Research on Deep Learning and Comparison with Naive Bayesian Analysis

Janki Kansara
Computer and Information Science
College of Science and Technology
Guided by: Prof. Pei Wang
Temple University
janki.kansara@temple.edu

***Abstract*— This paper is a research on predicting the probability of a dataset of S and P Stock Market , gathering it into the specific format and applying the Deep Learning Model using Tensorflow on that dataset to find the probability and accuracy of the same. For carrying out the same task using Naive Bayesian Analysis, dataset of the patients dealing with the diabetes have been taken which basically contains females over 21 years of age at Pima Indian Heritage. Many constraints have been added to the dataset in order to find the accuracy of the dataset used. Prior to testing the Deep Learning tensorflow model on this dataset, I developed a comparison dataset program to test the prediction power (which basically included the example of making the dataset of multiple days when a person goes for running or not and predicting the probability from that dataset if he would be going to run in the morning the next day or not). Comparison of Deep Learning and Naive Bayesian Analysis is leveraged as the end result.**

I. INTRODUCTION

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech.

On the other hand Naive Bayes Classifiers are highly scalable, they require a number of parameters linear in the number of variables(features/predictor) in a learning problem. Maximum-likelihood(a method of estimating the parameters of a statistical model given observations, by finding the parameter values that maximize the likelihood of making the observations given the parameters) can be done by evaluating a closed form expression, which takes linear time rather than by expensive iterative approximation as used for many other types of classifiers.

For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. In 2004, an analysis of the Bayesian classification problem showed that there are sound theoretical reasons for the apparently implausible efficacy of naive Bayes classifiers.Still, a comprehensive comparison with other classification algorithms in 2006 showed that Bayes classification is outperformed by other approaches, such as boosted trees or random forests. In this paper, we use the naive bayes classifier to find out the probability of the given dataset using the Naive Bayes Algorithm.

II. DEEP LEARNING MODEL AND TYPES

Playing around with data and using Deep Learning with Tensorflow gives a brief introduction about how a heavy dataset can be trained and tested within no time. Conventional machine-learning techniques were limited in their ability to process natural data in their raw form. Constructing a pattern-recognition or machine-learning system required careful engineering and domain expertise to design a feature extractor that transformed the raw data (such as the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input.

Deep Learning has been successful in discovering intricate structures in high-volume of data and therefore is applicable in many domain of science. In addition to beating records in image recognition and speech recognition, it has beaten other machine-learning techniques at predicting the activity of potential drug molecules, analysing particle accelerator data reconstructing brain circuits11, and predicting the effects of mutations in non-coding DNA on gene expression and disease.

A. *Summary of Supervised Learning for Deep Learning*

The most common form of machine learning, deep or not, is supervised learning. Imagine that we want to build a system that can classify images as containing, say, a house, a car, a person or a pet. We first collect a large data set of images of houses, cars, people and pets, each labelled with its category. During training, the machine is shown an image and produces an output in the form of a vector of scores, one for each category. We want the desired category to have the highest score of all categories, but this is unlikely to happen before training. We compute an objective function that measures the error (or distance) between the output scores and the desired pattern of scores. The machine then modifies its internal adjustable parameters to reduce this error. These adjustable parameters, often called weights, are real numbers that can be seen as knobs that define the inputoutput function of the machine. In a typical deep-learning system, there may be hundreds of millions of these adjustable weights, and hundreds of millions of labelled examples with which to train the machine.

In practice, most practitioners use a procedure called stochastic gradient descent (SGD). This consists of showing the input vector for a few examples, computing the outputs and the errors, computing the average gradient for those examples, and adjusting the weights accordingly. The process is repeated for many small sets of examples from the training set until the average of the objective function stops decreasing. It is called stochastic because each small set of examples gives a noisy estimate of the average gradient over all examples. This simple procedure usually finds a good set of weights surprisingly quickly when compared with far more elaborate optimization techniques After training, the performance of the system is measured on a different set of examples called a test set. This serves to test the generalization ability of the machine its ability to produce sensible answers on new inputs that it has never seen during training.

B. *Summary of Naive Bayes Classifier*

Naive Bayes Classifier is a conditional probabilistic model: given a problem instance to be classified, represented by a vector x=*x1,...,xn* representing some n features(independent variables) it assigns to this instance probabilities

$p(C_k \mid x_1, \dots, x_n)$ for each of K possible outcomes or classes C_k.

The problem with the above formulation is that if the number of features n is large or if a feature can take on a large number of values, then basing such a model on probability tables is infeasible. We therefore reformulate the model to make it more tractable. Using Bayes' theorem, the conditional probability can be decomposed as $p(C_k \mid \mathbf{x}) = \frac{p(C_k)\ p(\mathbf{x} \mid C_k)}{p(\mathbf{x})}$

Using Bayesian Probability, the posterior probability can be written as: $\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}$

where, we have a prior assumption that the probability distribution function is $p(x)$ and the likelihood function is probability of the evidence is given by the parameter: $p(\mathbf{x} \mid C_k)$

In practice, there is interest only in the numerator of that fraction, because the denominator does not depend on C and the values of the features x_i are given, so that the denominator is effectively constant. The numerator is equivalent to the joint probability model and can be rewritten as follows, using the chain rule for repeated applications of the definition of conditional probability:

$p(C_k, x_1, \dots, x_n)$

C. *Convolutional Neural Networks*

ConvNets are designed to process data that come in the form of multiple arrays, for example a colour image composed of three 2D arrays containing pixel intensities in the three colour channels. Many data modalities are in the form of multiple arrays: 1D for signals and sequences, including language; 2D for images or audio spectrograms; and 3D for video or volumetric images. There are four key ideas behind ConvNets that take advantage of the properties of natural signals: local connections, shared weights, pooling and the use of many layers.

The architecture of a typical ConvNet is structured as a series of stages. The first few stages are composed of two types of layers: convolutional layers and pooling layers. Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank. The result of this local weighted sum is then passed through a non-linearity such as a ReLU which is basically an activation function used as the positive part of the argument and the unit corresponding to this rectifier function is known as Rectified linear unit whose approximation is always analytical $f(x) = log(1 + exp(x))$. All units in a feature map share the same filter bank. Different feature maps in a layer use different filter banks.

III. Exploring Image Understanding using Deep Convolutional Neural Network

ConvNets have been applied with great success to the detection, segmentation and recognition of objects and regions in images. These were all tasks in which labelled data was relatively abundant, such as traffic sign recognition, the segmentation of biological images particularly for connectomics, and the detection of faces, text, pedestrians and human bodies in natural images. A major recent practical success of ConvNets is face recognition. Importantly, images can be labelled at the pixel level, which will have applications in technology, including autonomous mobile robots and self-driving cars.

When deep convolutional networks were applied to a data set of about a million images from the web that contained 1,000 different classes, they achieved spectacular results, almost halving the error rates of the best competing approaches. This success came from the efficient use of GPUs, ReLUs, a new regularization technique called dropout, and techniques to generate more training examples by deforming the existing ones Recent ConvNet architectures have 10 to

20 layers of ReLUs, hundreds of millions of weights, and billions of connections between units. Whereas training such large networks could have taken weeks only two years ago, progress in hardware, software and algorithm parallelization have reduced training times to a few hours. The performance of ConvNet-based vision systems has caused most major technology companies to initiate research and development projects and to deploy ConvNet-based image understanding products and services.

ConvNets are easily amenable to efficient hardware implementations in chips or field-programmable gate arrays. A number of companies are developing ConvNet chips to enable real-time vision applications in smart phones, cameras, robots and self-driving cars.

A. Distributed representations and language processing

Deep-learning theory shows that deep nets have two different advantages over classic learning algorithms that do not use distributed representations. Both of these advantages arise from the power of composition and depend on the underlying data-generating distribution having an appropriate componential structure.

- First, learning distributed representations enable generalization to new combinations of the values of learned features beyond those seen during training (for example, 2n combinations are possible with n binary features.
- Second, composing layers of representation in a deep net brings the potential for another exponential advantage (exponential in the depth).

The hidden layers of a multilayer neural network learn to represent the networks inputs in a way that makes it easy to predict the target outputs. This is nicely demonstrated by training a multilayer neural network to predict the next word in a sequence from a local context of earlier words. Each word in the context is presented to the network as a one-of-N vector, that is, one component has a value of 1 and the rest are 0. In the first layer, each word creates a different pattern of activations, or word vector In a language model, the other layers of the network learn to convert the input word vectors into an output word vector for the predicted next word, which can be used to predict the probability for any word in the vocabulary to appear as the next word. The network learns word vectors that contain many active components each of which can be interpreted as a separate feature of the word, as was first demonstrated in the context of learning distributed representations for symbols.

B. Recurrent Neural Networks

For tasks that involve sequential inputs, such as speech and language, it is often better to use RNNs. RNNs process an input sequence one element at a time, maintaining in their hidden units a state vector that implicitly contains information about the history of all the past elements of the sequence. When we consider the outputs of the hidden units at different discrete time steps as if they were the outputs of different neurons in a deep multilayer network it becomes clear how we can apply backpropagation to train RNNs.

RNNs are very powerful dynamic systems, but training them has proved to be problematic because the backpropagated gradients either grow or shrink at each time step, so over many time steps they typically explode or vanish. RNNs have been found to be very good at predicting the next character in the text or the next word in a sequence, but they can also be used for more complex tasks. For example, after reading an English sentence one word at a time, an English encoder network can be trained so that the final state vector of its hidden units is a good representation of the thought expressed by the sentence. This thought vector can then be used as the initial hidden state of (or as extra input to) a jointly trained French decoder network, which outputs a probability distribution for the first word of the French translation.

IV. EXPERIMENT

Data was exported from the already available dataset on github of the S&P Stock Market. The dataset contains n = 41266 minutes of data ranging from April to August 2017 on 500 stocks as well as the total S&P 500 index price. Index and stocks are arranged in wide format.

$\#Importdata\ data = pd.read_csv('data\_stocks.csv')$
$\#Dropdatevariable\ data = data.drop(['DATE'], 1)$
$\#Dimensionsofdataset\ n = data.shape[0]$
$p = data.shape[1]\ \#Makedataanumpyarray\ data = data.values$

The file that was used for extracting the information had already cleaned and prepared data, meaning missing stocks and index prices were LOCF's(last observation carried forward) so that the file did not contain any missing value. The dataset was split into training and test data. The training data contained 80% of the total dataset. The data was not shuffled but sequentially sliced.

A look at the Training and Testing code:

$\#Trainingandtestdata$
$train\_start = 0$
$train\_end = int(np.floor(0.8 * n))$
$test\_start = train\_end$
$test\_end = n$
$data\_train = data[np.arange(train\_start, train\_end), :]$
$data\_test = data[np.arange(test\_start, test\_end), :]$

A. Applying the Recurrent Neural Network on the model

After having defined the placeholders, variables, initializers, cost functions and optimizers of the network, the model needs to be trained. Usually, this is done by minibatch training. During minibatch training random data samples of $n = batch\_size$ are drawn from the training data and fed into the network. The training dataset gets divided into $n/batch\_size$ batches that are sequentially fed into the network. At this point the placeholders X and Y come into play. They store the input and target data and present them to the network as inputs and targets.

A sampled data batch of X flows through the network until it reaches the output layer. There, TensorFlow compares the models predictions against the actual observed targets Y in the current batch. Afterwards, TensorFlow conducts an optimization step and updates the networks parameters, corresponding to the selected learning scheme. After having updated the weights and biases, the next batch is sampled and the process repeats itself. The procedure continues until all batches have been presented to the network. One full sweep over all batches is called an epoch. The training of the network stops once the maximum number of epochs is reached or another stopping criterion defined by the user applies.

The results to the epochs were, after 10 epochs the dataset worked pretty well and adaptive to the model. The mean absolute percentage error of the forecast on the test set is equal to 5.31% which is pretty good. Note, that this is just a fit to the test data, no actual out of sample metrics in a real world scenario.

B. Calculating the Probability using Naive Bayes on a different dataset for comparison

I used an already generated data model which had dataset generated in it. To find out the extent to which Naive Bayes Classifier can predict the probability, I took the dataset of ”Pima Indians Diabetes Problem”. I used a dataset of about 200 patients out of the given 768 medical observations of Pima Indians Patent. The records describe instantaneous measurements taken from the patient such as their age, the number of times pregnant and blood workup. All patients are women aged 21 or older. All attributes are numeric, and their units vary from attribute to attribute.

Each record has a class value that indicates whether the patient suffered an onset of diabetes within 5 years of when the measurements were taken (1) or not (0). This is a standard dataset that has been studied a lot in machine learning literature. A good prediction accuracy is 70%-76%. Below is a sample from the pima-indians.data.csv file to get a sense of the data I worked with.

6,148,72,35,0,33.6,0.627,50,1
1,85,66,29,0,26.6,0.351,31,0
8,183,64,0,0,23.3,0.672,32,1
1,89,66,23,94,28.1,0.167,21,0
0,137,40,35,168,43.1,2.288,33,1

Table 2

The detailed description about this numbers is given in the table below:

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/m^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

Table 3

The purpose of using 200 dataset instead of 768 was to generate faster results on a whole. After getting the relevant dataset, we begin testing. The steps performed in predicting the probability are as follows:

C. Handling and Summarizing the Data

The first thing we need to do is load our data file. The data is in CSV format without a header line or any quotes. We can open the file with the open function and read the data lines using the reader function in the csv module.

We also need to convert the attributes that were loaded as strings into numbers that we can work with them. Below is the loadCsv() function for loading the Pima indians dataset. Next we need to split the data into a training dataset that Naive Bayes can use to make predictions and a test dataset that we can use to evaluate the accuracy of the model. We need to split the data set randomly into train and datasets with a ratio of 67% train and 33% test (which is a common ratio for testing an algorithm on a dataset).

The naive Bayes model is comprised of a summary of the data in the training dataset. This summary is then used when making predictions.

The summary of the training data collected involves the mean and the standard deviation for each attribute, by class value. For example, if there are three class values and 9 numerical attributes, then we need a mean and standard deviation for each attribute (9) and class value (3) combination, that is 27 attribute summaries.

These are required when making predictions to calculate the probability of specific attribute values belonging to each class value. The source from where I fetched the dataset used an easy method to summarize data by breaking down the preparation into following tasks:

- Separate Data By Class: The first task is to separate the training dataset instances by class value so that we can calculate statistics for each class. We can do that by creating a map of each class value to a list of instances that belong to that class and sort the entire dataset of instances into the appropriate lists.
- Calculate Mean and Standard Deviation:
- Summarize Dataset: For a given list of instances (for a class value) that has been obtained by performing the above function, we can calculate the mean and the standard deviation for each attribute.
  The zip function groups the values for each attribute across our data instances into their own lists so that we can compute the mean and standard deviation values for the attribute.
- Summarize Attributes By Class: We can pull it all together by first separating our training dataset into instances grouped by class. Then calculate the summaries for each attribute.

V. Predicting the Probability

Making predictions involves calculating the probability that a given data instance belongs to each class, then selecting the class with the largest probability as the prediction. As we are now ready with the summary of our dataset, generated from the following functions, we can use a Gaussian function to estimate the probability of a given attribute value, given the known mean and standard deviation for the attribute estimated from the training data.

Given that the attribute summaries where prepared for each attribute and class value, the result is the conditional probability of a given attribute value given a class value.

In summary known details are been pulled into the Gaussian (attribute value, mean and standard deviation) and reading off the likelihood that our attribute value belongs to the class. From the above methods, as we can predict the largest probability now,

The predictions can be compared to the class values in the test dataset and a classification accuracy can be calculated as an accuracy ratio between 0 & and 100%. The table showing the accuracy generated after calculating the probability on 200 instances is as follows:

Sr. No.	Dataset	Accuracy
1	15	0.6455
2	20	0.6455
3	50	0.6478
4	70	0.6343
5	120	0.6404
6	200	0.6545

Hence the above table shows that when I tried to test the dataset with different number of outcomes, the accuracy level varies for every outcome, depending on the number of iterations,performance of the computer,quality of the data etc. When the accuracy increases, it mainly corresponds to the performance of the machine and how it takes the data on a whole.

VI. COMPARISON OF DEEP LEARNING AND NAIVE BAYES ANALYSIS

Naive Bayes belongs to a category of models called generative. This means that during training (the procedure where the algorithm learns to classify), Naive Bayes tries to find out how the data (Pima Diabetes Dataset) was generated in the first place. It essentially tries to figure out the underlying distribution that produced the examples that are inputed to the model.

On the other hand Recurrent Neural Network is a discriminative model. It tries to figure out what the differences are between your positive and negative examples, in order to perform the classification. The deep learning models are hard to implement but the prediction is more accurate than that for the Naive Bayes Analysis.

VII. DIFFICULTIES FACED IN IMPLEMENTING DEEP LEARNING USING TENSORFLOW

This is by far the most important weakness and something which requires a little bit of extra effort. In the calculation of outcome probabilities using the tensorflow method, the implicit assumption is that all the attributes are mutually independent. This allows us to multiply the class conditional probabilities in order to compute the outcome probability.

When it is known beforehand that a few of the attributes are correlated , it is easy to ignore one of the correlated attributes. However, it is very hard to predict when we don't know which attribute is correlated with which other attribute in the dataset. That can lead to assuming the accuracy results without keeping in mind the conditions in which it is actually inter-related with another attribute.

When an attribute is continuous, computing the probabilities by the traditional method of frequency counts is not possible. In this case we would either need to convert the attribute to a discrete variable or use probability density functions to compute probability densities (not actual probabilities!). Most standard implementation automatically account for nominal and continuous attributes so the user does not need to worry about these transformations. However as a data scientist, it is important to be aware of the subtleties in the tool application.

The learning process of implementing Deep Learning using Tensorflow was a challenging one as it had lot of new topics that need to be covered. Developing a normal file of dataset of images (about 10) and testing and training the data using Tensorflow helped me get an idea about how Tensorflow works in finding the results of the dataset.

VIII. FUTURE SCOPE OF DEEP LEARNING

The release of TensorFlow was a landmark event in deep learning research. Its flexibility and performance allows researchers to develop all kinds of sophisticated neural network architectures as well as other ML algorithms. However, flexibility comes at the cost of longer time-to-model cycles compared to higher level APIs. Nonetheless, I feel that TensorFlow will make its way to the de-facto standard in neural network and deep learning development in research and practical applications. Many researchers and data scientists are already using TensorFlow or have started developing projects that employ TensorFlow models.

IX. CONCLUSION

This paper is the basic implementation to gain the knowledge of Deep Learning using Tensorflow on a set of Stock Market dataset to predict the fluctuating daily results in the market based on the given parameters. Keeping aside the drawbacks, Deep Learning is a good inference model for predicting the probabilities and finding out the possible outcome from the given dataset.

Deep Learning model using Tensorflow is a complex model to implement and use for predicting the probabilities but is important when it comes to finding out wide range of results for a huge set of data. It can also be used to find the usage of data in the next few years keeping in mind it's usage in the past years and using that as the dataset. There is still a wide scope of enhancement in the field of Deep Learning and one such is a need of a neat graphical user interface for designing and developing neural net architectures with TensorFlow backend.

Naive Bayes Classifier is a good inference model for predicting the probabilities and finding out the possible outcome from the given dataset. Though it is not as accurate as the Recurrent Neural Network, it is less complex and faster in implementation, thus providing a flexible way to get the prediction results. Naive Bayes Model is tremendously appealing to use because of its simplicity to implement, elegance and robustness. It is one of the oldest formal classification algorithms, and yet it is surprisingly effective being very easy. A large number of modifications have been introduced in this field using statistical, data mining, machine learning, and pattern recognition communities in an attempt to eliminate the possible drawbacks it has and to make it more flexible than before.

REFERENCES

[1] Krizhevsky, A., Sutskever, I. & Hinton, G. ImageNet classification with deep convolutional neural networks. In Proc. Advances in Neural Information Processing Systems 25 10901098 (2012).

[2] Farabet, C., Couprie, C., Najman, L. & LeCun, Y. Learning hierarchical features for scene labeling. IEEE Trans. Pattern Anal. Mach. Intell. 35, 19151929 (2013).

[3] Tompson, J., Jain, A., LeCun, Y. & Bregler, C. Joint training of a convolutional network and a graphical model for human pose estimation. In Proc. Advances in Neural Information Processing Systems 27 17991807 (2014).

[4] Szegedy, C. et al. Going deeper with convolutions. Preprint at http://arxiv.org/ abs/1409.4842 (2014).

[5] J. Hellerstein, Jayram Thathachar, and I. Rish. Recognizing end-user transactions in performance management. In Pro- ceedings of AAAI-2000, pages 596602, Austin, Texas, 2000.

[6] J. Hilden. Statistical diagnosis based on conditional independence does not require it. Comput. Biol. Med., 14(4):429435, 1984.

[7] R. Kohavi. Wrappers for performance enhancement and oblivious decision graphs. Technical report, PhD thesis, Department of Computer Science, Stanford, CA, 1995.

[8] H. Schneiderman and T. Kanade. A statistical method for 3d detection applied to faces and cars. In Proceedings of CVPR- 2000, 2000.

[9] Tom M. Mitchell. Machine Learning. McGraw-Hill, 1997.

[10] Ma, J., Sheridan, R. P., Liaw, A., Dahl, G. E. & Svetnik, V. Deep neural nets as a method for quantitative structure-activity relationships. J. Chem. Inf. Model. 55, 263274 (2015).

[11] Helmstaedter, M. et al. Connectomic reconstruction of the inner plexiform layer in the mouse retina. Nature 500, 168174 (2013).

[12] Kaggle. Higgs boson machine learning challenge. Kaggle https://www.kaggle. com/c/higgs-boson (2014).

[13] Leung, M. K., Xiong, H. Y., Lee, L. J. & Frey, B. J. Deep learning of the tissueregulated splicing code. Bioinformatics 30, i121i129 (2014).

[14] Xiong, H. Y. et al. The human splicing code reveals new insights into the genetic determinants of disease. Science 347, 6218 (2015)

[15] Jones, K. S. (1972). A statistical interpretation of term specificity and its application in retrieval. Journal of Documentation, 28, 1121.

[16] Hubel, D. H. & Wiesel, T. N. Receptive fields, binocular interaction, and functional architecture in the cats visual cortex. J. Physiol. 160, 106154 (1962)

[17] Jean, S., Cho, K., Memisevic, R. & Bengio, Y. On using very large target vocabulary for neural machine translation. In Proc. ACL-IJCNLP http://arxiv.org/ abs/1412.2007 (2015)

[18] Bottou, L. & Bousquet, O. The tradeoffs of large scale learning. In Proc. Advances in Neural Information Processing Systems 20 161168 (2007).

[19] Duda, R. O. & Hart, P. E. Pattern Classification and Scene Analysis

[20] Sutskever, I. Vinyals, O. & Le. Q. V. Sequence to sequence learning with neural networks. In Proc. Advances in Neural Information Processing Systems 27 31043112 (2014).

[21] Rosenblatt, F. The Perceptron A Perceiving and Recognizing Automaton. Tech. Rep. 85-460-1 (Cornell Aeronautical Laboratory, 1957).

[22] Parker, D. B. Learning Logic Report TR47 (MIT Press, 1985).

[23] Friedman, N. (1998). The Bayesian structural EM algorithm. Proc. UAI-98 (pp. 129138).

[24] Bengio, Y., Lamblin, P., Popovici, D. & Larochelle, H. Greedy layer-wise training of deep networks. In Proc. Advances in Neural Information Processing Systems 19 153160 (2006).