Supplementary Material: Randomized Entity-wise Factorization for Multi-Agent Reinforcement Learning

Shariq Iqbal [1] Christian A. Schroeder de Witt [2] Bei Peng [2] Wendelin Böhmer [3] Shimon Whiteson [2] Fei Sha [1 4]

1. Attention Layers and Models

Attention models have recently generated intense interest due to their ability to incorporate information across large contexts. Importantly for our purposes, they are able to process variable sized sets of inputs.

We now formally define the building blocks of our attention models. Given the input $\boldsymbol{X}$, a matrix where the rows correspond to entities, we define an entity-wise feedforward layer as a standard fully connected layer that operates independently and identically over entities:

$$\text{eFF}(\boldsymbol{X};\boldsymbol{W},\boldsymbol{b}) = \boldsymbol{X}\boldsymbol{W} + \boldsymbol{b}^\top, \boldsymbol{X} \in \mathbb{R}^{n^x \times d}, \boldsymbol{W} \in \mathbb{R}^{d\times h}, \boldsymbol{b} \in \mathbb{R}^h \tag{1}$$

Now, we specify the operation that defines an attention head, given the additional inputs of $\mathcal{S} \subseteq \mathbb{Z}^{[1,n^x]}$, a set of indices that selects which rows of the input $\boldsymbol{X}$ are used to compute queries such that $X_{\mathcal{S},*} \in \mathbb{R}^{|\mathcal{S}|\times d}$, and $\boldsymbol{M}$, a binary obserability mask specifying which entities each query entity can observe (i.e. $\boldsymbol{M}_{i,j} = 1$ when $i \in \mathcal{S}$ can incorporate information from $j \in \mathbb{Z}^{[1,n^x]}$ into its local context):

$$\text{Atten}(\mathcal{S},\boldsymbol{X},\boldsymbol{M};\boldsymbol{W}^Q,\boldsymbol{W}^K,\boldsymbol{W}^V) = \text{softmax}\left(\text{mask}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{h}},\boldsymbol{M}\right)\right)\boldsymbol{V} \in \mathbb{R}^{|\mathcal{S}|\times h} \tag{2}$$

$$\boldsymbol{Q} = \boldsymbol{X}_{\mathcal{S},*}\boldsymbol{W}^Q, \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K, \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V, \quad \boldsymbol{M} \in \{0,1\}^{|\mathcal{S}|\times n^x}, \boldsymbol{W}^Q,\boldsymbol{W}^K,\boldsymbol{W}^V \in \mathbb{R}^{d\times h} \tag{3}$$

The mask$(\boldsymbol{Y},\boldsymbol{M})$ operation takes two equal sized matrices and fills the entries of $\boldsymbol{Y}$ with $-\infty$ in the indices where $\boldsymbol{M}$ is equal to 0. After the softmax, these entries become zero, thus preventing the attention mechanism from attending to specific entities. This masking procedure is used in our case to uphold partial observability, as well as to enable "imagining" the utility of actions within sub-groups of entities. Only one attention layer is permitted in the decentralized execution setting; otherwise information from unseen agents can be propagated through agents that are seen. $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$, and $\boldsymbol{W}^V$ are all learnable parameters of this layer. Queries, $\boldsymbol{Q}$, can be thought of as vectors specifying the type of information that an entity would like to select from others, while keys, $\boldsymbol{K}$, can be thought of as specifying the type of information that an entity possesses, and finally, values, $\boldsymbol{V}$, hold the information that is actually shared with other entities.

We define multi-head-attention as the parallel computation of attention heads as such:

$$\text{MHA}\left(\mathcal{S},\boldsymbol{X},\boldsymbol{M}\right) = \text{concat}\left(\text{Atten}\left(\mathcal{S},\boldsymbol{X},\boldsymbol{M};\boldsymbol{W}_j^Q,\boldsymbol{W}_j^K,\boldsymbol{W}_j^V\right), j \in \left(1\ldots n^h\right)\right) \tag{4}$$

The size of the parameters of an attention layer does not depend on the number of input entities. Furthermore, we receive an output vector for each query vector.

[1]Department of Computer Science, University of Southern California [2]Department of Computer Science, University of Oxford [3]Department of Software Technology, Delft University of Technology [4]Google Research. Correspondence to: Shariq Iqbal <shariqiq@usc.edu>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

2. Augmenting QMIX with Attention

The standard QMIX algorithm relies on a fixed number of entities in three places: inputs of the agent-specific utility functions Q_a, inputs of the hypernetwork, and the number of utilities entering the mixing network, which must correspond the output of the hypernetwork since it generates the parameters of the mixing network. QMIX uses multi-layer perceptrons for which all these quantities have to be of fixed size. In order to adapt QMIX to the variable agent quantity setting, such that we can apply a single model across all tasks, we require components that accept variable sized sets of entities as inputs. By utilizing attention mechanisms, we can design components that are no longer dependent on a fixed number of entities taken as input. We define the following inputs: $\boldsymbol{X}^{\mathcal{E}}_{ei} := s^e_i, 1 \leq i \leq d, e \in \mathcal{E}; \boldsymbol{M}^{\mu}_{ae} := \mu(s^a, s^e), a \in \mathcal{A}, e \in \mathcal{E}$. The matrix $\boldsymbol{X}^{\mathcal{E}}$ is the global state $\mathbf{s}$ reshaped into a matrix with a row for each entity, and $\boldsymbol{M}^{\mu}$ is a binary observability matrix which enables decentralized execution, determining which entities are visible to each agent.

2.1. Utility Networks

While the standard agent utility functions map a flat observation, whose size depends on the number of entities in the environment, to a utility for each action, our attention-utility functions can take in a variable sized set of entities and return a utility for each action. The attention layer output for agent a is computed as MHA $(\{a\}, \boldsymbol{X}, \boldsymbol{M}^{\mu})$, where $\boldsymbol{X}$ is an row-wise transformation of $\boldsymbol{X}^{\mathcal{E}}$ (e.g., an entity-wise feedforward layer). If agents share parameters, the layer can be computed in parallel for all agents by providing $\mathcal{A}$ instead of $\{a\}$, which we do in practice.

2.2. Generating Dynamic Sized Mixing Networks

Another challenge in devising a QMIX algorithm for variable agent quantities is to adapt the hypernetworks that generate weights for the mixing network. Since the mixing network takes in utilities from each agent, we must generate feedforward mixing network parameters that change in size depending on the number of agents present, while incorporating global state information. Conveniently, the number of output vectors of a MHA layer depends on the cardinality of input set $\mathcal{S}$ and we can therefore generate mixing parameters of the correct size by using $\mathcal{S} = \mathcal{A}$ and concatenating the vectors to form a matrix with one dimension size depending on the number of agents and the other depending on the number of hidden dimensions. Attention-based QMIX (QMIX (Attention)) trains these models using the standard DQN loss in Equation 2 of the main text.

Our two layer mixing network requires the following parameters to be generated: $\boldsymbol{W}_1 \in \mathbb{R}^{+(|\mathcal{A}| \times h^m)}$, $\boldsymbol{b}_1 \in \mathbb{R}^{h^m}$, $\boldsymbol{w}_2 \in \mathbb{R}^{+(h^m)}$, $b_2 \in \mathbb{R}$, where h^m is the hidden dimension of the mixing network and $|\mathcal{A}|$ is the set of agents.

Note from Eq. (2) that the output size of the layer is dependent on the size of the query set. As such, using attention layers, we can generate a matrix of size $|\mathcal{A}| \times h^m$, by specifying the set of agents, $\mathcal{A}$, as the set of queries $\mathcal{S}$ from Eq. (2). We do not need observability masking since hypernetworks are only used during training and can be fully centralized. For each of the four components of the mixing network ($\boldsymbol{W}_1, \boldsymbol{b}_1, \boldsymbol{w}_2, b_2$), we introduce a hypernetwork that generates parameters of the correct size. Thus, for the parameters that are vectors ($\boldsymbol{b}_1$ and $\boldsymbol{w}_2$), we average the matrix generated by the attention layer across the $|\mathcal{A}|$ sized dimension, and for b_2, we average all elements. This procedure enables the dynamic generation of mixing networks whose input size varies with the number of agents. Assuming $\boldsymbol{q} = [Q^1(\tau^1, u^1), \ldots, Q^n(\tau^n, u^n)]$, then Q^{tot} is computed as:

$$Q^{\text{tot}}(\mathbf{s}, \tau, \mathbf{u}) = \sigma((\boldsymbol{q}^\top \boldsymbol{W}_1) + \boldsymbol{b}_1^\top)\boldsymbol{w}_2 + b_2 \tag{5}$$

where σ is an ELU nonlinearity (Clevert et al., 2015).

3. Environment Details

3.1. StarCraft with Variable Agents and Enemies

The standard version of SMAC loads map files with pre-defined and fixed unit types, where the global state and observations are flat vectors with segments corresponding to each agent and enemy. Partial observability is implemented by zeroing out segments of the observations corresponding to unobserved agents. The size of these vectors changes depending on the number of agents placed in the map file. Furthermore, the action space consists of movement actions as well as separate actions to attack each enemy unit. As such the action space also changes as the number of agents changes.

Our version loads empty map files and programmatically generates agents, allowing greater flexibility in terms of the units present to begin each episode. The global state is split into a list of equal-sized entity descriptor vectors (for both agents and enemies), and partial observability is handled by generating a matrix that shows what entities are visible to each agent.

The variable-sized action space is handled by randomly assigning each enemy a tag at the beginning of each episode and designating an action to attack each possible tag, of which there are a maximum number (i.e. the maximum possible number of enemies across all tasks). Agents are able to see the tag of the enemies they observe and can select the appropriate action that matches this tag in order to attack a specific enemy. Since UPDeT (Hu et al., 2021) naturally handles the variable-sized action space and the mapping of entities to actions, we ensure that the output action utilities for UPDeT are shuffled based on the tags to maintain the entity-to-action mapping.

4. Experimental Details

Our experiments were performed on a desktop machine with a 6-core Intel Core i7-6800K CPU and 3 NVIDIA Titan Xp GPUs, and a server with 2 16-core Intel Xeon Gold 6154 CPUs and 10 NVIDIA Titan Xp GPUs. Each experiment is run with 8 parallel environments for data collection and a single GPU. **REFIL** takes about 24 hours to run for 10M steps on STARCRAFT. QMIX (Attention) takes about 16 hours for the same number of steps on STARCRAFT. Reported times are on the desktop machine and the server runs approximately 15% faster due to more cores being available for running the environments in parallel.

5. Hyperparameters

Hyperparameters were based on the PyMARL (Samvelyan et al., 2019) implementation of QMIX and are listed in Table 1. All hyperparameters are the same in all STARCRAFT settings. Since we train for 10 million timesteps (as opposed to the typical 2 million in standard SMAC), we extend the epsilon annealing period (for epsilon-greedy exploration) from 50,000 steps to 500,000 steps. For hyperparameters new to our approach (hidden dimensions of attention layers, number of attention heads, λ weighting of imagined loss), the specified values in Table 1 were the first values tried, and we found them to work well. The robustness of our approach to hyperparameter settings, as well as the fact that we do not tune hyperparameters per environment, is a strong indicator of the general applicability of our method.

Table 1: Hyperparameter settings across all runs and algorithms/baselines.

Name	Description	Value
lr	learning rate	0.0005
optimizer	type of optimizer	RMSProp[1]
optim α	RMSProp param	0.99
optim ϵ	RMSProp param	$1e-5$
target update interval	copy live params to target params every _ episodes	200
bs	batch size (# of episodes per batch)	32
grad clip	reduce global norm of gradients beyond this value	10
$\|D\|$	maximum size of replay buffer (in episodes)	5000
γ	discount factor	0.99
starting ϵ	starting value for exploraton rate annealing	1.0
ending ϵ	ending value for exploraton rate annealing	0.05
anneal time	number of steps to anneal exploration rate over	500000
h^a	hidden dimensions for attention layers	128
h^r	hidden dimensions for RNN layers	64
h^m	hidden dimensions for mixing network	32
# attention heads	Number of attention heads	4
nonlinearity	type of nonlinearity (outside of mixing net)	ReLU
λ	Weighting between standard QMIX loss and imagined loss	0.5

[1]: Tieleman & Hinton (2012)

References

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Hu, S., Zhu, F., Chang, X., and Liang, X. {UPD}et: Universal multi-agent {rl} via policy decoupling with transformers. In *International Conference on Learning Representations*, 2021.

Samvelyan, M., Rashid, T., Schroeder de Witt, C., Farquhar, G., Nardelli, N., Rudner, T. G., Hung, C.-M., Torr, P. H., Foerster, J., and Whiteson, S. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 2186–2188. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.