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 \( (T_{\pi}v)(s) := r(s, \pi(s)) + \gamma \mathbb{E}_{S' \sim p(\cdot|s, \pi(s))}[v(s')] \) (Sutton and Barto, 1998) where value functions \( v: \mathcal{S} \to \mathbb{R} \) are defined over all states \( s \in \mathcal{S} \). We assume interaction with a Markov decision process (MDP) \( \mathcal{M} := \{ \mathcal{S}, \mathcal{A}, p, r, \gamma \} \) possessing an unknown average reward function \( r(s, a) := \mathbb{E}_{R \sim P_R(\cdot|s, \pi(s))}[R] \in [0, R_{\text{max}}] \), discrete actions set \( \mathcal{A} \), continuous states \( \mathcal{S} \) and unknown transition dynamics \( p(s'|s, a) \). 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

©2018 Ronnie Stafford and John Shawe-Taylor.

License: CC-BY 4.0, see https://creativecommons.org/licenses/by/4.0/.
the transition dynamics $E(s, a) := \mathbb{E}_{S' \sim p(\cdot | s, a)}[v(s')]$ are known, then through contraction arguments (Szepesvári, 2010) the Bellman operator $T^\pi$ is non-expansive in $|| \cdot ||_\infty$. Then $v_p = (T^\pi)^p v_0$ where $v_p = [v_p(s_1), ..., v_p(s_{|S|})] \top$ and $v^\pi = \lim_{p \to \infty}(v_p)$ is the fixed point of $T^\pi$, which is PI’s policy evaluation step. The policy improvement step performs a greedy update $\pi_v(s) := \arg\max_{a \in A} \langle r(s, a) + \gamma \mathbb{E}_{S' \sim p(\cdot | s, \pi(s))}[v_p(s')] \rangle$. By alternating between policy evaluation and policy improvement steps, the policy converges to the optimal policy.

Each PI iteration scales $\sim \mathcal{O}(|S|^2)$ and is therefore intractable for continuous state spaces. Usually a linear function approximation $v(s) \approx \langle v, \phi(s) \rangle_F$ is adopted in some function space $F \subset \mathbb{R}^S$ with state features $\phi: S \to F$. Traditional approximate DP chooses a parametric $F$ 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 $F$ whose $\phi$ 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 $S$ is continuous. By choosing $F$ as a reproducing kernel Hilbert space (RKHS) (Shawe-Taylor and Cristianini, 2004) $H_L$ with kernel $L: \mathcal{S} \times \mathcal{S} \to \mathbb{R}$, a CME (Grünewälder et al., 2012a) approximates the expectation in $T^\pi$ by

$$
\mathbb{E}_{S' \sim p(\cdot | s, a)}[v(S')] \approx \langle v, \hat{\mu}_{\text{CME}}(s, a) \rangle_{H_L} = \langle v, \sum_{j=1}^{n_k} \alpha_j(s, a) \phi(s'_j) \rangle_{H_L},
$$

$$
= \sum_{j=1}^{n_k} \alpha_j(s, a)v(s'_j) =: \hat{E}_{\mu}(s, a), \quad s' \in S_k, n_k = |S_k|,
$$

where $S_k$ is the discrete set of successor states in the transition data $\mathcal{D}_k := \{ (s, a, r, s'_i) \}_{i=1}^{n_k}$ collected up to the $k^{th}$ policy iteration from interacting with $M$. 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), $L(s'_i, s'_j) := \langle \phi(s'_i), \phi(s'_j) \rangle_{H_L}$ is the state kernel evaluation at $(s'_i, s'_j)$ and $\alpha := [\alpha_1, ..., \alpha_{n_k}] \top$ 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 $\hat{T}^\mu$, forms an approximate Bellman operator $\hat{T}^\mu: \mathcal{V} \to \mathcal{V}$ (where $\mathcal{V} := \{ v(s'_1), ..., v(s'_{n_k}) \}$). Value functions can then be solved exactly by policy evaluation over $S_k$ if there is a contraction mapping enforced by the constraint $||\alpha(s', a)||_1 \leq 1 \forall (s', a) \in S_k \times \mathcal{A}$. 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 $\alpha(\cdot)$. 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 $S_k$.

2. ACCME

Online Dynamics Model  Notation from Lever et al. (2016) is adopted to denote the compression set of successor states $C_k \subseteq S_k$ that the compressed version of the embedding (1) is defined over. The explanation of ACCME’s method of maintaining $C_k$ 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 $\alpha_\theta : S \times A \rightarrow \mathbb{R}^{|C_k|}$,

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{2nk} \sum_{i=1}^{nk} || \Phi_{C_k} \alpha_\theta(s_i, a_i) - \phi(s'_i)||_F^2 + \Omega(\theta),$$

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

$$\frac{\partial L_\alpha}{\partial \theta} \bigg|_B = \frac{1}{m} \sum_{i=1}^{m} (\alpha_\theta^\top(s_i, a_i)L_{C_kC_k} - L_{s'_iC_k}) \frac{\partial \alpha_\theta}{\partial \theta} \bigg|_{(s_i, a_i)} + \partial_\theta \Omega(\theta),$$

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

Architecture and Contraction Constraint  A CME naturally decomposes the kernel weights into $\alpha_{\text{CCME}}(\cdot) = W \psi(\cdot)$ where $W$ is a weight matrix and $\psi(\cdot)$ is non-parametric representation over $S \times A$ (which for a CCME at iteration $k$ is based on a sparsified set of state-actions $Q_k$). ACCME parametrises this decomposition as $\alpha_\theta(\cdot) = \sigma(W \psi_\theta(\cdot))$ where $W \in \mathbb{R}^{|C_k| \times \dim(\psi)}$ is the last layer weights and $\psi_\theta(\cdot)$ is the rest of the network up to the penultimate layer with weights $\theta$ (see figure 1 i)). We use the notation $\theta := \{W, \psi\}$ as the set of all weights. Setting $\sigma(\cdot) = \text{TopNMax}(\cdot) := \text{Softmax}(\text{KeepTopN}(\cdot))$ (Shazeer et al., 2017) as the final layer’s activation function, then the pseudo-MDP contraction constraint is satisfied. The network output $\alpha(s, a) \in \mathbb{R}^N$ is fixed $N$-sparse where $N$ is chosen a-priori and sorting the raw activations costs $\sim \mathcal{O}(N|C_k|)$. 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 $w_g$ are indexed by $g$ and accumulated in $W := [w_1, ..., w_{|C_k|}]^T$. When a new state is added $C_k \leftarrow C_k \cup s'_{\text{new}}$, then a new weight group $w_{\text{new}}$ is initialised and added to $W$. ii) Backprop a sparse gradient signal; active weight groups $w_g \in G$ are updated in the standard way, inactive $w_g \not\in G$ weight groups are shrunk by a truncated group lasso update. iii) When after training with $M$ minibatches at the $k^{th}$ policy iteration, then for all $g$ where $||w_g||_2$ has remained 0 for the last $\frac{3}{4}M$ minibatches, $w_g$ is removed from $W$ and $s'_{g}$ from $C_k$.

Compression Set  It is possible to use the Lasso algorithm in Lever et al. (2016) to maintain $C_k$ 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 $B$, then due to the last layer’s sparse activation function, only a subset $G$ of weight groups $w_g$ will be involved in the embedding’s activation. Therefore a sparse gradient signal provides only updates to last layer weights in $G$. Our novel approach is to apply truncated group lasso shrinkage to weight groups that are not in $G$. Infrequently used $w_g$ 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 therefore replaced by the following function acting on the last layer weights,

$$T_{\text{active}}(w_{gj}, w_g, \eta_{gj}) = \begin{cases} 
\max(0, w_{gj} - \eta_{gj} \lambda_2 |w_{gj}|/||w_g||_2) & w_{gj} > 0, w_g \not\in \mathcal{G}, \\
\min(0, w_{gj} + \eta_{gj} \lambda_2 |w_{gj}|/||w_g||_2) & w_{gj} < 0, w_g \not\in \mathcal{G}, \\
w_{gj} - \eta \lambda_2 w_{gj} & \text{otherwise},
\end{cases}$$

where $\eta_{gj}$ is the scalar weight update of $w_{gj}$ and $\lambda_2$ is the L21 parameter. This function is equivalent to applying the truncated L21 shrinkage to the inactive $w_g$ in the last layer.
where the weight element is defined $w_{gj} := W_{gj}$. 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 $k^{th}$ policy iteration, then for all groups $\|w_g\|_2$ that have remained at zero for the last $I_{\text{size}} = \frac{3}{4} M$ minibatches are removed from $W$ and their corresponding states $s'$ are removed from $C_k$. 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 $L_r|B| = \frac{1}{2m} \sum_{i=1}^{m} \|r_\beta(s_i, a_i) - r_i\|_2^2$, where reward estimate $r_\beta(s_i, a_i) = \beta^T \psi_\theta(s_i, a_i)$ uses the learnt representation over states and actions that already exists in the embedding network. Reward weights $\beta$ are updated using

$$\frac{\partial L_r}{\partial \beta} \bigg|_B = \frac{1}{m} \sum_{i=1}^{m} (r_\beta(s_i, a_i) - r_i) \psi_\theta(s_i, a_i),$$

such that $\beta \leftarrow \beta - \eta \frac{\partial L_r}{\partial \beta} \bigg|_B$. The gradient signal $\frac{\partial E_r}{\partial \psi} \bigg|_B$ is collected with the embedding’s $\frac{\partial E_r}{\partial \psi} \bigg|_B$ 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 $C$ and assumes knowledge of the reward function. Note that $c$ is the fixed incomplete Cholesky decomposition size, $N_{\text{L1}}}^*$ 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 $l = f(|C_k|, c)$ e.g $l = c|C_k|$. (Friedman et al., 2010) or $l = c|C_k|$ (Efron et al., 2004) (see Lever et al. (2016) for more details).

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Discrete $S$</th>
<th>CME</th>
<th>CCME</th>
<th>NK-CCME</th>
<th>TopNMAX</th>
<th>ACCME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constraint:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i)</td>
<td>$C$ maintenance</td>
<td>$-$</td>
<td>$-$</td>
<td>$l$</td>
<td>$l$</td>
<td>$</td>
</tr>
<tr>
<td>ii)</td>
<td>$\alpha$ model update (full)</td>
<td>$n^2_k$</td>
<td>$</td>
<td>Q_1</td>
<td>^2 +</td>
<td>C</td>
</tr>
<tr>
<td>iii)</td>
<td>$</td>
<td>\alpha(s, a)</td>
<td>\leq 1$</td>
<td>$l$</td>
<td>$C_i$</td>
<td>$</td>
</tr>
<tr>
<td>iv)</td>
<td>$\psi_n = (\psi_n)^{l_{\text{L1}}=\psi^d}$</td>
<td>$n^2_i + l_{\text{L1Proj}}^n</td>
<td>A</td>
<td>$</td>
<td>$C_i^2 + l_{\text{L1Proj}}^n</td>
<td>C</td>
</tr>
<tr>
<td>v)</td>
<td>$\pi(s)$</td>
<td>$</td>
<td>A</td>
<td></td>
<td>S</td>
<td>$</td>
</tr>
</tbody>
</table>

Figures 2 to 4 show variables (from left to right) vs. $k^{th}$ 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 $|A|=3$ (where $\dim(A)=1$) but for the quadrocopter tasks $|A|=81$ (where $\dim(A)=3$ and $\dim(A)=2$ for the navigation and holding pattern respectively). The state space for CartPole has $\dim(S)=2$ and for the quadrocopters $\dim(S)=13$.

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 $C$ 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 $C$ 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 $\|w_g\|_2$ 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 $\mathcal{M} = \{S,A,P,R,\gamma\}$, start-state distribution $P$, kernel $L : S \times S \to \mathbb{R}$, trajectory length $H$, shared ANN $\psi_{\vartheta}(\cdot,\cdot)$, sparse top- $N$ count $N$.

2. **Parameters:** Sample count $n_{new}=2H$, minibatch size $m=20$, minibatch count $M=3000$,$\lambda_2=1 \times 10^{-4}$, $\lambda_21=1 \times 10^{-1}$, $J_{imp}=10$, $J_{eval}=4000$, compression set $C$.

3. **Output:** $\pi_k(s) \approx \pi^*(s) \quad \forall s \in S$.

4. **Initialise:** $\vartheta$, $\beta \sim \mathcal{N}(0, \sigma^2=0.01)$, $W(0)=\emptyset$, $C_0=\emptyset$, $\alpha(\cdot,\cdot):=\sigma(W(0)\psi_{\vartheta}(\cdot,\cdot))$, $r(\cdot,\cdot):=\beta^T \psi_{\vartheta}(\cdot,\cdot)$, $q_0(\cdot,\cdot)=r(\cdot,\cdot)$, $D_0=\emptyset$, $n_0=0$, $\pi_1(\cdot)$=greedy $\alpha \in \mathcal{A}$ ($q_0(\cdot, \cdot)$).

5. **for** $k = 1, 2, \ldots$ **do**

6. **Data acquisition:** From behaviour policy $\rho_{\pi_k}$ collect $D_{new}=\{s_i, a_i, r_i, s'_i\}_{i=n_k+1}^{n_k+n_{new}}$.

7. **Augment $C$:** $C_k \leftarrow C_{k-1} \cup D_{new}$, $S_{new}'=\{s'_i\}_{i=n_k+1}^{n_k+n_{new}}$, $n_k \leftarrow n_{k-1}+n_{new}$.

8. **Train model:** $\alpha(\cdot,\cdot), C_k, r(\cdot,\cdot) \leftarrow \text{Train}(\alpha(\cdot,\cdot), C_k, r(\cdot,\cdot), D_k)$.

9. **Planning:** over $\nu:=\nu(s_i'), \ldots, \nu(s'_{[C_k]})^{\top}$, $(T_{\pi_k}^{\nu})v(\cdot):=r(s', \pi_k(\cdot)) + \gamma \alpha^{\top}(\cdot, \pi_k(\cdot))v$.

10. **for** $i = 1$ to $J_{imp}$ **do**

11. **for** $j = 1$ to $J_{eval}$ **do**

12. $v \leftarrow T_{\pi_k}^{\nu} v$.

13. **end for**

14. **end for**

15. $\pi_k(\cdot) \leftarrow \text{greedy}_{a \in A}[r(\cdot, \cdot) + \gamma \alpha^{\top}(\cdot, \cdot)v]$.

16. **end for**

17. $\pi_{k+1}(\cdot) \leftarrow \pi_k(\cdot)$.

18. **end for**

19. **return** $\pi_{k+1}(\cdot)$.

---

**Algorithm 2**

1. **function** $\text{Train}(\alpha(\cdot,\cdot), C, r(\cdot,\cdot), D)$

2. **for** $i = 1$ to $M$ **do**

3. **Draw minibatch** $B \sim D$

4. $\beta \leftarrow \beta - \eta \partial \mathcal{L}_{\vartheta} / \partial \vartheta |_B$.

5. **for each** $w_{gj} \in W$ **do**

6. $w_{gj} \leftarrow T^{active}_\alpha(w_{gj}, w_g, \eta_{gj})$.

7. $w_{gj} \leftarrow w_{gj} - \eta_{gj} \partial \mathcal{L}_{\vartheta} / \partial w_{gj} |_B$.

8. **end for**

9. $\vartheta \leftarrow \vartheta - \eta \left[ \partial \mathcal{L}_{\vartheta} / \partial \vartheta + \partial \mathcal{L}_{\alpha} / \partial \alpha \partial \vartheta / \partial \vartheta \right] |_B$, $\forall \vartheta \in \vartheta$.

10. **end for**

11. **for each** $w_g \in W$ **do**

12. **if** $\|w_g\|_2$ remained 0 over the last $\frac{1}{4}M$ minibatches **then**

13. $W \leftarrow W \setminus w_g$, $C \leftarrow C \setminus s'_i$.

14. **end if**

15. **end for**

16. **return** $\alpha(\cdot,\cdot), C, r(\cdot,\cdot)$. 

---

**Actively Compressed CMEs**
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


