Modeling Macroeconomic Dynamics with Deep Learning

Yuanhao Niu
yniu@nd.edu
University of Notre Dame

Cale Harnish
charnish@nd.edu
University of Notre Dame

ABSTRACT

We leverage Artificial Neural Networks (ANN) to approximate partial differential equations (PDEs), which represent macroeconomic dynamics. We first solve a baseline model in one dimension. We compare our solutions from deep learning with existing results from the literature in order to establish mathematical correctness and macroeconomic modeling capability. Then, we extend the baseline model with an additional input and a diffusion process. This algorithm complements traditional numerical techniques (*e.g.* finite difference method) and can be easily scaled to higher dimensions.

KEYWORDS

macroeconomics, partial differential equation, deep learning, neural networks, reinforcement learning

1 INTRODUCTION

Macroeconomics considers the study of large systems of interacting agents in order to explain the performance, structure, and behavior of the economy as a whole. Forecasts from macroeconomic models are then used to develop, or evaluate, economic policy for governments and institutions. Therefore, the ability to model macroeconomic dynamics and accurately predict future economic behavior is critical for the long term health of a nation.

The neoclassical growth model is the cornerstone of modern macroeconomics. It is well known in the literature that this model is mathematically equivalent to solving partial differential equations (PDEs) from optimal control theory [5] and dynamic programming [21]. However, the solution of these PDEs can be hard to achieve through traditional numerical techniques due to the "curse of dimensionality" [6]. Recently, deep learning algorithms have been developed which overcome this computational burden and produce solutions to high-dimensional parabolic PDEs [14]. In our work, we tailor these solution methods to the specific equations of macroeconomic dynamics studied in [1]. Specifically, we develop an artificial neural network (ANN) using reinforcement learning to find a solution to the PDEs with minimal error. We then compare our results with state-of-the-art solutions.

Paper Draft, 2019 Spring, Machine Learning

ACM ISBN .
https://doi.org/

2 RELATED WORK

There are many examples within the literature which utilize PDEs to model the distribution and evolution of economic variables. For example, optimal dynamic contracts and policies have been studied by Sannikov [28], Williams [31], and Farhi & Werning [12]. The labor market has been modeled by Alvarez & Shimer [4] and Lentz & Mortensen [18]. Furthermore, recent heterogeneous agent models have been investigated by Nuño [23] and Lucas & Moll [20].

In most cases, a numerical method is required in order to find solutions to these PDEs. Common techniques include: finite difference [13], finite volume [11], and finite elements [19]. While many of these have already been applied in the context of macroeconomics [1], these techniques are not easily applied to highly dimensional problems [6]. In contrast, machine learning algorithms regularly analyze highly dimensional data [2]. This has led some researchers to consider machine learning as a tool for solving PDEs, consider for example the work of Han *et al.* [14] and Kun Hu [16], which significantly influenced the algorithm described in this paper.

3 NEOCLASSICAL GROWTH MODEL

In order to verify the mathematical correctness of the algorithm, we first train the ANN on a simplified model with well understood solutions. The *Neoclassical Growth* (NCG) model is a workhorse of economics. The equations for this model can be derived by considering the discrete time Bellman equation

$$V(k_t) = \max_{c_t} \{u(c_t) + \beta V(k_{t+\Delta t})\} \tag{1}$$

where the value is a is function of the current capital $V(k_t)$ and it is defined recursively in terms of the utility of current consumption $u(c_t)$ and future values $V(k_{t+\Delta t})$ discounted by the factor β. We choose this discount factor such that:

$$\beta = e^{-\rho \Delta t},$$
$$\lim_{\Delta t \to 0} \beta = 1, \qquad \text{and} \qquad \lim_{\Delta t \to \infty} \beta = 0. \tag{2}$$

Now, by writing the recursive eq. (1) for an instant Δt into the future we have

$$V(k_t) = \max_{c_t} \left\{ \Delta t \; u(c_t) + e^{-\rho \Delta t} \; V(k_{t+\Delta t}) \right\} \tag{3}$$

and in the limit as $\Delta t \to 0$, this can be reformulated into the continuous description

$$\rho V(k) = \max_{c} \left\{ u(c) + \frac{\partial V}{\partial k} \frac{\partial k}{\partial t} \right\}. \tag{4}$$

and the agent has the constant relative risk aversion (CRRA) utility function

$$u(c) = \frac{c^{1-\sigma}}{1-\sigma} \qquad \text{and} \qquad \frac{\partial k}{\partial t} = k^{\alpha} - \delta k - c, \tag{5}$$

subject to the first order condition

$$\frac{\partial u}{\partial c} = \frac{\partial V}{\partial k}. \tag{6}$$

This model describes an economy which consists of a consumer, a producer, and a planner. The producer utilizes capital k to produce k^α, and capital depreciates by δk each period. The planner chooses the optimal amount of consumption c in order to optimize the consumer's welfare $u(c)$. The consumer enjoys the level of consumption as decided. The rest of the output $k^\alpha - \delta k - c$ will add to the capital for the next period.

Formulating the continuous description of the model helps reduce the computational burden of the max operator. This is because the first order conditions can yield a closed form solution for the optimal value of consumption c^*. The first order condition defines the optimal value as

$$c^* = \left(\frac{\partial V}{\partial k}\right)^{-\frac{1}{\sigma}}. \tag{7}$$

Substituting eqs. (5) and (7) into eq. (4) yields

$$\rho V(k) = \frac{1}{1-\sigma}\left(\frac{\partial V}{\partial k}\right)^{\frac{\sigma-1}{\sigma}} + \frac{\partial V}{\partial k}\left[k^\alpha - \delta k - \left(\frac{\partial V}{\partial k}\right)^{-\frac{1}{\sigma}}\right]. \tag{8}$$

This eq. (8) is the first PDE which we solve using the ANN and the results are detailed in section 6.

In the follow-up exercise, we increase the dimensionality of the model. In particular, we extend the NCG model to include the influence from additional parameters, such as productivity y. Therefore, eqs. (4) and (5) are replaced with:

$$\rho V(k,y) = \max_c \left\{u(c) + \frac{\partial V}{\partial k}\frac{\partial k}{\partial t} + \mu(y)\frac{\partial V}{\partial y} + \frac{\sigma^2(y)}{2}\frac{\partial^2 V}{\partial y^2}\right\}, \tag{9}$$

$$\frac{\partial k}{\partial t} = yk^\alpha - \delta k - c, \tag{10}$$

$$dy_t = -\theta(x_t - \mu)dt + \sigma dz_t \tag{11}$$

Equation (11) specifies that the dynamics of y_t follow a Ohrnstein-Uhlenbeck process, which translates to an autoregressive progress of order 1 (AR1) in discrete time. $\mu(y)$ and $\sigma(y)$ respectively define the drift and the variance as a function of productivity. Again using the first order condition to define the optimal value of consumption (*i.e.* eq. (7)), we can reformulate eq. (9) into

$$\rho V(k,y) = \frac{1}{1-\sigma}\left(\frac{\partial V}{\partial k}\right)^{\frac{\sigma-1}{\sigma}} + \frac{\partial V}{\partial k}\left[yk^\alpha - \delta k - \left(\frac{\partial V}{\partial k}\right)^{-\frac{1}{\sigma}}\right]\ldots + \mu(y)\frac{\partial V}{\partial y} + \frac{\sigma^2(y)}{2}\frac{\partial^2 V}{\partial y^2}. \tag{12}$$

This eq. (12) is the second PDE which we solve using the ANN and the results are detailed in section 6. Moreover, solutions from this model allow the economy to be influenced by more than one factor, which provides a more realistic simulation of a macroeconomy.

4 METHOD/APPROACH

This work uses artificial neural networks (ANNs) to approximate the solution to each PDE. In particular, we train each ANN such that it becomes a representation of the value function (*i.e.* $V(k)$ in eq. (8) and $V(k,y)$ in eq. (12)), as illustrated in fig. 1. In this figure, each circle represents a perceptron [27]. The architectures of the ANNs are defined by a number of hidden layers N_L and, in this work, each layer contains the same number of perceptrons N_U. Each connection in this network has an associated weight and the act of training the network corresponds to learning the values for these weights. One common applications of ANNs is called supervised learning, where both the input and desired output are provided as the training data [24]. In such a case, the error (*e.g.* the mean-squared error) between the predicted labels and the desired output defines a loss function. The back-propagation algorithm [30] minimizes the loss function and distributes the error term back through the layers to modify the weights at each perceptron. Additionally, updating the weights can be made more computationally efficient by using stochastic gradient descent to minimize the loss function [26]. Specifically, w_{ab} the weight connecting units a and b is updated by

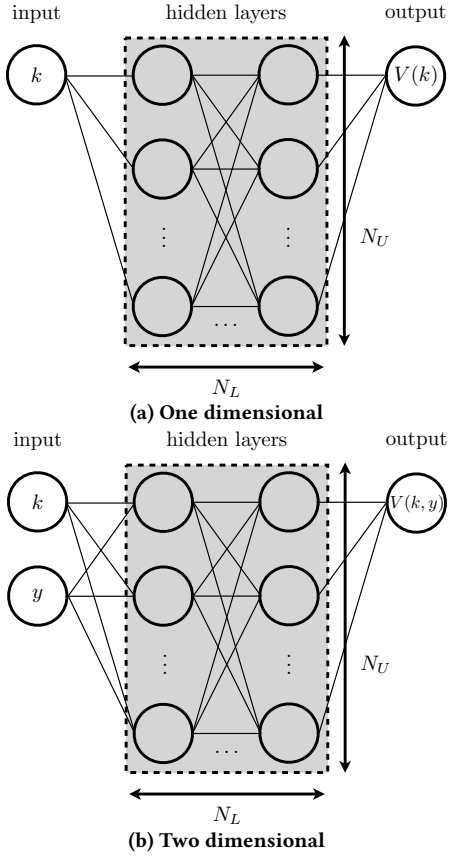


Figure 1: Artificial Neural Networks as a representation of the value function.

$$w_{ab}^{(i+1)} = w_{ab}^{(i)} - \eta\frac{\partial C}{\partial w_{ab}} + \xi(i) \tag{13}$$

where η is the user defined learning rate, C is the loss function, and $\xi(i)$ is a stochastic term. However, it is important to note this

method of training the ANN to learn $V(k)$ or $V(k, y)$ only works if these functions are known within the training data.

Instead, we do not know $V(k)$ or $V(k, y)$ explicitly, we only know that they are solutions to the PDEs eq. (8) and eq. (12) respectively. Therefore, the only input is a sampling of the domain $k \in [k_{\min}, k_{\max}]$, $y \in [y_{\min}, y_{\max}]$ and the associated labels are generated endogenously. This application of machine learning is often described as reinforcement learning [29]. In this method, the ANN interacts with its environment (*e.g.* the PDE) in discrete time steps. At each time t, the ANN receives an observation (*e.g.* the residual from the PDE). Finally, based on this observation, an action is performed (*e.g.* updating the approximation of the value function).

Specifically, our work begins with a uniform sampling of the inputs k, y and an initial guess for the value function $V^{(0)}$. Then, each iteration n of the learning process will calculate the residual from the PDE using the current value function $V^{(n)}$ and use this residual to define a new target for the ANN. The ANN is retrained on this new target. As the residual decreases in magnitude over time, the approximation of the value function becomes closer to the true solution of the PDE.

In this work the notation HJB is used to define the residuals because the PDEs in this work (*i.e.* eq. (8) and eq. (12)) can be related to the "Hamilton-Jacobi-Bellman" equation [16]. The residual for the one dimensional PDE is defined by

$$HJB(V) = \frac{1}{1-\sigma}\left(\frac{\partial V}{\partial k}\right)^{\frac{\sigma-1}{\sigma}} + \frac{\partial V}{\partial k}\left[k^{\alpha} - \delta k - \left(\frac{\partial V}{\partial k}\right)^{-\frac{1}{\sigma}}\right] - \rho V \tag{14}$$

and for the two dimensional PDE, the residual takes the form

$$HJB(V) = \frac{1}{1-\sigma}\left(\frac{\partial V}{\partial k}\right)^{\frac{\sigma-1}{\sigma}} + \frac{\partial V}{\partial k}\left[yk^{\alpha} - \delta k - \left(\frac{\partial V}{\partial k}\right)^{-\frac{1}{\sigma}}\right] \cdots + \mu(y)\frac{\partial V}{\partial y} + \frac{\sigma^2(y)}{2}\frac{\partial^2 V}{\partial y^2} - \rho V. \tag{15}$$

Finally, the target for each iteration is defined by

$$\text{target}^{(n)} = V^{(n)} + \Delta t \cdot HJB\left(V^{(n)}\right) \tag{16}$$

The parameter Δt in eq. (16) dictates how frequently the value function is updated and acts as the learning rate in reinforcement learning literature. This implementation is described more succinctly in algorithm 1, which we have implemented using the Python programming language and the TensorFlow library [10].

Algorithm 1 Learn Value Function

$n = 0$ and $\varepsilon = 10$
while $n <$ max. iterations and $\varepsilon >$ tolerance **do**
  $\text{target}^{(n)} = V^{(n)} + \Delta t \cdot HJB\left(V^{(n)}\right)$
  $V^{(n+1)} =$ ANN trained to fit $\text{target}^{(n)}$
  $\varepsilon = \left\| HJB\left(V^{(n)}\right)\right\|_1$
  $n = n + 1$

5 DATA

As described in section 4, this algorithm's use of data is slightly different than many common machine learning implementations. Since this is an application of reinforcement learning, the only input is a sampling of the domain $k \in [k_{\min}, k_{\max}]$, $y \in [y_{\min}, y_{\max}]$ and the associated labels are generated endogenously.

6 EVALUATION

The problems which we seek to solve are well defined by eq. (8) and eq. (12) and the method by which we produce their solutions is detailed in section 4. However, there remain a few parameters which we have not yet explicitly defined: N_L the number of hidden layers, N_U the number of hidden units in each layer, η the learning rate, Δt the time step size, and which activation function to use for the units in the ANN.

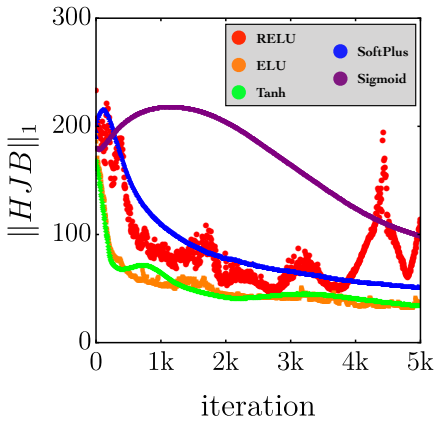
It has been shown in the literature that when using ANNs as function approximators (*i.e.* supervised regression learning), only one hidden layer is needed so long as it contains a sufficient number of hidden units [3, 9, 25]. However, the application described in this paper is more complex as it requires learning not just the function V but also the derivatives of V and the overall solution to a PDE. Therefore, much like the earlier work of [15, 16], we have decided to use three hidden layers throughout this work. The number of hidden units is varied to suit the complexity of each PDE and in general we have observed that a learning rate of $\eta = 10^{-3}$ is sufficient.

Furthermore, we have observed that choosing the activation function is an important consideration. This is due in large part to the fact that our ANN is a representation of the value function and our algorithm requires derivatives of the value function. Therefore, we frequently must differentiate the ANN with respect to its inputs. This corresponds to calculating derivatives of the activation function and imposes the constraint that the activation function must be differentiable. In our work, we have considered the following activation functions: RELU, ELU, Tanh, Soft Plus, and Sigmoid. More information about these functions and their explicit definitions can be found in [7, 8, 17, 22].

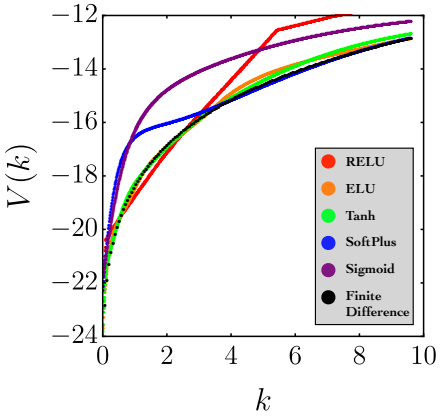
In order to judge the effectiveness of each activation function, the one dimensional PDE eq. (8) is solved using using each of the listed activation functions with 5 thousand iterations and the following parameters:

hidden layers (N_L)	3
hidden units (N_U)	[20,20,20]
Learning rate (η)	0.001
Gradient Descent	Stochastic

The magnitude of the residual, as defined by the one-norm of eq. (14), is calculated at each iteration and shown in fig. 2a. Additionally, the earlier work of [1] solved this same PDE using finite difference methods. Here, we present their solution next to our ANN solutions in fig. 2b. As can be seen in these figures, some activation functions provide notably different results. Based on these findings we have chosen to use the Tanh activation function in all applications of the algorithm described in this paper. Furthermore, these results demonstrate that our implementation simulates model dynamics comparable with traditional technology.



(a) Error over time



(b) The approximate solution $V(k)$

Figure 2: The effect of various activation functions.

Next, our solutions of the NCG model with two inputs eq. (12) allow us to verify that ANNs can successfully model macroeconomic dynamics in higher dimensions. Due to the additional complexity of this PDE, the following parameters were used for the neural network:

hidden layers (N_L)	3
hidden units (N_U)	[64,64,64]
Learning rate (η)	0.001
Gradient Descent	Adam

The domain is chosen such that the same values for k are used as in the one dimensional case and $y \in [0.8 \times \text{mean}, 1.2 \times \text{mean}]$. This is because, as described in section 3, the additional variable y represents productivity and low productivity will correspond to values below the mean while high productivity will correspond to values above the mean. The solution to this PDE is a two-dimensional surface in (k, y) space and fig. 3 depicts the superposition of three slices corresponding to low, mean, and high values of productivity. Note that the general trend of the value function is similar to the one dimensional case. Additionally, when the productivity is low the value is always worse than the mean. Similarly, when the productivity is high the value is always better than the mean.

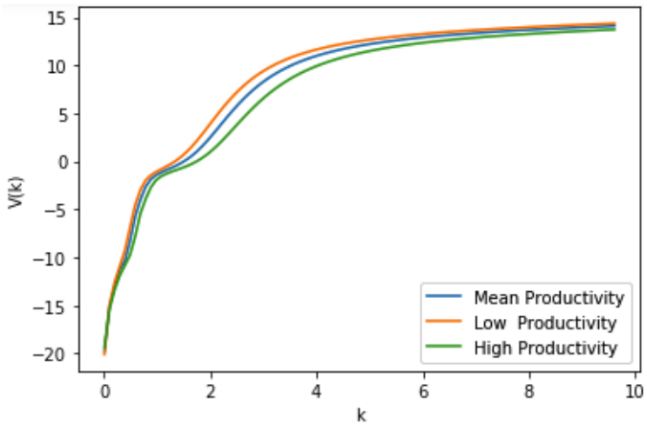


Figure 3: Three slices of the solution to eq. (12)

Such macroeconomic dynamics are consistent with theory and provide evidence that our algorithm provides economic simulation capability.

7 LIMITATIONS

In this work we have shown multiple benefits to solving PDEs using ANNs, however it should be noted that our findings also present multiple limitations to the current algorithm.

The first of these may already be clear to the reader through careful observation of fig. 2a. We have observed that while iterating towards the true solution of the PDE, the residual may rapidly decrease within the first thousand iterations but during most of the computational time this convergence becomes asymptotic. It should be noted that the residual does indeed decrease over time but it is at a remarkably slow rate. Improving the overall convergence rate would make this method much more attractive to the research community at-large.

Another important limitation arises from careful consideration of eq. (7). Since the exponent is a negative rational number, there are constraints on the first derivative of V with respect to k. In particular, we *must* have

$$\frac{\partial V}{\partial k} > 0. \tag{17}$$

Violating this constraint produces imaginary numbers or infinities which would prevent the algorithm from converging towards the true solution of the PDE. Ensuring that this constraint is never violated presents a non-trivial numerical challenge when using ANNs to represent the function V. Putting hard boundary does not work. We are currently try to put non-negative constraints on the weights during the training process.

8 CONCLUSION

In this work we have developed a ANN which models macroeconomic dynamics. Furthermore, we have demonstrated that this numerical method is accurate and capable of modeling macroeconomic dynamics comparable with modern numerical methods, such as [1]. Moreover, we have shown that solving PDEs of higher dimensionality with the algorithm described in this work will not

impose a prohibitive computational burden. In contrast to traditional numerical methods (*e.g.* the finite difference method) which face the "curse of dimensionality", the procedure described in this work needs only modification on the architecture as the dimensionality increases. This is expected to provide a much more feasible approach to high dimensional problem.

9 REFERENCES

[1] Yves Achdou, Jiequn Han, Jean-Michel Lasry, Pierre-Louis Lions, and Benjamin Moll. 2017. *Income and Wealth Distribution in Macroeconomics: A Continuous-Time Approach.* Working Paper 23732. National Bureau of Economic Research. https://doi.org/10.3386/w23732

[2] Omar Y. Al-Jarrah, Paul D. Yoo, Sami Muhaidat, George K. Karagiannidis, and Kamal Taha. 2015. Efficient Machine Learning for Big Data: A Review. *Big Data Research* 2, 3 (2015), 87–93.

[3] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. 2018. A Convergence Theory for Deep Learning via Over-Parameterization. (2018). arXiv:cs.LG/1811.03962

[4] Fernando Alvarez and Robert Shimer. 2011. Search and Rest Unemployment.(Report). *Econometrica* 79, 1 (2011), 75–122.

[5] Martin Beckmann and Richard Muth. 1954. On the solution to the fundamental equation of inventory theory. (1954).

[6] R. Bellman, Rand Corporation, and Karreman Mathematics Research Collection. 1957. *Dynamic Programming.* Princeton University Press, Princeton, NJ.

[7] FranÃğois BÃĪlisle, Yoshua Bengio, Charles Dugas, RenÃĪ Garcia, and Claude Nadeau. 2002. Incorporating Second-Order Functional Knowledge for Better Option Pricing. (2002). http://proxy.library.nd.edu/login?url=https://search-proquest-com.proxy.library.nd.edu/docview/1698152839?accountid=12874

[8] Djork-ArnÃĪ Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). (2015). https://arxiv.org/abs/1511.07289 ICLR 2016.

[9] G. Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2, 4 (1989), 303–314.

[10] Martín Abadi et al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. (2015). https://www.tensorflow.org/ Software available from tensorflow.org.

[11] Robert Eymard, Thierry GallouÃńt, and RaphaÃĺle Herbin. 2000. Finite volume methods. *Handbook of Numerical Analysis* 7 (2000), 713–1018.

[12] Emmanuel Farhi and IvÃąn Werning. 2013. Insurance and Taxation over the Life Cycle. *The Review of Economic Studies* 80, 2 (283) (2013), 596–635.

[13] Christian Grossmann. 2007. *Numerical treatment of partial differential equations.* Springer, Berlin ; New York.

[14] Jiequn Han, Arnulf Jentzen, and E Weinan. 2018. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences* 115, 34 (2018), 8505–8510.

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

[16] Kun Hu. 2019. *The Impact of Financial Frictions on Innovation, GibratâĂŹs Law, and Growth with Heterogeneous Firms.* Ph.D. Dissertation. UCLA.

[17] Bekir Karlik and A Vehbi Olgac. 2011. Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems* 1, 4 (2011), 111–122.

[18] Rasmus Lentz and Dale T. Mortensen. 2010. Labor Market Models of Worker and Firm Heterogeneity. *Annual Review of Economics* 2 (2010), 577–602.

[19] Daryl L Logan. 2002. *A first course in the finite element method* (3rd ed.. ed.). Brooks/Cole, Pacific Grove, CA.

[20] Robert E. Lucas and Benjamin Moll. 2014. Knowledge Growth and the Allocation of Time. *Journal of Political Economy* 122, 1 (2014), 1–51.

[21] Robert C. Merton. 1973. An Intertemporal Capital Asset Pricing Model. *Econometrica (pre-1986)* 41, 5 (09 1973), 867.

[22] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10).* Omnipress, USA, 807–814. http://dl.acm.org/citation.cfm?id=3104322.3104425

[23] Galo NuÃśo. 2013. Optimal control with heterogeneous agents in continuous time. (2013).

[24] Varun Kumar Ojha, Ajith Abraham, and VÃąclav SnÃąÅąel. 2017. Metaheuristic design of feedforward neural networks: A review of two decades of research. *Engineering Applications of Artificial Intelligence* 60, C (2017), 97–116.

[25] Allan Pinkus. 1999. Approximation theory of the MLP model in neural networks. *Acta Numerica* 8 (1999), 143–195.

[26] Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. *The Annals of Mathematical Statistics* 22, 3 (1951), 400–407. http://www.jstor.org/stable/2236626

[27] F. Rosenblatt. 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review* 65, 6 (1958), 386–408.

[28] Yuliy Sannikov. 2008. A Continuous- Time Version of the Principal: Agent Problem. *The Review of Economic Studies* 75, 3 (2008), 957–984. http://www.jstor.org/stable/20185061

[29] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction.* MIT press, Cambridge , MA.

[30] P. Werbos. 1975. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* Harvard University, Cambridge, MA.

[31] Noah Williams. 2011. Persistent Private Information. *Econometrica* 79, 4 (2011), 1233–1275. http://search.proquest.com/docview/878998328/