Learning the optimal state-feedback using deep networks

Carlos Sánchez-Sánchez and Dario Izzo
Advanced Concepts Team
European Space Agency
Noordwijk, The Netherlands
Email: carlos.sanchez@esa.int, dario.izzo@esa.int

Daniel Hennes
Robotics Innovation Center
German Research Center for Artificial Intelligence
Bremen, Germany
Email: daniel.hennes@dfki.de

***Abstract*—We investigate the use of deep artificial neural networks to approximate the optimal state-feedback control of continuous time, deterministic, non-linear systems. The networks are trained in a supervised manner using trajectories generated by solving the optimal control problem via the Hermite-Simpson transcription method. We find that deep networks are able to represent the optimal state-feedback with high accuracy and precision well outside the training area. We consider non-linear dynamical models under different cost functions that result in both smooth and discontinuous (bang-bang) optimal control solutions. In particular, we investigate the inverted pendulum swing-up and stabilization, a multicopter pin-point landing and a spacecraft free landing problem. Across all domains, we find that deep networks significantly outperform shallow networks in the ability to build an accurate functional representation of the optimal control. In the case of spacecraft and multicopter landing, deep networks are able to achieve safe landings consistently even when starting well outside of the training area.**

I. Introduction

Thanks to the ever decreasing cost of computational resources and advances in research to train neural networks with many hidden layers [1], [2], there is a renewed interest in artificial neural networks (ANN), and in particular in deep neural networks (DNN), in the context of optimal control. While deep networks representation capabilities are particularly appropriate for perception related tasks such as image and speech recognition, it has been pointed out that control problems also benefit from these models [3], [4], [5].

In these works, interesting results were obtained in cases where incomplete state information is available and the optimal control is approximated with a mixture of reinforcement learning techniques and dynamic programming algorithms. Instead, the use of deep artificial neural networks to approximate the state-action pairs computed from the solution to the optimal control problem of deterministic continuous non-linear systems, where the full state is directly observed, has been largely neglected. Successful applications have so far been limited to simple domains (e.g., linear systems often appearing in case studies) or to unbounded control [6], [7], [8]. As a consequence, their possible use in robotics and engineering is rather restricted.

Contributions to the solution of both the Hamilton-Jacobi-Belmann (HJB) equations and the two point boundary value problem resulting from Pontryagin's optimal control theory showed possible uses of ANNs in the domain of continuous control [9]. On the one hand, several methods were proposed for the approximation of the value function $v(t, \mathbf{x})$ by means of ANNs architectures [8], [10], [11]. On the other hand, ANNs have been proposed and studied to provide a trial solution to the states, to the co-states and to the controls so that their weights can be trained to make sure the assembled trial Hamiltonian respects Pontryagin's conditions [6]. Clearly, in this last case, the networks have to be retrained for each initial condition. Recently, recurrent networks have been used to learn near-optimal controllers for motion tasks [12], however, the velocities (kinematics), and not the actual control, are predicted in this case, limiting the objective functions that can be considered. In this paper we successfully use DNNs to represent the solution to the Hamilton-Jacobi-Belmann policy equation for deterministic, non-linear, continuous time systems, hence directly predicting the optimal control variables.

Although more extensive research of DNNs for optimal control has been carried out for discrete-time optimal control problems, there are fundamental differences between previous approaches in this domain and our work. In particular, we use feedforward DNN architectures trained in a supervised manner. The majority of the previous work in the field of discrete-time optimal control has been focused on reinforcement learning techniques such as guided-policy search [3], [4]. The supervised learning methods we adopt, however, offer a straightforward approach that is usually preferable when it is possible to generate enough training data. A deep neural network trained with supervised signals, in the form of a stacked auto-encoder, was recently shown [13] to be able to learn an accurate temporal profile of the optimal control and state in a point-to-point reach, non-linear limb model, but their network architecture enforces the notable restriction of having a fixed time problem.

The DNNs here trained achieve real-time optimal control capabilities in complex tasks, something infeasible with current optimal control solvers, including those used here to generate the training data. In particular, we investigate the aerospace domain of optimal landing. State-of-the-art solutions are often, in this domain, using a precomputed optimal guidance profile coupled to a control system tasked to track it during descent [14]. In contrast, our work suggests to generate the

control action directly on-board and in real time using a DNN, thus providing the spacecraft with a reactive control system having near-optimal capabilities.

II. BACKGROUND

We consider deterministic systems defined by the dynamics $\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$, where $\mathbf{x}(t) : \mathbb{R} \to \mathbb{R}^{n_x}$ and $\mathbf{u}(t) : \mathbb{R} \to \mathcal{U} \subset \mathbb{R}^{n_u}$. We address the fundamental problem of finding an admissible control policy $\mathbf{u}(t)$ able to steer the system from any $\mathbf{x}_0 = \mathbf{x}(0)$ to some target goal in a subset $\mathcal{S} \subset \mathbb{R}^{n_x}$. At t_f the system will be in its final state $\mathbf{x}_f = \mathbf{x}(t_f) \in \mathcal{S}$ having minimized the following cost function:

$$J(\mathbf{x}(\cdot), \mathbf{u}(\cdot)) = \int_0^{t_f} \mathcal{L}(\mathbf{x}(t), \mathbf{u}(t))dt + h(\mathbf{x}(t_f))$$

The value function, defined for values $0 < t < t_f$ as:

$$v(\mathbf{x}, t) = \min_{\mathbf{u}} J(\mathbf{x}(\cdot), \mathbf{u}(\cdot)) \tag{1}$$

represents the minimal cost to reach the goal, starting from $\mathbf{x}$. Equivalently, the value function can be introduced as the solution to the partial differential equation:

$$\min_{\mathbf{u}} \{\mathcal{L}(\mathbf{x}, \mathbf{u}) + f(\mathbf{x}, \mathbf{u}) \cdot \nabla_{\mathbf{x}} v(\mathbf{x}, t)\} = 0 \tag{2}$$

subject to the boundary conditions $v(\mathbf{x}_f, t) = h(\mathbf{x}(t_f))$, $\forall \mathbf{x}_f \in \mathcal{S}$. The optimal control policy is then:

$$\mathbf{u}^*(\mathbf{x}) = \text{argmin}_{\mathbf{u}} \{\mathcal{L}(\mathbf{x}, \mathbf{u}) + f(\mathbf{x}, \mathbf{u}) \cdot \nabla_{\mathbf{x}} v(\mathbf{x}, t)\} \tag{3}$$

Equations 2 and 3 are the Hamilton-Jacobi-Bellman (HJB) equations for the optimal control problem here considered. They are a set of extremely challenging partial differential equations (PDEs) whose solution, pursued in the "viscosity" sense, is the solution to the original optimal control problem [15]. The HJB equations show the existence of an optimal state-feedback $\mathbf{u}^*(\mathbf{x})$ and provide a way to compute it once the value function is known. Numerical approaches to solving HJB equation, thus, often rely on parametric approximations of the value function, e.g. using the Galerkin method [16], which have also included ANNs in the past [11].

Unlike previous work, we use deep neural networks (DNNs) to learn directly the optimal state-feedback $\mathbf{u}^*(\mathbf{x})$ thus obtaining, indirectly, also a representation of the value function $v(\mathbf{x}, t) = J(\mathbf{x}^*, \mathbf{u}^*)$, while avoiding to make use of the network gradients when converting from value function to the optimal policy. The DNN weights are trained using supervised learning on precomputed values of the optimal state-feedback for a set of initial states. Eventually, the trained DNN represents directly the optimal state-feedback and can be thus used, for example, in a non-linear model predictive control architecture [17] to achieve real-time optimal control capabilities.

III. OPTIMAL CONTROL PROBLEMS

We consider three different domains of increasing complexity and dimension: the inverted pendulum, the pinpoint landing of a multicopter and the landing of a spacecraft. Additionally we consider different cost functions, resulting in different types of control profiles, i.e. smooth, saturated and bang-bang. In each problem we introduce an initialization area $\mathcal{A} \subset \mathbb{R}^{n_x}$: only initial conditions in $\mathcal{A}$ will be considered for the purpose of the training data generation.

A. Inverted pendulum

The first and simplest domain we consider is the inverted pendulum swing-up problem ($n_x = 4$, $n_u = 1$), also known as the cart-pole system in the reinforcement learning community. The objective is to drive a pendulum of mass m and length l to the upright position. The pivot point of the pendulum is mounted on a cart of mass M freely moving on the horizontal axis. One control action u_1 acts on the system, representing an horizontal force applied to the cart, we consider a bounded control $|u_1| \leq u_{max} = 4$ [N].

The pendulum state is $\mathbf{x} = [x, \theta, v_x, v_\theta]$ and the system dynamics are described by the following set of Ordinary Differential Equations (ODEs):

$$\begin{aligned} \dot{x} &= v_x \\ \dot{\theta} &= v_\theta \\ \dot{v}_x &= \tfrac{u_1 + ml\sin(\theta)v_\theta^2 - mg\cos(\theta)\sin(\theta)}{M + m - (m\cos^2(\theta))} \\ \dot{v}_\theta &= \tfrac{u_1\cos(\theta) - (M+m)\sin(\theta) + ml\cos(\theta)\sin(\theta)v_\theta}{ml\cos^2(\theta) - (M+m)l} \end{aligned} \tag{4}$$

The length of the pendulum is $l = 1$ [m], the mass of the cart is $M = 0.5$ [kg], the mass of the pendulum is $m = 0.2$ [kg] and the uniform vertical gravitational field has strength $g = 9.81$ [m/s^2].

The target goal is the unstable equilibrium position at $v_x = 0$, $\theta = 0$ and $v_\theta = 0$. The initialization area $\mathcal{A}$ for the pendulum is defined as: $v_x = 0$, $v_\theta = 0$ and $\theta \in [\pi/2, 3\pi/2]$ [rad]. The values chosen are such that multiple "pump-ups" are necessary to reach the target.

The objective of the problem is to reach the target goal while minimizing the quadratic control cost function:

$$J_p = \int_0^{t_f} u_1^2 dt. \tag{5}$$

B. Pinpoint landing (multicopter)

As a second domain we consider that of a multicopter pinpoint landing ($n_x = 5$, $n_u = 2$). The state is $\mathbf{x} = [x, z, v_x, v_z, \theta]$ and the dynamical model is described by the following set of ODEs [18]:

$$\begin{aligned} \dot{x} &= v_x & \dot{v}_x &= u_1\sin(\theta) \\ \dot{z} &= v_z & \dot{v}_z &= u_1\cos(\theta) - g \\ & & \dot{\theta} &= u_2 \end{aligned} \tag{6}$$

Two control actions are available: the thrust $u_1 \in [0, 20]$ [N], and the pitch rate $u_2 \in [-2, 2]$ [rad/s]. We consider Earth's gravity ($g = 1.62$ [m/s^2]) and a quadrotor mass $m = 1$ [kg].

The goal state $x_f = 0$, $z_f = 0.1$, $v_{xf} = 0$, $v_{zf} = -0.1$, $\theta_f = 0$ corresponds to reaching a point close to the ground with low vertical velocity. The initialization area $\mathcal{A}$ is $x_0 \in [-5, 5]$ [m], $z_0 \in [5, 20]$ [m], $v_{x0} \in [-1, 1]$ [m/s], $v_{z0} \in [-1, 1]$ [m/s], $\theta_0 \in \left[-\frac{\pi}{10}, \frac{\pi}{10}\right]$ [rad].

In this case, two different cost functions are studied. The first one corresponds to quadratic control:

$$J_p = \int_0^{t_f} (\alpha_1 u_1^2 + \alpha_2 u_2^2) dt \tag{7}$$

with $\alpha_1 = 1$, $\alpha_2 = 1$. The second one seeks to minimize time:

$$J_p = t_f. \tag{8}$$

C. Landing (spacecraft)

The last domain is that of a spacecraft landing under a uniform gravity field ($n_x = 6$, $n_u = 2$). This corresponds to a mass-varying system where the state is $\mathbf{x} = [x, z, v_x, v_z, \theta, m]$ and the dynamics are described by the following set of ordinary differential equations [17]:

$$\begin{array}{ll} \dot{x} = v_x & \dot{v}_x = u_1 \sin(\theta)/m \\ \dot{z} = v_z & \dot{v}_z = u_1 \cos(\theta)/m - g \\ \dot{m} = -u_1/(Isp \cdot g) & \dot{\theta} = u_2 \end{array} \tag{9}$$

The spacecraft is controlled by the thrust, $u_1 \in [0, 45760]$ [N], and a momentum exchange wheel controlling the pitch rate $u_2 \in [-0.0698, 0.0698]$ [rad/s]. We consider the Moon gravity $g = 1.62$ [m/s^2] and a main engine with specific impulse $I_{sp} = 311$ [s].

The target is: $z_f = 10$ [m], $v_{xf} = 0$ [m/s], $v_{zf} = -0.1$ [m/s]. No target is defined for x as we consider a free landing scenario, hence the control action does not depend on x. The initialization area $\mathcal{A}$ is: $z_0 \in [500, 2000]$ [m], $m_0 \in [8000, 12000]$ [kg], $v_{x0} \in [-100, 100]$ [m/s], $v_{z0} \in [-30, 10]$ [m/s] and $\theta_0 \in \left[-\frac{\pi}{20}, \frac{\pi}{20}\right]$ [rad].

We study the minimum mass problem, the cost function being:

$$J_m = \int_{t_0}^{t_f} u_1/(Isp \cdot g) = m_f - m_0, \tag{10}$$

Considering that the optimal action u_2 is not well defined when $u_1 = 0$ (multiple equivalent actions exist), we add the term αu_2^2 thus minimizing the cost function :

$$J'_m = J_m + \int_{t_0}^{t_f} \alpha(u_2^2) dt, \tag{11}$$

with $\alpha = 10$.

Our approach generalizes to quadratic and time-optimal control, however, as results are similar to the corresponding cases in the multicopter domain they are not addressed further.

The optimal control solutions of these three domains represent different classes of control profiles. The quadratic control problems result in continuous functions saturated at their bounds. The time-optimal and mass-optimal cost functions, instead, result in discontinuous bang-off-bang control structures for the thrust. In the case of mass-optimal landing, complexity is increased by the change in behaviour of the pitch control during the thrust on and off phases. Pitch is constant until the switching point and smooth and continuous afterwards.

D. Training data generation

For each of the problems described above we generate a dataset containing pairs $(\mathbf{x}^*, \mathbf{u}^*)$, where $\mathbf{x}^*$ is a state and $\mathbf{u}^*$ is the corresponding optimal action. The dataset is generated solving the non-linear programming problem (NLP) resulting from using the Hermite-Simpson transcription method [19], a direct method to tackle optimal control problems. The solution is provided by a sequential quadratic programming NLP solver, namely SNOPT [20]. Although it is possible to use this approach to generate the training dataset, the convergence is not guaranteed and it is computationally expensive, hence not being suitable for real-time implementations.

For each of the considered problems we generate $150,000$ different trajectories starting from a point randomly sampled from the initialization area $\mathcal{A}$. Of each trajectory we store 60 (multicopter) or 100 (spacecraft and pendulum) state-control pairs uniformly taken along the trajectory, resulting in $9,000,000$-$15,000,000$ training samples. We use 90% of the trajectories to train the model while the rest is used for validation.

In the landing cases, due to known problems caused by the use of the direct method in connection with saturated controls (i.e. bang-bang), we observe small chattering effects that have a negative impact on the learning process. We address this problem by adding a regularization term to the objective function when computing the NLP solution. The added term corresponds to βJ_p, where J_p is the quadratic power goal function. If β is chosen sufficiently small, the effect of this term on the final value of $J(\mathbf{x}(\cdot))$ is negligible, but helps the sequential quadratic programming solver to converge towards a solution without chattering. We use $\alpha_1 = \alpha_2 = 1, \beta = 0.001$ for the multicopter time-optimal problem and $\alpha_1 = 1$, $\alpha_2 = 0$, $\beta = 10^{-10}$ for the spacecraft mass-optimal problem.

IV. STATE-FEEDBACK APPROXIMATION

The data, generated as described above, is used to train feed-forward neural networks in a supervised manner. For the multicopter and spacecraft models, we train separate networks for each control variable (i.e. thrust and pitch).

DNN architecture

We consider both models with only one (shallow) and several hidden layers (deep). The selection of the non-linearities used in the hidden units has been identified as one of the most important factors of DNN architectures [21]. We compare the classical sigmoid units to rectified linear units (ReLUs), which correspond to the activation function $max(0, x)$. It has been pointed out that ReLu units have two main benefits when

TABLE I
MEAN ABSOLUTE ERROR OF NETWORKS WITH DIFFERENT NON-LINEARITIES (HIDDEN - OUTPUT) FOR THE PROBLEM OF MASS OPTIMAL LANDING (SPACECRAFT).

	Architecture	Train	Val.
u_1	tanh - tanh	955.53	963.61
	ReLu - tanh	918.90	936.21
	ReLu - linear	973.20	978.35
u_2	tanh - tanh	0.00283	0.00284
	ReLu - tanh	0.00250	0.00252
	ReLu - linear	0.00257	0.00259

compared to sigmoid functions: they do not saturate, which avoids the units to stop learning after reaching a point (the vanishing gradient problem), and the output of the units is frequently zero, which forces a sparse representation that is often addressed as a way of regularization that improves the generalization capabilities of the model [22]. The sigmoid function used for the comparison is the hyperbolic tangent, selected based on their convergence behaviour compared to the standard logistic function [23]. We consistently obtain higher performance with ReLus (see table I) and thus they are used in the hidden units of our DNNs. The output units are $tanh$ units, which always provided a better result when we compare them to linear units. All the inputs and outputs are normalized by subtracting the mean and dividing by the standard deviation. The normalized outputs are then further scaled to the range [-1,1] to make sure they can all be represented by the $tanh$ functions.

Training

All networks are trained until convergence with stochastic gradient descent (SGD) and a batch size of 8. After every epoch the loss error is computed for the evaluation set and the learning process is stopped if there are more than 3 epochs with no increments affecting, at least, the third significant digit. We use Xavier's initialization method [24] to randomly set the initial weights. Although it was designed to improve the learning process for logistic units, it has been shown that this idea can also be beneficial for networks with ReLu units [25]. In our case, each weight w_i is drawn from a uniform distribution $U[-a, a]$, with $a = \sqrt{\frac{12}{\text{fan}_{\text{in}}+\text{fan}_{\text{out}}}}$, being fan_{in}, fan_{out} the number of units of the previous and following layers.

The training process seeks to minimize the squared loss function $C = \sum_{i=0}^{8} \frac{1}{8}(\mathcal{N}(\mathbf{x_i}) - y(\mathbf{x_i}))^2$ for the neural network output $\mathcal{N}(\mathbf{x_i})$ and the optimal action $y(\mathbf{x_i})$. The weights w_i are updated with a learning rate $\eta = 0.001$ and momentum with $\mu = 0.9$ [26]:

$$v_i \rightarrow v_i' = \mu v_i - \eta \frac{\partial C}{\partial w_i}$$

$$w_i \rightarrow w_i' = w_i + v_i'$$

DNN driven trajectories

Once trained, the DNNs can be used to predict the control given any state $\mathbf{x}$. This allows us to also compute the full trajectory by numerical integration of the system dynamics $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}^*) = \mathbf{f}(\mathbf{x}, \mathcal{N}(\mathbf{x}))$:

$$\mathbf{x}(t) = \int_0^t \mathbf{f}(\mathbf{x}, \mathcal{N}(\mathbf{x}))ds$$

The integration is stopped when the goal state is reached within some tolerance or a failure is detected (i.e., too much time has elapsed or a crash is detected, i.e. $z < 0$, in case of the multicopter or the spacecraft domain). In the case of the pendulum the tolerance is 0.005 in each of the variables θ, v_x, v_z. For the multicopter the tolerance to the goal position is set to 0.1 [m] and for the Ffinal velocity to 0.1 [m/s]. For the spacecraft landing case, a velocity tolerance of 1 [m/s] has to be achieved after passing the 10 [m] goal without crashing.

V. RESULTS

The mean absolute error (MAE) is used to evaluate the difference between the optimal actions and the network predictions along the states of the optimal trajectories. Although this measure is useful for comparison purposes, it is not a way to evaluate how well the required task is accomplished. Small errors are propagated through the whole trajectory and may result in sub-optimal trajectories as well as in catastrophic failures. We are interested both in determining whether the DNNs are able to reach the goal position and in computing the cost function along the DNN trajectory. The full DNN-driven trajectories are thus compared to the optimal ones. Figure 1 shows a comparison between a deep network and a shallow one in terms of predicted controls and (D)NN driven trajectories.

Figure 2 shows the optimal control and the predictions of the DNN for the states of the optimal trajectory for the simple problem of pendulum stabilization starting from different initial states. The network accurately predicts the optimal control action for trajectories with different number of swings as well as control with saturated regions. Additionally, the same figure includes all the state variables for one simulation where the trajectory is driven by the DNN, where it is possible to see how the DNN accurately reproduces the optimal trajectory. Overall the DNN driven trajectories are consistently able stabilize the pendulum in the upright position and the corresponding cost function (power) is on average 0.7% higher.

To study the more complex multicopter and spacecraft domains, 100 random initial positions are generated in $\mathcal{A}$ and the DNN driven trajectories are simulated. Table II summarizes the results. For each problem, we include the rate of successful landings as well as the average relative error of the value function obtained by the DNN with respect to the optimal one. All the models achieve a 100% success rate with high precision when they are initialized within $\mathcal{A}$. In the same table, for comparison purposes, we include the error for DNNs trained on a different objective, showing that the errors are, as expected, significantly higher.

Examples of the optimal control profile compared to the control along a DNN-driven trajectory are shown in Figure 3. The differences between the optimal and DNN controls are

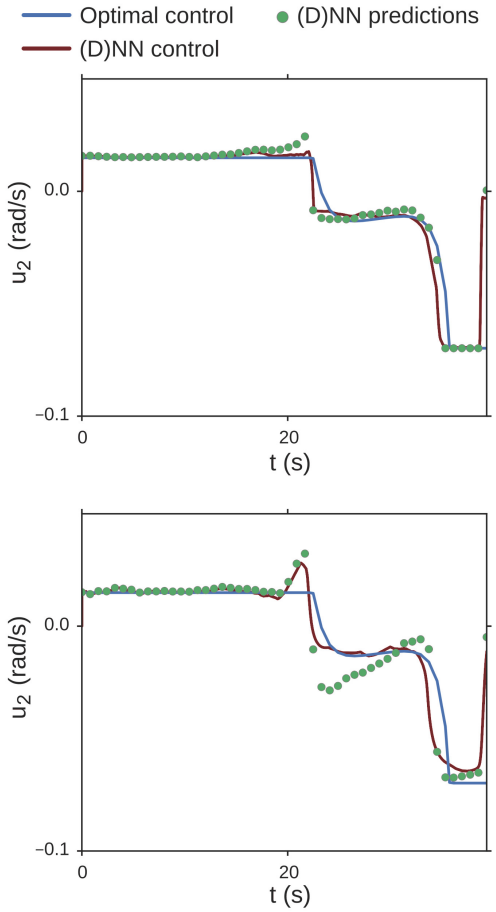

Fig. 1. Example of the u_2 control during a challenging mass optimal spacecraft landing. The optimal control, the (D)NN predictions for each state along the optimal trajectory and the full trajectory driven by the (D)NN are included. Top: deep network (4 hidden layers, 64 units, 12992 parameters). Bottom: shallow network (1 hidden layer, 2048 units, 16384 parameters). In this case the deep network achieves safe landing conditions, while the shallow network results in a catastrophic failure.

minor in all cases and do not prevent the models from reaching the target state.

A. Generalization

High success rates from outside of the training data are obtained without a major impact on performances as shown in table II. In the case of the multicopter trained with a quadratic cost function, if the trajectories are initialized from an extension of $\mathcal{A}$ of 5 [m] both in x and z, the DNN still achieves 100% of safe landings with minor deviations from the optimal cost (0.74%). The success rate remains high (84%) in the case of an extension of 10 [m]. For time optimal trajectories, the success rate outside of $\mathcal{A}$ is lower (70%), mainly due to the terminal velocity v_z violating the set tolerance.

In the spacecraft landing case, safe landing are achieved consistently from initial positions up to 1 [km] higher than those considered in $\mathcal{A}$ and more than 50% of safe landings are achieved considering an extension of 2 [km]. If the networks

TABLE II
SIMULATION OF 100 TRAJECTORIES WITH THE DNNS. AVERAGE RELATIVE ERROR WITH RESPECT TO THE OPTIMAL TRAJECTORY AND SUCCESS RATE (IN PARENTHESIS).

Multicopter - Power

Training area	Outside (5m)	Outside (10m)	Time
0.47% (100%)	0.73% (100%)	2.28% (64%)	9.75%

Multicopter - Time

Training area	Outside (5m)	Outside (10m)	Power
0.41% (100%)	2.12% (70%)	-	14.92%

Spacecraft - Mass

Training area	Outside (1km)	Outside (2km)	Time
1.50% (100%)	1.07% (100%)	1.83% (53%)	4.46%

have learned an approximation to the solution of the HJB equations, we would expect them to work in conditions that are not considered in the training data. Several other strong indications suggest that this is indeed the case.

In Figure 3 we see that, if the simulation is not stopped when the spacecraft or multicopter reaches the goal position, the spacecraft starts hovering close to the goal with no horizontal or vertical velocity using its thrust to compensate the gravitational force. For the case of the multicopter it is remarkable that the thrust predicted by the network is approximately 9.81 [N] also in the case of time optimal control, where the control is always saturated and thus that value is not present in the training data. In the spacecraft landing case the spacecraft, after having reached its goal, does not hover with precision, but it still shows a similar behaviour and it is possible to observe that the thrust (u_2) constantly decreases, which can be interpreted as an attempt to compensate for the mass reduction due to fuel consumption.

In addition to Table II, Figure 4 shows how both the multicopter and the spacecraft reach the goal position from initial states well outside of the bounds of the training area. This generalization happens not only for meaningful initial positions but also for points lower than the landing position, which requires to move upwards, a behaviour not presented during training.

B. Importance of depth

Table III shows how shallow networks with just two layers (one hidden layer and the output layer) are not able to approximate the state-feedback control as accurately as deep networks. Deep networks always outperform shallow networks with the same number of parameters. We include an example of this for each of the multicopter and spacecraft domains and we offer a more detailed study of u_2 for the spacecraft model, given that it has the most complex and representative control profile.

It is possible to see how, after some point, doubling the number of units in a shallow network does not produce signifi-

TABLE III
TRAIN AND VALIDATION MEAN ABSOLUTE ERROR (MAE) FOR THE MULTICOPTER AND SPACECRAFT MODELS. AN EXAMPLE, CORRESPONDING TO FIGURE 1, IS HIGHLIGHTED.

Multicopter - Power

	layers	units / layer	#weights	train error	val. error
u_1	2	1,104	8,832	0.0897	0.0902
	4	64	8,832	0.0668	0.0672
	5	64	17,216	0.0632	0.0637
u_2	2	1,104	8,832	0.0645	0.065
	4	64	8,832	0.0429	0.0431
	5	64	17,216	0.0416	0.0418

Multicopter - Time

	layers	units / layer	#weights	train error	val. error
u_1	2	1,104	8,832	0.232	0.234
	4	64	8,832	0.152	0.153
	5	64	17,216	0.145	0.146
u_2	2	1,104	8,832	0.105	0.104
	4	64	8,832	0.0852	0.0853
	5	64	17,216	0.0786	0.0789

Spacecraft - Mass

	layers	units / layer	#weights	train error	val. error
u_1	2	1,104	8,832	1243.78	1246.91-
	4	64	8,832	936.98	946.76
	5	64	17,216	930.71	938.16
u_2	2	256	2,048	0,00462	0,00460
	2	1,104	8,834	0,00379	0,00379
	2	**2,048**	**16,384**	**0,00370**	**0,00371**
	3	64	4,672	0,00307	0,00307
	3	128	17,536	0,00270	0,00272
	3	256	67,840	0,00263	0,00264
	4	64	8,832	0,00250	0,00252
	4	128	34,048	0,00241	0,00242
	5	**64**	**12,992**	**0,00236**	**0,00237**

cant improvements. In this case the performance is lower than deeper networks with a much lower number of parameters, which indicates that, given the data and the training methods used, it is not possible to learn a precise representation of the function with shallow networks.

C. Speed comparison

The DNN, compared to the original NLP solver, will only provide an advantage if its processing time is lower, thus being a viable candidate for real-time implementations.

We compute the time it takes to compute the control given the state with neural networks of different sizes (2 layers and 1,140 units and 5 layers and 64, 128 and 256 units) and the NLP solver with different number of nodes (5, 20, 50) for the quadrotor and spacecraft problems. In each case, the time is measured for 100 different trajectories and the average is computed. The results are included in Table IV.

For the spacecraft mass optimal control problem, computing the optimal control with a DNN with 4 hidden layers with 64 units is more than 30,000 times faster than the NLP solver used to generate the training data (NLP with 50 nodes). Even if a low number of nodes is used in the NLP solver, resulting in sub-optimal controllers, the DNNs are up to 1,500 times faster. Using the DNN with 4 layers and 128 units per hidden layer it is possible to compute the control with a frequency higher than 9kHz, which is suitable for the real-time applications here considered.

TABLE IV
TIME TO COMPUTE THE THE CONTROL GIVEN THE STATE USING AN INTEL(R) XEON(R) E5-2687W @ 3.10GHZ CPU.

Non linear programming (NLP) solver

	5 nodes	20 nodes	50 nodes
Multicopter (time)	38.86 ms	168.71 ms	594.87 ms
Multicopter (power)	32.60 ms	222.58 ms	1245.98 ms
Spacecraft (mass)	82.82 ms	280.70 ms	1830.57 ms

DNN (network for each variable)

1 - 1140	4 - 64	4 - 128	4 - 256
0.048 ms	0.056 ms	0.068 ms	0.11 ms

VI. CONCLUSIONS

We have shown that deep neural networks (DNN) can be trained to learn the optimal state-feedback of continuous time, deterministic, non-linear systems. The trained networks are not limited to predict the optimal state-feedback from points within the subset of the state space used during training, but are able to generalize to points well outside the training data, suggesting that the solution to Hamilton-Jacobi-Bellman (HJB) equations is the underlying model being learned. The depth of the networks has a great influence on the obtained results and we find that shallow networks, while trying to approximate the optimal state-feedback, are unable to learn its complex structure satisfactorily.

Alternative methods to compute the optimal trajectories, such as the direct methods used to generate our training data, require expensive computations, discarding them as candidates for real time applications. Our work opens to the possibility to design real-time optimal control architectures for complex non-linear systems using a DNN to drive directly the state-action selection. With this respect we show that, in the landing cases, the error introduced by the use of the trained DNN, not only does not have a significant impact on the final cost function achieved, but it is also safe in terms of avoiding catastrophic failures.

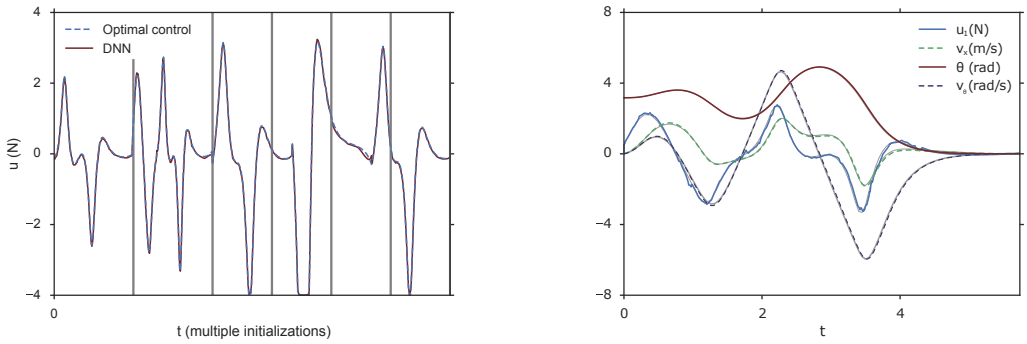

Fig. 2. The inverted pendulum problem. Left image shows the predictions of a DNN compared to the optimal control for multiple initial trajectories. Right image includes the variables v_x, θ, v_θ and the control u_1 applied to the cart. The values for the optimal trajectory are also displayed in gray, but are mostly perfectly overlapped and thus not visible.

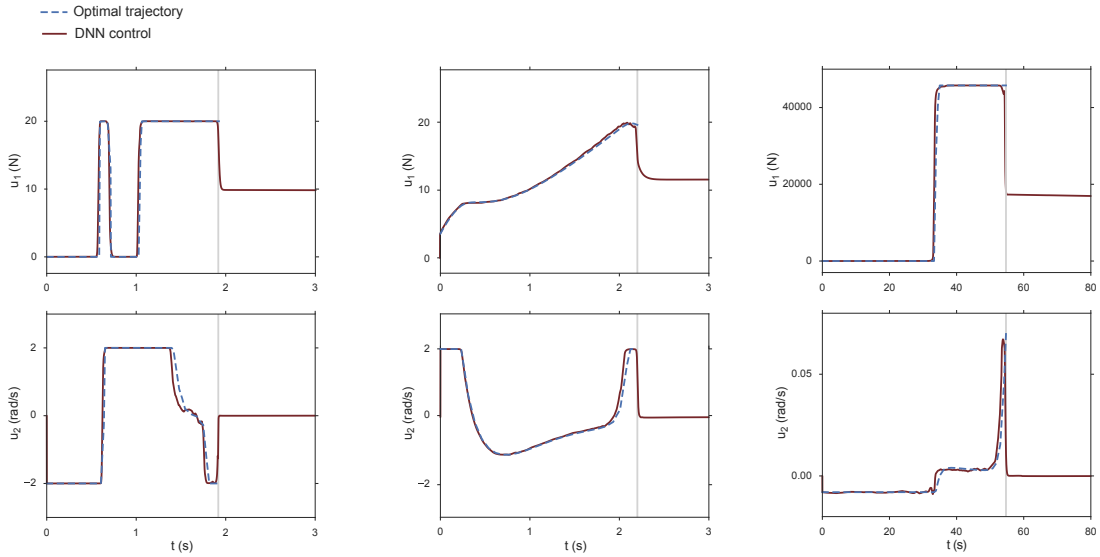

Fig. 3. Control profile along trajectories driven by a DNN for the multicopter power (left) and time (center) optimal problems and for the spacecraft mass problem (right). After reaching the goal (gray line) the trajectory computed by the network starts a hovering phase. For the multicopter cases this requires a constant thrust of 9.8 [N], for the spacecraft model the thrust decreases over time as the total mass decreases.

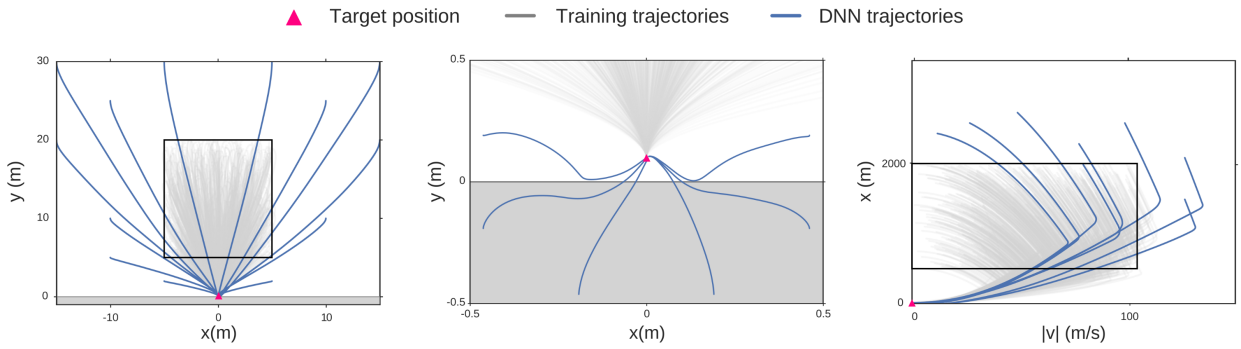

Fig. 4. Trajectories computed by DNNs from initial points outside of the training area both for the power optimal multicopter pin-point landing (left and center) and the mass optimal spacecraft landing (right). The black box indicates the initialization area and the gray lines trajectories randomly sampled from the training data.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, insight. [Online]. Available: http://dx.doi.org/10.1038/nature14539

[2] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[3] S. Levine, "Exploring deep and recurrent architectures for optimal control," *arXiv preprint arXiv:1311.1761*, 2013.

[4] V. Levine, Sergey; Koltun, "Guided policy search," *International Conference on Machine Learning*, 2013.

[5] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," *arXiv preprint arXiv:1509.06791*, 2015.

[6] S. Effati and M. Pakdaman, "Optimal control problem via neural networks," *Neural Computing and Applications*, vol. 23, no. 7-8, pp. 2093–2100, 2013.

[7] Y. Xiong, L. Derong, W. Ding, and M. Hongwen, "Constrained online optimal control for continuous-time nonlinear systems using neuro-dynamic programming," in *Control Conference (CCC), 2014 33rd Chinese*. IEEE, 2014, pp. 8717–8722.

[8] P. Medagam and F. Pourboghrat, "Optimal control of nonlinear systems using rbf neural network and adaptive extended kalman filter," in *American Control Conference, 2009. ACC '09.*, June 2009, pp. 355–360.

[9] E. Todorov, "Optimality principles in sensorimotor control," *Nature neuroscience*, vol. 7, no. 9, pp. 907–915, 2004.

[10] F. L. Lewis and M. Abu-Khalaf, "A hamilton-jacobi setup for constrained neural network control," in *Intelligent Control. 2003 IEEE International Symposium on*. IEEE, 2003, pp. 1–15.

[11] Y. Tassa and T. Erez, "Least squares solutions of the hjb equation with neural network value-function approximators," *Neural Networks, IEEE Transactions on*, vol. 18, no. 4, pp. 1031–1041, July 2007.

[12] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. V. Todorov, "Interactive control of diverse complex characters with neural networks," in *Advances in Neural Information Processing Systems*, 2015, pp. 3114–3122.

[13] M. Berniker and K. P. Kording, "Deep networks for motor control functions," *Frontiers in computational neuroscience*, vol. 9, 2015.

[14] B. Acikmese and S. R. Ploen, "Convex programming approach to powered descent guidance for mars landing," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 5, pp. 1353–1366, 2007.

[15] M. Bardi and I. Capuzzo-Dolcetta, *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Springer Science & Business Media, 2008.

[16] R. W. Beard, G. N. Saridis, and J. T. Wen, "Galerkin approximations of the generalized hamilton-jacobi-bellman equation," *Automatica*, vol. 33, no. 12, pp. 2159–2177, 1997.

[17] D. Izzo and G. de Croon, "Nonlinear model predictive control applied to vision-based spacecraft landing," in *Proceedings of the EuroGNC 2013, 2nd CEAS Specialist Conference on Guidance, Navigation & Control, Delft University of Technology*, 2013, pp. 91–107.

[18] M. Hehn, R. Ritz, and R. DAndrea, "Performance benchmarking of quadrotor systems using time-optimal control," *Autonomous Robots*, vol. 33, no. 1-2, pp. 69–88, 2012.

[19] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*. Siam, 2010, vol. 19.

[20] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[21] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?" in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 2146–2153.

[22] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[23] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Neural Networks: Tricks of the Trade: Second Edition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. Efficient BackProp, pp. 9–48. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-35289-8_3

[24] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *International conference on artificial intelligence and statistics*, 2010, pp. 249–256.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1026–1034.

[26] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th international conference on machine learning (ICML-13)*, 2013, pp. 1139–1147.