

---

# The Differentiable Cross-Entropy Method

---

Brandon Amos<sup>1</sup> Denis Yarats<sup>1,2</sup>

## Abstract

We study the cross-entropy method (CEM) for the non-convex optimization of a continuous and parameterized objective function and introduce a differentiable variant that enables us to differentiate the output of CEM with respect to the objective function’s parameters. In the machine learning setting this brings CEM inside of the end-to-end learning pipeline where this has otherwise been impossible. We show applications in a synthetic energy-based structured prediction task and in non-convex continuous control. In the control setting we show how to embed optimal action sequences into a lower-dimensional space. DCEM enables us to fine-tune CEM-based controllers with policy optimization.

## 1. Introduction

Recent work in the machine learning community has shown how optimization procedures can create new building-blocks for the end-to-end machine learning pipeline (Gould et al., 2016; Johnson et al., 2016; Amos et al., 2017; Amos & Kolter, 2017; Domke, 2012; Metz et al., 2016; Finn et al., 2017; Zhang et al., 2019; Belanger et al., 2017; Rusu et al., 2018; Srinivas et al., 2018; Amos et al., 2018; Agrawal et al., 2019a). In this paper we focus on the setting of optimizing an *unconstrained, non-convex, and continuous* objective function  $f_\theta(x) : \mathbb{R}^n \times \Theta \rightarrow \mathbb{R}$  as

$$\hat{x} := \arg \min_x f_\theta(x), \quad (1)$$

where we assume  $\hat{x}$  is unique and that  $f$  is parameterized by  $\theta \in \Theta$  and has inputs  $x \in \mathbb{R}^n$ . If it exists, *some* (sub-)derivative  $\nabla_\theta \hat{x}$  is useful in the machine learning setting to make the output of the optimization procedure end-to-end learnable. For example,  $\theta$  could parameterize a predictive model that generates outcomes conditional on  $x$ .

End-to-end learning in these settings can be done by defining a loss function  $\mathcal{L}$  on top of  $\hat{x}$  and taking gradient steps  $\nabla_\theta \mathcal{L}$ . If  $f_\theta$  were *convex* this gradient is easy to analyze and compute when it exists and is unique (Gould et al., 2016; Johnson et al., 2016; Amos et al., 2017; Amos & Kolter, 2017). Analyzing and computing a “derivative” through the *non-convex*  $\arg \min$  in eq. (1) is not as easy and is challenging in theory and practice. The derivative may not exist or may be *uninformative* in theory, it might not be unique, and even if it does, the numerical solver being used to compute the solution may not find a global or even local optimum of  $f$ . One promising direction to sidestep these issues is to approximate the  $\arg \min$  operation with an explicit optimization procedure that is interpreted as another compute graph and *unrolled* through, *i.e.* seen as a sequence of differentiable computations. This is most commonly done with gradient descent as in Domke (2012); Metz et al. (2016); Finn et al. (2017); Belanger et al. (2017); Rusu et al. (2018); Srinivas et al. (2018); Foerster et al. (2018); Zhang et al. (2019). This approximation adds definition and structure to an otherwise ill-defined desiderata at the cost of biasing the gradients and enabling the learning procedure to over-fit to the hyper-parameters of the optimization algorithm, such as the number of gradient steps or the learning rate.

In this paper we show how to use the *cross-entropy method* (CEM) (Rubinstein, 1997; De Boer et al., 2005) to approximate the derivative through an *unconstrained, non-convex, and continuous*  $\arg \min$ . CEM for optimization is a *zeroth-order optimizer* and works by generating a sequence of samples from the objective function. We show a simple and computationally negligible way of making CEM differentiable that we call DCEM by using the smooth top- $k$  operation from Amos et al. (2019). This also brings CEM into the end-to-end learning process in scenarios such as control where there is otherwise a disconnection between the objective that is being learned and the objective that is induced by deploying CEM on top of those models.

We first study DCEM in a simple non-convex *energy-based learning* setting for regression. We contrast using unrolled gradient descent and DCEM for optimizing over a SPEN (Belanger & McCallum, 2016). We show that unrolling through gradient descent in this setting over-fits to the number of gradient steps taken and that DCEM generates a more reasonable energy surface.

---

<sup>1</sup>Facebook AI Research <sup>2</sup>New York University.

Correspondence to: Brandon Amos <bda@fb.com>.

We next focus on using DCEM in the context of *non-convex continuous control* as a *differentiable policy class* that is end-to-end learnable. This setting is especially interesting as vanilla CEM is the state-of-the-art method for solving the control optimization problem with *neural network transition dynamics* as in Chua et al. (2018); Hafner et al. (2018). We show that DCEM is useful for embedding *action sequences* into a lower-dimensional space to make solving the control optimization process significantly less computationally and memory expensive. This controller induces a *differentiable policy class* parameterized by the model-based components. DCEM is a solution to the *objective mismatch problem* in model-based control (Lambert et al., 2020), which is the issue that arises when training model-based components with the objective of maximizing the data likelihood but then using the model-based components for the objective of control. We use PPO (Schulman et al., 2017) to *fine-tune* the model-based components, demonstrating that it is possible to use standard policy learning for model-based RL components in addition to maximum-likelihood fitting.

## 2. Background and Related Work

### 2.1. Differentiable optimization-based modeling in machine learning

*Optimization-based modeling* is a way of integrating specialized operations and domain knowledge into end-to-end machine learning pipelines, typically in the form of a parameterized  $\arg \min$  operation. *Convex, constrained, and continuous* optimization problems, e.g. as in Gould et al. (2016); Johnson et al. (2016); Amos et al. (2017); Amos & Kolter (2017); Agrawal et al. (2019a), capture many standard layers as special cases and can be differentiated through by applying the *implicit function theorem* to a set of optimality conditions from convex optimization theory, such as the *KKT conditions*. *Non-convex and continuous* optimization problems, e.g. as in Domke (2012); Belanger & McCallum (2016); Metz et al. (2016); Finn et al. (2017); Belanger et al. (2017); Rusu et al. (2018); Srinivas et al. (2018); Foerster et al. (2018); Amos et al. (2018); Pedregosa (2016); Jenni & Favaro (2018); Rajeswaran et al. (2019); Zhang et al. (2019), are more difficult to differentiate through. Differentiation is typically done by *unrolling gradient descent* or applying the implicit function theorem to *some* set of optimality conditions, sometimes forming a locally convex approximation to the larger non-convex problem. *Unrolling gradient descent* is the most common way and approximates the  $\arg \min$  operation with gradient descent for the forward pass and interprets the operations as another compute graph for the backward pass that can all be differentiated through. In contrast to these works, we show how *continuous* and *non-convex*  $\arg \min$  operations can also be approximated with the cross-entropy method.

### 2.2. Embedding domains for optimization problems

Oftentimes the *solution space* of high-dimensional optimization problems may have structural properties that an optimizer can exploit to find a better solution or to find the solution quicker than an otherwise naïve optimizer. Meta-learning approaches such as LEO (Rusu et al., 2018) and CAVIA (Zintgraf et al., 2019) turn the optimization problem for adaptation in a high-dimensional parameter space into a lower-dimensional latent embedded optimization problem. In the context of Bayesian optimization this has been explored with random feature embeddings, hand-coded embeddings, and auto-encoder-learned embeddings (Antonova et al., 2019; Oh et al., 2018; Calandra et al., 2016; Wang et al., 2016; Garnett et al., 2013; Ben Salem et al., 2019; Kirschner et al., 2019). Luo et al. (2018) and Gómez-Bombarelli et al. (2018) turn discrete search problems for architecture search and molecular design, respectively, into embedded continuous optimization problems. We show that DCEM is another reasonable way of learning an embedded domain for exploiting the structure in and efficiently solving larger optimization problems, with the significant advantage of DCEM being that the latent space is directly learned to be optimized over as part of the end-to-end learning pipeline.

### 2.3. RL and Control

High-dimensional non-convex optimization problems that have a lot of structure in the solution space naturally arise in the control setting where the controller seeks to optimize the same objective in the same controller dynamical system from different starting states. This has been investigated in, e.g., planning (Icher et al., 2018; Icher & Pavone, 2019; Mukadam et al., 2018; Kurutach et al., 2018; Srinivas et al., 2018; Yu et al., 2019; Lynch et al., 2019), and policy distillation (Wang & Ba, 2019). Chandak et al. (2019) shows how to learn an action space for model-free learning and Co-Reyes et al. (2018); Antonova et al. (2019) embed *action sequences* with a VAE. There has also been a lot of work on learning reasonable latent *state space* representations (Tasfi & Capretz, 2018; Zhang et al., 2018; Gelada et al., 2019; Miladinović et al., 2019) that may have structure imposed to make it more controllable (Watter et al., 2015; Banijamali et al., 2017; Ghosh et al., 2018; Anand et al., 2019; Levine et al., 2019; Singh et al., 2019). In contrast to these works, we learn how to encode action sequences directly with DCEM instead of auto-encoding the sequences. This has the advantages of 1) never requiring the expensive expert’s solution to the control optimization problem, 2) potentially being able to surpass the performance of an expert controller that uses the full action space, and 3) being end-to-end learnable through the controller for the purpose of finding a latent space of sequences that DCEM is good at searching over.

Another direction the RL and control communities has been pursuing is on the combination of model-based and model-free methods by differentiating through model-based components [Bansal et al. \(2017\)](#) does this with Bayesian optimization and locally linear models. [Okada et al. \(2017\)](#); [Pereira et al. \(2018\)](#) makes path integral control ([Theodorou et al., 2010](#)) differentiable. [Agrawal et al. \(2019b\)](#) considers a class of convex controllers and differentiates through them with [Agrawal et al. \(2019a\)](#). [Amos et al. \(2018\)](#) proposes differentiable MPC and only do imitation learning on the cartpole and pendulum tasks with known or lightly-parameterized dynamics — in contrast, we are able to 1) scale our differentiable controller up to the cheetah and walker tasks, 2) use neural network dynamics inside of our controller, and 3) backpropagate a policy loss through the output of our controller and into the internal components.

### 3. The Differentiable Cross-Entropy Method

The cross-entropy method (CEM) ([Rubinstein, 1997](#); [De Boer et al., 2005](#)) is an algorithm to solve optimization problems in the form of [eq. \(1\)](#). CEM is an *iterative* and *zerth-order* solver that uses a sequence of *parametric sampling distributions*  $g_\phi$  defined over the domain  $\mathbb{R}^n$ , such as Gaussians. Given a *sampling distribution*  $g_\phi$ , the hyperparameters of CEM are the number of *candidate points* sampled in each iteration  $N$ , the number of *elite candidates*  $k$  to use to fit the new sampling distribution to, and the number of iterations  $T$ . The iterates of CEM are the *parameters*  $\phi$  of the sampling distribution. CEM starts with an *initial* sampling distribution  $g_{\phi_1}(X) \in \mathbb{R}^n$ , and in each iteration  $t$  generates  $N$  samples from the domain  $[X_{t,i}]_{i=1}^N \sim g_{\phi_t}(\cdot)$ , evaluates the function at those points  $v_{t,i} := f_\theta(X_{t,i})$ , and re-fits the sampling distribution to the top- $k$  samples by solving the maximum-likelihood problem<sup>1</sup>

$$\phi_{t+1} := \arg \max_{\phi} \sum_i \mathbb{1}\{v_{t,i} \leq \pi(v_t)_k\} \log g_\phi(X_{t,i}), \quad (2)$$

where the indicator  $\mathbb{1}\{P\}$  is 1 if  $P$  is true and 0 otherwise,  $g_\phi(X)$  is the likelihood of  $X$  under the distribution  $g_\theta$ , and  $\pi(x)$  sorts  $x \in \mathbb{R}^n$  in ascending order so that

$$\pi(x)_1 \leq \pi(x)_2 \leq \dots \leq \pi(x)_n.$$

We can then map from the final distribution  $g_{\phi_T}$  back to the domain by taking the mean of it, *i.e.*  $\hat{x} := \mathbb{E}[g_{\phi_{T+1}}(\cdot)]$ . In some settings, the best sample can be returned as  $\hat{x}$ .

**Proposition 1.** *For multivariate isotropic Gaussian sampling distributions we have that  $\phi = \{\mu, \sigma^2\}$  and [eq. \(2\)](#) has a closed-form solution given by the sample mean and variance of the top- $k$  samples as  $\mu_{t+1} := 1/k \sum_{i \in \mathcal{I}_t} X_{t,i}$*

<sup>1</sup>CEM’s name comes from [eq. \(2\)](#) more generally optimizing the cross-entropy measure between two distributions.

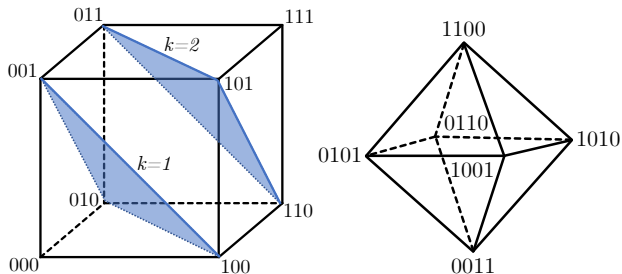


Figure 1. The *limited multi-label* (LML) polytope  $\mathcal{L}_{n,k}$  from [Amos et al. \(2019\)](#) is the set of points in the unit  $n$ -hypercube with coordinates that sum to  $k$ .  $\mathcal{L}_{n,1}$  is the  $(n-1)$ -simplex. The  $\mathcal{L}_{3,1}$  and  $\mathcal{L}_{3,2}$  polytopes (triangles) are on the left in blue. The  $\mathcal{L}_{4,2}$  polytope (an octahedron) is on the right. This polytope is also referred to as the *knapsack polytope* or *capped simplex*.

and  $\sigma_{t+1}^2 := 1/k \sum_{i \in \mathcal{I}_t} (X_{t,i} - \mu_{t+1})^2$ , where the top- $k$  indexing set is  $\mathcal{I}_t := \{i : v_{t,i} \leq \pi(v_t)_k\}$ .

This is well-known and is discussed in, *e.g.*, [Friedman et al. \(2001\)](#). We present this here to make the connections between CEM and DCEM clearer.

Differentiating through CEM’s output with respect to the objective function’s parameters with  $\nabla_{\theta} \hat{x}$  is useful, *e.g.*, to bring CEM into the end-to-end learning process in cases where there is otherwise a disconnection between the objective that is being learned and the objective that is induced by deploying CEM on top of those models. In the vanilla form presented above the top- $k$  operation in [eq. \(2\)](#) makes  $\hat{x}$  non-differentiable with respect to  $\theta$ . The function samples can usually be differentiated through with some estimator ([Mohamed et al., 2019](#)) such as the *reparameterization trick* ([Kingma & Welling, 2013](#); [Rezende et al., 2014](#); [Titsias & Lázaro-Gredilla, 2014](#)), which we use in all of our experiments.

The top- $k$  operation can be made differentiable by replacing it with a soft version ([Martins & Kreutzer, 2017](#); [Malaviya et al., 2018](#); [Amos et al., 2019](#)), or by using a stochastic oracle ([Brookes & Listgarten, 2018](#)). Here we use the *Limited Multi-Label Projection (LML) layer* ([Amos et al., 2019](#)), which is a *Bregman projection* of points from  $\mathbb{R}^n$  onto the *LML polytope* shown in [fig. 1](#) and defined by

$$\mathcal{L}_{n,k} := \{p \in \mathbb{R}^n \mid 0 \leq p \leq 1 \text{ and } 1^\top p = k\}. \quad (3)$$

The LML polytope is the set of points in the unit  $n$ -hypercube with coordinates that sum to  $k$  and is useful for modeling in multi-label and top- $k$  settings. If  $n$  is implied by the context we will leave it out and write  $\mathcal{L}_k$ .

We propose a *temperature-scaled LML* variant to project

onto the interior of the LML polytope with

$$\Pi_{\mathcal{L}_k}(x/\tau) := \arg \min_{0 < y < 1} -x^\top y - \tau H_b(y) \quad \text{s. t.} \quad \mathbf{1}^\top y = k \quad (4)$$

where  $\tau > 0$  is the temperature parameter and

$$H_b(y) := - \sum_i y_i \log y_i + (1 - y_i) \log(1 - y_i)$$

is the binary entropy function. We introduce the hyperparameter  $\tau$  to show how DCEM captures CEM as a special case as  $\tau \rightarrow 0$ . Equation (4) is a convex optimization layer and can be solved in a negligible amount of time with a GPU-amenable bracketing method on the univariate dual as described in Amos et al. (2019). The derivative  $\nabla_x \Pi_{\mathcal{L}_k}(x/\tau)$  necessary for backpropagation can be easily computed by implicitly differentiating the KKT optimality conditions as described in Amos et al. (2019).

We can use the LML layer to make a soft and differentiable version of eq. (2) as

$$\begin{aligned} \phi_{t+1} &:= \arg \max_{\phi} \sum_i \mathcal{I}_{t,i} \log g_{\phi}(X_{t,i}) \\ &\text{subject to } \mathcal{I}_t = \Pi_{\mathcal{L}_k}(v_t/\tau). \end{aligned} \quad (5)$$

This is now a *maximum weighted likelihood* estimation problem (Markatou et al., 1997; 1998; Wang, 2001; Hu & Zidek, 2002), which still admits an analytic closed-form solution in many cases, e.g. for the natural exponential family (De Boer et al., 2005). Thus using the soft top- $k$  operation with the reparameterization trick, e.g., on the samples from  $g$  results in a differentiable variant of CEM that we call DCEM and summarize in alg. 1. We usually also normalize the values in each iteration to help separate the scaling of the values from the temperature parameter.

**Proposition 2.** *The temperature-scaled LML layer  $\Pi_{\mathcal{L}_k}(x/\tau)$  approaches the hard top- $k$  operation as  $\tau \rightarrow 0^+$  when all components of  $x$  are unique.*

We prove this in app. A with the KKT conditions of eq. (4). The only difference between CEM and DCEM is the soft top- $k$  operation, thus when the soft top- $k$  operation approaches the hard top- $k$  operation, we can conclude:

**Corollary 1.** *DCEM becomes CEM as  $\tau \rightarrow 0^+$ .*

**Proposition 3.** *With an isotropic Gaussian sampling distribution, the maximum weighted likelihood update in eq. (5) becomes  $\mu_{t+1} := 1/k \sum_i \mathcal{I}_{t,i} X_{t,i}$  and  $\sigma_{t+1}^2 := 1/k \sum_i \mathcal{I}_{t,i} (X_{t,i} - \mu_{t+1})^2$ , where the soft top- $k$  indexing set is  $\mathcal{I}_t := \Pi_{\mathcal{L}_k}(v_t/\tau)$ .*

This is well-known and is discussed in, e.g., De Boer et al. (2005), and can be proved by differentiating eq. (5).

**Corollary 2.** *Prop. 3 captures prop. 1 as  $\tau \rightarrow 0^+$ .*

## 4. Applications

### 4.1. Energy-Based Learning

*Energy-based learning* for regression and classification estimate the conditional probability  $\mathbb{P}(y|x)$  of an output  $y \in \mathcal{Y}$  given an input  $x \in \mathcal{X}$  with a parameterized energy function  $E_{\theta}(y|x) \in \mathcal{Y} \times \mathcal{X} \rightarrow \mathbb{R}$  such that  $\mathbb{P}(y|x) \propto \exp\{-E_{\theta}(y|x)\}$ . Predictions are made by solving the optimization problem

$$\hat{y} := \arg \min_y E_{\theta}(y|x). \quad (6)$$

Historically linear energy functions have been well-studied, e.g. in Taskar et al. (2005); LeCun et al. (2006), as it makes eq. (6) easier to solve and analyze. More recently non-convex energy functions that are parameterized by neural networks are being explored — a popular one being *Structured Prediction Energy Networks* (SPENs) (Belanger & McCallum, 2016) which propose to model  $E_{\theta}$  with neural networks. Belanger et al. (2017) does supervised learning of SPENs by approximating eq. (6) with gradient descent that is then unrolled for  $T$  steps, i.e. by starting with some  $y_0$ , making gradient updates

$$y_{t+1} := y_t - \gamma \nabla_y E_{\theta}(y_t|x)$$

resulting in an output  $\hat{y} := y_T$ , defining a loss function  $\mathcal{L}$  on top of  $\hat{y}$ , and doing learning with gradient updates  $\nabla_{\theta} \mathcal{L}$  that go through the inner gradient steps.

In this context we can alternatively use DCEM to approximate eq. (6). One potential consideration when training deep energy-based models with approximations to eq. (6) is the impact and bias that the approximation is going to have on the energy surface. We note that for gradient descent, e.g., it may cause the energy surface to overfit to the number of gradient steps so that the output of the approximate inference procedure isn't even a local minimum of the energy surface. One potential advantage of DCEM is that the output is more likely to be near a local minimum of the energy surface so that, e.g., more test-time iterations can be used to refine the solution. We empirically illustrate the impact of the optimizer choice on a synthetic example in sect. 5.1.

### 4.2. Control and Reinforcement Learning

Our main application focus is in the *continuous control* setting where we show how to use DCEM to learn a *latent control space* that is easier to solve than the original problem and induces a *differentiable policy class* that allows parts of the controller to be fine-tuned with auxiliary policy or imitation losses.

We are interested in controlling *discrete-time* dynamical systems with *continuous* state-action spaces. Let  $H$  be the *horizon length* of the controller and  $\mathcal{U}^H$  be the space of



---

**Algorithm 1** DCEM( $f_\theta, g_\phi, \phi_1; \tau, N, k, T$ )

---

DCEM minimizes a parameterized objective function  $f_\theta$  and is differentiable w.r.t.  $\theta$ . Each DCEM iteration samples from the distribution  $g_\phi$ , starting with  $\phi_1$ . DCEM enables the derivative of  $\mathbb{E}[g_{\phi_{T+1}}(\cdot)]$  with respect to  $\theta$  to be computed by differentiating all of the iterative operations.

---

**for**  $t = 1$  to  $T$  **do**

$$[X_{t,i}]_{i=1}^N \sim g_{\phi_t}(\cdot)$$

▷ Sample  $N$  points from the domain. *Differentiate with reparameterization.*

$$v_{t,i} = f_\theta(X_{t,i})$$

▷ Evaluate the objective function at those points.

$$\mathcal{I}_t = \Pi_{\mathcal{L}_k}(v_t/\tau)$$

▷ Compute the soft top- $k$  projection of the values with eq. (4). *Implicitly differentiate*

Update  $\phi_{t+1}$  by solving the maximum weighted likelihood problem in eq. (5).

**end for**

**return**  $\mathbb{E}[g_{\phi_{T+1}}(\cdot)]$

---

**Algorithm 2** Learning an embedded control space with DCEM

---

**Fixed Inputs:** Dynamics  $f^{\text{trans}}$ , per-step state-action cost  $C_t(x_t, u_t)$  that induces  $C_\theta(z; x_{\text{init}})$ , horizon  $H$ , full control space  $\mathcal{U}^H$ , distribution over initial states  $\mathcal{D}$

**Learned Inputs:** Decoder  $f_\theta^{\text{dec}} : \mathcal{Z} \rightarrow \mathcal{U}^H$

---

**while** not converged **do**

Sample initial state  $x_{\text{init}} \sim \mathcal{D}$ .

$$\hat{z} = \arg \min_{z \in \mathcal{Z}} C_\theta(z; x_{\text{init}})$$

▷ Solve the embedded control problem eq. (8) with DCEM.

$$\theta \leftarrow \text{grad-update}(\nabla_\theta C_\theta(\hat{z}))$$

▷ Update the decoder to improve the controller's cost.

**end while**

---

control sequences over this horizon length, e.g.  $\mathcal{U}$  could be a multi-dimensional real space or box therein and  $\mathcal{U}^H$  could be the Cartesian product of those spaces representing the sequence of controls over  $H$  timesteps. We are interested in repeatedly solving the control optimization problem<sup>2</sup>

$$\begin{aligned} \hat{u}_{1:H} &:= \arg \min_{u_{1:H} \in \mathcal{U}^H} \sum_{t=1}^H C_t(x_t, u_t) \\ \text{subject to } &x_1 = x_{\text{init}} \\ &x_{t+1} = f^{\text{trans}}(x_t, u_t) \end{aligned} \quad (7)$$

where we are in an initial system state  $x_{\text{init}}$  governed by deterministic system *transition dynamics*  $f^{\text{trans}}$ , and wish to find the optimal sequence of actions  $\hat{u}_{1:H}$  such that we find a valid trajectory  $\{x_{1:H}, u_{1:H}\}$  that optimizes the cost  $C_t(x_t, u_t)$ . Equation (7) can be seen as an instance of eq. (1) by moving the rollout of the dynamics into the cost function. Typically these controllers are used for *receding horizon control* (Mayne & Michalska, 1990) where only the first action  $u_1$  is deployed on the real system, a new state is obtained from the system, and the eq. (7) is solved again from the new initial state. In this case we can say the controller induces a *policy*  $\pi(x_{\text{init}}) := \hat{u}_1$ <sup>3</sup> that solves eq. (7) and depends on the cost and transition dynamics, and potential parameters therein. In all of the cases we consider  $f^{\text{trans}}$  is deterministic, but may be approximated by a stochas-

tic model for learning. Some *model-based* reinforcement learning settings consider cases where  $f^{\text{trans}}$  and  $C$  are parameterized and potentially used in conjunction with another policy class.

For sufficiently complex dynamical systems, eq. (7) is computationally expensive and numerically unstable to solve and rife with sub-optimal local minima. The cross-entropy method is the state-of-the-art method for solving eq. (7) with neural network transitions  $f^{\text{trans}}$  (Chua et al., 2018; Hafner et al., 2018). CEM in this context samples full action sequences and refines the samples towards ones that solve the control problem. Hafner et al. (2018) uses CEM with 1000 samples in each iteration for 10 iterations with a horizon length of 12. This requires  $1000 \times 10 \times 12 = 120,000$  evaluations (!) of the transition dynamics to predict the control to be taken given a system state — and the transition dynamics may use a deep recurrent architecture as in Hafner et al. (2018) or an ensemble of models as in Chua et al. (2018). One comparison point here is a model-free neural network policy takes a *single* evaluation for this prediction, albeit sometimes with a larger neural network.

The first application we show of DCEM in the continuous control setting is to learn a *latent action space*  $\mathcal{Z}$  with a parameterized *decoder*  $f_\theta^{\text{dec}} : \mathcal{Z} \rightarrow \mathcal{U}^H$  that maps back up to the space of *optimal action sequences*, which we illustrate in fig. 8. For simplicity starting out, assume that the dynamics and cost functions are known (and perhaps even the ground-truth) and that the only problem is to estimate

<sup>2</sup>We omit some explicit variables from the arg min operator when they can be inferred by the context.

<sup>3</sup>We also omit the dependency of  $u_1$  on  $x_{\text{init}}$ .

the decoder in isolation, although we will show later that these assumptions can be relaxed. The motivation for having such a latent space and decoder is that the millions of times eq. (7) is being solved for the same dynamic system with the same cost, the solution space of *optimal* action sequences  $\hat{u}_{1:H} \in \mathcal{U}^H$  has an extremely large amount of *spatial* (over  $\mathcal{U}$ ) and *temporal* (over time in  $\mathcal{U}^H$ ) structure that is being ignored by CEM on the full space. The space of optimal action sequences only contains the knowledge of the trajectories that matter for solving the task at hand, such as different parts of an optimal gait, and not irrelevant control sequences. We argue that CEM over the full action space wastes a lot of computation considering irrelevant action sequences and show that these can be ignored by learning a latent space of more reasonable candidate solutions here that we search over instead. Given a decoder, the control optimization problem in eq. (7) can then be transformed into an optimization problem over  $\mathcal{Z}$  as

$$\hat{z} := \arg \min_{z \in \mathcal{Z}} C_\theta(z; x_{\text{init}}) := \sum_{t=1}^H C_t(x_t, u_t) \quad (8)$$

subject to  $x_1 = x_{\text{init}}$

$$x_{t+1} = f^{\text{trans}}(x_t, u_t)$$

$$u_{1:H} = f_\theta^{\text{dec}}(z)$$

which is still a challenging non-convex optimization problem that searches over a decoder’s input space to find the optimal control sequence. Equation (8) can be seen as an instance of eq. (1) by moving the decoder and rollout of the dynamics into the cost function  $C_\theta(z; x_{\text{init}})$  and can be solved with CEM and DCEM. We notationally leave out the dependence of  $\hat{z}$  on  $x_{\text{init}}$  and  $\theta$ .

We propose in alg. 2 to use DCEM to approximately solve eq. (8) and then *learn* the decoder directly to optimize the performance of eq. (7). Every time we solve eq. (8) with DCEM and obtain an optimal latent representation  $\hat{z}$  along with the induced trajectory  $\{x_t, u_t\}$ , we can take a gradient step to push down the resulting cost of that trajectory with  $\nabla_\theta C(\hat{z})$ , which goes through the DCEM process that uses the decoder to generate samples to obtain  $\hat{z}$ . The DCEM machinery behind this is not necessary *if* a reasonable local minima is consistently found as this is an instance of min-differentiation (Rockafellar & Wets, 2009, Theorem 10.13) but in practice this breaks down in non-convex cases when the minimum cannot be consistently found. DCEM helps by providing the derivative information  $\nabla_\theta \hat{z}$ . Antonova et al. (2019); Wang & Ba (2019) solve related problems in this space and we discuss them in sect. 2.3. Learning an action embedding also requires derivatives through the transition dynamics and cost functions to compute  $\nabla_\theta C(\hat{z})$ , even if the ground-truth dynamics are being used. This gives the latent space the knowledge of how the control cost will change as the decoder’s parameters change.

DCEM in this setting also induces a *differentiable policy class*  $\pi(x_{\text{init}}) := u_1 = f^{\text{dec}}(\hat{z})_1$ . This enables a policy or imitation loss  $\mathcal{J}$  to be defined on the policy that can fine-tune the parts of the controller (decoder, cost, and transition dynamics) gradient information from  $\nabla_\theta \mathcal{J}$ . In theory the same approach could be used with CEM on the full optimization problem in eq. (7). For realistic problems without modification this is intractable and memory-intensive as it would require storing and backpropagating through every sampled trajectory, although as a future direction we note that it may be possible to delete some of the low-influence trajectories to help overcome this.

## 5. Experiments

Our experiments demonstrate applications of the cross-entropy method in structured prediction, control, and reinforcement learning. sect. 5.1 illustrate a synthetic regression structured prediction task where gradient descent learns a counter-intuitive energy surface while DCEM retains the minimum. sect. 5.2 shows how DCEM can embed control optimization problems in a case when the ground-truth model is known or unknown, and we show that PPO (Schulman et al., 2017) can help improve the embedded controller.

Our PyTorch (Paszke et al., 2019) source code is openly available at [github.com/facebookresearch/dcem](https://github.com/facebookresearch/dcem) and uses the PyTorch LML implementation from [github.com/locuslab/lml](https://github.com/locuslab/lml) to compute eq. (4).

### 5.1. Unrolling optimizers for regression and structured prediction

In this section we briefly explore the impact of the inner optimizer on the energy surface of a SPEN as discussed in sect. 4.1. For illustrative purposes we consider a simple unidimensional regression task where the *ground-truth data* is generated from  $f(x) := x \sin(x)$  for  $x \in [0, 2\pi]$ . We model  $\mathbb{P}(y|x) \propto \exp\{-E_\theta(y|x)\}$  with a single neural network  $E_\theta$  and make predictions  $\hat{y}$  by solving the optimization problem eq. (6). Given the ground-truth output  $y^*$ , we use the loss  $\mathcal{L}(\hat{y}, y^*) := \|\hat{y} - y^*\|_2^2$  and take gradient steps of this loss to shape the energy landscape.

We consider approximating eq. (6) with unrolled gradient descent and DCEM with Gaussian sampling distributions. Both of these are trained to take 10 optimizer steps and we use an inner learning rate of 0.1 for gradient descent and with DCEM we use 10 iterations with 100 samples per iteration and 10 elite candidates, with a temperature of 1. For both algorithms we start the initial iterate at  $y_0 := 0$ . We show in app. B that both of these models attain the same loss on the training dataset but, since this is a unidimensional regression task, we can visualize the entire energy surfaces over the joint input-output space in fig. 2.

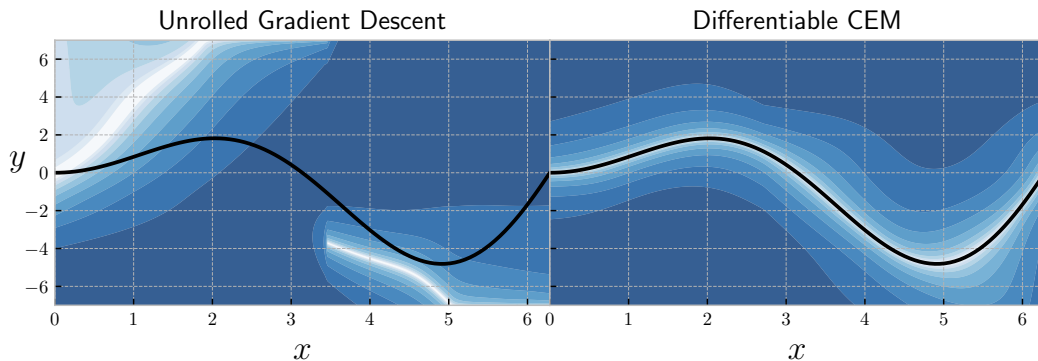


Figure 2. We trained an energy-based model with unrolled gradient descent and DCEM for 1D regression onto the black target function. Each method unrolls through 10 optimizer steps. The contour surfaces show the (normalized/log-scaled) energy surfaces, highlighting that unrolled gradient descent models can overfit to the number of gradient steps. The lighter colors show areas of lower energy.

This shows that gradient descent has learned to adapt from the initial  $y_0$  position to the final position by descending along the function’s surface as we would expect, but there is no reason why the energy surface should be a local minimum around the last iterate  $\hat{y} := y_{10}$ . The energy surface learned by CEM captures local minima around the regression target as the sequence of Gaussian iterates are able to capture a more global view of the function landscape and need to focus in on a minimum of it for regression. We show ablations in app. B from training for 10 inner iterations and then evaluating with a different number of iterations and show that gradient descent quickly steps away from making reasonable predictions.

**Discussion.** Other tricks *could* be used to force the output to be at a local minimum with gradient descent, such as using multiple starting points or randomizing the number of gradient descent steps taken — our intention here is to highlight this behavior in the vanilla case. DCEM is also susceptible to overfitting to the hyper-parameters behind it in similar, albeit less obvious ways.

## 5.2. Control

### 5.2.1. STARTING SIMPLE: EMBEDDING THE CARPOLE’S ACTION SPACE

We first show that it is possible to learn an embedded control space as discussed in sect. 4.2 in an isolated setting. We use the standard cartpole dynamical system from Barto et al. (1983) with a *continuous* state-action space. We assume that the ground-truth dynamics and cost are *known* and use the differentiable ground-truth dynamics and cost implemented in PyTorch from Amos et al. (2018). This isolates the learning problem to *only* learning the embedding so that we can study what this is doing without the additional complications that arise from exploration, estimating the dynamics, learning a policy, and other non-stationarities. We show experiments with these assumptions relaxed in sect. 5.2.2.

We use DCEM and alg. 2 to learn a 2-dimensional latent space  $\mathcal{Z} := [0, 1]^2$  that maps back up to the full control space  $\mathcal{U}^H := [0, 1]^H$  where we focus on horizons of length  $H := 20$ . For DCEM over the embedded space we use 10 iterations with 100 samples in each iteration and 10 elite candidates, again with a temperature of 1. We show the details in app. C that we *are* able to recover the performance of an expert CEM controller that uses an order-of-magnitude more samples fig. 3 shows a visualization of what the CEM and embedded DCEM iterates look like to solve the control optimization problem from the same initial system state. CEM spends a lot of evaluations on sequences in the control space that are unlikely to be optimal, such as the ones the bifurcate between the boundaries of the control space at every timestep, while our embedded space is able to learn more reasonable proposals.

### 5.2.2. SCALING UP TO CONTINUOUS LOCOMOTION

Next we show that we can relax the assumptions of having known transition dynamics and reward and show that we can learn a latent control space on top of a learned model on the `cheetah.run` and `walker.walk` continuous locomotion tasks from the DeepMind control suite (Tassa et al., 2018) using the MuJoCo physics engine (Todorov et al., 2012). We then fine-tune the policy induced by the embedded controller with PPO (Schulman et al., 2017), sending the policy loss directly back into the reward and latent embedding modules underlying the controller. Videos of our trained models are available at <https://sites.google.com/view/diff-cross-entropy-method>.

We start with a state-of-the-art model-based RL approach by noting that the PlaNet (Hafner et al., 2018) restricted state space model (RSSM) is a reasonable architecture for proprioceptive-based control in addition to just pixel-based control. We show the graphical model we use in fig. 8, which maintains deterministic hidden states  $h_t$  and stochas-

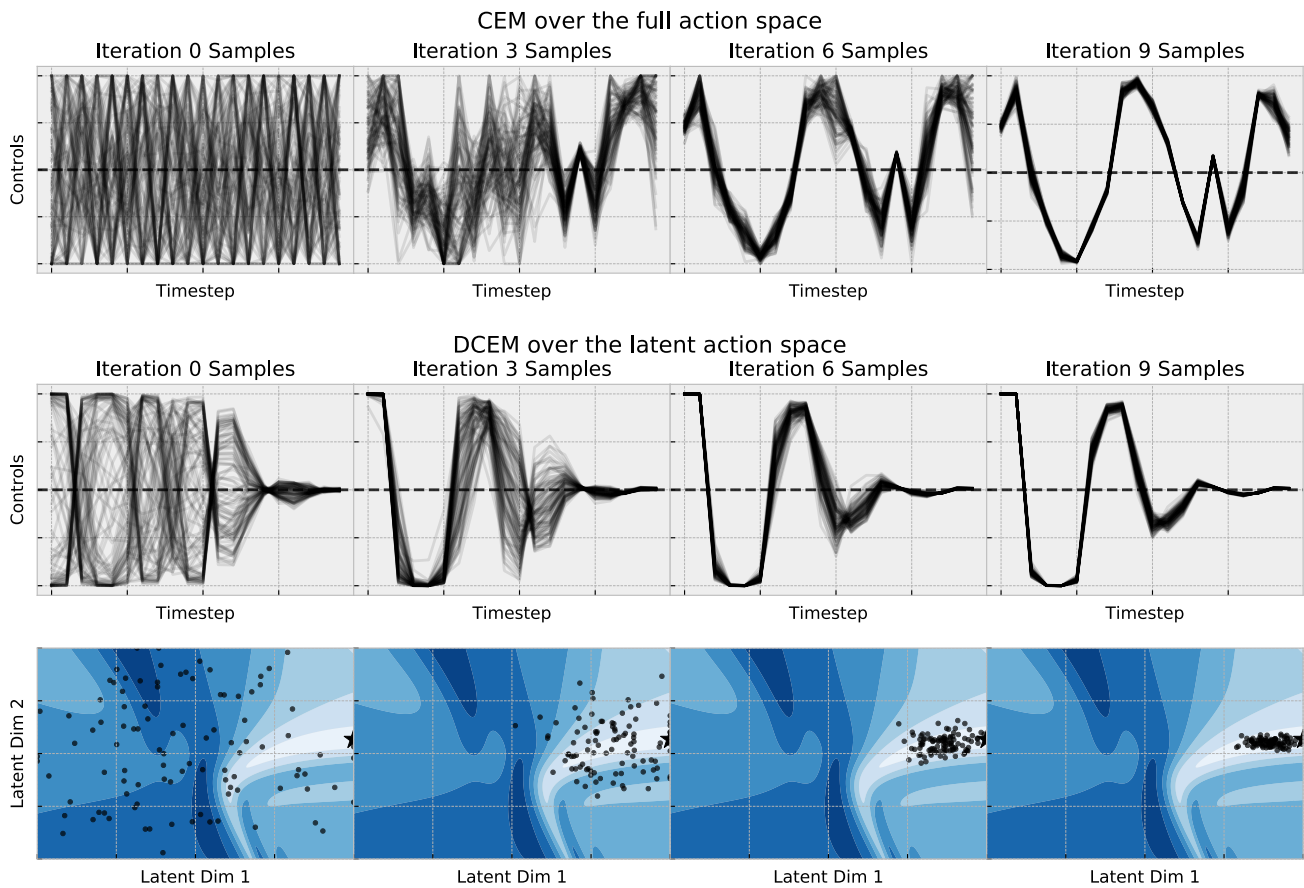


Figure 3. Visualization of the samples that CEM and DCEM generate to solve the cartpole task starting from the same initial system state. The plots starting at the top-left show that CEM initially starts with no temporal knowledge over the control space whereas embedded DCEM’s latent space generates a more feasible distribution over control sequences to consider in each iteration. Embedded DCEM uses an order of magnitude less samples and is able to generate a better solution to the control problem. The contours on the bottom show the controller’s cost surface  $C(z)$  from eq. (8) for the initial state — the lighter colors show regions with lower costs.

tic (proprioceptive) system observations  $x_t$  and rewards  $r_t$ . We model transitions as  $h_{t+1} = f_{\theta}^{\text{trans}}(h_t, x_t)$ , observations with  $x_t \sim f_{\theta}^{\text{dec}}(h_t)$ , rewards with  $r_t = f_{\theta}^{\text{rew}}(h_t, x_t)$ , and map from the latent action space to action sequences with  $u_{1:T} = f^{\text{dec}}(z)$ . We follow the online training procedure of Hafner et al. (2018) to initialize all of the models except for the action decoder  $f^{\text{dec}}$ , using approximately 2M timesteps. We then use a variant of alg. 2 to learn  $f^{\text{dec}}$  to embed the action space for control with DCEM, which we also do online while updating the models. We describe the full training process in app. D.

Our DCEM controller induces a differentiable policy class  $\pi_{\theta}(x_{\text{init}})$  where  $\theta$  are the parameters of the models that impact the actions that the controller is selecting. We then use PPO to define a loss on top of this policy class and fine-tune the components (the decoder and reward module) so that they improve the episode reward rather than the maximum-likelihood solution of observed trajectories. We chose PPO because we thought it would be able to fine-tune the policy

with just a few updates because the policy is starting at a reasonable point, but this did not turn out to be the case and in the future other policy optimizers can be explored. We implement this by making our DCEM controller the policy in the PPO implementation by Kostrikov (2018). We provide more details behind our training procedure in app. D.

We evaluate our controllers on 100 test episodes and the rewards in fig. 4 show that DCEM is almost (but not exactly) able to recover the performance of doing CEM over the full action space while using an order-of-magnitude less trajectory samples (1,000 vs 10,000). PPO fine-tuning helps bridge the gap between the performances.

**Discussion.** The future directions of DCEM in the control setting will help bring efficiency and policy-based fine-tuning to model-based reinforcement learning. Much more analysis and experimentation is necessary to achieve this as we faced many issues getting the model-based cheetah and walker tasks to work that did not arise in the ground-



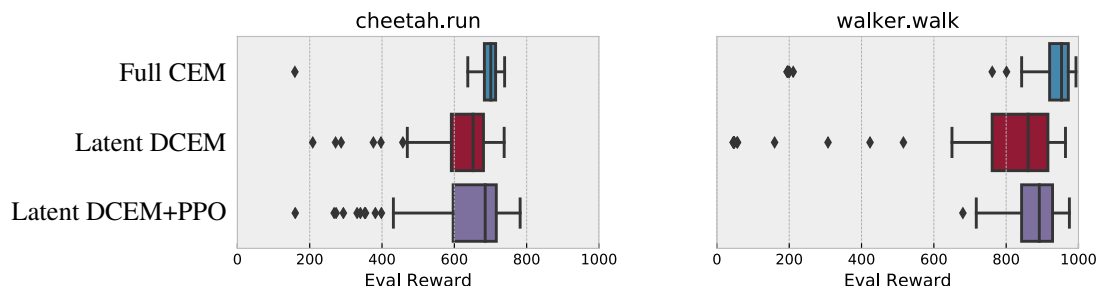


Figure 4. We evaluated our final models by running 100 episodes each on the cheetah and walker tasks. CEM over the full action space uses 10,000 trajectories for control at each time step while embedded DCEM samples only 1000 trajectories. DCEM almost recovers the performance of CEM over the full action space and PPO fine-tuning of the model-based components helps bridge the gap.

truth cartpole task. We discuss this more in app. D. We also did not focus on the sample complexity of our algorithms getting these proof-of-concept experiments working. Other reasonable baselines on this task could involve distilling the controller into a model-free policy and then doing search on top of that policy, as done in POPLIN (Wang & Ba, 2019).

## 6. Conclusions and Future Directions

We have shown how to differentiate through the cross-entropy method and have brought CEM into the end-to-end learning pipeline. Beyond further explorations in the energy-based learning and control contexts we showed here, DCEM can be used anywhere gradient descent is unrolled. We find this especially promising for meta-learning and can build on LEO (Rusu et al., 2018) or CAVIA (Zintgraf et al., 2019). Inspired by DCEM, other more powerful sampling-based optimizers could be made differentiable in the same way, potentially optimizers that leverage gradient-based information in the inner optimization steps (Sekhon & Mebane, 1998; Theodorou et al., 2010; Stulp & Sigaud, 2012; Maheswaranathan et al., 2018) or by also learning the hyper-parameters of structured optimizers (Li & Malik, 2016; Volpp et al., 2019; Chen et al., 2017).

### Acknowledgments

We thank David Belanger, Roberto Calandra, Yinlam Chow, Rob Fergus, Mohammad Ghavamzadeh, Edward Grefenstette, Shubhanshu Shekhar, and Zoltán Szabó for insightful discussions, and the anonymous reviewers for many useful suggestions and improvements to this paper.

We acknowledge the Python community (Van Rossum & Drake Jr, 1995; Oliphant, 2007) for developing the core set of tools that enabled this work, including PyTorch (Paszke et al., 2019), Hydra (Yadan, 2019), Jupyter (Kluyver et al., 2016), Matplotlib (Hunter, 2007), seaborn (Waskom et al., 2018), numpy (Oliphant, 2006; Van Der Walt et al., 2011), pandas (McKinney, 2012), and SciPy (Jones et al., 2014).

## References

- Agrawal, A., Amos, B., Barratt, S., Boyd, S., Diamond, S., and Kolter, J. Z. Differentiable convex optimization layers. In *Advances in neural information processing systems*, pp. 9562–9574, 2019a.
- Agrawal, A., Barratt, S., Boyd, S., and Stellato, B. Learning convex optimization control policies. *arXiv preprint arXiv:1912.09529*, 2019b.
- Amos, B. and Kolter, J. Z. Optnet: Differentiable optimization as a layer in neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 136–145. JMLR. org, 2017.
- Amos, B., Xu, L., and Kolter, J. Z. Input convex neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 146–155. JMLR. org, 2017.
- Amos, B., Jimenez, I., Sacks, J., Boots, B., and Kolter, J. Z. Differentiable mpc for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, pp. 8289–8300, 2018.
- Amos, B., Koltun, V., and Kolter, J. Z. The limited multi-label projection layer. *arXiv preprint arXiv:1906.08707*, 2019.
- Anand, A., Racah, E., Ozair, S., Bengio, Y., Côté, M.-A., and Hjelm, R. D. Unsupervised state representation learning in atari. *arXiv preprint arXiv:1906.08226*, 2019.
- Antonova, R., Rai, A., Li, T., and Kragic, D. Bayesian optimization in variational latent spaces with dynamic compression. *arXiv preprint arXiv:1907.04796*, 2019.
- Banijamali, E., Shu, R., Ghavamzadeh, M., Bui, H., and Ghodsi, A. Robust locally-linear controllable embedding. *arXiv preprint arXiv:1710.05373*, 2017.
- Bansal, S., Calandra, R., Xiao, T., Levine, S., and Tomlin, C. J. Goal-driven dynamics learning via Bayesian optimization. In *IEEE Conference on Decision and Control (CDC)*, pp. 5168–5173, 2017.
- Barto, A. G., Sutton, R. S., and Anderson, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, pp. 834–846, 1983.

- Belanger, D. and McCallum, A. Structured prediction energy networks. In *International Conference on Machine Learning*, pp. 983–992, 2016.
- Belanger, D., Yang, B., and McCallum, A. End-to-end learning for structured prediction energy networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 429–439. JMLR. org, 2017.
- Ben Salem, M., Bachoc, F., Roustant, O., Gamboa, F., and Tomaso, L. Sequential dimension reduction for learning features of expensive black-box functions. working paper or preprint, February 2019. URL <https://hal.archives-ouvertes.fr/hal-01688329>.
- Brookes, D. H. and Listgarten, J. Design by adaptive sampling. *arXiv preprint arXiv:1810.03714*, 2018.
- Calandra, R., Peters, J., Rasmussen, C. E., and Deisenroth, M. P. Manifold gaussian processes for regression. In *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 3338–3345. IEEE, 2016.
- Chandak, Y., Theocharous, G., Kostas, J., Jordan, S., and Thomas, P. S. Learning action representations for reinforcement learning. *arXiv preprint arXiv:1902.00183*, 2019.
- Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillcrap, T. P., Botvinick, M., and de Freitas, N. Learning to learn without gradient descent by gradient descent. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 748–756. JMLR. org, 2017.
- Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- Chua, K., Calandra, R., McAllister, R., and Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pp. 4754–4765, 2018.
- Co-Reyes, J. D., Liu, Y., Gupta, A., Eysenbach, B., Abbeel, P., and Levine, S. Self-consistent trajectory autoencoder: Hierarchical reinforcement learning with trajectory embeddings. *arXiv preprint arXiv:1806.02813*, 2018.
- De Boer, P.-T., Kroese, D. P., Mannor, S., and Rubinstein, R. Y. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.
- Domke, J. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pp. 318–326, 2012.
- Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.
- Foerster, J., Chen, R. Y., Al-Shedivat, M., Whiteson, S., Abbeel, P., and Mordatch, I. Learning with opponent-learning awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 122–130. International Foundation for Autonomous Agents and Multiagent Systems, 2018.
- Friedman, J., Hastie, T., and Tibshirani, R. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.
- Garnett, R., Osborne, M. A., and Hennig, P. Active learning of linear embeddings for gaussian processes. *arXiv preprint arXiv:1310.6740*, 2013.
- Gelada, C., Kumar, S., Buckman, J., Nachum, O., and Bellemare, M. G. Deepmdp: Learning continuous latent space models for representation learning. *arXiv preprint arXiv:1906.02736*, 2019.
- Ghosh, D., Gupta, A., and Levine, S. Learning actionable representations with goal-conditioned policies. *arXiv preprint arXiv:1811.07819*, 2018.
- Gómez-Bombarelli, R., Wei, J. N., Duvenaud, D., Hernández-Lobato, J. M., Sánchez-Lengeling, B., Sheberla, D., Aguilera-Iparraguirre, J., Hirzel, T. D., Adams, R. P., and Aspuru-Guzik, A. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.
- Gould, S., Fernando, B., Cherian, A., Anderson, P., Cruz, R. S., and Guo, E. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *arXiv preprint arXiv:1607.05447*, 2016.
- Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. Learning latent dynamics for planning from pixels. *arXiv preprint arXiv:1811.04551*, 2018.
- Hu, F. and Zidek, J. V. The weighted likelihood. *Canadian Journal of Statistics*, 30(3):347–371, 2002.
- Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90, 2007.
- Ichter, B. and Pavone, M. Robot motion planning in learned latent spaces. *IEEE Robotics and Automation Letters*, 4(3):2407–2414, 2019.
- Ichter, B., Harrison, J., and Pavone, M. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7087–7094. IEEE, 2018.
- Jenni, S. and Favaro, P. Deep bilevel learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 618–633, 2018.
- Johnson, M., Duvenaud, D. K., Wiltchko, A., Adams, R. P., and Datta, S. R. Composing graphical models with neural networks for structured representations and fast inference. In *Advances in neural information processing systems*, pp. 2946–2954, 2016.
- Jones, E., Oliphant, T., and Peterson, P. {SciPy}: Open source scientific tools for {Python}. 2014.
- Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- Kirschner, J., Mutný, M., Hiller, N., Ischebeck, R., and Krause, A. Adaptive and safe bayesian optimization in high dimensions via one-dimensional subspaces. *arXiv preprint arXiv:1902.03229*, 2019.

- Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J. B., Grout, J., Corlay, S., et al. Jupyter notebooks-a publishing format for reproducible computational workflows. In *ELPUB*, pp. 87–90, 2016.
- Kostrikov, I. Pytorch implementations of reinforcement learning algorithms. <https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail>, 2018.
- Kurutach, T., Tamar, A., Yang, G., Russell, S. J., and Abbeel, P. Learning plannable representations with causal infogan. In *Advances in Neural Information Processing Systems*, pp. 8733–8744, 2018.
- Lambert, N., Amos, B., Yadan, O., and Calandra, R. Objective mismatch in model-based reinforcement learning. *arXiv preprint arXiv:2002.04523*, 2020.
- LeCun, Y., Chopra, S., Hadsell, R., Ranzato, M., and Huang, F. A tutorial on energy-based learning. *Predicting structured data*, 1(0), 2006.
- Levine, N., Chow, Y., Shu, R., Li, A., Ghavamzadeh, M., and Bui, H. Prediction, consistency, curvature: Representation learning for locally-linear control. *arXiv preprint arXiv:1909.01506*, 2019.
- Li, K. and Malik, J. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.
- Luo, R., Tian, F., Qin, T., Chen, E., and Liu, T.-Y. Neural architecture optimization. In *Advances in neural information processing systems*, pp. 7816–7827, 2018.
- Lynch, C., Khansari, M., Xiao, T., Kumar, V., Tompson, J., Levine, S., and Sermanet, P. Learning latent plans from play. *arXiv preprint arXiv:1903.01973*, 2019.
- Maheswaranathan, N., Metz, L., Tucker, G., Choi, D., and Sohl-Dickstein, J. Guided evolutionary strategies: Augmenting random search with surrogate gradients. *arXiv preprint arXiv:1806.10230*, 2018.
- Malaviya, C., Ferreira, P., and Martins, A. F. Sparse and constrained attention for neural machine translation. *arXiv preprint arXiv:1805.08241*, 2018.
- Markatou, M., Basu, A., and Lindsay, B. Weighted likelihood estimating equations: The discrete case with applications to logistic regression. *Journal of Statistical Planning and Inference*, 57(2): 215–232, 1997.
- Markatou, M., Basu, A., and Lindsay, B. G. Weighted likelihood equations with bootstrap root search. *Journal of the American Statistical Association*, 93(442):740–750, 1998.
- Martins, A. F. and Kreutzer, J. Learning what’s easy: Fully differentiable neural easy-first taggers. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 349–362, 2017.
- Mayne, D. Q. and Michalska, H. Receding horizon control of nonlinear systems. *IEEE Transactions on automatic control*, 35(7):814–824, 1990.
- McKinney, W. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython*. " O’Reilly Media, Inc.", 2012.
- Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. Unrolled generative adversarial networks. *CoRR*, abs/1611.02163, 2016. URL <http://arxiv.org/abs/1611.02163>.
- Miladinović, Đ., Gondal, M. W., Schölkopf, B., Buhmann, J. M., and Bauer, S. Disentangled state space representations. *arXiv preprint arXiv:1906.03255*, 2019.
- Mohamed, S., Rosca, M., Figurnov, M., and Mnih, A. Monte carlo gradient estimation in machine learning. *arXiv preprint arXiv:1906.10652*, 2019.
- Mukadam, M., Dong, J., Yan, X., Dellaert, F., and Boots, B. Continuous-time gaussian process motion planning via probabilistic inference. *The International Journal of Robotics Research*, 37(11):1319–1340, 2018.
- Oh, C., Gavves, E., and Welling, M. Bock: Bayesian optimization with cylindrical kernels. *arXiv preprint arXiv:1806.01619*, 2018.
- Okada, M., Rigazio, L., and Aoshima, T. Path integral networks: End-to-end differentiable optimal control. *arXiv preprint arXiv:1706.09597*, 2017.
- Oliphant, T. E. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.
- Oliphant, T. E. Python for scientific computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pp. 8026–8037, 2019.
- Pedregosa, F. Hyperparameter optimization with approximate gradient. *arXiv preprint arXiv:1602.02355*, 2016.
- Pereira, M., Fan, D. D., An, G. N., and Theodorou, E. Mpc-inspired neural network policies for sequential decision making. *arXiv preprint arXiv:1802.05803*, 2018.
- Rajeswaran, A., Finn, C., Kakade, S., and Levine, S. Meta-learning with implicit gradients. *arXiv preprint arXiv:1909.04630*, 2019.
- Rao, C. R. Convexity properties of entropy functions and analysis of diversity. *Lecture Notes-Monograph Series*, pp. 68–77, 1984.
- Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.
- Rockafellar, R. T. and Wets, R. J.-B. *Variational analysis*, volume 317. Springer Science & Business Media, 2009.
- Rubinstein, R. Y. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997.
- Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.
- Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

- Sekhon, J. S. and Mebane, W. R. Genetic optimization using derivatives. *Political Analysis*, 7:187–210, 1998.
- Singh, S., Richards, S. M., Sindhvani, V., Slotine, J.-J. E., and Pavone, M. Learning stabilizable nonlinear dynamics with contraction-based regularization. *arXiv preprint arXiv:1907.13122*, 2019.
- Srinivas, A., Jabri, A., Abbeel, P., Levine, S., and Finn, C. Universal planning networks. *arXiv preprint arXiv:1804.00645*, 2018.
- Stulp, F. and Sigaud, O. Path integral policy improvement with covariance matrix adaptation. *arXiv preprint arXiv:1206.4621*, 2012.
- Tasfi, N. and Capretz, M. Dynamic planning networks. *arXiv preprint arXiv:1812.11240*, 2018.
- Taskar, B., Chatalbashev, V., Koller, D., and Guestrin, C. Learning structured prediction models: A large margin approach. In *Proceedings of the 22nd international conference on Machine learning*, pp. 896–903. ACM, 2005.
- Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.
- Theodorou, E., Buchli, J., and Schaal, S. A generalized path integral control approach to reinforcement learning. *The Journal of Machine Learning Research*, 11:3137–3181, 2010.
- Titsias, M. and Lázaro-Gredilla, M. Doubly stochastic variational bayes for non-conjugate inference. In *International conference on machine learning*, pp. 1971–1979, 2014.
- Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033. IEEE, 2012.
- Van Der Walt, S., Colbert, S. C., and Varoquaux, G. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.
- Van Rossum, G. and Drake Jr, F. L. *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.
- Volpp, M., Fröhlich, L., Doerr, A., Hutter, F., and Daniel, C. Meta-learning acquisition functions for bayesian optimization. *arXiv preprint arXiv:1904.02642*, 2019.
- Wang, S. X. *Maximum weighted likelihood estimation*. PhD thesis, University of British Columbia, 2001.
- Wang, T. and Ba, J. Exploring model-based planning with policy networks. *arXiv preprint arXiv:1906.08649*, 2019.
- Wang, Z., Hutter, F., Zoghi, M., Matheson, D., and de Freitas, N. Bayesian optimization in a billion dimensions via random embeddings. *Journal of Artificial Intelligence Research*, 55: 361–387, 2016.
- Waskom, M., Botvinnik, O., O’Kane, D., Hobson, P., Ostblom, J., Lukauskas, S., Gemperline, D. C., Augspurger, T., Halchenko, Y., Cole, J. B., Warmenhoven, J., de Ruiter, J., Pye, C., Hoyer, S., Vanderplas, J., Villalba, S., Kunter, G., Quintero, E., Bachant, P., Martin, M., Meyer, K., Miles, A., Ram, Y., Brunner, T., Yarkoni, T., Williams, M. L., Evans, C., Fitzgerald, C., Brian, and Qalieh, A. mwaskom/seaborn: v0.9.0 (july 2018), July 2018. URL <https://doi.org/10.5281/zenodo.1313201>.
- Watter, M., Springenberg, J., Boedecker, J., and Riedmiller, M. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in neural information processing systems*, pp. 2746–2754, 2015.
- Yadan, O. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL <https://github.com/facebookresearch/hydra>.
- Yu, T., Shevchuk, G., Sadigh, D., and Finn, C. Unsupervised visuomotor control through distributional planning networks. *arXiv preprint arXiv:1902.05542*, 2019.
- Zhang, M., Vikram, S., Smith, L., Abbeel, P., Johnson, M. J., and Levine, S. Solar: Deep structured latent representations for model-based reinforcement learning. *arXiv preprint arXiv:1808.09105*, 2018.
- Zhang, Y., Hare, J., and Prügel-Bennett, A. Deep set prediction networks. *arXiv preprint arXiv:1906.06565*, 2019.
- Zintgraf, L., Shiarli, K., Kurin, V., Hofmann, K., and Whiteson, S. Fast context adaptation via meta-learning. In *International Conference on Machine Learning*, pp. 7693–7702, 2019.