European Workshop on Reinforcement Learning 14 (2018) October 2018, Lille, France.

# ACCME: Actively Compressed Conditional Mean Embeddings for Model-Based Reinforcement Learning

**Ronnie Stafford** R.STAFFORD.12@UCL.AC.UK

**John Shawe-Taylor** J.SHAWE-TAYLOR@UCL.AC.UK
*Department of Computer Science*
*University College London*
*London, UK*

## Abstract

We present a novel approach for integrating deep non-linear parametric function approximators into an existing reinforcement learning (RL) control algorithm while maintaining stable policy updates. *Actively compressed conditional mean embeddings* (ACCME) replaces computationally expensive batch kernel regression with a stochastically-trained neural network architecture for learning the kernel weights of a conditional mean embedding (CME) transition model. The embeddings model is then used in a model-based dynamic programming (DP) control algorithm. The ACCME variant i) improves the practicality of continual training of a CME model in online and data-abundant environments, ii) maintains a fast-evaluated contraction constraint by a sparse softmax activation function. Additionally we propose a neurobiologically-inspired mechanism for adding and *removing* states from the set of successor states that the embedding is defined over. This is in contrast to the original CME and later the compressed CME (CCME) models that only *add* new states to the set, which is problematic for maintaining non-parametric value functions in large Markov decision processes (MDPs).

**Keywords:** Online Model-Based Reinforcement Learning, Neural Networks, Sparsity, Dynamic Programming, Policy Iteration, Continuous Learning.

## 1. Introduction

Existing model-based control algorithms either i) separate transition models and policies (Weber et al., 2017; Ha and Schmidhuber, 2018) in order to plan with simulated rollouts or ii) use models as a source of predicted one-step transitions to augment model-free value function updates (Sutton, 1991; Gu et al., 2016). Both approaches render policy improvement guarantees not entirely forthcoming and in general there is no consensus on how best to integrate rich online models within RL while maintaining stable policy updates.

We focus on an existing model-based *policy iteration* (PI) (Howard, 1960) algorithm whose models are *integral* to the Bellman operator $(T^{\pi}v)(\mathbf{s}) := r(\mathbf{s}, \pi(\mathbf{s})) + \gamma\mathbb{E}_{S' \sim p(\cdot|\mathbf{s},\pi(\mathbf{s}))}[v(\mathbf{s}')]$ (Sutton and Barto, 1998) where value functions $v : \mathcal{S} \to \mathbb{R}$ are defined over all states $\mathbf{s} \in \mathcal{S}$. We assume interaction with a Markov decision process (MDP) $\mathcal{M} := \{\mathcal{S}, \mathcal{A}, p, r, \gamma\}$ possessing an unknown average reward function $r(\mathbf{s}, \mathbf{a}) := \mathbb{E}_{R \sim P_R(\cdot|\mathbf{s},\pi(\mathbf{s}))}[R] \in [0, R_{\max}]$, discrete actions set $\mathcal{A}$, continuous states $\mathcal{S}$ and unknown transition dynamics $p(\mathbf{s}'|\mathbf{s}, \mathbf{a})$. Agents draw actions from a deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ and seek to learn an optimal policy that maximises the collection of cumulative discounted reward. If $\mathcal{S}$ is discrete, both the reward function and the transition dynamics $E(\mathbf{s},\mathbf{a}):=\mathbb{E}_{S'\sim p(\cdot|\mathbf{s},\mathbf{a})}[v(\mathbf{s}')]$ are known, then through contraction arguments (Szepesvári, 2010) the Bellman operator $T^\pi$ is non-expansive in $||\cdot||_\infty$. Then $\mathbf{v}_p:=(T^\pi)^p\mathbf{v}_0$ where $\mathbf{v}_p=[v_p(\mathbf{s}_1),...,v_p(\mathbf{s}_{|\mathcal{S}|})]^\top$ and $\mathbf{v}^\pi=\lim_{p\to\infty}(\mathbf{v}_p)$ is the fixed point of $T^\pi$, which is PI's *policy evaluation* step. The *policy improvement* step performs a greedy update $\pi_{v_p}(\mathbf{s}):=\arg\max_{\mathbf{a}\in\mathcal{A}}[r(\mathbf{s},\mathbf{a})+\gamma\mathbb{E}_{S'\sim p(\cdot|\mathbf{s},\pi(\mathbf{s}))}[v_p(\mathbf{s}')]]$. By alternating between policy evaluation and policy improvement steps, the policy converges to the optimal policy.

Each PI iteration scales $\sim\mathcal{O}(|\mathcal{S}|^2)$ and is therefore intractable for continuous state spaces. Usually a linear function approximation $v(\mathbf{s})\approx\langle v,\boldsymbol{\phi}(\mathbf{s})\rangle_\mathcal{F}$ is adopted in some function space $\mathcal{F}\subset\mathbb{R}^\mathcal{S}$ with state features $\boldsymbol{\phi}:\mathcal{S}\to\mathcal{F}$. Traditional approximate DP chooses a parametric $\mathcal{F}$ where an *explicit* $\phi$ is either assumed a-priori or learnt. However in the policy evaluation step this approximate scheme requires solving the projected Bellman equations (Bertsekas, 2011) and may also induce policy chatter (Gordon, 1995; Munos, 2003).

## 1.1 Approximate PI with Finite Induced Pseudo MDPs

Algorithms that work with *non-parametric* $\mathcal{F}$ whose $\boldsymbol{\phi}$ is *implicit*, allow value functions to be defined over a discrete set of states. This gives the opportunity to solve the Bellman equation exactly on these discrete states, even if $\mathcal{S}$ is continuous. By choosing $\mathcal{F}$ as a reproducing kernel Hilbert space (RKHS) (Shawe-Taylor and Cristianini, 2004) $\mathcal{H}_L$ with kernel $L{:}\mathcal{S}\times\mathcal{S}\to\mathbb{R}$, a CME (Grünewälder et al., 2012a) approximates the expectation in $T^\pi$ by

$$\begin{aligned}\mathbb{E}_{S'\sim p(\cdot|\mathbf{s},\mathbf{a})}[v(S')] &\approx \langle v,\hat{\mu}_{\text{CME}}(\mathbf{s},\mathbf{a})\rangle_{\mathcal{H}_L}=\Big\langle v,\sum_{j=1}^{n_k}\alpha_j(\mathbf{s},\mathbf{a})\boldsymbol{\phi}(\mathbf{s}'_j)\Big\rangle_{\mathcal{H}_L}, \\ &=\sum_{j=1}^{n_k}\alpha_j(\mathbf{s},\mathbf{a})v(\mathbf{s}'_j)=:\hat{E}_{\hat{\mu}}(\mathbf{s},\mathbf{a}),\quad \mathbf{s}'{\in}\mathcal{S}_k, n_k{=}|\mathcal{S}_k|, \end{aligned}\tag{1}$$

where $\mathcal{S}_k$ is the discrete set of successor states in the transition data $\mathcal{D}_k:=\{(\mathbf{s},\mathbf{a},r,\mathbf{s}')_i\}_{i=1}^{n_k}$ collected up to the $k^{\text{th}}$ policy iteration from interacting with $\mathcal{M}$. The second line (equation (1)) is due to the reproducing property (Aronszajn, 1950), $\hat{\mu}_{\text{CME}}$ is the empirical conditional mean embedding (Song et al., 2009), $L(\mathbf{s}'_i,\mathbf{s}'_j):=\langle\phi(\mathbf{s}'_i),\phi(\mathbf{s}'_j)\rangle_{\mathcal{H}_L}$ is the state kernel evaluation at $(\mathbf{s}'_i,\mathbf{s}'_j)$ and $\boldsymbol{\alpha}:=[\alpha_1,...\alpha_{n_k}]^\top$ are the kernel weights to be determined as a function of $\mathcal{S}\times\mathcal{A}$. Formalised by Yao et al. (2014), replacing the dynamics term in $T^\pi$ with its *finite induced pseudo*-MDP $\hat{E}_\mu$, forms an approximate Bellman operator $\hat{T}^\pi_\mu{:}\mathcal{V}\to\mathcal{V}$ (where $\mathcal{V}{:=}\{v(\mathbf{s}'_1),..,v(\mathbf{s}'_{n_k})\}$). Value functions can then be solved *exactly* by *policy evaluation* over $\mathcal{S}_k$ if there is a contraction mapping enforced by the constraint $||\boldsymbol{\alpha}(\mathbf{s}',\mathbf{a})||_1{\le}1\ \forall(\mathbf{s}',\mathbf{a})\in\mathcal{S}_k{\times}\mathcal{A}$. Greedy policy improvement suboptimality (Singh and Yee, 1994) is established as a function of model accuracy i.e. the more accurate the pseudo MDP to the true dynamics, the less suboptimal the greedy policy update.

The CME algorithm uses batch regularised kernel regression to estimate a pseudo-MDP which is equivalent to calculating the conditional weights $\boldsymbol{\alpha}(\cdot)$. We focus on developing the CME into a fully online algorithm whose model can be continuously updated. We replace batch kernel regression, which scales $\sim\mathcal{O}(n_k^3)$ every time a model update is required, with a neural architecture trained using stochastic gradient descent (SGD). This is tantamount to replacing *some* of the non-parametric model components with a deep neural architecture.

Critically however, we retain the non-parametric characteristics of the value function so that we can still execute *exact* policy evaluation in a pseudo-MDP. This is achieved by i) driving back-propagation with a non-parametric loss function and ii) maintaining the contraction constraint with a sparse softmax activation function. Our method is directly compared throughout to the *compressed* CME (CCME) Lever et al. (2016), which also provides methods to scale the CME algorithm, in particular by the compression of the set $\mathcal{S}_k$.

## 2. ACCME

**Online Dynamics Model** Notation from Lever et al. (2016) is adopted to denote the compression set of successor states $\mathcal{C}_k \subseteq \mathcal{S}_k$ that the compressed version of the embedding (1) is defined over. The explanation of ACCME's method of maintaining $\mathcal{C}_k$ is deferred until below. We modify the (batch regularised) CME empirical risk minimisation problem (Grünewälder et al., 2012b,a) by replacing the conditional kernel weights with a vector-valued neural network $\boldsymbol{\alpha}_{\boldsymbol{\theta}}: \mathcal{S}\times\mathcal{A}\to\mathbb{R}^{|\mathcal{C}_k|}$,

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}}[\frac{1}{2n_k}\sum_{i=1}^{n_k}||\boldsymbol{\Phi}_{\mathcal{C}_k}\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\mathbf{s}_i,\mathbf{a}_i) - \boldsymbol{\phi}(\mathbf{s}'_i)||^2_{\mathcal{F}} + \Omega(\boldsymbol{\theta})], \tag{2}$$

where $\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\cdot) := [\alpha_1(\cdot),..,\alpha_{|\mathcal{C}_k|}(\cdot)]^\top$, $\boldsymbol{\Phi}_{\mathcal{C}_k} := [\phi(\mathbf{s}'_1),..,\phi(\mathbf{s}'_{|\mathcal{C}_k|})]$, network weights $\boldsymbol{\theta}$, $\mathcal{F}=\mathcal{H}_L$ and $\Omega$ is a regulariser whose description is deferred until below. We re-emphasize that $L: \mathcal{S}\times\mathcal{S}\to\mathbb{R}$ is a reproducing kernel such that $L(\mathbf{s}'_m,\mathbf{s}'_n) = \langle\boldsymbol{\phi}(\mathbf{s}'_m),\boldsymbol{\phi}(\mathbf{s}'_n)\rangle_{\mathcal{H}_L}$ which defines a kernel matrix whose entries are $\mathbf{L}_{ij} := L(\mathbf{s}'_i,\mathbf{s}'_j)$. We adopt the notation $\mathbf{L}_{\mathcal{C}_k\mathcal{C}_k}$ as the kernel matrix over the entire compression set at the $k^{th}$ policy iteration and define the vector $\mathbf{L}_{\mathbf{s}'_i\mathcal{C}_k} := [L(\mathbf{s}'_i,\mathbf{s}'_1),...,L(\mathbf{s}'_i,\mathbf{s}'_{|\mathcal{C}_k|})]$. Differentiating equation (2) with respect to a weight $\theta\in\boldsymbol{\theta}$ gives a non-parametric gradient signal,

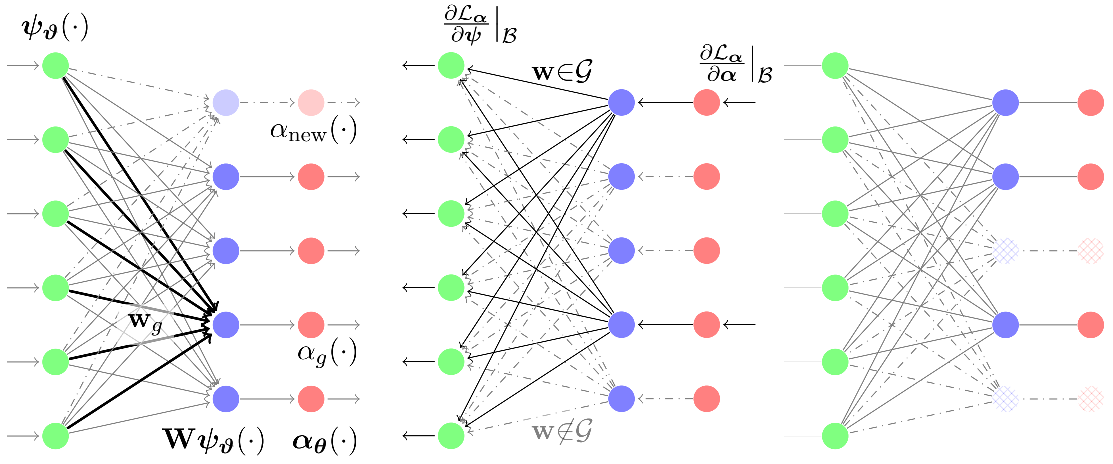
$$\frac{\partial\mathcal{L}_{\boldsymbol{\alpha}}}{\partial\theta}\bigg|_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{\alpha}^\top_{\boldsymbol{\theta}}(\mathbf{s}_i,\mathbf{a}_i)\mathbf{L}_{\mathcal{C}_k\mathcal{C}_k} - \mathbf{L}_{\mathbf{s}'_i\mathcal{C}_k}\right)\frac{\partial\boldsymbol{\alpha}_{\boldsymbol{\theta}}}{\partial\theta}\bigg|_{(\mathbf{s}_i,\mathbf{a}_i)} + \partial_\theta\Omega(\boldsymbol{\theta}), \tag{3}$$

where $\partial_\theta\Omega(\boldsymbol{\theta})$ is a subgradient (applied in a non-standard way whose explanation is deferred until below). The derivative is evaluated over a minibatch $\mathcal{B} := \{(\mathbf{s},\mathbf{a},\mathbf{s}')_i\}_{i=1}^m$ drawn from an experience replay (Mnih et al., 2015) memory $\mathcal{B}\sim\mathcal{D}_k$, which is a repository of all transitions experienced up to the $k^{\text{th}}$ policy iteration. Equation (3) drives a standard backprop that updates weights $\theta \leftarrow \theta - \eta\frac{\partial\mathcal{L}_{\boldsymbol{\alpha}}}{\partial\theta}\big|_{\mathcal{B}} - \eta\partial_\theta\Omega(\boldsymbol{\theta})$ with learning rate $\eta$.

**Architecture and Contraction Constraint** A CME naturally decomposes the kernel weights into $\boldsymbol{\alpha}_{\text{CCME}}(\cdot) = \mathbf{W}\boldsymbol{\psi}(\cdot)$ where $\mathbf{W}$ is a weight matrix and $\boldsymbol{\psi}(\cdot)$ is non-parametric representation over $\mathcal{S}\times\mathcal{A}$ (which for a CCME at iteration $k$ is based on a sparsified set of state-actions $\mathcal{Q}_k$). ACCME parametrises this decomposition as $\boldsymbol{\alpha}_{\boldsymbol{\theta}}(\cdot) = \boldsymbol{\sigma}(\mathbf{W}\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\cdot))$ where $\mathbf{W}\in\mathbb{R}^{|\mathcal{C}_k|\times\dim(\boldsymbol{\psi})}$ is the last layer weights and $\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\cdot)$ is the rest of the network up to the penultimate layer with weights $\boldsymbol{\vartheta}$ (see figure 1 i)). We use the notation $\boldsymbol{\theta} := \{\mathbf{W},\boldsymbol{\vartheta}\}$ as the set of all weights. Setting $\sigma(\cdot) = \textsc{TopNmax}(\cdot) := \textsc{Softmax}\big(\textsc{KeepTopN}(\cdot)\big)$ (Shazeer et al., 2017) as the final layer's activation function, then the pseudo-MDP contraction constraint is satisfied. The network output $\boldsymbol{\alpha}(\mathbf{s},\mathbf{a})\in\mathbb{R}^N$ is fixed $N$-sparse where $N$ is chosen a-priori and sorting the raw activations costs $\sim\mathcal{O}(N|\mathcal{C}_k|)$. Note that Lever et al. (2016) lazily enforce their contraction constraint only when the CCME embedding is evaluated. In contrast

ACCME's constraint is an integral part of the model architecture and is both present during training and is fast-evaluated.

Figure 1: i) Last layer weight groups $\mathbf{w}_g$ are indexed by $g$ and accumulated in $\mathbf{W} := [\mathbf{w}_1, ..., \mathbf{w}_{|\mathcal{C}_k|}]^\top$. When a new state is added $\mathcal{C}_k \leftarrow \mathcal{C}_k \cup \mathbf{s}'_{\text{new}}$, then a new weight group $\mathbf{w}_{\text{new}}$ is initialised and added to $\mathbf{W}$. ii) Backprop a sparse gradient signal; *active* weight groups $\mathbf{w}_g \in \mathcal{G}$ are updated in the standard way, *inactive* $\mathbf{w}_g \notin \mathcal{G}$ weight groups are shrunk by a truncated group lasso update. iii) When after training with $M$ minibatches at the $k^{\text{th}}$ policy iteration, then for all $g$ where $||\mathbf{w}_g||_2$ has remained 0 for the last $\frac{3}{4}M$ minibatches, $\mathbf{w}_g$ is removed from $\mathbf{W}$ and $\mathbf{s}'_g$ from $\mathcal{C}_k$.



$\psi_{\vartheta}(\cdot)$
$\alpha_{\text{new}}(\cdot)$
$\mathbf{w}_g$
$\alpha_g(\cdot)$
$\mathbf{W}\psi_{\vartheta}(\cdot)$
$\alpha_{\theta}(\cdot)$
$\frac{\partial \mathcal{L}_{\alpha}}{\partial \psi}\big|_{\mathcal{B}}$
$\mathbf{w} \in \mathcal{G}$
$\frac{\partial \mathcal{L}_{\alpha}}{\partial \alpha}\big|_{\mathcal{B}}$
$\mathbf{w} \notin \mathcal{G}$

i) Feed Forward Architecture. ii) Backprop sparse $\frac{\partial \mathcal{L}_{\alpha}}{\partial \alpha}\big|_{\mathcal{B}}$. iii) Remove all $||\mathbf{w}_g||_2 = 0$.

**Compression Set** It is possible to use the Lasso algorithm in Lever et al. (2016) to maintain $\mathcal{C}_k$ as more transition data is experienced, however this set is not adaptive and will always grow. We offer an alternative mechanism that adaptively removes successor states and can either be used on its own or in conjunction with the CCME's Lasso algorithm. We refer to figure 1 ii) and iii) in the following discussion.

We modify the online truncated gradient (Langford et al., 2009) for lasso (Tibshirani, 1996) into a *group lasso* (Yuan and Lin, 2006) variant. For each minibatch $\mathcal{B}$, then due to the last layer's sparse activation function, only a subset $\mathcal{G}$ of weight groups $\mathbf{w}_g$ will be involved in the embedding's activation. Therefore a sparse gradient signal provides only updates to last layer weights in $\mathcal{G}$. Our novel approach is to apply truncated group lasso shrinkage to weight groups that are *not* in $\mathcal{G}$. Infrequently used $\mathbf{w}_g$ are therefore driven to zero over $M$ minibatches. This is equivalent to applying the truncated L21 shrinkage to the *inactive* weight groups in the last layer. The subgradient term in equation (3) is therfore replaced by the following function acting on the last layer weights,

$$T^{\text{active}}(w_{gj}, \mathbf{w}_g, \eta_{gj}) = \begin{cases} \max(0, w_{gj} - \eta_{gj}\lambda_{21}|w_{gj}|/||\mathbf{w}_g||_2) & w_{gj} > 0, \mathbf{w}_g \notin \mathcal{G}, \\ \min(0, w_{gj} + \eta_{gj}\lambda_{21}|w_{gj}|/||\mathbf{w}_g||_2) & w_{gj} < 0, \mathbf{w}_g \notin \mathcal{G}, \\ w_{gj} - \eta\lambda_2 w_{gj} & \text{otherwise}, \end{cases} \quad (4)$$

where the weight element is defined $w_{gj} := \mathbf{W}_{gj}$. Critically the last term is an L2-norm shrinkage that stops the last layer weights from growing out of control (which is exacerbated due to the adding and removing of weight groups). At the end of training with $M$ minibatches at each $k^{\text{th}}$ policy iteration, then for all groups $||\mathbf{w}_g||_2$ that have remained at zero for the last $I_{\text{gaze}}{=}\frac{3}{4}M$ minibatches are removed from $\mathbf{W}$ and their corresponding states $\mathbf{s}'$ are removed from $\mathcal{C}_k$. In practice it was found that the regulariser be applied to the middle third minibatches at each policy iteration.

**Learning the Reward Function** ACCME also learns the immediate reward function using loss $\mathcal{L}_r|_{\mathcal{B}}{:=}\frac{1}{2m}\sum_{i=1}^{m}||r_{\boldsymbol{\beta}}(\mathbf{s}_i,\mathbf{a}_i)-r_i||_2^2$, where reward estimate $r_{\boldsymbol{\beta}}(\mathbf{s},\mathbf{a}){=}\boldsymbol{\beta}^\top\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\mathbf{s}_i,\mathbf{a}_i)$ uses the learnt representation over states and actions that already exists in the embedding network. Reward weights $\boldsymbol{\beta}$ are updated using

$$\frac{\partial\mathcal{L}_r}{\partial\boldsymbol{\beta}}\bigg|_{\mathcal{B}} {=} \frac{1}{m}\sum_{i=1}^{m}\big(r_{\boldsymbol{\beta}}(\mathbf{s}_i,\mathbf{a}_i){-}r_i\big)\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\mathbf{s}_i,\mathbf{a}_i), \tag{5}$$

such that $\boldsymbol{\beta}{\leftarrow}\boldsymbol{\beta}{-}\eta\frac{\partial\mathcal{L}_r}{\partial\beta}\big|_{\mathcal{B}}$. The gradient signal $\frac{\partial\mathcal{L}_r}{\partial\boldsymbol{\psi}}\big|_{\mathcal{B}}$ is collected with the embedding's $\frac{\partial\mathcal{L}_{\boldsymbol{\alpha}}}{\partial\boldsymbol{\psi}}\big|_{\mathcal{B}}$ and backpropagated to the rest of the network during training.

## 3. Experiments

Algorithm 1 refers to the full ACCME implementation. Table 1 shows algorithm component computational complexities comparing our architecture to the kernel-based CME and CCME algorithms. The NK-CCME variant is the prequel to ACCME which has the same neural architecture but like the CCME, uses Lasso to only add states to $\mathcal{C}$ and assumes knowledge of the reward function. Note that $c$ is the fixed incomplete Cholesky decomposition size, $N^*_{\text{L1-Proj}}$ and $N^*_{\text{Lasso}}$ are the lazily constrained sparse embedding sizes (which are not* strictly constant over any set of state-actions) and CCME Lasso components have complexity $l{=}f(|\mathcal{C}_k|,c)$ e.g $l{=}c|\mathcal{C}_k|^2$ (Friedman et al., 2010) or $l{=}c|\mathcal{C}_k|$ (Efron et al., 2004) (see Lever et al. (2016) for more details).

Table 1: Computational complexity of algorithm components.

| Algorithm: | | Discrete $\mathcal{S}$ | CME | CCME | **NK-CCME** | | **ACCME** |
|---|---|---|---|---|---|---|---|
| Constraint: | | | L1ProjSparse | LassoSparse | Softmax | TopNmax | TopNmax |
| i) | $\mathcal{C}$ maintenance | – | – | $l$ | $l$ | $l$ | $\lvert\mathcal{C}_k\rvert$ |
| ii) | $\boldsymbol{\alpha}$ model update (full) | – | $n_k^3$ | $\lvert\mathcal{Q}_k\rvert^3{+}\lvert\mathcal{C}_k\rvert^3$ | $\lvert\mathcal{C}_k\rvert^2$ | $N\lvert\mathcal{C}_k\rvert$ | $N\lvert\mathcal{C}_k\rvert$ |
| iii) | $\lvert\lvert\boldsymbol{\alpha}(\mathbf{s},\mathbf{a})\rvert\rvert_1{\le}1$ | – | $n_k$ | $l$ | $\lvert\mathcal{C}_k\rvert$ | $N\lvert\mathcal{C}_k\rvert$ | $N\lvert\mathcal{C}_k\rvert$ |
| iv) | $\hat{\mathbf{v}}^{\pi_k}{=}(\hat{T}_\mu^{\pi_k})^{J_{\text{eval}}}\mathbf{v}^0$ | $J_{\text{eval}}\lvert\mathcal{S}\rvert^2$ | $n_k^2{+}J_{\text{eval}}N^*_{\text{L1-Proj}}n_k$ | $\lvert\mathcal{C}_k\rvert^2{+}J_{\text{eval}}N^*_{\text{Lasso}}\lvert\mathcal{C}_k\rvert$ | $(1{+}J_{\text{eval}})\lvert\mathcal{C}_k\rvert^2$ | $\lvert\mathcal{C}_k\rvert^2{+}J_{\text{eval}}N\lvert\mathcal{C}_k\rvert$ | $\lvert\mathcal{C}_k\rvert^2{+}J_{\text{eval}}N\lvert\mathcal{C}_k\rvert$ |
| v) | $\pi_k(s)$ | $\lvert\mathcal{A}\rvert\lvert\mathcal{S}\rvert$ | $\lvert\mathcal{A}\rvert n_k$ | $\lvert\mathcal{A}\rvert l$ | $\lvert\mathcal{A}\rvert\lvert\mathcal{C}_k\rvert$ | $\lvert\mathcal{A}\rvert N\lvert\mathcal{C}_k\rvert$ | $\lvert\mathcal{A}\rvert N\lvert\mathcal{C}_k\rvert$ |

Figures 2 to 4 show variables (from left to right) vs. $k^{\text{th}}$ policy iteration, where the variables are i) empirical average cumulative discounted reward, ii) the average model update times and iii) the average planning time. Also compared are CCME algorithms for two types of lazy constraints L1ProjSparse and LassoSparse. The prequel NKCCME is also included where full softmax and sparse TopN softmax are the last layer activation functions. The final ACCME results include two variants where the first assumes the known reward function and ACCME-R learns the unknown reward function. Figure 5 shows how the compression set size varies over the life of each algorithm. Three model-free (value-based) DQN (Mnih et al., 2015)-# algorithms are also shown where '#' refers to the number of TD updates the Q-network experiences at each transition.

During each experiment each variable was sampled 20 times at each policy iteration, then this was averaged over 20 experiments to provide error bars. From top to bottom the results of three control tasks are included where CartPole and Quadrocopter Navigation are taken from Lever et al. (2016), whose trajectory lengths are $H$=100, thus $n_{\text{new}}$=200 samples are added to experience replay every $k^{\text{th}}$ iteration. An additional quadrocopter holding pattern experiment is included whose objective is to get the quadrocopter to assume an orbital holding pattern $10m$ in diameter whose $H$=200. This task was created to test the algorithm's handling of data abundance. For CartPole $|\mathcal{A}| = 3$ (where dim($\mathcal{A}$)=1) but for the quadrocopter tasks $|\mathcal{A}|$=81 (where dim($\mathcal{A}$)=3 and dim($\mathcal{A}$)=2 for the navigation and holding pattern respectively). The state space for CartPole has dim($\mathcal{S}$)=2 and for the quadrocopters dim($\mathcal{S}$)=13.

## 4. Discussion

We provide empirical evidence in figures 2 to 5 to claim that ACCME although somewhat sacrifices the CCME's sample efficiency (yet it remains competitive to model-free DQN), it improves upon several CCME practical shortcomings: i) ACCME model updates do not suffer from increasing computational costs that full CCME updates incur as $k$ increases (middle column). CCME's suffer this mainly due to a combination of the the cross validation scheme used in matching pursuit to sparsify $\psi(\cdot)$ and the final batch regression on the weights. ii) CCME's have to solve a Lasso minimisation problem every time the lazy contraction constraint is enforced and this is costly for large action spaces during planning (third column). ACCME's fast-evaluated activation function mitigates this problem. iii) Using the CCME's Lasso algorithm to maintain $\mathcal{C}$ is demonstrated to work in the NKCCME variant, however this leads to compression sets increasing in size (figure 5). ACCME's truncated gradient algorithm is demonstrated to show $\mathcal{C}$ maintaining a consistent size.

## 5. Related Work

Biologically inspired mechanisms to prune and compress artificial networks to improve computational/memory costs and improve generalisation (LeCun et al., 1990) are widespread. The use of group lasso regularisers to enforce network structural sparsity has also been extensively studied (Kong et al., 2014; Yoon and Hwang, 2017; Wen et al., 2016; Lebedev and Lempitsky, 2016). However a naive application of group lasso subgradients to shrink group weights does not produce sparsity amongst weight groups. As discussed by Langford et al. (2009), no $||\mathbf{w}_g||_2$ can ever reach exactly zero and instead an arbitrary fine-tuned threshold parameter is required to define when a weight group can be pruned. Exact sparsity is a critical requirement for ACCME in order to effectively remove states from the compression set without disrupting the function approximator. Our group lasso version of the *truncated* gradient algorithm *does* produce sparsity by zeroing weights as they pass zero. This technique has proven effective across several RL control tasks without additional fine tuning. To the best of our knowledge, shrinking *non-active* weight groups (induced by a sparse softmax activation function during backprop) for each minibatch is novel.

**Algorithm 1** ACCME-R Policy Iteration

```
1:  Input: Unknown $\mathcal{M}{=}\{\mathcal{S},\mathcal{A},P,P_R,\gamma\}$, start-state distribution $P_1$, kernel $L:\mathcal{S}{\times}\mathcal{S}\to\mathbb{R}$,
    trajectory length $H$, shared ANN $\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\cdot,\cdot)$, sparse top-N count $N$.
2:  Parameters: Sample count $n_{\text{new}}{=}2H$, minibatch size $m{=}20$, minibatch count $M{=}3000$,
    $\lambda_2{=}1{\times}10^{-4}$, $\lambda_{21}{=}1{\times}10^{-1}$, $\eta_{21}{=}1$, $J_{\text{imp}}{=}10$, $J_{\text{eval}}{=}4000$, compression set $\mathcal{C}$.
3:  Output: $\pi_k(\mathbf{s})\approx\pi^*(\mathbf{s}) \quad \forall\mathbf{s}\in\mathcal{S}$.
4:  Initialise: $\boldsymbol{\vartheta}$, $\boldsymbol{\beta}\sim\mathcal{N}(0,\sigma^2{=}0.01)$, $\mathbf{W}^{(0)}{=}\emptyset$, $\mathcal{C}_0{=}\emptyset$, $\boldsymbol{\alpha}(\cdot,\cdot){:=}\boldsymbol{\sigma}(\mathbf{W}^{(0)}\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\cdot,\cdot))$,
    $r(\cdot,\cdot){:=}\boldsymbol{\beta}^\top\boldsymbol{\psi}_{\boldsymbol{\vartheta}}(\cdot,\cdot)$, $q_0(\cdot,\cdot){=}r(\cdot,\cdot)$, $\mathcal{D}_0{=}\emptyset$, $n_0{=}0$, $\pi_1(\cdot){=}\text{greedy}_{\mathbf{a}\in\mathcal{A}}(q_0(\cdot,\mathbf{a}))$.
5:  for $k=1,2,...$ do                                   ▷ $k^{\text{th}}$ policy iteration master index
6:      Data acquisition: From behaviour policy $\rho^{\pi_k}$ collect $\mathcal{D}_{\text{new}}{=}\{\mathbf{s}_i,\mathbf{a}_i,r_i,\mathbf{s}'_i\}_{i=n_{k\text{-}1}+1}^{n_{k\text{-}1}+n_{\text{new}}}$
7:      $\mathcal{D}_k{\leftarrow}\mathcal{D}_{k-1}{\cup}\,\mathcal{D}_{\text{new}}$, $\mathcal{S}'_{\text{new}}{=}\{\mathbf{s}'_i\}_{i=n_{k\text{-}1}+1}^{n_{k\text{-}1}+n_{\text{new}}}$, $n_k{\leftarrow}n_{k\text{-}1}{+}n_{\text{new}}$.        ▷ aggregate data
8:      Augment $\mathcal{C}$: $\mathcal{C}_k{\leftarrow}\mathcal{C}_{k-1}{\cup}\,\mathcal{S}'_{\text{new}}$, $\mathbf{W}^{\text{new}}{\leftarrow}\mathbb{R}^{n_{\text{new}}\times\dim(\boldsymbol{\psi})}{\sim}\mathcal{N}(0,\sigma^2)$,
        $\mathbf{W}^{(k)}{\leftarrow}[\mathbf{W}^{(k-1)};\mathbf{W}^{\text{new}}]$.
9:      Train model: $\boldsymbol{\alpha}(\cdot,\cdot)$, $\mathcal{C}_k$, $r(\cdot,\cdot){\leftarrow}\textsc{Train}\big(\boldsymbol{\alpha}(\cdot,\cdot),\,\mathcal{C}_k,\,r(\cdot,\cdot),\,\mathcal{D}_k\big)$.
10:     Planning: over $\mathbf{v}{:=}[v(\mathbf{s}'_1),...,v(\mathbf{s}'_{|\mathcal{C}_k|})]^\top$, $(\hat{T}^{\pi_k}_{\mu}v)(\cdot){:=}r(\mathbf{s}',\pi_k(\cdot)){+}\gamma\boldsymbol{\alpha}^\top(\cdot,\pi_k(\cdot))\mathbf{v}$
11:     for $i=1$ to $J_{\text{imp}}$ do                                   ▷ planning index
12:         $\mathbf{v}{\leftarrow}\mathbf{0}$
13:         for $j=1$ to $J_{\text{eval}}$ do                          ▷ exact policy evaluation
14:             $\mathbf{v}{\leftarrow}\hat{T}^{\pi_k}_{\mu}\mathbf{v}$
15:         end for
16:         $\pi_k(\cdot){\leftarrow}\text{greedy}_{\mathbf{a}\in\mathcal{A}}[r(\cdot,\mathbf{a}){+}\gamma\boldsymbol{\alpha}^\top(\cdot,\mathbf{a})\mathbf{v}]$          ▷ policy improvement
17:     end for
18:     $\pi_{k+1}(\cdot){\leftarrow}\pi_k(\cdot)$
19: end for
20: return $\pi_{k+1}(\cdot)$
```

```
1:  function $\textsc{Train}(\boldsymbol{\alpha}(\cdot,\cdot),\,\mathcal{C},r(\cdot,\cdot),\,\mathcal{D})$
2:  for $i=1$ to $M$ do
3:      Draw minibatch $\mathcal{B}{\sim}\mathcal{D}$
4:      $\boldsymbol{\beta}{\leftarrow}\boldsymbol{\beta}{-}\eta\frac{\partial\mathcal{L}_r}{\partial\boldsymbol{\beta}}\big|_{\mathcal{B}}$                                  ▷ update all reward weights.
5:      for each $w_{gj}{\in}\mathbf{W}$ do                ▷ update all last layer embedding weights
6:          $w_{gj}{\leftarrow}\mathrm{T}^{active}\big(w_{gj},\mathbf{w}_g,\eta_{gj}\big)$                     ▷ diminish inactive weights
7:          $w_{gj}{\leftarrow}w_{gj}{-}\eta_{gj}\frac{\partial\mathcal{L}_{\boldsymbol{\alpha}}}{\partial w_{gj}}\big|_{\mathcal{B}}$              ▷ sparse non-parametric gradient signal
8:      end for
9:      $\vartheta{\leftarrow}\vartheta{-}\eta\big[\big(\frac{\partial\mathcal{L}_{\boldsymbol{\alpha}}}{\partial\boldsymbol{\psi}}{+}\frac{\partial\mathcal{L}_r}{\partial\boldsymbol{\psi}}\big)\frac{\partial\boldsymbol{\psi}}{\partial\vartheta}\big]\big|_{\mathcal{B}}, \quad \forall\vartheta{\in}\boldsymbol{\vartheta}$            ▷ update the rest of the network
10: end for
11: for each $\mathbf{w}_g{\in}\mathbf{W}$ do                         ▷ remove $\mathbf{w}_g$ that have been zeroed
12:     if $||\mathbf{w}_g||_2$ remained 0 over the last $\frac{3}{4}M$ minibatches then
13:         $\mathbf{W}{\leftarrow}\mathbf{W}\backslash\mathbf{w}_g$, $\mathcal{C}{\leftarrow}\mathcal{C}\backslash\mathbf{s}'_g$        ▷ remove weight group from $\mathbf{W}$ and state from $\mathcal{C}$
14:     end if
15: end for
16: return $\boldsymbol{\alpha}(\cdot,\cdot)$, $\mathcal{C},r(\cdot,\cdot)$.
```

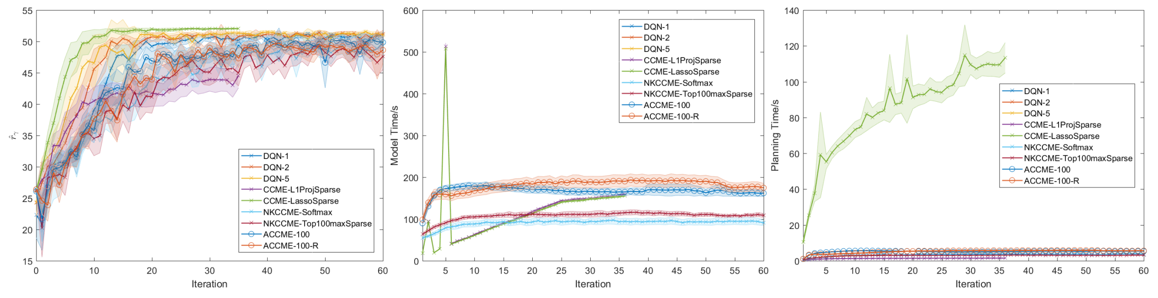


Figure 2: Cart Pole

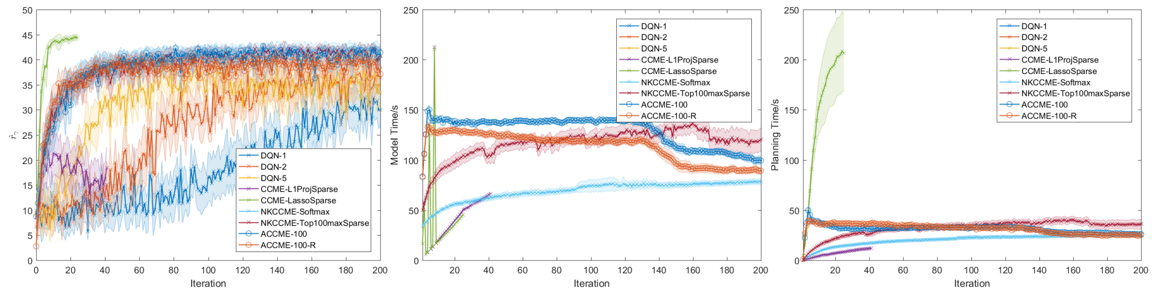


Figure 3: Quadrocopter Navigation

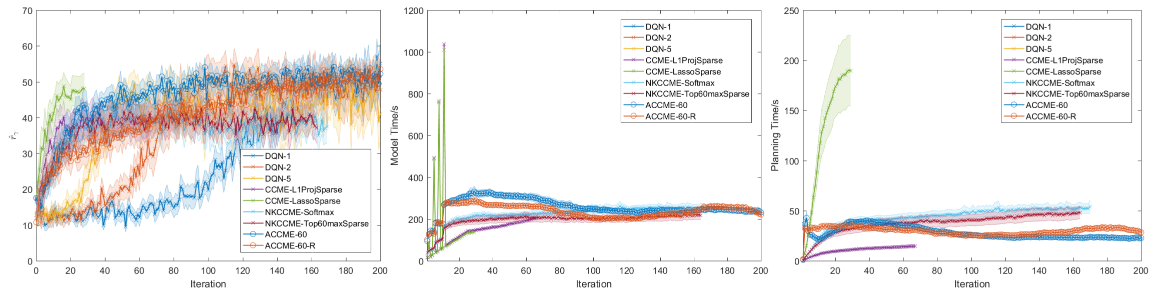


Figure 4: Quadrocopter Holding Pattern

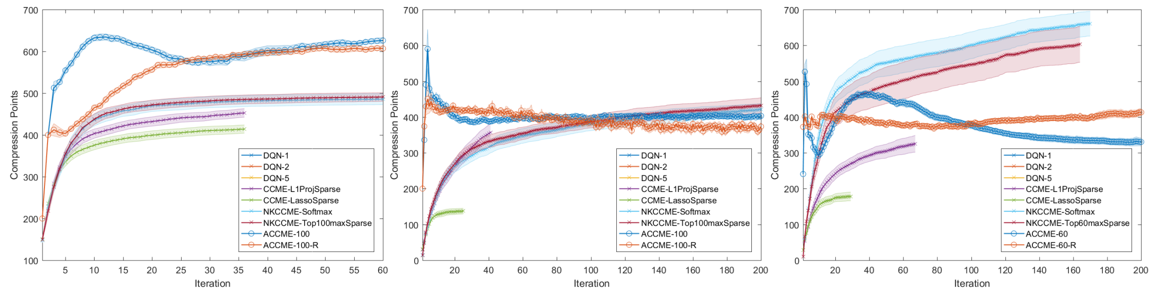


Figure 5: Compression points (CartPole, Quadrocopter Navigation and Quadrocopter Holding Pattern)

## References

Nachman Aronszajn. Theory of Reproducing Kernels. *Transactions of the American Mathematical Society*, 68(3):337–404, 1950. URL `http://www.jstor.org/stable/1990404`.

Dimitri Bertsekas. Approximate policy iteration: A survey and some new methods. *Journal of Control Theory and Applications*, 9(3):310–335, 2011.

Bradley Efron, Trevor Hastie, Iain Johnstone, Robert Tibshirani, Hemant Ishwaran, Keith Knight, Jean Michel Loubes, Pascal Massart, David Madigan, Greg Ridgeway, Saharon Rosset, J. I. Zhu, Robert A. Stine, Berwin A. Turlach, Sanford Weisberg, Iain Johnstone, and Robert Tibshirani. Least angle regression. *Annals of Statistics*, 32(2):407–499, 2004.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal Of Statistical Software*, 33(1), 2010. URL `https://www.jstatsoft.org/index`.

Geoffrey Gordon. Stable Function Approximation in Dynamic Programming. In *Proceedings of the 12th International Conference on Machine Learning (ICML)*, 1995.

Steffen Grünewälder, Guy Lever, Luca Baldassarre, Arthur Gretton, and Massimiliano Pontil. Modelling transition dynamics in MDPs with RKHS embeddings. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 535–542, Edinburgh, Scotland, UK, 2012a.

Steffen Grünewälder, Guy Lever, Luca Baldassarre, Sam Patterson, Arthur Gretton, and Massimiliano Pontil. Conditional Mean Embeddings as Regressors. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pages 1823–1830, Edinburgh, Scotland, UK, 2012b.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous Deep Q-Learning with Model-based Acceleration. *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

David Ha and Jürgen Schmidhuber. World Models. *CoRR*, 2018. doi: 10.5281/zenodo.1207631. URL `http://arxiv.org/abs/1803.10122`.

Ronald A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, MA, USA, 1960.

Deguang Kong, Ryohei Fujimaki, Ji Liu, Feiping Nie, and Chris Ding. Exclusive Feature Learning on Arbitrary Structures via L12-norm. In Z Ghahramani, M Welling, C Cortes, N D Lawrence, and K Q Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1655–1663, 2014. URL `https://papers.nips.cc/paper/5631-exclusive-feature-learning-on-arbitrary-structures-via-ell{_}12-norm`.

John Langford, Lihong Li, and Tong Zhang. Sparse Online Learning via Truncated Gradient. *Journal of Machine Learning Research (JMLR)*, 10(1):777–801, 2009. URL `http://www.jmlr.org/papers/volume10/langford09a/langford09a.pdf`.

Vadim Lebedev and Victor Lempitsky. Fast ConvNets Using Group-wise Brain Damage. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. URL `http://arxiv.org/abs/1506.02515`.

Yann LeCun, John S Denker, and Sara A Solla. Optimal Brain Damage. In D S Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 598–605. Morgan-Kaufmann, 1990. URL `http://papers.nips.cc/paper/250-optimal-brain-damage.pdf`.

Guy Lever, John Shawe-Taylor, Ronnie Stafford, and Csaba Szepesvári. Compressed Conditional Mean Embeddings for Model-Based Reinforcement Learning. In *Association for the Advancement of Artificial Intelligence (AAAI)*, pages 1779–1787, Phoenix, Arizona, 2016. URL `http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12436`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei a Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. URL `http://dx.doi.org/10.1038/nature14236`.

Rémi Munos. Error Bounds for Approximate Policy Iteration. *Proceedings of the 20th International Conference on Machine Learning (ICML)*, 2:560–567, 2003.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 1 edition, jun 2004. ISBN 0521813972.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *International Conference on Learning Representations (ICLR)*, 2017. URL `http://arxiv.org/abs/1701.06538`.

Satinder Singh and Richard Yee. An Upper Bound on the Loss from Approximate Optimal-Value Functions. *Machine Learning*, 16(3):227–233, sep 1994. URL `http://dx.doi.org/10.1023/A:1022693225949`.

Le Song, Jonathan Huang, Alex Smola, and Kenji Fukumizu. Hilbert Space Embeddings of Conditional Distributions with Applications to Dynamical Systems. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 961–968, 2009. URL `http://portal.acm.org/citation.cfm?doid=1553374.1553497`.

Richard Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2:160–163, 1991.

Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. A Bradford Book - MIT Press, Cambridge, MA, USA, 1st edition, mar 1998. ISBN 0262193981.

Csaba Szepesvári. *Algorithms for Reinforcement Learning*, volume 4 of *Synthesis Lectures on Artificial Intelligence and Machine Learning*. Morgan & Claypool, San Rafael, CA, USA, jan 2010. URL `http://dx.doi.org/10.2200/s00268ed1v01y201005aim009`.

Robert Tibshirani. Regression selection and shrinkage via the lasso. *Journal of the Royal Statistical Society B*, 58(1):267–288, 1996. URL `http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.7574`.

Theophane Weber, Sébastien Racanière, David P Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adrià Puigdomènech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, Razvan Pascanu, Peter Battaglia, David Silver, and Daan Wierstra. Imagination-Augmented Agents for Deep Reinforcement Learning. *CoRR*, abs/1707.0, 2017. URL `http://arxiv.org/abs/1707.06203`.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning Structured Sparsity in Deep Neural Networks. In *30th Conference on Neural Information Processing Systems (NIPS 2016)*, 2016. URL `http://papers.nips.cc/paper/6504-learning-structured-sparsity-in-deep-neural-networks`.

Hengshuai Yao, Csaba Szepesvári, Bernardo Avila Pires, and Xinhua Zhang. Pseudo-MDPs and Factored Linear Action Models. In *Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, pages 1–9, 2014.

Jaehong Yoon and Sung Ju Hwang. Combined Group and Exclusive Sparsity for Deep Neural Networks. In *Proceedings of the 34 th International Conference on Machine Learning (ICML)*, Sydney, Australia, 2017.

Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 68(1):49–67, 2006. URL `http://pages.stat.wisc.edu/{~}myuan/papers/glasso.final.pdf`.