Journal of Machine Learning Research 21 (2019) 1-10 Submitted 11/19; Published 12/19

Using a Stacked Auto-Encoder to Improve Classification and Regression Performance of Multilayer Perceptron Networks

Brandon Sladek BRANDONSLADEK@GMAIL.COM
School of Computing
Montana State University
Bozeman, MT 59718, USA

Jared Thompson J.A.THOMPSON22@GMAIL.COM
School of Computing
Montana State University
Bozeman, MT 59718, USA

Kyle Hagerman HAGERMANKYLE96@GMAIL.COM
School of Computing
Montana State University
Bozeman, MT 59718, USA

Ryan Hansen RYANHANSEN2222@GMAIL.COM
School of Computing
Montana State University
Bozeman, MT 59718, USA

Abstract

Multilayer perceptrons (MLP) are a very popular type of neural network both historically and in contemporary research and applications. An auto-encoder (AE) is another form of neural network that is capable of learning in an unsupervised manner in contrast to the supervised learning nature of MLP with backpropagation. Backpropagation is a common way to train MLPs, and can also be used for training AEs as well. Multiple auto-encoders can be stacked together, creating a "stacked auto-encoder" (SAE). Finally, an MLP network can be placed on top of the SAE in order to make classification/regression predictions. This combination of unsupervised and supervised learning over multiple auto-encoder and hidden MLP layers results in a "deep" network capable of recognizing underlying structure in the data and using it to the advantage of the prediction MLP layer. In this paper we explore the classification/regression potential of an SAE with 1, 2, and 3 auto-encoder layers using 6 different data sets from the UCI machine learning repository (UCI (2019)). Specifically, we are investigating the potential classification/regression performance improvement that can be gained by pre-training an SAE network as input to a standard MLP network.

Keywords: Stacked Auto-Encoder, Multilayer Perceptron, Cost Function Topology

1. Introduction

Artificial neural networks have quickly become one of the most popular machine learning strategies in industry today. Their ability to learn complex decision boundaries for a wide variety of problems has led to the successful application of neural networks in fields as diverse as medical imaging and financial forecasting. Additionally, the substantial increase

in available computing power over the last decade has further lead to a resurgence of interest in neural networks within the research community. This research explores the potential benefits of using a stacked auto-encoder (SAE) to increase the performance of a multilayer perceptron (MLP) in terms of classification accuracy and regression error.

Stacked auto-encoders are a special type of neural network in which the input layer is the same size as the output layer. In the context of a stacked auto-encoder, the goal of the network is to learn a compressed (encoded) representation of the input vector that can then be decoded to form the output layer in order to recover the original input vector. In other words, we want the network to learn a configuration of weights and biases such that the output matches the input as closely as possible, while forming a "bottleneck" by compressing the input in the middle layers. In real world applications, stacked auto-encoders are most often used as a dimensionality reduction technique used to form a more compact input vector that preserves the most important features of the original input vector (Vincent et al. (2010)). In this way, stacked auto-encoders can be said to "learn" the structure of the data, which can then be used as a reduced data set in other applications.

2. Problem

In the greatly influential research paper "Multilayer Feedforward Networks are Universal Approximators" published in 1989, the authors rigorously established that standard multilayer feedforward neural networks, with arbitrary activation ("squashing") functions, can approximate essentially any function to any degree of accuracy, provided the network is large enough (Hornik et al. (1989)). Furthermore, the authors claim that "failures" in the networks can be attributed to poor training results, poor choices for hyperparameter values, or simply a lack of a deterministic relationship between the input and the expected output. In other words, theoretically, a neural network can be built and trained to any degree of accuracy, provided there is an actual relationship between the input and expected output, and the network is built in such a way that it can adequately learn the features of the input space and map them to an accurate output space with respect to the expected output.

The most popular training algorithm for MLP neural networks is backpropagation with stochastic gradient descent (SGD). It has been shown that an MLP neural network is sensitive to its initial random configuration of weights and biases during training. Due to the nature of gradient descent, the initial parameter configuration of the network determines the initial basin of attraction in the topology of the cost function. Although using mechanisms like momentum can help the algorithm jump into deeper basins, the initial basin of attraction has a significant impact on determining where the solution will move until convergence, namely the center of the initial basin. Furthermore, the first few iterations of gradient descent have the largest impact on determining which basin of attraction the final solution will settle in. Subsequent iterations have a lesser and lesser impact on determining this basin, meaning the starting point and early iterations make the most impact on determining the best final accuracy/error results that can be obtained with that particular run of SGD. Therefore, if those beginning iterations of gradient descent can be improved or made more efficient by placing the solution in a region of the cost function topology that is relatively easy to navigate, then perhaps we can expect to find better results overall.

Stacked auto-encoders are typically used as a dimensionality reduction technique, or as a way to de-noise noisy data during the preprocessing phase of a more complex pipeline (Vincent et al. (2010)). By tasking a stacked auto-encoder with reproducing the input as best as it can while creating a bottleneck in the middle layer, the SAE is essentially finding a way to "compress" the input data into a smaller form (less neurons), while still being able to reliably reproduce the data as output from the network. It is generally thought that a SAE "learns" the dominant features in the data, and thus the overall structure of the data it has been given so far. The output of the SAE can then be fed into a regular MLP in order to make classification/regression predictions. Researchers have shown that passing the input data into a SAE before feeding it into a MLP can have a positive impact on results by "priming" the input data to be more easily learned and give the SGD process a better chance of finding a deeper basin of attraction and thus a better solution.

We seek to verify this claim by running various stacked auto-encoder implementations on various data sets using various parameter values. With repeated training/testing of the networks using different parameter values, we will be able to gain insight into which parameter values work best for each data set and network configuration. Furthermore, we seek to compare our stacked auto-encoder results with our results using just MLP networks in Project 3, to determine whether using a stacked auto-encoder as input to an MLP network can improve performance results.

3. Networks

In this section we outline the fundamental mathematical concept behind feedforward neural networks. Additionally, we note our specific design choices for implementation (e.g. activation functions).

3.1 Feedforward Neural Networks

Overview - The core idea behind a neural network is to generate any arbitrarily complex function by simulating the way neurons are thought to transfer and potentially "remember" information. Given some input, a series of neurons in the first (input) layer activate. The activations in the next layer are calculated as the weighted sum of the activations in the preceding layer, plus a bias vector for the neurons in the next layer, eventually leading to activations of the output neurons in the last (output) layer, which represent a meaningful output in the context of the problem. Neurons are related to each other via weights, and outputs are normalized via activation functions such as the Sigmoid or ReLU functions.

Formalism - We can represent a feedforward neural network, N, as a collection of layers forming a directed acyclic graph, $N = \{L_i\}$. Each layer L_i, is composed of a collection of n_i nodes $L_i = \{p_1, p_2, ..., p_{n_i}\}$. Each node $p_j \in L_i$ for $p'_k \in L_{i-1}$ has an *activation value*

$$a(p_j) = f(\beta_j + \Sigma_{k=0}^{n_{i-1}} w_k a(p'_k)) \tag{1}$$

The activation value of each node is given by the activation value of the previous layer, scaled by some weight, shifted by some bias number, and finally normalized through some normalization function. Notice, then we can generalize the feedforward process as a series

of subsequent matrix multiplications.

$$\mathbb{A}_i = f(\beta_i + \mathbb{W}_{i-1,i} \cdot \mathbb{A}_{i-1}.) \tag{2}$$

Where $\mathbb{A}_i$ is the activation vector of layer i, β_i is the shifting bias vector, and $\mathbb{W}_{i-1,i}$ is the weight matrix mapping from layer $i-1$ to layer i. We call f the *activation function* and β is the bias term. For classification data, each output node represents a class. Regression includes only one output node - the class value of the regression data point.

Training - The network is initialized with arbitrary weights and biases, along with predetermined activation functions for each layer. The network is trained through supervised learning. Given a training data point x, with desired output value Y, there will be a subsequent output activation in the output nodes $N(x)$. To train the network, we take the squared error, $Err = (Y - N(x))^2$, and attempt to minimize it through a gradient descent learning algorithm referred to as backpropagation. To do this, we optimize ∇Err, where we are taking derivatives with respect to weights. The goal is to learn the weights and biases that minimize the squared error. Subsequently, we update each weight by

$$\mathbb{W}_{new} = \mathbb{W}_{old} - \alpha \nabla Err \tag{3}$$

where α is a tuned learning rate. We propagate backwards through each layer, until each weight in our network has been updated.

3.2 Multilayer Perceptron (MLP)

This is a special case of a feed forward neural network. In this network, we get to choose an arbitrary number of layers, the number of nodes in each layer, and the activation function. The entire network is trained with gradient descent using backpropagation.

Overview - The MLP is perhaps one of the most general implementations of a feed forward network. The idea is to stack an arbitrary number of layers of nodes to try to capture patterns in data.

Formalism - The main theory behind the MLP is encompassed in its activation functions. Common activation functions intend to normalize the activation of each node between 0 and 1, examples being the Sigmoid and ReLU functions. This means a low activation should have negligible impacts on the output, and high activation should have larger effects, as desired (M.W Gardner (1998)).

Our Implementation - We implemented the option to use either the Sigmoid or ReLU activation functions, but ran most experiments with the Sigmoid function. We then varied the hyperparameters, particularly the number of layers and number of neurons in each layer, in order to find good results with respect to our loss functions.

3.3 Stacked Auto-Encoder (SAE)

An auto-encoder (AE) is a specific type of feedforward neural network. auto-encoder neural networks have the same number of neurons in their output layer as their input layer, since their primary objective is to reproduce the input using a non-trivial network architecture that typically has a bottleneck in the middle layer. An AE is broken into three pieces. A series of encoding layers and an equal in number series of decoding layers are separated by

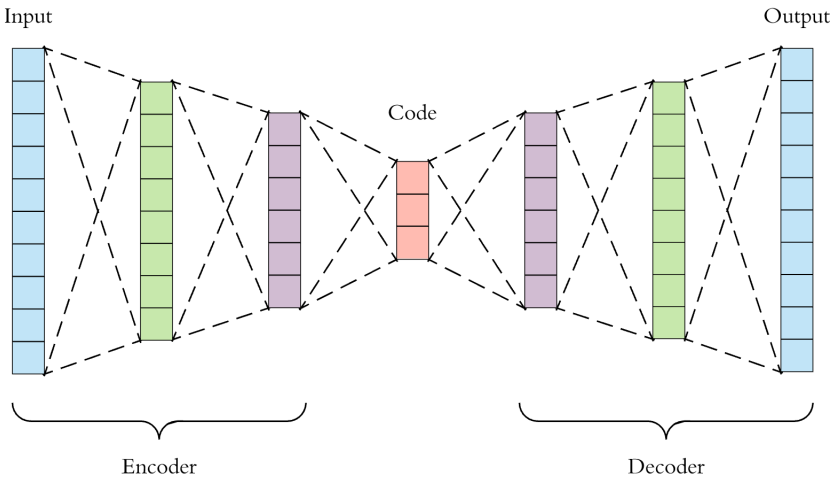

Figure 1: An example of an auto-encoder with three layers of encoding/decoding.

a "code" layer that is the smallest layer in the network in terms of the number of neurons. Each succeeding encoding layer will have less nodes than the previous and its respective decoding layer will have the same number of nodes. A common model of an AE can be found in Figure 1.

auto-encoders use a method of learning known as "unsupervised pre-training." Unsupervised pre-training is a method for identifying underlying structure in a data set. Supervised learning will directly read some classifying categorical label or regression value to determine error. In contrast, unsupervised learning attempts to generate the same data point that was given as input, and therefore will construct a series of weights and biases specific to the data set without reading any labels.

A stacked auto-encoder neural network is just multiple auto-encoders "stacked" on top of one another. In this context, we will be pre-training an auto-encoder, and then placing a multilayer perceptron network on top in order to do classification and regression with the specified data sets. This stacking of networks will result in a "deep" network, and will attempt to build a relative structure of the data before entering an MLP. While the SAE trains, we are actually finding a better starting point for gradient descent in the MLP. This means that our actual MLP will have better results and/or take less time to train if we stack it on top of the pre-trained SAE.

The "code" layer size (number of nodes) will be determined through the hyperparameter tuning process. Since the typical use case for auto-encoders is dimensionality reduction, we will seek the smallest code layer size possible while still preserving the auto-encoder's ability to recover the input and classify using the code layer as a compressed feature vector that improves the ability of gradient descent to find an acceptable basin of attraction.

4. Experiment

In this section we explain our general approach for accepting or rejecting our hypothesis for the various data sets, network configurations, and corresponding parameter values.

4.1 Approach

Our goal was to compare the effectiveness of prepending a SAE to an MLP network in order to "prime" the gradient descent process for finding better solutions. Across 6 constant data sets, we tested the SAE networks with 1, 2, and 3 auto-encoder layers, along with MLP networks with 0, 1, and 2 hidden layers, with variable numbers of nodes in each layer.

Using Python and the Pandas library (McKinney (2010)), we opted to run 10-fold class validation on each of the data sets. Cross validation helps to ensure results are not particular to a specific training/test set. In probabilistic experiments, it is possible to get good or bad results due to random chance. Cross validation helps mitigate this problem by running the same process (in this case the algorithms) on various samples of the data (Kohavi (1995)). Finally, to ensure that each column attribute in our feature vector held equal weight, we normalized the column attributes to be values between 0 and 1.

Finally, we specify our loss functions for comparing the algorithms. We are interested in two components of the algorithms - some form of accuracy and running time. *Accuracy*: Accuracy refers to how correct the network predicts outputs. For classification data, we use accuracy (0-1 loss) and for regression data we use mean squared error. *Convergence Time*: Convergence time refers to how long the program took to train and execute. We compare these times directly.

4.2 Hypothesis

Our hypothesis was a ranking based on the performance of each network on each of the six data sets. We outline our ranking and justification in the table below.

Algorithm	Accuracy Rank	Convergence Time Rank
SAE 1 Hidden Layer	1 - The data is 'compressed' less. We are doing less generalization	1 - Least HL
SAE 2 Hidden Layer	2 - More compressed than 1, less compressed than 3	2 - Medium HL
SAE 3 Hidden Layer	3 - Most compressesd. Patterns in 3rd HL are least fitted to data.	3 - Most HL

Figure 2: Table depicting our performance hypothesis. Justification is included.

4.3 Preprocessing

Some of the data sets had missing values. Since the number of missing values in each data set was relatively small, we chose to simply remove all the missing values from every data set. We also normalized all of the data so that each attribute would contribute equally to the calculations, and no single attribute would dominate. Finally, since we used the stacked auto-encoder as a way to "pre-train" the network before feeding the output into the MLP, we should note that the SAE was used as a form of data/network pre-processing.

4.4 Tuning

Due to the prohibitive running times of the learning algorithms used in this research, the tuning process was mostly done by trial and error. In order to cut down on redundant processing time, we modified the Project 3 ExperimentRunner script to add the ability to save a pre-trained MLP network instance by serializing it to a persisted file in the experiment_runner/logs/ directory of the code. When running the SAE ExperimentRunner and appending an MLP network, this modification gave us the ability to deserialize and append a previously trained MLP network in much less time than it would have taken to re-train a newly created MLP.

When deciding which MLP networks were worth saving, we mostly drew from our observations of the results in Project 3, namely that simpler network architectures (0 hidden layers) typically gave us the best results, with the Car data set being the only exception. Once we had a few saved network instances for each data set, we were able to focus the rest of our time on developing/testing the SAE code and testing the effect of varying SAE-specific hyperparameters such as the number and size of the auto-encoder layers. Ideally, if we had more time we would do further experiments with both the SAE and MLP network configurations, as well as the full SAE+MLP network architecture.

4.5 Results

As mentioned in previous sections, our performance analysis is twofold. First, we gauge a form of correctness through 0-1 loss (accuracy) based on results from classification tests or mean squared error (MSE) for regression. A higher percentage correct or lower error indicates a better performance. Second, we test convergence rate in terms of running time. A higher running time is a worse convergence rate, and we will compare times directly. Figure 3 below shows results using a SAE and MLP combined network for this research. Figure 4 shows base line results using just an MLP network from Project 3.

Segmentation	Car	Abalone	Machine	Forest Fires	Wine
(20, 4, 2) [19, 14, 19, 19, 7] accuracy: 55.7% time: 31m36s	(20, 4, 2) [6, 4, 6, 6, 4] accuracy: 71.6% time: 73m19s	N/A	(20, 4, 2) [9, 6, 9, 9, 3, 1] error: 0.020026 time: 12m36s	(20, 4, 2) [12, 9, 12, 12, 6, 1] error: 0.0034639 time: 43m33s	(20, 4, 2) [11, 8, 11, 11, 1] error: 0.23981 time: 47m7s (1 CV iteration)

Figure 3: Table displaying SAE+MLP performance results. A higher accuracy indicates better performance for the classification data sets, and a lower error indicates better performance for the regression data sets.

Segmentation	Car	Abalone	Machine	Forest Fires	Wine
(50, 10, 5) [19, 7] accuracy: 83.3% time: 16m10s	(10, 10, 3) [6, 4, 2, 4] accuracy: 84.9% time: 21m41s	N/A	(20, 4, 5) [9, 1] error: 0.004 time: 2m5s	(10, 20, 3) [12, 6, 1] error: 0.0035 time: 5m49s	N/A

Figure 4: Table displaying MLP performance results from Project 3. We did not obtain valid results for the "Abalone" or "Wine" data sets.

5. Analysis

The long running times required to train these networks made it difficult to do meaningful comparisons between networks and parameter/network configurations. However, we still have interesting results that lead to a few insights.

In general, we found that simpler network configurations led to the best results. It seems like adding layers only resulted in decreasing performance. Decreasing performance with respect to increasing layer complexity likely is expected in this case. If there is no particular structure in the data that could potentially be picked up by auto-encoder pre-training or additional hidden layers, then doing pre-training or adding hidden layers will likely only serve to complicate the network's cost function, consequently making it more difficult to train, and more difficult to achieve satisfactory performance. Although pre-training and/or adding hidden layers can, in some cases, lead to better results, if the relevant underlying structure is missing from the data, better accuracy/error is not likely to be achieved by such methods (Hornik et al. (1989)).

One thing we noticed while testing and running the experiments is that the performance results are sensitive to the initial random configuration of weights and biases. Each cross validation partition would start gradient descent with a different initial accuracy/error value, and the starting point often determined how much the results could improve over the gradient descent iterations for that partition. This is a known obstacle that hinders all neural networks trained with gradient descent. As discussed in Section 2 above, the initial location of the parameter vector in the overall topological space of the cost function plays a significant role in determining which local minimum is reached (Erhan et al. (2010)). Momentum can help alleviate this issue, but in many cases, the final results are dependent on the initial starting point for gradient descent.

In theory and practice, stacked auto-encoders can help improve MLP performance results by pre-training and "priming" the gradient descent process to find a deeper basin of attraction. Unfortunately, we did not see such an improvement by using stacked auto-encoders for MLP prediction in this research. We believe our results are mostly due to not using the best hyperparameter values for each data set. If we had more time to test different hyperparameter configurations to see which combinations worked best, we are confident we could find better results than we found in Project 3.

6. Conclusion

This research involved a significant amount of programming, along with a fair amount of reasoning about the data sets and how best to approach and analyze each one. Ideally, we would have been able to test many parameter combinations for each test in an attempt to maximize classification accuracy and minimize regression error. However, as running times were prohibitively long, generating meaningful results proved difficult within the allotted time frame. With that said, we did manage to get decent results on the Segmentation data set using 1 auto-encoder layer with 14 neurons and 0 hidden layers in the MLP network, and almost equivalent results (with respect to Project 3) on the Car data set using 1 auto-encoder layer with 4 neurons and 0 hidden layers.

Our main takeaway from this work is that each data set needs to be analyzed and tuned independently, using a wide range of parameter values and network configurations, in order to gain insight into which configurations achieve the best performance results, both in terms of prediction accuracy and training time.

References

Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Ron Kohavi. A study of cross validation and bootstrap for accuracy estimation and model selection. *International Joint Conference on Artificial Intelligence*, 1995.

Wes McKinney. Data structures for statistical computing in python, 2010. URL `https://pandas.pydata.org/`.

S.R Dorling M.W Gardner. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(Aug):2627–2636, 1998.

UCI. UCI machine learning repository, 2019. URL `https://archive.ics.uci.edu/ml/datasets/`.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.