Accelerated Approximation of Bellman Equation Solutions: Agent Policy Optimization With a Feedforward Neural Network

Victoria M. Garibay[1][0000−0003−0399−0591]

University of Amsterdam, Netherlands Victoria.Garibay@gmail.com

Abstract. Solving recursive equations through iteration can be a computationally expensive endeavour, and the time required to reach an optimal solution delays the progress of any dependent processes. To address this issue for a specific use case of decision-making in an agent-based model, a method of replacing the iterative function used in said model, a Bellman equation, with a feedforward multilayer perceptron was developed. A hyperparameter grid search was performed to determine the combination of architecture, learning rate, and batch size which produced results deviating the least from those of the original iterative method. With the resulting neural network, accepting four inputs and yielding two outputs, the time required to compute outputs scales sublinearly with the number of agents. Excluding training time, for a set of 1,000 agents, the selected neural network produces output at over 66,000 times the speed of the original function. It achieves this acceleration while maintaining a 99.3% accuracy in adaptation strategy selection and 0.10 mean absolute error in consumption, leading to its ready adoption as an acceptable replacement for the original method.

Keywords: Dynamic Programming, Recursive Function, Surrogate Model, Multilayer Perceptron

1 Introduction

Modelling autonomous agents interacting in an environment is an increasingly computationally intensive task. As new technologies develop improving the efficiency and capacity of agent based models (ABMs), new levels of complexity become realizable [9, 3]. Rising towards this higher potential can sometimes involve reconsidering the approach to certain traditional model aspects. The particular case addressed in this study is an ABM which follows the evolution of stock capital for heterogeneous household agents in a society subjected to exogenous shocks. Loosely founded on the Boltzmann wealth model, within a timestep, agents exchange capital and decide how to optimally consume and invest their resources depending on their attributes and perceptions of their changing environment. The decision on how to partition their capital is based on a Bellman equation [2], which has a recursive structure accommodating sequential processes such as forming an optimal consumption policy based on its

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

expected future values. The form the Bellman equation took in the case model did not have a known analytical solution. Solutions to the Bellman equation have been approximated in a variety of creative ways in the past [23, 22, 15, 13[1]]. One such calculation applicable to the custom Bellman equation arrangement required by the ABM is policy iteration, a numerical method which approximates an optimal policy by alternately evaluating and optimizing the result of a value function until convergence is reached [22]. This method in its established form was not a feasible option for the case model due to the desired scale (1,000,000+ agents), which a newly developed tensor-based framework (DGL-ABM) was otherwise capable of supporting. Due to the imperfect nature of the process of human decision making, there was some flexibility in the performance of a replacement and, conveniently, the capacity to generate samples as needed with the existing method, limited only by time and memory dedicated to their production. To overcome the barrier posed by iterative calculation for a problem set of sufficient magnitude to serve a million potentially unique agents, several options for interpolation from known results to form a reference landscape map or response surface were considered [12, 24[2]]. However, the dimensionality issues rising from a four-input, two-output structure and the irregularity the custom Bellman equation made traditional interpolation methods poorly suited to the task. Instead, attention shifted to the development of a surrogate model of the equation. The judgment was reached that a Gaussian process regression was inappropriate due to the piecewise discontinuity of certain specialized aspects of the equation, moderately high dimensionality of the problem, and the potentially large sample size required to accommodate this dimensionality. With these restrictions in mind, it was decided to pursue a neural network approximation for the equation. There are many examples of neural networks being used to model equations, particularly in engineering and earth sciences [4, 6, 16]. The potential applicability of neural networks to agent behavior is now also being appreciated [15, 14]. However, approaches used in existing research differ from that taken for the task in this study, as they primarily focus on using recursive neural networks to capture the learning process of agents. For the proposed use case, instead of considering agents to be themselves neural networks, the equation which describes their decision-making is being approximated with a multilayer perceptron (MLP) trained on computed results. A similar technique has been applied to economic models in the past, primarily seeking proof of concept, but literature on practical application attempts, successful or not, is limited [17]. The contents of this manuscript are a written record of the motivation, exploration, and formation of a static mapping technique for the results of a custom Bellman equation computation. The primary objectives of this research were to 1) identify a superior combination of hyperparmeters with which to train a neural network for the outlined computational task and 2) assess whether the resulting MLP could be a worthy replacement for the iterative method—specifically, if it could achieve the substantial speed gains required by the ABM setup while main-

[1] Based on methods currently found in [7]
[2] Based on methods from [1]

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

taining, at minimum, a 90% fidelity in investment strategy and, at maximum, an average absolute error of 0.25 in consumption. The following methods were developed out of need, naive to the specific science and standards surrounding neural network design. Through the process, the conductor of this research has gained knowledge, become more aware of limitations in the field, and come to appreciate that there is much remaining for all researchers to understand and discover in this vast frontier. As such, it is hoped that the methods documented and ensuing discussion can contribute as precedent for improved, streamlined efforts to design similar approximation tools in the future.

2 Methods

Subsequent sections describe the methods followed in the process of training an MLP as a potential replacement for an existing solver used as a decision model for agents in an ABM.

2.1 Problem Context

The ABM itself is not the focus of this manuscript, thus it will not be explained in great detail. However, readers may find it helpful to know a summary of relevant points regarding the agent decision process. Agent decisions are described by a constrained Bellman equation satisfied by a custom value function maximizing the value of the state of stock capital, k, resulting from a particular consumption, c, and investment in adaptation i_a (Eq. 1). Maximization is constrained by the rule that consumption and investment do not exceed available funds at the following time step. Investment in adaptation is associated with a multiplier which reduces the impact of a perceived exogenous shock, rendering an adjusted shock factor $\theta_{m,t}$. The function u is a standard isoelastic utility function with exact value dependent on the risk aversion of an agent, σ, while f_{income}—a function of aptitude, α, and stock capital—is the maximum value from the results of a capital-only Cobb-Douglas production function evaluated for available capital output elasticity exponents, γ, minus the *cost* assigned to their use (u and f_{income} are described further in Algorithm 1 in the Appendix). The depreciation, δ, and discount factor, β, were considered model constants, homogeneous for all agents.

$$V(k_t) \equiv \max_{c_t, i_{a,t}} \{u(c_t, \sigma) + \beta V(\theta_{m,t}[f_{income}(\alpha, k_t) + (1-\delta)k_t - c_t - i_{a,t}])\} \quad (1)$$

2.2 Bellman Equation Sample Data Generation

Before attempting to model the Bellman equation (Eq. 1) with a neural network, it was necessary to develop a training and testing dataset. As mentioned, the solution to the equation was being computed iteratively relying on techniques

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

documented by Stachurski [22]. For additional context, this iterative optimization was occurring up to three times per set of agent properties as conceptually represented in Algorithm 1, found in the Appendix[3]; this was because three variations of the Bellman equation, one for each adaptation option, needed to be evaluated to determine which produced the highest value for the agent. The inputs to the algorithm were the agent properties k_t, stock capital, θ_t, shock perception, σ, risk aversion, and α, aptitude for income generation, as well as the information on adaptation options, technology coefficients, and their associated costs. The outputs were the optimal consumption and choice of adaptation investment, i_a, between options of a N(one), L(ow), and H(igh) protection multiplier, m. The multiplier affects Equation 1 with $\theta_{m,t} = \theta_t + m(1 - \theta_t)$.

A Sobol' sequence-based Saltelli sampling matrix (N = 8,192) was formed from the distributions specified in Table 1 [21, 20]. Duplicate agents were removed, leaving 47,935 unique agents. All agents for which the Bellman solution resulted in an error or did not converge were removed from the dataset, resulting in 47,281 remaining samples (a 98.8% retention rate). Prior to training and testing the model, each input variable was scaled according to its distribution:

– For uniformly distributed:

$$\text{input} = \frac{\text{input} - \text{minimum}}{\text{maximum}}$$

– For normally distributed:

$$\text{input} = \frac{\text{input} - \text{mean}}{\text{standard deviation}}$$

Likewise, the output variables were scaled according to their maximum values:

$$\text{output} = \frac{\text{output}}{\text{maximum}}$$

Variable	Distribution Type	Distribution Parameters
θ	uniform	[0,1)
k	uniform	[0.1,10)
α	normal	loc = 1.08; scale = 0.074
σ	uniform*	[0.1,2)

* Value rounded to nearest tenth.

Table 1. Distributions used in Saltelli sampling of agent attributes.

[3] The code used to generate samples is also available as a notebook [5].

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

2.3 Hyperparameter Grid Search

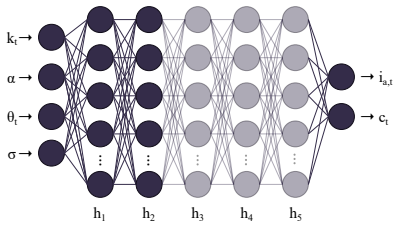

Fig. 1. Highly generalized diagram of the multilayer perceptron used to map the iterative Bellman equation input to output. The number of hidden layers and their respective widths were varied according to Table 3 to execute a grid search for the best architecture and hyperparameter combination.

An exhaustive grid search was performed to determine the most appropriate model architecture and hyperparameters for the equation mapping task. In the process of defining ranges and points for the search, formal and informal guidelines were consulted (e.g., Philipp [18], Goodfellow et al. [8], Richetti et al. [19]), but they were—by their own admission in some cases—fairly vague and supported the idea that hyperparameter selection can be highly application-dependent. The MLP neural network for which the search was conducted had an input layer of four nodes, output layer of two nodes, and between two and five hidden layers (Fig. 1). The relative shape of the network was generalized into five architectural variations, or ratios of layer widths, as specified in Table 3. The n_{max} referenced in that table is the maximum number of nodes (i.e., the number of nodes in the widest layer of the network). All layers used a basic ReLU (rectified linear unit) activation function. Model training used the Adam optimizer and L2 loss (i.e., mean squared error, MSE, Eq. 3) to minimize the difference between the predicted values and the training dataset consisting of 80% of the agent sample data. The remaining 20% of the sample was reserved for validation. Training ran for 200 epochs with an early stop triggered after 20 epochs with no improvement in validation MSE. Other considerations for the grid search were batch size and learning rate. Due primarily to storage constraints, three random seeds were utilized for each unique combination of architecture type, learning rate, n_{max}, and batch size listed in Table 2. The code used for the model grid search can be found publicly [5]. The foundational network training and testing scripts followed the Pytorch Template Project [11].

2.4 Metrics & Model Selection

The metrics recorded for the best model of each combination of parameters included mean absolute error (MAE, Eq. 2) for the model, percentage of incorrect guesses for i_a, and MAE for c_t of the validation data. For each of these three metrics, the average value produced from the three seeds was considered to represent the performance of each unique architecture and hyperparameter combination. It was decided that the primary metric of consideration would be the combined MAE, provided that the same model also fell within the top ten performers in the other two metrics. In the event of failure to meet this secondary requirement,

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

Architecture Type	n_{max}	Learning Rate	Batch Size
ThreeLayer (3L)	512	0.001	64
FourLayer (4L)	1024	0.01	128
FiveLayer (5L)	2048	0.0001	512
PudgeFiveLayer (P5L)	-	-	-
PudgeSixLayer (P6L)	-	-	-

Table 2. Grid search space for which combinations of one item of each column determined the model and hyperparameters for training.

Architecture Type	h_1	h_2	h_3	h_4	h_5
ThreeLayer (3L)	$n_{\max}$	$n_{\max}/2$	-	-	-
FourLayer (4L)	$n_{\max}$	$n_{\max}/2$	$n_{\max}/4$	-	-
FiveLayer (5L)	$n_{\max}$	$n_{\max}/2$	$n_{\max}/4$	$n_{\max}/8$	-
PudgeFiveLayer (P5L)	$n_{\max}/2$	$n_{\max}$	$n_{\max}/2$	$n_{\max}/4$	-
PudgeSixLayer (P6L)	$n_{\max}/4$	$n_{\max}/2$	$n_{\max}$	$n_{\max}/2$	$n_{\max}/4$

Table 3. Layer widths specified in relation to maximum number of nodes (n_{max}) for each architecture type. The column titles $h_1 ... h_5$ correspond to the hidden layers as labeled in the stylized illustration of the network provided by Figure 1.

the next best model would be evaluated, loosening the top n standard by one, and so on down the rankings.

Beyond selecting the best model for the intended purpose, additional information was collected on differences in model performances to investigate any potential trends for variables in the grid search space. For this aspect, model MSE and MAE were the chosen metrics of comparison.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \text{ and} \tag{2}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3}$$

where y_i is the basis point, $\hat{y}_i$ is the predicted value, and n is the sample size.

3 Results & Discussion

With respect to execution time alone, using an MLP as opposed to the, admittedly poorly optimized, iterative computation method was a tremendous success. In Figure 2, it is possible to observe the linear scaling of the iterative version and

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

the sublinear scaling of the MLP version with increasing sample size. It should be taken into account that the iterative computation method was applied to agents in series, implying that time efficiency could have been increased through parallelization.

While an improvement in computation time was anticipated with the MLP method, the sheer magnitude of the improvement, emphasized by the identical log scales of the boxplots, is extremely advantageous in the context of developing a population-scale ABM and far beyond any improvement that could currently be achieved through simple parallelization of the iterative computation method (Fig. 2). For 10 runs of a 1,000-agent sample, the average execution time was 38957s (standard deviation 956s) for the iterative method and 0.53s (standard deviation 0.23s) for the MLP method [5]. The following subsections contain more information on the MLP itself and how its performance compared to the original iterative computation.

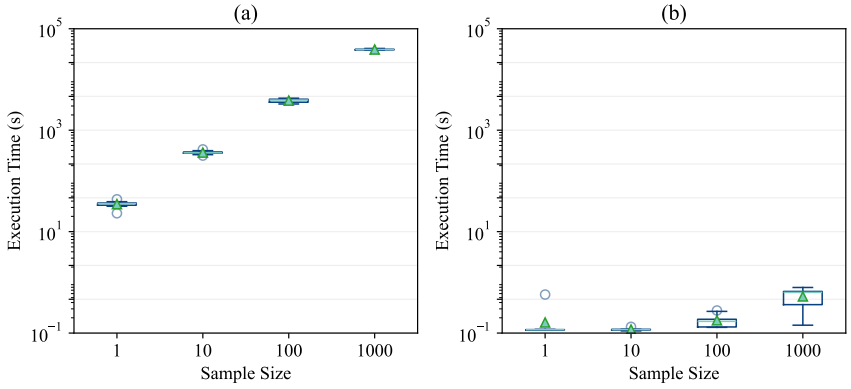

Fig. 2. Boxplots of execution time in seconds on a log scale for (a) the original iterative solution method and (b) the neural network solution method for increasing sample size demand. Triangles indicate the mean of 10 runs. The y-axis scale is kept constant for easier comparison.

3.1 Model Selection

In Figure 3, the best and worst performers in the metrics chosen as a basis of model selection are shown. The highest ranked model for overall combined MAE was the P6L architectural ratio with a maximum layer width of 2,048 nodes, a learning rate of 0.001, and a batch size of 64. Note that, while the same model was also ranked first for the metric pertaining to i_a, it placed tenth for c_t MAE. The closeness of the former and discrepancy of the later may be due to the order of magnitude difference between the scales for the outputs; the error in i_a

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

(maximum 0.5) had finer granularity in its influence on MSE loss during training than that of c_t (maximum 40). The precise model applied to the ABM used the previously stated hyperparameters with a seed of 84 and took 954s to train.

Other findings well highlighted by the plots are the great similarity in performance among the better models, and the poor performance of 3L models with 0.0001 learning rates in terms of overall MAE (Fig. 3a). Also, visible in Figure 3c, the higher learning rate, 0.01, produced results with greater variance in response to seed change. This tendency was also generally true of the middle values not shown in the figure. The trend is sensible since a high rate leads to coarser network updates, making the initialization more crucial and potentially pushing the model into suboptimal regions of the loss landscape [8]. It should also be noted that neither batch normalization nor dropout were used in the network model.

3.2 Trends in Hyperparameter Optimization

While conducting hyperparameter grid search without optimization in the search itself may be deemed inefficient, it was elected primarily as a straightforward, thorough way of scientifically selecting a model and in response to the lack of empirical studies reporting architecture and hyperparameter selection in the context of MLPs for equation mapping. As computational budgets for this task permitted, the exhaustive search was conducted to determine if any general trends could be identified to direct future attempts at similar mapping problems. It may be speculated that while the findings in this research cannot be generalized per se, they may perhaps serve as a way to justify targeting a particular region of the search space.

The differences in performance by controlled hyperparameter averaged over the remaining three are shown in Figure 4. For the maximum layer width, n_{max}, the downward trend in medians is very subtle and not observable in the mean at all, indicating that after 1024 nodes there is little improvement; however, there is a reduction in variance as number of nodes increases within the studied range (Fig. 4a). The response of changing the learning rate is more complex, with more of the combinations performing their worst with the lowest learning rate and their best at a moderate learning rate of 0.001 (Fig. 4b). This is consistent with the prior observation about high variance with high learning rate and for the lowest learning rate may be speculatively attributed to the limitation to 200 epochs [8]. For the architectural shape, while the improvement from adding a fourth layer is substantial, it appears that adding layers beyond four results in very small marginal improvements in mean performance (Fig. 4c). Another note of interest is the small gain achieved by ramping up to the widest hidden layer as demonstrated in the differences in median and mean between 5L and P5L (Fig. 4c, Tab. 3). As documented in literature, an increase in batch size is one technique sometimes used to decrease model training time; this choice is, at some point, to the detriment of the model performance [25]. Within the parameters of this grid search there was an upwards trend in error with increasing batch size (Fig. 4d). This suggested that, for the tested learning rates, most of the batch

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

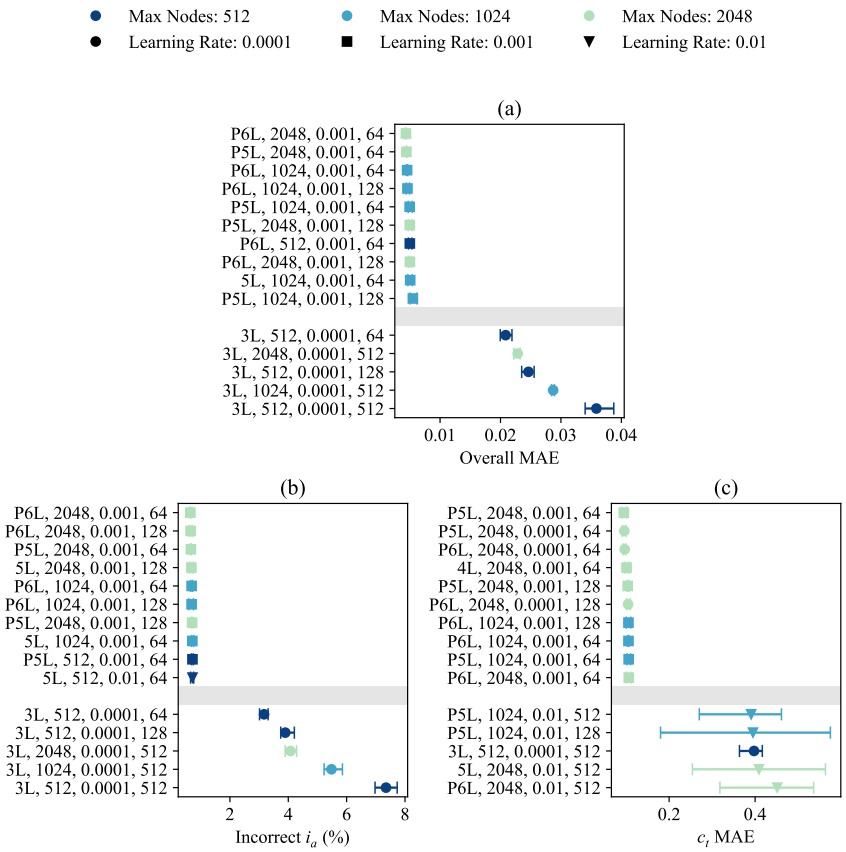

Fig. 3. The ten highest- and five lowest-ranked hyperparameter combinations performance in (a) overall mean absolute error, (b) percent of incorrect guesses for adaptation investment, and (c) mean absolute error in consumption for the validation sample in comparison with the original iterative Bellman equation method. The combination labels follow the convention: Architecture Type, n_{max}, Learning Rate, and Batch Size as specified in Table 2. For each point marking the mean value of the metric, the error bars represent the minimum and maximum value, the point shape indicates the learning rate, and the point color is indicative of n_{max}.

sizes were larger than the critical size, beyond which gradient noise becomes too low, decreasing the generalizability of the trained model to testing data. All of the observations stated for the MAE aligned closely with results for MSE.

For all hyperparameters, the variance in performance suggested the importance of their interaction effects. The heatmaps in Figure 5 facilitate the investigation of relationships between hyperparameters as well as successful and poor combinations. Similarly to the boxplots, it can be observed that learning rate had a strong influence, but in the heatmap, it is more clear that the width of

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

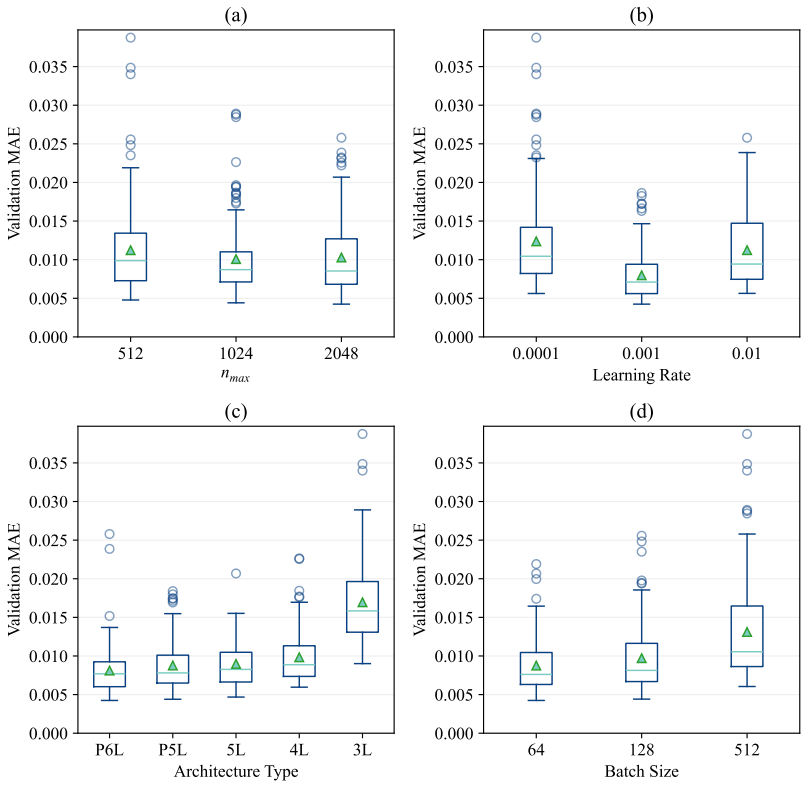

Fig. 4. Boxplots marking the spread of performance in overall mean absolute error for the validation sample categorized by (a) maximum number of nodes, (b) learning rate, (c) architecture type, and (d) batch size. Triangles indicate the mean value.

the network had an impact on the magnitude of that effect (Fig. 5f). As mentioned before, the average performance decreased with increase in batch size and improved with the addition of layers, however the learning rate appears to modulate these effects (Fig. 5d,e). From these plots, general combinations to embrace or avoid in future training exercises can be identified; e.g., a model with batch size of 64, n_{max} of 1024, learning rate of 0.001, and P6L structure will likely perform well, while a model with batch size of 512, n_{max} of 512, learning rate of 0.0001, and 3L structure will likely perform very poorly (Fig. 5).

3.3 Limitations, Notes, & Future Work

A weak point in making generalizations, even in the very specific context of this case study task, is the small model seed sample size of three. Although the variance for points of the same combination was fairly low, particularly for

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

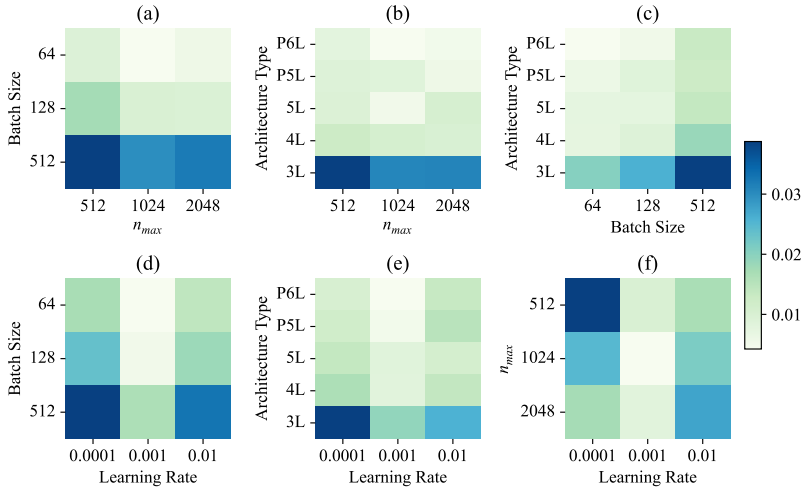

Fig. 5. Heat maps showing the combined effects of two hyperparameters on the overall mean absolute error for the validation sample. The hyperparmeters compared include (a) batch size to maximum nodes, (b) architecture type to maximum nodes, (c) architecture type to batch size, (d) batch size to learning rate, (e) architecture type to learning rate, and (f) maximum nodes to learning rate. Dark blue indicates relatively high error while light green indicates relatively low error.

the best-performing models, uncertainty in performance and ranking could be reduced by training more models with each hyperparameter combination. Upon further research into general guidelines surrounding MLPs, it was surmised that 2,048 nodes is considered very wide for a network with only four inputs and that such an overabundance of nodes may potentially lead to memorization of training sets [8]. For models with an n_{max} of 2,048, the average MSE of the validation dataset was approximately 93% higher than that of the training dataset, strongly suggesting that these models were subject to overfitting; however, as the average absolute difference was ∼0.00044, the practical consequences were considered negligible, and the initially selected model was retained for use with the ABM. This knowledge will be considered for the next iteration of hyperparameter search and training, which will incorporate the adaptation option information as input variables so that more flexibility can be gained in the ABM implementation. Future efforts will also use larger training sample sizes to improve generalizability and reduce overfitting. Expanding the model inputs would provide some answers as to whether it might be appropriate to generalize the research to other Bellman equations. Further comparison against optimized solvers of other equation forms would be useful in determining the robustness of the method and findings, but this type of testing was far beyond the scope and aspiration of this research.

The evidence that the shape of the architecture had relevance, more specifically that ramping the width up then down versus abruptly up was better for

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

results, deserves further experimentation. It is unclear whether the total number of nodes being slightly higher was the driver of this outcome or the shape itself was responsible. Although only tangentially related, a similar observation was made in a study of convolutional neural networks [10]. Other experiments which were conducted outside of the scope of this manuscript included splitting the task into two MLPs (one for each output with i_a as categorical), experimenting with alternate activation functions (Swish and Mish), and experimenting with scaling inputs and outputs. With regard to splitting the model, preliminary results did indicate that there was minor improvement, particularly in results for consumption, which is to be expected given the current arrangement in which it is effectively underrepresented in the loss calculation during training. By using Swish and Mish in place of ReLU, there was improvement in the average metrics for incorrect i_a and c_t MAE, particularly with Mish. However, in both the split case and the alternate activation function case, the improvement was so marginal, that it was deemed unnecessary to replace the equation in the ABM. Still, these aspects may be explored further in future training exploits. The experiments on scaling included omitting the scaling process for inputs, outputs, or both. While the outcome of these experiments will not be fully summarized here, the most telling statistics were that, in an average performance, using both unscaled inputs and outputs yielded a 10,940% increase in model MSE, 1,259% increase in overall MAE, and 18% increase in training time over fully scaled counterparts.

4 Conclusion and Takeaway

As ABMs and other models which rely on computationally complex subfunctions grow in scale, occasions where their implementation is inhibited by techniques which were previously acceptable become more prevalent. The case study solved as documented in this manuscript is but a single example to set a precedent for the use of MLPs to overcome this situation. While the journey taken in the process of conducting this research could definitely be termed a learning experience, the results met the need that was established well enough to allow the modelling of agent behavior with a Bellman equation to continue without a major overhaul in fundamental basis. While perhaps not a perfect replacement, the neural network performs with sufficient accuracy, at a far faster rate, and scales sublinearly with the addition of more agents. This provides hope that other models may benefit from similar replacement strategies in the future, perhaps incorporating some of the caveats against overfitting, observations on thresholds of diminishing return for node and layer additions, and notes on architectural performance differences.

Acknowledgments. The author acknowledges and is grateful for constructive academic discussions with Debraj Roy and Alex Gabel. This research was conducted with support from the Dutch Research Council (NWO) under contract 27020G08, titled "Computing societal dynamics of climate change adaptation in cities".

Disclosure of Interests. The author is aware of no competing interests regarding the content of this article.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

5 Appendix

Algorithm 1: Computation of optimal consumption and adaptation

Input: Dictionary of agent parameters with k_t, θ_t, σ, α and $AdapTable$ (array of i_a and m pairs); $TechTable$ (array of γ and $cost$ pairs)
Output: $i_{a,t} \in \{N, L, H\}$ and c_t

$feasible \leftarrow []$
for $i_a \in AdapTable$ **do**
  if $i_a < \max_{i=1,\ldots,n} [\mathtt{f}(\alpha, k_t, TechTable) + \theta_t(1-\delta)k_t]$ **then**
    $(c, v, \text{convergence}) \leftarrow$ `solve_bellman(BellmanEquation(u,f,`$k_t, \theta_t, \sigma, \alpha, i_a, m$`))`;
    $feasible$`.append(`$[c, v, \text{convergence}]$`)`;
$index \leftarrow$"NaN", $converged \leftarrow []$, $values \leftarrow []$, $labels = [N, L, H]$
for $result \in feasible$ **do**
  converged.`append(`$result[2]$`)`
  values.`append(`$result[1]$`)`
if $False \in converged$ **then**
  return *"NaN", "NaN"*
else
  $index \leftarrow values$`.argmax();`
  $i_a \leftarrow labels[index]$ $max \leftarrow feasible[index]$ $c_t \leftarrow max[0]$
  return i_a, c_t
Function `solve_bellman(BellmanEquation):`
  Function `solve_bellman(BellmanEquation)` is primarily a while loop of `update_bellman(`v_{grid} `, BellmanEquation)` based on the method described in Stachurski [22]. If errors for the policy solution are not within tolerances at loop exit, $convergence \leftarrow False$
  return $c, v, convergence$
Function `update_bellman(`v_{grid}`, BellmanEquation):`
  Function `update_bellman(`v_{grid}`, BellmanEquation)` iterates over an array of grid points to maximize and update their value function results, v_{grid}, through optimization of consumption, c_{grid}, very similar to the method described in *[22].*
  return c_{grid}, v_{grid}
Function `u(`c, σ`):`
  Function `u(`c, σ`)` returns utility of consumption σ
  if $\sigma \neq 1$ **then**
    return $\frac{c^{1-\sigma}-1}{1-\sigma}$
  else
    return $\ln(c)$
Function `f(`k_t, α, $TechTable$`):`
  Function `f(`k, α, `TechTable)`returns agent income with
  $income \leftarrow []$
  for $i \in TechTable$`.keys()` **do**
    $entry \leftarrow \alpha * k^{TechTable[i][0]} - TechTable[i][1]$
    $income$`.append(`$entry$`)`
  return `max(`$income$`)`

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

Bibliography

[1] Barber, C.B., Dobkin, D.P., Huhdanpaa, H.: The quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software **22**(4), 469–483 (1996), https://doi.org/10.1145/235815.235821

[2] Bellman, R.: On the theory of dynamic programming. Proceedings of the National Academy of Sciences **38**(8), 716–719 (1952), https://doi.org/10.1073/pnas.38.8.716

[3] DeAngelis, D.L., Diaz, S.G.: Decision-making in agent-based modeling: A current review and future prospectus. Frontiers in Ecology and Evolution **6**, 237 (2019), https://doi.org/10.3389/fevo.2018.00237

[4] Espinosa Barcenas, O.U., Quijada Pioquinto, J.G., Kurkina, E., Lukyanov, O.: Surrogate aerodynamic wing modeling based on a multilayer perceptron. Aerospace **10**(2), 149 (2023), https://doi.org/10.3390/aerospace10020149

[5] Garibay, V.M.: nnbellman (2025), URL https://github.com/vmgaribay/nnbellman

[6] Garzón, A., Kapelan, Z., Langeveld, J., Taormina, R.: Machine learning-based surrogate modeling for urban water networks: Review and future research directions. Water Resources Research **58**(5), e2021WR031808 (2022), https://doi.org/10.1029/2021WR031808

[7] Garín, J., Lester, R., Simms, E.: Intermediate macroeconomics (2021), URL https://sites.nd.edu/esims/textbook/

[8] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org

[9] Grignard, A., Taillandier, P., Gaudou, B., Vo, D.A., Huynh, N.Q., Drogoul, A.: Gama 1.6: Advancing the art of complex agent-based modeling and simulation. In: Boella, G., Elkind, E., Savarimuthu, B.T.R., Dignum, F., K.Purvis, M. (eds.) PRIMA 2013: Principles and Practice of Multi-Agent Systems, pp. 117–131, Springer Berlin Heidelberg (2013), ISBN 978-3-642-44927-7

[10] Han, D., Kim, J., Kim, J.: Deep pyramidal residual networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 5927–5935 (2017), https://doi.org/10.48550/arXiv.1610.02915

[11] Huang, V.: pytorch-template (2020), URL https://github.com/victoresque/pytorch-template, accessed: 2024-03-04

[12] Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global optimization **13**, 455–492 (1998), https://doi.org/10.1023/A:1008306431147

[13] Joppan, N.T.: Modelling Poverty Alleviation Strategies Using Resilience Thinking. Master's thesis, University of Amsterdam (2021), URL https://scripties.uba.uva.nl/search?id=record_30354

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17

[14] Jäger, G.: Replacing rules by neural networks a framework for agent-based modelling. Big Data and Cognitive Computing **3**(4) (2019), ISSN 2504-2289, https://doi.org/10.3390/bdcc3040051
[15] Kuriksha, A.: An economy of neural networks: Learning from heterogeneous experiences. PIER Working Paper No. 21-027 (2021), https://doi.org/10.2139/ssrn.3973697
[16] Ma, C., Zhu, B., Xu, X.Q., Wang, W.: Machine learning surrogate models for landau fluid closure. Physics of Plasmas **27**(4) (2020), https://doi.org/10.1063/1.5129158
[17] Maliar, L., Maliar, S., Winant, P.: Will artificial intelligence replace computational economists any time soon? CEPR Discussion Paper No. DP14024 (2019), URL https://ssrn.com/abstract=3464569
[18] Philipp, G.: The nonlinearity coefficient - a practical guide to neural architecture design. ArXiv (2021), https://doi.org/10.48550/arXiv.2105.12210
[19] Richetti, J., Diakogianis, F.I., Bender, A., Colaço, A.F., Lawes, R.A.: A methods guideline for deep learning for tabular data in agriculture with a case study to forecast cereal yield. Computers and Electronics in Agriculture **205**, 107642 (2023), ISSN 0168-1699, https://doi.org/10.1016/j.compag.2023.107642
[20] Saltelli, A.: Making best use of model evaluations to compute sensitivity indices. Computer Physics Communications **145**(2), 280–297 (2002), ISSN 0010-4655, https://doi.org/10.1016/S0010-4655(02)00280-1
[21] Sobol, I.M.: Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates. Mathematics and Computers in Simulation **55**(1), 271–280 (2001), ISSN 0378-4754, https://doi.org/10.1016/S0378-4754(00)00270-6, the Second IMACS Seminar on Monte Carlo Methods
[22] Stachurski, J.: Economic Dynamics, second edition: Theory and Computation. MIT Press (2022), ISBN 9780262544771
[23] Su, J., Cheng, H., Guo, H., Huang, R., Peng, Z.: An approximate quadratic programming for efficient bellman equation solution. IEEE Access **7**, 126077–126087 (2019), https://doi.org/10.1109/ACCESS.2019.2939161
[24] Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., Contributors, S..: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods **17**, 261–272 (2020), https://doi.org/10.1038/s41592-019-0686-2
[25] Zhang, G., Li, L., Nado, Z., Martens, J., Sachdeva, S., Dahl, G.E., Shallue, C.J., Grosse, R.: Which algorithmic choices matter at which batch sizes? insights from a noisy quadratic model (2019), https://doi.org/10.48550/arXiv.1907.04164

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97557-8_17