartigues, Knut Engedal, Miia Kivipelto, et al. Target risk factors for dementia prevention: a systematic review and delphi consensus study on the evidence from observational studies. *International journal of geriatric psychiatry*, 2014.

[10] Shobeir Fakhraei, Hamid Soltanian-Zadeh, Farshad Fotouhi, and Kost Elisevich. Confidence in medical decision making: application in temporal lobe epilepsy data mining. In *Proceedings of the 2011 workshop on Data mining for medicine and healthcare*, pages 60–63. ACM, 2011.

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International conference on artificial intelligence and statistics*, pages 249–256, 2010.

[12] Geoffrey Hinton. A practical guide to training restricted boltzmann machines. *Momentum*, 9(1):926, 2010.

[13] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[14] Eric J Humphrey, Juan P Bello, and Yann LeCun. Feature learning and deep architectures: new directions for music informatics. *Journal of Intelligent Information Systems*, 41(3):461–481, 2013.

[15] van Boxtel M.P.J. Ponds R.H.W.M Jolles J., Houx P.J. *The Maastricht Ageing Study: Determinants of*. Maastricht: Neuropsych Publishers, 1995.

[16] Quoc V Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[17] Znaonui Liang, Gang Zhang, Jimmy Xiangji Huang, and Qmming Vivian Hu. Deep learning for healthcare decision making with emrs. In *Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on*, pages 556–559. IEEE, 2014.

[18] M. Prince, M. Guerchet, and M. Prina. Policy brief for heads of government: The global impact of dementia 20132050, adi, london., November 2013. Online; Last accessed 10/12/13.

[19] M. Roantree, P. Hickey, A. Crilly, J. Cardiff, and J. Murphy. Metadata modelling for healthcare applications in a federated database system. In Otto Spaniol, Claudia Linnhoff-Popien, and Bernd Meyer, editors, *Trends in Distributed Systems CORBA and Beyond*, volume 1161 of *Lecture Notes in Computer Science*, pages 71–83. Springer Berlin Heidelberg, 1996.

[20] Mark Roantree, Donall McCann, and Niall Moyna. Integrating sensor streams in phealth networks. In *Parallel and Distributed Systems, 2008. ICPADS'08. 14th IEEE International Conference on*, pages 320–327. IEEE, 2008.

[21] Mark Roantree, Jim ODonoghue, Noel OKelly, Martin van Boxtel, and Sebastian Kohler. Automating the integration of clinical studies into medical ontologies. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pages 2938–2947. IEEE, 2014.

[22] Mark Roantree, Jim ODonoghue, Noel OKelly, Maria Pierce, Kate Irving, Martin Van Boxtel, and Sebastian Köhler. Mapping longitudinal studies to risk factors in an ontology for dementia. *Health Informatics Journal*, pages 1–13, 2015.

[23] Mark Roantree, Jie Shi, Paolo Cappellari, Martin F. OConnor, Michael Whelan, and Niall Moyna. Data transformation and query management in personal health sensor networks. *Journal of Network and Computer Applications*, 35(4):1191 – 1202, 2012. Intelligent Algorithms for Data-Centric Sensor Networks.

[24] Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009.

[25] Nitish Srivastava and Ruslan Salakhutdinov. Multimodal learning with deep boltzmann machines. In *NIPS*, pages 2231–2239, 2012.

[26] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013.

[27] Antonio Jimeno Yepes, Andrew MacKinlay, Justin Bedo, Rahil Garnavi, and Qiang Chen. Deep belief networks and biomedical text categorisation. In *Australasian Language Technology Association Workshop 2014*, page 123.