Using Deep Learning to Solve Parabolic Partial Differential Equations Arising in Heterogeneous Agent Models

René M. Glawion
Department of Computer Science
Stanford University
rglawion@stanford.edu

Josie Oetjen
Department of Computer Science
Stanford University
josieoet@stanford.edu

Abstract

Why are income and wealth so unequally distributed? To answer such questions, economists study so-called Mean-Field games, which can be summarized as a system of two coupled partial differential equations (PDEs), one to describe the optimal behavior of agents and one to model macroeconomic distributions. However, as these models' complexity increases due to economists' growing desire to design them most realistically, the underlying PDEs get increasingly difficult to solve with standard numerical methods like finite differences or finite elements. We use a feed-forward neural network trained with Adam to solve these systems of equations in the workhorse heterogeneous agent model of [2]. Compared to the benchmark finite difference solution, our approximation achieves a mean percentage deviation of 0.178% on a 50x50 development set grid and 0.200% on a 40x40 test set grid.

1 Introduction

Partial differential equations (PDEs) naturally arise in economic models featuring stochastic uncertainty. But the increasing complexity of economic models, driven by the desire to design these models in the most realistic way, makes it harder to find closed-form solutions for these PDEs. Thus, it is of high relevance to develop efficient and robust numerical methods to solve PDEs in economic frameworks. However, the traditionally employed finite difference methods suffer from the curse of dimensionality, i.e., as the model becomes higher dimensional, the grid to solve it grows exponentially large, and those methods become more or less impossible to apply in practice [30]. We hope that deep neural networks can provide such robust solution methods. Our project aims to solve (Hamilton-Jacobi-)Bellman equations (HJB) since these equations describe the optimal economic behavior in a wide range of models.
We focus on a macroeconomic heterogeneous agent model, more precisely a Mean-field game (MFG) introduced by [27]. The idea of MFGs is to couple the HJB equation, which describes the optimal behavior of agents, with the Fokker-Plank equation (FP), which keeps track of the distributions of agents among different dimensions. Such models are of rising relevance as they help analyze distributions of macroeconomic variables, e.g., wealth inequality, an emerging problem in the United States and many other countries [37]. For example, in 2021, the top 1% of the wealthiest Americans owned 27% of the wealth, and therefore for the first time in history, more than the 60% of middle-class Americans combined [14].
As an application, we use the current workhorse model of [2], which models income and wealth inequality simultaneously and has an accurate finite difference solution. However, this model still lacks a lot of real-world complexity, which is needed for real-life applications and, more specifically, policy analysis. For example, it does not feature a banking sector. With our project, we hope to be able to provide neural network solutions for much more comprehensive models to develop (policy) instruments reducing income and wealth inequality.
We use a feed-forward neural network to approximate the solution of the HJB, which we denote as the value function V. The network's input is a generated two-dimensional grid of data points representing agents' labor income and wealth. The network's output is the value of V at the specific point on the grid. We train our model using Adam optimizer and evaluate the performance for our main error metric mean squared error (MSE) for our development and test set, using the finite difference solution as the benchmark. Further, we present and discuss the metrics maximum absolute error (MAE) and mean percentage error (MPE) and compare the two approaches regarding runtime and usability.

2 Related Work

Deep learning-based approximation methods for PDEs have first been proposed in the 1990s by [12] and [26]. However, only recently, with the rise of deep learning, the idea of using neural networks to solve PDEs has gained traction again, and a variety of methods to tackle such equations emerged.

Theoretical Results: Currently, no theorem proves that neural networks can overcome the curse of dimensionality for all kinds of PDEs, which would mean that the number of parameters used to describe the approximating neural network is growing at most polynomially in the PDE dimension $d \in \mathbb{N}$ and the reciprocal $1/\varepsilon$ of the desired approximation accuracy $\varepsilon \in \mathbb{R}^+$ [6]. Nonetheless, there are some first results for different classes of PDEs. [24] show that feed-forward neural networks using ReLU activations can do so for PDEs with Lipschitz continuous nonlinearities. Other cases include PDEs for pricing American options, neural networks to approximate the heat equation, elliptic PDEs, and controlled PDEs with uncertain initial conditions and source terms [17; 13; 18; 34]. For an overview, we refer to [23].
Feed-Forward Neural Networks: [33] propose deep learning methods to solve PDEs in physics. They, for example, use feed-forward neural networks with gradient descent to solve the Schrödinger equation. Based on that, [29] created DeepXDE, a library for solving physics informed neural networks. We build on those results and convey the ideas to economic models, where we similarly have parabolic PDEs. Additionally, [38] developed the Deep Galerikin Method, where they approximate PDE solutions by a neural network instead of a linear combination of basis functions. We tried their random sampling approach during training, but for our 2-dimensional PDE the results did not improve.
Deep BSDE Method: [20] show how deep learning networks trained with stochastic gradient descent can be used to tackle high dimensional PDEs. However, their approach uses backward stochastic differential equations (BSDEs), specifically to draw the connection between BSDEs and quasi-/semilinear and even nonlinear PDEs, making it difficult to apply in practice. Roughly speaking, one has to reformulate the underlying PDEs as stochastic optimization problems on an infinite-dimensional function space, using the idea behind the Feynman-Kac theorem (cf. [31]). However, the method gets applied in various fields like finance and robotics since it allows providing error bounds [22; 11; 32; 23; 24].
Other Approaches: [36] use a ResNet to work with a Lagrangian formulation of the optimization problem and enforce the underlying HJB. Similar to the BSDE formulation, this requires a reformulation of the problem. [28] and [8] parameterize the value and density functions of the MFG by two neural networks. Then, solving the MFG is a special case of training a generative adversarial network (GAN), which in their case is a Wasserstein GAN. The disadvantage is that a Wasserstein GAN is a particular instance of a deterministic MFG. Using our approach, we can directly solve the stochastic formulation and thus do not need a second neural network to solve for the resulting density function.
Economic Literature: In the field of MFGs, [19] use deep learning to develop approximation methods for Markovian Nash equilibria of stochastic games with a finite number of agents based on approximations for HJBs. [10; 9; 28; 36] apply deep learning to solve for Markovian Nash equilibria of stochastic games with an infinite number of agents based on MFG theory and approximations for HJB PDEs similar to our setting, although with a finite time-horizon which makes the solution easier by allowing to solve the HJB backward in time. The closest study to ours is [15], who use deep learning to model migration patterns. They employ feed-forward neural networks to solve for the optimal behavior of agents using minibatch gradient descent. We construct our model similarly but can simplify their approach. For example, we do not need to explicitly account for our model's (boundary) constraints but rather do it implicitly. Second, we use the Adam optimizer, letting training converge much faster. Third, we achieve a better training performance on an equidistant grid, whereas [15] drew training points from the ergodic distribution similar to [38], which can be computationally costly if there does not exist a closed-form solution for the ergodic distribution.

3 Problem Description

We apply our method to the continuous-time heterogeneous agent model developed by [2], in particular the version in Online Appendix G, to which we refer for additional details of the parameters and exact functional forms. The model features a continuum of agents who maximize their lifetime utility from consumption c and savings s, given their wealth a and labor income z. The wage process follows a truncated Vasicek mean reversion process with mean reversion speed φ, normalized long-run mean 1, and variance σ. Further, agents are risk-averse with parameter γ, and they discount future consumption with rate ρ. Each agent faces the same interest rate r, aggregate capital stock K, which firms use to produce goods, capital depreciation rate δ, and factor share of capital α, a production function parameter.
Given that setting, one can describe the agents' optimal program for choosing the path of consumption, c, by the HJB. We denote the solution for the value function as V and the solution of the FP equation as the density p, which tracks the distribution of labor income and wealth. We solve the stationary equilibrium of the model, which takes the form

$$\begin{aligned}\rho V(a,z) &= \max_c \left\{ \frac{c^{1-\gamma}}{1-\gamma} + \partial_a V(a,z)(z+ra-c) + \partial_z V(a,z)\varphi(w-z) \right.\\ &\qquad\qquad \left. + \tfrac{1}{2}\partial_{zz}V(a,z)\sigma^2 \right\}, \qquad (1)\\ 0 &= -\partial_a\left(sp(a,z)\right) + \partial_z\left(\partial_z \frac{\sigma^2}{2} p(a,z) - \varphi(w-z)p(a,z)\right),\end{aligned}$$

where we have the constraint that the density p integrates to 1. Note that in our code, we solve the Fokker-Planck equation by using the adjoint of the infinitesimal generator of the HJB following [2] based on an upwind scheme developed in [4]. If that is not possible, numerical simulations indicate that deep learning-based approximation algorithms for linear FP equations might be more efficient than standard Monte Carlo approximation methods, not just at a fixed space-time point but on an entire region, such as on high-dimensional cubes cf. [5, Sec. 3]. This would require a second feed-forward neural network, which we will leave for future research.

Table 1: Hyperparameter Tuning.
The table summarizes our hyperparameter tuning process. It displays the hyperparameters we tune, the initial grid values, and the finally selected values. We choose the hyperparameters by randomly sampling from the grid and evaluating the performance on the development set.

Hyperparameter	*Grid Values*	*Selected Value*
Layers	[2, 4, 6, 8, 10]	6
Neurons	[32, 64, 128, 256, 512]	128
Hidden Layer Activations	[tanh, ReLU]	ReLU
Learning Rate	30 points, logarithmic scale from 10^{-5}- 10^{-1}	$1.26 \cdot 10^{-4}$
Epochs	10 points, linear scale from 10,000 to 100,000	50,000

4 Dataset and Features

First, we set the global parameters of the model to K = 3.8, $\alpha = \frac{1}{3}$, $w = (1-\alpha)K^{\alpha}$, $\sigma = 0.1$, $\gamma = 2$, $\rho = 0.05$, $\delta = 0.1$, and $r = \alpha K^{\alpha-1} - \delta$, in line with economic literature [2; 35].
The inputs we vary, the features for our neural network, are the state variables for wealth a and wage z, whereas the outcomes are the respective values of the value function. To generate data for our training process, we define 100 equidistant points for a ranging from -1 to 30 and 100 for z ranging from 0.5 to 1.5, following [2] for the ranges. We then collect the points in a grid, resulting in 10,000 combinations of input features (a, z) in total. As the development (test) set, we use a 50x50 (40x40) grid on the same domain, resulting in 2,500 (1,600) points. Due to the sufficient description of the data by the grid parameters, we refrain from including examples of our data in this report.
Before feeding the data to the model, we normalize the data by subtracting the mean and dividing by the standard deviation. This is especially important for decreasing training runtime since the ranges of a and z differ substantially.

5 Methods

5.1 Solution Algorithm

To approximate the value function V, we define the feed-forward neural network $\tilde{V}(a, z; \theta^V) : \mathbb{R}^2 \to \mathbb{R}$, where the inputs are the grid points for a, z, and the weights θ^V and the output is the respective value of V. We train the network using Adam optimizer [25] on the full batch because Adam performs better than SGD, which we also tried, and overall training converges faster on the whole batch than when using mini-batches. We implement the network using Python 3.8.10, NumPy 1.22.3 [21], SciPy 1.8.0 [39], and TensorFlow 2.8.0 [1] on a NVIDIA GeForce RTX 3090.

5.2 Cost Function

We define our cost function as the difference between the left-hand side and the right-hand side of the value function (1),

$$J_{\tilde{V}}(a, z; \theta^V) \equiv || - \rho\tilde{V}(a, z; \theta^V) + \frac{c^{1-\gamma}}{1-\gamma} + \partial_a\tilde{V}(a, z; \theta^V)(z + ra - c) + \partial_z\tilde{V}(a, z; \theta^V)\varphi(w - z)$$
$$+ \tfrac{1}{2}\partial_{zz}\tilde{V}(a, z; \theta^V)\sigma^2(z)||_2 + \text{penalty for non-increasing } v.$$

We add a penalty term because we struggled to generate a strictly increasing and concave value function during the early stage of training, which is assumed in economic theory (more wage/wealth always has a higher value for the agent). Thus, we follow the notion of [3], Appendix G.3, and include a (large) penalty term to the cost function for infeasible predictions in the following way: (1) Set a punishment parameter $\varepsilon^{\text{punish}} = 10^{-8}$. (2) If the value function is not increasing between two grid points, add a penalty $1/\varepsilon^{\text{punish}}$ for each violation. The punishment term will guide the parameter updates so that improved iterations of the value functions do not feature non-increasing areas. Note that we do not need to explicitly account for boundary conditions compared to earlier works discussed in section 2.

6 Experiments, Results and Discussion

6.1 Hyperparameters

To tune the hyperparameters, we define a grid with different values on appropriate scales, where we choose the ones to tune and the grid values following related literature. For example, [34] use ReLU, whereas [33] uses tanh for activations. Then, we try the hyperparameter combinations on this set of points at random and select the combination with the lowest costs on the development set. We display the tuned hyperparameters, the initially defined values, and the finally selected parameters in Table 2. We leave the Adam hyperparameters β_1 and β_2 at their default values 0.9 and 0.999.

6.2 Evaluation Metrics

As main evaluation metrics to measure the performance and reliability of our approach on the development and test set, we use one optimizing metric, the MSE, and one satisfying metric, which requires the value function to be concave, a

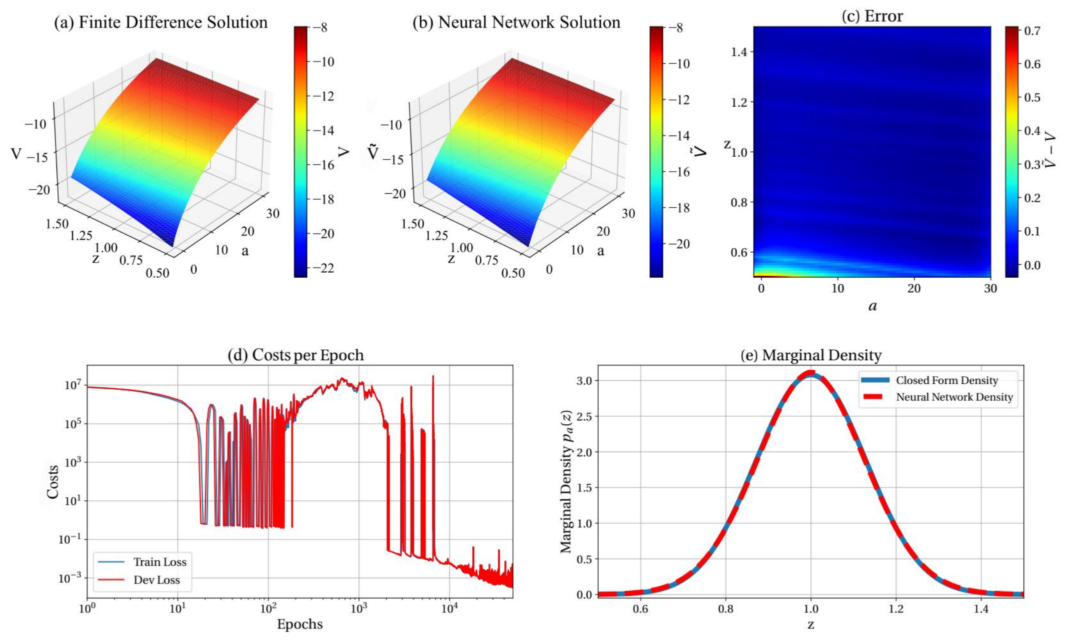

Figure 1: The figure summarizes some of our results. Panel (a) depicts the value function as a function of labor income z and wealth a, numerically obtained using finite difference, our benchmark solution. Panel (b) displays the value function for our neural network solution for 50,000 training epochs. Panel (c) illustrates the approximation error, defined as the difference between the finite difference solution and the neural network solution. Panel (d) shows the costs as a function of epochs for the train set and the development set, and Panel (e) depicts the marginal density for z compared to the normal distribution implied by the Vasicek model for the process z.

fact from economic theory. We compute the MSE as the mean squared difference between the finite difference solution, which we denote as the true value, and the solution obtained from the neural network,

$$MSE = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(V^{\text{true}}(a_i, z_j) - \tilde{V}(a_i, z_j; \theta^V) \right)^2,$$

where N and M refer to the number of points along the a and z-axis in the train/development/test set. We decide on the MSE as our main metric because in economics, we prefer approximations with many small errors over ones with fewer but larger errors, and the MSE weights large deviations more heavily than small ones. The satisfying metric is required since a flat surface function that always outputs a value of zero also solves the PDE but violates economic assumptions. Further, we present and discuss the MAE and the MPE to provide a more throughout discussion of the final approximation, which are defined as

$$MAE = \max_{\substack{i=1\dots,N \\ j=1,\dots,M}} |V^{\text{true}}(a_i, z_j) - \tilde{V}(a_i, z_j; \theta^V)|, \quad \text{and} \quad MPE = \frac{100}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{|V^{\text{true}}(a_i, z_j) - \tilde{V}(a_i, z_j; \theta^V)|}{|V^{\text{true}}(a_i, z_j)|}.$$

The MAE gives us valuable insights into the maximum deviation of our solution that can be observed somewhere on the grid and, thus, by chance on a point that can be realized in actual data. The MPE describes the relative error and thus allows for a comparison of our networks' performance with approaches in the literature solving other PDEs.

6.3 Results and Discussion

Figure 1 and Table 2 summarize our results. In Figure 1, by comparing Panel (a) and (b), we see that our neural network solution is close to the benchmark finite difference solution while preserving the functions' concavity. To better visualize the overall error on the entire domain, we refer to Panel (c). Here we observe that the error is largest for small values of z and a, measuring around 0.3. However, with increasing values along both dimensions, the error quickly stabilized at low values below 0.1. We likely observe this pattern because the function has a stark curvature for low grid values, making it harder for the network to approximate it. Panel (d) shows that the costs for both the train and the development set oscillate heavily for the initial 10,000 epochs and then decreases more consistently. The oscillation can be explained by the large penalty term during the period where the network still learns to compute an increasing function. Lastly, we show the marginal density for the labor income z in Panel (e). Since the wage process follows a Vasicek process, we know its asymptotic distribution is normal, and we can plot the closed-form solution to compare it with the implied marginal density of our neural network solution. As we can see, the two densities are almost identical, a further indicator of our approximation's good fit and an essential result for distributional economic policy analysis.

Table 2: Error Metrics for Different Epoch Numbers on the Train, Development and Test Set. The table shows the MSE, MAE, and MPE on the train set, the development set, the test set, and the corresponding training runtime in seconds for different epochs of training.

	Train Set			*Development Set*			*Test Set*			
Epochs	*MSE*	*MAE*	*MPE*	*MSE*	*MAE*	*MPE*	*MSE*	*MAE*	*MPE*	*Runtime*
1,000	44.965	11.451	48.989	45.110	11.494	48.901	45.175	11.529	48.861	77.086
5,000	0.066	0.716	1.281	0.074	0.754	1.352	0.078	0.769	1.383	384.878
10,000	0.029	0.590	0.612	0.034	0.625	0.706	0.037	0.639	0.741	772.992
25,000	0.004	0.568	0.479	0.006	0.599	0.477	0.007	0.616	0.479	1,928.684
50,000	0.003	0.710	0.149	0.005	0.741	0.178	0.006	0.759	0.200	3,916.951

We summarize the MSE, the MAE, the MPE, and the runtime for the train, development, and test set for different numbers of epochs in Table 2. The error measures almost always decrease with the number of epochs. At 50,000 epochs, we have a good approximation on the train set with a MSE of 0.003, although we still have a slight avoidable bias since the Bayes error for this task is 0. Further, the metrics are slightly worse for the development and test set since the network learns to approximate the function for the specific points in the train set instead of learning the exact functional form. Whether this indicates a variance problem that can be resolved with regularization or with a more densely sampled train set is left for future research. Interesting to note is that the MSE and MPE, indicating our average performance, always decrease with the number of epochs, while the MAE again increases from 25,000 epochs on for all sets. As such, although our performance on the entire grid improves, we are further off for some single points after more training epochs. We achieve a mean percentage error of 0.178% for the development set and 0.200% for the test set, which is more than sufficiently precise with regard to the usual measurement errors in macroeconomics. Nevertheless, for example, [33] achieve a MSE of $3.495 \cdot 10^{-5}$ for their network solving the Allen-Cahn equation on generated data points, a performance we do not meet. However, it should be noted that their functions range itself is smaller, and, additionally, [40] tried but failed to replicate their results. In terms of runtime, training the network for 50,000 epochs requires a bit more than an hour, which is an acceptable timeframe. Yet, especially when trying various hyperparameter combinations and more epochs, it takes significantly longer than the finite difference approach.

7 Conclusion and Future Work

We show that feed-forward neural networks are capable of solving parabolic PDEs arising in complex economic models featuring a continuum of heterogeneous agents. We compare the resulting approximation to the standard finite difference solution of the underlying model. Our neural network solution is almost identical to the finite difference solution, and we demonstrate how researchers can modify the cost function to preserve functional properties like concavity. It achieves a MPE of 0.178% for the development and 0.200% for the test set, which is a sufficient approximation when working with macroeconomic data. However, the neural network solution is usually not preferable for problems where finite difference solutions exist. The main reasons are the slight deviations from the true solution, especially for areas of high curvature. Additionally, although implementing the neural network itself is easier and quicker, much time is devoted to hyperparameter tuning and training the model in general. Nevertheless, finite difference underlies the curse of dimensionality in higher-dimensional settings, and neural networks might provide the only feasible solution in those cases. With our work, we hope to encourage economists to develop more complex models and try solving them with deep networks if their conventional numeric methods fail.
However, our work comes with a few limitations. First, our way of evaluating the satisfying metric requires knowledge about the geometrical form of the solution of the underlying PDE, which one might not always have readily available. A more founded and mathematical evaluation would increase our work's objectivity. Further, we chose our test set to be equidistantly distributed along both axes. In real-world data, we observe much higher proportions of agents below zero wealth, e.g., 32% of US households cannot afford a $400 emergency payment [7]. Thus, we have disproportionately often data points in the grid area where our performance is below average. Therefore, our MSE evaluated on our equidistant test set grid is likely lower than the MSE for a set of actual data points.
For future research, in terms of improving our specific network, we suggest addressing the higher error in areas where the solution features stark curvature. For instance, future work could sample more points in that particular area, for example, by using a log grid on the a-axis instead of an equidistant one. This results in more grid points for low values of a than high values of a, where the value function becomes more linear and the approximation more accurate. Furthermore, we advise more comprehensive initial sets for the hyperparameter search with a coarse to fine search process. This especially applies to the number of epochs, where further exploration was not possible due to the limited time for the project and the high training runtime. Regarding the general topic of solving PDEs in economics, it would be necessary to try solving equations with more dimensions than two, as the problems of finite difference then start to show. However, finding a benchmark solution then presents a challenge. Possible solutions may be to test the algorithm on a class of high-dimensional free boundary PDEs which have the special property that error bounds can be calculated for any approximate solution, as done in [38], or to employ high dimensional Monte-Carlo methods [16].

8 Contributions

Both group members contributed equally to the project. We would like to thank CS230 Project TA Tianhe Yu for project guidance throughout the quarter.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. Software available from tensorflow.org.

[2] Y. Achdou, J. Han, J.-M. Lasry, P.-L. Lions, and B. Moll, *Income and wealth distribution in macroeconomics: A continuous-time approach*, The review of economic studies, 89 (2022), pp. 45–86.

[3] M. Azinovic, L. Gaegauf, and S. Scheidegger, *Deep equilibrium nets*, International Economic Review, forthcoming (2022).

[4] G. Barles and P. E. Souganidis, *Convergence of approximation schemes for fully nonlinear second order equations*, Asymptotic analysis, 4 (1991), pp. 271–283.

[5] C. Beck, S. Becker, P. Grohs, N. Jaafari, and A. Jentzen, *Solving the kolmogorov pde by means of deep learning*, Journal of Scientific Computing, 88 (2021), pp. 1–28.

[6] C. Beck, M. Hutzenthaler, A. Jentzen, and B. Kuckuck, *An overview on deep learning-based approximation methods for partial differential equations*, arXiv preprint arXiv:2012.12348, (2020).

[7] Board of Governors of the Federal Reserve System, *Economic well-being of u.s. households in 2021*, Research & Analysis, (2022).

[8] H. Cao, X. Guo, and M. Laurière, *Connecting gans, mfgs, and ot*, arXiv preprint arXiv:2002.04112, (2020).

[9] R. Carmona and M. Laurière, *Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games: Ii–the finite horizon case*, arXiv preprint arXiv:1908.01613, (2019).

[10] ———, *Convergence analysis of machine learning algorithms for the numerical solution of mean field control and games i: the ergodic case*, SIAM Journal on Numerical Analysis, 59 (2021), pp. 1455–1485.

[11] Q. Chan-Wai-Nam, J. Mikael, and X. Warin, *Machine learning for semi linear pdes*, Journal of Scientific Computing, 79 (2019), pp. 1667–1712.

[12] M. Dissanayake and N. Phan-Thien, *Neural-network-based approximations for solving partial differential equations*, communications in Numerical Methods in Engineering, 10 (1994), pp. 195–201.

[13] D. Elbrächter, P. Grohs, A. Jentzen, and C. Schwab, *Dnn expression rate analysis of high-dimensional pdes: Application to option pricing*, Constructive Approximation, 55 (2022), pp. 3–71.

[14] Feredal Reserve, *Distribution of Household Wealth in the U.S. since 1989*, tech. rep., 2022.

[15] J. Fernandez-Villaverde, G. Nuno, G. Sorg-Langhans, and M. Vogler, *Solving high-dimensional dynamic programming problems using deep learning*, Working Paper, https://maximilianvogler.github.io/My_Website/Deep_Learning.pdf, (2020, accessed May 3rd, 2022).

[16] P. Glasserman, *Monte Carlo methods in financial engineering*, vol. 53, Springer, 2004.

[17] L. Gonon, P. Grohs, A. Jentzen, D. Kofler, and D. Šiška, *Uniform error estimates for artificial neural network approximations for heat equations*, arXiv preprint arXiv:1911.09647, (2019).

[18] P. Grohs and L. Herrmann, *Deep neural network approximation for high-dimensional elliptic pdes with boundary conditions*, arXiv preprint arXiv:2007.05384, (2020).

[19] J. Han and R. Hu, *Deep fictitious play for finding markovian nash equilibrium in multi-agent games*, in Mathematical and Scientific Machine Learning, PMLR, 2020, pp. 221–245.

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

[21] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, *Array programming with NumPy*, Nature, 585 (2020), pp. 357–362.

[22] P. Henry-Labordere, *Deep primal-dual algorithm for bsdes: Applications of machine learning to cva and im*, Available at SSRN 3071506, (2017).

[23] C. Huré, H. Pham, and X. Warin, *Deep backward schemes for high-dimensional nonlinear pdes*, Mathematics of Computation, 89 (2020), pp. 1547–1579.

[24] M. Hutzenthaler, A. Jentzen, T. Kruse, and T. A. Nguyen, *A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations*, SN partial differential equations and applications, 1 (2020), pp. 1–34.

[25] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, (2014).

[26] I. E. Lagaris, A. Likas, and D. I. Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE transactions on neural networks, 9 (1998), pp. 987–1000.

[27] J.-M. Lasry and P.-L. Lions, *Mean field games*, Japanese journal of mathematics, 2 (2007), pp. 229–260.

[28] A. T. Lin, S. W. Fung, W. Li, L. Nurbekyan, and S. J. Osher, *Apac-net: Alternating the population and agent control via two neural networks to solve high-dimensional stochastic mean field games*, arXiv preprint arXiv:2002.10113, (2020).

[29] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, *Deepxde: A deep learning library for solving differential equations*, SIAM Review, 63 (2021), pp. 208–228.

[30] W. M. McEneaney, *Max-Plus Methods for Nonlinear Control and Estimation*, Birkhäuser, 2006.

[31] N. Nüsken and L. Richter, *Solving high-dimensional hamilton–jacobi–bellman pdes using neural networks: perspectives from the theory of controlled diffusions and measures on path space*, Partial Differential Equations and Applications, 2 (2021), pp. 1–48.

[32] M. Pereira, Z. Wang, I. Exarchos, and E. A. Theodorou, *Learning deep stochastic optimal control policies using forward-backward sdes*, arXiv preprint arXiv:1902.03986, (2019).

[33] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational physics, 378 (2019), pp. 686–707.

[34] C. Reisinger and Y. Zhang, *Rectified deep neural networks overcome the curse of dimensionality for nonsmooth value functions in zero-sum games of nonlinear stiff systems*, Analysis and Applications, 18 (2020), pp. 951–999.

[35] D. Romer, *Advanced Macroeconomics*, New York: McGraw-Hill, 2012.

[36] L. Ruthotto, S. J. Osher, W. Li, L. Nurbekyan, and S. W. Fung, *A machine learning framework for solving high-dimensional mean field game and mean field control problems*, Proceedings of the National Academy of Sciences, 117 (2020), pp. 9183–9193.

[37] E. Saez and G. Zucman, *The rise of income and wealth inequality in america: Evidence from distributional macroeconomic accounts*, Journal of Economic Perspectives, 34 (2020), pp. 3–26.

[38] J. Sirignano and K. Spiliopoulos, *Dgm: A deep learning algorithm for solving partial differential equations*, Journal of computational physics, 375 (2018), pp. 1339–1364.

[39] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python*, Nature Methods, 17 (2020), pp. 261–272.

[40] S. Wang, S. Sankaran, and P. Perdikaris, *Respecting causality is all you need for training physics-informed neural networks*, arXiv preprint arXiv:2203.07404, (2022).