A APPENDIX

A.1 NOVAS: DERIVATION AND IMPLEMENTATION DETAILS

Given the optimization problem defined in (3) we calculate the gradient with respect to the sampling distribution parameter ρ

$$\nabla_\rho \ln \int S(F(x))f(x;\rho)\mathrm{d}x = \frac{\int S(F(x))\nabla_\rho f(x;\rho)\mathrm{d}x}{\int S(F(x))f(x;\rho)\mathrm{d}x}, \tag{5}$$

$$= \frac{\int S(F(x))\nabla_\rho \ln f(x;\rho) f(x;\rho)\mathrm{d}x}{\int S(F(x))f(x;\rho)\mathrm{d}x}, \tag{6}$$

$$= \frac{\mathbb{E}[S(F(x))\nabla_\rho \ln f(x;\rho)]}{\mathbb{E}[S(F(x))]}, \tag{7}$$

where the second equality is obtained using the log trick. The exponential family distribution is characterized by the probability distribution function

$$f(x;\rho) = h(x)\exp(\rho^{\mathrm{T}}T(x) - A(\rho)), \tag{8}$$

where $h(x)$ is the base measure whose formula depends on the particular choice of distribution, $T(x)$ is the vector of sufficient statistics and $A(\rho) = \ln\{\int h(x)\exp(\rho^{\mathrm{T}}T(x))\mathrm{d}x\}$ is the log partition. The gradient with respect to log distribution can then be calculated as

$$\nabla_\rho \ln f(x;\rho) = T(x) - \nabla_\rho A(\rho). \tag{9}$$

A Gaussian $\mathcal{N}(\mu, \Sigma)$ is fully defined by its mean and covariance, but one can choose to optimize (3) only over the mean μ, and sample using a fixed covariance. This results in the following parameters of the distribution:

$$T(x) = \Sigma^{-1/2}x, \qquad \nabla_\rho A(\rho) = \Sigma^{-1/2}\mu, \tag{10}$$

and for $\rho = \mu$ the gradient can be calculated as

$$\nabla_\mu \ln \mathbb{E}[S(F(x))f(\mu)] = \frac{\mathbb{E}[S(F(x))(x-\mu)]}{\mathbb{E}[S(F(x))]}. \tag{11}$$

One reason for choosing to optimize over the mean only is the simplicity of the resulting update expression, as well as numerical reasons, since applying gradient descent on the covariance matrix can lead to non-positive definiteness if the initial values or the learning rate are not chosen carefully. An intermediate solution between using a fixed covariance matrix and its gradient descent-based update law is assuming a diagonal covariance matrix and calculating each element of the diagonal via a simple weighted empirical estimator. The resulting update scheme is given in Alg. 1. Note that we also investigated using the full gradient descent-based update rule for the covariance, as well as the use of the Hessian of the objective function (3) and other techniques like momentum, line search, trainable optimization hyper-parameter values etc. to speed up convergence, but the results were inconclusive as to their additional benefit. We tested two different shape functions: $S(y;\kappa) = \exp(\kappa y)$, as well as a shape function suggested by Zhou & Hu (2014), namely $S(y;\kappa,\gamma) = (y - y_{\min})/\big(1+\exp(-\kappa(y-\gamma))\big)$, where $y_{\min}$ is the minimum of the sampled values and γ is the n-th biggest value of the assorted y values. For the first choice it is numerically advantageous to include the normalization step in the function definition and replace the exponential with the `softmax` function. The latter choice is a differentiable function approximating the level/indicator function used in CEM. Though both shape functions exhibited similar performance, we noticed that the former was slightly faster, and the latter lead to slightly more accurate results in the SPEN example (but not in the FBSDE example, where they were equivalent). All parameter values such as κ can be made trainable parameters of the network, though we did not notice an improvement in doing so. In fact, the algorithm seems to be quite insensitive to its parameter values including the learning rate α, with the sole exception of σ which does indeed affect its output significantly.

A.2 STOCHASTIC OPTIMAL CONTROL USING FBSDEs

In this section we introduce the deep FBSDE framework for solving PDEs and show that combining NOVAS with the deep FBSDE allows us to extend the capabilities of the latter framework

(Pereira et al., 2019b;a; Wang et al., 2019) in addressing Stochastic Optimal Control (SOC) problems. The mathematical formulation of a SOC problem leads to a nonlinear PDE, the Hamilton-Jacobi-Bellman PDE. This motivates algorithmic development for stochastic control that combine elements of PDE theory with deep learning. Recent encouraging results (Han et al., 2018; Raissi, 2018) in solving nonlinear PDEs within the deep learning community illustrate the scalability and numerical efficiency of neural networks. The transition from a PDE formulation to a trainable neural network is done via the concept of a *system of Forward-Backward Stochastic Differential Equations* (FBSDEs). Specifically, certain PDE solutions are linked to solutions of FBSDEs, which are the stochastic equivalent of a two-point boundary value problem and can be solved using a suitably defined neural network architecture. This is known in the literature as the *deep FBSDE* approach. In what follows, we will first define the SOC problem and present the corresponding HJB PDE, as well as its associated system of FBSDEs. The FBSDEs are then discretized over time and solved on a neural network graph.

Consider a SOC problem with the goal of minimizing an expected cost functional subject to dynamics:

$$\inf_{u\in\mathcal{U}[0,T]} J\big(u\big) = \inf_{u\in\mathcal{U}[0,T]} \mathbb{E}\bigg[\phi\big(x(T)\big) + \int_0^T l\big(x(t),u(t)\big)\,\mathrm{d}t\bigg], \tag{12}$$

$$\text{s.t.}\quad \mathrm{d}x(t) = f\big(x(t),u(t)\big)\,\mathrm{d}t + \Sigma\big(x(t),u(t)\big)\,\mathrm{d}w(t), \quad x(0)=\xi, \tag{13}$$

where $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$ are the state and control vectors respectively, $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is a non-linear vector-valued drift function, $\Sigma : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^{n\times v}$ is the diffusion matrix, $w \in \mathbb{R}^v$ is vector of mutually independent Brownian motions, $\mathcal{U}$ is the set of all admissible controls and $l : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ and $\phi : \mathbb{R}^n \to \mathbb{R}$ are the running and terminal cost functions respectively. Equation (13) is a controlled Itô drift-diffusion stochastic process.

Through the value function definition $V\big(x,t\big) = \inf_{u\in\mathcal{U}[0,T]} J\big(u\big)|_{x_0=x,t_0=t}$ and using Bellman's principle of optimality, one can derive the Hamilton Jacobi Bellman PDE, given by

$$V_t + \inf_{u\in\mathcal{U}[0,T]} \bigg[\frac{1}{2}\mathrm{tr}\big(V_{xx}\Sigma\Sigma^T\big) + V_x^T f\big(x,u\big) + l(x,u)\bigg] = 0, \quad V\big(x,T\big) = \phi\big(x\big), \tag{14}$$

where we drop explicit time dependencies for brevity, and use subscripts to indicate partial derivatives with respect to time and the state vector. The term inside the infimum operation is called the *Hamiltonian*:

$$\mathcal{H}\big(x,u,V_x,V_{xx}\Sigma\Sigma^T\big) \triangleq \frac{1}{2}\mathrm{tr}\big(V_{xx}\Sigma\Sigma^T\big) + V_x^T f\big(x,u\big) + l(x,u). \tag{15}$$

Given that a solution u^* to the minimization of $\mathcal{H}$ exists, the unique solution of (14) corresponds by virtue of the non-linear Feynman-Kac lemma (see for example Pardoux & Peng (1990)) to the following system of FBSDEs:

$$x(t) = \xi + \int_0^t f\big(x(t),u^*(t)\big)\,\mathrm{d}t + \int_0^t \Sigma\big(x(t),u^*(t)\big)\,\mathrm{d}w_t, \qquad \text{(FSDE)} \tag{16}$$

$$V(x(t),t) = \phi(x(T)) + \int_t^T l\big(x(t),u^*(t)\big)\,\mathrm{d}t - \int_t^T V_x^T(x(t),t)\Sigma\big(x(t),u^*(t),t\big)\,\mathrm{d}w, \quad \text{(BSDE)} \tag{17}$$

$$u^*(t) = \arg\min_u \mathcal{H}\big(x(t),u,V_x(x(t),t),V_{xx}(x(t),t)\Sigma(x(t),u)\Sigma(x(t),u)^T\big). \tag{18}$$

Here, $V(x(t),t)$ denotes an evaluation of $V(x,t)$ along a path of $x(t)$, thus $V(x(t),t)$ is a stochastic process (and similarly for $V_x(x(t),t)$ and $V_{xx}(x(t),t)$). Note that $x(t)$ evolves forward in time (due to its initial condition $x(0)=\xi$), whereas $V(x(t),t)$ evolves backwards in time, due to its terminal condition $\phi(x(T))$, thus leading to a system that is similar to a two-point boundary value problem. While we can easily simulate a forward process by sampling noise and then performing Euler integration, a simple backward integration of $V(x(t),t)$ would result in it depending explicitly on future values of noise, which is not desirable for a non-anticipating process, i.e., a process that does not exploit knowledge on future noise values. Two remedies exist to mitigate this problem: either back-propagate the conditional expectation of $V(x(t),t)$ (e.g., as in Exarchos & Theodorou (2018)), or forward-propagate $V(x(t),t)$ starting from an initial condition guess, compare its terminal value $V(x(T),T)$ to the terminal condition, and adjust the initial condition accordingly so that the terminal condition is satisfied approximately. For this forward evolution of the BSDE, the above system is discretized in time as follows:

$$x_{k+1} = x_k + f(x_k, u_k^*)\Delta t + \Sigma(x_k, u_k^*)\Delta w_k, \qquad x_0 = \xi, \qquad \text{(FSDE)} \tag{19}$$

$$V_{k+1} = V_k - l(x_k, u_k^*)\Delta t + V_{x,k}^T \Sigma(x_k, u_k^*)\,\Delta w_k, \qquad V_0 = \psi, \qquad \text{(BSDE)} \tag{20}$$

$$u_k^* = \arg\min_u \mathcal{H}\big(x_k, u, V_{x,k}, V_{xx,k}\Sigma(x_k, u)\Sigma(x_k, u)^T\big). \tag{21}$$

Here, Δw_k is drawn from $\mathcal{N}(0, \Delta t)$ and $\mathcal{H}$ is given by eq. (15). Note that for every sampled trajectory $\{x_k\}_{k=1}^K$ there is a corresponding trajectory $\{V_k\}_{k=1}^K$. Under the deep FBSDE controller framework, $V_0 = \psi$ and $V_{x,0}$ are set to be trainable parameters of a deep neural network that approximates $V_x\big(x(t), t\big)$ at every time step under forward-propagation, using an LSTM[3]. The terminal value of the propagated $V\big(x(t), t\big)$, namely $V(x(T), T)$, is then compared to $\phi\big(x(T)\big)$ to compute a loss function to train the network. Note that since the Hamiltonian can have any arbitrary non-linear dependence on the control, the resulting minimization problem (21) is generally non-covex and does not have a closed-form solution. Furthermore, it must be solved for each time step, and for utilization within the deep FBSDE controller framework, the non-convex optimizer must be differentiable to facilitate end-to-end learning. This makes NOVAS a good fit. The neural network architecture is shown in Fig. 3. Since the non-convex Hamiltonian minimization procedure is performed at every time step leading to a repeated use of NOVAS in the architecture, the ability to avoid unrolling the inner-loop computation graph is crucial.

A.3 FBSDE STOCHASTIC OPTIMAL CONTROL FOR AFFINE-QUADRATIC SYSTEMS

We now show how the previous state-of-the-art (Exarchos & Theodorou, 2018; Pereira et al., 2019b) deals with the problem of the Hamiltonian $\min$ operator by assuming a special structure of the problem. Specifically, they restrict the dynamics of eq. (13) to be affine in control, i.e., of the form $f(x, u) = F(x) + G(x)u$, and the cost in eq. (12) to be quadratic in control, i.e., $l(x, u) = q(x) + u^T R u$. In this case, and if $\Sigma(x, t)$ is not a function of u, one can perform explicit minimization of the Hamiltonian with respect to u in eq. (21) to find the optimal control:

$$u^* = -R^{-1} G^T V_x. \tag{22}$$

This is done by simply setting $\partial \mathcal{H} / \partial u = 0$ and solving for u. Substituted back into the HJB PDE, this yields a simplified expression without a $\min$ operator:

$$V_t + \frac{1}{2}\mathrm{tr}(V_{xx}\Sigma\Sigma^T) + V_x^T F + q - \frac{1}{2} V_x^T G R^{-1} G^T V_x = 0, \qquad V(x, T) = \phi(x).$$

Thus, for this restricted class of systems, the deep FBSDE neural neural network architecture does not require a numerical minimization operation over u at every time step, as in eq. (21). The cart-pole swing-up task of the next section is an example of a system that satisfies these restrictions. A similar closed-form solution exists for some cases of L^1-optimal control (Exarchos et al., 2018), as well as some differential games (Exarchos et al., 2019). While simplifying the problem significantly, this approach comes with an important caveat: several dynamical systems do not have a control-affine structure, and penalizing control energy ($u^T R u$) is not always meaningful in every setting.

A.3.1 CART-POLE SWING-UP PROBLEM

We define the state vector to be $X = \big[x, \theta, \dot{x}, \dot{\theta}\big]^{\mathrm{T}}$, where x represents the cart-position, θ represents the pendulum angular-position, $\dot{x}$ represents the cart-velocity, and $\dot{\theta}$ represents the pendulum angular-velocity. Let $u \in \mathbb{R}$ be the control force applied to the cart. The deterministic equations of motion for the cart-pole system are,

$$\ddot{x} = \frac{u + m_p \sin\theta (l\dot{\theta} + g\cos\theta)}{m_c + m_p \sin\theta}$$

$$\ddot{\theta} = \frac{-u\cos\theta - m_p l \dot{\theta}\cos\theta\sin\theta}{l(m_c + m_p \sin\theta)}$$

---

[3] In this work, we additionally use the same LSTM to predict a column of the Hessian $V_{xx}\big(x(t), t\big)$.

For our experiments, we consider the case where noise enters the velocity channels of the state. The stochastic dynamics therefore take the following form,

$$dX = d\begin{bmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \dfrac{m_p \sin\theta (l\dot{\theta} + g\cos\theta)}{m_c + m_p \sin\theta} \\ \dfrac{-m_p l\dot{\theta}\cos\theta\sin\theta}{l(m_c + m_p \sin\theta)} \end{bmatrix} dt + \begin{bmatrix} 0 \\ 0 \\ \dfrac{1}{m_c + m_p\sin\theta} \\ \dfrac{-\cos\theta}{l(m_c + m_p\sin\theta)} \end{bmatrix} u\, dt + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \tilde{\sigma} & 0 \\ 0 & \tilde{\sigma} \end{bmatrix} \begin{bmatrix} dw_1 \\ dw_2 \end{bmatrix}$$

The task is to perform a swing-up i.e. starting from an initial state of $X = \begin{bmatrix}0, 0, 0, 0\end{bmatrix}^{\mathrm{T}}$ at time $t_0 = 0$, reach the target state of $X = \begin{bmatrix}0, \pi, 0, 0\end{bmatrix}^{\mathrm{T}}$ by the end of the time horizon $t = T$. We consider $T =$ 1.5s with a time discretization step of $\Delta t = 0.02$s. Notice that the dynamics are affine in control, and selecting the running cost to be $l = u^T Ru$, minimization of the Hamiltonian with respect to u assumes a closed-form solution, namely that of eq. (22). This fact allows us to replace the $\min$ operator in favor of this solution (Pereira et al., 2019b). Here, we test NOVAS by avoiding this replacement. We consider a running and terminal cost matrix of $\mathrm{diag}(Q) = [0.0, 10.0, 3.0, 0.5]$ and the control cost matrix of $R = 0.1$. The cart-pole parameters considered are $m_p = 0.01\,kg$, $m_c = 1.0\,kg$, $l = 0.5\,m$, which are the mass of the pendulum, mass of cart, and length of the pendulum, respectively. For the noise standard deviation, $\tilde{\sigma} = 0.5$ was used. As far as the hyper-parameters for learning the deep FBSDE controller are concerned, we used a two-layer LSTM network as shown in Fig. 5(e) with hidden dimension of 16 in each layer, a batch size of 128, and trained the network using the Adam optimizer for 3500 iterations with a learning rate of $5e^{-3}$. For the NOVAS layer at every time step, we used 5 inner-loop iterations and 100 samples for both training and inference. A shape function of $S = \exp(\cdot)$, initial $\mu = 0$, and initial $\sigma = 10$ were used.

With reference to parameters in Alg. 1, for this experiment we used 75 time steps, which means that the LSTM graph can be viewed as a 75 layered feed-forward network when unrolled. Additionally, at each time step we use a **NOVAS_Layer** to compute the optimal control. Thus, the total number of network layers is $L = 75 + 74 = 149$ with f_i's being **NOVAS_Layer** for $i = 2,\ 4,\ 6, \ldots$.

A.3.2 PORTFOLIO OPTIMIZATION PROBLEM

We now consider a problem for which an explicit solution of the Hamiltonian $\min$ operator does not exist. Let N be the total number of stocks that make up an index I such that $I = \frac{1}{N}\sum_{i=1}^{N} S_i$, where S_i is the stock price process of the i-th stock. Let M be the number of a fixed selection of traded stocks taken from those N stocks such that $M < N$. Furthermore, let $u \in \mathbb{R}^{M+1}$ be the control vector. The $(N+1)$ dimensional state vector is comprised N stock prices and a wealth process W. The dynamics of each stock price and wealth process are given by

$$\pi_k = \big[\text{softmax}(u)\big]_k = \frac{e^{u_k}}{\sum_{m=1}^{M+1} e^{u_m}}, \quad (k = 1, 2, \cdots, M+1) \tag{23}$$

$$\mathrm{d}S_i(t) = S_i(t)\,\mu_i\,\mathrm{d}t + S_i(t)\,\mathrm{d}\eta_i \quad \left(\text{where, } i = 1, 2, \cdots, N \text{ and } \mathrm{d}\eta_i = \sum_{j=1}^{N} \sigma_{i,j}\,\mathrm{d}w_j(t)\right) \tag{24}$$

$$\mathrm{d}W(t) = W(t)\left(\pi_1\, r\,\mathrm{d}t + \sum_{m=2}^{M+1} \pi_m\,\mu_m\,\mathrm{d}t + \sum_{m=2}^{M+1} \pi_m\,\mathrm{d}\eta_m\right) \tag{25}$$

where π_k is the fraction of wealth invested in the k-th traded stock, r is rate of return per period of the risk-free asset, μ_i is the rate of return of the i^{th} stock. Here, $\sigma_{i,j}$ denotes the standard deviation of noise terms entering the $i-$the stock process wherein $i = j$ indicates the contribution of the process' own noise as opposed to $i \neq j$, which indicates the interaction of noises between stocks (correlation). All w_i's are mutually independent standard Brownian motions. To obtain the σ's, we used randomly generated synthetic covariance matrices which mimic real stock-market data. Note that the M traded stocks were randomly picked and were not constrained to be any specific sub-selection of the N stocks. Separate noise realizations were used during training, validation, and testing to ensure that the network does not over-fit to a particular noise profile.

For our experiments we use $N = 100$ stocks that make up the index I and $M = 20$ traded stocks. We used a scaled squared-softplus function as terminal cost, given by

$$\phi(x(T)) = q\bigg(\frac{1}{\beta} \cdot \log\big(1 + e^{\beta(I(T) - W(T))}\big)\bigg)^2,$$

with $\beta = 10$, $q = 500$, and no running cost, focusing on investment outperformance at the end of the planning horizon of one year. To simulate the stock dynamics we used a time discretization of $dt = 1/52$, which amounts to controls (and thus amounts invested) being applied on a weekly basis, for a total time of 1 year. The deep FBSDE-NOVAS hyperparameters were as follows: 16 neurons each in a two-layer LSTM network to predict the gradient of the value function at each time step, a batch size of 32, an initial learning rate set to $1e^{-2}$ and reduced by factor of 0.1 after 4000 and 4500 training iterations. Training was done using the Adam optimizer for a total of 5000 iterations. For NOVAS, we used 100 samples with 5 inner-loop iterations for training and 200 samples with 50 inner-loop iterations for inference. The shape function used was $S(x) = \exp{(x)}$.

With reference to parameters in Alg. 1, for this experiment we used 52 time steps, which means that the LSTM graph can be viewed as a 52 layered feed-forward network when unrolled. Additionally, at each time step (except for the last time step) we use a **NOVAS_Layer** to compute the optimal control. Thus, the total number of network layers is $L = 52 + 51 = 103$ with f_i's being **NOVAS_Layer** for $i = 2,\, 4,\, 6, \ldots$.

A.3.3 LOSS FUNCTION FOR TRAINING DEEP FBSDE CONTROLLERS

The loss function used in our experiments to train the deep FBSDE controller with the NOVAS layer is as follows:

$$\begin{aligned}\mathcal{L} &= l_1 \cdot H_\delta\big(V(x_T, T) - V^*(x_T, T)\big) + l_2 \cdot H_\delta\big(V_x(x_T, T) - V_x^*(x_T, T)\big)\\ &+ l_3 \cdot H_\delta\big(V_{xx}(x_T, T) - V_{xx}^*(x_T, T)\big) + l_4 \cdot \big(V^*(x_T, T)\big)^2 + l_5 \cdot \big(V_x^*(x_T, T)\big)^2 + l_6 \cdot \big(V_{xx}^*(x_T, T)\big)^2,\end{aligned}$$

where

$$H_\delta(a) = \begin{cases} a^2, & \text{for } |a| < \delta, \\ \delta(2|a| - \delta), & \text{otherwise.} \end{cases}$$

Here, x_T denotes $x(T)$, $V(x_T, T)$, $V_x(x_T, T)$, and $V_{xx}(x_T, T)$ are the predicted value function, its predicted gradient, and the predicted last column of the Hessian matrix, respectively, at the terminal time step. The corresponding targets are obtained through the given terminal cost function $\phi\big(x(T)\big)$ so that $V^*(x_T, T) = \phi(x_T)$, $V_x(x_T, T) = \phi_x(x_T)$ and $V_{xx}(x_T, T) = \phi_{xx}(x_T)$. Each term is computed by averaging across the batch samples. Additionally, we may choose to add terms that directly minimize the targets. This is possible because gradients flow through the dynamics functions and therefore the weights of the LSTM can influence what the terminal state $x(T)$ will be.

For the cart-pole problem we used $\delta = 50$ and $[l_1, l_2, l_3, l_4, l_5, l_6] = [1, 1, 0, 1, 1, 0]$, and for the portfolio optimization problem we used $\delta = 50$ and $[l_1, l_2, l_3, l_4, l_5, l_6] = [1, 1, 1, 1, 0, 0]$.

A.3.4 HARDWARE CONFIGURATION AND RUN-TIMES

All experiments were run on a NVIDIA GeForce RTX 2080Ti graphics card with 12GB memory. The PyTorch (Paszke et al., 2019) implementation of the 101-dimensional portfolio optimization problem had a run-time of 2.5 hours.