Automated Systems & Technologies
14-15 June 2017 • St. Petersburg, Russia

COMPARATIVE ANALYSIS OF APPLYING DEEP-LEARNING ON PID PROCESS

Farzin M. Khortabi, Maaz Ahmed Khan, Vyacheslav V. Potekhin

Peter the Great St. Petersburg Polytechnic University, Institute of Computer Sciences and Technologies
195251, St. Petersburg, Russia
f_b258@yahoo.com, maazkhanswati@gmail.com, slava.potekhin@gmail.com

Abstract

The deep learning approach to machine learning emphasizes high-capacity, scalable models that learn distributed representations of their input. This paper addresses the use of deep learning algorithm to design the controller; to explore the feasibility of applying deep learning into control problems. The proposed deep learning controller is designed by learning discrete and continues PID controller which is most commonly used in industry. The input/output of the PID controller are used as the learning data set for the deep learning network. SAE (Stacked Autoencoder) algorithm is used to design the deep learning controller. We develop an extension of sparse autoencoder by incorporating a stacking 2 autoencoders and softmax layer for hidden units. Unlike the standard autoencoder, our model uses the backpropagation algorithm to fine-tuning which leads to promote category-dependent sharing of learned features, which tends to improve the generalization performance. The simulation is performed using MatLab/Simulink and the detailed results of a comparison study between the proposed deep learning controller and a PID controller was conducted. We also analyzed other neural network based control system in comparison to our proposed model. It is demonstrated that neural network based controllers can perform even better than conventional controllers.

INTRODUCTION

Studying machine learning has scientific value both as a way to understand computation and as a way to understand learning, but for science to matter it must have a positive impact on the world. By always maintaining a connection to important practical problems, one increases the chance that their research on machine learning will have such a positive impact. One great property of the field of machine learning is that its methods can help us solve many specific problems of practical and commercial interest. However, if as researchers our only concern is making scientific progress, should we still labor over specific applications of our more general methods? Obviously we need to try new methods on actual problems to make sure that they work, but perhaps we can start with a new method and then look for problems it can attack.

Alternatively, we can start with a problem and then do whatever it takes to solve it; sometimes solving a problem will require new methods and sometimes not, but in either case good research will help us learn the limitations and advantages of existing methods as well as what aspects of the problem are important. Our goal is to test the limits of models based on Stacked Autoencoder on problems in these areas, to determine how well successful recipes generalize across problems, and to extend the methods as needed to make them more effective by introducing new techniques where appropriate. In particular, We focus much of our attention on applying deep model to problem.

The machine learning algorithms can lead to significant advances in automatic control. The biggest single advance occurred nearly four decades ago with the introduction of the Expectation-Maximization (EM) algorithm for training Hidden Markov Models (HMMs) [1]. With the EM algorithm, it became possible to develop control systems for real world tasks using the richness of Gaussian mixture models (GMM) [2] to represent the relationship between HMM states and the reference input. GMMs have a number of advantages that make them suitable for modeling the probability distributions over vectors of input features that are associated with each state of an HMM [3]. Despite all their advantages, GMMs have a serious short coming – they are statistically inefficient for modeling data that lie on or near a non-linear manifold in the data space [3].

Recently deep learning has been attracting a significant attention from the wide range of applications. Compare to the conventional neural networks, the key features of deep learning are to have more hidden layers and neurons, and to improve learning performance. Using these features, large and complex problems that could not be solved with conventional neural networks can be resolved by deep learning algorithms. Consequently, deep learning has been applied to various applications including pattern recognition and classification problems; for example, speech recognition [3], handwritten digit recognition [4], human action recognition [5], and so on. However, to the best knowledge of the authors, a few results has been published in the automatic control field. Thus, this work focuses on presenting the utilizing possibility of deep learning in control areas. This study was designed to mimic the PID controller using a AE algorithm. The simulation is performed using MatLab/Simulink and the detailed results of a comparison study between the

proposed deep learning controller and a PID controller was conducted to demonstrate the performance and effectiveness of the proposed algorithm.

Limitations of PID control

While PID controllers are applicable to many control problems, and often perform satisfactorily without any improvements or only coarse tuning, they can perform poorly in some applications, and do not in general provide optimal control. The fundamental difficulty with PID control is that it is a feedback control system, with constant parameters, and no direct knowledge of the process, and thus overall performance is reactive and a compromise. While PID control is the best controller in an observer without a model of the process [6], better performance can be obtained by overtly modeling the actor of the process without resorting to an observer.

THEORETICAL BACKGROUND

Autoencoder

Supervised learning is one of the most powerful tools of AI, and has led to automatic zip code recognition, speech recognition, self-driving cars, and a continually improving understanding of the human genome. Despite its significant successes, supervised learning today is still severely limited. Specifically, most applications of it still require that we manually specify the input features *x* given to the algorithm. Once a good feature representation is given, a supervised learning algorithm can do well.

An autoencoder, autoassociator or Diabolo network [7] is an artificial neural network used for unsupervised learning of efficient coding [8][9]. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. Recently, the autoencoder concept has become more widely used for learning generative models of data [10][11].

Autoencoder Structure

Architecturally, the simplest form of an autoencoder is a feedforward, non-recurrent neural network very similar to the multilayer perceptron (MLP) – having an input layer, an output layer and one or more hidden layers connecting them –, but with the output layer having the same number of nodes as the input layer, and with the purpose of *reconstructing* its own inputs (instead of predicting the target value *Y* given inputs *X*). Therefore, autoencoders are unsupervised learning models.

An autoencoder always consists of two parts, the encoder and the decoder, which can be defined as transitions Φ and ψ such that:

$$\Phi : \mathcal{X} \rightarrow \mathcal{F}$$
$$\Psi : \mathcal{F} \rightarrow \mathcal{X}$$
$$\Phi, \Psi = arg \min_{\Phi,\Psi} \|X - (\Phi \circ \Psi)X\|^2$$

In the simplest case, where there is one hidden layer, the encoder stage of an autoencoder takes the input x $\in \mathbb{R}^d = \mathcal{X}$ and maps it to $\mathbf{z} \in \mathbb{R}^p = \mathcal{F}$:

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \quad (1)$$

This **z** is usually referred to as code, latent variables, or latent representation. Here, σ is an element-wise activation function such as a sigmoid function or a rectified linear unit. **W** is a weight matrix and **b** is a bias vector. After that, the decoder stage of the autoencoder maps **z** to the *reconstruction* $\mathbf{x}'$ of the same shape as **x**:

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}') \quad (2)$$

where σ', $\mathbf{W}'$, and $\mathbf{b}'$ for the decoder may differ in general from the corresponding σ, **W** and **b** for the encoder, depending on the design of the autoencoder.

Autoencoders are also trained to minimize reconstruction errors (such as squared errors):

$$\mathcal{L}(\mathrm{x}, \mathrm{x}') = \|\mathrm{x} - \mathrm{x}'\|^2 = \left\|\mathrm{x} - \sigma'\left(\mathbf{W}'\left(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})\right)\right) + \mathrm{b}'\right\|^2 \quad (3)$$

where **x** is usually averaged over some input training set.

If the feature space $\mathcal{F}$ has lower dimensionality than the input space $\mathcal{X}$, then the feature vector $\phi(x)$ can be regarded as a compressed representation of the input x. If the hidden layers are larger than the input layer, an autoencoder can potentially learn the identity function and become useless. However, experimental results have shown that autoencoders might still learn useful features in these cases [7].

Sparse autoencoder

By imposing sparsity on the hidden units during training (whilst having a larger number of hidden units than inputs), an autoencoder can learn useful structures in the input data. This allows sparse representations of inputs. These are useful in pretraining for classification tasks. Figure illustrate schematic of sparse autoencoder. Sparsity may be achieved by additional terms in the loss function during training (by comparing the probability distribution of the hidden unit activations with some low desired value), or by manually zeroing all but the few strongest hidden unit activations (referred to as a k-sparse autoencoder) [12].

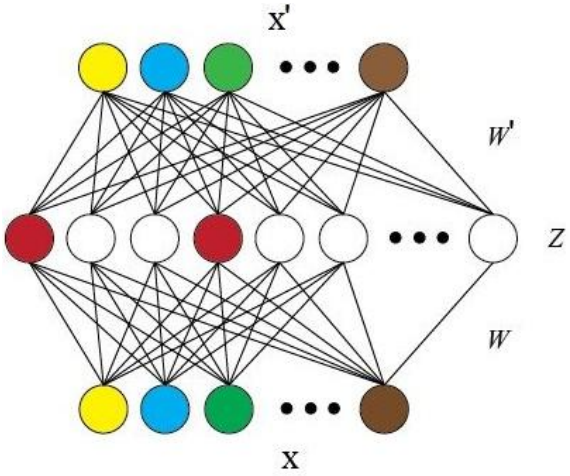


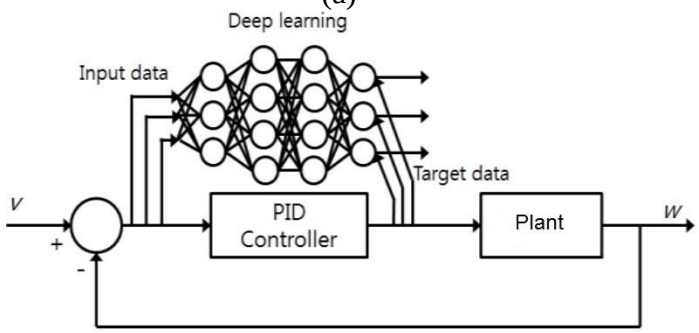
Figure 1: A diagram of a sparse autoencoder network. The input vector x is converted to a sparse representation on the hidden layer as z and then reconstructed as $\boldsymbol{x}'$.

METHODOLOGY

Deep Learning Controller

To design the deep learning controller, a PID controller was first performed and checked, Fig. 2-a; and then by using the performed PID's input/output information as the input/target data of the learning algorithm, respectively, the deep learning controller was tuned to be capable of replacing the original PID controller, Fig. 2-b.

(a)



(b)

Figure 2: Design of deep learning controller (a) block model of PID conreoller, (b) deep learning controller

The considered deep learning algorithm is based on Stacked Autoencoder, the greedy layer wise approach for pretraining a deep network works by training each layer in turn. In this section, we will find out how autoencoders can be "stacked" in a greedy layer wise fashion for pretraining (initializing) the weights of a deep network.

PID Control System Models

To check the validity of our proposed deep learning controller, we have used two different plants for Continuous and discrete plants, respectively. We chose DC motor System and Water tank problem to verify our results for continuous and discrete PID controlled model systems. Following are the Simulink® models.

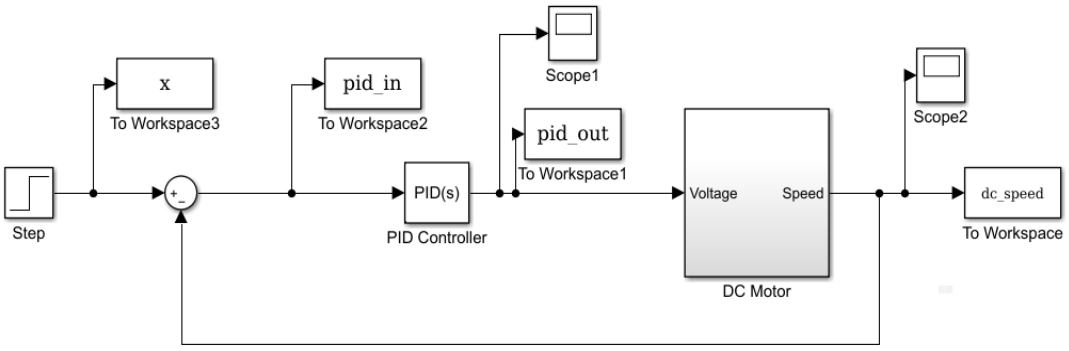


Figure 3: Simulink Model for PID controlled DC Motor

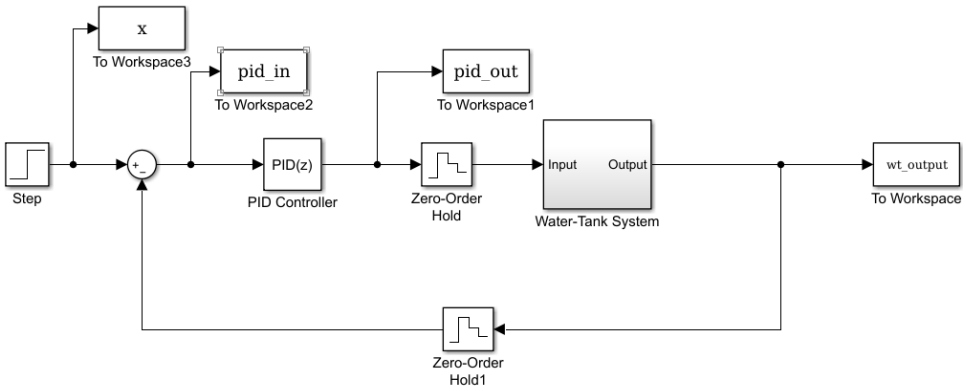
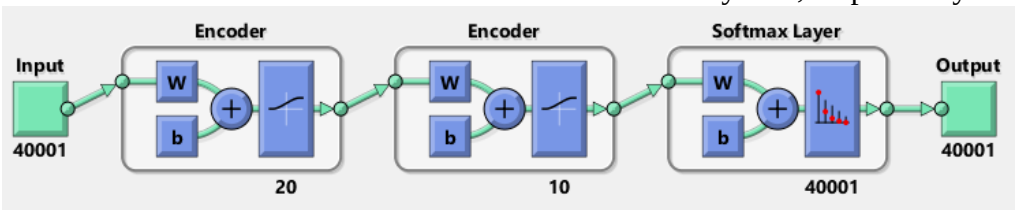


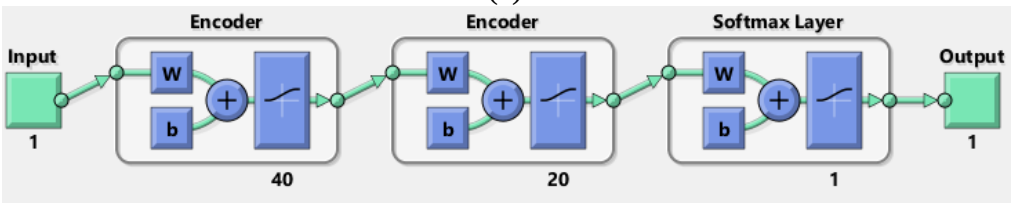
Figure 4: Simulink® of watertank PID

Training and Stacking of the Autoencoders

As described in earlier, an autoencoder is a neural network which attempts to replicate its input at its output. Thus, the size of its input will be the same as the size of its output. When the number of neurons in the hidden layer is less than the size of the input, the autoencoder learns a compressed representation of the input. For continuous model system we will try to make our controller learn the compressed representation of the input. For the discrete part we will analyze our system by teaching our model on the other way around. Figure 5 demonstrate the stacked autoencoder model for the continuous and discrete control system, respectively.



(a)



(b)

Figure 05: Stacked Autoencoders for (a) continuous and for (b) discrete models

Fine-tuning Stacked AEs

Fine tuning is a strategy that is commonly found in deep learning. As such, it can also be used to greatly improve the performance of a stacked autoencoder. From a high level perspective, fine tuning treats all layers of a stacked autoencoder as a single model, so that in one iteration, we are improving upon all the weights in the stacked autoencoder. In order to compute the gradients for all the layers of the

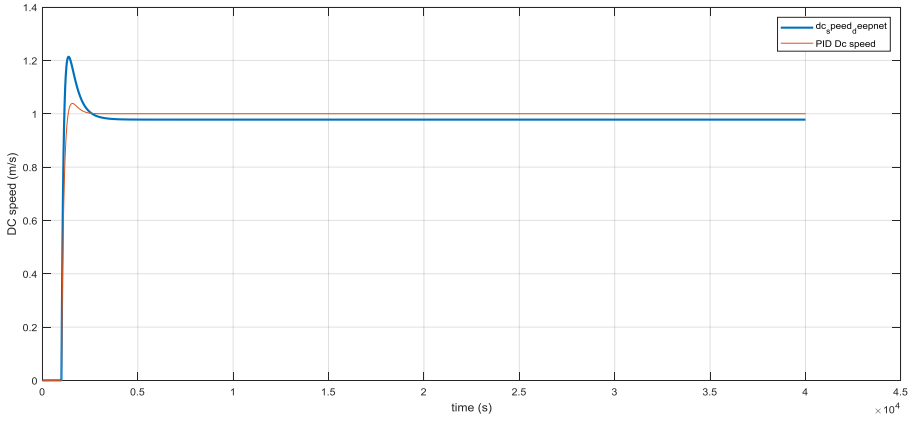
stacked autoencoder in each iteration, we use the Backpropagation Algorithm, as discussed in pervious chapter. As the backpropagation algorithm can be extended to apply for an arbitrary number of layers, we can actually use this algorithm on a stacked autoencoder of arbitrary depth.

The network is fined tune by retraining it on the training data in a supervised approach. So in this section the deep network is trained on the PID input and Output values. After doing some post processing of the data we achieved a deep Learning Controller.

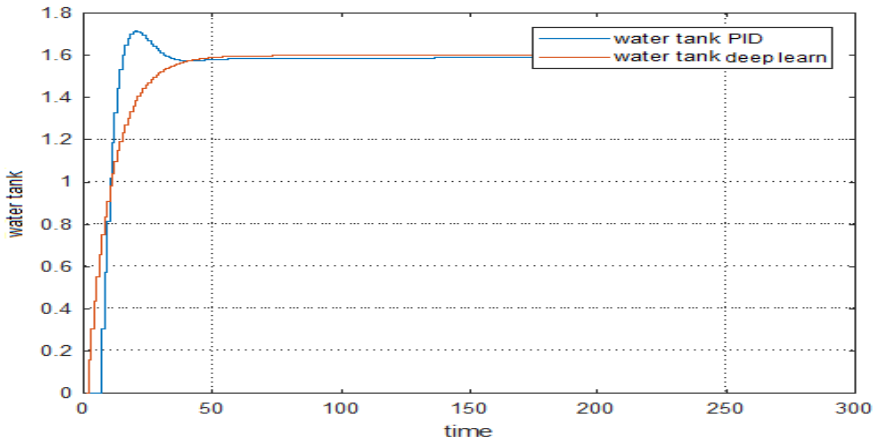
RESULTS AND COMPARISON

In this section, the results we have gained throughout the research are demonstrated. For the analysis purposes we will compare the RMSE values amongst the PID and Deep learning Controller based models.

In Figure 6, the outputs from PID controlled system and Deep learning Controlled system are compared in order to analyze how close we could teach our system to mimic the PID controller. Specifically in Figure 6(b), our proposed controller has avoided the initial peak shown by the PID controller that has resulted into having good RMSE.



(a)



(b)

Figure 6 Controllers Output VS time for continuous (a) and discrete (b)

Table 1, illustrates the RMSE differences we have achieved and its comparison with the PID controlled model. The RMSE are quite close for both the continuous and discrete systems and shows a satisfactory performance.

Table 1 RMSE values comparison

System	RMSE for PID model	RMSE for Deep Controller
Continuous System	0.0376	0.0425
Discrete System	0.1014	0.0900

CONCLUSION

In this paper, a deep learning controller based on SAE algorithm is designed to explore the ability of applying the deep learning algorithm to the control problems. A comparison study between the PID controller and the proposed deep learning controller was performed to verify the feasibility of the use of deep learning in control theory. The simulation results demonstrate the effectiveness of the proposed deep learning controller to be used as a control tool.

REFERENCES

[1] X. Zhu, C. Guan, J. Wu. and Y. Cheng, "Expectation MAximization Method for EEG-Based countinuos Cursor Control," *EURASIP on Applied Signal Processing*, vol. 1, p. 26, 2007.

[2] A. S. a. S. H. M. Toussaint, "ExpectationMaximization methods for solving (PO) MDPs and optimal," in *Inference and Learning in Dynamic Models*, 2010.

[3] G. Hinton, L. Deng, D. Yu and G. Dahl, "Deep Neural Networks for," *IEEE Signal Processing Magazine*, Nov 2012.

[4] I. Arel, D. Rose and T. Karnowski, "Deep Machine Learning—A New Frontier in Artificial Intelligence Research," *IEEE Computational Intelligence Magazine*, Nov 2010.

[5] W. X. M. Y. K. Y. S Ji, "3D Convolutional Neural Networks for Human Action Recognition," *IEEE Transon Pattern Analysis and Machine Intelligence*, Vols. vol. 35, no. 1, Jan 2013.

[6] Y. Bengio and Y. LeCun, "Scaling learning algorithms towards AI," in *Large-Scale Kernel Machines*, MIT Press, 2007.

[7] B. Y, "Learning Deep Architectures for AI," Foundations and Trends in Machine Learning. 2, no. doi:10.1561/2200000006, 2009.

[8] C.-Y. H. J.-C. Y. W.-C. Liou, "Modeling word perception using the Elman network," Neurocomputing, vol. 71, no. doi:10.1016/j.neucom.2008.04.030, p. 3150–3157, 2008.

[9] C.-Y. C. C.-W. L. J.-W. L. D.-R. Liou, "Autoencoder for Words," Neurocomputing, vol. 139, no. doi:10.1016/j.neucom.2013.09.055, p. 84–96, 2014.

[10] D. W. M. Kingma, "Auto-Encoding Variational Bayes," ArXiv e-prints, no. arxiv.org/abs/1312.6114, 2013.

[11] A. Boesen, L. L. and S. S.K., "Generating Faces with Torch," 2015. [Online]. Available: torch.ch/blog/2015/11/13/gan.html.

[12] A. M and B. F, "k-Sparse Autoencoders," Cornell University Library, no. arXiv:1312.5663, 2014.