# Benchmarking Neural Networks for American Option Pricing

*Anna Bauer*
*Department of Statistics and Applied Probability, University of California, Santa Barbara*

## Abstract

Machine learning techniques have revolutionized the field of financial engineering by providing accurate and efficient methods for pricing American options. This research project aims to explore the effectiveness of deep learning algorithms in accurately pricing American options. The project is divided into two schemes: Scheme I employs a sequence of neural networks, while Scheme II utilizes a single aggregate neural network to eliminate time discretization. By testing various combinations of neural network hyperparameters in both schemes, we seek to optimize the accuracy and computational speed for pricing nine different Put and Call options. Our results are compared against existing efficient algorithms, such as polynomial regression and random forest, as documented in [5]. Based on the analysis of optimal hyperparameters that enhance the accuracy of machine learning-based American option pricing, we identify the top five solvers (hyperparameter sets) in Scheme I and the top two solvers in Scheme II. These solvers are benchmarked and reproducible, serving as reference points for future comparisons with prior studies.

*Research Partner: Aidan Baker*
*Project Advisors: Mike Ludkovski and Cosmin Borsa*

# 1. Background

American-style financial contracts provide an option holder the right, but not the obligation, to transact at any point before a specified expiration date. Thus, when an investor purchases an option, there is a set period of time by which the option must be exercised; otherwise, it will expire worthless. The most famous example is the American Put, where the buyer must determine whether it is profitable to *sell* the underlying asset for the specified strike value or to wait until the Put potentially becomes more valuable. We also consider the American Call, where the option holder decides whether *buying* the underlying asset for the specified strike is profitable.

American options differ from other options in that the holder has the flexibility to exercise (stop at a given time and collect the reward) their option at any time before the expiration date. Because American options can be exercised continuously (i.e., at any instant), they are very difficult to value. As an approximation, modelers utilize Bermudan-style versions. Bermudan options are similar to American options in the sense that they allow the holder to exercise early, but different in that they instead allow modelers to specify discrete time steps. Each time step represents a point where the option holder can exercise. As the number of time steps increases, the Bermudan option begins to act more and more like an American option.

The option is considered "in the money" if it has positive intrinsic value and "out of the money" otherwise. The intrinsic value measures the difference between the strike price, which is the price the contract allows the option holder to buy or sell for, and some measure for the price of the underlying assets. A basket Put option, for example, involves the average asset prices, so the intrinsic value is the difference between the strike price and the average price of the assets. Intrinsic value is positive if the strike price is above the price measure for a Put and the reverse for a Call. Options should only be exercised if they are in the money; otherwise, their intrinsic value is zero, so the option holder should continue with the chance of the value increasing in the future.

To maximize the expected payoff, we consider an optimal stopping problem with a finite number of stopping opportunities, creating a discrete-time problem. We use an algorithm called the Longstaff-Schwartz method (see Section 2) to find the optimal stopping time $\tau^*$ and its associated payoff. When solving the optimal stopping problem, one compares the immediate payoff and the future expected value of the option, known as the continuation value. The continuation value at a given time step estimates the discounted average of option values at the following time step across all paths. In this project, we look to find the timing function for when an option should be exercised to achieve optimal payoff, as well as the parameters that most accurately approximate the payoff of a given option.

The price of an option is given by its expected payoff $v(t_0, S_0)$ from Equation 1. The option value serves as the summary statistic that we use for benchmarking, and at time step $t_n$ it is given by Equation 1,

$$v(t_n, \cdot) = \max[q(t_n, \cdot), h(t_n, \cdot)] \tag{1}$$

where $q(t_n, \cdot)$ is the continuation value in present dollars, and $h(t_n)$ is the payoff if the option holder were to exercise at that time step. $v(t_0)$ measures the initial price of an option, which is the expected payoff across all possible trajectories from the time the contract begins, given the optimal exercise strategy. The timing value $T(t_n, \cdot)$ marks the difference between the continuation value and the immediate payoff, as shown in Equation 2.

$$T(t_n, \cdot) = q(t_n, \cdot) - h(t_n, \cdot). \tag{2}$$

When the timing value is negative, one should exercise the option at $t_n$ as the expected future payoff is no longer greater than the immediate payoff. To estimate the timing values of American options, we utilize neural networks, which are powerful machine learning algorithms that can handle large quantities of data and complex, non-linear relationships between input and output variables. Their flexibility and ability to recognize patterns makes them especially well-suited for this task.

## 2. Methods

The project is divided into two schemes that use the Longstaff-Schwartz method [4], an algorithm used to solve an optimal stopping problem with a single binary decision: stop at time $t$, or continue in hopes of a higher reward at some point in the future. The ultimate goal of an optimal stopping problem is to maximize some expected payoff. One must first know the payoffs at each time step to find the optimal stopping time. Next, one should estimate the continuation values by using the Longstaff-Schwartz method. After sequentially estimating pathwise continuation values, they are compared with immediate rewards to compute pathwise optimal stopping policies.

The Longstaff-Schwartz method [4] utilizes backward induction and one-step-ahead continuation values with a sequence of regression emulators to estimate current continuation values. To estimate the continuation maps sequentially, we simulate using a Monte Carlo method with $j$ independent paths of the stock prices. The continuation values are estimated using backward induction between consecutive time steps $t_n$ and $t_{n+1}$. Namely, the pathwise continuation value $q(t_n, x_n^k)$ at time $t_n$, is calculated as shown in Equation 3, with $\Delta t = t_{n+1} - t_n$ being the time interval between $n$ and $n+1$.

$$q(t_n, x_n^k) = \begin{cases} h(t_n, x_n^k) & \text{if } \hat{q}(t_n, x_n^k) \leq h(t_n, x_n^k) \text{ and } h(t_n, x_n^k) \geq 0 \\ e^{-r\Delta t} q(t_{n+1}, x_{n+1}^k) & \text{otherwise,} \end{cases} \tag{3}$$

The first condition holds true because the continuation value is equal to the immediate payoff at exercise time. Otherwise, the pathwise continuation value is recursively updated from the next time step, as shown in the second condition. Since the continuation value represents a future payoff, it must be discounted by $e^{-r\Delta t}$ to obtain the present value at step $t_n$.

Regression emulators sequentially take in the one-step-ahead prices of the underlying asset(s) to

produce an estimate of the current continuation value at each time step, forming the continuation maps. The maps can then be used to evaluate the optimal stopping time $\tau_n^*$ and its associated payoff. For a given path, if the immediate payoff at a given time step $\tau_n$ is positive and greater than the estimated continuation value, then the option holder should stop at $\tau_n$.

In Scheme I, we derive $\hat{q}(t_n, \cdot)$, the estimated continuation value at time $t_n$, by training a sequence of regression emulators —one for each $t_n$— on the active (in the money) paths of an asset, looping backward from $N-1$ to 1. In each iteration, the current continuation value is updated to be used as an input for the computation of the next continuation value [2]. Scheme II follows the same methodology with one key difference; instead of sequentially approximating the timing value for each time step, as in Scheme I, Scheme II approximates the timing value with a single approximation across all time steps (see Section 3.3). It is noted that both schemes work with discrete-time Markov processes by using the Bermudan-style option formulation.

## 2.1. Neural Networks for LSM

The regressions for $\hat{q}$ can be formed through various algorithms, such as linear regression, tree-based regressors, and neural networks. For this project, neural networks are used. Neural networks are an excellent choice as regression emulators due to their flexibility and implementation, which does not depend on the problem dimension. A neural network consists of interconnected nodes, called perceptrons, organized into layers. The number of nodes and layers can be adjusted, and larger networks with more nodes and layers can capture more intricate relationships. The activation function of the network determines whether a perceptron should be activated or not based on the weighted sum of its inputs. If the perceptron is activated, it will produce an output signal that is propagated to the next layer. Activation functions are used to introduce non-linearity into the model, allowing it to learn complex relationships between the input data and output predictions. This becomes especially helpful when considering multidimensional options with more than one underlying asset.

The goal of training a neural network is to minimize the loss function by adjusting its weights [6]. Loss is a measure of how well the network performs; for this project, mean squared error (MSE) and mean squared logarithmic error (MSLE) are used to measure loss. Optimization minimizes the loss by finding the set of weights that produce the best performance on the training data. Optimization algorithms use the loss (and gradients of the loss) to adjust the weights iteratively. The training data is separated into subsets called batches, enabling the optimizer to update the weights more frequently. This reduces the memory required to train the network. The training data is also passed through the network multiple times, or epochs, to adjust the weights of the perceptrons and improve the accuracy of the predictions made by the network.

Each of the variables mentioned above are considered hyperparameters, which are parameters set prior to training and do not change as the algorithm learns. Hyperparameters are contrasted with the neural network weights and biases (i.e., parameters), which are learned by the respective

optimizer. Hyperparameters control the behavior of the network and how the algorithm learns from the training data. They are chosen by the modeler and can greatly affect model performance. An important part of developing algorithms is to tune these hyperparameters.

Most hyperparameters have an infinite number of possible values. For example, there could be anywhere from one to infinite layers in a neural network. Non-numerical hyperparameter choices, such as loss, have a discrete number of typical methods used to measure them, but theoretically, infinite methods could be developed. Thus, both numerical and categorical hyperparameters impose the impossible challenge of testing every possible hyperparameter value. Therefore, we choose a discrete set of values, often called a grid space, to test the algorithm on. Our grid space is developed based on the the successes of previous studies. There exist methods to automate the process of selecting hyperparameters from the grid space; however, for our problem, it is more appropriate to select hyperparameters manually.

Thus, our method is to manually test wide ranges of hyperparameter values for the neural networks, focusing on the epochs, nodes, layers, activation functions, and batches. The goal is to find a set of hyperparameters that yield the highest option price. Since there are thousands of hyperparameter combinations, we deduce the parameter sets that provide the best results and test those. If a value provides a worse price than the value tested before it, it is assumed that the best value lies closer to the prior value. From there, we test hyperparameters within gradually narrowing ranges. When testing one hyperparameter, all other hyperparameters are fixed to ensure that any variability in output price is not the result of variability in any hyperparameters other than the one being tested. For example, if optimal number of nodes is tested, all other hyperparameters will be kept the same for each node number variation.

There are some limitations to this testing method, which are further discussed in Section 3.1. One example is that this testing method keeps the number of nodes the same in each layer. In reality, the best result could be to vary the number of nodes across layers. In the future, more of these possibilities should be tested.

## 2.2. Benchmarks

An important goal of this project is to benchmark our results and compare them with the performance of various known efficient emulators, such as random forest algorithms. This allows us to provide a standard baseline for evaluating the effectiveness of different emulators and enables future researchers to identify the strengths and weaknesses of their approaches. As mentioned in [5], the methods in many published works regarding option pricing using Regression Monte Carlo are difficult, if not impossible, to reproduce. Thus, we aim to expand the results of [5], adding additional verifiable results and algorithms for future research.

With the goal of finding the hyperparameters most suitable for a wide variety of options, our analysis is performed on nine various Put and Call payoffs. The underlying model dynamics use geometric Brownian motion (GBM), a continuous-time stochastic process that assumes that the

logarithm of an asset price follows a Brownian motion. GBM is most commonly used for Black Scholes models. Each model is also defined by its dimension, number of steps, payoff, and problem geometry:

- Dimension ($d$): number of assets in the option contract
- Steps ($K$): number of time steps (stopping points)
- Payoff: Put, basket average Puts, max-Calls
- Problem geometry: symmetric and asymmetric asset dynamics. Symmetric models have independent, identically distributed (i.i.d) assets while asymmetric models' assets are correlated and not i.i.d. since they have different volatilities.

Table 1 lists all models considered. We use the M1-M8 models from [5] to remain consistent and build upon existing results, citing them with the same naming scheme for clarity. Some of these models appear in other publications as well. Two additional models, labeled M10 and M11, are sourced from [2]. Models M10 and M11 are the basis for our initial hyperparameter tuning, while the rest are only used to test the final subset of hyperparameter choices.

*Table 1: Specification of model parameters for the nine models tested. Models M1-M8 are the same as models M1-M8 in [5] and models M10 and M11 are sourced from [2].*

| Model | Dim | Steps | Payoff | Notes |
|---|---|---|---|---|
| M1 | 1 | 25 | Put | Classic 1D at-the-money Put |
| M2 | 1 | 25 | Put | Same as M1 but out-of-the-money |
| M3 | 2 | 25 | Basket Put | 2D symmetric in-the-money basket Put |
| M4 | 2 | 9 | Max Call | 2D symmetric in-the-money max-Call |
| M6 | 3 | 9 | Max Call | 3D symmetric out-of-the-money max-Call |
| M7 | 5 | 9 | Max Call | 5D symmetric at-the-money max-Call |
| M8 | 5 | 9 | Max Call | 5D asymmetric out-of-the-money max-Call |
| M10 | 1 | 20 | Put | 1D at-the-money Put |
| M11 | 2 | 9 | Max Call | 2D symmetric out-of-the-money max-Call |

As shown in Table 1, all defined models involve either basket Puts, max-Calls, or single-dimensional payoffs. Max-Call options compare the strike and market prices of the highest priced asset in the option bundle. Thus, the payoff of exercising a max-Call at a given time step is given by Equation 4, where $S$ is the vector of asset prices, $S^{(i)}$ is the price of the $i$-th asset at time $t$, and $K$ is the strike price paid by the option holder. This payoff is discounted by $e^{-rt}$, giving us its present value. Basket Put payoffs involve the strike price and the weighted average of all $d$ assets in the basket, as given by $h_{Put}(t, S)$ in Equation 5, where $S_t^{(i)}$ is the market price of asset $i$ at time $t$ and $K$ is again the strike price.

$$h_{MaxCall}(t, S) = e^{-rt} \max[0, \max_i S^{(i)} - K] \tag{4}$$

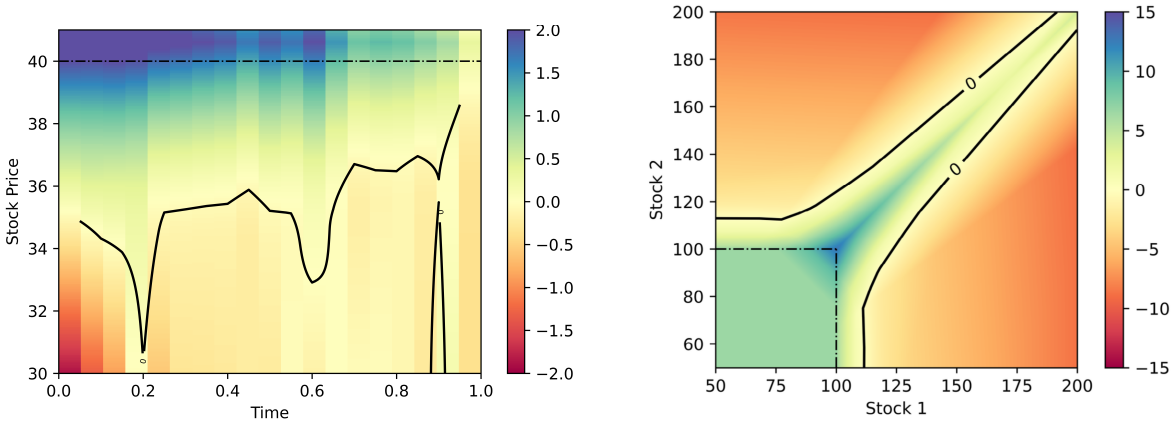
$$h_{Put}(t, S) = e^{-rt} \max[0, K - \frac{1}{d} \sum_{i=1}^{d} S_t^{(i)}] \tag{5}$$

With the chosen models defined, we can test them on a variety of hyperparameter sets. Only the M10 and M11 models are used to test the hyperparameters because they encompass both the max-Call and the Put without the complexity of considering all nine models. As we narrow our choices, we develop "solvers", which represent the hyperparameter configurations that perform the best on the M10 and M11 models. The resulting solvers are then tested on all nine models for their performance in pricing and computation time, yielding benchmarks for future research.

## 2.3. Illustrations

Figure 1 depicts the timing values of the M10 and M11 models in Scheme I. The timing value $T(t_n, \cdot)$, as defined in Equation 2, is represented by the colored shading in each plot. The option holder should exercise the option and collect the reward when the timing value is negative. The black contour lines represent the optimal stopping boundary, along which the timing value is zero. In the left panel of Figure 1, the dotted line represents the strike price. When the stock price lies above the contour line, one should continue in hopes of a higher reward in the future. Below the contour line, one should stop and exercise immediately as the future expected payoff of the option is less than the immediate payoff. Notice that the contour line is not smooth and the color values abruptly between time steps. This is because the timing values are computed from a sequence of neural network approximations, so the values between time steps are not continuous.

*Figure 1: Contour plots showing the timing value.* Left: *M10, the one-dimensional at-the-money Put across all time steps.* Right: *M11, the two-dimensional out-of-the-money max-Call at a fixed time step. In both panels, the optimal stopping boundaries are marked by the black lines and the color represents the timing value. Both models use the baseline parameters listed in Table 2*



The right panel of Figure 1 depicts the timing values of a two-dimensional max-Call option with the two assets as the axis variables. This illustration depicts a snapshot of the timing values at a given time step, so the time variable is left out of this plot. Here, one should exercise the option if the market prices lie outside the two contour lines. As in Figure 1, the dotted lines represent the predetermined strike price for the option. The right panel of Figure 1 gives a smoother approximation of the timing values with stopping boundaries that are far more distinct than in the left panel. This approximation for the M11 max-Call option's timing values is far more plausible than the approximation for the Put.

# 3. Results

## 3.1. Scheme I

Various hyperparameter values are tested on the M10 one-dimensional Put and the M11 two-dimensional Call. As described in Section 2, we begin with a wide range of possibilities and gradually narrow the range based on the performance of each consecutive test. A baseline parameter set, sourced from [2], is utilized as the starting range for hyperparameter testing. From there, the goal is to find hyperparameter values that significantly improve upon the baseline parameter set, as given in Table 2.

*Table 2: Parameters chosen as a baseline for testing based on prior experiments in [2].*

| Activation | Initializer | Optimizer | Loss | Epochs | Batches | Nodes | Layers |
|---|---|---|---|---|---|---|---|
| ELU | TruncNorm(0,0.5) | ADAM | MSE | 50 | 64 | 25 | 4 |

Here we test the option price, $v(t_0, S_0)$, as defined in Equation 1, with the above hyperparameters. Each variable in Table 2 is tested except for the initializer and the number of batches. To set the initial weights for optimization, a truncated normal distributed initalizer with a mean of 0 and a standard deviation of 0.5 is used for each test case. The normal distribution is bounded to ensure that the weights and biases are not too large. While the initializer can significantly impact the model's performance, this hyperparameter is left for future research. The number of batches, or the number of subsets the training data is separated into, is also left for future research. Thus, every solver listed in the tables below has an initializer equal to TruncNorm(0,0.5) and a batch size of 64.

The number of epochs, or times the training data is passed through the network, can significantly impact performance, so testing begins there. Values above and below the baseline of 50 are tested, both leading to a significant decrease in output price. Thus, 50 is left as the best number of epochs among the test cases, and it is determined that epochs are a weakly significant factor in the performance of the neural networks. With 50 epochs established, the number of nodes (perceptrons in the neural network) is tested. 60 nodes yield the highest price, so a constant value of 60 nodes per layer is used when testing the number of layers. The number of nodes drastically changes the output price, so this hyperparameter is found to be strongly significant

in the performance of the neural networks. Finally, the optimal number of layers in the neural network is tested. Three and five layers are tested, which are below and above our baseline, respectively. Neither three nor five layers improved the output price, so it is determined that four layers produce the best result among those tested. Varying the number of layers does not change the price significantly, especially for the M10 Put.

*Table 3: Option prices (in dollars) as a function of epochs, nodes, and layers. We report the M10 Put prices and M11 Call prices as a function of the neural network hyperparameters in the first three columns.*

| Epochs | Nodes | Layers | M10 Put Price | M11 Call Price |
|---|---|---|---|---|
| 50 | 25 | 4 | 2.4045 | 17.3905 |
| 25 | 25 | 4 | 2.4089 | 17.3151 |
| 75 | 25 | 4 | 2.4013 | 17.3196 |
| 50 | 40 | 4 | 2.3995 | 17.3230 |
| 50 | 50 | 4 | 2.3941 | 17.3726 |
| 50 | 60 | 4 | 2.4014 | 17.4280 |
| 50 | 70 | 4 | 2.4004 | 17.3455 |
| 50 | 60 | 3 | 2.4056 | 17.3430 |
| 50 | 60 | 5 | 2.3949 | 17.2543 |

Once the best number of epochs, nodes, and layers is established, the testing process shifts to activation functions, optimizers, and loss functions. We again start with the baseline parameters, as listed in Table 2. Variations of activation functions cause the most significant changes in output price in both M10 and M11 models out of any variable tested in Table 3 and Table 4. We test exponential linear unit (ELU), rectified linear unit (RELU), Gaussian error linear unit (GELU), sigmoid, and hyperbolic tangent (Tanh) activation functions. For the M11 Call, RELU consistently provides the highest output price among activation functions. The M10 Put price varies less among activation functions; however, ELU and GELU generally provide the highest prices. We also test an adaptive moment estimation (ADAM) [3] and a stochastic gradient descent optimizer with a learning rate of 0.01 and a momentum of 0.9 (SGD(0.01,0.9)). ADAM generally performs better with higher output prices. We test the methods of mean squared error (MSE) and mean squared log error (MSLE) for loss measurement. MSE is found to outperform MSLE.

## 3.2. Comparing to Prior Results

Once we have sufficiently tested various neural network hyperparameters, we eliminate poor-performing hyperparameter sets and focus solely on high-performing hyperparameter sets. We choose five sets of hyperparameters that produce the highest output prices and set those as benchmarks for future testing. Each hyperparameter set is denoted as a "solver" with a solver name between S1 and S5. This allows us to test various models with neural network parameters that are shown to perform well. The solvers are listed in Table 5.

Since the solvers in Table 5 are top performers in Table 3 and Table 4 for the M10 and M11 option models, we test them on each model in Table 1 so that the outcomes can be compared with

*Table 4: Option prices (in dollars) as a function of activation functions, optimizers and loss. We report the M10 Put prices and M11 Call prices as a function of the neural network hyperparameters in the first three columns.*

| Activation | Optimizer | Loss | M10 Put Price | M11 Call Price |
|---|---|---|---|---|
| ELU | ADAM | MSE | 2.4046 | 16.8917 |
| RELU | ADAM | MSE | 2.4027 | 17.2380 |
| GELU | ADAM | MSE | 2.4108 | 17.1562 |
| Sigmoid | ADAM | MSE | 2.4009 | 16.1195 |
| Tanh | ADAM | MSE | 2.3992 | 16.8625 |
| ELU | SGD(.01,.9) | MSE | 2.4086 | 16.1970 |
| RELU | SGD(.01,.9) | MSE | 2.3994 | 17.0622 |
| GELU | SGD(.01,.9) | MSE | 2.4039 | 16.7792 |
| Sigmoid | SGD(.01,.9) | MSE | 2.2748 | 14.3615 |
| Tanh | SGD(.01,.9) | MSE | 2.4113 | 15.9954 |
| ELU | SGD(.01,.9) | MSLE | 2.4064 | 16.1192 |

*Table 5: Specification of the 5 top-performing benchmark solvers. We summarize activation function, optimizer, loss function, epochs, nodes, batches, and layers for each solver chosen.*

| Solver | Activation | Optimizer | Loss | Epochs | Nodes | Batches | Layers |
|---|---|---|---|---|---|---|---|
| S1 | RELU | ADAM | MSE | 5 | 16 | 64 | 4 |
| S2 | ELU | ADAM | MSE | 10 | 60 | 64 | 3 |
| S3 | ELU | ADAM | MSE | 5 | 25 | 64 | 4 |
| S4 | RELU | ADAM | MSLE | 5 | 64 | 64 | 4 |
| S5 | ELU | ADAM | MSE | 3 | 16 | 64 | 4 |

those in [5]. The output prices are given in Table 6, with the best value shown in bold for each model.

Solver performance varies from model to model; however, in general, S1 tends to perform the best, producing the highest output price for seven of the nine models. S4 also performs well, producing three of the highest prices. S2 and S3 perform poorly overall.

In comparison to the neural network emulator (S5-NNet) in [5], a single-layer neural network with 20/40/50 nodes, our solvers perform similarly price-wise. However, our neural network solvers performed relatively well compared to the other solvers in [5] Table 3, as shown by the 'mlOSP Range' column in Table 6. Our solvers especially improved upon the results of the linear model and random forest solvers for Put options. For Call options, our solvers did not make any significant improvements in pricing performance.

Running time is also a consideration for choosing solvers; see Table 7. The running time is based on the following machine specifications: 2.3 GHz Dual-Core Intel Core i5, 8 GB 2133 MHz LPDDR3 memory, and Intel Iris Plus Graphics 640 1536 MB, utilizing Python 3.8.13. Since Scheme I implements a sequence of neural networks, the running times listed represent the total time it took to run through the sequence and output a final price. This way, the running times between aggregate

*Table 6: Bermudan option prices (in dollars) from the benchmarks in Table 5. Models are sourced from [5] and [2], and solvers S1-S5 represent the neural network hyperparameter sets from Table 5. mlOSP Range represents the pricing performance range among solvers in [5].*

| Model | S1 | S2 | S3 | S4 | S5 | mlOSP Range |
|---|---|---|---|---|---|---|
| M1 | **2.23** | 2.21 | **2.23** | 2.20 | 2.20 | [2.07,2.31] |
| M2 | **1.06** | **1.06** | **1.06** | 1.05 | 1.05 | [1.01,1.10] |
| M3 | **1.41** | 1.39 | 1.40 | 1.40 | **1.41** | [1.23,1.46] |
| M4 | **20.92** | 20.86 | 20.76 | **20.92** | 20.82 | [20.21,21.48] |
| M6 | 10.95 | 10.86 | 10.81 | **10.98** | 10.75 | [10.85,11.15] |
| M7 | **25.41** | 25.40 | 25.01 | 25.37 | 25.00 | [25.10,25.84] |
| M8 | 11.04 | 10.86 | 10.97 | **11.15** | 10.91 | [11.39,11.81] |
| M10 | **2.42** | 2.41 | 2.40 | 2.40 | 2.41 | N/A |
| M11 | **17.11** | 16.94 | 16.13 | 17.01 | 15.98 | N/A |

and sequential neural networks can be compared.

The running times for each model vary quite a bit between solvers; however, less so than in [5] because we only include variations of neural networks instead of the wide variety of regression algorithms utilized in [5]. The solvers in Table 5 improve upon S5-NNet timewise by quite a bit for M3, M4, M6, M7, and M8. Solver S5, for example, computes the price for M8 at a speed nearly 14 times faster than S5-NNet. In comparison with the other solvers in [5], our solvers are middle-of-the-pack for computational time performance. They significantly improve upon the adaptive sequential batching (S8-ASDA) solver for every model tested. All five solvers perform especially well compared to the solvers in [5] for Call options. S5 performs at the highest speeds for every model tested by far. S1 and S4 perform similarly, still generally improving upon the speeds of S5-NNet in [5]. S2 and S3 remain the poorest performing solvers for both time and accuracy, although they improve from S5-NNet for M3, M4, and M6-8.

*Table 7: Bermudan option computation time (in seconds) from the benchmarks in Table 5. Models are sourced from [5] and [2], and solvers represent our top five performing neural network hyperparameter sets. mlOSP Range represents the timing performance range among solvers in [5].*

| Model | S1 | S2 | S3 | S4 | S5 | mlOSP Range |
|---|---|---|---|---|---|---|
| M1 | 125.33 | 183.86 | 139.23 | 133.00 | **121.24** | [9.2,206.7] |
| M2 | 102.53 | 108.54 | 96.08 | 95.01 | **89.31** | [9.4,208.4] |
| M3 | 133.35 | 174.13 | 128.12 | 124.91 | **119.35** | [13.4,422.7] |
| M4 | 53.12 | 86.52 | 53.32 | 51.07 | **45.81** | [2.0,521.8] |
| M6 | 40.38 | 61.94 | 42.77 | 45.86 | **36.77** | [3.9,2023.8] |
| M7 | 60.04 | 95.56 | 63.90 | 57.42 | **46.21** | [4.5,2838.4] |
| M8 | 33.87 | 45.82 | 34.17 | 33.22 | **32.84** | [3.5,2011.8] |
| M10 | 100.42 | 145.11 | 102.39 | 101.39 | **89.00** | N/A |
| M11 | 123.54 | 219.04 | 136.35 | 129.99 | **105.49** | N/A |

## 3.3. Scheme II

In Scheme II, the sequence of neural network regressors is replaced with a single surrogate that takes in both the prices of the assets and the time step value, outputting an approximation of the option timing value. In Scheme I, the sequential neural networks only involve one time period each, so they don't include $t_n$ as an input. This marks the main difference between the sequence of neural networks and the aggregate network. Aggregating the sequence of regression emulators into one single surrogate removes the time discretization involved in Scheme I and enables far greater capabilities. With Scheme II, any amount of time steps in the Bermudan option can be chosen, which brings us closer to the goal of pricing American options.

The aggregate neural network is trained similarly to the sequential neural networks using the Longstaff-Schwartz method. Three hundred thousand possible paths are simulated for the underlying assets, and the neural network is trained sequentially, starting at the final time step and looping backward. The network is retrained on both new and old data to prevent catastrophic forgetting during sequential training. A $\rho$ value of 2 for a 2:1 ratio of old to new data is chosen for Scheme II testing because it proved successful in [2]. The rest of the baseline parameters are chosen the same way as in Section 3.1, as shown in Table 8.

*Table 8: Parameters chosen as a baseline for testing in Scheme II.*

| Activation | Initializer | Optimizer | Loss | Epochs | Nodes | Batches | Layers | $\rho$ |
|---|---|---|---|---|---|---|---|---|
| ELU | TruncNorm(0,0.5) | ADAM | MSE | 5 | 25 | 64 | 4 | 2 |

The initializer, optimizer, loss function, number of batches, and $\rho$ values were not changed during Scheme II testing because in Scheme I, changes to these values did not yield a significant difference in performance. Instead, the focus moves to finding the optimal number of epochs, nodes, and layers and finding the best activation function.

As shown in Table 9, 10 epochs and 32 nodes consistently produce the highest prices. Changes to the number of epochs and nodes yield the most significant differences in output prices, while the number of layers and the activation function make a relatively insignificant impact. RELU and ELU activation functions perform similarly, and 4-6 layers perform well across both the Put and the Call. In similar fashion to Scheme I, these "solvers" are chosen as the best performers and benchmarked on seven additional option models.

The emulator denotes the difference between Scheme I and Scheme II; as introduced in Section 2, Scheme I implements a sequence of neural networks (denoted NN-seq in Table 10) to estimate the option price as a function of the immediate reward and the continuation value at each time step. Each neural network in the NN-seq emulator takes in only the prices of the assets and does not include the time step as an input variable. S6 and S7 represent the two top-performing hyperparameter configurations with an aggregate neural network (NN-agg) as the regression emulator.

*Table 9: Option prices (in dollars) as a function of activation function, epochs, nodes, and layers. We report the M10 Put prices and M11 Call prices as a function of the neural network hyperparameters in the first four columns.*

| Epochs | Nodes | Layers | Activ | M10 Put Price | M11 Call Price |
|---|---|---|---|---|---|
| 5 | 25 | 4 | RELU | 2.3380 | 16.8932 |
| 10 | 25 | 4 | RELU | 2.4075 | 17.2380 |
| 20 | 25 | 4 | RELU | 2.3639 | 17.1154 |
| 25 | 25 | 4 | RELU | 2.3476 | 17.1065 |
| 10 | 16 | 4 | RELU | 2.3802 | 17.1656 |
| 10 | 32 | 4 | RELU | 2.4112 | 17.2391 |
| 10 | 64 | 4 | RELU | 2.3284 | 16.8542 |
| 10 | 128 | 4 | RELU | 2.3122 | 16.8467 |
| 10 | 32 | 4 | RELU | 2.4112 | 17.2391 |
| 10 | 32 | 6 | RELU | 2.4120 | 17.2395 |
| 10 | 32 | 8 | RELU | 2.4115 | 17.2387 |
| 10 | 32 | 4 | RELU | 2.4112 | 17.2391 |
| 10 | 32 | 6 | ELU | 2.4065 | 17.2256 |

*Table 10: Specification of the Scheme II top-performing benchmark solvers, with reference to the S1 solver from Scheme I. We summarize regression emulator, activation function, optimizer, loss, epochs, nodes, batches, and layers for each solver chosen.*

| Solver | Emulator | Activation | Optimizer | Loss | Epochs | Nodes | Batches | Layers |
|---|---|---|---|---|---|---|---|---|
| S1 | NN-seq | RELU | ADAM | MSE | 5 | 16 | 64 | 4 |
| S6 | NN-agg | RELU | ADAM | MSE | 10 | 32 | 64 | 4 |
| S7 | NN-agg | ELU | ADAM | MSE | 10 | 32 | 64 | 6 |

The Scheme II S6 and S7 solvers are now tested on the same models defined in Table 1. This allows for a comparison between an aggregate neural network's performance and a sequence of neural networks' performance in pricing options. We expect Scheme II solvers to perform similarly to Scheme I solvers, increasing algorithm capabilities without losing performance. Scheme II removes time discretization, incorporating the time value as an input parameter, which should improve performance for periods where the sequential neural network performed poorly. The use of old data for training should also help the network make decisions based on past data. However, as shown in Table 6, the Scheme I solvers already perform well compared to the solvers in [5], so the Scheme II solvers are unlikely to make drastic improvements from Scheme I.

The highest output prices, as emphasized in Table 11, are equally spread between S1 and S6, with S7 as the clear underperformer. The range in price performance among solvers is also quite narrow, with most models showing only a few cent difference between solvers. This aligns well with our prediction that, because S1 performed near the top of the performance range in [5], Scheme II would not be able to make any significant improvements.

One concerning result regards the M4 and M10 models. In these cases, the aggregate neural networks perform worse than the sequential neural network, meaning that additional training on old

*Table 11:* Left: *Bermudan option prices (in dollars) from the benchmarks in Table 10, in reference to S1 from Table 5. S6 and S7 represent the top two performing aggregate neural network hyperparameter sets from the testing in Table 9.* Right*: respective computation times (in seconds).*

| Model | S1 | S6 | S7 |
|---|---|---|---|
| M1 | **2.23** | 2.22 | 2.22 |
| M2 | **1.06** | **1.06** | 1.05 |
| M3 | 1.41 | **1.42** | 1.40 |
| M4 | **20.92** | 20.78 | 20.85 |
| M6 | 10.95 | 10.97 | **10.98** |
| M7 | **25.41** | **25.41** | 25.39 |
| M8 | 11.04 | **11.06** | 11.05 |
| M10 | **2.42** | 2.41 | 2.41 |
| M11 | 17.11 | **17.23** | 17.22 |

| Model | S1 | S6 | S7 |
|---|---|---|---|
| M1 | 125.33 | **471.91** | 477.53 |
| M2 | 102.53 | 469.72 | **464.71** |
| M3 | 133.35 | **488.52** | 498.68 |
| M4 | 53.12 | 231.27 | **225.42** |
| M6 | 40.38 | **161.65** | 165.30 |
| M7 | 60.04 | **243.12** | 251.64 |
| M8 | 33.87 | **137.46** | 146.82 |
| M10 | 100.42 | **436.91** | 448.90 |
| M11 | 123.54 | 472.54 | **463.12** |

data and the removal of time discretization decreases the algorithm's performance. This could be due to a number of factors, primarily the number of epochs and the value for $\rho$. It is assumed that a 2:1 ratio of old to new data would be sufficient in mitigating catastrophic forgetting; however, other values of $\rho$ are not tested. A hypothesis could be made that a higher value of $\rho$ would further mitigate catastrophic forgetting and thus yield higher option prices. Additionally, the number of epochs is not adjusted proportionately from Scheme I to Scheme II, which could be a cause for the pricing performance decline in the M4 and M10 models.

Solvers are also tested for their performance in computational efficiency, which denotes the amount of time it takes to compute the option price. In relation to [5], Scheme I produces relatively fast compute times among solvers, with S1 and S5 producing the fastest compute times. Compute time in Scheme I is measured as the total time it takes to loop through each sequential neural network and produce an average payoff among trajectories. In Scheme II, compute time is measured as the total time it takes for the aggregate network to iterate backward through each time step and produce an average price among trajectories. The same machine and specifications are utilized for both Scheme I and Scheme II, as defined in Section 3.2. Thus, both Scheme I and Scheme II timing estimates measure the time it takes for the same machine to produce the final option price.

As shown in Table 11, S1 from Scheme I consistently outperforms both S6 and S7 from Scheme II in terms of computational efficiency. S1 typically computes the option prices around four times faster than S6 and S7. S6 again outperforms S7 in terms of computation time, however, when compared to S1, S6 performs poorly. Naturally, it is expected that Scheme II requires more compute time than Scheme I because we retrain the network on old data in Scheme II. In Scheme II, the primary method to reduce computation time is to reduce the ratio of old to new data, $\rho$:1. However, the optimal value for $\rho$ leaves an optimization problem between computation time and accuracy; reducing $\rho$ may negatively impact the performance of the option pricing in Table 11.

The biggest indicator of computational efficiency, in general, is epochs. As the number of epochs

increases, the number of times the weights are updated increases, directly affecting compute time. Since S1 has half the epochs of S6 and S7, this could be a cause for its faster computation times.

# 4. Conclusion

As shown in sections 3.1 and 3.3, S1 is presented as our best solver in terms of accuracy and computational efficiency. S1 outperforms both Scheme I and Scheme II solvers, showing that at this stage of research, the sequence of neural networks is performing just as well as or better than the single aggregate neural network. S1 also performs well when compared with the results of [5], improving upon the computational time and accuracy for many of the models tested. Additional testing would yield an improvement from S1 since there are infinite possibilities for hyperparameter combinations and only a finite number of configurations tested. Upon comparison between various hyperparameter sets, it appears that the number of nodes and the number of epochs holds the most significant weight of the parameters tested in determining the computational time and price accuracy. When comparing activation functions, we find it best to use RELU or ELU; however, there is no significant difference in performance between the two. The number of layers in the network also makes insignificant impact on the price. In our experiments, there is no material difference between Scheme I and Scheme II performance, though the latter is around four times slower than Scheme I. In the future, we would like to improve Stage II's computation time and accuracy as a step toward the approximation of American options.

A straightforward comparison between our solvers and future research is enabled by this project's utilization of previously benchmarked option models and the benchmarking of new successful solvers. The results are fully reproducible, with the code given in [1], the solver specifications defined in Section 3, and the model specifications defined in [5].

# References

[1] Anna Bauer and Aidan Baker. Neural Networks for American Option Pricing (GitHub). https://github.com/abauer726/nn_options, 2023.

[2] Cosmin Borsa. Ph.D. Advancement Report. 2022.

[3] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, 2017.

[4] F. Longstaff and E. Schwartz. Valuing American Options by Simulation: A Simple Least-Squares Approach. *The Review of Financial Studies*, 14(1):113–147, 2001.

[5] Mike Ludkovski. mlOSP: Towards a Unified Implementation of Regression Monte Carlo Algorithms. 2022.

[6] Michael A Nielsen. *Neural Networks and Deep Learning*, volume 25. Determination Press San Francisco, CA, USA, 2015.