Reinforcement Learning Assignment 3: Proximal Policy Optimisation

Gaspard Gaches [1]

Abstract

Proximal Policy Optimization (PPO) is one of the latest and most widely used state-of-the-art methods in deep reinforcement learning (RL), introducing an improvement of the earlier Trust Region Policy Optimization (TRPO) proposed by the same authors (Schulman et al., 2017). In this work, we seek to understand the individual impact of specific components of the standard implementation of the PPO-Clip algorithm in the CartPole-v1 environment. Our results show that not all hyperparameters have the same impact on agent performance, and that some specific code-level optimisations commonly used in PPO and deep RL in general do not bring about any significant improvements in CartPole.

1. Introduction

In the policy-based subfield of reinforcement learning (RL), actor-critic methods are widely considered the most efficient as they blend the value-based and policy-based approaches, which are respectively low-variance and low-bias, therefore offering a satisfactory middle ground in the bias-variance trade-off (Plaat, 2023). Trust region optimisation seeks to further reduce the high variability in policy updates by introducing a specialised loss function with an added constraint that limits the step size during optimisation (Plaat, 2023). A notable disadvantage of TRPO (Schulman et al., 2017) is that it uses second-order derivatives, making it a computationally complex method (Plaat, 2023). PPO was designed to bring sizeable improvements in speed and sample efficiency compared to TRPO, but research has shown that the clipping mechanism introduced by PPO is not always the main driver of its superior performance (Engstrom et al., 2020). In this study, we aim to verify this hypothesis in the CartPole environment.

[1]Leiden University. Correspondence to: Gaspard Gaches <s4645251@vuw.leidenuniv.nl>.

2. Background

In the paper introducing TRPO (2017), Schulman et al. prove the guarantee of improvement in the expected return of the next iteration policy $\eta(\pi_\theta)$ with unknown parameters θ over the expected return of the current policy $\eta(\pi_{\theta_k})$ at iteration k [1]. Both TRPO and PPO share the same goal: finding the optimal policy parameters $\theta = \theta_{k+1}$ so that the step size is maximized while staying within our trust region. Schulman et al. (2017) introduce the expected return $\eta(\pi_\theta)$ as follows:

$$\eta(\pi_\theta) = \eta(\pi_{\theta_k}) + \sum_s \rho_{\pi_\theta}(s) \sum_a \pi_\theta(a|s) A_{\pi_{\theta_k}}(s,a) \quad (1)$$

The expected return $\eta(\pi_\theta)$ depends on ρ_{π_θ}, which is the discounted visitation frequency under the next iteration policy:

$$\rho_{\pi_\theta}(s) = \sum_t \gamma^t P_{\pi_\theta}(s_t = s)$$

The issue here is that ρ_{π_θ} depends on the next iteration policy, the same we are trying to estimate, so a workaround is needed: Schulman et al. introduce a local approximation of the expected return, which ignores the change in the state visitation density brought by the new policy π_θ:

$$L_{\pi_{\theta_k}}(\pi_\theta) = \eta(\pi_{\theta_k}) + \sum_s \rho_{\pi_{\theta_k}}(s) \sum_a \pi_\theta(a|s) A_{\pi_{\theta_k}}(s,a) \quad (2)$$

Note that if we set $\theta = \theta_k$ in the formula above, the second term of $L_{\pi_{\theta_k}}$ becomes zero. This is because under the current policy π_{θ_k}, the mean of the advantages also generated from the current policy is effectively 0:

$$\sum_a \pi_{\theta_k}(a|s) A_{\pi_{\theta_k}}(s,a) = 0 \quad \forall s$$

This axiom allows for the following equations, provided that L and η are differentiable at θ_k:

$$L_{\pi_{\theta_k}}(\pi_{\theta_k}) = \eta(\pi_{\theta_k}),$$

[1]While some papers switch between multiple notations within the same work (e.g., alternating between policy-based and parameter-based notations), we will stick to a consistent notation throughout this paper to avoid confusion.

$$\nabla_\theta L_{\pi_{\theta_k}}(\pi_\theta)\Big|_{\theta=\theta_k} = \nabla_\theta \eta(\pi_\theta)\Big|_{\theta=\theta_k} \quad \forall k$$

The gradient equality above says that for a given k, as long as θ stays sufficiently close to θ_k, if a step is taken that improves $L_{\pi_{\theta_k}}$, it will also improve η. Two popular techniques were invented that strive to keep θ close to θ_k while maximising $L_{\pi_{\theta_k}}$, namely KL divergence penalty and clipping. In this study, the clipping method was chosen, as it simplifies the implementation of the trust region algorithm. We recall our goal from equation 2:

$$\max_\theta \sum_s \rho_{\pi_{\theta_k}}(s) \sum_a \pi_\theta(a|s) A_{\pi_{\theta_k}}(s,a)$$

$$\text{subject to clipping constraint } g(\cdot)$$

Note that maximising this objective requires sampling actions from the new policy, which we do not have access to, since we are trying to estimate its parameters θ. We only have actions sampled from the current policy, therefore we use importance sampling to approach our objective:

$$\begin{aligned}\sum_a \pi_\theta(a|s) A_{\pi_{\theta_k}}(s,a) &= \sum_a \pi_{\theta_k}(a|s) \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_{\theta_k}}(s,a) \\ &= \mathbb{E}_{a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_{\theta_k}}(s,a)\right]\end{aligned} \tag{3}$$

The quantity that is clipped is the weighted advantage inside the expectation. The ϵ hyper-parameter controls how far θ is allowed to get from θ_k - in other words, the size of our trust region - and we define L^{CLIP}:

$$\begin{aligned}&L^{CLIP}(s,a,\theta_k,\theta) = \\ &\quad \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_{\theta_k}}(s,a),\ g\left(\epsilon, A_{\pi_{\theta_k}}(s,a)\right)\right)\end{aligned} \tag{4}$$

$$\text{where } g(\epsilon, A) = \begin{cases}(1+\epsilon)A & \text{if } A \geq 0 \\ (1-\epsilon)A & \text{if } A < 0\end{cases}$$

Our objective is maximising the surrogate objective subject to this clipping constraint with regards to θ, therefore we introduce the PPO policy loss:

$$\mathcal{L}_{policy}(\theta) = -\mathbb{E}_{s\sim\rho_{\theta_k},\, a\sim\pi_{\theta_k}}\left[L^{CLIP}(s,a,\theta_k,\theta)\right] \tag{5}$$

The value loss used in PPO is typically the same as in other actor-critic algorithms:

$$\mathcal{L}_{value} = \mathbb{E}_{s\sim\rho_{\theta_k},\, a\sim\pi_{\theta_k}}\left[A_{\pi_{\theta_k}}(s,a)^2\right] \tag{6}$$

Algorithm 1 PPO-Clip (OpenAI)

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k \in 0, 1, 2, ...$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy π_{θ_k} in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates $\hat{A}_t$ based on the current value function V_{ϕ_k}.

Update the policy by maximising the PPO-Clip objective:

$$\begin{aligned}\theta_{k+1} = \arg\max_\theta \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \\ \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_{\theta_k}}(s,a),\ g(\epsilon, A_{\pi_{\theta_k}}(s,a))\right)\end{aligned}$$

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k|T} \sum_{\tau\in\mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2$$

end for

return Parameters θ

3. Methods

The general structure of our algorithm was inspired by OpenAI's PPO-Clip algorithm (Algorithm 1). In this section, we present our implementation of that base algorithm, our hyperparameter tuning process, an ablation study that gauges the individual impact of specific features of the algorithm on agent performance, and various code-level optimisations that were brought to the algorithm in order to study their effect on average returns and return variance.

Experimental setup and implementation

Environment

All experiments in this article were carried out in the CartPole-v1 environment from OpenAI's Gymnasium library, which was inspired from the cart-pole problem described by Barto, Sutton, and Anderson. Quoting the original description of CartPole on the Gymnasium website: *"A pole is attached by an un-actuated joint to a cart, which moves along a frictionless track. The pendulum is placed upright on the cart and the goal is to balance the pole by applying forces in the left and right direction on the cart."*. Therefore, the action space for this problem is simply $\{\text{left}, \text{right}\}$. The state space is 4-dimensional: the position of the cart, the velocity of the cart, the angle of the pole and the angular velocity of the pole. The maximum return

that can be achieved in CartPole-v1 is 500, therefore we will consider an algorithm optimal if its average return remains constantly at 500 over the course of training with a variance as close as possible to zero after the algorithm has converged (with convergence ideally being achieved as fast as possible).

IMPLEMENTATION

The PPO-Clip algorithm was implemented using the PyTorch deep learning framework. We opted for a two-headed neural network with an actor head and a policy head: the input and hidden layer(s) are shared, and the two distinct output layers have sizes 2 (the cart can only go left or right) and 1 (the critic only outputs the value of the state it was fed), respectively. Our algorithm features a surrogate objective (Equation 4) that clips the probability ratio introduced in Equation 3. No entropy bonus was incorporated into the loss, nor did we weigh the policy and value losses: the main loss was simply computed as the sum of the policy loss (Equation 5) and the value loss (Equation 6). The gradients obtained through backpropagating that loss were applied to the neural network using an Adam optimiser. A fixed number of experiences (4096) were collected for each iteration of the loop in Algorithm 1. Gradient updates were performed on mini-batches, and across several sweeps of the full batch, for each iteration, and the experiences were shuffled at the beginning of each epoch to break temporal correlation. Eight different seeds were used in all experiments presented in this study to prevent spurious discrepancies due to randomness.

3.1. Hyperparameter tuning

SEARCH SPACE

Not all hyperparameters have the same effect on the performance of our agent. This can easily be seen by performing a grid search on two given parameters and observing the variance of the measured outcome along either dimension. Four hyperparameters were tuned via two different grid searches: first the learning rate and the neural network structure, and second the gradient update epochs and the mini-batch size. The ranges explored for these four parameters are shown in Table 1.

Table 1. Hyperparameter search space used during grid search

Hyperparameter	Search space
Learning rate	$[5 \times 10^{-5}, 5 \times 10^{-4}]$
Network structure	$\{(1, 32), (1, 64), (2, 32), (2, 64)\}$
Epochs per update	$\{1, 5, 10, 25\}$
Mini-batch size	$\{32, 64, 128, 256\}$

Network structure: (hidden layers, neurons per layer)

Due to constraints on time and computational resources, each configuration in these two grid searches was tested in 5×10^5 steps, each with 8 different seeds, which we considered enough to gauge differences between configurations given that the best converged in around just 10^5 steps.

BOOTSTRAPPING

Because visualising 32 curves is far from practical, the performance of the different configurations was instead assessed by bootstrapping from a fixed number of test runs performed after the 5×10^5 steps training. The trained agent was put through 100 test runs for all seeds for a total of 800 test runs, then 1000 samples (Engstrom et al., 2020 used a similar number) of size 800 as well were drawn with replacement from this vector of test run returns. Two bootstrapped metrics were computed from these samples: the average return and the standard deviation of these returns, along with their standard errors.

3.2. Ablation study

In section 2, we laid out the logic behind the probability ratio clipping mechanism, which prevents large gradient updates and improves the stability of our algorithm. The baseline PPO-Clip algorithm that we introduce in this section uses clipping (for computing the policy loss) by default; therefore, we sought to visualise the impact of clipping on agent performance by disabling clipping and comparing the resulting performance against that of our baseline algorithm. The clipping threshold is fixed at $\epsilon = 0.2$ (this ϵ value is a common choice in research) in all experiments that used clipping in this study.

In a previous study revisiting design choices in PPO algorithms, Hsu et al., 2020 found that normalising advantages in fact did not always result in improved performance regardless of the task on which the agent was trained. In that study, agents were trained on various MuJuCo tasks, and in some of them, the agent that did not normalise advantages clearly outperformed the agent that normalised advantages. Our baseline PPO-Clip algorithm normalises advantages by default, as is common practice in deep reinforcement learning. We sought to measure the impact that normalising advantages has on agent performance in the CartPole-v1 environment by removing that feature from our baseline algorithm and comparing the resulting performance against that of the baseline algorithm.

3.3. Feature engineering

Engstrom et al., 2020 sought to study the source of the superior performance of PPO over older reinforcement learning algorithms. They showed that the gain in efficiency of PPO brought by simplifying the complex mathematics behind TRPO (Schulman et al.) did not turn out to be the main

driver in performance gain, and emphasis was put on the importance of various code optimisations which were shown to have a greater impact on agent performance. We sought to gauge the impact of two of those code-level optimisations studied by Engstrom et al., 2020 on agent performance in the CartPole-v1 environment, namely observation normalisation and gradient norm clipping. We used a gradient norm clipping limit of 0.5, which is the value used in Engstrom et al., 2020. Each of these two features was independently added to the base algorithm in separate trials, and their individual effects on agent performance were evaluated relative to the baseline.

4. Experiments

4.1. Hyperparameter tuning

The results of the two grid searches described in section 3.1 are presented in the Appendix. In Table 2, we see that the row corresponding to a learning rate $\alpha = 2.5 \times 10^{-4}$ has the highest average returns across all network structures, with $\alpha = 5 \times 10^{-4}$ also producing very good results. Lower α values yielded noticeably worse returns, meaning that our agent was learning too slowly from its experiences to be able to converge. It is also worth noting that the variance of the average returns is higher along the α dimension than along the network structure dimension, which shows that the learning rate has more effect on agent performance than the structure of the neural network.

The results presented in Table 3 come from the second grid search of the best update epochs and mini-batch size values, which was carried out with $\alpha = 2.5 \times 10^{-4}$ and a neural network consisting of 1 hidden layer with 32 neurons - it only made sense to pick the simplest network structure out of those with the best average returns from Table 2 for the sake of efficiency. We make a similar observation here as in Table 2: the variance of the average returns is higher along the update epochs dimension than along the mini-batch size dimension, therefore, the number of update epochs is a hyperparameter of greater importance.

In Figure 1, we compare four of the best parameter combinations found in the previous subsection. The indigo curve converged the fastest, and is barely visible afterwards as its variance was effectively 0 throughout training, except for small instabilities around 8×10^5 steps. The green curve has the same learning rate and update epochs, but an additional hidden layer and a larger mini-batch size: the average return (solid curve) is noticeably more unstable than the purple curve, and also has higher variance as the shaded regions show. The cyan curve featuring a smaller learning rate and number of neurons (which is the configuration the second grid search was carried out with) also shows impressive stability in the second half of training, but struggled to stabilise in the first half compared to the indigo curve.

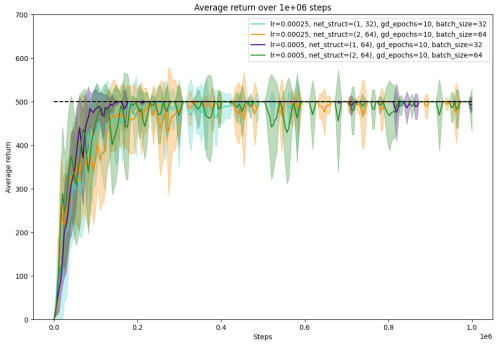

Figure 1. Four of the best hyperparameter combinations from two grid searches. The solid curves are the average returns, and the shaded regions are delimited by ± 1 standard deviation. The black dashed horizontal line $y = 500$ helps visualise the maximum return achievable in CartPole-v1.

4.2. Ablation study

The results of the ablation study described in section 3.2 can be visualised in Figure 2. The orange curve corresponding to the agent without clipping could not converge over a million steps, and shows very high variance. The indigo curve corresponding to the agent without advantage normalisation shows superior learning speed over the first 50,000 steps or so, but struggles to achieve stable convergence afterwards, and also has higher variance than the baseline cyan curve. Overall, the baseline agent clearly had a superior performance as it converges in less than 2×10^5 steps and is much more stable.

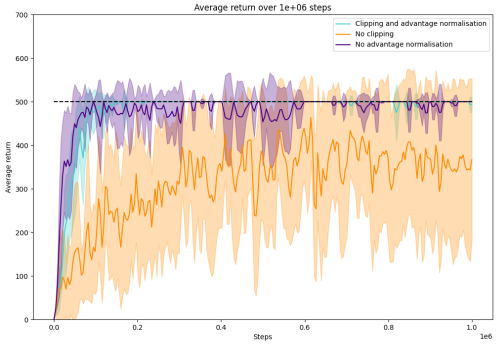

Figure 2. Performance of the PPO algorithm after ablation of clipping and advantage normalisation against our baseline algorithm featuring both. The benefits of clipping on average return and return variance become obvious. Disabling advantage normalisation incurs instabilities in average returns and higher variance.

4.3. Feature engineering

The results of the feature engineering tests described in section 3.3 can be visualised in Figure 3. The orange curve corresponding to the agent that normalised observations could not converge over a million steps, and gave an even worse performance than clipping ablation as the curve stays mostly flat below 200 on average over the course of training. The indigo curve corresponding to the agent with gradient norm clipping shows similar convergence speed as the baseline, although we notice a few minor instabilities constantly appearing throughout training, and higher variance than the baseline.

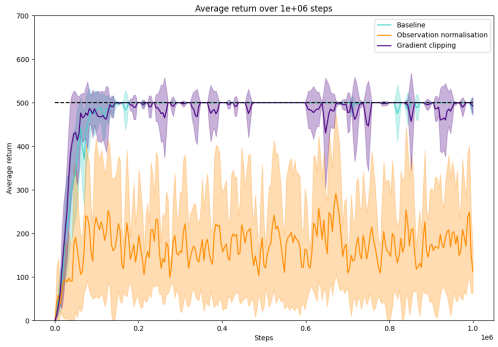

Figure 3. Performance of the PPO algorithm after separately adding observation normalisation and gradient norm clipping against our baseline algorithm featuring neither. Observation normalisation gave a terrible performance, showing almost no learning at all and high variance. Gradient norm clipping shows minor instabilities compared to the baseline.

5. Discussion

Four hyperparameters were tuned through two grid searches and the performance was measured with bootstrapped estimates of the mean and the variance of returns from rollouts performed post-training. The learning rate was found to have a greater impact on the average return than the structure of the neural network, and the α value that produced the highest returns across all network structures matched one of the most commonly used values in the literature. More complex network structures did not show any improvement over the simpler ones, probably because of the simplicity of the CartPole environment relative to more complex ones such as MuJuCo environments. The number of epochs per policy update showed a larger effect on the average return than the mini-batch size. A larger number of epochs was correlated with better average returns, the reason being that the optimal θ_{k+1} introduced in Algorithm 1 is approximated through gradient descent, and the quality of our approximation depends on the number of gradient descent steps, hence the number of epochs. However, more epochs also means longer runtime, and the little gain in overall performance (across all mini-batch sizes) obtained from pushing it beyond 10 epochs in our experiments was not worth the longer runtime. Indeed, the closer our approximation attempts to get to the true optimal θ_{k+1}, the longer it takes to make significant progress toward that target. Smaller mini-batch sizes were associated with higher average returns, which means that our algorithm benefits from more frequent updates that capture finer-grained patterns in the data.

The ablation of the probability ratio clipping mechanism involved in the calculation of the policy loss showed that it is one of the main drivers of the superior performance of PPO relative to older algorithms, as our agent was unable to converge without clipping after a million-step training. This result proves the negative effect on agent performance caused by large gradient updates. Disabling advantage normalisation had a largely smaller negative effect on performance. Ultimately, it was found that advantage normalisation does indeed improve the stability of average returns and helps reduce variance in the CartPole environment.

The addition of observation normalisation hugely undermined the performance of our agent, even more than the ablation of clipping. This came as somewhat of a surprise, because in CartPole, although the cart position and pole angle dimensions are bounded, the other two velocity dimensions are not, which means the linear velocity of the cart and angular velocity of the pole can technically tend toward infinity. Given that, we would expect observation normalisation to bring more stability, as states are fed to the critic network to calculate advantages which are used in both the policy and value losses. Gradient norm clipping did not bring about any significant improvement either, showing a slightly higher variance and less stable average returns compared to the baseline.

Ultimately, our best configuration (cyan curve in Figure 3) displayed significant improvements in convergence speed and post-convergence stability compared to deep Q-learning, for which a baseline trial is shown in Figure 4.

Due to computational and time constraints, all our experiments were run on only eight different random seeds, thus future work could seek to replicate them with more seeds in order to definitively rule out random effects on performance. Future research may explore the effect of the probability clipping ratio and the gradient norm clipping ratio on agent performance by tuning them with a grid search, in order to explore potential improvements in convergence speed and stability.

6. Conclusion

The primary goal of this study was to evaluate the individual impact of specific components of the PPO-Clip algorithm on agent performance within the CartPole-v1 environment. By tuning four key hyperparameters, we assessed their individual influence on agent performance and identified those with the most significant impact. This process led to a configuration that enabled fast convergence and stable behaviour with near-zero return variance post-convergence. We found that clipping the probability ratio notably improved both average return and stability, and that the latter were also improved by normalising advantages, although to a lesser extent. In contrast, neither gradient clipping nor observation normalisation resulted in any measurable improvements.

References

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. Implementation matters in deep policy gradients: A case study on PPO and TRPO. *CoRR*, abs/2005.12729, 2020. URL https://arxiv.org/abs/2005.12729.

Hsu, C. C., Mendler-Dünner, C., and Hardt, M. Revisiting design choices in proximal policy optimization. *CoRR*, abs/2009.10897, 2020. URL https://arxiv.org/abs/2009.10897.

Plaat, A. *Deep Reinforcement Learning*. Springer Nature, 2023.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. Trust region policy optimization, 2017. URL https://arxiv.org/abs/1502.05477.

Appendix

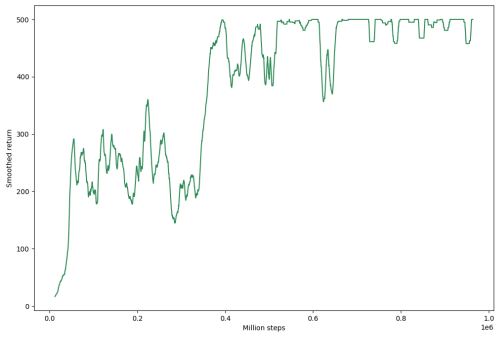

Figure 4. Baseline DQN returns averaged over two trials on Cartpole-v1.

Table 2. First grid search: learning rate and network structure.

Pair	Mean	SE	Std	SE
α=5.10^{-5}, ns=(1,32)	274.29	3.96	112.91	2.68
α=5.10^{-5}, ns=(1,64)	370.15	3.3	96.6	2.68
α=5.10^{-5}, ns=(2,32)	435.56	3.87	112.89	3.35
α=5.10^{-5}, ns=(2,64)	371.0	4.07	115.52	2.55
α=10^{-4}, ns=(1,32)	387.35	5.48	152.44	2.73
α=10^{-4}, ns=(1,64)	452.36	2.34	64.95	5.41
α=10^{-4}, ns=(2,32)	490.45	1.74	49.95	8.16
α=10^{-4}, ns=(2,64)	470.21	2.72	80.9	4.85
α=$2.5.10^{-4}$, ns=(1,32)	495.01	1.71	48.95	8.61
α=$2.5.10^{-4}$, ns=(1,64)	489.27	1.74	50.86	7.95
α=$2.5.10^{-4}$, ns=(2,32)	457.95	3.68	108.38	4.48
α=$2.5.10^{-4}$, ns=(2,64)	495.08	1.67	48.55	8.64
α=5.10^{-4}, ns=(1,32)	444.05	3.55	100.48	3.71
α=5.10^{-4}, ns=(1,64)	494.98	1.8	48.96	9.2
α=5.10^{-4}, ns=(2,32)	449.13	3.35	93.33	4.0
α=5.10^{-4}, ns=(2,64)	495.01	1.75	48.85	8.95

Network structure (ns): (hidden layers, neurons per layer)

Table 3. Second grid search: update epochs and mini-batch size.

Pair	Mean	SE	Std	SE
epochs=1, batch=32	252.55	4.74	135.47	2.27
epochs=1, batch=64	278.93	5.0	138.27	2.52
epochs=1, batch=128	210.92	5.9	168.78	3.33
epochs=1, batch=256	202.89	6.45	178.21	3.31
epochs=5, batch=32	466.76	2.51	74.37	5.06
epochs=5, batch=64	370.94	4.68	127.21	2.85
epochs=5, batch=128	447.69	3.26	92.4	4.22
epochs=5, batch=256	323.1	4.32	125.58	2.4
epochs=10, batch=32	495.09	1.69	48.47	8.79
epochs=10, batch=64	495.07	1.69	48.61	8.68
epochs=10, batch=128	466.21	2.57	72.0	5.28
epochs=10, batch=256	449.25	3.93	113.09	4.61
epochs=25, batch=32	495.05	1.75	48.67	8.89
epochs=25, batch=64	466.38	3.0	89.17	4.6
epochs=25, batch=128	495.09	1.81	48.37	9.34
epochs=25, batch=256	476.16	2.47	68.91	5.92